

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术201707**

**学 号： U201714786**

**姓 名： 王占成**

**指导教师： 祝建华**

**报告日期： 2019年1月2日**

**计算机科学与技术学院**

# 目 录

[目 录 2](#_Toc534180501)

[1 基于顺序存储结构的线性表实现 4](#_Toc534180502)

[1.1 问题描述 4](#_Toc534180503)

[1.2 系统设计 5](#_Toc534180504)

[1.3 系统实现 6](#_Toc534180505)

[1.4 实验小结 21](#_Toc534180506)

[2 基于链式存储结构的线性表实现 23](#_Toc534180507)

[2.1 问题描述 23](#_Toc534180509)

[2.2 系统设计 23](#_Toc534180510)

[2.3 系统实现 31](#_Toc534180511)

[2.4 实验小结 47](#_Toc534180512)

[3基于二叉链表的二叉树实现 48](#_Toc534180513)

[3.1 实验目的 48](#_Toc534180514)

[3.2 模块设计 48](#_Toc534180515)

[3.3 模块实现 52](#_Toc534180516)

[3.4 模块测试 59](#_Toc534180517)

[3.5 实验小结 76](#_Toc534180518)

[4基于邻接表结构的图实现 77](#_Toc534180519)

[4.1 问题描述 77](#_Toc534180520)

[4.2 模块设计 77](#_Toc534180521)

[4.3 模块实现 80](#_Toc534180522)

[4.4 模块测试 86](#_Toc534180523)

[4.5 实验小结 101](#_Toc534180524)

[参考文献 102](#_Toc534180525)

[附录A 103](#_Toc534180526)

[附录B 122](#_Toc534180527)

[附录C 142](#_Toc534180528)

[附录D 185](#_Toc534180529)

# 1 基于顺序存储结构的线性表实现

## 问题描述

### 实验目标：

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

### 实验内容：

采用顺序表作为线性表的物理结构，实现多种对表的操作。同时本程序具有菜单演示的功能。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，并且给出适当的操作提示。演示系统实现了多个线性表管理，并且在不同的表中进行切换，来真正地实现对表的多种操作。

线性表在物理内存中可以以顺序表的方式实现，即物理上存储位置相邻的两个元素是线性表中的相邻元素，且数据元素的前后关系不变。

实验要完成的顺序表算法：

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,a)；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有相等学号的学生的位序。

8. 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。

9. 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。

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；操作结果是输出表中每个数据元素的值。

13． 读取文件：函数名称是ReadFromFile(\*L,moro,\*Isinit,\*hw,wh)，初始条件是线性表L已存在，操作结果是将txt文件中的内容读取到顺序表中。

14. 创建多表：没有具体函数，无初始条件，操作结果是创建一个顺序表来储存多个顺序表。

15. 切换多表：无具体函数，初始条件是多表已被创建，操作结果是将当前操作的表切换为其它表。

16. 保存文件：函数名称是WriteToFile(\*L)，初始条件是线性表L已存在，操作结果是将表中的内容保存在txt文件中。

## 系统设计

演示系统可进行单表和多表的操作，在单表操作的过程中可切换为多表操作。多表之间可以进行切换，切换过程中可选择保存当前表或者在最后退出程序的时候进行保存。当选择对当前的表进行保存时，可自由的命名文件名，文件将最终以txt文件的形式保存在文件夹中。（如图1-1所示）

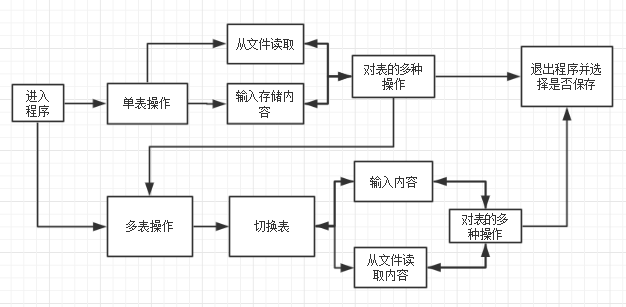


图1-1 程序的系统设计流程图

## 系统实现

本次实验中所使用的环境配置如下：

(1) 操作系统版本：Windows 10

(2) 编译器及其版本：mingw32-g++ ver3.12.3

(3) 编译工具：MinGw ver5.0

(4) 编程环境：CLion

实验的完整代码参见附录A

在进入系统时，首先默认的操作为单表操作，此时表未初始化。此时可以进行三种操作：

### 1.3.1 初始化表

初始化表，并且向表中键入内容，然后可以进行对表的任何操作。其中比较特别的时进入多表操作和从文件读取内容。

#### 1.3.1.1 多表操作

此时需要选择是否保存单表，若单表不保存，默认为被舍弃。若单表被保存，则需要输入文件名。之后进入多表操作，选择要同时操作表的数目，之后可进行与单表同样的操作(注意：当多表已经创建时，不可以再次创建多表)。当在不同的表进行切换时，选择是否保存当前表，若不保存，可以在程序退出时再次选择是否保存。退出当前表时，程序将询问是否保存当前表，若保存，则输入文件名；否则，舍弃当前表。切换表时不可再次切换到该表。之后程序将自动切换到其他的表，当所有的表被保存或被舍弃时，则退出程序。

#### 1.3.1.2 从文件读取内容

此时需要输入文件名，若文件存在，则自动的清空表并且将文件的内容读取到表中，之后可以进行与单表操作同样的操作。

若文件不存在，则读取失败。若想重新向表中输入内容，需要销毁表并且重新初始化。

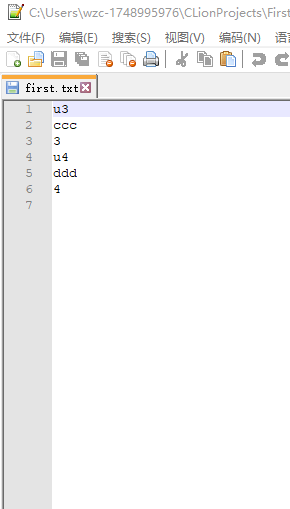
### 1.3.2 读取文件

系统自动初始化表，并且从文件中读取内容。此时可以进行与单表操作同样的操作。

### 1.3.3 进行多表操作

直接进入多表操作。因为并没有初始化单表，所以并不会询问是否保存单表。然后可以进行与1.1 同样的操作。

### 测试样例：

在文件夹中有first.txt文件，文件的内容如下：

测试用例及其结果如下（各函数测试为独立测试，测试初始数据相同，不受上个函数测试影响）：

1. 测试函数：IntiaList

测试步骤及结果如表1-1所示

表1-1 IntiaList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1进入函数  2.按提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3. | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |

1. 测试函数：DestroyList

测试步骤及结果如表1-2所示

表1-2 DestroyList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 主界面输入2进入函数 | 输出“销毁顺序表成功”！ | 输出“销毁顺序表成功”！ |
| 4 | 主界面输入4进入函数 | 输出“线性表未创建！请重新选择你的操作！” | 输出“线性表未创建！请重新选择你的操作！” |

1. 测试函数：ClearList

测试步骤及结果如表1-3所示

表1-3 ClearList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 主界面输入3进入函数 | 输出“清空顺序表成功！” | 输出“清空顺序表成功！” |
| 4 | 主界面输入4进入函数 | 输出“顺序表为空” | 输出“顺序表为空” |

1. 测试函数：ListEmpty

测试步骤及结果如表1-4所示

表1-4 ListEmpty函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 主界面输入4进入函数 | 输出“顺序表不为空” | 输出“顺序表不为空” |
| 4 | 主界面输入3进入函数 | 输出“清空顺序表成功！” | 输出“清空顺序表成功！” |
| 5 | 主界面输入4进入函数 | 输出“顺序表为空” | 输出“顺序表为空” |

1. 测试函数：ListLength

测试步骤及结果如表1-5所示

表1-5 ListLength函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 主界面输入5进入函数 | 输出“顺序表的长度为3” | 输出“顺序表的长度为3” |

1. 测试函数：GetElem

测试步骤及结果如表1-6所示

表1-6 GetElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入6进入函数  2.按提示输入要被查找学生的学号u1 | 输出“学号: u1 学生的信息为:  ID:u1  Name:aaa  Score:1” | 输出“学号: u1 学生的信息为:  ID:u1  Name:aaa  Score:1” |

1. 测试函数：LocateElem

测试步骤及结果如表1-7所示

表1-7 LocateElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入7进入函数  2.按提示输入要被查找学生的学号u1 | 输出“目标学生成绩表中的位序为1” | 输出“目标学生成绩表中的位序为1” |

1. 测试函数：PriorElem

测试步骤及结果如表1-8所示

表1-8 PriorElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入8进入函数  2.按提示输入顺序表中被查找学生的下面学生的学号u2 | 输出“存在你所查找的学生:  ID:u1  Name:aaa  Score:1” | 输出“存在你所查找的学生:  ID:u1  Name:aaa  Score:1” |

1. 测试函数：NextElem

测试步骤及结果如表1-9所示

表1-9 NextElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入9进入函数  2.按提示输入顺序表中被查找学生的上面学生的学号u1 | 输出“存在你所查找的学生:  ID:u2  Name:bbb  Score:2” | 输出“存在你所查找的学生:  ID:u2  Name:bbb  Score:2” |

1. 测试函数：ListInsert

测试步骤及结果如表1-10所示

表1-10 ListInsert函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入10进入函数  2.按提示输入你想插入的位置：3. 输入你想插入的学生的学生信息：  u3 ccc 3 | 输出“插入成功！” | 输出“插入成功！” |
| 4 | 主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2  ID:u3  Name:ccc  Score:3” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2  ID:u3  Name:ccc  Score:3” |

1. 测试函数：ListDelete

测试步骤及结果如表1-11所示

表1-11 ListDelete函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入11进入函数  2.按提示输入被删除学生的学号u1 | 输出“你删除的学生的信息为：ID:u1  Name:aaa  Score:1  删除成功！” | 输出“你删除的学生的信息为：ID:u1  Name:aaa  Score:1  删除成功！” |
| 4 | 主界面输入12进入函数 | 输出“ID:u2  Name:bbb  Score:2” | 输出“ID:u2  Name:bbb  Score:2” |

1. 测试函数：ListTrabverse

测试步骤及结果如表1-12所示

表1-12 ListTrabverse函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 1.主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |

1. 测试函数：ReadFromFile

测试步骤及结果如表1-13所示

表1-13 ReadFromFile函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要读取的txt文件first | 无输出，文件内容已被读取到当前操作的表中 | 无输出，文件内容已被读取到当前操作的表中 |
| 2 | 主界面输入12进入函数 | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” |

1. 测试函数：MultiList

测试步骤及结果如表1-14所示

表1-14 MultiList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入14进入函数  2.按提示输入要操作的线性表数目，输入2 | 输出“操作成功！当前操作的表为表 1” | 输出“操作成功！当前操作的表为表 1” |
| 2 | 1.主界面输入13进入函数  2.按提示输入要读取的txt文件first | 无输出，文件内容已被读取到当前操作的表中 | 无输出，文件内容已被读取到当前操作的表中 |
| 3 | 1.主界面输入12进入函数 | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” |
| 4 | 1.主界面输入15进入函数  2. 根据提示是否要保存(若不保存，退出程序时可再次选择是否保存)上一个表?是：1 否：0  0  请输入被切换到表的序号(输入数字0退出切表功能)  2 | 输出“当前操作的为表 2” | 输出“当前操作的为表 2” |
| 5. | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 6. | 1.主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 7. | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |

15）测试函数：SwitchList

测试步骤及结果如表1-15所示

表1-15 SwitchList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入14进入函数  2.按提示输入要操作的线性表数目，输入2 | 输出“操作成功！当前操作的表为表 1” | 输出“操作成功！当前操作的表为表 1” |
| 2 | 1.主界面输入13进入函数  2.按提示输入要读取的txt文件first | 无输出，文件内容已被读取到当前操作的表中 | 无输出，文件内容已被读取到当前操作的表中 |
| 3 | 1.主界面输入12进入函数 | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” |
| 4 | 1.主界面输入15进入函数  2. 根据提示是否要保存(若不保存，退出程序时可再次选择是否保存)上一个表?是：1 否：0  0  请输入被切换到表的序号(输入数字0退出切表功能)  2 | 输出“当前操作的为表 2” | 输出“当前操作的为表 2” |
| 5. | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 6. | 1.主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 7. | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |
| 8. | 1.主界面输入15进入函数  2. 根据提示是否要保存(若不保存，退出程序时可再次选择是否保存)上一个表?是：1 否：0  0  请输入被切换到表的序号(输入数字0退出切表功能)  1 | 输出“当前操作的为表 1” | 输出“当前操作的为表 1” |
| 9. | 1.主界面输入12进入函数 | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” |

16）测试函数：SaveList

测试步骤及结果如表1-16所示

表1-16 SaveList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1进入函数  2.按提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 | 输出“当前已初始化表!”并且提示输入学生的信息，学生的数目不能超过100个 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3. | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |
| 4. | 1.主界面输入16进入函数  2.根据提示输入：  是否要保存当前表?是：1 否：0  1  输入文件名称（不超过20个字符）:  second | 无输出，顺序表内容保存在second.txt文件中 | 无输出，顺序表内容保存在second.txt文件中 |
| 5. | 主界面输入2进入函数 | 输出“销毁顺序表成功”！ | 输出“销毁顺序表成功”！ |
| 6. | 1.主界面输入13进入函数  2.按提示输入要读取的txt文件second | 无输出，文件内容已被读取到当前操作的表中 | 无输出，文件内容已被读取到当前操作的表中 |
| 7. | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |

## 实验小结

在本次实验中，让我对顺序存储结构有了更深刻的了解。当进行单表操作设计的时候，并没有遇到比较棘手的问题。但是，在对多表操作设计的时候，遇到了比较难的问题。怎么样来判断不同的表的状态，是已经被初始化，被保存，还是被舍弃。后来想到可以用一个数组用数字来存储表的不同的状态。之前想用引用来对表进行操作，后来发现到多表操作无法进行，还是指针好用。实验之前，最好设计一个表格，来详细地表现出系统的结构，这样便能够高效地设计程序。

# 2 基于链式存储结构的线性表实现



## 问题描述

### 实验目标：

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

### 实验内容：

采用单链表作为线性表的物理结构，实现多种对表的操作。同时本程序具有菜单演示的功能。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，并且给出适当的操作提示。演示系统实现了多个线性表管理，并且在不同的表中进行切换，来真正地实现对表的多种操作。

线性表在物理内存中可以单链表的方式实现，即线性表中的相邻元素在物理内存的位置上不一定相邻，但数据元素的前后关系不变。

## 系统设计

系统模块构成如图 2-2-1 所示

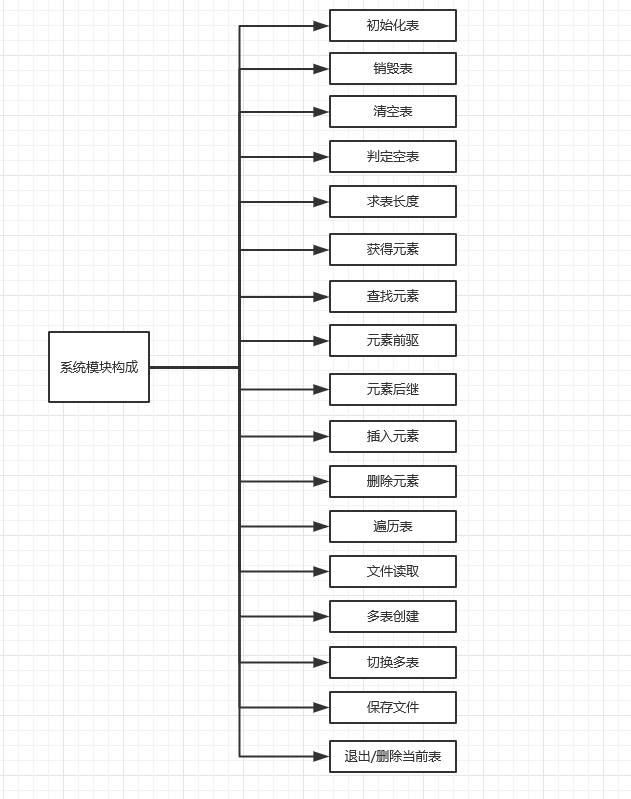


图2-2-1 系统模块构成图

各个模块的输入、输出以及功能实现介绍如下：   
1.初始化表：

条件一：线性表L不存在

输入：根据提示输入学生的学号、姓名以及成绩

输出：线性表创建成功！

功能实现：创建一个存储学生信息的线性表

条件二：线性表L存在

输入：无

输出：线性表创建失败！

功能实现：重新选择你的操作！

2.销毁表：

条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在

输入：无

输出：销毁线性表成功！

功能实现：销毁当前的线性表，线性表成为未初始化(未创建)状态！  
3.清空表：

条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在

输入：无

输出：清空线性表成功！

功能实现：清空当前的线性表  
4.判定空表：

条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在且为空

输入：无

输出：线性表为空

功能实现：判定当前线性表是否为空表

条件三：线性表存在且不为空

输入：无

输出：线性表不为空

功能实现：判定当前线性表是否为空表  
5.求表长：

条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在

输入：无

输出：线性表的长度为\*\*

功能实现：输出当前线性表的长度，即包含多少个数据元素   
6.获得元素：

条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在

输入：学生学号

输出：若目标学生存在，打印学生的学生信息；若学生不存在，则打印“查无此人”

功能实现：查找学生并打印学生的信息  
7.查找元素：

条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在

输入：学生学号

输出：若目标学生存在，打印目标学生在表中的位序号；若学生不存在，则打印“查无此人”

功能实现：查找学生并打印学生的位序   
8.获得前驱：

条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在

输入：学生学号

输出：若学生(元素)前驱存在，则打印前驱结点的学生信息；若学生(元

素)前驱不存在，则打印“查无此人”

功能实现：获得并打印目标学生的前驱信息   
9.获得后继：

条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在

输入：学生学号

输出：若学生(元素)后继存在，则打印后继结点的学生信息；若学生(元

素)后继不存在，则打印“查无此人”

功能实现：获得并打印目标学生的后继信息  
10.插入元素：

条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在

输入：输入要插入的位序以及学生的信息

输出：若位序合法，打印“插入成功”；若位序不合法，则打印“输入的位序不合法”

功能实现：在合法的位置插入学生信息   
11.删除元素：条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在

输入：输入要被删除学生的学号

输出：若学生存在，则打印被删除学生的信息；否则，打印“目标学生不存在”

功能实现：删除表中已存在的学生的信息   
12.遍历表：

条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在

输入：无

输出：按次序输出表中学生的信息

功能实现：遍历并输出表中学生的信息  
13．读取文件：

条件一：线性表L不存在

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：线性表L存在

输入：输入要读取的文件名称

输出：若文件存在并打开成功，打印“读取成功”；否则，打印“不存在此文件”

功能实现：将已存在的文件读取到线性表中  
14.创建多表：

条件一：线性表L不存在

输入：输入你要创建的多表数量

输出：多表创建成功！当前操作的表位表1

功能实现：创建指定数量的多表

条件二：线性表L存在

输入：根据提示是否保存当前单表。若保存，输入1后输入文件名称；否则，输入0后操作情况与条件一相同

输出：创建文件成功!(保存)

功能实现：询问是否保存单表并创建多表

条件三：多表已存在

输入：无

输出：当前已创建多表，无需再次创建

功能实现：拒绝重复创建多表   
15.切换多表：

条件一：多表不存在

输入：无

输出：不存在多表！

功能实现：单表无法切换表

条件二：多表存在且目标表存在

输入：输入目标表的序号

输出：当前操作的表为表\*

功能实现：切换到指定的表

条件三：多表存在且目标表不存在或已被删除

输入：输入目标表的序号

输出：此表已被舍弃！无法切换到该表！请重新输入！

功能实现：拒绝切换到被舍弃(删除)的表

条件四：多表已存在且输入的表的序号不存在

输入：输入目标表的序号

输出：输入的表序不合法，请重新输入

功能实现：拒绝切换到不存在的表   
16.保存文件：

条件一：当前表未创建

输入：无

输出：线性表未创建！请重新选择你的操作！

功能实现：重新选择你的操作

条件二：当前表已创建

输入：输入保存的文件名称

输出：创建文件成功!

功能实现：将表中的内容保存到指定文件中

功能实现：拒绝切换到不存在的表演示系统可进行单表和多表的操作，在单表操作的过程中可切换为多表操作。多表之间可以进行切换，切换过程中可选择保存当前表或者在最后退出程序的时候进行保存。当选择对当前的表进行保存时，可自由的命名文件名，文件将最终以txt文件的形式保存在文件夹中。（如图2-2-2所示）

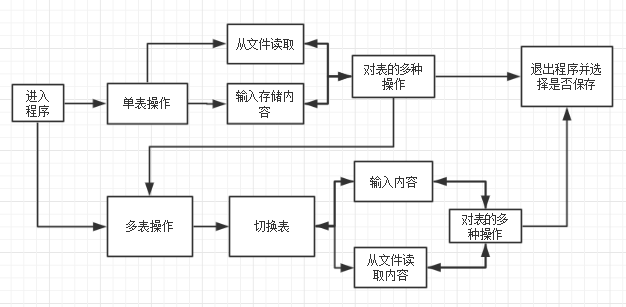


图2-2-2 程序的系统设计流程图

## 系统实现

本次实验中所使用的环境配置如下：

(1) 操作系统版本：Windows 10

(2) 编译器及其版本：mingw32-g++ ver3.12.3

(3) 编译工具：MinGw ver5.0

(4) 编程环境：CLion

实验的完整代码参见附录B

### 系统各模块算法实现以及函数调用

1.初始化表：引用的函数名称是InitaList(\*L)

在函数中为表初始化，即为第一个元素结点分配空间，第一个结点不存储任何信息；之后根据用户输入来不断地为后面的结点分配空间。当用户的输入结束时，系统停止分配空间并将尾结点的指针设置为空。之后在模块中将表的状态设置为已创建的状态。

2.销毁表：引用的函数是DestroyList(\*L)

在函数中从首结点遍历到尾结点，将各个结点所占的空间逐一释放掉。之后在模块中将表的状态设置为未创建的状态。

3.清空表：引用的函数是ClearList(\*L)

在函数中将表的长度设置为0，模块中不做任何操作。

4.判定空表：引用的函数是ListEmpty(L)

在函数判断表的长度是否为0，若表的长度为0，当前表判定为空表；否则，当前表位非空表。模块中不做任何操作。

5.求表长：引用的函数是ListLength(L)

在函数中返回当前表的长度并且在模块中输出函数的返回值，从而得到当前操作表的长度。

6.获得元素：引用的函数是GetElem(L,stu,e)

在模块中首先创建一个Elemtype类型的变量e和一个字符类型指针stu。e中用来存储函数中查询到的学生信息，stu用来存储用户输入的学生学号。在函数中逐个遍历所有的结点，若结点中的存在学生的id与stu相同，则将学生的信息复制到e中，函数返回OK并在模块中输出学生的信息；否则，函数返回FALSE并在模块中输出告知用户学生不存在的信息。

7.查找元素：引用的函数是LocateElem(L,e,a)

在模块中首先创建一个Elemtype类型的变量e和一个整数类型a。e中的id用来存储用户查询学生的id ，a用来存储函数返回的学生位序。函数中逐个遍历所有的结点，若结点中的存在学生的id与e中的id信息相同，将学生的位序复制到a中，函数返回OK并在模块中输出学生的位序；否则，函数返回FALSE并在模块中输出告知用户学生不存在的信息。

8.获得前驱：引用的函数是PriorElem(L,cur\_e,pre\_e)

cur\_e中的id用来存储用户输入的学生学号，pre\_e用来存储函数中查找结点的前驱信息。函数中逐个遍历所有的结点，若表中存在结点中的id与cur\_e中的id相同，则将学生的信息复制到pre\_e中，函数返回OK并在模块中输出前驱结点学生的信息；否则，函数函数返回FALSE并在模块中输出告知用户学生不存在的信息。

9.获得后继：引用的函数是NextElem(L,cur\_e,next\_e)

cur\_e中的id用来存储用户输入的学生学号，next\_e用来存储函数中查找结点的后继信息。函数中逐个遍历所有的结点，若表中存在结点中的id与cur\_e中的id相同，则将学生的信息复制到next\_e中，函数返回OK并在模块中输出后继结点学生的信息；否则，函数函数返回FALSE并在模块中输出告知用户学生不存在的信息。

10.插入元素：引用的函数是ListInsert(\*L,i,e)

e用来存储用户要插入的学生信息，i用来存储用户输入的要插入的位置。若该位置合法(存在),则在该位置插入学生的信息，并且在将其前面的结点指针设置为指向新结点，将新结点的指针指向前驱结点的指针原来指向的位置，函数返回OK；否则，函数返回FALSE并且模块中输出告知用户位置不合法的信息。

11.删除元素：引用的函数是ListDelete(\*L,i,e)

e用来存储用户要插入的学生信息，i用来存储用户输入的要删除结点的位置。若该位置合法(存在),则该位置学生的信息，并且在将其前面的结点指针设置为指向其后面的结点，函数返回OK；否则，函数返回FALSE并且模块中输出告知用户位置不合法的信息

12.遍历表：引用的函数是ListTraverse(L)

在函数中遍历表中所有的结点并且输出结点的信息，若结点的指针指向为空，则遍历结束，函数返回OK。

13.读取文件：引用的函数是ReadFromFile(\*L,moro,\*Isinit,\*hw,wh)

moro用来判断当前表是否为多表中一表，Isinit用来判断当前表是否被初始化，hw用来读取多表中某表的状态，wh用来判断当前在多表中的位置(若该表为多表中的一表)。经过判断后，若当前表已将创建，则询问用户是否保存当前表的信息，若保存，则调用WriteToFile(\*L)；否则，输入要读取的文件名。读取文件时，若当前表已创建，则调用DestroyList(\*L)。然后从文件中逐个读取信息，读取一个信息就分配一个结点的空间来存储结点的信息，直到读取到文件尾。

14.创建多表：无具体函数

在模块中创建多个表指针来指向不同的表。创建表后，当前操作的表默认为表1。

15.切换多表：无具体函数

在切换之前，首先询问用户是否保存当前表。若保存，则调用WriteToFile(\*L);若不保存，则程序继续运行。在模块中首先判断要切换的表是否存在，若已经被删除，则不能切换到该表并且输出告知用户该表已被删除的信息。若该表存在，则切换到该表。

16.保存文件：引用的函数是WriteToFile(\*L)

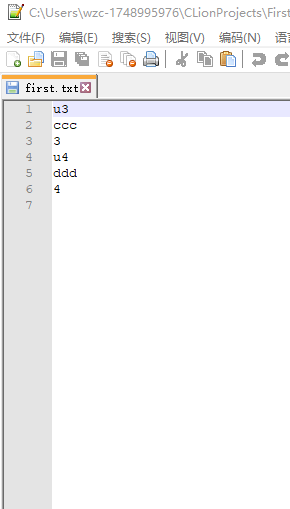
在函数中要求用户输入文件名，之后遍历表中所有的结点并且将结点中的信息打印到文件中。若文件已存在，则会覆盖原文件的内容。

0.退出：无引用函数

经过和13读取文件 开始部分相同的判断，若当前表为单表，则询问用户是否保存。若保存，则调用WriteToFile(\*L);否则，将退出程序。若当前表为多表中的一表，则询问用户是否保存。若不保存，则删除该表并且自动切换到多表中未删除的表。

### 测试样例：

在文件夹中有first.txt文件，文件的内容如下：



测试用例及其结果如下（各函数测试为独立测试，测试初始数据相同，不受上个函数测试影响）：

1. 测试函数：IntiaList

测试步骤及结果如表2-1所示

表2-1 IntiaList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1进入函数  2.按提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3. | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |

1. 测试函数：DestroyList

测试步骤及结果如表2-2所示

表2-2 DestroyList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 主界面输入2进入函数 | 输出“销毁线性表成功”！ | 输出“销毁线性表成功”！ |
| 4 | 主界面输入4进入函数 | 输出“线性表未创建！请重新选择你的操作！” | 输出“线性表未创建！请重新选择你的操作！” |

1. 测试函数：ClearList

测试步骤及结果如表2-3所示

表2-3 ClearList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 主界面输入3进入函数 | 输出“清空顺序表成功！” | 输出“清空顺序表成功！” |
| 4 | 主界面输入4进入函数 | 输出“顺序表为空” | 输出“顺序表为空” |

1. 测试函数：ListEmpty

测试步骤及结果如表2-4所示

表2-4 ListEmpty函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 主界面输入4进入函数 | 输出“线性表不为空” | 输出“线性表不为空” |
| 4 | 主界面输入3进入函数 | 输出“清空线性表成功！” | 输出“清空线性表成功！” |
| 5 | 主界面输入4进入函数 | 输出“线性表为空” | 输出“线性表为空” |

1. 测试函数：ListLength

测试步骤及结果如表2-5所示

表2-5 ListLength函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 主界面输入5进入函数 | 输出“线性表的长度为3” | 输出“线性表的长度为3” |

1. 测试函数：GetElem

测试步骤及结果如表2-6所示

表2-6 GetElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入6进入函数  2.按提示输入要被查找学生的学号u1 | 输出“学号: u1 学生的信息为:  ID:u1  Name:aaa  Score:1” | 输出“学号: u1 学生的信息为:  ID:u1  Name:aaa  Score:1” |

1. 测试函数：LocateElem

测试步骤及结果如表2-7所示

表2-7 LocateElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入7进入函数  2.按提示输入要被查找学生的学号u1 | 输出“目标学生成绩表中的位序为1” | 输出“目标学生成绩表中的位序为1” |

1. 测试函数：PriorElem

测试步骤及结果如表2-8所示

表2-8 PriorElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入8进入函数  2.按提示输入顺序表中被查找学生的下面学生的学号u2 | 输出“存在你所查找的学生:  ID:u1  Name:aaa  Score:1” | 输出“存在你所查找的学生:  ID:u1  Name:aaa  Score:1” |

1. 测试函数：NextElem

测试步骤及结果如表2-9所示

表2-9 NextElem函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入9进入函数  2.按提示输入顺序表中被查找学生的上面学生的学号u1 | 输出“存在你所查找的学生:  ID:u2  Name:bbb  Score:2” | 输出“存在你所查找的学生:  ID:u2  Name:bbb  Score:2” |

1. 测试函数：ListInsert

测试步骤及结果如表2-10所示

表2-10 ListInsert函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入10进入函数  2.按提示输入你想插入的位置：3. 输入你想插入的学生的学生信息：  u3 ccc 3 | 输出“插入成功！” | 输出“插入成功！” |
| 4 | 主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2  ID:u3  Name:ccc  Score:3” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2  ID:u3  Name:ccc  Score:3” |

1. 测试函数：ListDelete

测试步骤及结果如表2-11所示

表2-11 ListDelete函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入11进入函数  2.按提示输入被删除学生的学号u1 | 输出“你删除的学生的信息为：ID:u1  Name:aaa  Score:1  删除成功！” | 输出“你删除的学生的信息为：ID:u1  Name:aaa  Score:1  删除成功！” |
| 4 | 主界面输入12进入函数 | 输出“ID:u2  Name:bbb  Score:2” | 输出“ID:u2  Name:bbb  Score:2” |

1. 测试函数：ListTrabverse

测试步骤及结果如表2-12所示

表2-12 ListTrabverse函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 1.主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3 | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |

1. 测试函数：ReadFromFile

测试步骤及结果如表2-13所示

表2-13 ReadFromFile函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入13进入函数  2.按提示输入要读取的txt文件first | 无输出，文件内容已被读取到当前操作的表中 | 无输出，文件内容已被读取到当前操作的表中 |
| 2 | 主界面输入12进入函数 | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” |

14) 测试函数：MultiList

测试步骤及结果如表2-14所示

表2-14 MultiList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入14进入函数  2.按提示输入要操作的线性表数目，输入2 | 输出“操作成功！当前操作的表为表 1” | 输出“操作成功！当前操作的表为表 1” |
| 2 | 1.主界面输入13进入函数  2.按提示输入要读取的txt文件first | 无输出，文件内容已被读取到当前操作的表中 | 无输出，文件内容已被读取到当前操作的表中 |
| 3 | 1.主界面输入12进入函数 | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” |
| 4 | 1.主界面输入15进入函数  2. 根据提示是否要保存(若不保存，退出程序时可再次选择是否保存)上一个表?是：1 否：0  0  请输入被切换到表的序号(输入数字0退出切表功能)  2 | 输出“当前操作的为表 2” | 输出“当前操作的为表 2” |
| 5. | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 6. | 1.主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 7. | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |

15）测试函数：SwitchList

测试步骤及结果如表2-15所示

表2-15 SwitchList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入14进入函数  2.按提示输入要操作的线性表数目，输入2 | 输出“操作成功！当前操作的表为表 1” | 输出“操作成功！当前操作的表为表 1” |
| 2 | 1.主界面输入13进入函数  2.按提示输入要读取的txt文件first | 无输出，文件内容已被读取到当前操作的表中 | 无输出，文件内容已被读取到当前操作的表中 |
| 3 | 1.主界面输入12进入函数 | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” |
| 4 | 1.主界面输入15进入函数  2. 根据提示是否要保存(若不保存，退出程序时可再次选择是否保存)上一个表?是：1 否：0  0  请输入被切换到表的序号(输入数字0退出切表功能)  2 | 输出“当前操作的为表 2” | 输出“当前操作的为表 2” |
| 5. | 1.主界面输入1初始化表  2.按照提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 6. | 1.主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 7. | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |
| 8. | 1.主界面输入15进入函数  2. 根据提示是否要保存(若不保存，退出程序时可再次选择是否保存)上一个表?是：1 否：0  0  请输入被切换到表的序号(输入数字0退出切表功能)  1 | 输出“当前操作的为表 1” | 输出“当前操作的为表 1” |
| 9. | 1.主界面输入12进入函数 | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” | 输出“ID:u3  Name:ccc  Score:3  ID:u4  Name:ddd  Score:4” |

16）测试函数：SaveList

测试步骤及结果如表2-16所示

表2-16 SaveList函数测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试步骤 | 测试输入 | 理论结果 | 运行结果 |
| 1 | 1.主界面输入1进入函数  2.按提示向表中输入要存储的内容 | 输出“当前已初始化表!”并且提示输入学生的信息 | 输出“当前已初始化表!”并且提示输入学生的信息 |
| 2 | 主界面输入学生信息：u1 aaa 1  u2 bbb 2 u3 ccc 3 # | 无输出，学生信息存储在表中 | 无输出，学生信息存储在表中 |
| 3. | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |
| 4. | 1.主界面输入16进入函数  2.根据提示输入：  是否要保存当前表?是：1 否：0  1  输入文件名称（不超过20个字符）:  second | 无输出，线性表内容保存在second.txt文件中 | 无输出，线性表内容保存在second.txt文件中 |
| 5. | 主界面输入2进入函数 | 输出“销毁线性表成功”！ | 输出“销毁线性表成功”！ |
| 6. | 1.主界面输入13进入函数  2.按提示输入要读取的txt文件second | 无输出，文件内容已被读取到当前操作的表中 | 无输出，文件内容已被读取到当前操作的表中 |
| 7. | 1.主界面输入12进入函数 | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” | 输出“ID:u1  Name:aaa  Score:1  ID:u2  Name:bbb  Score:2” |

## 实验小结

在本次实验中，让我对线性存储结构有了更深刻的了解。经过第一次顺序表的数据结构实验，本次实验的和上次实验内容相似，这次实验设计比较顺利，谢谢老师和助教对我的精心指导！

# 3基于二叉链表的二叉树实现

## 3.1 实验目的

二叉树是一种树型结构，二叉树的每个结点至多只有两棵子树。二叉树的二叉链表表示指的是数据元素在物理位置上不相邻的二叉树，而是通过附加指针域的方法将各个数据元素联系起来，使各个数据元素之间具有一定的逻辑关系。本实验要求封装一个基于二叉链表结构的二叉树ADT模块，提供二叉树ADT基本的、常见的20种操作，并且为该二叉树ADT提供文件的I/O方法。

## 3.2 模块设计

### 3.2.1 总体架构

根据实验的要求，给出整个模块的总体架构：

模块包括一个二叉树实现文件(.cpp)

模块还包括了一个栈ADT模块和一个队列ADT模块，以实现非递归遍历二叉树和层序遍历二叉树，同时在其他功能上也有对栈ADT模块的引用

模块的结构定义、结构声明、函数声明、函数指针以及相应的别名typedef均在实现文件的开头，函数实现在实现文件的尾部

模块的ADT操作实现被包括在实现文件中部

### 3.2.2 数据结构设计

数据结构的定义

根据实验的要求以及二叉树ADT的定义，给出基于C++语言实现的二叉树ADT的结构定义和别名：

typedef struct RooTNode{

BiTree child;

}RooTNode,\* RooTree;

其中，child指向该二叉树的根结点。同时为了防止指针写法的混乱，将RooTNode \* 定义为RooTree

二叉树ADT的结点BiTNode的定义和别名如下：

typedef struct BiTNode{

ElemType \*elem;

struct BiTNode \* lchild;

struct BiTNode \* rchild;

} BiTNode, \* BiTree;

其中，elem是ElemType类型的指针，用于指向学生的数据，struct BiTNode \* lchild是指向左孩子的指针， struct BiTNode \* rchild是指向右孩子的指针，同时为了防止指针写法的混乱，将BiTNode\*定义为BiTree。

在本次的实验中，采用了如下的结构作为数据元素：

typedef struct{

char id[EVERY];

char name[EVERY];

char phone[EVERY];

} ElemType;

该结构表示一个用户通讯录，其中，id表示的是学生的学号。Name表示的是学生的姓名，phone表示的是学生的联系方式。

### 3.2.3 ADT操作设计

根据实验的要求以及二叉树ADT的定义，该二叉树ADT应包括如下的操作：

InitTree（初始化），DestroyTree（销毁），CreateTree（构造），ClearTree（清空），TreeEmpty（判断树是否为空），TreeDepth（求树的深度），TreeRoot（取得根节点），Value（取得结点的值），Assign（为结点赋值），Parent（取得结点的父结点），LeftChild（取得结点的左孩子），RighitChild（取得结点的右孩子），LeftSibling（取得结点的左兄弟），RightSibling （取得结点的右兄弟），InsertChild（插入子树），DeleteChild（删除子树），PreOrderTraverse（前序遍历），InOrderTraverse（中序遍历），PostOrderTraverse（后序遍历）。

根据相关ADT操作的特性，给出如下基于C++语言的相关定义。

为了ADT操作统一性以及错误处理，在C++语言中给出如下的通用函数原型：

Status MethodName (Typename1 arg1, Typename2 arg2, ...);

其中，Status表示为预定义的枚举类型，用来表明函数执行的结果以及错误的处理，在C++语言中的定义如下：

#define BEF 0 //先序序列输入

#define MID 1 //中序序列输入

#define AFT 2 //后序序列输入

#define FALSE 0

#define OK 1

#define ERROR 0

#define EVERY 20

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 10

#define MAXQSIZE 100

#define NOTINIT 0

#define INIT 1

#define CREATE 2

#define ABANDON 3

BEF表示用户选择先序输入，MID表示用户选择中序输入，AFT表示用户选择后序输入，FALSE表示函数执行失败，ERROR表示函数异常退出，EVERY表示id、name和phone最大长度，STACK\_INIT\_SIZE表示为栈分配的空间，STACKINCREMENT表示栈空间不够时重新分配增加的空间，MAXQSIZE表示队列分配的空间，NOTINIT表示树未初始化，INIT表示树已初始化，CREATE表示树已创建，ABANDON表示树已被舍弃。

为了模块的拓展性，可能会用到访问元素并打印全部信息和访问函数只打印id所用到的函数PrintElem、VisitElem，以及它们所用到的访问函数Visit，在C++语言中定义如下：

Status PrintElem(ElemType \*e);

Status VisitElem(ElemType \*e);

基于以上的定义，给出所有ADT操作在C++语言中对应的函数原型：

Status InitBiTree(RooTree \*T);

Status DestroyBiTree(RooTree \*\*T,int moro,int \*where,int \*Isinit,int \*\*how,RooTree \*\*TS,int \*choice,int number);

Status CreateBiTree(BiTree \*T,int definition);

Status ClearBiTree (BiTree \*T);

Status BiTreeEmpty(BiTree T);

int BiTreeDepth(BiTree T);

Status Root(BiTree T);

Status Value(BiTree T,BiTNode e);

Status Assign(BiTree \*T,BiTNode &e,ElemType value);

Status Parent(BiTree T,BiTree e);

Status LeftChild(BiTree T,BiTree e);

Status RightChild(BiTree T,BiTree e);

Status LeftSibling(BiTree T,BiTree e);

Status RightSibling(BiTree T,BiTree e);

Status InsertChildBiTNode (BiTree \*T,BiTree p,int LR);

Status DeleteChild(BiTree \*T,BiTree p,int LR);

Status PreOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e));

Status InOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e));

Status PostOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e));

Status LevelOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e));

### 3.2.4 文件存储设计

根据实验的要求以及二叉树ADT的数据结构，该二叉树ADT应包括如下的文件操作：

WriteToFile(将二叉树写入文件)、ReadFromFile(从文件中读取二叉树)

根据ADT操作设计中的定义，给出文件存储操作在C语言中对应的函数原型：

Status Write(BiTree T,ofstream &outfile);

Status WriteToFile(BiTree T);

Status Read(BiTree \*T,ifstream &infile);

Status ReadFromFile(BiTree \*T);

### 3.2.5 配套演示系统设计

配套演示系统采用文本菜单界面，该系统每次从输入流接受一个数字，并且根据数字的不同执行不同的操作。

## 3.3 模块实现

### 3.3.1 开发环境

开发本系统时的操作环境为Windows 10

编程环境

IDE：Clion

Toolchains：

Environment：MinGW version：5.0

C Compiler：mingw32-gcc

C++ Compiler：mingw32-g++

### 3.3.2 ADT操作实现

InitTree(RooTree \*T)

该操作接受一个RooTree \*指针

该操作通过动态分配空间的方式分配一个RooTNode结构变量，将其指针存入\*T中，设计该二叉树的根结点为空

该二叉树初始化为空树

该操作的时间复杂度O(1)，空间复杂度O(1)。

DestroyBiTree(RooTree \*\*T,int moro,int \*where,int \*Isinit,int \*\*how,RooTree \*\*TS,int \*choice,int number)

该操作接受一个RooTree \*\*指针、一个int moro、一个int\* where、一个int\* Isinit、一个int \*\* how、一个RooTree \*\*指针、一个int\* choice和一个int number

该操作首先检查根结点是否存在，若不存在，则会告知用户该树未创建；否则，首先询问用户是否保存该树于文件中，然后执行用户相应的选择。最后调用递归函数ClearBiTree清空树。若该树为单树，销毁树；若该树为多树中的单树，则删除该单树。

该操作的时间复杂度为O(n)，空间复杂度为O(logn)。

CreateBiTree(BiTree \*T,int definition)

该操作接受一个BiTree\*指针和一个int definition

该操作首先判断用户选择的输入方式，然后根据用户的选择使用递归的方式依次从控制台读取用户输入的信息，存储到树中

该操作的时间复杂度为O(n)，空间复杂度为O(logn)。

ClearBiTree (BiTree \*T)

该操作接受一个BiTree\*指针

该操作和DestroyTree操作相似，不同之处是该操作只是简单的清空树且释放空间，并没有其他的功能

该操作使用递归的方式来释放空间

该操作的时间复杂度为O(n)，空间复杂度为O(logn)

BiTreeEmpty(BiTree T)

该操作接受一个BiTree指针

该操作通过判断根结点是否为空来判断该树是否为空树，若根结点为空，则告知用户该树为空树；否则，告知用户该树不为空

该操作的时间复杂度为O(1),空间复杂度为O(1)。

BiTreeDepth(BiTree T)

该操作接受一个BiTree指针，该函数的返回值为整型

该操作使用递归的方式计算二叉树的深度，并且将二叉树的深度作为返回值返回。递归程序会判断自己的左孩子和右孩子是否非空，对于非空的孩子调用下一次递归计算深度，并且返回更大的返回值+1

该操作的时间复杂度为O(n)，空间复杂度为O(logn)。

Root(BiTree T)

该操作接受一个BiTree指针

该操作首先判断根结点是否为空，若为空，则告知用户该树为空树；否则，打印根结点所包含的信息

该操作的时间复杂度为O(1)，空间复杂度为O(1)。

Value(BiTree T,BiTNode e)

该操作接受一个BiTree指针和一个BiTNode结构

该操作通过调用栈的方式来非递归寻找与e中id相同的结点

该操作首先初始化栈，然后非递归的方式先序遍历各个结点，若存在结点其id与e中id相同，则打印该结点的所有信息；否则，告知用户结点不存在，然后释放栈的空间

该操作的时间复杂度为O(n)，空间复杂度为O(n)。

Assign(BiTree \*T,BiTNode &e,ElemType value)

该操作接受一个BiTree指针、一个BiTNode引用和一个ElemType结构

该操作通过调用栈的方式来非递归寻找与value中id相同的结点

该操作首先初始化栈，然后非递归的方式先序遍历各个结点，若存在结点其id与value中id相同，则使得e指向该结点，打印该结点的所有信息，然后将value的值赋予给该结点，打印改变之后的结点信息；否则，告知用户结点不存在，然后释放栈的空间

该操作的时间复杂度为O(n)，空间复杂度为O(n)。

Parent(BiTree T,BiTree e)

该操作接受一个BiTree指针和一个BiTree指针

该操作通过调用栈的方式来非递归寻找与e中id相同的结点

该操作首先判断e中id是否与根结点中的id相同，若相同，则告知用户结点为根结点不存在双亲；否则，初始化栈，然后非递归的方式先序遍历各个结点，若存在结点的左孩子或者右孩子其id与e中id相同，则打印该结点左孩子或右孩子的所有信息；否则，告知用户结点不存在，然后释放栈的空间

该操作的时间复杂度为O(n)，空间复杂度为O(n)。

LeftChild(BiTree T,BiTree e)

该操作接受一个BiTree指针和一个BiTree指针

该操作通过调用栈的方式来非递归寻找与e中id相同的结点

该操作首先初始化栈，然后非递归的方式先序遍历各个结点，若存在结点的左孩子其id与e中id相同，则打印该结点左孩子的所有信息；否则，告知用户不存在，然后释放栈的空间

该操作的时间复杂度为O(n)，空间复杂度为O(n)。

RightChild(BiTree T,BiTree e)

该操作接受一个BiTree指针和一个BiTree指针

该操作通过调用栈的方式来非递归寻找与e中id相同的结点

该操作首先初始化栈，然后非递归的方式先序遍历各个结点，若存在结点的右孩子其id与e中id相同，则打印该结点右孩子的所有信息；否则，告知用户不存在，然后释放栈的空间

该操作的时间复杂度为O(n)，空间复杂度为O(n)。

LeftSibling(BiTree T,BiTree e)

该操作接受一个BiTree指针和一个BiTree指针

该操作通过调用栈的方式来非递归寻找与e中id相同的结点

该操作首先初始化栈，然后非递归的方式先序遍历各个结点，若存在结点的右孩子其id与e中id相同，则判断该结点的左孩子是否存在，若左孩子存在，则打印左孩子结点的信息，否则告知用户所找结点的左兄弟不存在；若不存在结点的右孩子其id与e中id相同，告知用户不存在，然后释放栈的空间

该操作的时间复杂度为O(n)，空间复杂度为O(n)。

RightSibling(BiTree T,BiTree e)

该操作接受一个BiTree指针和一个BiTree指针

该操作通过调用栈的方式来非递归寻找与e中id相同的结点

该操作首先初始化栈，然后非递归的方式先序遍历各个结点，若存在结点的左孩子其id与e中id相同，则判断该结点的右孩子是否存在，若右孩子存在，则打印右孩子结点的信息，否则告知用户所找结点的右兄弟不存在；若不存在结点的右孩子其id与e中id相同，告知用户不存在，然后释放栈的空间

该操作的时间复杂度为O(n)，空间复杂度为O(n)。

InsertChildBiTNode (BiTree \*T,BiTree p,int LR)

该操作接受一个BiTree指针、一个BiTree指针和一个int LR

该操作通过调用栈的方式来非递归寻找与p中id相同的结点

该操作首先初始化栈，然后按照用户选择输入的方式来接受用户再次创建的树，然后非递归的方式遍历各个结点，若存在结点其id与p中的id相同，通过用户的选择来执行相应的插入操作；若不存在结点其id与p中的id相同，则告知用户所找结点不存在，然后释放栈的空间

该操作的时间复杂度为O(n)，空间复杂度为O(n)。

DeleteChild(BiTree \*T,BiTree p,int LR)

该操作接受一个BiTree指针、一个BiTree指针和一个int LR

该操作通过调用栈的方式来非递归寻找与p中id相同的结点

该操作首先初始化栈，然后以非递归的方式遍历各个结点，若存在结点其id与p中的id相同，通过用户的选择来执行相应的删除操作；若不存在结点其id与p中的id相同，则告知用户所找结点不存在，然后释放栈的空间

该操作的时间复杂度为O(n)，空间复杂度为O(n)。

PreOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e))

该操作接受一个BiTree指针和一个Visit函数指针，按照先序遍历当前二叉树

该操作首先检查二叉树是否存在以及函数指针是否为空，如果存在且非空，则用递归方式先序遍历整棵二叉树。递归程序接收一个BiTree指针和一个Visit函数指针，对当前结点执行Visit函数，然后依次对于当前结点的左孩子和右孩子调用子递归。

该操作的时间复杂度为O(n)，空间复杂度为O(logn)。

InOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e))

该操作接受一个BiTree指针和一个Visit函数指针，按照中序遍历当前二叉树

该操作首先检查二叉树是否存在以及函数指针是否为空，如果存在且非空，则用递归方式中序遍历整棵二叉树。递归程序接收一个BiTree指针和一个Visit函数指针，对当前结点的左孩子调用子递归，执行Visit函数，然后对于当前结点的右孩子调用子递归。

该操作的时间复杂度为O(n)，空间复杂度为O(logn)。

PostOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e))

该操作接受一个BiTree指针和一个Visit函数指针，按照后序遍历当前二叉树

该操作首先检查二叉树是否存在以及函数指针是否为空，如果存在且非空，则用递归方式后序遍历整棵二叉树。递归程序接收一个BiTree指针和一个Visit函数指针，对当前结点的左孩子和右孩子调用子递归，最后对当前结点执行Visit函数。

该操作的时间复杂度为O(n)，空间复杂度为O(logn)。

LevelOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e))

该操作接受一个BiTree指针和一个Visit函数指针，按照层序遍历当前二叉树

该操作首先判断二叉树是否为空。若不为空，则执行一下操作：

1.初始化一个队列

2.将根结点入队

3.判断队列是否为空，若队列不为空；否则，循环结束。将队列中第一个结点出队，然后打印该结点的信息。判断出队结点的左孩子是否存在，若存在，则将左孩子入队。再判断出队结点的右孩子是否存在，若存在，则将右孩子入队。

最后释放队列的空间

该操作的时间复杂度为O(n)，空间复杂度为O(n)。

先序遍历树的非递归方式：

传入BiTree指针T

1. 访问结点T，并将其入栈

2. 判断T的左孩子是否为NULL，如果为NULL，则访问栈顶结点并且出栈，并且将其右结点设为T，执行1。如果不为NULL，则将T的左孩子设为T。

3. 若T为NULL且栈为空，遍历结束。

### 3.3.3 文件存储实现

Write(BiTree T,ofstream &outfile)

该操作通过递归的方式先序将树信息打印到文件中

1.接受一个BiTree指针T和一个输出流引用outfile。

2.向文件中输入内容时，首先判断该结点是否为空，若该结点为空，则向文件中打印字符串“#“。若该结点不为空，则打印该结点信息，打印id、name和phone时，每打印一次就换行继续打印。然后按照递归的方式先序打印该树的所有结点信息

3.当前树存储完成

该操作的时间复杂度是O(n)，空间复杂度是O(logn)。

WriteToFile(BiTree T)

接受用户输入所创建文件的名称，并且判断文件是否打开成功。若文件打开成功，则执行Write函数，否则，告知用户打开文件失败。

Read(BiTree \*T,ifstream &infile)

该操作通过递归的方式先序从文件中读取信息

1.接受一个BiTree\*指针T和一个输入流引用infile。

2.读取文件内容时，首先读取第一个字符串，若字符串为“#“，则使得当前结点\*T为空(nullptr)。若第一个字符串不为”#“，则动态分配一个结点的空间，将刚刚读取的信息和下面两个字符串依次赋给结点的id、name和phone。然后按照递归方式将文件内容先序构建树。

3.树构建完成

该操作的时间复杂度是O(n)，空间复杂度是O(logn)。

ReadFromFile(BiTree \*T)

接受用户输入所读取文件的名称，并且判断文件是否打开成功。若文件打开成功，则执行Read函数，否则，告知用户打开文件失败。

### 3.3.4 配套演示系统实现

外部函数实现

visit：

该操作接收一个ElemType一级指针，指向特定类型的数据元素，将其输出。

情况1：在该实验的演示系统中，该操作将ElemType数据结构的三个成员变量id、name和phone的值依次打印输出。

情况2：在该实验的演示系统中，该操作将ElemType数据结构的三个成员变量id的值打印输出。

该系统采用简单文本界面，实现的过程相对而言简单——首先输出系统菜单，然后从输入流中读入一个数字，在根据数字所对应的功能执行相应的代码。

## 3.4 模块测试

### 3.4.1 配套演示系统操作介绍

演示系统的界面如下图所示：



该演示系统操作简单，输入对应的数字并按下回车即可，输入1--25表明运行对应的功能，且相应操作的结果会被输出到屏幕上，输入0退出系统。

### 3.4.2 测试方案

根据程序要实现的功能，拟定了以下一个测试方案：

1. 初始化一个二叉树。
2. 构建一棵有11个元素的满二叉树。
3. 测试该树是否为空以及该树的深度。
4. 先序以及后序遍历该二叉树。
5. 将该二叉树写入文件。
6. 执行多树功能并创建两个二叉树
7. 创建第一个二叉树，二叉树的元素和结构与单树相同
8. 切换到第二个二叉树，创建树使树为有3个元素的满二叉树
9. 切换到第一个二叉树
10. 打印根结点的值。
11. 将第二课树插入进第一棵树的右子树。
12. 测试该树的深度。
13. 层序遍历该二叉树。
14. 销毁第一棵二叉树。
15. 切换到第二个二叉树
16. 从文件中加载单树于该树中。
17. 中序遍历该二叉树。
18. 打印该树树形
19. 退出程序

### 3.4.3 测试数据

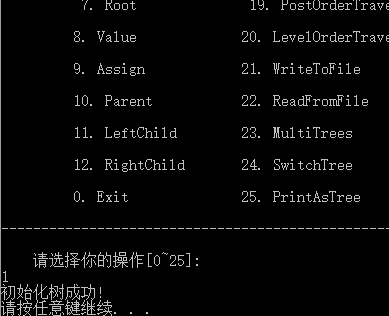
单树：A a 1 B b 2 D d 3 G g 4 # # H h 5 # # E e 6 # # C c 7 K k 8 # # F f 9 I i 10 # J j 11 # # #

多树中的第二颗树：

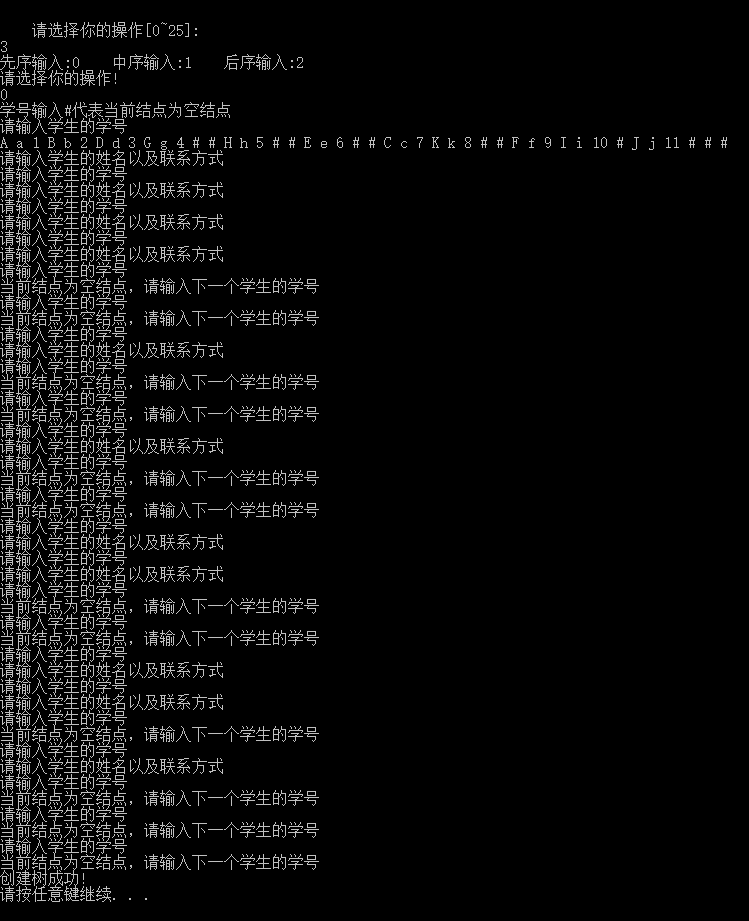
O o 12 P p 13 # # Q q 14 # #

### 3.4.4 测试结果

1. 初始化一个二叉树。

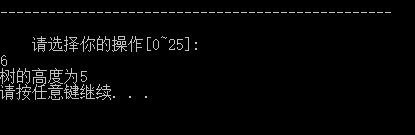


1. 构建一棵有11个元素的满二叉树。

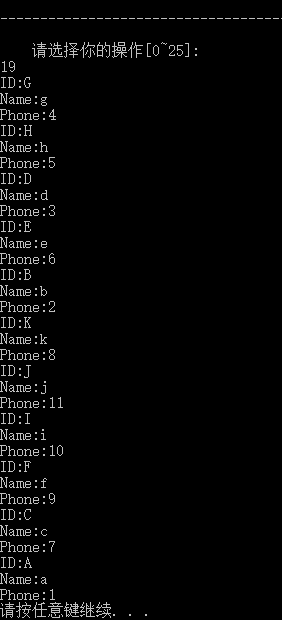
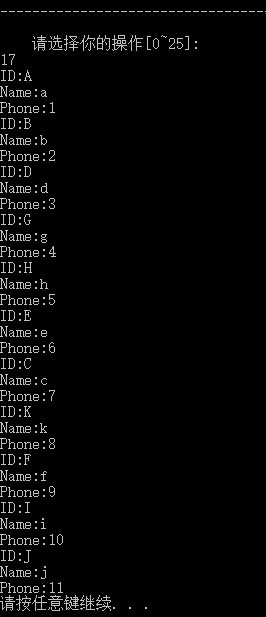


1. 测试该树是否为空以及该树的深度。

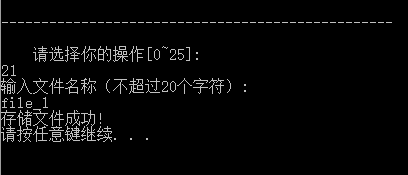




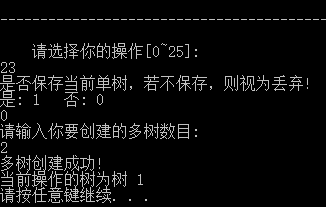
1. 先序以及后序遍历该二叉树。



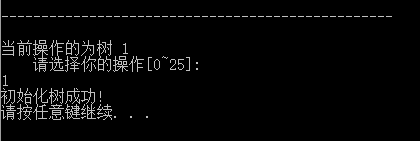
1. 将该二叉树写入文件。

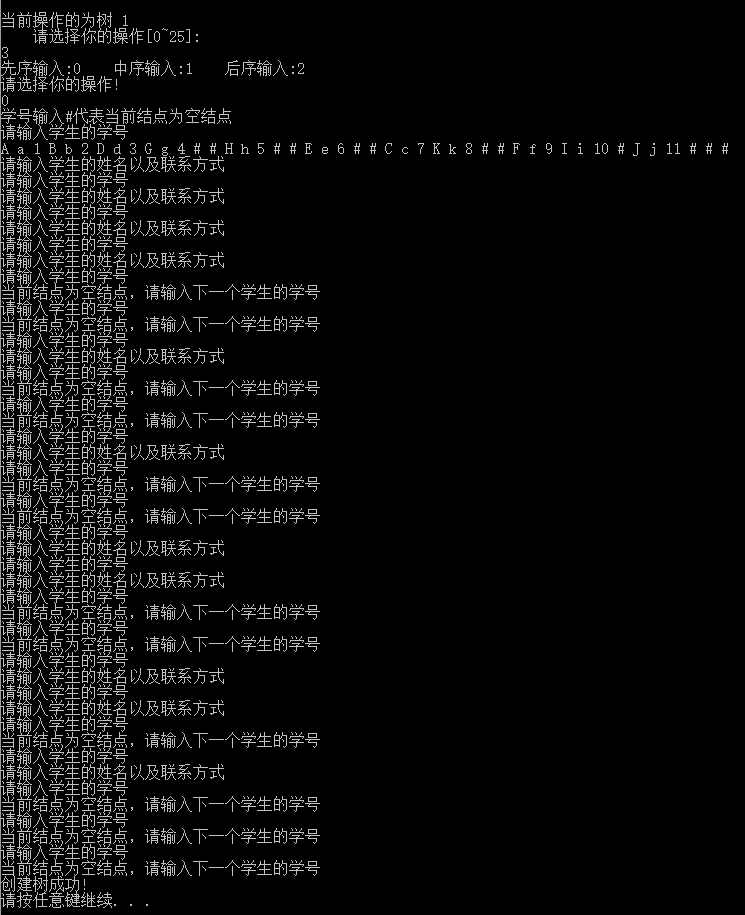


1. 执行多树功能并创建两个二叉树



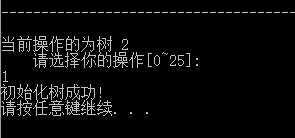
1. 创建第一个二叉树，二叉树的元素和结构与单树相同

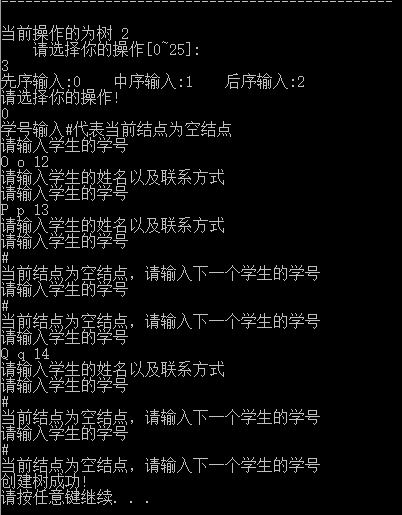




1. 切换到第二个二叉树，创建树使树为有3个元素的满二叉树



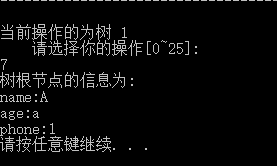




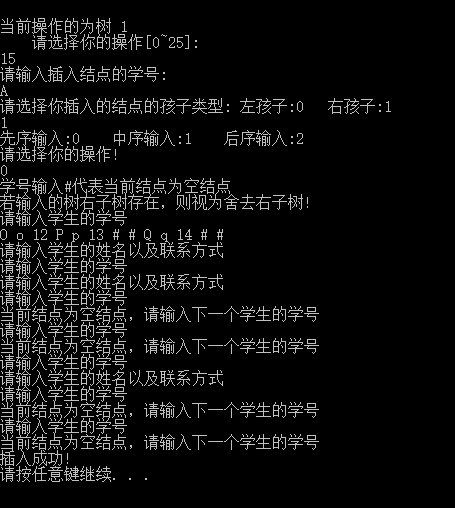
1. 切换到第一个二叉树



1. 打印根结点的值。

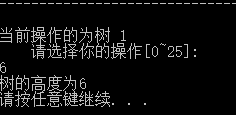


1. 将第二颗树的元素插入进第一棵树的右子树。

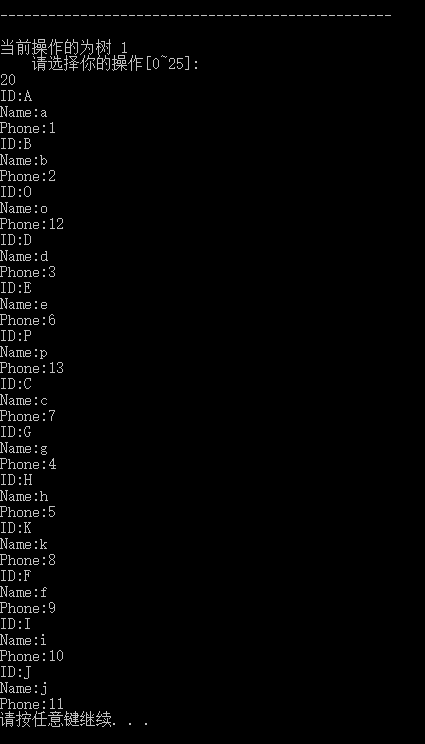


1. 测试该树的深度。

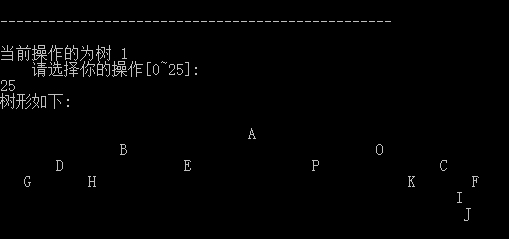
此时我们插入的树存在右子树，所以舍弃了插入树的右子树



1. 层序遍历该二叉树。



1. 以树形打印第一颗二叉树



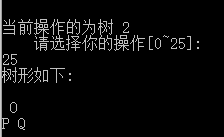
1. 销毁第一个二叉树



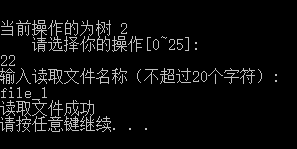
1. 自动切换到第二个二叉树



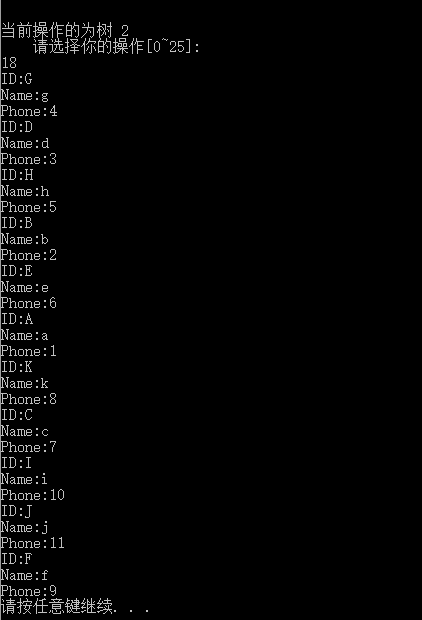
1. 树形打印该树



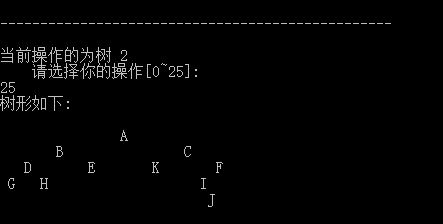
1. 从文件中加载单树于该树中。



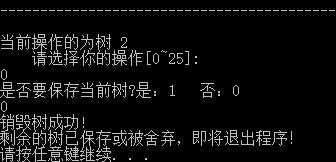
1. 中序遍历该二叉树。



1. 打印该树树形



1. 退出程序



## 3.5 实验小结

通过本次实验，我加深了对于基于二叉链表的二叉树的理解。

在本次实验中，我对递归有了更深层次的了解。实验中，有一个比较意外的收获是在清空树的时候，我重复释放内存空间。在不同的编译器产生了不同的反应，也因此让我对递归和编译器有了更深的理解。

此外，在这次的实验中，我多次使用栈进行遍历，对非递归遍历二叉树也有了更好的理解，本次实验加深了我对于二叉树操作的理解，也使得我的二叉树更加的全面和完整。

**4基于邻接表结构的图实现**

**4.1 问题描述**

一个图由顶点集V和边集E组成，每条边是一个点对（v，w），其中v，w属于V，如果点对是有序的，则图称为有向图，反之称为无向图。邻接表是对于每一个顶点，都有一个表保存所有邻接的顶点的一种图的表示方法。本实验要求封装一个基于邻接表结构的图ADT模块，提供图ADT基本的、常见的13种操作，并且为该图ADT提供文件的I/O方法。通过实验达到加深对图的概念、基本运算的理解、熟练掌握图的逻辑结构与物理结构的关系、以邻接表作为物理结构和熟练掌握图基本运算的实现。

**4.2 模块设计**

**4.2.1 总体架构**

根据实验的要求，给出整个模块的总体架构：

模块包括一个图实现文件（.cpp）

模块的宏定义define、结构定义、结构声明、函数指针以及相应的别名typedef、函数声明被包括在实现文件的首部

模块所声明的函数的函数体以及文件操作实现函数在实现文件的尾部

模块的ADT操作实现包括在实现文件中部

**4.2.2 数据结构设计**

数据结构的定义

根据实验的要求以及图ADT的定义，给出基于C++语言实现的图ADT的结构定义和别名：

typedef struct Graph {

struct ElemType \*elemtype;

int number;

} Graph, \*Graphptr;

其中elemtypes是一个ElemType结构类型的指针，指向顶点数组的首结点。number是一个整型，用来存储图中顶点数目

然后给出顶点ElemType的结构定义和别名：

typedef struct ElemType {

char id[MAXCHARLENGTH];

char name[MAXCHARLENGTH];

char birth[MAXCHARLENGTH];

ElemNext \*next\_elem;

} ElemType;

其中，id、name以及birth都是一个长度为MAXCHARLENGTH的字符类型数组，用于保存学生的信息。next\_elem是一个ElemNext类型的指针，用于指向顶点的邻接顶点。顶点保存的便是一个学生的联系方式。

最后给出邻接点的结构定义和别名：

typedef struct ElemNext {

ElemNext \*next\_elem;

int current;

} ElemNext;

其中，next\_elem是一个ElemNext类型的指针，用于指向顶点的邻接顶点。current是一个整型，用于保存邻接顶点的数组索引。

为了便于确定函数的返回状态，采用了如下的typedef：

typedef int Status;

Status是一个整型，用于表示函数的执行状态。

宏定义

#define MAXCHARLENGTH 20 表示每个字符数组的最大长度

#define OK 1 表示函数执行目标功能

#define FALSE 0 表示函数未执行目标功能

#define MAX 100 表示循环队列的最大空间

**4.2.3 ADT操作设计**

根据实验的要求以及图ADT的定义，该图ADT应包括如下的操作：

CreateGraph（创建图），DestroyGraph（销毁图），LocateVex（返回顶点位置信息），GetVex（得到顶点信息），PutVex（设置顶点信息），FirstAdjVex（求顶点的第一邻接点），NextAdjVex（求顶点相对某顶点的下一邻接点），InsertVex（插入顶点），DeleteVex（删除顶点），InsertArc（插入弧），DeleteArc（删除弧），DFSTraverse（深度优先遍历），BFSTraverse（广度优先遍历）。

为了模块的扩展性，DFSTraverse内部函数所用到的访问函数VisitFunc由外部提供，使用函数指针作为参数，在C++语言中的定义如下：

Status (\*VisitFunc)(ElemType \*e);

基于以上的定义，给出所有ADT操作在C++语言中对应得函数原型：

Status CreateVex(ElemType \*\*pElemType,Graphptr \*G);

Status CreateArc(ElemNext \*\*pElemNext,int number,ElemType \*elemType);

Status CreateGraph(Graphptr \*G,ElemType \*V,ElemNext \*VR);

Status DestroyGraph(Graphptr \*G);

Status LocateVex(Graphptr G, ElemType example);

Status GetVex(Graphptr G, ElemType example);

Status PutVex(Graphptr \*G, ElemType example, ElemType value);

Status FirstAdjVex(Graphptr G, ElemType example);

Status NextAdjVex(Graphptr G, ElemType example, ElemType w);

Status InsertVex(Graphptr \*G, ElemType value);

Status DeleteVex(Graphptr \*G, ElemType example);

Status InsertArc(Graphptr \*G, ElemType example, ElemType example\_to);

Status DeleteArc(Graphptr \*G, ElemType example, ElemType example\_to);

void DFS(Graphptr G, int v);

Status DFSTraverse(Graphptr G, Status(\*Visit)(ElemType \*e));

Status BFSTraverse(Graphptr G, Status(\*Visit)(ElemType \*e));

Status VisitElem(ElemType \*e);

**4.2.4 文件存储设计**

根据实验的要求以及图ADT的数据结构，该图ADT应包括如下的文件操作：

WriteToFile（将图写入文件）、ReadFromFile（从文件中读取图）

根据ADT操作设计中的定义，给出文件存储操作在C++语言中对应的函数声明：

Status WriteToFile(Graphptr G);

Status ReadFromFile(Graphptr \*G);

**4.2.5 配套演示系统**

配套演示系统采用文本菜单界面，该系统每次从输入流接受一个数字，并且根据数字的不同执行不同的操作。

**4.3 模块实现**

**4.3.1 开发环境**

开发本系统时的操作环境为Windows 10

编程环境

IDE：Clion

Toolchains：

Environment：MinGW version：5.0

C Compiler：mingw32-gcc

C++ Compiler：mingw32-g++

**4.3.2 ADT操作实现**

CreateVex：

该操作接受一个ElemType \*指针pElemType和一个Graphptr 指针，用来创建一个顶点集

该操作首先接受用户创建顶点的数目，并检查用户输入的数目是否合法。然后根据用户所要输入顶点的数目为所有的顶点分配一块连续的空间，并将顶点的数目保存在图中

该操作依次读取用户输入的顶点信息并且判断当前输入顶点在顶点集中是否已经存在。然后根据是否存在执行相应的结果

该操作的时间复杂度O(n3)，空间复杂度O(n)。

CreateArc：

该操作接受ElemNext \*指针\*pElemNext、一个整型 number和一个ElemType指针elemType

该操作首先根据用户顶点的数目创建一块连续的空间，然后每个顶点的索引对应空间中next\_elem指向其顶点的邻接顶点，每当为顶点创建邻接关系时，存储邻接顶点的空间都是单独分配，然后再用头插法插入到邻接链表中。创建邻接关系时，首先确定该邻接关系是否已经存在。然后根据是否存在执行相应的结果

该操作的时间复杂度以及空间复杂度需要根据创建关系的数目来确定。

CreateGraph：

该操作接收一个Graphptr 指针G、一个ElemType 指针V和一个ElemNext 指针VR，用所有的参数来构成一张图。

该操作将图中的指针elemtype指向顶点数组的首顶点，然后再将每个顶点中指针next\_elem指向其对应的邻接链表的首结点，然后创建图成功

该操作的时间复杂度O(1)，空间复杂度O(1)。

DestroyGraph：

该操作接收一个Graphptr指针G，将指针所指向的图销毁。

该操作首先检查指针以及图是否存在，非空且存在的话则会先按照循环依次释放所有的弧头结点所动态分配的空间，再循环依次释放所有顶点所动态分配的空间。最后将图的数目设置为0

该操作的时间复杂度为O(n+e)，空间复杂度为O(1)。

LocateVex：

该操作接收一个Graphptr指针G和一个ElemType结构example

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，如果存在顶点与传入的顶点相同，则打印顶点相应的序号，否则告知用户顶点不存在

该操作的时间复杂度为O(n)，空间复杂度为O(1)。

GetVex：

该操作接收一个Graphptr指针G和一个ElemType结构example

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，如果存在顶点与传入的顶点相同，则打印顶点所有的信息，否则告知用户顶点不存在

该操作的时间复杂服为O(n)，空间复杂度为O(1)。

PutVex：

该操作接收一个Graphptr指针G、一个ElemType结构example和一个ElemType结构value

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，如果存在顶点与传入的顶点相同，则打印顶点所有的信息并且将修改后的顶点信息赋给顶点，然后打印出来

该操作的时间复杂服为O(n)，空间复杂度为O(1)。

FirstAdjVex：

该操作接收一个Graphptr指针G和一个ElemType结构example

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，如果存在顶点与传入的顶点相同，则检查改顶点的第一个邻接顶点是否存在，若存在，则打印第一个邻接顶点的信息

该操作的时间复杂度是O(n)，空间复杂度是O(1)。

NextAdjVex：

该操作接收一个Graphptr指针G、一个ElemType结构example和一个ElemType结构example\_to

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，如果存在顶点与传入的顶点相同，则遍历顶点的所有邻接顶点，然后检查用户输入的邻接顶点在顶点的所有邻接顶点是否存在，若存在，则检查该邻接顶点是否存在后继邻接顶点；若存在，则打印后继邻接顶点的信息

该操作的时间复杂度是O(n+e)，空间复杂度是O(1)。

InsertVex：

该操作接收一个Graphptr指针G和一个ElemType结构example

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，检查是否用户输入的顶点在图中已经存在，若不存在，创建一块连续的空间，空间的大小比原顶点数组所占的空间多一个顶点的空间，然后将原顶点数组的所有信息(包括每个顶点的邻接顶点的指针)复制到新的顶点数组中，最后将新顶点插入到新顶点数组的尾部

该操作的时间复杂度是O(n)，空间复杂度是O(n)。

DeleteVex：

该操作接收一个Graphptr指针G和一个ElemType结构example

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，检查是否用户输入的顶点在图中已经存在，若存在，创建一块连续的空间，空间的大小比原顶点数组所占的空间少一个顶点的空间，然后删除每个顶点中所有指向被删除顶点有关的弧关系并且将每个顶点的所有邻接顶点中索引值大于被删除顶点的索引值减1，接着将原顶点数组除被删除顶点以外的所有顶点的所有信息(包括每个顶点的邻接顶点的指针)复制到新的顶点数组中

该操作的时间复杂度是O(n)，空间复杂度是O(n)。

InsertArc：

该操作接收一个Graphptr 指针G,、ElemType结构example和一个ElemType结构example\_to

该操作首先判断图是否为空，然后遍历所有顶点，检查是否存在用户输入的弧尾顶点和弧头顶点；若都存在，遍历弧尾结点的所有邻接顶点，检查是否存在邻接顶点与弧头顶点相同；若不存在，则用头插法将弧头顶点插入到弧尾顶点的邻接顶点中

该操作的时间复杂度是O(n+e)，空间复杂度是O(1)。

DeleteArc：

该操作接收一个Graphptr 指针G、ElemType结构example和一个ElemType结构example\_to

该操作首先判断图是否为空，然后遍历所有顶点，检查是否存在用户输入的弧尾顶点和弧头顶点；若都存在，遍历弧尾结点的所有邻接顶点，检查是否存在邻接顶点与弧头顶点相同；若存在，则将弧头顶点从弧尾顶点的邻接顶点中删除

该操作的时间复杂度是O(n+e)，空间复杂度是O(1)。

DFS：

该操作接收一个Graphptr 指针G和一个整型v

该操作首先判断索引值为v的顶点是否访问过；若没有访问过，则改变顶点的访问状态，并且访问索引值为v的顶点，然后对其邻接顶点调用递归

该操作的时间复杂度是O(n+e)，空间复杂度是O(n)。

DFSTraverse：

该操作接收一个Graphptr 指针G和一个Visit函数指针，深度优先遍历全图

该操作首先判断图是否为空，如果存在且非空则首先访问标志数组visited并置0，然后循环遍历该图的全部顶点，对没有访问过的顶点调用DFS函数遍历全图

该操作的时间复杂度是O(n)，空间复杂度是O(1)。

BFSTraverse：

该操作接收一个Graphptr 指针G和一个Visit函数指针，广度优先遍历全图

该操作首先判断图是否为空，如果存在且非空则首先访问标志数组visited并置0，然后初始化一个队列并且循环遍历图的所有顶点，在每一次循环中先判断该顶点是否被访问过，若没有被访问过则访问并将其入队，然后以队列不为空为条件循环，首先将元素出队，遍历其所有邻接顶点，若未被访问则访问并且将其入队，依此循环。

该操作的时间复杂度是O(n+e)，空间复杂度是O(n)。

**4.3.3 文件存储实现**

WriteToFile：

该操作接收一个Graphptr指针G实现将图存储进文件的功能

该操作首先检查图是否存在以及文件是否能够被打开／创建，如果存在且文件可以正常打开，则依次将图的顶点数，每个顶点的信息以及它所有邻接顶点信息写入文件中，最后关闭文件

该操作的时间复杂度是O(n+e)，空间复杂度是O(n+e)。

ReadFromFile：

该操作接收一个Graphptr指针G实现从文件读取图的功能

该操作首先检查图是否存在以及文件是否能够被打开，如果图不存在且文件可以正常打开，则创建图并且依次将文件中存储的顶点数，每个顶点的信息以及它所有邻接顶点信息写入图中，最后关闭文件

该操作的时间复杂度是O(n+e)，空间复杂度是O(n+e)。

**4.3.4 配套演示系统实现**

外部函数实现

VisitElem：

该操作接收一个ElemType结构指针，将指针指向的顶点信息输出。

在该实验的演示系统中，该操作将ElemType数据结构的三个成员变量id，name，birth的值依次打印输出。

演示功能实现

该系统采用简单文本界面，实现的过程相对而言简单——首先输出系统菜单，然后从输入流中读入一个数字，在根据数字所对应的功能执行相应的代码即可。

**4.4 模块测试**

**4.4.1 配套演示系统操作介绍**

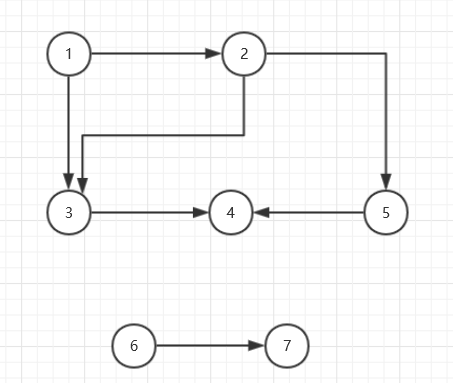
演示系统的界面如下图所示：

该演示系统操作简单，输入对应的数字并按下回车即可，输入1-17表明运行对应的功能，且相应操作的结果会被输出到屏幕上，输入0退出系统。

**4.4.2 测试方案**

测试数据应兼顾包括有效数据无效数据两个方面，有效数据旨在验证系统功能的正确性，而无效甚至是错误的数据旨在验证系统的鲁棒性。根据以上的要求，拟定了以下一个测试方案：

1. 在未创建图的情况下执行销毁图，插入顶点、深度优先遍历
2. 创建一个有7个顶点的有向图，有7条边

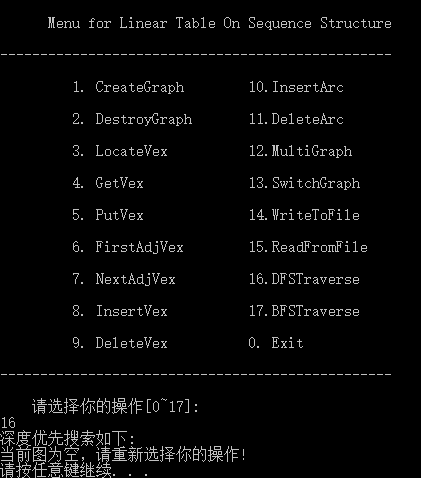


1. 确定ID为5的顶点的位置信息（比较是否正确）
2. 确定ID为2的第一和第二邻接顶点
3. 取得ID为1的顶点的值
4. 修改ID为1的顶点的值
5. 插入一个新的顶点8和一条新的弧（8，6）
6. 广度优先遍历该图
7. 保存该图
8. 关闭并重新打开后读取该图
9. 深度优先遍历该图
10. 删除下标为6的顶点
11. 广度优先遍历该图

测试创建图的时候输入为：1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6 7 7 7 1 1 2 3 \* 2 3 5 \* 3 4 \* 5 4 \* 6 7 \* #

**4.4.3 测试结果**

1. 在未创建图的情况下执行销毁图、插入顶点、深度优先遍历



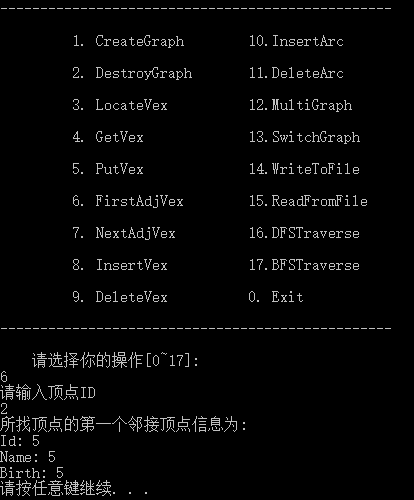
1. 创建一个有7个顶点的有向图，有7条边



1. 确定ID为5的顶点的位置信息（比较是否正确）

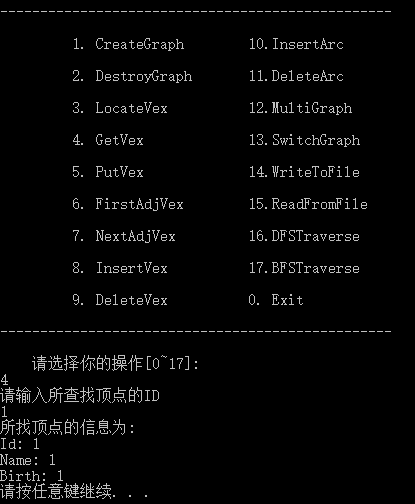


1. 确定ID为2的第一和第二邻接顶点

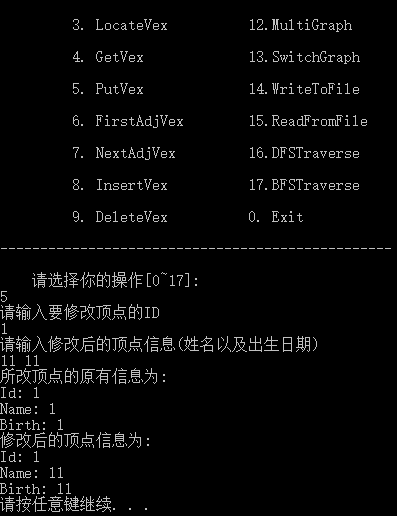


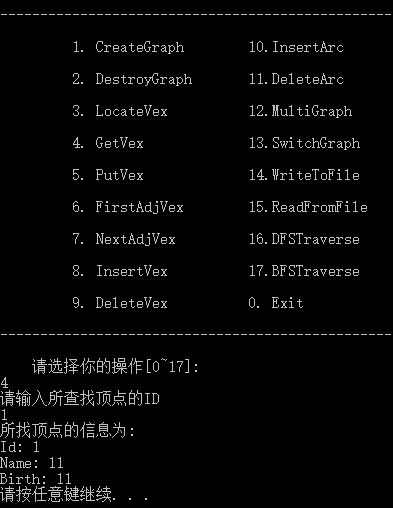


1. 取得ID为1的顶点的值

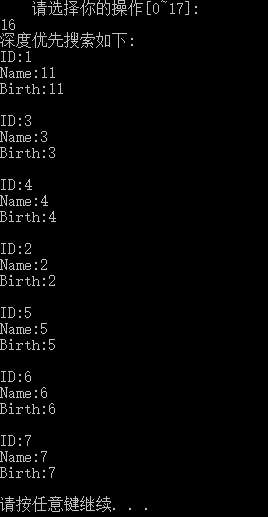


1. 修改ID为1的顶点的值



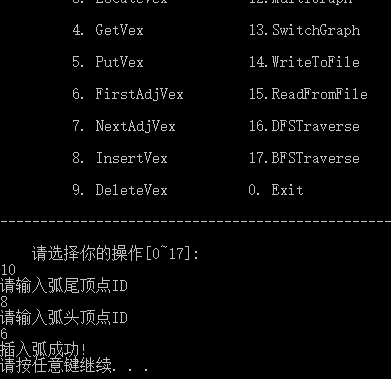


1. 深度优先遍历该图

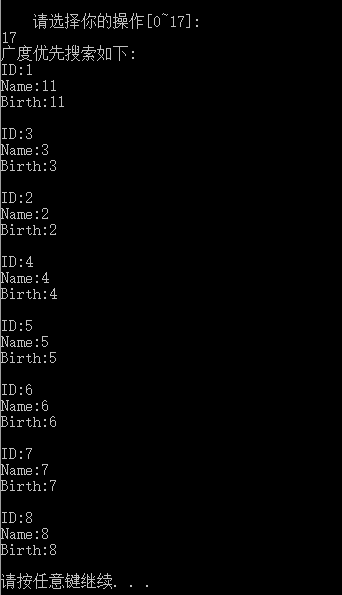


1. 插入一个新的顶点8和一条新的弧（8，6）

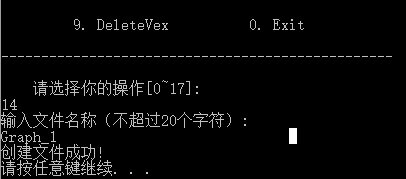




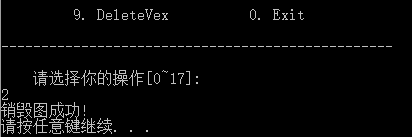
1. 广度优先遍历该图

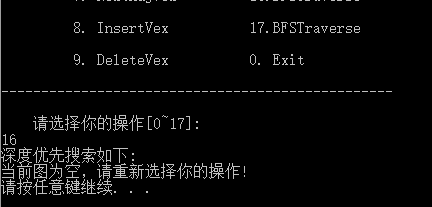


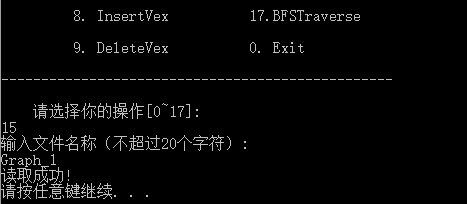
1. 保存该图



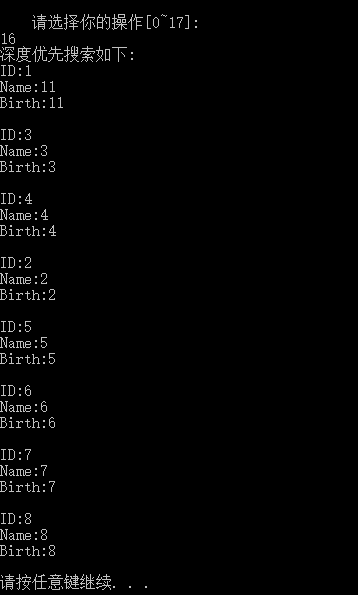
1. 销毁图后先深度优先遍历，然后重新读取该图







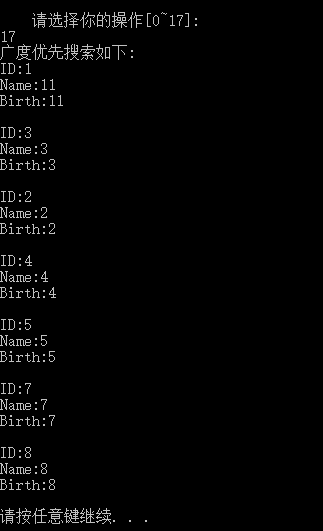
1. 深度优先遍历该图



1. 删除下标为6的顶点



1. 广度优先遍历该图



**4.5 实验小结**

通过本次实验，我加深了对于基于邻接表的图的理解,熟悉了如何用邻接表表示图，并且对图的各种操作加深了印象，更好地理解了图这种数据结构。

通过了本次实验，我收获颇丰，但是也仍然存在一些不足。在实验的过程中也有许多没有考虑到的地方，例如：在删除顶点的时候，一开始只考虑到删除所有与顶点有关的弧关系，但没有考虑到部分邻接顶点的索引值也要相应地改变，所以导致了程序出现了异常。

# 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

/\*\*

\* 本次代码实现的原理如下：

\* 开辟一段连续的内存空间来储存指针

\* 每个指针指向不同的学生

\* 每个学生信息的储存均为一段连续的内存空间

\*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <iostream>

#include <cstring>

#include <fstream>

using namespace std;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

#define EVERY 20

typedef int status; //数据元素类型定义

typedef struct {

char id[EVERY];

char name[EVERY];

float score;

} ElemType;

typedef struct { //顺序表的定义

ElemType \*elem = nullptr;

int length;

int listsize;

} SqList;

status IntiaList(SqList \*L);

status DestroyList(SqList \*L);

status ClearList(SqList \*L);

status ListEmpty(SqList L);

int ListLength(SqList L);

status GetElem(SqList L, char \*stu, ElemType \*e);

status LocateElem(SqList L, ElemType e,int &a);

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

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

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

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

status ListTrabverse(SqList L);

status ReadFromFile(SqList \*L, int moro, int \*Isinit, int \*hw, int wh);

status WriteToFile(SqList \*L);

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

int main(void) {

SqList Q;

SqList \* L = &Q;

int choice = 1;

int Isinit = 0;

int hm = 0;//用来保存多表的数目

int wh = 0;//定位当前表在多表中哪一个表

int moro = 0;//判断是多表操作还是单表操作

SqList \* multilist = nullptr;

int \*hw = nullptr;//保存表的状态 0：未初始化 1：初始化 2：已保存 3：不保存

while (choice) {

cout << "\n\n" << endl;

cout <<" Menu for Linear Table On Sequence Structure \n" << endl;

cout <<"-------------------------------------------------\n" << endl;

cout <<" 1. IntiaList 9. NextElem\n" << endl;

cout <<" 2. DestroyList 10. ListInsert\n" << endl;

cout <<" 3. ClearList 11. ListDelete\n" << endl;

cout <<" 4. ListEmpty 12. ListTrabverse\n" << endl;

cout <<" 5. ListLength 13. ReadFromFile\n" << endl;

cout <<" 6. GetElem 14. MultiList\n" << endl;

cout <<" 7. LocateElem 15. SwitchList\n" << endl;

cout <<" 8. PriorElem 16. SaveList\n" << endl;

cout <<" 0. Exit\n" << endl;

cout <<"-------------------------------------------------\n" << endl;

if(moro == 1)

cout << "当前操作的为表 " << wh << endl;

cout <<" 请选择你的操作[0~15]:" << endl;

cin >> choice;

switch (choice) {

case 1: {

if(moro == 1){

Isinit = hw[wh - 1];

}

if(Isinit == 1){

cout << "当前已初始化表!" << endl;

break;

}

if (IntiaList(L)) {

if(moro == 1) {

hw[wh - 1] = 1;

Isinit = hw[wh - 1];

}

else

Isinit = 1;

int number;

cout << "线性表创建成功！" << endl;

cout << "输入学生的信息，依次输入学号、姓名、得分(学生的个数不大于" << LIST\_INIT\_SIZE << ")，学号中输入符号#结束输入" << endl;

for (number = 0; number <= LIST\_INIT\_SIZE; number++) {

cout << "请输入第" << number + 1 << "个学生的信息" << endl;

cin.sync();

cin >> (\*L).elem[number].id;

if (number == LIST\_INIT\_SIZE && strcmp((\*L).elem[number].id,"#") != 0) {

cout << "超出范围!" << endl;

break;

}

if (strcmp((\*L).elem[number].id,"#") == 0)

break;

cin >> (\*L).elem[number].name;

cin >> (\*L).elem[number].score;

(\*L).length++;

}

} else

cout << "线性表创建失败！" << endl;

system("pause");

break;

}

case 2: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

break;

}

if (DestroyList(L)) {

cout << "销毁顺序表成功" << endl;

Isinit = 0;

if(moro == 1){

hw[wh - 1] = 3;

}

}

system("pause");

break;

}

case 3: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

if (ClearList(L))

cout << "清空顺序表成功" << endl;

system("pause");

break;

}

case 4: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

if (ListEmpty(\*L))

cout << "顺序表为空" << endl;

else {

cout << "顺序表不为空" << endl;

}

system("pause");

break;

}

case 5: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

cout << "顺序表的长度为 " << ListLength(\*L) << endl;

system("pause");

break;

}

case 6: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

char stu[EVERY];

ElemType s;

ElemType \*e = &s;

cout << "请输入目标学生的学号:" << endl;

cin >> stu;

if (GetElem(Q, stu, e)) {

cout << "学号: " << stu << " 学生的信息为:\n"

<< "ID:" << (\*e).id << "\n"

<< "Name:" << (\*e).name << "\n"

<< "Score:" << (\*e).score << endl;

} else {

cout << "查无此人" << endl;

}

system("pause");

break;

}

case 7:{

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

ElemType s;

cout << "请输入定位学生的学号:" << endl;

cin >> s.id;

int a = 0;

int &b = a;

if (LocateElem(Q,s,b))

cout <<"目标学生成绩表中的位序为" << b <<endl;

else

cout << "查无此人" << endl;

system("pause");

break;

}

case 8: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

cout << "输入成绩表中被查找学生的下面学生的学号" << endl;

ElemType cur\_e;

ElemType s;

ElemType \*pre\_e = &s;

cin >> cur\_e.id;

if (PriorElem(Q, cur\_e, pre\_e)) {

cout << "存在你所查找的学生:\n"

<< "ID:" << (\*pre\_e).id << "\n"

<< "Name:" << (\*pre\_e).name << "\n"

<< "Score:" << (\*pre\_e).score << endl;

} else

cout << "不存在被查找学生" << endl;

system("pause");

break;

}

case 9: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

cout <<"输入成绩表中被查找学生的上面学生的学号" << endl;

ElemType cur\_e;

ElemType s;

ElemType \*next\_e = &s;

cin >> cur\_e.id;

if (NextElem(Q, cur\_e, next\_e)) {

cout << "存在你所查找的学生:\n"

<< "ID:" << (\*next\_e).id << "\n"

<< "Name:" << (\*next\_e).name << "\n"

<< "Score:" << (\*next\_e).score << endl;

} else

cout << "不存在被查找学生" << endl;

system("pause");

break;

}

case 10: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

cout << "输入你想插入的位置：" << endl;

int i;

cin >> i;

cout << "输入你想插入的学生的学生信息：" << endl;

ElemType s;

cin >> s.id >> s.name >> s.score;

if (ListInsert(L, i, s))

cout << "插入成功!" << endl;

else

cout << "输入的位序不合法，请重新输入" << endl;

system("pause");

break;

}

case 11:{

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

cout << "请输入你想删除的学生的学号：" << endl;

ElemType s;

cin >> s.id;

ElemType ds;

ElemType \*e1 = &ds;

if (ListDelete(L, s, e1)) {

cout << "你删除的学生的信息为："

<< "ID:" << (\*e1).id << "\n"

<< "Name:" << (\*e1).name << "\n"

<< "Score:" << (\*e1).score << endl;

cout << "删除成功！" << endl;

} else

cout << "不存在此学生" << endl;

system("pause");

break;

}

case 12: {

if (Isinit != 1) {

cout << "线性表未创建!" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

if (!ListTrabverse(\*L))

cout << "线性表是空表！" << endl;

system("pause");

break;

}

case 13: {

cout << "请输入你想读取的文件名" << endl;

ReadFromFile(L,moro,&Isinit,hw,wh);//表已经被初始化

system("pause");

break;

}

case 14: {

if(moro == 0 && Isinit == 1){

cout << "是否保存(若不保存，则单表中的内容将被清空)当前的单表? 是：1 否：0" << endl;

int i;

cin >> i;

if (i) {

WriteToFile(L);

} else

DestroyList(L);

}

if(moro == 1){

cout << "当前已创建多表，无需再创建!" << endl;

system("pause");

break;

}

cout << "请输入同时操作表的数目" << endl;

int a;

cin >> hm;

if(hm > 0) {

moro = 1;

multilist = (SqList \* )malloc( hm \* sizeof(SqList));

hw = (int \*)malloc( hm \* sizeof(int));

for (a = 0; a < hm; a++) {

hw[a] = 0;

}

wh = 1;

L = &(multilist[0]);

Isinit = hw[0];

cout << "操作成功！当前操作的表为表 1" << endl;

}else{

cout << "输入的数目不合法，将退出本功能！" << endl;

}

system("pause");

break;

}

case 15: {

if( hm == 0){

cout << "不存在多表！" << endl;

system("pause");

break;

}

//切换表的时候询问上一个表是否保存，若不保存，后面退出的时候可以选择保存

if(moro == 1)

Isinit = hw[wh - 1];

if(Isinit == 1){

cout << "是否要保存(若不保存，退出程序时可再次选择是否保存)上一个表?是：1 否：0" << endl;

int i;

cin >> i;

if (i) {//若文件已经被保存，则切换表的时候无法再次切换到该表

char filename[20];

cout << "输入文件名称（不超过20个字符）:" << endl;

cin >> filename;

strcat(filename,".txt");

ofstream outile;

outile.open(filename,std::ios::out);

if(outile){

if(moro == 1)

hw[wh -1] = 2;

int number;

for(number = 0;number < (\*L).length;number++) {

outile << (\*L).elem[number].id << endl;

outile << (\*L).elem[number].name << endl;

outile << (\*L).elem[number].score << endl;

}

outile.close();

} else

cout << "创建文件失败!" << endl;

}

}

cout << "请输入被切换到表的序号(输入数字0退出切表功能)" << endl;

int a;

while(true){

cin >> a;

if( a == 0)

break;

if( hw[a - 1] == 2 && a <= hm && a > 0){

cout << "此表已被保存！无法切换到该表！请重新输入！" << endl;

}

if( hw[a - 1] == 3 && a <= hm && a > 0){

cout << "此表已被舍弃！无法切换到该表！请重新输入！" << endl;

}

if(a > 0 && a <= hm && hw[a - 1] != 2) {

wh = a;

Isinit = hw[wh - 1];

L = &(multilist[wh - 1]);

break;

}

cout << "输入的表序不存在，请在1--" << hm << "选择输入" << endl;

}

system("pause");

break;

}

case 16: {

int i;

cout << "是否要保存当前表?是：1 否：0" << endl;

cin >> i;

if (i) {

WriteToFile(L);

}

system("pause");

break;

}

case 0: {

int i;

if(moro == 1)//让Isinit为多表中被操作表的状态

Isinit = hw[wh - 1];

if (Isinit != 1) {

if(Isinit == 2)

cout << "此表已保存！" << endl;

else if(Isinit == 0){

cout << "线性表未初始化！" << endl;

cout << "即将退出本功能!" << endl;

if(moro == 1)

hw[ wh -1 ] = 3;

}

}

else{

cout << "是否要保存当前表?是：1 否：0" << endl;

cin >> i;

if (i) {

if (moro == 1)

hw[wh - 1] = 2;

WriteToFile(L);

} else{

//将表改变不保存的状态，即删除该表的存在

if (moro == 1)

hw[wh - 1] = 3;

}

}

//判断是否所有的表被保存，

//若没有，则切换到未保存的表。

if(moro == 1) {

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

if(hw[i] == 1 || hw[i] == 0){

Isinit = hw[i];

L = &multilist[i];

wh = i + 1;

choice = 17;//只是为了再次进入循环，而不是具体的操作命令

break;

}

}

if(choice == 0){

cout << "剩余的表均被舍弃(退出)，即将退出程序！" << endl;

}

}

system("pause");

break;

}

default:

cout << "指令不存在，请重新输入！" << endl;

}//end of switch

}//end of while

cout << "欢迎下次再使用本系统!" << endl;

return 1;

}//end of main()

status IntiaList(SqList \*L) {

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

if (!(\*L).elem)

exit(OVERFLOW);

(\*L).length = 0;

(\*L).listsize = LIST\_INIT\_SIZE;

return OK;

}

status DestroyList(SqList \*L) {

if ((\*L).elem != nullptr) {

free((\*L).elem);

return TRUE;

} else {

return ERROR;

}

}

status ClearList(SqList \*L) {

(\*L).length = 0;

return TRUE;

}

status ListEmpty(SqList L) {

if (L.length == 0)

return TRUE;

else

return FALSE;

}

int ListLength(SqList L) {

return L.length;

}

status GetElem(SqList L, char \*stu, ElemType \*e) {

int a;

for(a = 0;a < L.length;a++) {

if (strcmp(stu,L.elem[a].id) == 0) {

\*e = L.elem[a];

return OK;

}

}

return FALSE;

}

status LocateElem(SqList L, ElemType e,int &a) {

int i;

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

if (strcmp(L.elem[i].id,e.id) == 0) {

a = i + 1;

return TRUE;

}

}

return FALSE;

}

status PriorElem(SqList L, ElemType cur\_e, ElemType \*pre\_e) {

int i;

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

if (strcmp(L.elem[i].id,cur\_e.id) == 0 && i != 0) {

\*pre\_e = L.elem[i - 1];

return TRUE;

}

}

return FALSE;

}

status NextElem(SqList L, ElemType cur\_e, ElemType \*next\_e) {

int i;

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

if (strcmp(L.elem[i].id,cur\_e.id) == 0 && i != (L.length - 1)) {

\*next\_e = L.elem[i + 1];

return TRUE;

}

}

return ERROR;

}

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

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

return ERROR;

if ((\*L).length >= (\*L).listsize) {

ElemType \*newbase;

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

if (!newbase)

exit(OVERFLOW);

(\*L).elem = newbase;

(\*L).listsize += LISTINCREMENT;

}

ElemType \*q, \*p;

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

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

for (; p >= q; --p)

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

\*q = e;

++((\*L).length);

return OK;

}

status ListDelete(SqList \*L, ElemType st, ElemType \*e) {

int i;

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

if(strcmp(st.id,(\*L).elem[i].id) == 0)

break;

}

if((\*L).length == i)

return FALSE;

ElemType \*p, \*q;

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

\*e = \*p;

q = &((\*L).elem[(\*L).length - 1]);

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

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

--((\*L).length);

return OK;

}

status ListTrabverse(SqList L) {

if (!L.elem || L.length == 0)

return ERROR;

int i;

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

cout << "ID:"<< L.elem[i].id << "\n"

<< "Name:" << L.elem[i].name << "\n"

<< "Score:" << L.elem[i].score << "\n" << endl;

}

return TRUE;

}

status ReadFromFile(SqList \*L, int moro, int \*Isinit, int \*hw, int wh) {

char filename[20];

cin >> filename;

strcat(filename, ".txt");

ifstream infile;

infile.open(filename, std::ios::in);

if (infile) {

//根据是多表操作还是单表操作来改变它们的状态，

//从而在前面判断是否被初始化

if(moro == 1)

\*Isinit = hw[wh - 1];

if(\*Isinit == 1){

ClearList(L);

}

if(\*Isinit == 0 && moro == 1) {

IntiaList(L);

\*Isinit = hw[wh - 1] = 1;

} else if(\*Isinit == 0 && moro == 0){

IntiaList(L);

\*Isinit = 1;

}

int a;

for (a = 0;; a++) {

infile >> (\*L).elem[a].id >> (\*L).elem[a].name >> (\*L).elem[a].score;

if(infile.peek() == EOF )

break;

++((\*L).length);

if ((\*L).length >= (\*L).listsize) {

ElemType \*newbase;

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

if (!newbase)

exit(OVERFLOW);

(\*L).elem = newbase;

(\*L).listsize += LISTINCREMENT;

}

}

} else {

cout << "不存在此文件!" << endl;

}

infile.close();

return TRUE;

}

status WriteToFile(SqList \*L) {

char filename[20];

cout << "输入文件名称（不超过20个字符）:" << endl;

cin >> filename;

strcat(filename,".txt");

ofstream outile;

outile.open(filename,std::ios::out);

if(outile){

int number;

for(number = 0;number < (\*L).length;number++) {

outile << (\*L).elem[number].id << endl;

outile << (\*L).elem[number].name << endl;

outile << (\*L).elem[number].score << endl;

}

outile.close();

} else

cout << "创建文件失败!" << endl;

return TRUE;

}

# 附录B

/\*\*

\* 本次代码实现的原理如下：

\* 开辟一段连续的内存空间来储存指针

\* 每个指针指向不同的学生，第一个结点不存储任何信息

\* 每个学生信息的为链式存储

\*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <iostream>

#include <cstring>

#include <fstream>

using namespace std;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define EVERY 20

typedef int status; //数据元素类型定义

typedef struct ElemType{//数据元素的定义，为链式存储

char id[EVERY] = "";

char name[EVERY] = "";

float score = 0;

struct ElemType \*next = nullptr;

} ElemType;

typedef struct LNode{//每个表的信息

ElemType \*elem = nullptr;

int length = 0;

} LNode, \*LinkList;

status IntiaList(LinkList \*L);

status DestroyList(LinkList \*L);

status ClearList(LinkList \*L);

status ListEmpty(LNode L);

int ListLength(LNode L);

status GetElem(LNode L, char \*stu, ElemType \*e);

status LocateElem(LNode L, ElemType e,int &a);

status PriorElem(LNode L, ElemType cur\_e, ElemType \*pre\_e);

status NextElem(LNode L, ElemType cur\_e, ElemType \*next\_e);

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

status ListDelete(LinkList \*L, ElemType st, ElemType \*e);

status ListTrabverse(LNode L);

status ReadFromFile(LinkList \*L, int moro, int \*Isinit, int \*hw, int wh);

status WriteToFile(LinkList \*L);

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

int main(void) {

LNode Q;

LinkList L = &Q;

int choice = 1;

int Isinit = 0;

int hm = 0;//用来保存多表的数目

int wh = 0;//定位当前表在多表中哪一个表

int moro = 0;//判断是多表操作还是单表操作

LinkList LinkLists;

int \*hw = nullptr;//保存表的状态 0：未初始化 1：初始化 2：已保存 3：不保存

while (choice) {

cout << "\n\n" << endl;

cout <<" Menu for Linear Table On Sequence Structure \n" << endl;

cout <<"-------------------------------------------------\n" << endl;

cout <<" 1. IntiaList 9. NextElem\n" << endl;

cout <<" 2. DestroyList 10. ListInsert\n" << endl;

cout <<" 3. ClearList 11. ListDelete\n" << endl;

cout <<" 4. ListEmpty 12. ListTrabverse\n" << endl;

cout <<" 5. ListLength 13. ReadFromFile\n" << endl;

cout <<" 6. GetElem 14. MultiList\n" << endl;

cout <<" 7. LocateElem 15. SwitchList\n" << endl;

cout <<" 8. PriorElem 16. SaveList\n" << endl;

cout <<" 0.Exit\n" << endl;

cout <<"-------------------------------------------------\n" << endl;

if(moro == 1)

cout << "当前操作的为表 " << wh << endl;

cout <<" 请选择你的操作[0~15]:" << endl;

cin >> choice;

switch (choice) {

case 1: {

if(moro == 1){

Isinit = hw[wh - 1];

}

if(Isinit == 1){

cout << "当前已初始化表!" << endl;

break;

}

if (IntiaList(&L)) {

if(moro == 1) {

hw[wh - 1] = 1;

Isinit = hw[wh - 1];

}

else

Isinit = 1;

int number;

cout << "线性表创建成功！" << endl;

cout << "输入学生的信息，依次输入学号、姓名、得分,学号中输入符号#结束输入" << endl;

ElemType \*q = L -> elem;

ElemType \*p = L -> elem -> next = (ElemType \*)malloc(sizeof(ElemType));

for(number = 0;;number++) {

cout << "请输入第" << number + 1 << "个学生的信息" << endl;

cin.sync();

cin >> p -> id;

if (strcmp(p -> id,"#") == 0) {

free(p);

q -> next = nullptr;

break;

}

cin >> p -> name;

cin >> p -> score;

(\*L).length++;

p -> next = (ElemType \*)malloc(sizeof(ElemType));

q = p;

p = p -> next;

}

} else

cout << "线性表创建失败！" << endl;

system("pause");

break;

}

case 2: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

if (DestroyList(&L)) {

cout << "销毁线性表成功!" << endl;

if(moro == 1) {

hw[wh - 1] = 3;

Isinit = hw[wh - 1];

}

else

Isinit = 0;

}

system("pause");

break;

}

case 3: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

if (ClearList(&L))

cout << "清空线性表成功!" << endl;

system("pause");

break;

}

case 4: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

if (ListEmpty(\*L))

cout << "线性表为空" << endl;

else {

cout << "线性表不为空" << endl;

}

system("pause");

break;

}

case 5: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

cout << "线性表的长度为 " << ListLength(\*L) << endl;

system("pause");

break;

}

case 6: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

char stu[EVERY];

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

cout << "请输入目标学生的学号:" << endl;

cin >> stu;

if (GetElem(\*L, stu, e)) {

cout << "学号: " << stu << " 学生的信息为:\n"

<< "ID:" << e -> id << "\n"

<< "Name:" << e -> name << "\n"

<< "Score:" << e -> score << endl;

} else {

cout << "查无此人" << endl;

}

system("pause");

break;

}

case 7:{

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

ElemType s;

cout << "请输入定位学生的学号:" << endl;

cin >> s.id;

int a = 0;

int &b = a;

if (LocateElem(\*L,s,b))

cout <<"目标学生成绩表中的位序为" << b <<endl;

else

cout << "查无此人" << endl;

system("pause");

break;

}

case 8: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

cout << "输入成绩表中被查找学生的下面学生的学号" << endl;

ElemType cur\_e;

ElemType \*pre\_e = (ElemType \*)malloc(sizeof(ElemType));

cin >> cur\_e.id;

if (PriorElem(\*L, cur\_e, pre\_e)) {

cout << "存在你所查找的学生:\n"

<< "ID:" << pre\_e -> id << "\n"

<< "Name:" << pre\_e -> name << "\n"

<< "Score:" << pre\_e -> score << endl;

} else

cout << "不存在被查找学生" << endl;

system("pause");

break;

}

case 9: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

cout <<"输入成绩表中被查找学生的上面学生的学号" << endl;

ElemType cur\_e;

ElemType \*next\_e = (ElemType \*)malloc(sizeof(ElemType));//搜一下

cin >> cur\_e.id;

if (NextElem(\*L, cur\_e, next\_e)) {

cout << "存在你所查找的学生:\n"

<< "ID:" << next\_e -> id << "\n"

<< "Name:" << next\_e -> name << "\n"

<< "Score:" << next\_e -> score << endl;

} else

cout << "不存在被查找学生" << endl;

system("pause");

break;

}

case 10: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

cout << "输入你想插入的位置：" << endl;

int i;

cin >> i;

cout << "输入你想插入的学生的学生信息：" << endl;

ElemType s;

cin >> s.id >> s.name >> s.score;

if (ListInsert(&L, i, s))

cout << "插入成功!" << endl;

else

cout << "输入的位序不合法" << endl;

system("pause");

break;

}

case 11:{

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

cout << "请输入你想删除的学生的学号：" << endl;

ElemType s;

cin >> s.id;

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

if (ListDelete(&L, s, e)) {

cout << "你删除的学生的信息为："

<< "ID:" << e -> id << "\n"

<< "Name:" << e -> name << "\n"

<< "Score:" << e -> score << endl;

cout << "删除成功！" << endl;

}else{

cout << "目标学生不存在!" << endl;

}

system("pause");

break;

}

case 12: {

if (Isinit != 1) {

cout << "线性表未创建!" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

if (!ListTrabverse(\*L))

cout << "线性表是空表！" << endl;

system("pause");

break;

}

case 13: {

cout << "请输入你想读取的文件名" << endl;

ReadFromFile(&L,moro,&Isinit,hw,wh);//表已经被初始化

system("pause");

break;

}

case 14: {

if(moro == 0 && Isinit == 1){

cout << "是否保存(若不保存，则单表中的内容将被清空)当前的单表? 是：1 否：0" << endl;

int i;

cin >> i;

if (i) {

WriteToFile(&L);

} else

DestroyList(&L);

}

if(moro == 1){

cout << "当前已创建多表，无需再创建!" << endl;

system("pause");

break;

}

cout << "请输入同时操作表的数目" << endl;

int a;

cin >> hm;

if(hm > 0) {

moro = 1;

LinkLists = (LinkList)malloc( hm \* sizeof(LNode));

hw = (int \*)malloc( hm \* sizeof(int));

for (a = 0; a < hm; a++) {

hw[a] = 0;

}

wh = 1;

L = &(LinkLists[0]);

Isinit = hw[0];

cout << "多表创建成功!" << endl;

cout << "当前操作的表为表 1" << endl;

}else{

cout << "输入的数目不合法，将退出本功能！" << endl;

}

system("pause");

break;

}

case 15: {

if( hm == 0){

cout << "不存在多表！" << endl;

system("pause");

break;

}

//切换表的时候询问上一个表是否保存，若不保存，后面退出的时候可以选择保存

if(moro == 1)

Isinit = hw[wh - 1];

if(Isinit == 1){

cout << "是否要保存(若不保存，退出程序时可再次选择是否保存)上一个表?是：1 否：0" << endl;

int i;

cin >> i;

if (i) {//若文件已经被保存，则切换表的时候无法再次切换到该表

char filename[20];

cout << "输入文件名称（不超过20个字符）:" << endl;

cin >> filename;

strcat(filename,".txt");

ofstream outile;

outile.open(filename,std::ios::out);

if(outile){

if(moro == 1)

hw[wh -1] = 2;

ElemType \*p = L -> elem -> next;

while(p != nullptr) {

outile << p -> id << endl;

outile << p -> name << endl;

outile << p -> score << endl;

p = p -> next;

}

outile.close();

} else

cout << "创建文件失败!" << endl;

}

}

cout << "请输入被切换到表的序号(输入数字0退出切表功能)" << endl;

int a;

while(true){

cin >> a;

if( a == 0)

break;

if( hw[a - 1] == 3 && a <= hm && a > 0){

cout << "此表已被舍弃！无法切换到该表！请重新输入！" << endl;

continue;

}

if(a > 0 && a <= hm) {

wh = a;

Isinit = hw[wh - 1];

L = &(LinkLists[wh - 1]);

break;

}

cout << "输入的表序不存在，请在1--" << hm << "选择输入" << endl;

}

system("pause");

break;

}

case 16: {

if (Isinit != 1) {

cout << "线性表未创建！" << endl;

cout << "请重新选择你的操作!" << endl;

system("pause");

break;

}

int i;

cout << "是否要保存当前表?是：1 否：0" << endl;

cin >> i;

if (i) {

WriteToFile(&L);

}

system("pause");

break;

}

case 0: {

int i;

if(moro == 1)//让Isinit为多表中被操作表的状态

Isinit = hw[wh - 1];

if (Isinit != 1) {

if(Isinit == 2)

cout << "此表已保存！" << endl;

else if(Isinit == 0){

cout << "线性表未初始化！" << endl;

cout << "即将退出本功能!" << endl;

if(moro == 1)

hw[wh - 1] = 3;

}

}

else{

cout << "是否要保存当前表?是：1 否：0" << endl;

cin >> i;

if (i) {

if (moro == 1)

hw[wh - 1] = 2;

WriteToFile(&L);

} else{

//将表改变不保存的状态，即删除该表的存在

if (moro == 1)

hw[wh - 1] = 3;

DestroyList(&L);

}

}

//判断是否所有的表被保存，

//若没有，则切换到未保存的表。

if(moro == 1) {

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

if(hw[i] == 1 || hw[i] == 0){

Isinit = hw[i];

// L = &LinkLists[i];

L = &(LinkLists[i]);

wh = i + 1;

choice = 17;//只是为了再次进入循环，而不是具体的操作命令

cout << "当前操作的表为表" << wh << endl;

break;

}

}

if(choice == 0){

cout << "剩余的表已保存或被舍弃，即将退出程序！" << endl;

}

}

system("pause");

break;

}

default: {

cout << "指令不存在，请重新输入！" << endl;

break;

}

}//end of switch

}//end of while

cout << "欢迎下次再使用本系统!" << endl;

return 1;

}//end of main()

status IntiaList(LinkList \*L) {

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

(\*L) -> elem -> next = nullptr;

(\*L) -> length = 0;

return OK;

}

status DestroyList(LinkList \*L) {

ElemType \*p = (\*L) -> elem;

ElemType \*q;

while(p != nullptr){

q = p -> next;

free(p);

p = q;

}

return TRUE;

}

status ClearList(LinkList \*L) {

(\*L) -> length = 0;

return TRUE;

}

status ListEmpty(LNode L) {

if (L.length == 0 || L.elem ->next == nullptr)

return TRUE;

else

return FALSE;

}

int ListLength(LNode L) {

return L.length;

}

status GetElem(LNode L, char \*stu, ElemType \*e) {

ElemType \*p;

p = L.elem;

while(p -> next != nullptr) {

if (strcmp(stu,p -> id) == 0) {

\*e = \*p;

return OK;

}

p = p -> next;

}

return FALSE;

}

status LocateElem(LNode L, ElemType e,int &a) {

int i;

ElemType \*p = L.elem;

for (i = 0;p -> next != nullptr; i++) {

if (strcmp(p -> id,e.id) == 0) {

a = i;

return TRUE;

}

p = p -> next;

}

return FALSE;

}

status PriorElem(LNode L, ElemType cur\_e, ElemType \*pre\_e) {

int i;

ElemType \*p = L.elem -> next;

ElemType \*q;

for (i = 0; i < L.length && p != nullptr; i++) {

if (strcmp(p -> id,cur\_e.id) == 0 && i != 0) {

\*pre\_e = \*q;

return TRUE;

}

q = p;

p = p -> next;

}

return FALSE;

}

status NextElem(LNode L, ElemType cur\_e, ElemType \*next\_e) {

int i;

ElemType \*p = L.elem -> next;

cout << L.length << endl;

for (i = 0; i < L.length && p ->next != nullptr; i++) {

if (strcmp(p -> id,cur\_e.id) == 0 && i != (L.length - 1)) {

\*next\_e = \*(p -> next);

return TRUE;

}

p = p -> next;

}

return ERROR;

}

status ListInsert(LinkList \*L, int i, ElemType e) {

int a = 0;

if (i < 1 || i > (\*L) -> length + 1)

return ERROR;

ElemType \*p = (\*L) -> elem;

ElemType \*q = (\*L) -> elem;

ElemType \*h = (ElemType\*)malloc(sizeof(ElemType));

\*h = e;

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

q = q -> next;//第i个位置的元素

for(a = 0;a < i - 1;a++)

p = p -> next;//第(i-1)个位置的元素

p -> next = h;

h -> next = q;

++((\*L) -> length);

return OK;

}

status ListDelete(LinkList \*L, ElemType st, ElemType \*e) {

ElemType \*m = (\*L) -> elem;//所要删除的元素

ElemType \*n = nullptr;

while(m -> next != nullptr) {

if (strcmp(m -> id,st.id) == 0) {

break;

}

n = m;

m = m -> next;

}

if(m -> next == nullptr)

return FALSE;

\*e = \*m;

n -> next = m -> next;

--((\*L) -> length);

return OK;

}

status ListTrabverse(LNode L) {

if (!L.elem || L.length == 0)

return ERROR;

ElemType \*p = L.elem -> next;

while (p != nullptr) {

cout << "ID:"<< p -> id << "\n"

<< "Name:" << p -> name << "\n"

<< "Score:" << p -> score << "\n" << endl;

p = p -> next;

}

return TRUE;

}

status ReadFromFile(LinkList \*L, int moro, int \*Isinit, int \*hw, int wh) {

char filename[20];

cin >> filename;

strcat(filename, ".txt");

ifstream infile;

infile.open(filename, std::ios::in);

if (infile) {

//根据是多表操作还是单表操作来改变它们的状态，

//从而在前面判断是否被初始化

if(moro == 1)

\*Isinit = hw[wh - 1];

if(\*Isinit == 1){

DestroyList(L);

\*Isinit = 0;

}

if(\*Isinit == 0 && moro == 1) {

IntiaList(L);

\*Isinit = hw[wh - 1] = 1;

} else if(\*Isinit == 0 && moro == 0){

IntiaList(L);

\*Isinit = 1;

}

int a;

ElemType \*q;

ElemType \*p = (\*L) -> elem -> next = (ElemType \*)malloc(sizeof(ElemType));

for (a = 0;; a++) {

infile >> p -> id >> p -> name >> p -> score;

if(infile.peek() == EOF) {

free(p);

q -> next = nullptr;

break;

}

++((\*L) -> length);

p -> next = (ElemType \*)malloc(sizeof(ElemType));

q = p;

p = p -> next;

}

infile.close();

cout << "读取成功!" << endl;

return OK;

} else {

cout << "不存在此文件!" << endl;

return ERROR;

}

}

status WriteToFile(LinkList \*L) {

char filename[20];

cout << "输入文件名称（不超过20个字符）:" << endl;

cin.sync();

cin >> filename;

cin.sync();

strcat(filename,".txt");

ofstream outfile;

outfile.open(filename,std::ios::out);

if(outfile){

ElemType \*p = (\*L) -> elem -> next;

while(p != nullptr) {

outfile << p -> id << endl;

outfile << p -> name << endl;

outfile << p -> score << endl;

p = p -> next;

}

outfile.close();

cout << "创建文件成功!" << endl;

return OK;

} else

cout << "创建文件失败!" << endl;

return FALSE;

}

# 附录C

/\*\*

\* 本实验每个树都有一个首结点

\* 且首节点的左孩子和右孩子均指向该树的根结点

\* 函数中传入的值一般是根结点的左孩子(或右孩子)，即根结点

\*/

/\*\*

\* 当树未创建时，不能销毁树

\* 当树创建时，销毁树

\* 1.单树时，单树成为未初始化状态

\* 2.多树时，将舍弃多树中的此树

\* 未创建和已创建的条件下均可以通过exit退出当前树

\*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <iostream>

#include <cstring>

#include <fstream>

#include <cmath>

using namespace std;

#define BEF 0 //先序序列输入

#define MID 1 //中序序列输入

#define AFT 2 //后序序列输入

#define FALSE 0

#define OK 1

#define ERROR 0

#define EVERY 20

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 10

#define MAXQSIZE 100

#define NOTINIT 0

#define INIT 1

#define CREATE 2

#define ABANDON 3

typedef int Status;

typedef struct{

char id[EVERY];

char name[EVERY];

char phone[EVERY];

} ElemType;

typedef struct BiTNode{

ElemType \*elem;

struct BiTNode \* lchild;

struct BiTNode \* rchild;

} BiTNode, \* BiTree;

typedef struct RooTNode{

BiTree child;

}RooTNode,\* RooTree;

typedef struct {

BiTree \*base;

BiTree \*top;

int stacksize;

} BiStack;

typedef struct {

BiTree \* base;

int front;

int rear;

}LinkQueue;

Status InitStack(BiStack &S);

Status Push(BiStack &S,BiTree t);

Status Pop(BiStack &S,BiTree &t);

Status InitQueue(LinkQueue &Q);

Status DestroyQueue(LinkQueue &Q);

Status EnQueue(LinkQueue &Q,BiTree N);

Status DeQueue(LinkQueue &Q,BiTree &N);

Status InitBiTree(RooTree \*T);

Status DestroyBiTree(RooTree \*\*T,int moro,int \*where,int \*Isinit,int \*\*how,RooTree \*\*TS,int \*choice,int number);

Status CreateBiTree(BiTree \*T,int definition);

Status ClearBiTree (BiTree \*T);

Status BiTreeEmpty(BiTree T);

int BiTreeDepth(BiTree T);

Status Root(BiTree T);

Status Value(BiTree T,BiTNode e);

Status Assign(BiTree \*T,BiTNode &e,ElemType value);

Status Parent(BiTree T,BiTree e);

Status LeftChild(BiTree T,BiTree e);

Status RightChild(BiTree T,BiTree e);

Status LeftSibling(BiTree T,BiTree e);

Status RightSibling(BiTree T,BiTree e);

Status InsertChildBiTNode (BiTree \*T,BiTree p,int LR);

Status DeleteChild(BiTree \*T,BiTree p,int LR);

Status PreOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e));

Status InOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e));

Status PostOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e));

Status LevelOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e));

Status Write(BiTree T,ofstream &outfile);

Status WriteToFile(BiTree T);

Status Read(BiTree \*T,ifstream &infile);

Status ReadFromFile(BiTree \*T);

Status Change(int \*Inisinit,int moro,int where,int \* \*how, int state);

Status PrintAsTree(BiTree T,Status (\* Visit)(ElemType \*e));

Status PrintElem(ElemType \*e);

Status VisitElem(ElemType \*e);

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

int main(void) {

int moro = 0;//判断是多树还是单树

int choice = 1;//判断用户的选择

int where = 0;//确定多树的位置

int \*how;//用来保存每个树的状态 0:未初始化 1：初始化 2：已创建 3：已舍弃 4：已保存

int Isinit = 0;//用来判断树是否初始化

int number = 0;

int definition;

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

RooTree \* TS = nullptr;//顺序表来存储多树的指针

while (choice) {

cout << "\n\n" << endl;

cout << " Menu for Linear Table On Sequence Structure \n" << endl;

cout << "-------------------------------------------------\n" << endl;

cout << " 1. InitBiTree 13. LeftSibling\n" << endl;

cout << " 2. DestroyBiTree 14. RightSibling\n" << endl;

cout << " 3. CreateBiTree 15. InsertChildBiTNode\n" << endl;

cout << " 4. ClearBiTree 16. DeleteChild\n" << endl;

cout << " 5. BiTreeEmpty 17. PreOrderTraverse\n" << endl;

cout << " 6. BiTreeDepth 18. InOrderTraverse\n" << endl;

cout << " 7. Root 19. PostOrderTraverse\n" << endl;

cout << " 8. Value 20. LevelOrderTraverse\n" << endl;

cout << " 9. Assign 21. WriteToFile\n" << endl;

cout << " 10. Parent 22. ReadFromFile \n" << endl;

cout << " 11. LeftChild 23. MultiTrees\n" << endl;

cout << " 12. RightChild 24. SwitchTree\n" << endl;

cout << " 0. Exit 25. PrintAsTree\n" << endl;

cout << "-------------------------------------------------\n" << endl;

if (moro == 1)

cout << "当前操作的为树 " << where << endl;

cout << " 请选择你的操作[0~25]:" << endl;

cin >> choice;

switch (choice) {

case 1:{

if(Isinit == 1){

cout << "当前已初始化树!" << endl;

system("pause");

break;

}else if(Isinit == 2){

cout << "当前树已创建!无法再次初始化，若要初始化树，请先销毁树!" << endl;

system("pause");

break;

}

if(InitBiTree(T)) {

cout << "初始化树成功!" << endl;

Change(&Isinit,moro,where,& how,INIT);

}

else

cout << "初始化树失败!" << endl;

Isinit = 1;

system("pause");

break;

}

case 2:{

if(Isinit == 0){

cout << "树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}

else if (Isinit == 1){

cout << "树未创建，无法清空树，请重新选择你的操作!" << endl;

system("pause");

break;

}

else{

DestroyBiTree(&T,moro,&where,&Isinit,&how,&TS,&choice,number);

system("pause");

break;

}

}

case 3:{

if(Isinit == 0){

cout << "树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if (Isinit == 2){

cout << "当前树已创建，无法再次创建树!" << endl;

system("pause");

break;

}else if(Isinit == 3 && moro == 1){

cout << "当前树已被舍弃，无法再次创建!" << endl;

system("pause");

break;

}

cout << "先序输入:0 中序输入:1 后序输入:2" << endl;

cout << "请选择你的操作!" << endl;

while(cin >> definition){

if(definition == 0 || definition == 1 || definition == 2)

break;

else{

cout << "输入的值不合法，请重新输入!" << endl;

continue;

}

}

cout << "学号输入#代表当前结点为空结点" << endl;

CreateBiTree(&(\*T) -> child,definition);

if((\*T) -> child != nullptr) {

Change(&Isinit, moro, where, &how, CREATE);

cout << "创建树成功!" << endl;

} else{

cout << "根结点为空，故树未创建!" << endl;

}

system("pause");

break;

}

case 4:{

if(Isinit == 0){

cout << "树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 1){

cout << "当前树已初始化，但未被创建，因此无法清空树!" << endl;

system("pause");

break;

}

else if(Isinit == 3 && moro == 1){

cout << "当前树已被舍弃，无法再次创建!" << endl;

system("pause");

break;

}

else if(Isinit == 2){

ClearBiTree(&(\*T) -> child);

Change(&Isinit,moro,where,& how,INIT);

cout << "清空树成功!" << endl;

}

system("pause");

break;

}

case 5:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

BiTreeEmpty((\*T) -> child);

system("pause");

break;

}

case 6:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

int deepth = BiTreeDepth((\*T) -> child);

cout << "树的高度为" << deepth << endl;

system("pause");

break;

}

case 7:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

Root((\*T) -> child);

system("pause");

break;

}

case 8:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

BiTNode e;

e.elem = (ElemType \*)malloc(sizeof(ElemType));

cout << "请输入你要查找的学生学号:" << endl;

cin >> e.elem -> id;

Value((\*T) -> child,e);

free(e.elem);

system("pause");

break;

}

case 9:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

BiTNode e;

ElemType value;

e.elem = (ElemType \*)malloc(sizeof(ElemType));

cout << "请输入目标学生的学号" << endl;

cin >> e.elem -> id;

cout << "请依次输入改变后的学号，姓名以及手机号码" << endl;

cin >> value.id >> value.name >> value.phone;

if(Assign(&(\*T) -> child,e,value)){

cout << "操作成功!" << endl;

} else{

cout << "操作失败!" << endl;

}

free(e.elem);

system("pause");

break;

}

case 10:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

BiTree e;

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

e -> elem = (ElemType \*)malloc(sizeof(ElemType));

cout << "请输入要查询结点的子结点的学号" << endl;

cin >> e -> elem -> id;

Parent((\*T) -> child,e);

free(e -> elem);

free(e);

system("pause");

break;

}

case 11:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

BiTree e;

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

e -> elem = (ElemType \*)malloc(sizeof(ElemType));

cout << "请输入父亲结点的学号:" << endl;

cin >> e -> elem -> id;

LeftChild((\*T) -> child,e);

free(e -> elem);

free(e);

system("pause");

break;

}

case 12:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

BiTree e;

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

e -> elem = (ElemType \*)malloc(sizeof(ElemType));

cout << "请输入父亲结点的学号:" << endl;

cin >> e -> elem -> id;

RightChild((\*T) -> child,e);

free(e -> elem);

free(e);

system("pause");

break;

}

case 13:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

BiTree e;

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

e -> elem = (ElemType \*)malloc(sizeof(ElemType));

cout << "请输入目标结点的学号:" << endl;

cin >> e -> elem -> id;

LeftSibling((\*T) -> child,e);

free(e -> elem);

free(e);

system("pause");

break;

}

case 14:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

BiTree e;

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

e -> elem = (ElemType \*)malloc(sizeof(ElemType));

cout << "请输入目标结点的学号:" << endl;

cin >> e -> elem -> id;

RightSibling((\*T) -> child,e);

free(e -> elem);

free(e);

system("pause");

break;

}

case 15:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

cout << "请输入插入结点的学号:" << endl;

BiTree p;

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

p -> elem = (ElemType\*)malloc(sizeof(ElemType));

cin >> p -> elem -> id;

cout << "请选择你插入的结点的孩子类型: 左孩子:0 右孩子:1" << endl;

int LR;

while(cin >> LR){

if(LR == 0 || LR == 1)

break;

else

cout << "输入的值不合法，请重新选择孩子类型: 左孩子:0 右孩子:1" << endl;

}

if(InsertChildBiTNode(&(\*T) -> child,p,LR))

cout << "插入成功!" << endl;

else

cout << "插入失败!" << endl;

system("pause");

break;

}

case 16:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

cout << "请输入要删除结点的父结点的学号:" << endl;

BiTree p;

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

p -> elem = (ElemType\*)malloc(sizeof(ElemType));

cin >> p -> elem -> id;

cout << "请选择你要删除的孩子类型: 左孩子:0 右孩子:1" << endl;

int LR;

while(cin >> LR){

if(LR == 0 || LR == 1)

break;

else

cout << "输入的值不合法，请重新选择孩子类型: 左孩子:0 右孩子:1" << endl;

}

DeleteChild(&(\*T) -> child,p,LR);

system("pause");

break;

}

case 17:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

PreOrderTraverse((\*T) -> child,VisitElem);

system("pause");

break;

}

case 18:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

InOrderTraverse((\*T) -> child,VisitElem);

system("pause");

break;

}

case 19:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

PostOrderTraverse((\*T) -> child,VisitElem);

system("pause");

break;

}

case 20:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

LevelOrderTraverse((\*T) -> child,VisitElem);

system("pause");

break;

}

case 21:{

if(Isinit == 0){

cout << "当前树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}else if(Isinit == 1){

cout << "当前树未创建，请重新选择你的操作!" << endl;

system("pause");

break;

} else if(Isinit == 3){

cout << "当前树已被舍弃，请重新选择你的操作!" << endl;

system("pause");

break;

}

if(WriteToFile((\*T) -> child))

cout << "存储文件成功!" << endl;

system("pause");

break;

}

case 22:{

if(Isinit == 0){

cout << "树未初始化，请重新选择你的操作!" << endl;

system("pause");

break;

}

if(Isinit == 2){

if(ReadFromFile(&(\*T) -> child)) {

cout << "读取文件成功" << endl;

Change(&Isinit, moro, where, &how, CREATE);

}

}else{

if(ReadFromFile(&(\*T) -> child)) {

cout << "读取文件成功" << endl;

Change(&Isinit, moro, where, &how, CREATE);

}

}

system("pause");

break;

}

case 23:{

if(moro == 1){

cout << "当前已创建多树。无需再次创建!" << endl;

system("pause");

break;

}else if(moro == 0 && Isinit == 2){

cout << "是否保存当前单树，若不保存，则视为丢弃!" << endl;

cout << "是: 1 否: 0" << endl;

int a;

cin >> a;

if(a == 1)

WriteToFile((\*T) -> child);

}

cout << "请输入你要创建的多树数目:" << endl;

cin >> number;

if(number > 0) {

TS = (RooTree \*) malloc(number \* sizeof(RooTree));

how = (int \*) malloc(number \* sizeof(int));

for (int i = 0; i < number; i++) {

how[i] = 0;

}

moro = 1;

where = 1;

T = &TS[where - 1];

Change(&Isinit, moro, where, &how, NOTINIT);

cout << "多树创建成功!" << endl;

cout << "当前操作的树为树 1" << endl;

} else{

cout << "输入的值必须大于0，请重新选择你的操作!" << endl;

}

system("pause");

break;

}

case 24:{

if( moro == 0){

cout << "不存在多树！" << endl;

system("pause");

break;

}

//切换树的时候询问上一个树是否保存，若不保存，后面退出的时候可以选择保存

if(moro == 1)

Isinit = how[where - 1];

if(Isinit == 2){

cout << "是否要保存(若不保存，退出程序时可再次选择是否保存)上一个树?是：1 否：0" << endl;

int i;

cin >> i;

if (i) {

WriteToFile((\*T) -> child);

}

}

cout << "请输入被切换到树的序号(输入数字0退出切树功能)" << endl;

int a;

while(true){

cin >> a;

if( a == 0)

break;

if( how[a - 1] == 3 && a <= number && a > 0){

cout << "此树已被舍弃！无法切换到该树！请重新输入！" << endl;

continue;

}

if(a > 0 && a <= number) {

where = a ;

Isinit = how[where - 1];

T = &TS[where - 1];

break;

}

cout << "输入的树序不存在，请在1--" << number << "选择输入" << endl;

}

system("pause");

break;

}

case 25:{

cout << "树形如下: " << '\n' << endl;

PrintAsTree((\*T) -> child,VisitElem);

break;

}

case 0:{

int i;

if(moro == 1)//让Isinit为多表中被操作表的状态

Isinit = how[where - 1];

if (Isinit != 2 && moro == 0) {

cout << "此树未被创建或已被舍弃！" << endl;

cout << "即将退出本程序!" << endl;

}

else{

DestroyBiTree(&T,moro,&where,&Isinit,&how,&TS,&choice,number);

}

//判断是否所有的树被保存，

//若没有，则切换到未保存的树。

if(moro == 1) {

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

if(how[i] != 3){

Isinit = how[i];

T = &TS[i];

where = i + 1;

choice = 100;//只是为了再次进入循环，而不是具体的操作命令

cout << "当前操作的树为树" << where << endl;

break;

}

}

}

if(moro == 1 && how[where] == 3)

cout << "即将退出程序!" << endl;

system("pause");

break;

}

default :{

cout << "指令不存在，请重新输入!" << endl;

system("pause");

break;

}

}

}

return 0;

}

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

Status InitStack(BiStack &S){

//构造一个空栈

S.base = (BiTree \*)malloc(STACK\_INIT\_SIZE \* sizeof(BiTree));

if(!S.base)

exit(OVERFLOW);

S.top = S.base;

S.stacksize = STACK\_INIT\_SIZE;

return OK;

}

Status Push(BiStack &S,BiTree t){

//插入e元素为新的栈顶元素

if(S.top - S.base >= S.stacksize){

S.base = (BiTree \*)realloc(S.base,(S.stacksize + STACKINCREMENT) \*sizeof(BiTree));

if(!S.base)

exit(OVERFLOW);

S.top = S.base + S.stacksize;

S.stacksize += STACKINCREMENT;

}

\* (S.top) = t;

S.top ++;

return OK;

}

Status Pop(BiStack &S,BiTree &T){

//若栈不为空，则删除S的栈顶元素，并用e返回其值，并且返回OK；否则返回ERROR

if(S.top == S.base)

return ERROR;

T = \*(--S.top);

return OK;

}

Status InitQueue(LinkQueue &Q){

Q.base = (BiTree\*)malloc(MAXQSIZE \* sizeof(BiTree));

if(!Q.base)

exit(OVERFLOW);

Q.front = Q.rear = 0;

return OK;

}

Status DestroyQueue(LinkQueue &Q){

free(Q.base);

return OK;

}

Status EnQueue(LinkQueue &Q,BiTree N){

if((Q.rear + 1) % MAXQSIZE == Q.front)

return ERROR;

Q.base[Q.rear] = N;

Q.rear = (Q.rear + 1) % MAXQSIZE;

return OK;

}

Status DeQueue(LinkQueue &Q, BiTree &N){

if(Q.front == Q.rear)

return ERROR;

N = Q.base[Q.front];

Q.front = (Q.front + 1 ) % MAXQSIZE;

return OK;

}

Status InitBiTree(RooTree \*T){

(\*T) = (RooTree)malloc(sizeof(RooTNode));

(\*T) -> child = nullptr;

if(!(\*T))

return OVERFLOW;

else{

return OK;

}

}

Status DestroyBiTree(RooTree \*\*T,int moro,int \*where,int \*Isinit,int \*\*how,RooTree \*\*TS,int \*choice,int number){

if(\*Isinit != 0 && (\*\*T) -> child){

int i;

if(moro == 1)//让Isinit为多表中被操作表的状态

\*Isinit = (\*how)[\*where - 1];

if(moro == 0){

cout << "是否保存当前树?是：1 否：0" << endl;

cin >> i;

if(i) {

WriteToFile((\*\*T)->child);

cout << "保存文件成功!" << endl;

}

ClearBiTree(&(\*\*T) -> child );

Change(Isinit,moro,\*where,how,NOTINIT);

if(\*choice == 0)

cout << "即将退出本程序!" << endl;

return OK;

}

else if(moro == 1 && \*Isinit == 2){

cout << "是否要保存当前树?是：1 否：0" << endl;

cin >> i;

if (i) {

WriteToFile((\*\*T) -> child);

cout << "保存文件成功!" << endl;

(\*how)[\*where - 1] = 3;

ClearBiTree(&(\*\*T)->child);

Change(Isinit, moro, \*where, how, ABANDON);

cout << "销毁树成功!" << endl;

} else {

//将多树中改变不保存的状态，即删除该树的存在；单树视为清空

ClearBiTree(&(\*\*T)->child);

Change(Isinit, moro, \*where, how, ABANDON);

cout << "销毁树成功!" << endl;

}

}

//判断是否所有的树被保存，

//若没有，则切换到未保存的树。

if(moro == 1) {

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

if((\*how)[i] != 3){

\*Isinit = (\*how)[i];

\*T = &((\*TS)[i]);

\*where = i + 1;

\*choice = 100;//只是为了再次进入循环，而不是具体的操作命令

cout << "当前操作的树为树" << \*where << endl;

break;

}

}

if(i == number) {

cout << "剩余的树已保存或被舍弃，即将退出程序！" << endl;

\*choice = 0;

}

}

return OK;

}else {

if(\*choice != 0)

cout << "树未创建，无法销毁!" << endl;

else if(moro == 1)

cout << "删除当前树成功!" << endl;

Change(Isinit,moro,\*where,how,ABANDON);

return FALSE;

}

}

Status CreateBiTree(BiTree \*T,int definition){

//按照先序次序输入二叉树结点的值

if(definition == BEF){

ElemType e;

cout << "请输入学生的学号" << endl;

cin >> e.id;

if(!strcmp(e.id,"#")) {

cout << "当前结点为空结点，请输入下一个学生的学号" << endl;

(\*T) = nullptr;

}

else{

cout << "请输入学生的姓名以及联系方式" << endl;

cin >> e.name >> e.phone;

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

exit(OVERFLOW);

(\*T) -> elem = nullptr;

if (!((\*T)->elem = (ElemType \*) malloc(sizeof(ElemType))))

exit(OVERFLOW);

(\*T) -> lchild = nullptr;

(\*T) -> rchild = nullptr;

\*((\*T) -> elem) = e;

CreateBiTree(&(\*T) -> lchild,definition);

CreateBiTree(&(\*T) -> rchild,definition);

}

return OK;

}

else if(definition == MID){

ElemType e;

cin >> e.id;

if(!strcmp(e.id,"#"))

(\*T) = nullptr;

else{

cin >> e.name >> e.phone;

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

exit(OVERFLOW);

(\*T) -> elem = nullptr;

if (!((\*T)->elem = (ElemType \*) malloc(sizeof(ElemType))))

exit(OVERFLOW);

(\*T) -> lchild = nullptr;

(\*T) -> rchild = nullptr;

CreateBiTree(&(\*T) -> lchild,definition);

\*((\*T) -> elem) = e;

CreateBiTree(&(\*T) -> rchild,definition);

}

return OK;

}

else if(definition == AFT){

ElemType e;

cin >> e.id;

if(!strcmp(e.id,"#"))

(\*T) = nullptr;

else{

cin >> e.name >> e.phone;

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

exit(OVERFLOW);

(\*T) -> elem = nullptr;

if (!((\*T)->elem = (ElemType \*) malloc(sizeof(ElemType))))

exit(OVERFLOW);

(\*T) -> lchild = nullptr;

(\*T) -> rchild = nullptr;

CreateBiTree(&(\*T) -> lchild,definition);

CreateBiTree(&(\*T) -> rchild,definition);

\*((\*T) -> elem) = e;

}

return OK;

}

return OK;

}

//此函数在case实例中在后面不会删除首结点

Status ClearBiTree (BiTree \*T){

if(\*T){

ClearBiTree(&(\*T) -> lchild);

ClearBiTree(&(\*T) -> rchild);

//释放树结点中的所有变量

free((\*T) -> elem);

free((\*T));

(\*T) = nullptr;

}else

return OK;

}

Status BiTreeEmpty(BiTree T){

if(!T || !(T -> elem))

cout << "树为空!" << endl;

else

cout << "树非空!" << endl;

return OK;

}

int BiTreeDepth(BiTree T){

int hl,hr,max;

if(T){

hl = BiTreeDepth(T -> lchild);

hr = BiTreeDepth(T -> rchild);

max = (hl > hr) ? hl : hr;

return (max + 1);

}

else

return 0;

}

Status Root(BiTree T){

if(T){

cout << "树根节点的信息为:" << endl;

cout << "name:" << T -> elem -> id << endl;

cout << "age:" << T -> elem -> name << endl;

cout << "phone:" << T -> elem -> phone << endl;

return OK;

}

else

cout << "树为空!" << endl;

return FALSE;

}

Status Value(BiTree T,BiTNode e){

BiStack S;

InitStack(S);

while (T || S.top != S.base){

while (T){

Push(S,T);

T = T -> lchild;

}

if(S.base != S.top){

Pop(S,T);

if(!strcmp(e.elem -> id, T -> elem -> id)){

cout << "目标学生的信息为:" << endl;

cout << "id: " << T -> elem -> id << endl;

cout << "name: " << T -> elem -> name << endl;

cout << "phone: " << T -> elem -> phone << endl;

free(S.base);

return OK;

}

T = T -> rchild;

}

}

cout << "目标学生不存在!" << endl;

free(S.base);

return FALSE;

}

Status Assign(BiTree \*T,BiTNode &e,ElemType value){

BiStack S;

InitStack(S);

while ((\*T) || S.top != S.base){

while ((\*T)){

Push(S,(\*T));

(\*T) = (\*T) -> lchild;

}

if(S.base != S.top){

Pop(S,(\*T));

if(!strcmp(e.elem -> id,(\*T) -> elem -> id)){

cout << "目标学生的信息为:" << endl;

cout << "name: " << (\*T) -> elem -> id << endl;

cout << "name: " << (\*T) -> elem -> name << endl;

cout << "name: " << (\*T) -> elem -> phone << endl;

\*((\*T) -> elem) = value;

cout << "修改后的学生信息为:" << endl;

cout << "name: " << (\*T) -> elem -> id << endl;

cout << "name: " << (\*T) -> elem -> name << endl;

cout << "name: " << (\*T) -> elem -> phone << endl;

free(S.base);

return OK;

}

(\*T) = (\*T) -> rchild;

}

}

cout << "目标学生不存在!" << endl;

free(S.base);

return FALSE;

}

Status Parent(BiTree T,BiTree e){

if(!strcmp(T -> elem -> id,e ->elem -> id)){

cout << "目标结点为根结点，不存在父结点，请重新选择你的操作!" << endl;

return FALSE;

}

BiStack S;

InitStack(S);

while (T || S.top != S.base){

while (T){

if(T -> lchild && !strcmp(T -> lchild -> elem -> id,e -> elem -> id)){

cout << "目标结点的双亲的信息为:" << endl;

cout << "name: " << T -> elem -> id << endl;

cout << "name: " << T -> elem -> name << endl;

cout << "name: " << T -> elem -> phone << endl;

free(S.base);

return OK;

}

Push(S,T);

T = T -> lchild;

}

if(S.base != S.top){

Pop(S,T);

if(T -> rchild && !strcmp(T -> rchild -> elem -> id,e -> elem -> id)){

cout << "目标结点的双亲的信息为:" << endl;

cout << "name: " << T -> elem -> id << endl;

cout << "name: " << T -> elem -> name << endl;

cout << "name: " << T -> elem -> phone << endl;

free(S.base);

return OK;

}

T = T -> rchild;

}

}

cout << "目标学生不存在!故其双亲不存在!" << endl;

free(S.base);

return FALSE;

}

Status LeftChild(BiTree T,BiTree e){

BiStack S;

InitStack(S);

while (T || S.top != S.base){

while (T){

if(!strcmp(T -> elem -> id,e -> elem -> id)){

if(T -> lchild){

cout << "目标结点的左孩子信息为:" << endl;

cout << "name: " << T -> lchild -> elem -> id << endl;

cout << "age: " << T -> lchild -> elem -> name << endl;

cout << "phone: " << T -> lchild -> elem -> phone << endl;

free(S.base);

return OK;

}else{

cout << "目标结点不存在左孩子!" << endl;

free(S.base);

return FALSE;

}

}

Push(S,T);

T = T -> lchild;

}

if(S.base != S.top){

Pop(S,T);

T = T -> rchild;

}

}

free(S.base);

cout << "目标结点不存在，请重新选择你的操作!" << endl;

return FALSE;

}

Status RightChild(BiTree T,BiTree e){

BiStack S;

InitStack(S);

while (T || S.top != S.base){

while (T){

Push(S,T);

T = T -> lchild;

}

if(S.base != S.top){

Pop(S,T);

if(!strcmp(T -> elem -> id,e -> elem -> id)){

if(T -> rchild){

cout << "目标结点的右孩子信息为:" << endl;

cout << "name: " << T -> rchild -> elem -> id << endl;

cout << "age: " << T -> rchild -> elem -> name << endl;

cout << "phone: " << T -> rchild -> elem -> phone << endl;

free(S.base);

return OK;

}else{

cout << "目标结点不存在右孩子!" << endl;

free(S.base);

return FALSE;

}

}

T = T -> rchild;

}

}

free(S.base);

cout << "目标结点不存在，请重新选择你的操作!" << endl;

return FALSE;

}

Status LeftSibling(BiTree T,BiTree e){

if(!strcmp(T -> elem -> id,e ->elem -> id)){

cout << "目标结点为根结点，不存在左兄弟，请重新选择你的操作!" << endl;

return FALSE;

}

BiStack S;

InitStack(S);

while (T || S.top != S.base){

while (T){

Push(S,T);

T = T -> lchild;

}

if(S.base != S.top){

Pop(S,T);

if(T -> rchild && !strcmp(T -> rchild -> elem -> id,e -> elem -> id)){

if(T -> lchild){

cout << "目标结点的左兄弟信息为:" << endl;

cout << "name: " << T -> lchild -> elem -> id << endl;

cout << "age: " << T -> lchild -> elem -> name << endl;

cout << "phone: " << T -> lchild -> elem -> phone << endl;

free(S.base);

return OK;

}

else{

cout << "目标结点不存在左兄弟" << endl;

free(S.base);

return FALSE;

}

}

else if(T -> lchild && !strcmp(T -> lchild -> elem -> id,e -> elem -> id)){

free(S.base);

cout << "目标结点不存在左兄弟!" << endl;

return FALSE;

}

T = T -> rchild;

}

}

free(S.base);

cout << "目标结点不存在!" << endl;

return FALSE;

}

Status RightSibling(BiTree T,BiTree e){

if(!strcmp(T -> elem -> id,e ->elem -> id)){

cout << "目标结点为根结点，不存在右兄弟，请重新选择你的操作!" << endl;

return FALSE;

}

BiStack S;

InitStack(S);

while (T || S.top != S.base){

while (T){

Push(S,T);

if(T -> lchild && !strcmp(T -> lchild -> elem -> id,e -> elem -> id)){

if(T -> rchild){

cout << "目标结点的右兄弟信息为:" << endl;

cout << "name: " << T -> rchild -> elem -> id << endl;

cout << "age: " << T -> rchild -> elem -> name << endl;

cout << "phone: " << T -> rchild -> elem -> phone << endl;

free(S.base);

return OK;

}

else{

cout << "目标结点不存在右兄弟" << endl;

free(S.base);

return FALSE;

}

}

else if(T -> rchild && !strcmp(T -> rchild -> elem -> id,e -> elem -> id)){

free(S.base);

cout << "目标结点不存在右兄弟!" << endl;

return FALSE;

}

T = T -> lchild;

}

if(S.base != S.top){

Pop(S,T);

T = T -> rchild;

}

}

cout << "目标结点不存在!" << endl;

free(S.base);

return FALSE;

}

Status InsertChildBiTNode (BiTree \*T,BiTree p,int LR){

BiStack S;

InitStack(S);

BiTree G = \*T;

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

int definition;

cout << "先序输入:0 中序输入:1 后序输入:2" << endl;

cout << "请选择你的操作!" << endl;

while(cin >> definition){

if(definition == 0 || definition == 1 || definition == 2)

break;

else{

cout << "输入的值不合法，请重新输入!" << endl;

continue;

}

}

cout << "学号输入#代表当前结点为空结点" << endl;

cout << "若输入的树右子树存在，则视为舍去右子树!" << endl;

CreateBiTree(&H,definition);

while ((\*T) || S.top != S.base){

while ((\*T)){

Push(S,(\*T));

(\*T) = (\*T) -> lchild;

}

if(S.base != S.top){

Pop(S,(\*T));

if(!strcmp((\*T) -> elem -> id,p -> elem -> id)){

p = (\*T);

if(LR == 0){

H -> rchild = p -> lchild;

(\*T) -> lchild = H;

}else{

H -> rchild = p -> rchild;

(\*T) -> rchild = H;

}

\*T = G;

return OK;

}

(\*T) = (\*T) -> rchild;

}

}

\*T = G;

cout << "不存在目标结点!" << endl;

return FALSE;

}

Status DeleteChild(BiTree \*T,BiTree p,int LR){

BiTree H = (\*T);

BiStack S;

InitStack(S);

while (LR != 0 && LR != 1) {

cout << "选择左右子树指令不合法，请确定你要删除的左右子树(左:0 右:1)" << endl;

cin >> LR;

if(LR == 0 || LR == 1)

break;

}

while ((\*T) || S.top != S.base){

while ((\*T)){

Push(S,(\*T));

(\*T) = (\*T) -> lchild;

}

if(S.base != S.top){

Pop(S,(\*T));

if(!strcmp((\*T) -> elem -> id,p -> elem -> id)){

if(LR == 0){

if((\*T) -> lchild) {

cout << "删除结点成功!" << endl;

(\*T)->lchild = nullptr;

}

else

cout << "目标结点不存在左子树，无法删除!" << endl;

} else if(LR == 1){

if((\*T) -> rchild) {

cout << "删除结点成功!" << endl;

(\*T)->rchild = nullptr;

}

else

cout << "目标结点不存在右子树，无法删除!" << endl;

}

(\*T) = H;

return OK;

}

(\*T) = (\*T) -> rchild;

}

}

(\*T) = H;

return OK;

}

Status PreOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e)){

if(T) {

if (Visit(T->elem)) {

if (PreOrderTraverse(T->lchild,Visit))

if (PreOrderTraverse(T->rchild,Visit))

return OK;

}

return ERROR;

}

else

return OK;

}

Status InOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e)){

if(T) {

if (InOrderTraverse(T->lchild,Visit)) {

if (Visit(T -> elem)) {

if (InOrderTraverse(T->rchild,Visit))

return OK;

}

else

return ERROR;

}

}

else

return OK;

}

Status PostOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e)){

if(T) {

if (PostOrderTraverse(T->lchild,Visit))

if (PostOrderTraverse(T->rchild,Visit)){

if(!Visit(T -> elem))

exit(ERROR);

}

return OK;

}

else

return OK;

}

Status LevelOrderTraverse(BiTree T,Status(\* Visit)(ElemType \*e)){

if(!T){

cout << "树为空树!" << endl;

return OK;

}

LinkQueue Q;

InitQueue(Q);

EnQueue(Q,T);

while(Q.front != Q.rear) {

DeQueue(Q,T);

if(!Visit(T -> elem))

exit(ERROR);

if(T -> lchild)

EnQueue(Q,T -> lchild);

if(T -> rchild)

EnQueue(Q,T -> rchild);

}

DestroyQueue(Q);

return OK;

}

Status Write(BiTree T,ofstream &outfile){

if(T) {

if (outfile << T -> elem -> id << '\n'

<< T -> elem -> name << '\n'

<< T -> elem -> phone << endl)

{

if (Write(T->lchild,outfile))

if (Write(T->rchild,outfile))

return OK;

}

return ERROR;

}

else {

outfile << "#" << endl;

return OK;

}

}

Status WriteToFile(BiTree T){

char filename[20];

cout << "输入文件名称（不超过20个字符）:" << endl;

cin >> filename;

strcat(filename,".txt");

ofstream outfile;

outfile.open(filename,std::ios::out);

if(outfile){

Write(T,outfile);

outfile.close();

return OK;

} else

cout << "创建文件失败!" << endl;

return FALSE;

}

Status Read(BiTree \*T,ifstream &infile){

ElemType e;

infile >> e.id;

if(!strcmp(e.id,"#"))

(\*T) = nullptr;

else{

infile >> e.name >> e.phone;

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

exit(OVERFLOW);

(\*T) -> elem = nullptr;

if (!((\*T)->elem = (ElemType \*) malloc(sizeof(ElemType))))

exit(OVERFLOW);

(\*T) -> lchild = nullptr;

(\*T) -> rchild = nullptr;

\*((\*T) -> elem) = e;

Read(&(\*T) -> lchild,infile);

Read(&(\*T) -> rchild,infile);

}

return OK;

}

Status ReadFromFile(BiTree \*T){

char filename[20];

cout << "输入读取文件名称（不超过20个字符）:" << endl;

cin >> filename;

strcat(filename, ".txt");

ifstream infile;

infile.open(filename, std::ios::in);

if(infile){

ClearBiTree(T);

Read(T,infile);

infile.close();

return OK;

} else {

cout << "不存在此文件!" << endl;

return ERROR;

}

}

Status Change(int \*Inisinit,int moro,int where,int \* \*how, int state){

if(moro == 1){

(\*how)[where - 1] = state;

\*Inisinit = (\*how)[where - 1];

} else

\*Inisinit = state;

return OK;

}

Status PrintAsTree(BiTree T,Status(\* Visit)(ElemType \*e)){

if(!T){

cout << "树为空树!" << endl;

return OK;

}

BiTree F = (BiTree)malloc(sizeof(BiTNode));//充当空结点

F -> elem = nullptr;

F -> lchild = F -> rchild = nullptr;

int length;

length = strlen(T -> elem ->id);//id的长度

int height;

height = BiTreeDepth(T);//树的高度

int depth = 1;//当前所在的层数(根结点打印后状态)

int children = 1;//已打印结点的数量(根结点打印后的状态)

LinkQueue Q;

InitQueue(Q);

EnQueue(Q,T);

//根结点前面空部分打印

for(int a = 0; a < length \* (pow(2,height - depth) - 1);a++){

printf(" ");

}

//根结点打印

if(!PrintElem(T -> elem))

exit(ERROR);

//剩余结点打印

while(Q.front != Q.rear) {

if(children == pow(2,depth) - 1) {//每一层前面的空部分

depth++;

printf("\n");

for (int a = 0; a < length \* (pow(2,height - depth) - 1); a++) {

printf(" ");

}

}

if(children == pow(2,height) - 1)

break;

DeQueue(Q,T);

if(T -> lchild) {

EnQueue(Q, T->lchild);

if(!PrintElem(T -> lchild -> elem))

exit(ERROR);

for(int a = 0;a < length \* (pow(2,height - depth + 1) - 1);a++)

printf(" ");

children++;

} else{

EnQueue(Q,F);

for(int a = 0;a < length;a++)//充当结点

printf(" ");

for(int a = 0;a < length \* (pow(2,height - depth + 1) - 1);a++)

printf(" ");

children++;

}

if(T -> rchild) {

EnQueue(Q, T->rchild);

if(!PrintElem(T -> rchild -> elem))//打印结点

exit(ERROR);

for(int a = 0;a < length \* (pow(2,height - depth + 1) - 1);a++)

printf(" ");

children++;

}else{

EnQueue(Q,F);

for(int a = 0;a < length;a++)//充当结点

printf(" ");

for(int a = 0;a < length \* (pow(2,height - depth + 1) - 1);a++)

printf(" ");

children++;

}

}

DestroyQueue(Q);

return OK;

}

Status VisitElem(ElemType \*e){

cout << "ID:" << e -> id << endl;

cout << "Name:" << e -> name << endl;

cout << "Phone:" << e -> phone << endl;

return OK;

}

Status PrintElem(ElemType \*e) {

printf("%s",e -> id);

return OK;

}

# 附录D

#include <iostream>

#include <cstring>

#include <fstream>

#define MAXCHARLENGTH 20

#define OK 1

#define FALSE 0

#define MAX 100

using namespace std;

typedef int Status;

typedef struct ElemNext {

ElemNext \*next\_elem;

int current;

} ElemNext;

typedef struct ElemType {

char id[MAXCHARLENGTH];

char name[MAXCHARLENGTH];

char birth[MAXCHARLENGTH];

ElemNext \*next\_elem;

} ElemType;

typedef struct Graph {

struct ElemType \*elemtype;

int number;

} Graph, \*Graphptr;

typedef struct {

int \*base;

int front;

int rear;

} LinkQueue;

int visited[MAX];

Status (\*VisitFunc)(ElemType \*e);

Status CreateVex(ElemType \*\*pElemType,Graphptr \*G);

Status CreateArc(ElemNext \*\*pElemNext,int number,ElemType \*elemType);

Status CreateGraph(Graphptr \*G,ElemType \*V,ElemNext \*VR);

Status DestroyGraph(Graphptr \*G);

Status LocateVex(Graphptr G, ElemType example);

Status GetVex(Graphptr G, ElemType example);

Status PutVex(Graphptr \*G, ElemType example, ElemType value);

Status FirstAdjVex(Graphptr G, ElemType example);

Status NextAdjVex(Graphptr G, ElemType example, ElemType w);

Status InsertVex(Graphptr \*G, ElemType value);

Status DeleteVex(Graphptr \*G, ElemType example);

Status InsertArc(Graphptr \*G, ElemType example, ElemType example\_to);

Status DeleteArc(Graphptr \*G, ElemType example, ElemType example\_to);

Status WriteToFile(Graphptr G);

Status ReadFromFile(Graphptr \*G);

void DFS(Graphptr G, int v);

Status DFSTraverse(Graphptr G, Status(\*Visit)(ElemType \*e));

Status BFSTraverse(Graphptr G, Status(\*Visit)(ElemType \*e));

Status VisitElem(ElemType \*e);

Status InitQueue(LinkQueue &Q);

Status DestroyQueue(LinkQueue &Q);

Status EnQueue(LinkQueue &Q, int v);

Status DeQueue(LinkQueue &Q, int &w);

int main() {

int number = 0;

int moro = 0;

int choice = 1;

int where = 0;

Graphptr G = (Graphptr) malloc(sizeof(Graph));

ElemNext \*pElemNext = nullptr;

ElemType \*pElemType = nullptr;

G->number = 0;

Graphptr \*GS = nullptr;

while (choice) {

cout << "\n\n" << endl;

cout << " Menu for Linear Table On Sequence Structure \n" << endl;

cout << "-------------------------------------------------\n" << endl;

cout << " 1. CreateGraph 10.InsertArc\n" << endl;

cout << " 2. DestroyGraph 11.DeleteArc\n" << endl;

cout << " 3. LocateVex 12.MultiGraph\n" << endl;

cout << " 4. GetVex 13.SwitchGraph\n" << endl;

cout << " 5. PutVex 14.WriteToFile\n" << endl;

cout << " 6. FirstAdjVex 15.ReadFromFile\n" << endl;

cout << " 7. NextAdjVex 16.DFSTraverse\n" << endl;

cout << " 8. InsertVex 17.BFSTraverse\n" << endl;

cout << " 9. DeleteVex 0. Exit\n" << endl;

cout << "-------------------------------------------------\n" << endl;

if (moro == 1)

cout << "当前操作的为图 " << where + 1 << endl;

cout << " 请选择你的操作[0~17]:" << endl;

cin >> choice;

switch (choice) {

case 1: {

cout << "下面创建顶点集!" << endl;

system("pause");

CreateVex(&pElemType,&G);

cout << "下面开始建立邻接关系:" << endl;

system("pause");

CreateArc(&pElemNext,G -> number,pElemType);

CreateGraph(&G,pElemType,pElemNext);

system("pause");

break;

}

case 2: {

DestroyGraph(&G);

free(pElemNext);//释放邻接顶点前的首部所占用的空间

system("pause");

break;

}

case 3: {

ElemType example;

cout << "请输入所查找顶点的ID" << endl;

cin >> example.id;

LocateVex(G, example);

system("pause");

break;

}

case 4: {

ElemType example;

cout << "请输入所查找顶点的ID" << endl;

cin >> example.id;

GetVex(G, example);

system("pause");

break;

}

case 5: {

ElemType example;

cout << "请输入要修改顶点的ID" << endl;

cin >> example.id;

ElemType value;

cout << "请输入修改后的顶点信息(姓名以及出生日期)" << endl;

cin >> value.name >> value.birth;

for(int a = 0;a < MAXCHARLENGTH;a++)

value.id[a] = example.id[a];

PutVex(&G, example, value);

system("pause");

break;

}

case 6: {

ElemType example;

cout << "请输入顶点ID" << endl;

cin >> example.id;

FirstAdjVex(G, example);

system("pause");

break;

}

case 7: {

ElemType example;

cout << "请输入顶点ID" << endl;

cin >> example.id;

ElemType w;

cout << "请输入邻接顶点ID" << endl;

cin >> w.id;

NextAdjVex(G, example, w);

system("pause");

break;

}

case 8: {

ElemType value;

cout << "请输入插入顶点信息(学号、姓名以及出生日期)" << endl;

cin >> value.id >> value.name >> value.birth;

InsertVex(&G, value);

system("pause");

break;

}

case 9: {

ElemType example;

cout << "请输入顶点ID" << endl;

cin >> example.id;

DeleteVex(&G, example);

system("pause");

break;

}

case 10: {

ElemType example;

cout << "请输入弧尾顶点ID" << endl;

cin >> example.id;

ElemType example\_to;

cout << "请输入弧头顶点ID" << endl;

cin >> example\_to.id;

InsertArc(&G, example, example\_to);

system("pause");

break;

}

case 11: {

ElemType example;

cout << "请输入弧尾顶点ID" << endl;

cin >> example.id;

ElemType example\_to;

cout << "请输入弧头顶点ID" << endl;

cin >> example\_to.id;

DeleteArc(&G, example, example\_to);

system("pause");

break;

}

case 12: {

if (moro == 1) {

cout << "当前已创建多图，无需再次创建!" << endl;

system("pause");

break;

} else {

cout << "请输入建立多图的数目!" << endl;

while (cin >> number) {

if (number > 0)

break;

else

cout << "输入的数字不合法，请重新输入!" << endl;

}

//建立多图

moro = 1;

GS = (Graphptr \*) malloc(number \* sizeof(Graphptr));

//将每个图初始化

for (int i = 0; i < number; i++) {

GS[i] = (Graphptr) malloc(sizeof(Graph));

GS[i]->number = 0;

}

//切换为第一个图

where = 0;

G = GS[where];

cout << "创建多图成功!" << endl;

system("pause");

break;

}

}

case 13: {

if (moro == 0) {

cout << "不存在多图，无法切换，请重新选择你的操作!" << endl;

system("pause");

break;

} else {

cout << "请输入图的序号" << endl;

int c;

while (cin >> c) {

if (c >= 1 && c <= number)

break;

else

cout << "输入的序号不合法，请重新输入你的序号!" << endl;

}

where = c - 1;

G = GS[where];

cout << "切换成功!" << endl;

system("pause");

break;

}

}

case 14: {

WriteToFile(G);

system("pause");

break;

}

case 15: {

ReadFromFile(&G);

system("pause");

break;

}

case 16: {

cout << "深度优先搜索如下:" << endl;

DFSTraverse(G, VisitElem);

system("pause");

break;

}

case 17: {

cout << "广度优先搜索如下:" << endl;

BFSTraverse(G, VisitElem);

system("pause");

break;

}

case 0: {

cout << "即将推出本程序!" << endl;

cout << "感谢您的使用!" << endl;

system("pause");

break;

}

default: {

cout << "输入指令不存在，请重新选择你的操作!" << endl;

system("pause");

break;

}

}

}

}

Status CreateVex(ElemType \*\*pElemType,Graphptr \*G){

if ((\*G) -> number != 0) {

cout << "当前图不为空图，请销毁后再创建!" << endl;

return FALSE;

}

int number;

cout << "请输入顶点的数目:" << endl;

while (cin >> number) {

if (number > 0)

break;

else

cout << "顶点数目不合法，请重新输入!" << endl;

}

int a;

(\*G) -> number = number;

(\*pElemType) = (ElemType \*)malloc(number \* sizeof(ElemType));

ElemType e;

e.next\_elem = nullptr;

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

(\*pElemType)[i].next\_elem = nullptr;

for (int i = 0; i < number; ++i) {

cout << "请依次输入第" << i + 1 << "顶点的学生学号、姓名以及出生日期(每条信息不超过20个字符)" << endl;

while(cin >> e.id >> e.name >> e.birth){

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

if(!strcmp(e.id,(\*pElemType)[a].id)){

cout << "当前输入顶点的ID已存在，请重新输入学生的所有信息!" << endl;

break;//跳出遍历图

}

}

if(a == i + 1)

break;//跳出重复输入

}

(\*pElemType)[i] = e;

}

cout << "创建顶点集成功!" << endl;

return OK;

}

Status CreateArc(ElemNext \*\*pElemNext,int number,ElemType\* elemType){

(\*pElemNext) = (ElemNext\*)malloc(number \* sizeof(ElemNext));

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

(\*pElemNext)[i].next\_elem = nullptr;

//删除原有关系

cout << "若当前图的顶点已建立邻接关系，再次执行此操作会覆盖顶点之前建立的邻接关系!" << endl;

cout << "是否继续? 是：1 否：0" << endl;

int choice;

if (cin >> choice && choice == 0)

return OK;

//建立新的关系

int a = 0;//当前顶点索引

int b = 0;//被指顶点的索引

cout << "所有的顶点信息如下:" << endl;

for (int i = 0; i < number; i++) {

cout << i + 1 << ": " << "\nID: " << elemType[i].id

<< "\nName:" << elemType[i].name

<< "\nBirth: " << elemType[i].birth

<< endl;

}

ElemType search;

while (true) {

cout << "请输入要建立关系的顶点的学号(学号输入 # 停止建立关系)" << endl;

cin >> search.id;

if (!strcmp("#", search.id))

break;

for (a = 0; a < number; a++) {

if (!strcmp(search.id, elemType[a].id))

break;

if (a == number - 1)

cout << "无符合关系的顶点" << endl;

}

if (a == number) {

cout << "请重新输入要建立关系的顶点的学号(学号输入 # 停止建立关系)" << endl;

continue;

} else {

ElemType e;

while (true) {

cout << "请输入学号: " << elemType[a].id << "顶点所指向顶点的学号(输入 \* 结束输入)" << endl;

cin >> e.id;

if (!strcmp(e.id, "\*"))

break;

for (b = 0; b < number; b++) {

if (!strcmp(e.id, elemType[b].id))

break;

if (b == number - 1)

cout << "无符合关系的顶点" << endl;

}

if (b == number) {

cout << "请重新输入学号: " << elemType[a].id << "顶点所指向结点的学号(输入 \* 结束输入)" << endl;

continue;

} else {

//头插法

ElemNext \*elemNext = (\*pElemNext)[a].next\_elem;

while(elemNext){

if(elemNext -> current == b){

cout << "当前弧已存在，请重新输入!" << endl;

break;

}

elemNext = elemNext -> next\_elem;

}

if(!elemNext) {

ElemNext \*p = (\*pElemNext)[a].next\_elem;

(\*pElemNext)[a].next\_elem = (ElemNext \*) malloc(sizeof(ElemNext));

(\*pElemNext)[a].next\_elem->next\_elem = p;

(\*pElemNext)[a].next\_elem->current = b;

}//插入弧关系

}

}

}

}

cout << "创建关系集成功!" << endl;

return OK;

}

Status CreateGraph(Graphptr \*G,ElemType \*V,ElemNext \*VR){

(\*G) -> elemtype = V;

for(int i = 0;i < (\*G) -> number; i++)

(\*G) -> elemtype[i].next\_elem = VR[i].next\_elem;

cout << "创建图成功!" << endl;

}

Status DestroyGraph(Graphptr \*G) {

if ((\*G)->number == 0) {

cout << "当前图未创建，故无法销毁!" << endl;

return FALSE;

}

ElemNext \*H;

ElemNext \*P;

for (int i = 0; i < (\*G)->number; i++) {

H = (\*G)->elemtype[i].next\_elem;

while (H) {

P = H -> next\_elem;

free(H);

H = P;

}

}

free((\*G)->elemtype);

(\*G)->number = 0;

cout << "销毁图成功!" << endl;

return OK;

}

Status LocateVex(Graphptr G, ElemType example) {

if(!G -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

int i;

for (i = 0; i < G->number; i++) {

if (!strcmp(G->elemtype[i].id, example.id)) {

cout << "所找顶点的序号为:" << i + 1 << endl;

break;

}

if (i == G->number - 1)

cout << "不存在所找顶点，请重新选择你的操作!" << endl;

}

return OK;

}

Status GetVex(Graphptr G, ElemType example) {

if(!G -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

int i;

for (i = 0; i < G->number; i++) {

if (!strcmp(G->elemtype[i].id, example.id)) {

cout << "所找顶点的信息为:" << endl;

cout << "Id: " << G->elemtype[i].id << endl;

cout << "Name: " << G->elemtype[i].name << endl;

cout << "Birth: " << G->elemtype[i].birth << endl;

break;

}

if (i == G->number - 1)

cout << "不存在所找顶点，请重新选择你的操作!" << endl;

}

return OK;

}

Status PutVex(Graphptr \*G, ElemType example, ElemType value) {

if(!(\*G) -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

int i;

for (i = 0; i < (\*G)->number; i++) {

if (!strcmp((\*G)->elemtype[i].id, example.id)) {

cout << "所改顶点的原有信息为:" << endl;

cout << "Id: " << (\*G)->elemtype[i].id << endl;

cout << "Name: " << (\*G)->elemtype[i].name << endl;

cout << "Birth: " << (\*G)->elemtype[i].birth << endl;

value.next\_elem = (\*G)->elemtype[i].next\_elem;

(\*G)->elemtype[i] = value;

cout << "修改后的顶点信息为:" << endl;

cout << "Id: " << (\*G)->elemtype[i].id << endl;

cout << "Name: " << (\*G)->elemtype[i].name << endl;

cout << "Birth: " << (\*G)->elemtype[i].birth << endl;

break;

}

if (i == (\*G)->number - 1)

cout << "不存在所找顶点，请重新选择你的操作!" << endl;

}

return OK;

}

Status FirstAdjVex(Graphptr G, ElemType example) {

if(!G -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

int i;

for (i = 0; i < G->number; i++) {

if (!strcmp(G->elemtype[i].id, example.id)) {

if (G->elemtype[i].next\_elem != nullptr) {

cout << "所找顶点的第一个邻接顶点信息为:" << endl;

cout << "Id: " << G->elemtype[G->elemtype[i].next\_elem->current].id << endl;

cout << "Name: " << G->elemtype[G->elemtype[i].next\_elem->current].name << endl;

cout << "Birth: " << G->elemtype[G->elemtype[i].next\_elem->current].birth << endl;

}else

cout << "所找顶点不存在第一个邻接顶点!" << endl;

break;

}

if (i == G->number - 1)

cout << "不存在所找顶点，请重新选择你的操作!" << endl;

}

return OK;

}

Status NextAdjVex(Graphptr G, ElemType example, ElemType w) {

if(!G -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

int i;

ElemNext \*e;

for (i = 0; i < G->number; i++) {

if (!strcmp(G->elemtype[i].id, example.id)) {

e = G->elemtype[i].next\_elem;

while (e != nullptr) {

if (!strcmp(G->elemtype[e->current].id, w.id)) {

if (e->next\_elem != nullptr) {

cout << "符合条件的顶点信息为:" << endl;

cout << "Id: " << G->elemtype[e->next\_elem->current].id << endl;

cout << "Name: " << G->elemtype[e->next\_elem->current].name << endl;

cout << "Birth: " << G->elemtype[e->next\_elem->current].birth << endl;

return OK;

} else {

cout << "所找邻接顶点的下一个邻接顶点为空!" << endl;

return FALSE;

}

}

e = e->next\_elem;

}

if (!e)

cout << "不存在所找顶点的邻接顶点" << endl;

break;

}

if (i == G->number - 1)

cout << "不存在所找顶点，请重新选择你的操作!" << endl;

}

return FALSE;

}

Status InsertVex(Graphptr \*G, ElemType value) {

if(!(\*G) -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

for(int i = 0;i < (\*G) -> number;i++){

if(!strcmp(value.id,(\*G) -> elemtype[i].id)){

cout << "当前顶点ID已存在，请重新选择你的操作!" << endl;

return FALSE;

}

}

value.next\_elem = nullptr;

Graphptr GNew = (Graphptr) malloc(sizeof(Graph));

GNew -> elemtype = (ElemType \*) malloc(((\*G)->number + 1) \* sizeof(ElemType));

GNew -> number = ((\*G)->number + 1);

for (int i = 0; i < ((\*G)->number + 1); ++i) {

GNew->elemtype[i].next\_elem = nullptr;

if (i == (\*G)->number)

GNew->elemtype[i] = value;

else

GNew->elemtype[i] = (\*G)->elemtype[i];

}

free((\*G) -> elemtype);

\*G = GNew;

cout << "插入顶点成功!" << endl;

return OK;

}

Status DeleteVex(Graphptr \*G, ElemType example) {

if(!(\*G) -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

int a;

Graphptr GNew = (Graphptr) malloc(sizeof(Graph));

GNew -> elemtype = (ElemType \*) malloc(((\*G)->number - 1) \* sizeof(ElemType));

GNew -> number = ((\*G)->number - 1);

for (a = 0; a < (\*G)->number; a++) {

if (!strcmp((\*G)->elemtype[a].id, example.id))

break;

if (a == (\*G)->number - 1) {

cout << "不存在所找顶点，请重新选择你的操作!" << endl;

return FALSE;

}

}

for(int i = 0;i < (\*G) -> number;i++){

ElemNext \*h = (\*G) -> elemtype[i].next\_elem;

ElemNext \*k = h;

ElemNext \*m;

while(h){

m = h -> next\_elem;//在h未被释放掉的时候取h的下一个邻接顶点

if(h -> current == a)

{

if((\*G) -> elemtype[i].next\_elem -> current == h -> current){

ElemNext \*q = (\*G) -> elemtype[i].next\_elem;

(\*G) -> elemtype[i].next\_elem = q -> next\_elem;

free(q);

}

else{

k -> next\_elem = h -> next\_elem;

free(h);

}

}

else if(h -> current > a)

--(h -> current);

k = h;

h = m;

}

}//删除所有与删除顶点有关的弧关系

for (int i = 0; i < (\*G)->number - 1; ++i) {

GNew->elemtype[i].next\_elem = nullptr;

if (i < a)

GNew->elemtype[i] = (\*G)->elemtype[i];

else

GNew->elemtype[i] = (\*G)->elemtype[i + 1];

}//复制图

ElemNext \*elemNext = (\*G) -> elemtype[a].next\_elem;

ElemNext \*p;

while(elemNext){

p = elemNext -> next\_elem;

free(elemNext);

elemNext = p;

}//释放顶点存储所有的邻接顶点占用的连续空间

free((\*G) -> elemtype);

\*G = GNew;

cout << "删除顶点成功!" << endl;

return OK;

}

Status InsertArc(Graphptr \*G, ElemType example, ElemType example\_to) {

if(!(\*G) -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

int i;

for (i = 0; i < (\*G)->number; i++) {

if (!strcmp((\*G)->elemtype[i].id, example.id)) {

break;

}

if (i == (\*G)->number - 1) {

cout << "不存在所找弧尾顶点，请重新选择你的操作!" << endl;

return FALSE;

}

}

int l;

for (l = 0; l < (\*G)->number; l++) {

if (!strcmp((\*G)->elemtype[l].id, example\_to.id)) {

break;

}

if (l == (\*G)->number - 1) {

cout << "不存在所找弧头顶点，请重新选择你的操作!" << endl;

return FALSE;

}

}

ElemNext \*H = (\*G)->elemtype[i].next\_elem;

if (H == nullptr) {

(\*G)->elemtype[i].next\_elem = (ElemNext \*) malloc(sizeof(ElemNext));

(\*G)->elemtype[i].next\_elem->current = l;

(\*G)->elemtype[i].next\_elem->next\_elem = nullptr;

cout << "插入弧成功!" << endl;

return OK;

}

while (H->next\_elem) {

if (!strcmp((\*G)->elemtype[H->current].id, (\*G)->elemtype[l].id)) {

cout << "当前弧关系已存在，请重新选择你的操作!" << endl;

return FALSE;

}

H = H->next\_elem;

}

//头插法

H = (ElemNext \*) malloc(sizeof(ElemNext));

H -> current = l;

H -> next\_elem = (\*G) -> elemtype[i].next\_elem;

(\*G) -> elemtype[i].next\_elem = H;

cout << "插入弧成功!" << endl;

return OK;

}

Status DeleteArc(Graphptr \*G, ElemType example, ElemType example\_to) {

if(!(\*G) -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

int i;

for (i = 0; i < (\*G)->number; i++) {

if (!strcmp((\*G)->elemtype[i].id, example.id)) {

break;

}

if (i == (\*G)->number - 1) {

cout << "不存在所找弧尾顶点，请重新选择你的操作!" << endl;

return FALSE;

}

}

int l;

for (l = 0; l < (\*G)->number; l++) {

if (!strcmp((\*G)->elemtype[l].id, example\_to.id)) {

break;

}

if (l == (\*G)->number - 1) {

cout << "不存在所找弧头顶点，请重新选择你的操作!" << endl;

return FALSE;

}

}

ElemNext \*H = (\*G)->elemtype[i].next\_elem;

ElemNext \*P = H;

if (H == nullptr) {

cout << "所找顶点不存在邻接顶点，请重新选择你的操作!" << endl;

return FALSE;

} else if (!strcmp((\*G)->elemtype[H->current].id, (\*G)->elemtype[l].id)) {

//第一个邻接顶点符合条件

(\*G)->elemtype[i].next\_elem = H->next\_elem;

free(H);

cout << "删除弧成功!" << endl;

return OK;

} else {

while (H) {

if (!strcmp((\*G)->elemtype[H->current].id, (\*G)->elemtype[l].id)) {

P -> next\_elem = H -> next\_elem;

cout << "删除弧成功!" << endl;

free(H);

return OK;

}

P = H;

H = H->next\_elem;

}

if (!H) {

cout << "不存在所找弧关系，请重新选择你的操作!" << endl;

return FALSE;

}

}

}

Status WriteToFile(Graphptr G) {

if(!G -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

char filename[20];

cout << "输入文件名称（不超过20个字符）:" << endl;

cin >> filename;

strcat(filename, ".txt");

ofstream outfile;

outfile.open(filename, std::ios::out);

if (outfile) {

outfile << G->number << endl;

for (int i = 0; i < G->number; i++) {

outfile << G->elemtype[i].id << endl;

outfile << G->elemtype[i].name << endl;

outfile << G->elemtype[i].birth << endl;

ElemNext \*elemNext = G->elemtype[i].next\_elem;

while (elemNext) {

outfile << elemNext->current << endl;

elemNext = elemNext->next\_elem;

}

outfile << "-1" << endl;

}

outfile.close();

cout << "创建文件成功!" << endl;

return OK;

} else

cout << "创建文件失败!" << endl;

return FALSE;

}

Status ReadFromFile(Graphptr \*G) {

if((\*G) -> number != 0){

cout << "当前图不为空，请销毁图后再读取文件!" << endl;

return FALSE;

}

char filename[20];

cout << "输入文件名称（不超过20个字符）:" << endl;

cin >> filename;

strcat(filename, ".txt");

ifstream infile;

infile.open(filename, std::ios::in);

if (infile) {

infile >> (\*G)->number;

(\*G)->elemtype = (ElemType \*) malloc((\*G)->number \* sizeof(ElemType));

ElemType elemType;

for (int i = 0; i < (\*G)->number; ++i) {

(\*G)->elemtype[i].next\_elem = nullptr;

infile >> elemType.id >> elemType.name >> elemType.birth;

elemType.next\_elem = nullptr;

(\*G)->elemtype[i] = elemType;

ElemNext elemNext;

//处理第一个邻接顶点

if (infile >> elemNext.current && elemNext.current != -1) {

(\*G)->elemtype[i].next\_elem = (ElemNext \*) malloc(sizeof(ElemNext));

(\*G)->elemtype[i].next\_elem->current = elemNext.current;

(\*G)->elemtype[i].next\_elem->next\_elem = nullptr;

} else

continue;

//处理其他的邻接顶点

ElemNext \*e = (\*G)->elemtype[i].next\_elem;

while (infile >> elemNext.current && elemNext.current != -1) {

e->next\_elem = (ElemNext \*) malloc(sizeof(ElemNext));

e = e->next\_elem;

e->current = elemNext.current;

e->next\_elem = nullptr;

}

}

infile.close();

cout << "读取成功!" << endl;

return OK;

} else {

cout << "不存在此文件!" << endl;

return FALSE;

}

}

void DFS(Graphptr G, int v) {

//此处改变标志组的状态

visited[v] = 1;

VisitFunc(&(G->elemtype[v]));

ElemNext \*elemNext = G->elemtype[v].next\_elem;

while (elemNext) {

if (!visited[elemNext->current])

DFS(G, elemNext->current);

elemNext = elemNext->next\_elem;

}

}

Status DFSTraverse(Graphptr G, Status(\*Visit)(ElemType \*e)) {

if(!G -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

for (int i = 0; i < G->number; i++) {

visited[i] = 0;

}//标志数组初始化

VisitFunc = Visit;

int v;

for (v = 0; v < G->number; v++)

if (!visited[v])

DFS(G, v);

}

Status BFSTraverse(Graphptr G, Status(\*Visit)(ElemType \*e)) {

if(!G -> number){

cout << "当前图为空，请重新选择你的操作!" << endl;

return OK;

}

for (int i = 0; i < G->number; i++) {

visited[i] = 0;

}//标志数组初始化

LinkQueue Q;

InitQueue(Q);

int v;

int w = -1;

ElemNext \*elemNext;

for (v = 0; v < G->number; v++) {

if (!visited[v]) {

//访问顶点

EnQueue(Q, v);

visited[v] = 1;

Visit(&(G->elemtype[v]));

//访问邻接顶点

while (Q.front != Q.rear) {//判断队列是否为空

DeQueue(Q, w);

elemNext = G->elemtype[w].next\_elem;

while (elemNext) {

if (!visited[elemNext->current]) {

Visit(&(G->elemtype[elemNext->current]));

EnQueue(Q, elemNext->current);

visited[elemNext->current] = 1;

}

elemNext = elemNext->next\_elem;

}

}

}

}

DestroyQueue(Q);

return OK;

}

Status VisitElem(ElemType \*e) {

cout << "ID:" << e->id << endl;

cout << "Name:" << e->name << endl;

cout << "Birth:" << e->birth << endl;

cout << endl;

return OK;

}

Status InitQueue(LinkQueue &Q) {

Q.base = (int \*) malloc(MAX \* sizeof(int));

if (!Q.base)

exit(-1);

Q.front = Q.rear = 0;

return OK;

}

Status DestroyQueue(LinkQueue &Q) {

free(Q.base);

return OK;

}

Status EnQueue(LinkQueue &Q, int v) {

if ((Q.rear + 1) % MAX == Q.front)

return -2;//队列空间满

Q.base[Q.rear] = v;

Q.rear = (Q.rear + 1) % MAX;

return OK;

}

Status DeQueue(LinkQueue &Q, int &w) {

if (Q.front == Q.rear)

return -2;//队列空间满

w = Q.base[Q.front];

Q.front = (Q.front + 1) % MAX;

return OK;

}