

華中科技大学

课程实验报告

课程名称： 数据结构实验

专业班级 CS2210

学 号 U202115473

姓 名 刘欣逸

指导教师 郑渤龙

报告日期 2023年5月29日

计算机科学与技术学院

## 目 录

1	基于顺序存储结构的线性表实现.....	1
1.1	问题描述 .....	1
1.2	系统设计 .....	2
1.3	系统实现 .....	6
1.4	系统测试 .....	6
2	基于二叉链表的二叉树实现 .....	17
2.1	问题描述 .....	17
2.2	系统设计 .....	19
2.3	系统实现 .....	23
2.4	系统测试 .....	23
3	实验总结 .....	29
4	附录 A 基于顺序存储结构线性表实现的源程序 .....	29
5	附录 B 基于链式存储结构线性表实现的源程序.....	48
6	附录 C 基于二叉链表二叉树实现的源程序 .....	69
7	附录 D 基于邻接表图实现的源程序 .....	91

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

要求构造一个具有菜单的功能演示系统。该演示系统实现多个线性表管理。其中，在主函数中准备函数调用所需实参值、显示函数执行结果，并给出适当的操作提示。

实现依据最小完备性和常用性相结合的原则确定的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 12 种基本运算，具体运算功能定义如下。

1. 初始化表：函数名称是**InitaList(L)**；初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。
2. 销毁表：函数名称是**DestroyList(L)**；初始条件是线性表L已存在；操作结果是销毁线性表L。
3. 清空表：函数名称是**ClearList(L)**；初始条件是线性表L已存在；操作结果是将L重置为空表。
4. 判定空表：函数名称是**ListEmpty(L)**；初始条件是线性表L已存在；操作结果是若L为空表则返回TRUE, 否则返回FALSE。
5. 求表长：函数名称是**ListLength(L)**；初始条件是线性表L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L已存在， $1 \leq i \leq \text{ListLength}(L)$ ；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,compare())**；初始条件是线性表L已存在；操作结果是返回L中第一个与e满足关系**compare()**关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。
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，

$1 \leq i \leq \text{ListLength}(L) + 1$ ; 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， $1 \leq i \leq \text{ListLength}(L)$ ; 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, visit()), 初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。

此外，还在基础功能的基础上实现了附加功能：

1. 最大连续子数组和：函数名称是MaxSubArray(L); 初始条件是线性表L已存在且非空，请找出一个具有最大和的连续子数组（子数组最少包含一个元素），操作结果是其最大和；
2. 和为k的子数组：函数名称是SubArrayNum(L, k); 初始条件是线性表L已存在且非空，操作结果是该数组中和为k的连续子数组的个数；
3. 顺序表排序：函数名称是SortList(L); 初始条件是线性表L已存在；操作结果是将L由小到大排序；
4. 实现线性表的文件形式保存：其中，
  - 需要设计文件数据记录格式，以高效保存线性表数据逻辑结构(D, {R})的完整信息；
  - 需要设计线性表文件保存和加载操作合理模式。
5. 实现多个线性表管理：设计相应的数据结构管理多个线性表的查找、添加、移除等功能。

实验目的：

1. 加深对线性表的概念、基本运算的理解；
2. 熟练掌握线性表的逻辑结构与物理结构的关系；
3. 物理结构采用顺序表，熟练掌握线性表的基本运算的实现。

## 1.2 系统设计

本系统提供一个采用顺序存储方式的线性表及其操作实现。系统可供选择的操作有：

- 基本操作：初始化线性表、销毁表、清空表、判定空表，求表长、获得元素、查找元素、获得某元素的前驱、获得某元素的后继、插入元素、删除元素、遍历线性表。
- 附加功能：最大连续子数组和、和为K的子数组、顺序表排序、实现线性表的文件形式保存、实现多个线性表管理。

## 1.2.1 头文件、宏和类型定义

```
#include <stdio.h>                                1
#include <stdlib.h>                                 2
#include <string.h>                                 3
#include <limits.h>                                 4

#define TRUE 1                                     5
#define FALSE 0                                    6
#define OK 1                                       7
#define ERROR 0                                     8
#define INFEASIBLE -1                            9
#define OVERFLOW -2                             10
#define ISEMPTRY -3                            11
#define LIST_INIT_SIZE 1000                      12
#define LISTINCREMENT 10                         13
#define max(i,j) ((i)>(j)?(i):(j))           14
                                              15
typedef int status;                               16
typedef int ElemType; // 数据元素类型定义      17
typedef struct{ // 顺序表（顺序结构）的定义    18
    ElemType * elem;                           19
    int length;                            20
    int listszie;                          21
} SqList;                                         22

typedef struct THELISTS{ // 顺序表的管理表定义   23
    struct ALIST{
        char name[30];
        SqList L;
    } elem[50];
    int length = 0;
    int listssize = 50;
} LISTS;                                         24
                                              25
                                              26
                                              27
                                              28
                                              29
                                              30
                                              31
                                              32
```

### 1.2.2 函数设计

1. 函数名称: `InitaList(L);`

初始条件: 线性表L不存在;

操作结果: 是构造一个空的线性表;

设计思路: 先分配存储空间后, 将表长设为0, 再将线性表容量变量设为预定义的初始存储容量。

2. 函数名称: `DestroyList(L);`

初始条件: 线性表L已存在;

操作结果: 销毁线性表L;

设计思路: 释放内存并将其他结构成员设置为初值。

3. 函数名称: `ClearList(L);`

初始条件: 线性表L已存在;

操作结果: 将L重置为空表;

设计思路: 将表长设为0。

4. 函数名称: `ListEmpty(L);`

初始条件: 线性表L已存在;

操作结果: 若L为空表则返回TRUE, 否则返回FALSE;

设计思路: 表长为0则为空表, 否则不是空。

5. 函数名称: `ListLength(L);`

初始条件: 线性表L已存在;

操作结果: 返回L中数据元素的个数;

设计思路: 返回线性表表长的值。

6. 函数名称: `GetElem(L, i, e);`

初始条件: 线性表已存在,  $1 \leq i \leq \text{ListLength}(L)$ ;

操作结果: 用e返回L中第i个数据元素的值;

设计思路: 将线性表中第i个数据元素的值赋值给e。

7. 函数名称: `LocateElem(L, e, compare());`

初始条件: 线性表已存在;

操作结果: 返回线性表中第1个与e相等的数据元素的位置, 若这样的数据元素不存在, 则返回值为0;

设计思路: 从索引低位开始顺序查找, 如果找到返回该元素的位序

8. 函数名称: `PriorElem(L,cur_e,pre_e);`

初始条件: 线性表L已存在;

操作结果: 若`cur_e`是L的数据元素并且不是第一个数据元素, 用`pre_e`返回它的前驱, 否则操作失败。

设计思路: 首先判断该元素不是第一个数据元素, 再查找该结点, 如果查找成功且该结点有前驱元素, 则将前驱元素赋值给`pre_e`。若未找到该结点, 或者该结点是第一个数据元素, 则返回`ERROR`。

9. 函数名称: `NextElem(L,cur_e,next_e)`

初始条件: 线性表L已存在;

操作结果: 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。

设计思路: 首先判断该元素不是最后一个数据元素, 再查找该结点, 如果查找成功且该结点有后继元素, 则将后继元素赋值给`next_e`。若未找到该结点, 或者该结点是最后一个数据元素, 则返回`ERROR`。

10. 函数名称: `ListInsert(L,i,e)`

初始条件: 线性表L已存在且非空,  $1 \leq i \leq \text{ListLength}(L) + 1$ ;

操作结果: 在L的第*i*个位置之前插入新的数据元素e。

设计思路: 先遍历顺序表, 若线性表L已存在且不为空, 输入的*i*值不合法, 则返回`ERROR`; 若*i*的值合法, 则在线性表的第*i*个位置前插入新数据元素e, 返回`OK`。若线性表L不存在, 返回`INFEASIBLE`.

11. 函数名称: `ListDelete(L,i,e)`;

初始条件: 线性表L已存在且非空,  $1 \leq i \leq \text{ListLength}(L)$ ;

操作结果: 删除L的第*i*个数据元素, 用e返回其值;

设计思路: 先遍历顺序表, 如果线性表L已存在且非空, 并且输入的*i*值不合法, 则返回`ERROR`。若满足线性表L已存在且L非空, 并且*i*的值合法, 则删除线性表的第*i*个位置的数据元素, 并用e返回其值, 返回`OK`。

12. 函数名称: `ListTraverse(L)`;

初始条件: 线性表L已存在;

操作结果: 依次遍历L的每个数据元素;

设计思路: 若线性表L存在, 则依序遍历元素; 否则返回`ERROR`。

13. 函数名称: `SaveList(L,filename)`;

初始条件：线性表L已存在；

操作结果：将线性表L保存到指定文件；

设计思路：用`fprintf`保存为文件。

14. 函数名称：`LoadList(L)`；

初始条件：文件`filename`已存在；

操作结果：从文件中加载数据到L；

设计思路：用`fscanf`将文件读取顺序表。

15. 函数名称：`MaxSubArray(L)`；

初始条件：线性表L已存在；

操作结果：返回线性表L的最大连续子数组和；

设计思路：逐个枚举子数组求出最大连续子数组和。

16. 函数名称：`SubArrayNum(L, k)`；

初始条件：线性表L已存在；

操作结果：返回线性表L的和为k的连续子数组个数；

设计思路：逐个枚举子数组求出和为k的连续子数组个数。

17. 函数名称：`SortList(L)`

初始条件：线性表L已存在；

操作结果：将L中的元素按升序排序；

设计思路：用冒泡排序算法将L的数据元素按升序排序。

## 1.3 系统实现

见《附录A 基于顺序存储结构线性表实现的源程序》。

## 1.4 系统测试

先用插入操作创建包含元素1、2、3、4、5、6、7的顺序表。再测试判空、查找、删除、遍历、销毁等基础功能。

1. `InitList()`

# 华中科技大学课程实验报告

```
; exit;
(base) xinyiliu@xinyideMacBook-Air ~ % /Users/xinyiliu/Desktop/华科2023春/数据结构实验/顺序表实验/顺序表实验/executable ; exit;

Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作 [0~21]:
1
顺序表创建成功!
```

2. ListInsert() 依次插入了元素 1、2、3、4、5、6、7。

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作 [0~21]:10
请输入插入的元素的逻辑序号 (从1开始) : 1
请输入插入的元素的值 : 1
顺序表插入成功!
10
```

# 华中科技大学课程实验报告

## 3. ListEmpty()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作[0~21]:4
顺序表不为空!

Now working on List default = [0]
Menu for Linear Table On Sequence Structure
```

## 4. ListLength()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作[0~21]:5
顺序表长度为： 7 。

Now working on List default = [0]
```

# 华中科技大学课程实验报告

## 5. GetElem()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit

请选择你的操作[0~21]:6
请输入查找的元素的逻辑序号（从1开始）：3
顺序表的第 3 个元素为 3。
```

## 6. LocateElem()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit

请选择你的操作[0~21]:7
请输入查找的目标元素：4
元素 4 第一次出现在第 4 个元素上。
```

# 华中科技大学课程实验报告

## 7. PriorElem()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit

请选择你的操作 [0~21]:8
请输入要查找前驱的元素: 4
元素 4 的前驱是 3。
```

## 8. NextElem()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit

请选择你的操作 [0~21]:9
请输入要查找后继的元素: 4
元素 4 的后继是 5。
```

# 华中科技大学课程实验报告

## 9. ListDelete()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit

请选择你的操作 [0~21]:11
请输入要删除的元素的逻辑序号： 7
被删除的元素是 7。
```

## 10. ListTraverse()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit

请选择你的操作 [0~21]:12
-----all elements -----
1 2 3 4 5 6
-----end -----
遍历成功！
```

# 华中科技大学课程实验报告

## 11. ClearList()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作 [0~21]:3
顺序表清空成功!
```

## 12. DestroyList()

```
子数组和为 6 的个数为 2 。

Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作 [0~21]:2
顺序表删除成功！

Now working on List default = [0]
```

# 华中科技大学课程实验报告

对顺序表 7,6,5,4,3,2,1 测试附加功能：

## 1. MaxSubArray()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作 [0~21]:13
最大子数组和为 28 。
```

## 2. SubArrayNum()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作 [0~21]:14
请输入子数组和K: 6
子数组和为 6 的个数为 2 。
```

### 3. SortList()

```
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作[0~21]:15
排序操作成功！

Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作[0~21]:12
-----
all elements -----
1 2 3 4 5 6 7
-----
end -----
遍历成功！
```

# 华中科技大学课程实验报告

4. SaveList()、LoadList()、List()等多线性表管理功能。

```
xinyiliu — executable — executable — 69x25
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList    9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit

请选择你的操作[0~21]:16
请输入你要保存的文件名:
file
保存顺序表到文件操作成功!
```

```
xinyiliu — executable — executable — 69x25
Now working on List default = [0]
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList    9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit

请选择你的操作[0~21]:18
请输入你要添加的顺序表名:
b
添加顺序表成功!
```

```
Now working on List b
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作 [0~21]:20
请输入你要切换到的顺序表名:
b
切换到顺序表 b 成功!
```

```
Now working on List b
Menu for Linear Table On Sequence Structure
-----
1. InitList      7. LocateElem
2. DestroyList   8. PriorElem
3. ClearList     9. NextElem
4. ListEmpty     10. ListInsert
5. ListLength    11. ListDelete
6. GetElem       12. ListTraverse
13. MaxSubArray  14. SubArrayNum
15. SortList     16. SaveList
17. LoadList     18. AddList
19. RemoveList   20. SwitchList
21. ShowLists    0. Exit
-----
请选择你的操作 [0~21]:17
请输入你要读取的文件名:
file
请输入你要读取到的顺序表名:
b
-----all elements -----
1 2 3 4 5 6 7
-----end -----
从文件读取到顺序表操作成功!
```

## 2 基于二叉链表的二叉树实现

要求构造一个具有菜单的功能演示系统。该演示系统实现二叉树管理。其中，在主函数中准备函数调用所需实参值、显示函数执行结果，并给出适当的操作提示。

实现依据最小完备性和常用性相结合的原则确定的创建二叉树、销毁二叉树、清空二叉树、判定空二叉树、求二叉树深度等 12 种基本运算，具体运算功能定义如下。

### 2.1 问题描述

要求构造一个具有菜单的功能演示系统。该演示系统实现二叉树管理。其中，在主函数中准备函数调用所需实参值、显示函数执行结果，并给出适当的操作提示。

实现依据最小完备性和常用性相结合的原则确定的创建二叉树、销毁二叉树、清空二叉树、判定空二叉树、求二叉树深度等 12 种基本运算，具体运算功能定义如下。

1. 创建二叉树：函数名称是 `CreateBiTree(T,definition)`；初始条件是 `T` 不存在；`definition` 给出二叉树 `T` 的定义，如带空子树的二叉树前序遍历序列、或前序 + 中序、或后序 + 中序；操作结果是按 `definition` 构造二叉树 `T`；（要求 `T` 中各结点关键字具有唯一性）
2. 销毁二叉树：函数名称是 `DestroyBiTree(T)`；初始条件是二叉树 `T` 已存在；操作结果是销毁二叉树 `T`；
3. 清空二叉树：函数名称是 `ClearBiTree(T)`；初始条件是二叉树 `T` 存在；操作结果是将二叉树 `T` 清空；
4. 判定空二叉树：函数名称是 `BiTreeEmpty(T)`；初始条件是二叉树 `T` 存在；操作结果是若 `T` 为空二叉树则返回 `TRUE`，否则返回 `FALSE`；
5. 求二叉树深度：函数名称是 `BiTreeDepth(T)`；初始条件是二叉树 `T` 存在；操作结果是返回 `T` 的深度；
6. 查找结点：函数名称是 `LocateNode(T,e)`；初始条件是二叉树 `T` 已存在，`e` 是和 `T` 中结点关键字类型相同的给定值；操作结果是返回查找到的结点指针，如无关键字为 `e` 的结点，返回 `NULL`；

7. 结点赋值: 函数名称是**Assign(T,e,value)**; 初始条件是二叉树T已存在, e是和T中结点关键字类型相同的给定值; 操作结果是关键字为e的结点赋值为value;
8. 获得兄弟结点: 函数名称是**GetSibling(T,e)**; 初始条件是二叉树T存在, e是和T中结点关键字类型相同的给定值; 操作结果是返回关键字为e的结点的(左或右)兄弟结点指针。若关键字为e的结点无兄弟, 则返回NULL;
9. 插入结点: 函数名称是**InsertNode(T,e,LR,c)**; 初始条件是二叉树T存在, e是和T中结点关键字类型相同的给定值, LR为0或1, c是待插入结点; 操作结果是根据LR为0或者1, 插入结点c到T中, 作为关键字为e的结点的左或右孩子结点, 结点e的原有左子树或右子树则为结点c的右子树(LR为-1时, 作为根结点插入, 原根结点作为c的右子树);
10. 删除结点: 函数名称是**DeleteNode(T,e)**; 初始条件是二叉树T存在, e是和T中结点关键字类型相同的给定值。操作结果是删除T中关键字为e的结点; 同时, 如果关键字为e的结点度为0, 删除即可; 如关键字为e的结点度为1, 用关键字为e的结点孩子代替被删除的e位置; 如关键字为e的结点度为2, 用e的左孩子代替被删除的e位置, e的右子树作为e的左子树中最右结点的右子树;
11. 前序遍历: 函数名称是**PreOrderTraverse(T,Visit)**; 初始条件是二叉树T存在, Visit是一个函数指针的形参(可使用该函数对结点操作); 操作结果: 先序遍历, 对每个结点调用函数Visit一次且一次, 一旦调用失败, 则操作失败。
12. 中序遍历: 函数名称是**InOrderTraverse(T,Visit)**; 初始条件是二叉树T存在, Visit是一个函数指针的形参(可使用该函数对结点操作); 操作结果是中序遍历T, 对每个结点调用函数Visit一次且一次, 一旦调用失败, 则操作失败;
13. 后序遍历: 函数名称是**PostOrderTraverse(T,Visit)**; 初始条件是二叉树T存在, Visit是一个函数指针的形参(可使用该函数对结点操作); 操作结果是后序遍历T, 对每个结点调用函数Visit一次且一次, 一旦调用失败, 则操作失败。(注: 前序、中序和后序三种遍历算法, 要求至少一个用非递归算法实现)
14. 按层遍历: 函数名称是**LevelOrderTraverse(T,Visit)**; 初始条件是二叉

树T存在，Visit是对结点操作的应用函数；操作结果是层序遍历T，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

此外，还在基础功能的基础上实现了附加功能：

1. 最大路径和：函数名称是MaxPathSum(T)，初始条件是二叉树T存在；操作结果是返回根结点到叶子结点的最大路径和(以结点关键字为权重)；
2. 最近公共祖先：函数名称是LowestCommonAncestor(T, e1, e2)；初始条件是二叉树T存在；操作结果是该二叉树中e1结点和e2结点的最近公共祖先；
3. 翻转二叉树：函数名称是InvertTree(T)，初始条件是线性表L已存在；操作结果是将T翻转，使其所有结点的左右结点互换；
4. 实现二叉树的文件形式保存，其中：
  - 需要设计文件数据记录格式，以高效保存二叉树数据的完整信息；
  - 需要设计二叉树文件保存和加载操作合理模式。
5. 实现多个二叉树：创建、添加、移除二叉树。

## 2.2 系统设计

1. 函数名称：CreateBiTree(L, definition)；  
初始条件：二叉树T不存在；  
操作结果：按照定义前序遍历序列 definition 构造二叉树T；  
设计思路：将CreateBiTree() 作为一个递归函数  
`RecurvalCreateBiTree(definition[], start)`  
的入口；由RecurvalCreateBiTree() 递归构造左右子树。
2. 函数名称：DestroyBiTree(T)；  
初始条件：二叉树T存在；  
操作结果：销毁二叉树T；  
设计思路：释放根结点内存后，调用DestroyBiTree() 销毁左右子树，当参数为空树时返回。
3. 函数名称：ClearBiTree(T)；  
初始条件：二叉树T存在；  
操作结果：将二叉树的结点的所有数据域中的数据删除；  
设计思路：清空根结点后，递归清空左右子树，当参数为空树时返回。

4. 函数名称: BiTreeEmpty;

初始条件: 二叉树存在;

操作结果: 若T为空二叉树则返回TRUE, 否则返回FALSE;

设计思路: 如果T是空指针则返回TRUE, 否则返回FALSE;

5. 函数名称: BiTreeDepth(T);

初始条件: 二叉树T存在;

操作结果: 返回二叉树的深度;

设计思路: 递归求左右子树的深度, 求它们的最大值D, 返回D+1;

6. 函数名称: LocateNode(T,e);

初始条件: 二叉树T存在;

操作结果: 返回查找到的结点指针, 如果无关键字为e的结点返回 NULL;

设计思路: 先查找根结点, 再递归查找左右子树, 返回查找到的指针值, 如果参数是空树返回 NULL;

7. 函数名称: Assign(T,e,value);

初始条件: 二叉树T存在;

操作结果: 关键字为 e 的结点赋值为 value;

设计思路: 调用函数LocateNode(T,e)查找到关键字为e的结点, 再赋值为value;

8. 函数名称: GetSibling(T,e);

初始条件: 二叉树T存在;

操作结果: 返回关键字为e的结点的兄弟结点指针, 如果关键字为e的结点无兄弟, 则返回NULL;

设计思路: 编写函数LocateParents(T,e), 该函数返回关键字为e的双亲结点指针, 再由双亲结点找到关键字为e结点的兄弟结点, 并返回指向兄弟结点的指针值;

9. 函数名称: InsertNode(T,e,LR,c);

初始条件: 二叉树T存在;

操作结果: 根据LR为0或者1, 插入结点c到T中, 作为关键字为e的结点的左或右孩子结点, 结点e的原有左子树或右子树则为结点c的右子树, LR为-1时, 作为根结点插入, 原根结点作为c的右子树。

设计思路: 先调用LocateNode找到关键字为e的结点, 再按要求插入结点。

10. 函数名称: DeleteNode(T,e);

初始条件：二叉树T存在；

操作结果：操作结果是删除T中关键字为e的结点；同时，如果关键字为e的结点度为0，即删除，如关键字为e的结点度为1，用关键字为e的结点的孩子代替被删除的e位置；如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为e的左子树中最右结点的右子树；

设计思路：先调用LocateNode找到关键字为e的结点，再按要求删除结点。

11. 函数名称：PreOrderTraverse(T)；

初始条件：二叉树T存在；

操作结果：按前序遍历序列依次调用visit()函数访问树上的结点；

设计思路：先访问根结点，再前序遍历左子树，再前序遍历右子树；

12. 函数名称：InOrderTraverse(T)；

初始条件：二叉树T存在；

操作结果：按中序遍历序列依次调用visit()函数访问树上的结点；

设计思路：先中序遍历左子树，再访问根结点，再中序遍历右子树（非递归实现，用一个栈确定访问的顺序）；

13. 函数名称：PostOrderTraverse(T)；

初始条件：二叉树T存在；

操作结果：按后序遍历序列依次调用visit()函数访问树上的结点；

设计思路：先后序遍历左子树，再后序遍历右子树，再访问根结点；

14. 函数名称：LevelOrderTraverse(L)；

初始条件：二叉树T存在；

操作结果：按层次顺序调用visit()函数访问树上的结点；

设计思路：用一个队列保存被访问结点的子结点来实现按层遍历；

15. 函数名称：MaxPathSum(T)；

初始条件：二叉树T存在；

操作结果：返回最大路径和；

设计思路：对左、右子树递归调用MaxPathSum()，求其最大值D，返回D+1；

16. 函数名称：LowestCommonAncestor(T,e1,e2)；

初始条件：二叉树T存在；

操作结果：返回最近的公共祖先；

设计思路：迭代调用LocateParents上溯，找到e1、e2到根结点的路径交汇

# 华 中 科 技 大 学 课 程 实 验 报 告

---

处；

17. 函数名称: InvertTree;

初始条件: 二叉树T存在;

操作结果: 将二叉树的所有结点左右子树交换;

设计思路: 先交换根结点左右孩子, 再递归翻转左子树、右子树;

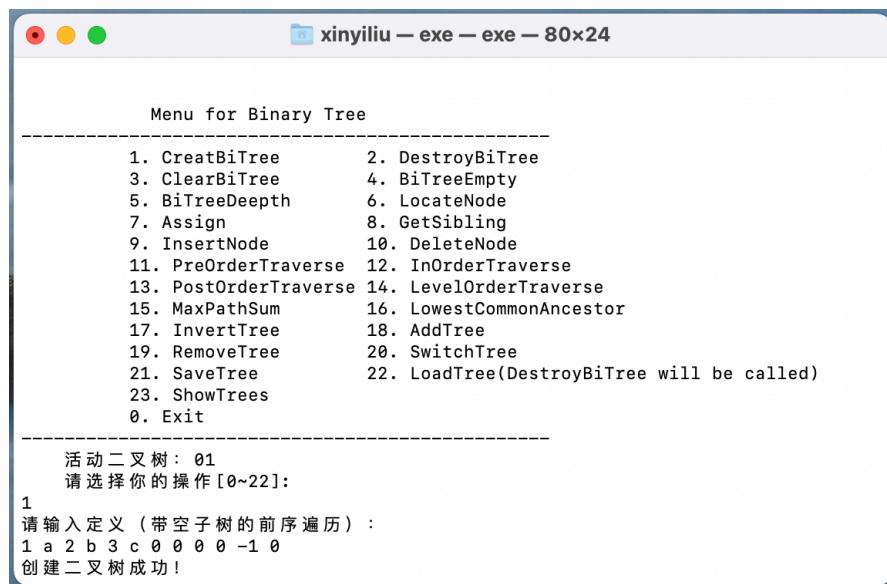
## 2.3 系统实现

见《附录 C 基于二叉链表二叉树实现的源程序》。

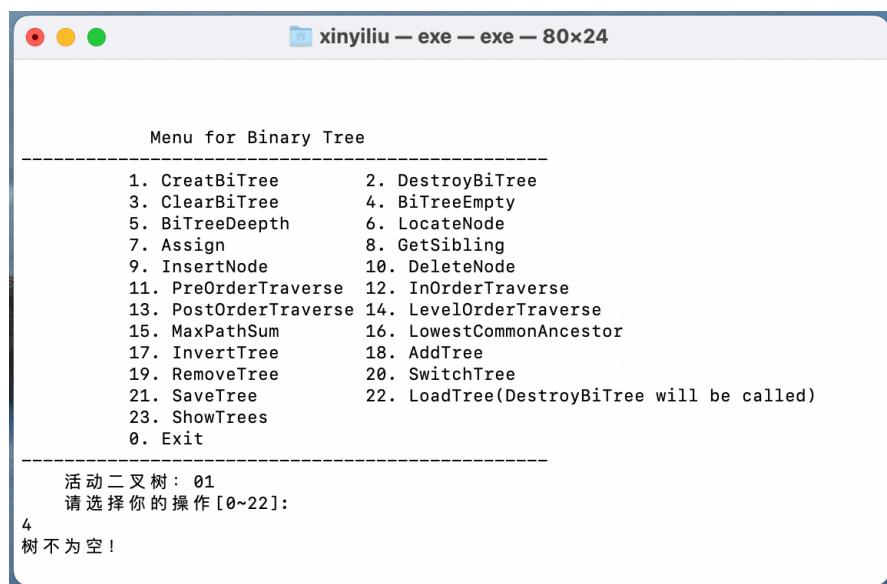
## 2.4 系统测试

先插入结点再依次测试各项功能；下面给出了基础功能的测试截图，测试是按图片的顺序进行的。

### 1. CreateBiTree()



### 2. BiTreeEmpty()



# 华中科技大学课程实验报告

## 3. LocateNode()

xinyiliu — exe — exe — 80x24

Menu for Binary Tree

---

1. CreatBiTree	2. DestroyBiTree
3. ClearBiTree	4. BiTreeEmpty
5. BiTreeDepth	6. LocateNode
7. Assign	8. GetSibling
9. InsertNode	10. DeleteNode
11. PreOrderTraverse	12. InOrderTraverse
13. PostOrderTraverse	14. LevelOrderTraverse
15. MaxPathSum	16. LowestCommonAncestor
17. InvertTree	18. AddTree
19. RemoveTree	20. SwitchTree
21. SaveTree	22. LoadTree(DestroyBiTree will be called)
23. ShowTrees	
0. Exit	

---

活动二叉树： 01  
请选择你的操作 [0~22]:

6  
请输入要查找的关键字：  
3  
结点内容： key: 3 , others : c .

## 4. Assign()

xinyiliu — exe — exe — 80x27

Menu for Binary Tree

---

1. CreatBiTree	2. DestroyBiTree
3. ClearBiTree	4. BiTreeEmpty
5. BiTreeDepth	6. LocateNode
7. Assign	8. GetSibling
9. InsertNode	10. DeleteNode
11. PreOrderTraverse	12. InOrderTraverse
13. PostOrderTraverse	14. LevelOrderTraverse
15. MaxPathSum	16. LowestCommonAncestor
17. InvertTree	18. AddTree
19. RemoveTree	20. SwitchTree
21. SaveTree	22. LoadTree(DestroyBiTree will be called)
23. ShowTrees	
0. Exit	

---

活动二叉树： 01  
请选择你的操作 [0~22]:

7  
请输入要赋值的结点的关键字：  
3  
请输入要赋的关键字的值：  
4  
请输入要赋的 others 值：  
d  
赋值成功！

# 华中科技大学课程实验报告

## 5. GetSibling()

```
xinyiliu — exe — exe — 80x27

Menu for Binary Tree
-----
1. CreatBiTree      2. DestroyBiTree
3. ClearBiTree     4. BiTreeEmpty
5. BiTreeDepth      6. LocateNode
7. Assign           8. GetSibling
9. InsertNode       10. DeleteNode
11. PreOrderTraverse 12. InOrderTraverse
13. PostOrderTraverse 14. LevelOrderTraverse
15. MaxPathSum      16. LowestCommonAncestor
17. InvertTree       18. AddTree
19. RemoveTree       20. SwitchTree
21. SaveTree         22. LoadTree(DestroyBiTree will be called)
23. ShowTrees
0. Exit

活动二叉树： 01
请选择你的操作 [0~22]：
8
请输入目标关键字：
4
兄弟结点： key : 2 ; others: b .
```

## 6. InsertNode()

```
xinyiliu — exe — exe — 80x32

Menu for Binary Tree
-----
1. CreatBiTree      2. DestroyBiTree
3. ClearBiTree     4. BiTreeEmpty
5. BiTreeDepth      6. LocateNode
7. Assign           8. GetSibling
9. InsertNode       10. DeleteNode
11. PreOrderTraverse 12. InOrderTraverse
13. PostOrderTraverse 14. LevelOrderTraverse
15. MaxPathSum      16. LowestCommonAncestor
17. InvertTree       18. AddTree
19. RemoveTree       20. SwitchTree
21. SaveTree         22. LoadTree(DestroyBiTree will be called)
23. ShowTrees
0. Exit

活动二叉树： 01
请选择你的操作 [0~22]：
9
请输入目标关键字：
4
0表示作为左孩子插入，1表示作为右孩子插入，-1代表作为根结点插入，请输入(0 or 1)
:
0
请输入插入的结点的关键字、数据内容：
5
f
在关键字为 4 的结点插入关键字为 5 数据内容为 f 的
结点作为左孩子成功！
```

# 华中科技大学课程实验报告

## 7. DeleteNode()

xinyiliu — exe — exe — 80x27

在关键字为 4 的结点插入关键字为 5 数据内容为 f 的  
结点作为左孩子成功！

Menu for Binary Tree

1. CreatBiTree      2. DestroyBiTree  
3. ClearBiTree      4. BiTreeEmpty  
5. BiTreeDepth      6. LocateNode  
7. Assign      8. GetSibling  
9. InsertNode      10. DeleteNode  
11. PreOrderTraverse      12. InOrderTraverse  
13. PostOrderTraverse      14. LevelOrderTraverse  
15. MaxPathSum      16. LowestCommonAncestor  
17. InvertTree      18. AddTree  
19. RemoveTree      20. SwitchTree  
21. SaveTree      22. LoadTree(DestroyBiTree will be called)  
23. ShowTrees  
0. Exit

活动二叉树： 01  
请选择你的操作 [0~22]:  
10  
请输入目标关键字：  
5  
删除成功！

## 8. PreOrderTraverse()

xinyiliu — exe — exe — 80x27

删除成功！

Menu for Binary Tree

1. CreatBiTree      2. DestroyBiTree  
3. ClearBiTree      4. BiTreeEmpty  
5. BiTreeDepth      6. LocateNode  
7. Assign      8. GetSibling  
9. InsertNode      10. DeleteNode  
11. PreOrderTraverse      12. InOrderTraverse  
13. PostOrderTraverse      14. LevelOrderTraverse  
15. MaxPathSum      16. LowestCommonAncestor  
17. InvertTree      18. AddTree  
19. RemoveTree      20. SwitchTree  
21. SaveTree      22. LoadTree(DestroyBiTree will be called)  
23. ShowTrees  
0. Exit

活动二叉树： 01  
请选择你的操作 [0~22]:  
11  
1 a 2 b 4 d

# 华中科技大学课程实验报告

## 9. InOrderTraverse()

xinyiliu — exe — exe — 80x27  
1 a 2 b 4 d

Menu for Binary Tree

---

1. CreatBiTree	2. DestroyBiTree
3. ClearBiTree	4. BiTreeEmpty
5. BiTreeDepth	6. LocateNode
7. Assign	8. GetSibling
9. InsertNode	10. DeleteNode
11. PreOrderTraverse	12. InOrderTraverse
13. PostOrderTraverse	14. LevelOrderTraverse
15. MaxPathSum	16. LowestCommonAncestor
17. InvertTree	18. AddTree
19. RemoveTree	20. SwitchTree
21. SaveTree	22. LoadTree(DestroyBiTree will be called)
23. ShowTrees	
0. Exit	

---

活动二叉树： 01  
请选择你的操作 [0~22]:  
12  
2 b 1 a 4 d

## 10. PostOrderTraverse()

xinyiliu — exe — exe — 80x27  
2 b 1 a 4 d

Menu for Binary Tree

---

1. CreatBiTree	2. DestroyBiTree
3. ClearBiTree	4. BiTreeEmpty
5. BiTreeDepth	6. LocateNode
7. Assign	8. GetSibling
9. InsertNode	10. DeleteNode
11. PreOrderTraverse	12. InOrderTraverse
13. PostOrderTraverse	14. LevelOrderTraverse
15. MaxPathSum	16. LowestCommonAncestor
17. InvertTree	18. AddTree
19. RemoveTree	20. SwitchTree
21. SaveTree	22. LoadTree(DestroyBiTree will be called)
23. ShowTrees	
0. Exit	

---

活动二叉树： 01  
请选择你的操作 [0~22]:  
13  
2 b 4 d 1 a

# 华中科技大学课程实验报告

## 11. LevelOrderTraverse()



### 3 实验总结

## 4 附录 A 基于顺序存储结构线性表实现的源程序

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <limits.h>
5 #define TRUE 1
6 #define FALSE 0
7 #define OK 1
8 #define ERROR 0
9 #define INFEASIBLE -1
10 #define OVERFLOW -2
11 #define ISEMPY -3
12 #define LIST_INIT_SIZE 1000
13 #define LISTINCREMENT 10
14 #define max(i,j) ((i)>(j)?(i):(j))
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36

```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define ISEMPY -3
#define LIST_INIT_SIZE 1000
#define LISTINCREMENT 10
#define max(i,j) ((i)>(j)?(i):(j))

typedef int status;
typedef int ElemType; // 数据元素类型定义
typedef struct{ // 顺序表（顺序结构）的定义
    ElemType * elem;
    int length;
    int listszie;
} SqList;

typedef struct THELISTS{ // 顺序表的管理表定义
    struct ALIST{
        char name[30];
        SqList L;
    } elem[50];
    int length = 0;
    int listssize = 50;
} LISTS;

/*
    初始化顺序表，如果分配空间失败，返回OVERFLOW；否则，返回OK。
*/
status InitList(SqList& L){

```

# 华中科技大学课程实验报告

```
if(L.elem!=NULL) return INFEASIBLE;          37
L.elem = (ElemType *)malloc( LIST_INIT_SIZE * sizeof (ElemType));
if(!L.elem) exit(OVERFLOW);                  38
L.length=0;                                39
L.listsize=LIST_INIT_SIZE;                  40
return OK;                                  41
}

/*
如果顺序表L存在，销毁顺序表L，释放数据元素的空间，返回OK，否则返回INFEASIBLE。
*/
status DestroyList(SqList& L)              42
{
    if(L.elem){
        free(L.elem);
        L.elem = NULL;
        L.length = 0;
        L.listsize = 0;
        return OK;
    }else return INFEASIBLE;
}

/*
如果顺序表L存在，删除顺序表L中的所有元素，返回OK，否则返回INFEASIBLE。
*/
status ClearList(SqList& L)                58
{
    if(L.elem){
        L.length = 0;
        return OK;
    }else return INFEASIBLE;
}

/*
如果顺序表L存在，判断顺序表L是否为空，空就返回TRUE，否则返回FALSE；
如果顺序表L不存在，返回INFEASIBLE。
*/
status ListEmpty(SqList L) {                 70
    if(L.elem){
        if(L.length == 0) return TRUE;
        else return FALSE;
    }
}
```

# 华中科技大学课程实验报告

```
    }
    else return INFEASIBLE;
}

}

/*
如果顺序表L存在，返回顺序表L的长度，否则返回INFEASIBLE。
*/
int ListLength(SqList L)
{
    if(L.elem){
        return L.length;
    }
    else return INFEASIBLE;
}

/*
如果顺序表L存在，获取顺序表L的第i个元素，保存在e中，返回OK；
如果i不合法，返回ERROR；如果顺序表L不存在，返回INFEASIBLE。
*/
status GetElem(SqList L,int i,ElemType &e)
{
    if(L.elem){
        if(i > L.length || i<=0) return ERROR;
        e = L.elem[i-1];
        return OK;
    }
    else return INFEASIBLE;
}

/*
如果顺序表L存在，查找元素e在顺序表L中的位置序号并返回该序号；
如果e不存在，返回0；当顺序表L不存在时，返回INFEASIBLE（即-1）。
*/
int LocateElem(SqList L,ElemType e,int (*compare)(ElemType,ElemType))
{
    if(L.elem){
        for(int i=1;i<=L.length;i++)
            if(compare(L.elem[i-1],e)) return i;
    }
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
    return ERROR;
}
else return INFEASIBLE;
}

/*
如果顺序表L存在，获取顺序表L中元素e的前驱，保存在pre中，返回OK；
如果没有前驱，返回ERROR；如果顺序表L不存在，返回INFEASIBLE。
*/
status PriorElem(SqList L,ElemType e,ElemType &pre)
{
    if(L.elem){
        if(L.elem[0]==e) return ERROR;
        for(int i=2;i<=L.length;i++)
            if(L.elem[i-1]==e){
                pre = L.elem[i-2];
                return OK;
            }
        return ERROR;
    }
    else return INFEASIBLE;
}

/*
如果顺序表L存在，获取顺序表L元素e的后继，保存在next中，返回OK；
如果没有后继，返回ERROR；如果顺序表L不存在，返回INFEASIBLE。
*/
status NextElem(SqList L,ElemType e,ElemType &next)
{
    if(L.elem){
        for(int i=1;i<=L.length;i++)
            if(L.elem[i-1]==e){
                if(i==L.length) return ERROR;
                next = L.elem[i];
                return OK;
            }
        return ERROR;
    }
    else return INFEASIBLE;
}
```

# 华中科技大学课程实验报告

```
/*
如果顺序表L存在，将元素e插入到顺序表L的第i个元素之前，返回OK;
当插入位置不正确时，返回ERROR; 如果顺序表L不存在，返回INFEASIBLE.
*/
status ListInsert(SqList &L,int i,ElemType e)
{
    if(L.elem){
        if(i <= 0 || i > L.length) return ERROR;
        if(L.length + 1 > L.listsize){
            L.elem = (ElemType*)realloc(L.elem, sizeof(int)*2*L.listsize);
            L.listsize <= 1;
        }
        int j;
        for(j = L.length; j >= i; j--)
            L.elem[j] = L.elem[j-1];
        L.elem[j] = e;
        L.length++;
        return OK;
    }
    else return INFEASIBLE;
}

/*
如果顺序表L存在，删除顺序表L的第i个元素，并保存在e中，返回OK;
当删除位置不正确时，返回ERROR; 如果顺序表L不存在，返回INFEASIBLE.
*/
status ListDelete(SqList &L,int i,ElemType &e)
{
    if (L.length==0) {
        return ISEMPTY;
    }
    if(L.elem){
        if(i <= 0 || i > L.length) return ERROR;
        int j;
        e = L.elem[i-1];
        for(j = i - 1; j < L.length - 1; j++)
            L.elem[j] = L.elem[j+1];
        L.length--;
        return OK;
    }
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
}

else return INFEASIBLE;

}

/*
如果顺序表L存在，依次显示顺序表中的元素，每个元素间空一格，返回OK;
如果顺序表L不存在，返回INFEASIBLE.

*/
status ListTraverse(SqList L, int (*visit)(ElemType))
{
    if(L.length==0){211
        printf("顺序表是空表! ");
        return OK;
    }212
    if(L.elem){213
        printf("\n-----all elements -----\\n");
        for(int j = 0; j < L.length; j++) {217
            visit(L.elem[j]);
        }218
        printf("\n-----end -----\\n");
        return OK;
    }219
    else return INFEASIBLE;
}222
}225

/*
LocateElem()使用的compare()函数
*/
int ElemEqual(ElemType i, ElemType j){
    return i==j;
}231
}232

/*
ListTraverse()使用的visit()函数
*/
int TraversePrint(ElemType e){236
    printf("%d ", e);
    return 0;
}238
}239

/*
240
241
```

# 华中科技大学课程实验报告

```
判断顺序表是否存在，存在返回0，不存在返回1 242
*/
int NonExist(SqList &L){ 243
    return L.elem != NULL ? 0 : 1; 244
}

/*
最大连续子数组和 245
*/
long long int MaxSubArray(SqList &L){ 246
    ElemType *arr = L.elem; 247
    long long maxSum = LLONG_MIN; 248
    for (int i = 0; i < L.length; i++) { // 枚举左端点 249
        long long sum = 0; 250
        for (int j = i; j<L.length; j++) { // 枚举右端点 251
            sum += arr[j]; 252
            maxSum = max(maxSum,sum); 253
        }
    }
    return maxSum;
} 254

/*
求和为K的子数组个数。 255
*/
long long int SubArrayNum(SqList &L, long long k){ 256
    ElemType *arr = L.elem; 257
    long long cnt = 0; 258
    for (int i = 0; i < L.length; i++) { // 枚举左端点 259
        long long sum = 0; 260
        for (int j = i; j<L.length; j++) { // 枚举右端点 261
            sum += arr[j];
            if(sum == k) cnt++;
        }
    }
    return cnt;
} 262

/*
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
*/
```

# 华中科技大学课程实验报告

```
顺序表排序。 283
*/
status SortList(SqList &L){ 284
    if(L.elem){ 285
        if(L.length==0) return OK; 286
        for(int i = 0; i<L.length; i++) 287
            for (int j = 0; j<L.length-i-1; j++) { 288
                if(L.elem[j]>L.elem[j+1]) 289
                {
                    ElemType tmp = L.elem[j]; 290
                    L.elem[j] = L.elem[j+1]; 291
                    L.elem[j+1] = tmp; 292
                }
            }
        return OK; 293
    } 294
    else return INFEASIBLE; 295
} 296
297
/*
如果顺序表L存在，将顺序表L的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。 303
*/
status SaveList(SqList L,char FileName[])
{
    if(L.elem) { 308
        FILE* fp; 309
        if((fp = fopen(FileName, "w"))== NULL){ 310
            printf("打开文件失败"); 311
            return ERROR; 312
        }
        fprintf(fp, "%d ", L.length); 313
        fprintf(fp, "%d ", L.listsize); 314
        for(int i=0; i<L.length; i++) { 315
            fprintf(fp, " %d", L.elem[i]); 316
        }
        fclose(fp); 317
        return OK; 318
    } else return INFEASIBLE; 319
} 320
321
322
323
```

# 华中科技大学课程实验报告

```
/*
如果顺序表L不存在，将FileName文件中的数据读入到顺序表L中，返回OK，否则返回INFEASIBLE
.

*/
status LoadList(SqList &L, char FileName[]){
    if(!L.elem) {
        L.elem = (ElemType*)malloc(LIST_INIT_SIZE*sizeof(ElemType));
        L.length = 0;
        L.listsize = LIST_INIT_SIZE;
        FILE* fp = fopen(FileName, "r");
        if(fp == NULL){
            printf("打开文件失败");
            return ERROR;
        }
        int i = 0;
        fscanf(fp, "%d%d", &L.length);
        while(i < L.length && fscanf(fp, "%d", &L.elem[i++])==1) {
            continue;
        }
        fclose(fp);
        return OK;
    }
    else return INFEASIBLE;
}

/*
Lists中删除一个名称为ListName的顺序表
*/
status RemoveList(LISTS &Lists,char ListName[])
{
    int flag = 0;
    for(int i=0; i<Lists.length; i++){
        if(strcmp(ListName, Lists.elem[i].name) == 0){
            flag = 1;
            for(int j=i; j<Lists.length-1; j++){
                Lists.elem[j] = Lists.elem[j+1];
            }
            Lists.length--;
        }
    }
}
```

# 华中科技大学课程实验报告

```
if(flag == 0) return ERROR; 364
else return OK; 365
}

366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404

/*
Lists中添加一个名称为ListName的顺序表
*/
status AddList(LISTS &Lists,char ListName[])
{
    if(Lists.length == Lists.listssize) {
        printf("顺序表数目达到最大值!\n");
        return ERROR;
    }

    THELISTS::ALIST & list = Lists.elem[Lists.length];
    strcpy(list.name, ListName);

    list.L.elem = (ElemType*) malloc(LIST_INIT_SIZE*sizeof(ElemType));
    if(list.L.elem==NULL) return ERROR;

    list.L.length = 0;
    list.L.listsize = LIST_INIT_SIZE;
    Lists.length++;
    return OK;
}

/*
Lists中查找一个名称为ListName的顺序表
*/
int LocateList(LISTS Lists,char ListName[])
{
    for(int i=0; i<Lists.length; i++) {
        if(strcmp(Lists.elem[i].name, ListName)==0) return i + 1;
    }
    return 0;
}

/*
主函数
*/
int main(void){
```

# 华中科技大学课程实验报告

```
int op=20; 405
LISTS Lists; 406
int idt = 1; 407
for(int i = 0;i<Lists.length;i++){
    Lists.elem[i].L.elem = NULL; 408
    Lists.elem[i].L.length = 0; 409
}
strcpy(Lists.elem[0].name, "default = [0]"); 410
Lists.elem[0].L.elem = NULL; 411
SqList *PtrL = &Lists.elem[0].L; 412
while(op){ 413
    SqList & L = *PtrL; 414
    printf("\n\n"); 415
    printf("      Now working on List %s \n", Lists.elem[idt-1].name); 416
    printf("      Menu for Linear Table On Sequence Structure \n"); 417
    printf("-----\n"); 418
    printf("          1. InitList      7. LocateElem\n"); 419
    printf("          2. DestroyList   8. PriorElem\n"); 420
    printf("          3. ClearList     9. NextElem \n"); 421
    printf("          4. ListEmpty     10. ListInsert\n"); 422
    printf("          5. ListLength    11. ListDelete\n"); 423
    printf("          6. GetElem       12. ListTraverse\n"); 424
    printf("          13. MaxSubArray  14. SubArrayNum \n"); 425
    printf("          15. SortList     16. SaveList\n"); 426
    printf("          17. LoadList     18. AddList\n"); 427
    printf("          19. RemoveList   20. SwitchList\n"); 428
    printf("          21. ShowLists    0. Exit\n"); 429
    printf("-----\n"); 430
    printf("      请选择你的操作[0~21]:"); 431
    scanf("%d",&op); 432
    switch(op){
        case 1: 433
            if(InitList(L)==OK) printf("顺序表创建成功! \n");
            else printf("顺序表创建失败! 顺序表已经存在。 \n");
            getchar();getchar();
            break; 434
        case 2: 435
            if(DestroyList(L)==OK) printf("顺序表删除成功! \n");
            else printf("顺序表删除失败! 顺序表不存在。 \n");
            getchar();getchar();
            break; 436
    }
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
case 3: 446
    if(ClearList(L)==OK) printf("顺序表清空成功! \n");
    else printf("顺序表清空失败! 顺序表不存在。 \n");
    getchar();getchar();
    break; 450
case 4: 451
    int empty;
    if((empty = ListEmpty(L))!=INFEASIBLE) 452
        if(empty) printf("顺序表为空! \n");
        else printf("顺序表不为空! \n");
    else printf("顺序表判空失败! 顺序表不存在。 \n");
    getchar();getchar();
    break; 456
case 5: 459
    int len;
    if((len = ListLength(L))!=INFEASIBLE) 460
        printf("顺序表长度为: %d 。 \n", len);
    else printf("顺序表求长度失败, 顺序表不存在! \n");
    getchar();getchar();
    break; 464
case 6: 466
{
    int i;
    ElemType e;
    printf("请输入查找的元素的逻辑序号 (从1开始) : ");
    scanf("%d",&i);
    if(GetElem(L, i, e)!=INFEASIBLE) printf("顺序表的第 %d 个元素为 %d 。 \n"
        ", i, e);
    else printf("顺序表查找失败! 顺序表不存在。 \n");
    getchar();getchar();
} 475
break; 476
case 7: 477
{
    ElemType e;
    int i;
    printf("请输入查找的目标元素: ");
    scanf("%d",&e);
    switch(i=LocateElem(L, e, &ElemEqual)){
        case INFEASIBLE: 483
            printf("查找元素失败, 顺序表不存在! \n");
} 485
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
        break; 486
    case ERROR: 487
        printf("查找元素失败，顺序表中不存在该元素！\n"); 488
        break; 489
    default: 490
        printf("元素 %d 第一次出现在第 %d 个元素上。\\n", e, i); 491
        break; 492
    }
    getchar();getchar(); 493
    break; 494
}
case 8: 495
{
    ElemType e,pre; 499
    printf("请输入要查找前驱的元素："); 500
    scanf("%d",&e); 501
    switch(PriorElem(L, e, pre)){
        case OK: 502
            printf("元素 %d 的前驱是 %d 。\\n",e,pre); 504
            break; 505
        case ERROR: 506
            printf("查找前驱元素失败，该元素不存在，或无前驱！ \\n"); 507
            break; 508
        case INFEASIBLE: 509
            printf("查找前驱元素失败，顺序表不存在！ \\n"); 510
            break; 511
        default: 512
            break; 513
    }
    getchar();getchar(); 514
    break; 515
}
case 9: 517
{
    ElemType e,ne; 520
    printf("请输入要查找后继的元素："); 521
    scanf("%d",&e); 522
    switch(NextElem(L, e, ne)){
        case OK: 523
            printf("元素 %d 的后继是 %d 。\\n",e,ne); 525
            break; 526
    }
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
case ERROR:  
    printf("查找后继元素失败, 该元素不存在, 或无后继! \n");  
    break;  
case INFEASIBLE:  
    printf("查找后继元素失败, 顺序表不存在! \n");  
    break;  
default:  
    break;  
}  
getchar();getchar();  
break;  
}  
case 10:  
{  
    ELEMType e;  
    int j;  
    printf("请输入插入的元素的逻辑序号 (从1开始) : ");  
    scanf("%d",&j);  
    printf("请输入插入的元素的值 : ");  
    scanf("%d",&e);  
    switch(ListInsert(L, j, e)){  
        case OK:  
            printf("顺序表插入成功! \n");  
            break;  
        case ERROR:  
            printf("顺序表插入失败, 插入位置非法! \n");  
            break;  
        case INFEASIBLE:  
            printf("顺序表插入失败, 顺序表不存在! ");  
            break;  
        default:  
            break;  
    }  
    getchar();getchar();  
    break;  
}  
case 11:{  
    int i;  
    ELEMType e;  
    printf("请输入要删除的元素的逻辑序号: ");  
    scanf("%d",&i);  
    527  
    528  
    529  
    530  
    531  
    532  
    533  
    534  
    535  
    536  
    537  
    538  
    539  
    540  
    541  
    542  
    543  
    544  
    545  
    546  
    547  
    548  
    549  
    550  
    551  
    552  
    553  
    554  
    555  
    556  
    557  
    558  
    559  
    560  
    561  
    562  
    563  
    564  
    565  
    566  
    567
```

# 华中科技大学课程实验报告

```
switch(ListDelete(L, i, e)){
    case OK:
        printf("被删除的元素是 %d 。\n", e);
        break;
    case ERROR:
        printf("删除元素失败，指定的逻辑序号不合法！\n");
        break;
    case INFEASIBLE:
        printf("删除元素失败，顺序表不存在！\n");
        break;
    case ISEMPY:
        printf("删除元素失败，顺序表是空表！\n");
        break;
    default:
        break;
}
getchar();getchar();
break;
}
case 12:
{
if(ListTraverse(L,&TraversePrint)==OK)
    printf("\n遍历成功！\n");
else
    printf("遍历失败，顺序表不存在！\n");
getchar();getchar();
break;
}
case 13:
{
if(ListEmpty(L)==TRUE)
{
    printf("操作失败，顺序表为空！\n");
    getchar();getchar();
    break;
}
if(NonExist(L)){
    printf("操作失败，线性表不存在！\n");
    getchar();getchar();
    break;
}
printf("最大子数组和为 %lld 。", MaxSubArray(L));
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
        getchar();getchar();
        break;
    }
    case 14:
    {
        if(ListEmpty(L)==TRUE)
        {
            printf("操作失败，顺序表为空！\n");
            getchar();getchar();
            break;
        }
        if(NonExist(L)){
            printf("操作失败，线性表不存在！\n");
            getchar();getchar();
            break;
        }
        long long int k;
        printf("请输入子数组和K: ");
        scanf("%lld",&k);
        printf("子数组和为 %lld 的个数为 %lld 。", k, SubArrayNum(L,k));
        getchar();getchar();
        break;
    }
    case 15:
    {
        if (SortList(L)!=INFEASIBLE) {
            if(ListEmpty(L)) printf("顺序表是空表。\\n");
            printf("排序操作成功！\\n");
        }
        else printf("操作失败，顺序表不存在！\\n");
        getchar();getchar();
        break;
    }
    case 16:
    {
        char FileName[128];
        printf("请输入你要保存的文件名：\\n");
        scanf("%s",FileName);
        switch(SaveList(L, FileName))
        {
            case OK:
```

# 华中科技大学课程实验报告

```
    printf("保存顺序表到文件操作成功! \n");
    break;
case ERROR:
    printf("操作失败, 请检查文件名。 \n");
    break;
case INFEASIBLE:
    printf("操作失败, 顺序表不存在! \n");
    break;
}
getchar();getchar();
break;
}
case 17:
{
    char FileName[128];
    char ListName[30];
    printf("请输入你要读取的文件名: \n");
    scanf("%s",FileName);
    printf("请输入你要读取到的顺序表名: \n");
    scanf("%s",ListName);
    int i = LocateList(Lists, ListName);
    if(i==0){
        printf("不存在顺序表%s!\n",ListName);
    }
    else
        Lists.elem[i-1].L.elem = NULL;
    switch(LoadList(Lists.elem[i-1].L, FileName))
    {
        case OK:
            ListTraverse(Lists.elem[i-1].L,&TraversePrint);
            printf("从文件读取到顺序表操作成功! \n");
            break;
        case ERROR:
            printf("操作失败, 请检查文件名。 \n");
            break;
        case INFEASIBLE:
            printf("操作失败, 顺序表已存在! \n");
            break;
    }
    getchar();getchar();
    break;
}
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
```

# 华中科技大学课程实验报告

```
    }
case 18:
{
    char SqListName[30];
    printf("请输入你要添加的顺序表名: \n");
    scanf("%s", SqListName);
    if(LocateList(Lists, SqListName)){
        printf("添加顺序表失败,名称重复! \n");
        break;
    }
    switch (AddList(Lists, SqListName)) {
        case OK:
            printf("添加顺序表成功! \n");
            break;
        case ERROR:
            printf("添加顺序表失败! \n");
            break;
        default:
            break;
    }
    getchar();getchar();
    break;
}
case 19:
{
    char SqListName[30];
    printf("请输入你要删除的顺序表名: \n");
    scanf("%s", SqListName);
    switch (RemoveList(Lists, SqListName)) {
        case OK:
            printf("删除顺序表成功! \n");
            break;
        case ERROR:
            printf("删除顺序表失败! \n");
            break;
        default:
            break;
    }
    getchar();getchar();
    break;
}
```

# 华中科技大学课程实验报告

```
case 20:  
{  
    char SqListName[30];  
    printf("请输入你要切换到的顺序表名: \n");  
    scanf("%s", SqListName);  
    switch ((idt=LocateList(Lists, SqListName))) {  
        case 0:  
            printf("切换顺序表失败! \n");  
            break;  
        default:  
            PtrL = &Lists.elem[idt-1].L;  
            printf("切换到顺序表 %s 成功! \n", SqListName);  
            break;  
    }  
    getchar();getchar();  
    break;  
}  
case 21:  
{  
    printf("\n-----Lists-----\n");  
    for (int i = 0; i<Lists.length; i++) {  
        printf("%s\n", Lists.elem[i].name);  
    }  
    printf("-----EndLists-----\n");  
    getchar();getchar();  
    break;  
}  
case 0:  
    break;  
default:  
    printf("操作编号不正确! \n");  
    getchar();getchar();  
    break;  
}// end of switch  
}// end of while  
printf("欢迎下次再使用本系统! \n");  
}// end of main()
```

## 5 附录 B 基于链式存储结构线性表实现的源程序

```

#include "stdio.h"                                1
#include "stdlib.h"                               2
#include "string.h"                                3
#include "limits.h"                               4
#include "stack"                                 5
#define TRUE 1                                  6
#define FALSE 0                                 7
#define OK 1                                    8
#define ERROR 0                                9
#define INFEASIBLE -1                         10
#define UNDEF OVERFLOW                         11
#define OVERFLOW -2                            12
#define max(i,j) ((i)>(j)?(i):(j))           13
                                                14

typedef int status;                             15
typedef int ElemType; // 数据元素类型定义      16
typedef int ElemType;                          17
typedef struct LNode{ // 单链表（链式结构）结点的定义 18
    ElemType data;
    struct LNode *next;
}LNode,*LinkList;                           19
                                            20
                                            21
                                            22

typedef struct THELISTS{                     23
    struct ALIST{
        char name[30] = {0,};
        LinkList L;
    }elem[50];
    int length = 0, listssize=50;
} LISTS;                                     24
                                            25
                                            26
                                            27
                                            28
                                            29
                                            30
                                            31

/* 初始化单链表，成功返回OK，失败返回OVERFLOW。*/
status InitList(LinkList &L)                  32
{
    if(!L){                                     33
        L = (LNode*) malloc(sizeof(LNode));     34
        L->next = NULL;                      35
        return OK;                           36
    }                                         37
                                            38
                                            39
}

```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
    else return INFEASIBLE;          40
}

/* 如果单链表L存在，销毁单链表L，释放数据元素的空间，返回OK，否则返回INFEASIBLE。 */
status DestroyList(LinkList &L)          41
{
    if(L){
        LNode *p = L, *q;
        while(p){          42
            q = p->next;
            free(p);
            p = q;
        }
        L = NULL;
        return OK;
    }
    else return INFEASIBLE;          43
}                                      44

/* 如果单链表L存在，删除单链表L中的所有元素，返回OK，否则返回INFEASIBLE。 */
status ClearList(LinkList &L)          45
{
    if(L){          46
        LNode *p = L->next,*q;
        while(p){          47
            q = p->next;
            free(p);
            p = q;
        }
        L->next = NULL;          48
        return OK;
    }
    else return INFEASIBLE;          49
}                                      50

/* 如果单链表L存在，判断单链表L是否为空，空就返回TRUE，否则返回FALSE；如果单链表L不存在，返
   回INFEASIBLE。 */
status ListEmpty(LinkList L)          51
{
    if(L) {          52
        if(L->next)
            return FALSE;
        else
            return TRUE;
    }
    else return INFEASIBLE;          53
}                                      54

}                                      55

}                                      56

}                                      57

}                                      58

}                                      59

}                                      60

}                                      61

}                                      62

}                                      63

}                                      64

}                                      65

}                                      66

}                                      67

}                                      68

}                                      69

}                                      70

}                                      71

}                                      72

}                                      73

}                                      74

}                                      75

}                                      76

}                                      77

}                                      78

}                                      79
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
    if(L->next) return FALSE;
    else return TRUE;
}
return INFEASIBLE;
}

/* 如果单链表L存在，返回单链表L的长度，否则返回INFEASIBLE。 */
int ListLength(LinkList L)
{
    if(L) {
        LNode *p = L->next;
        int i=0;
        while(p){
            i++;
            p = p->next;
        }
        return i;
    }
    else return INFEASIBLE;
}

/*
如果单链表L存在，获取单链表L的第i个元素，保存在e中，返回OK;
如果i不合法，返回ERROR; 如果单链表L不存在，返回INFEASIBLE。
*/
status GetElem(LinkList &L,int i,ElemType &e)
{
    if(L) {
        if(i<1) return ERROR;
        LNode *p = L->next;
        int j=0;
        while(p){
            j++;
            if(j==i) {e = p->data; return OK;}
            p = p->next;
        }
        return ERROR;
    }
    else return INFEASIBLE;
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
/* LocateElem()使用的compare()函数 */ 121
int ElemEqual(ElemType i, ElemtType j){ 122
    return i==j; 123
} 124
125
/* ListTraverse()使用的visit()函数 */ 126
int TraversePrint(ElemType e){ 127
    printf("%d ", e); 128
    return 0; 129
} 130
131
132
133
134
135
int NonExist(LinkList L){ 136
    return L->next==NULL?1:0; 137
} 138
139
/*
如果单链表L存在，查找元素e在单链表L中的位置序号；如果e不存在，返回ERROR； 140
当单链表L不存在时，返回INFEASIBLE。 141
*/
142
status LocateElem(LinkList L, ElemtType e, int (*compare)(ElemtType, ElemtType)) 143
{
    if(L) { 144
        LNode *p = L->next; 145
        int j=0; 146
        while(p){ 147
            j++; 148
            if(compare(p->data,e)) return j; 149
            p = p->next; 150
        } 151
        return ERROR; 152
    } 153
    else return INFEASIBLE; 154
} 155
156
/*
如果单链表L存在，获取单链表L中元素e的前驱，保存在pre中，返回OK； 157
如果没有前驱，返回ERROR；如果单链表L不存在，返回INFEASIBLE。 158
159
160
161
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
/*
status PriorElem(LinkList L,ElemType e,ElemType &pre)
{
    if(L) {
        LNode *p = L->next;
        while(p&&p->next){
            if(p->next->data==e) {pre = p->data; return OK;}
            p = p->next;
        }
        return ERROR;
    }
    else return INFEASIBLE;
}

/*
如果单链表L存在，获取单链表L元素e的后继，保存在next中，返回OK；
如果没有后继，返回ERROR；如果单链表L不存在，返回INFEASIBLE。
*/
status NextElem(LinkList L,ElemType e,ElemType &next)
{
    if(L) {
        LNode *p = L->next;
        while(p&&p->next){
            if(p->data==e) {next = p->next->data; return OK;}
            p = p->next;
        }
        return ERROR;
    }
    else return INFEASIBLE;
}

/*
如果单链表L存在，将元素e插入到单链表L的第i个元素之前，返回OK；
当插入位置不正确时，返回ERROR；如果单链表L不存在，返回INFEASIBLE。
*/
status ListInsert(LinkList &L,int i,ElemType e)
{
    if(L) {
        if(i<1) return ERROR;
        LNode *p = L;
        int j=-1;
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
while(p->next){ 203
    j++;
    if(j==i-1) { 204
        LNode *q = (LNode*) malloc(sizeof(LNode));
        q->next = p->next;
        p->next = q;
        q->data = e;
        return OK;
    } 210
    p = p->next;
} 212
if(i==j+2){ 214
    LNode *q = (LNode*) malloc(sizeof(LNode));
    q->next = NULL;
    p->next = q;
    q->data = e;
    return OK;
} 219
}
return ERROR;
} 222
else return INFEASIBLE;
} 224
225
/*
如果单链表L存在，删除单链表L的第i个元素，并保存在e中，返回OK； 227
当删除位置不正确时，返回ERROR；如果单链表L不存在，返回INFEASIBLE。 228
*/
status ListDelete(LinkList &L,int i,ElemType &e) 230
{
    if(L) { 231
        if(i<1) return ERROR;
        LNode *p = L;
        int j=-1;
        while(p->next){ 236
            j++;
            if(j==i-1) { 238
                e = p->next->data;
                LNode *q = p->next;
                p->next = q->next;
                free(q);
                return OK;
            } 242
        } 237
    } 239
} 240
241
242
243
```

# 华中科技大学课程实验报告

```
        }
        p = p->next;
    }
    return ERROR;
}
else return INFEASIBLE;
}
/*
如果单链表L存在，依次显示单链表中的元素，每个元素间空一格，返回OK;
如果单链表L不存在，返回INFEASIBLE。
*/
status ListTraverse(LinkList L, int (*visit)(ElemType))
{
    if(L){
        LNode *p = L->next;
        printf("\n----- all elements-----\n");
        if(ListEmpty(L)) printf("\t\t\t单链表是空表。");
        while(p){
            visit(p->data);
            p = p->next;
        }
        printf("\n-----end all elements-----\n");
        return OK;
    }
    else return INFEASIBLE;
}

/* 13. 链表翻转 */
status ReverseList(LinkList & L){
    if(L){
        if(ListLength(L)==0){
            printf("单链表是空表! \n");
            return OK;
        }
        std::stack<int> St;
        LNode * p = L->next;
        while (p) {
            St.push(p->data);
            p = p->next;
        }
        while (!St.empty()){
            L->next = St.top();
            St.pop();
            L = L->next;
        }
        L->next = NULL;
    }
}
```

# 华中科技大学课程实验报告

```
}

p = L->next;

while (p) {
    p->data = St.top();
    St.pop();
    p = p->next;
}

return OK;
}

else return INFEASIBLE;
}

/* 15.链表排序 */

status SortList(LinkList & L){
    if(L){

        LNode * i = L->next, * j;

        while(i&&i->next){

            int iNextData = i->next->data;

            j = L;

            int flag = 1;

            while(j != i){

                if((j==L || j->data < iNextData) && j->next->data > iNextData){

                    LNode * k = (LNode*)malloc(sizeof(LNode));

                    k->data = iNextData;

                    k->next = j->next;

                    j->next = k;

                    k = i->next;

                    i->next = i->next->next;

                    flag = 0;

                    free(k);

                    break;
                }

                j = j->next;
            }

            if(flag) i = i->next;
        }

        return OK;
    }

    else return INFEASIBLE;
}

```

# 华中科技大学课程实验报告

```
/* 14.删除链表的倒数第k个结点 */ 326
status RemoveNthFromEnd(LinkList & L, int n){ 327
    if(L){ 328
        int l, i=1, k; 329
        if(n > (l=ListLength(L)) || ListEmpty(L))return ERROR; 330
        k = l-n+1; 331
        LNode * p = L, * q; 332
        while(k != i){ 333
            p = p->next; 334
            i++; 335
        } 336
        q = p->next; 337
        p->next = p->next->next; 338
        free(q); 339
        return OK; 340
    } 341
    else return INFEASIBLE; 342
} 343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365

/* 16:如果单链表L存在，将单链表L的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。
 */
status SaveList(LinkList & L,const char FileName[])
{
    if(L){
        FILE *fp = fopen(FileName,"wb");
        if(fp == NULL){
            printf("打开文件失败\n");
            return ERROR;
        }
        LNode *p = L->next;
        while(p){
            fprintf(fp, "%d", p->data);
            p = p->next;
        }
        fclose(fp);
        return OK;
    }else return INFEASIBLE;
}
```

# 华中科技大学课程实验报告

```
366  
367  
368  
/* 17:如果单链表L不存在，将FileName文件中的数据读入到单链表L中，返回OK，否则返回INFEASIBLE  
。 */  
369  
status LoadList(LinkList &L, const char FileName[])  
370  
{  
    if(!L){  
        FILE *fp = fopen(FileName, "rb");  
        if(fp == NULL){  
            printf("打开文件失败!\n");  
            return ERROR;  
        }  
        L = (LinkList) malloc(sizeof(LNode)); // 空的头结点  
        L->next = NULL;  
        LNode *p = L;  
        int k;  
        while(!feof(fp)&&fscanf(fp, "%d", &k)){  
            p->next = (LNode*)malloc(sizeof(LNode));  
            if(!p->next) return ERROR;  
            p = p->next;  
            p->data = k;  
        }  
        p->next = NULL;  
        fclose(fp);  
        return OK;  
    }  
    else return INFEASIBLE;  
}  
390  
391  
392  
393  
/* 22:Lists中查找一个名称为ListName的单链表 */  
394  
int LocateList(LISTS Lists, char ListName[])  
395  
{  
    for(int i=0; i<Lists.length; i++) {  
        if(strcmp(Lists.elem[i].name, ListName)==0) return i + 1;  
    }  
    return 0;  
}  
396  
397  
398  
399  
400  
401  
402  
/* 19:Lists中删除一个名称为ListName的单链表 */  
403  
status RemoveList(LISTS &Lists, char ListName[])  
404  
{  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405
```

# 华中科技大学课程实验报告

```
int flag = 0; 406
for(int i=0; i<Lists.length; i++){
    if(strcmp(ListName, Lists.elem[i].name) == 0){
        flag = 1;
        for(int j=i; j<Lists.length-1; j++){
            Lists.elem[j] = Lists.elem[j+1];
        }
        Lists.length--;
    }
}
if(flag == 0) return ERROR; 416
else return OK; 417
} 418

/* 18:Lists中添加一个名称为ListName的单链表 */
status AddList(LISTS &Lists, char ListName[])
{
    if(Lists.length == Lists.listssize) { 423
        printf("单链表数目达到最大值!\n");
        return ERROR; 425
    }
    if(LocateList(Lists, ListName)){ 427
        printf("不允许表名重复! \n");
        return ERROR; 429
    }
    THELISTS::ALIST & list = Lists.elem[Lists.length]; 431
    strcpy(list.name, ListName); 432
    list.L = NULL; 433
    Lists.length++; 434
    return OK; 435
} 436

/* 22:DisplayList */
status DisplayList(LISTS&Lists, char filename[]){
    int i;
    if(i=LocateList(Lists, filename)){ 441
        printf("ListName:%s is the NO.%d list\n",Lists.elem[i-1].name,i); 442
        if(ListTraverse(Lists.elem[i-1].L, &TraversePrint)==OK); 443
        else printf("单链表未初始化! \n");
        return OK; 445
    }else return ERROR; 446
}
```

# 华中科技大学课程实验报告

```
}

int main(void){
    int op=1, idt=1;
    LISTS Lists;
    for(int i = 0;i < 50;i++) Lists.elem[i].L = NULL;
    strcpy(Lists.elem[0].name, "Default: Lists.elem[0]");
    while(op){

        LinkList &L = Lists.elem[idt-1].L;
        printf("\n\n");
        printf("Menu for Linear Table On Single Linked Structure \n");
        printf("-----\n");
        printf("      1. InitList      2. DestroyList\n");
        printf("      3. ClearList     4. ListEmpty\n");
        printf("      5. ListLength    6. GetElem \n");
        printf("      7. LocateElem    8. PriorElem\n");
        printf("      9. NextElem      10. ListInsert\n");
        printf("     11. ListDelete    12. ListTraverse\n");
        printf("     13. ReverseList   14. RemoveNthFromEnd\n");
        printf("     15. SortList      16. SaveList\n");
        printf("     17. LoadList      18. AddList\n");
        printf("     19. RemoveList    20. SwitchList\n");
        printf("     21. ShowLists     22. DisplayList\n");
        printf("     0. Exit\n");
        printf("-----\n");
        printf("\t活动单链表: %s\n", Lists.elem[idt-1].name);
        printf("    请选择你的操作 [0~22]:\n");
        scanf("%d",&op);
        switch(op){

            case 1:
                if(InitList(L)==OK) printf("单链表创建成功! \n");
                else printf("单链表创建失败! 单链表已存在\n");
                getchar();getchar();
                break;
            case 2:
                if(DestroyList(L)==OK) printf("单链表删除成功! \n");
                else printf("单链表删除失败! 单链表不存在。 \n");
                getchar();getchar();
                break;
            case 3:
                if(ClearList(L)==OK) printf("单链表清空成功! \n");
                else printf("单链表清空失败! \n");
                getchar();getchar();
                break;
            case 4:
                if(ListEmpty(L)) printf("单链表为空!\n");
                else printf("单链表不为空!\n");
                getchar();getchar();
                break;
            case 5:
                printf("单链表长度为: %d\n", ListLength(L));
                getchar();getchar();
                break;
            case 6:
                printf("单链表第%d个元素为: %d\n", PriorElem(L));
                getchar();getchar();
                break;
            case 7:
                printf("单链表第%d个元素为: %d\n", LocateElem(L));
                getchar();getchar();
                break;
            case 8:
                printf("单链表第%d个元素为: %d\n", NextElem(L));
                getchar();getchar();
                break;
            case 9:
                if(ListInsert(L, 10, 10) == OK) printf("插入成功!\n");
                else printf("插入失败!\n");
                getchar();getchar();
                break;
            case 10:
                if(ListInsert(L, 10, 20) == OK) printf("插入成功!\n");
                else printf("插入失败!\n");
                getchar();getchar();
                break;
            case 11:
                if(ListDelete(L, 10) == OK) printf("删除成功!\n");
                else printf("删除失败!\n");
                getchar();getchar();
                break;
            case 12:
                if(ListTraverse(L) == OK) printf("遍历成功!\n");
                else printf("遍历失败!\n");
                getchar();getchar();
                break;
            case 13:
                if(ReverseList(L) == OK) printf("反转成功!\n");
                else printf("反转失败!\n");
                getchar();getchar();
                break;
            case 14:
                if(RemoveNthFromEnd(L, 10) == OK) printf("删除成功!\n");
                else printf("删除失败!\n");
                getchar();getchar();
                break;
            case 15:
                if(SortList(L) == OK) printf("排序成功!\n");
                else printf("排序失败!\n");
                getchar();getchar();
                break;
            case 16:
                if(SaveList(L) == OK) printf("保存成功!\n");
                else printf("保存失败!\n");
                getchar();getchar();
                break;
            case 17:
                if(AddList(L) == OK) printf("添加成功!\n");
                else printf("添加失败!\n");
                getchar();getchar();
                break;
            case 18:
                if(SwitchList(L) == OK) printf("切换成功!\n");
                else printf("切换失败!\n");
                getchar();getchar();
                break;
            case 19:
                if(RemoveList(L) == OK) printf("删除成功!\n");
                else printf("删除失败!\n");
                getchar();getchar();
                break;
            case 20:
                if(DisplayList(L) == OK) printf("显示成功!\n");
                else printf("显示失败!\n");
                getchar();getchar();
                break;
            case 21:
                if>ShowLists(L) == OK) printf("显示成功!\n");
                else printf("显示失败!\n");
                getchar();getchar();
                break;
            case 22:
                if(DisplayList(L) == OK) printf("显示成功!\n");
                else printf("显示失败!\n");
                getchar();getchar();
                break;
            case 0:
                printf("退出系统...\n");
                break;
        }
    }
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
else printf("单链表清空失败！单链表不存在。\\n");
getchar();getchar();
break;
case 4:
int empty;
if((empty = ListEmpty(L))!=INFEASIBLE)
    if(empty) printf("单链表为空！\\n");
    else printf("单链表不为空！\\n");
    else printf("单链表判空失败！单链表不存在。\\n");
getchar();getchar();
break;
case 5:
int len;
if((len = ListLength(L))!=INFEASIBLE)
    printf("单链表长度为: %d 。 \\n", len);
else printf("单链表求长度失败，单链表不存在！\\n");
getchar();getchar();
break;
case 6:
{
int i;
ElemType e;
printf("请输入查找的元素的逻辑序号（从1开始）：");
scanf("%d",&i);
if(GetElem(L, i, e)!=INFEASIBLE) printf("单链表的第 %d 个元素为 %d 。\\n", i
    , e);
else printf("单链表查找失败！\\n");
getchar();getchar();
}
break;
case 7:
{
ElemType e;
int i;
printf("请输入查找的目标元素：");
scanf("%d",&e);
switch(i=LocateElem(L, e, &ElemEqual)){
    case INFEASIBLE:
        printf("查找元素失败，或单链表不存在！\\n");
        break;
    case ERROR:
```

# 华中科技大学课程实验报告

```
    printf("查找元素失败，单链表中不存在该元素! \n");
    break;
default:
    printf("元素 %d 第一次出现在第 %d 个元素上。 \n", e, i);
    break;
}
getchar();getchar();
break;
}
case 8:
{
    ElemType e,pre;
    printf("请输入要查找前驱的元素: ");
    scanf("%d",&e);
    switch(PriorElem(L, e, pre)){
        case OK:
            printf("元素 %d 的前驱是 %d 。 \n",e,pre);
            break;
        case ERROR:
            printf("查找前驱元素失败，该元素不存在或不存在前驱! \n");
            break;
        case INFEASIBLE:
            printf("查找前驱元素失败，单链表不存在! \n");
            break;
        default:
            break;
    }
    getchar();getchar();
    break;
}
case 9:
{
    ElemType e,ne;
    printf("请输入要查找后继的元素: ");
    scanf("%d",&e);
    switch(NextElem(L, e, ne)){
        case OK:
            printf("元素 %d 的后继是 %d 。 \n",e,ne);
            break;
        case ERROR:
            printf("查找后继元素失败，该元素不存在或不存在后继! \n");
            break;
    }
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
        break;
    case INFEASIBLE:
        printf("查找后继元素失败, 单链表不存在! \n");
        break;
    default:
        break;
}
getchar();getchar();
break;
}
case 10:
{
    ELEMType e;
    int j;
    printf("请输入插入的元素的逻辑序号 (从1开始) : ");
    scanf("%d",&j);
    printf("请输入插入的元素的值 : ");
    scanf("%d",&e);
    switch(ListInsert(L, j, e)){
        case OK:
            printf("单链表插入成功! \n");
            break;
        case ERROR:
            printf("单链表插入失败, 插入位置非法! \n");
            break;
        case INFEASIBLE:
            printf("单链表插入失败, 单链表不存在! ");
            break;
        default:
            break;
}
getchar();getchar();
break;
}
case 11:{
    int i;
    ELEMType e;
    printf("请输入要删除的元素的逻辑序号: ");
    scanf("%d",&i);
    switch(ListDelete(L, i, e)){
        case OK:
            break;
        case ERROR:
            printf("删除失败, 单链表不存在! ");
            break;
        default:
            break;
}
getchar();getchar();
break;
}
```

# 华中科技大学课程实验报告

```
    printf("被删除的元素是 %d 。\n", e); 610
        break;
    case ERROR:
        printf("删除元素失败，指定的逻辑序号不合法！\n"); 611
        break;
    case INFEASIBLE:
        printf("删除元素失败，单链表不存在！\n"); 612
        break;
    default:
        break;
    }
    getchar();getchar(); 613
    break;
}
case 12:
{
    if(ListTraverse(L,&TraversePrint)==OK) 614
        printf("\n遍历成功！\n");
    else
        printf("遍历失败，单链表不存在！\n"); 615
    getchar();getchar(); 616
    break;
}
case 13:
{
    if(ReverseList(L)==OK){ 617
        printf("链表翻转成功！\n");
    }
    else printf("链表翻转失败！单链表不存在。"); 618
    getchar();getchar(); 619
    break;
}
case 14:
{
    int n;
    printf("删除倒数第n个元素！请输入n: ");
    scanf("%d", &n);
    switch (RemoveNthFromEnd(L, n)) { 620
        case OK:
            printf("删除倒数第n个元素成功！\n"); 621
            break;
    }
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
651  
case INFEASIBLE:  
    printf("删除倒数第n个元素失败！单链表不存在\n");  
    break;  
  
655  
case ERROR:  
    printf("删除倒数第n个元素失败！单链表是空表，或者n不合法！\n");  
    break;  
  
659  
default:  
    break;  
}  
getchar();getchar();  
break;  
}  
case 15:  
{  
    if (SortList(L)!=INFEASIBLE) {  
        if(ListEmpty(L)) printf("单链表是空表。\n");  
        printf("排序操作成功！\n");  
    }  
    else printf("操作失败，单链表不存在！\n");  
    getchar();getchar();  
    break;  
}  
case 16:  
{  
    char FileName[128];  
    printf("请输入你要保存的文件名：\n");  
    scanf("%s",FileName);  
    switch(SaveList(L, FileName))  
    {  
        case OK:  
            printf("保存单链表到文件操作成功！\n");  
            break;  
        case ERROR:  
            printf("操作失败，请检查文件名。!\n");  
            break;  
        case INFEASIBLE:  
            printf("操作失败，单链表不存在！\n");  
            break;  
    }  
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
    }
    getchar();getchar();
    break;
}
case 17:
{
    char FileName[128];
    char ListName[30];
    printf("请输入你要读取的文件名: \n");
    scanf("%s",FileName);
    printf("请输入你要读取到的单链表名: \n");
    scanf("%s",ListName);
    int i = LocateList(Lists, ListName);
    if(i==0){
        printf("不存在单链表%s!\n",ListName);
        getchar();getchar();
        break;
    }
    else
        L = NULL;
    switch(LoadList(L, FileName))
    {
        case OK:
            ListTraverse(Lists.elem[i-1].L,&TraversePrint);
            printf("从文件读取到单链表操作成功! \n");
            break;
        case ERROR:
            printf("操作失败, 请检查文件名。 \n");
            break;
        case INFEASIBLE:
            printf("操作失败, 单链表已存在! \n");
            break;
    }
    getchar();getchar();
    break;
}
case 18:
{
    char SqListName[30];
    printf("请输入你要添加的单链表名: \n");
    scanf("%s",SqListName);
```

# 华中科技大学课程实验报告

```
switch (AddList(Lists, SqListName)) { 733
    case OK: 734
        printf("添加单链表成功! \n");
        break; 735
    case ERROR: 736
        printf("添加单链表失败! \n");
        break; 737
    default: 738
        break; 739
}
getchar();getchar(); 740
break; 741
}
case 19: 742
{
    char SqListName[30]; 743
    printf("请输入你要删除的单链表名: \n"); 744
    scanf("%s",SqListName); 745
    int i; 746
    if(i=LocateList(Lists, SqListName)){ 747
        if(i<idt){ 748
            idt--;
        } 749
        else; 750
    } 751
    switch (RemoveList(Lists, SqListName)) { 752
        case OK: 753
            printf("删除单链表成功! \n");
            break; 754
        case ERROR: 755
            printf("删除单链表失败! \n");
            break; 756
        default: 757
            break;
    } 758
    getchar();getchar(); 759
    break; 760
}
case 20: 761
{
    char SqListName[30]; 762
    printf("请输入你要插入的单链表名: \n"); 763
    scanf("%s",SqListName); 764
    int i; 765
    if(i=LocateList(Lists, SqListName)){ 766
        if(i>idt){ 767
            idt++;
        } 768
        else; 769
    } 770
    switch (InsertList(Lists, SqListName)) { 771
        case OK: 772
            printf("插入单链表成功! \n");
            break; 773
        case ERROR: 774
            printf("插入单链表失败! \n");
            break;
    }
}
```

# 华中科技大学课程实验报告

```
printf("请输入你要切换到的单链表名: \n");
scanf("%s", SqListName);
switch ((idt=LocateList(Lists, SqListName))) {
    case 0:
        printf("切换单链表失败! \n");
        break;
    default:
        printf("切换到单链表 %s 成功! \n", SqListName);
        break;
}
getchar();getchar();
break;
}
case 21:
{
    printf("\n-----Lists-----\n");
    for (int i = 0; i<Lists.length; i++) {
        printf("%s\n", Lists.elem[i].name);
    }
    printf("-----EndLists-----\n");
    getchar();getchar();
    break;
}
case 22:
{
    char name[30];
    printf("请输入要显示的表名: \n");
    scanf("%s", name);
    if(DisplayList(Lists, name)==OK){
        printf("显示链表成功! \n");
    }else printf("显示链表失败, 没有找到该链表。 \n");
    getchar();getchar();
    break;
}
case 0:
    break;
default:
    printf("输入的操作编号不合法! \n");
    break;
}// end of switch
}// end of while
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
    printf("欢迎下次再使用本系统! \n");
} // end of main()
```

815

816

## 6 附录 C 基于二叉链表二叉树实现的源程序

```

#include <cstring>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <stack>
#include <vector>
#include <queue>
#include <iostream>
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define UNDEF OVERFLOW
#define OVERFLOW -2
#define max(x,y) ((x)>(y)?(x):(y)) // 用宏代替最大值函数
typedef int status;
typedef int KeyType;
typedef struct { // 二叉树结点类型定义
    KeyType key;
    char others[20];
} TElemType;
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39

typedef struct BiTNode{ // 二叉链表结点的定义
    TElemType data;
    struct BiTNode *lchild,*rchild;
} BiTNode, *BiTree;

typedef struct THETREES{ // 多二叉树管理表的定义
    struct ATREE{
        BiTree T;
        char name[30];
    }elem[16];
    int length = 0;
} TREES;
36
37
38
39

/*
CreateBiTree()的递归化核心函数

```

# 华中科技大学课程实验报告

```
/*
BiTree RecurvalCreateBiTree(TElemType definition[], int & start) {
    BiTree p = (BiTNode *)malloc(sizeof(BiTNode));
    if(definition[start].key<=0) {
        start++;
        return NULL;
    };
    p->data = definition[start];
    start++;
    p->lchild = RecurvalCreateBiTree(definition, start);
    p->rchild = RecurvalCreateBiTree(definition, start);
    return p;
}

/*
根据带空枝的二叉树先根遍历序列definition构造一棵二叉树,
将根节点指针赋值给T并返回OK, 如果有相同的关键字, 返回ERROR
*/
status CreateBiTree(BiTree &T,TElemType definition[])
{
    if(T) return INFEASIBLE;
    for(int i=0; ; i++) {
        if(definition[i].key == -1) break;
        if(definition[i].key > 0)
            for(int j=i+1; ;j++){
                if(definition[j].key== -1) break;
                if(definition[i].key == definition[j].key)
                    return ERROR; // 检查是否有重复关键字
            }
    }
    int start = 0;
    T = RecurvalCreateBiTree(definition,start);
    return OK;
}

/*
删除所有结点, 释放结点空间
*/
status DestoryBiTree(BiTree &T)
{
    if(T==NULL) return INFEASIBLE;
```

# 华中科技大学课程实验报告

```
if(T->lchild) DestoryBiTree(T->lchild);           81
if(T->rchild) DestoryBiTree(T->rchild);           82
free(T);                                              83
T=NULL;                                                 84
return OK;                                             85
}

/*
将二叉树结点数据域清空                           88
*/
status ClearBiTree(BiTree &T)                      91
{
    if(T==NULL) return INFEASIBLE;                   93
    if(T->lchild) ClearBiTree(T->lchild);          94
    if(T->rchild) ClearBiTree(T->rchild);          95
    T->data.key=0;                                  96
    for(int i=0;i<30;i++) T->data.others[i]='\0';  97
    return OK;                                         98
}
/*
二叉树判空                                         101
*/
int BiTreeEmpty(BiTree &T){                         104
    if(T==NULL) return 1;                            105
    else return 0;                                 106
}
/*
求二叉树T的深度                                     109
*/
int BiTreeDepth(BiTree T)                           112
{
    if(T==NULL) return 0;                            114
    return 1 + fmax(BiTreeDepth(T->lchild),BiTreeDepth(T->rchild)); 115
}
/*
查找结点                                         118
*/
BiTNode * LocateNode(BiTree T, KeyType e)          121
```

# 华中科技大学课程实验报告

```
{ 122
    if(T==NULL) return NULL; 123
    if(T->data.key==e) return T; 124
    BiTNode * p; 125
    if((p=LocateNode(T->lchild, e))!=NULL) return p; 126
    if((p=LocateNode(T->rchild, e))!=NULL) return p; 127
    else return NULL; 128
} 129
130
/*
    结点赋值 131
*/
status Assign(BiTTree &T, KeyType e, TElemType value) 132
{
    if(T==NULL) return INFEASIBLE; 133
    BiTNode * p;
    if((p = LocateNode(T, e))!=NULL){ 134
        if(e!=value.key) if(LocateNode(T, value.key)) return ERROR; 135
        p->data = value; 136
        return OK; 137
    }
    else return ERROR; 138
}
139
/*
    查找双亲结点 140
*/
BiTNode * LocateParents(BiTTree T, KeyType e) 141
{
    if(T==NULL) return NULL; 142
    BiTNode * p;
    if(T->lchild) { 143
        if(T->lchild->data.key==e) return T; 144
        if((p=LocateParents(T->lchild, e))!=NULL) return p; 145
    }
    if(T->rchild) { 146
        if(T->rchild->data.key==e) return T; 147
        if((p=LocateParents(T->rchild, e))!=NULL) return p; 148
    }
    return NULL; 149
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
/*
    辅助函数: 获得兄弟结点
*/
BiTNode * GetSibling(BiTTree T,KeyType e)
{
    BiTNode * p = LocateParents(T, e);
    if(p==NULL) return NULL;
    if(p->lchild) if(p->lchild->data.key==e) return p->rchild;
    if(p->rchild) if(p->rchild->data.key==e) return p->lchild;
    return NULL;
}

/*
    插入结点
*/
status InsertNode(BiTTree &T,KeyType e,int LR,TElemType c)
{
    if(T == NULL && LR != -1) return INFEASIBLE;
    BiTNode * p = LocateNode(T, e);
    if(p == NULL && LR != -1) return ERROR;
    if(LocateNode(T, c.key)) return ERROR;
    switch (LR) {
        case -1:
        {
            BiTNode * tmp = (BiTNode * )malloc(sizeof(BiTNode));
            tmp->data = c;
            tmp->lchild = NULL;
            tmp->rchild = T;
            T = tmp;
            break;
        }
        case 0:
        {
            BiTNode * tmp = (BiTNode * )malloc(sizeof(BiTNode));
            tmp->data = c;
            tmp->lchild = NULL;
            tmp->rchild = p->lchild;
            p->lchild = tmp;
            break;
        }
    }
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
case 1:  
{  
    BiTNode * tmp = (BiTNode * )malloc(sizeof(BiTNode));  
    tmp->data = c;  
    tmp->lchild = NULL;  
    tmp->rchild = p->rchild;  
    p->rchild = tmp;  
    break;  
}  
default:  
    break;  
}  
return OK;  
}  
  
/*  
计算结点的度  
*/  
int Degree(BiTNode *p){  
    int degree=0;  
    if(p==NULL) return 0;  
    if(p->lchild) degree++;  
    if(p->rchild) degree++;  
    return degree;  
}  
  
/*  
删除结点  
*/  
status DeleteNode(BiTree &T,KeyType e)  
{  
    if(T==NULL) return INFEASIBLE;  
    BiTNode * p = LocateNode(T, e);  
    if(p==NULL) return ERROR;  
    BiTNode * parents = LocateParents(T, e);  
    if(parents==NULL) {  
        switch (Degree(p)) {  
            case 0:  
            {  
                free(T);  
                T = NULL;  
            }  
        }  
    }  
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
        break; 245
    }
    case 1: 246
    {
        p = p->lchild?p->lchild:p->rchild; 247
        free(T); 248
        T = p;
        break; 249
    }
    case 2: 250
    {
        BiTNode * q = p->rchild; 251
        p = p->lchild; 252
        free(T);
        T = p;
        while(p->rchild) p = p->rchild; 253
        p->rchild = q;
        break; 254
    }
    default: 255
    {
        break;
    }
    return OK;
}// if(parents==NULL) 256
switch (Degree(p)) { 257
    case 0: 258
    {
        if(parents->lchild==p) { 259
            parents->lchild=NULL; 260
            free(p); 261
        }
        if(parents->rchild==p) { 262
            parents->rchild=NULL; 263
            free(p); 264
        }
        break; 265
    }
    case 1: 266
    {
        int lr; 267
    }
}
```

# 华中科技大学课程实验报告

```
if(p->lchild) lr = 0; else lr = 1; 286
if(parents->lchild==p) { 287
    parents->lchild = lr ? p->rchild : p->lchild; 288
    free(p); 289
}
if(parents->rchild==p) 290
{
    parents->rchild = lr ? p->rchild : p->lchild; 291
    free(p); 292
}
break; 293
}
case 2: 294
{
    BiTNode * extright = p->lchild; 295
    while (extright->rchild) 296
        extright = extright->rchild;
    extright->rchild = p->rchild; 297
    if(parents->lchild==p) { 298
        parents->lchild=p->lchild; 299
        free(p); 300
    }
    if(parents->rchild==p) 301
    {
        parents->rchild=p->lchild; 302
        free(p); 303
    }
    break; 304
}
default: 305
{
    break; 306
}
}//switch(Degree(p)) 307
return OK; 308
}

/*
先序遍历二叉树
*/
status PreOrderTraverse(BiTTree T,void (*visit)(BiTree))
{
    if(T==NULL) return ERROR; 321
}
322
323
324
325
326
```

# 华中科技大学课程实验报告

```
visit(T);                                327
    PreOrderTraverse(T->lchild, visit);   328
    PreOrderTraverse(T->rchild, visit);   329
    return OK;                            330
}

/*
    中序遍历二叉树
*/
status InOrderTraverse(BiTree T,void (*visit)(BiTree))
{
    if(T==NULL) return ERROR;              338
    std::stack<BiTNode*> s;              339
    BiTNode * p = T;                     340
    while(p||!s.empty()){                341
        if(p){                           342
            s.push(p);
            p = p->lchild;             343
        }else{                           344
            p = s.top(); s.pop();      345
            visit(p);                 346
            p = p->rchild;             347
        }
    }
    return OK;                            351
}                                         352
                                         353

/*
    后序遍历二叉树
*/
status PostOrderTraverse(BiTree T,void (*visit)(BiTree))
{
    if(T==NULL) return ERROR;              359
    PostOrderTraverse(T->lchild, visit); 360
    PostOrderTraverse(T->rchild, visit); 361
    visit(T);                            362
    return OK;                            363
}                                         364
                                         365

/*
    按层遍历二叉树

```

# 华中科技大学课程实验报告

```
/*
status LevelOrderTraverse(BiTree T,void (*visit)(BiTree))
{
    if(T==NULL) return ERROR;
    std::queue<BiTNode*> qu;
    qu.push(T);
    while(!qu.empty()){
        BiTNode * p = qu.front();
        visit(p);
        qu.pop();
        if(p->lchild) qu.push(p->lchild);
        if(p->rchild) qu.push(p->rchild);
    }
    return OK;
}

/*
先序遍历二叉树T，构建先序遍历定义序列
*/
void SaveTraverse(BiTree T, TElemType def[], int &start)
{
    if(T==NULL) {def[start].key = 0; start++; return;}
    else def[start] = T->data;
    start++;
    SaveTraverse(T->lchild, def, start);
    SaveTraverse(T->rchild, def, start);
    return;
}

/*
将二叉树的结点数据写入到文件<FileName>中
*/
status SaveBiTree(BiTree T, char FileName[])
{
    if(T==NULL) return INFEASIBLE;
    TElemType def[128];
    memset((void*)def, 0, sizeof(def));
    int start=0;
    SaveTraverse(T,def,start);
    def[127].key = -1;
    FILE *fp = fopen(FileName, "wb");

```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
if(fp == NULL) return ERROR; 409
fwrite(def, sizeof(TElemType), 128, fp); 410
fclose(fp); 411
return OK; 412
}

/*
    读入文件<FileName>的结点数据，创建二叉树
*/
status LoadBiTree(BiTree &T, char FileName[])
{
    if(T) return INFEASIBLE; 420
    TElemType def[128];
    FILE *fp = fopen(FileName, "rb"); 422
    if(fp == NULL) return ERROR; 423
    fread(def, sizeof(TElemType), 128, fp); 424
    CreateBiTree(T, def); 425
    return OK; 426
}
/*
    在管理表中查找给定名称的二叉树
*/
int LocateTree(TREES &Trees, char name[]){
    for(int i=0;i<Trees.length;i++)
    {
        if(strcmp(name, Trees.elem[i].name)==0) return i; 435
    }
    return -1; 437
}

/*
    辅助函数Visit，打印传入结点的数据域的内容
*/
void visit(BiTree T){ 443
    printf("%d %s ", T->data.key, T->data.others); 444
}

/*
    最大路径和
*/

```

# 华中科技大学课程实验报告

```
int MaxPathSum(BiTree T){ 450
    if(T==NULL) return 0; 451
    return T->data.key + max(MaxPathSum(T->lchild), MaxPathSum(T->rchild)); 452
}
/*
    最近公共祖先
*/
BiTNode * LowestCommonAncestor(BiTree T, KeyType e1, KeyType e2) { 457
    if(T->data.key==e1||T->data.key==e2) return T; 458
    if(!LocateNode(T, e1) || !LocateNode(T, e2)) { 459
        return NULL; 460
    }
    BiTNode *t[2][2]; 461
    memset(t, 0, sizeof(t));
    t[0][0] = LocateNode(T->lchild, e1); 464
    t[0][1] = LocateNode(T->lchild, e2); 465
    t[1][0] = LocateNode(T->rchild, e1); 466
    t[1][1] = LocateNode(T->rchild, e2); 467
    if(t[0][0] && t[0][1]) return LowestCommonAncestor(T->lchild, e1, e2); 468
    if(t[1][0] && t[1][1]) return LowestCommonAncestor(T->rchild, e1, e2); 469
    return T; 470
} 471
/*
    翻转二叉树
*/
void InvertTree(BiTree T) { 476
    if(T==NULL) return; 477
    BiTNode * tmp = T->lchild; 478
    T->lchild = T->rchild; 479
    T->rchild = tmp; 480
    InvertTree(T->lchild); 481
    InvertTree(T->rchild); 482
} 483
/*
    主函数
*/
int main(void){
    for(int i=0;i<5;i++) std::cout<<std::endl;
    std::cout<<"*****二叉树实验*****"<<std::endl; 489
} 490
```

# 华中科技大学课程实验报告

```
    endl;  
491  
for(int i=0;i<5;i++) std::cout<<std::endl;  
492  
int op=1, idt=0;  
493  
TREES Trees;  
493  
while(op){  
494  
    if(Trees.length==0) {  
        printf("二叉树管理表为空, 请先建立二叉树! \n");  
496  
        op = 18;  
497  
    }  
498  
    else  
499  
    {  
500  
        printf("\n\n");  
501  
        printf("          Menu for Binary Tree \n");  
502  
        printf("-----\n");  
503  
        printf("          1. CreatBiTree      2. DestroyBiTree\n");  
504  
        printf("          3. ClearBiTree     4. BiTreeEmpty\n");  
505  
        printf("          5. BiTreeDepth     6. LocateNode \n");  
506  
        printf("          7. Assign         8. GetSibling\n");  
507  
        printf("          9. InsertNode     10. DeleteNode\n");  
508  
        printf("          11. PreOrderTraverse 12. InOrderTraverse\n");  
509  
        printf("          13. PostOrderTraverse 14. LevelOrderTraverse\n");  
510  
        printf("          15. MaxPathSum     16. LowestCommonAncestor\n");  
511  
        printf("          17. InvertTree     18. AddTree\n");  
512  
        printf("          19. RemoveTree     20. SwitchTree\n");  
513  
        printf("          21. SaveTree       22. LoadTree(DestroyBiTree will be  
514  
              called)\n");  
515  
        printf("          23. ShowTrees      \n");  
516  
        printf("          0. Exit\n");  
516  
        printf("-----\n");  
517  
        printf("    活动二叉树: %s\n", Trees.elem[idt].name);  
518  
        printf("    请选择你的操作 [0~22]:\n");  
519  
        scanf("%d",&op);  
520  
    }  
521  
    BiTree & T = Trees.elem[idt].T;  
522  
    switch(op){  
523  
        case 1:  
524  
        {  
525  
            if(T) {printf("活动二叉树已存在! \n");getchar();getchar();break;}  
526  
            printf("请输入定义 (带空子树的前序遍历) : \n");  
527  
            TElemType def[128];  
528  
            int i=0;  
529
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
while(1){  
    std::cin>>def[i].key>>def[i].others;  
    if(def[i].key == -1) break;  
    i++;  
}  
CreateBiTree(T, def);  
printf("创建二叉树成功! \n");  
getchar();getchar();  
break;  
}  
case 2:  
{  
    switch(DestroyBiTree(T))  
    {  
        case OK:  
            printf("销毁二叉树成功! \n");  
            break;  
        case INFEASIBLE:  
            printf("销毁二叉树失败, 二叉树已经不存在! \n");  
            break;  
    }  
    getchar();getchar();  
    break;  
}  
case 3:  
{  
  
    switch(ClearBiTree(T))  
    {  
        case OK:  
            printf("清空二叉树成功! \n");  
            break;  
        case INFEASIBLE:  
            printf("清空二叉树失败, 二叉树已经不存在! \n");  
            break;  
    }  
    getchar();getchar();  
    break;  
}  
case 4:
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
{ 571  
    switch(BiTreeEmpty(T)) {  
        case 0: 572  
            printf("树不为空! \n");  
            break; 573  
        case 1: 574  
            printf("树为空! \n");  
            break; 575  
        default: 576  
            break; 577  
    } 578  
    getchar();getchar();  
    break; 579  
 580  
}  
 581  
case 5:  
{ 582  
    printf("树的深度是 %d 。\n", BiTreeDepth(T));  
    getchar();getchar();  
    break; 583  
}  
 584  
case 6:  
{ 585  
    printf("请输入要查找的关键字: \n");  
    KeyType key; 586  
    scanf("%d", &key);  
    BiTree p = LocateNode(T, key); 587  
    if(p) printf("结点内容: key: %d , others : %s .", p->data.key, p->data.others);  
    else 588  
        printf("查找该关键字失败! ");  
    getchar();getchar();  
    break; 589  
}  
 590  
case 7:  
{ 591  
    TElemType value;  
    KeyType key;  
    printf("请输入要赋值的结点的关键字: \n");  
    scanf("%d", &key); 592  
 593  
 594  
 595  
 596  
 597  
 598  
 599  
 600  
 601  
 602  
 603  
 604  
 605  
 606  
 607  
 608  
 609  
 610
```

# 华中科技大学课程实验报告

```
printf("请输入要赋的关键字的值: \n");
scanf("%d", &value.key);
printf("请输入要赋的others值: \n");
scanf("%s", value.others);
switch (Assign(T, key, value)) {
    case INFEASIBLE:
        printf("赋值失败, 二叉树为空! \n");
        break;
    case OK:
        printf("赋值成功! \n");
        break;
    case ERROR:
        printf("赋值失败! 目标关键字不存在或赋值后有关键字相同。 \n");
        break;
    default:
        break;
}
getchar();getchar();
break;
}
case 8:
{
    printf("请输入目标关键字: \n");
    KeyType key;
    scanf("%d", &key);
    BiTree sibling = GetSibling(T, key);
    if(sibling)
        printf("兄弟结点: key : %d ; others: %s .\n", sibling->data.key,
               sibling->data.others);
    else
        printf("查找兄弟结点失败, 目标结点无兄弟结点, 或者目标结点是根结点。 \n"
               );
    getchar();getchar();
    break;
}
case 9:
{
    printf("请输入目标关键字: \n");
    KeyType key;
    scanf("%d", &key);
```

# 华中科技大学课程实验报告

```
TElemType newdata;
printf("0表示作为左孩子插入，1表示作为右孩子插入，-1代表作为根结点插入，\n");
   请输入(0 or 1) : \n";
int lr;
scanf("%d", &lr);
printf("请输入插入的结点的关键字、数据内容:\n");
scanf("%d %s", &newdata.key, newdata.others);
switch(InsertNode(T, key, lr, newdata)){
    case INFEASIBLE:
    {
        printf("插入失败，树是空树！\n");
        break;
    }
    case OK:
    {
        char namelr[5];
        if(lr) strcpy(namelr, "右");
        else strcpy(namelr, "左");
        printf("在关键字为 %d 的结点插入关键字为 %d 数据内容为 %s 的 \
                结点作为%s孩子成功！",
               key, newdata.key, newdata.others, namelr);
        break;
    }
    case ERROR:
    {
        printf("插入结点失败！，可能找不到目标结点，或者出现名称冲突。");
        break;
    }
}
getchar();getchar();
break;

}

case 10:
{
    printf("请输入目标关键字: \n");
    KeyType key;
    scanf("%d", &key);
    switch (DeleteNode(T, key)) {
        case OK:
            printf("删除成功！\n");
            break;
    }
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
        break;                                690
    case INFEASIBLE:
        printf("删除失败！树是空树！");
        break;                                691
    case ERROR:
        printf("删除失败！树上没有目标结点。");
        break;                                692
    default:
        break;                                693
    }
getchar();getchar();
break;                                694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
        break; 731
    }
    case 15:
    {
        int theMaxPathSum;
        if((theMaxPathSum = MaxPathSum(T))){ 732
            printf("最大路径和为 %d 。\n", theMaxPathSum); 733
        }
        else printf("树是空树！无法求最大路径和。 \n"); 734
        getchar();getchar();
        break; 735
    }
    case 16:
    {
        KeyType e1, e2;
        printf("请输入要查找公共祖先的两个结点关键字: \n"); 736
        scanf("%d %d", &e1, &e2);
        BiTNode * LCA = LowestCommonAncestor(T, e1, e2); 737
        if(LCA) printf("公共祖先是:\n key : %d ;\n others : %s \n",LCA->data. 738
            key, LCA->data.others);
        else printf("查找公共祖先失败！ \n"); 739
        getchar();getchar();
        break; 740
    }
    case 17:
    {
        if(T==NULL) printf("树为空树！ \n"); 741
        InvertTree(T);
        printf("翻转二叉树成功！ \n"); 742
        getchar();getchar();
        break; 743
    }
    case 18:
    {
        printf("请输入要添加的二叉树名称: \n"); 744
        char name[30];
        scanf("%s", name);
        if(LocateTree(Trees, name) >= 0){ 745
            printf("添加二叉树失败，名称不可以重复！ \n"); 746
            getchar();getchar();
            break; 747
        }
    }
}
```

# 华中科技大学课程实验报告

```
    }
    else{
        strcpy(Trees.elem[Trees.length].name, name);
        Trees.elem[Trees.length++].T = NULL;
        printf("添加二叉树 %s 成功! \n", name);
        getchar();getchar();
        break;
    }
}
case 19:
{
    printf("请输入要移除的二叉树名称: \n");
    char name[30];
    scanf("%s", name);
    int RmNum = LocateTree(Trees, name);
    char nowName[30];
    strcpy(nowName, Trees.elem[idt].name);
    for (int i = RmNum; i < Trees.length; i++) {
        Trees.elem[i] = Trees.elem[i+1];
    }
    Trees.length--;
    int newidt;
    if((newidt = LocateTree(Trees, nowName)) >= 0)
        idt = newidt;
    else
        idt++;
    getchar();getchar();
    break;
}
case 20:
{
    char name[30];
    int j;
    printf("请输入要切换到的二叉树名称: \n");
    scanf("%s", name);
    if((j=LocateTree(Trees,name))>=0) idt = j;
    getchar();getchar();
    break;
}
case 21:
{
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
char filename[30];                                812
printf("请输入要保存到的文件的名称: \n");          813
scanf("%s", filename);                            814
switch(SaveBiTree(T, filename)){                  815
    case INFEASIBLE:                           816
    {
        printf("保存失败, 树是空树! \n");
        break;                                  817
    }
    case OK:                                 818
    {
        printf("保存成功! \n");
        break;                                  819
    }
    case ERROR:                             820
    {
        printf("保存失败, 打开文件失败! \n");
        break;                                  821
    }
}
getchar();getchar();                                822
break;                                              823
}
case 22:                                         824
{
    char filename[30];                                825
DestoryBiTree(T);
printf("读取内容到当前二叉树! 树已经被销毁! \n"); 826
printf("请输入要读取内容的文件的名称: \n");          827
scanf("%s", filename);
switch(LoadBiTree(T, filename)){                  828
    case INFEASIBLE:                           829
    {
        printf("读取失败, 树非空! \n");
        break;                                  830
    }
    case OK:                                 831
    {
        printf("读取成功! \n");
        break;                                  832
    }
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
case ERROR:  
{  
    printf("读取失败， 打开文件失败! \n");  
    break;  
}  
}  
getchar();getchar();  
break;  
}  
case 23:  
{  
    printf("管理表中的树: ");  
    for(int i=0; i<Trees.length; i++){  
        printf("\tTree%d : %s", i, Trees.elem[i].name);  
    }  
    printf("\n");  
    getchar();getchar();  
    break;  
}  
default:  
    break;  
}// end of switch  
}// end of while  
printf("欢迎下次再使用本系统! \n"); // op==0  
}// end of main()  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877
```

## 7 附录 D 基于邻接表图实现的源程序

```

#include <stdio.h>                                1
#include <stdlib.h>                               2
#include <iostream>                                3
#include <cstring>                                4
#include <queue>                                 5
#include <cmath>                                 6
#include <limits.h>                               7
#define TRUE 1                                  8
#define FALSE 0                                9
#define OK 1                                    10
#define ERROR 0                                11
#define INFEASIBLE -1                         12
#define OVERFLOW                                13
#define OVERFLOW -2                           14
#define MAX_VERTEX_NUM 200                      15
                                         16
typedef int status;                             17
typedef int KeyType;                            18
typedef enum {DG,DN,UDG,UDN} GraphKind;        19
                                         20
typedef struct {
    KeyType key;
    char others[128];
} VertexType; // 顶点类型定义                23
                                         24
typedef struct ArcNode { // 表顶点类型定义
                                         25
    int adjvex; // 顶点位置编号                26
    struct ArcNode * nextarc; // 下一个表顶点指针 27
} ArcNode;                                     28
                                         29
typedef struct VNode { // 头顶点及其数组类型定义
    VertexType data; // 顶点信息                30
    ArcNode * firstarc; // 指向第一条弧          31
} VNode, AdjList[MAX_VERTEX_NUM];               32
                                         33
typedef struct Graph { // 邻接表的类型定义
    AdjList vertices; // 头顶点数组            34
    int vexnum=0,arcnum=0; // 顶点数、弧数        35
    GraphKind kind = UDG; // 图的类型           36
} ALGraph;                                     37
                                         38
typedef struct GRAPHS{                         39

```

# 华中科技大学课程实验报告

```
struct AGRAPH{  
    ALGraph G;  
    char name[30];  
    }elem[16];  
    int length = 0;  
} GRAPHS;  
  
/*  
根据u在图G中查找顶点，查找成功返回位序，否则返回-1  
*/  
int LocateVex(ALGraph G, KeyType u){  
    for(int i=0; i<G.vexnum; i++){  
        if(G.vertices[i].data.key == u) return i;  
    }  
    return -1;  
}  
  
/*  
根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK；  
如果查找失败或关键字不唯一，返回ERROR  
*/  
status PutVex(ALGraph &G,KeyType u,VertexType value)  
{  
    int i = LocateVex(G, u);  
    if(i == -1) return ERROR;  
    for(int j=0; j<G.vexnum; j++)  
    {  
        if(j==i) continue;  
        if(G.vertices[j].data.key == value.key) return ERROR;  
    }  
    G.vertices[i].data = value;  
    return OK;  
}  
  
/* 根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，否则返回-1 */  
int FirstAdjVex(ALGraph G,KeyType u)  
{  
    int i = LocateVex(G, u);  
    if(i==-1) return -1;  
    if(G.vertices[i].firstarc==NULL) return -1;  
    return G.vertices[i].firstarc->adjvex;
```

# 华中科技大学课程实验报告

```
}

/* v对应G的一个顶点,w对应v的邻接顶点; 操作结果是返回v的(相对于w)下一个邻接顶点的位序; 如果w是最后一个邻接顶点, 或v、w对应顶点不存在, 则返回-1。 */
int NextAdjVex(ALGraph G,KeyType v,KeyType w)
{
    int i = LocateVex(G, v);
    int j = LocateVex(G, w);
    if(i<0||j<0) return -1;
    ArcNode * p = G.vertices[i].firstarc;
    while (p) {
        if(G.vertices[p->adjvex].data.key == w) break;
        p = p->nextarc;
    }
    if(!p||!p->nextarc) return -1;
    return p->nextarc->adjvex;
}

/*在图G中插入顶点v, 成功返回OK, 否则返回ERROR*/
status InsertVex(ALGraph &G,VertexType v)
{
    if(G.vexnum+1>MAX_VERTEX_NUM) return ERROR;
    if(LocateVex(G, v.key)>=0) return ERROR;
    G.vertices[G.vexnum].data = v;
    G.vertices[G.vexnum].firstarc = NULL;
    G.vexnum++;
    return OK;
}

/* 在图G中删除关键字v对应的顶点以及相关的弧, 成功返回OK, 否则返回ERROR */
status DeleteVex(ALGraph &G,KeyType v)
{
    int i = LocateVex(G, v);
    if(i==-1) return ERROR;
    if(G.vexnum==1) return ERROR;
    ArcNode * p = G.vertices[i].firstarc;
    while(p){
        ArcNode * q = G.vertices[p->adjvex].firstarc;
        if(!q) return ERROR;
        if(q->adjvex==i){
            G.vertices[p->adjvex].firstarc = G.vertices[p->adjvex].firstarc->nextarc;
        }
    }
}
```

# 华中科技大学课程实验报告

```
    free(q);                                121
}else
{
    while (q){
        if(q->nextarc&&q->nextarc->adjvex == i){
            ArcNode * s = q->nextarc;
            q->nextarc = q->nextarc->nextarc;
            free(s);
        }
        q = q->nextarc;
    }
    q = p;
    p = p->nextarc;
    free(q);
    G.arcnum--;
}
for(int j=i; j<G.vexnum-1; j++) G.vertices[j] = G.vertices[j+1];
G.vexnum--;
for(int j=0; j<G.vexnum; j++){
    ArcNode * p = G.vertices[j].firstarc;
    while (p) {
        if(p->adjvex>i) p->adjvex--;
        p = p->nextarc;
    }
}
return OK;
}

/*在图G中增加弧<v,w>, 成功返回OK,否则返回ERROR*/
status InsertArc(ALGraph &G,KeyType v,KeyType w)
{
    int i = LocateVex(G, v), j = LocateVex(G, w);
    if(i<0||j<0) return ERROR;
    ArcNode * p = G.vertices[i].firstarc;
    while (p) {
        if(p->adjvex == j) return ERROR;
        p = p->nextarc;
    }
    p = (ArcNode *)malloc(sizeof(ArcNode));
    p->adjvex = j;
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
p->nextarc = G.vertices[i].firstarc; 162
G.vertices[i].firstarc = p; 163
164
p = (ArcNode * )malloc(sizeof(ArcNode)); 165
p->adjvex = i; 166
p->nextarc = G.vertices[j].firstarc; 167
G.vertices[j].firstarc = p; 168
169
G.arcnum++; 170
return OK; 171
} 172
173
/*在图G中删除弧<v,w>, 成功返回OK, 否则返回ERROR*/ 174
status DeleteArc(ALGraph &G,KeyType v,KeyType w) 175
{
    int i = LocateVex(G, v), j = LocateVex(G, w); 177
    if(i<0||j<0) return ERROR; 178
    int flag1 = 0, flag2 = 0;
    ArcNode * p = G.vertices[i].firstarc; 180
    if(G.vertices[i].firstarc->adjvex == j){ 181
        ArcNode * s = G.vertices[i].firstarc;
        G.vertices[i].firstarc = G.vertices[i].firstarc->nextarc; 182
        free(s); 183
        flag1 = 1; 184
    } 185
    else while(p) { 186
        if(p->nextarc && p->nextarc->adjvex == j) { 188
            ArcNode * s = p->nextarc;
            p->nextarc = p->nextarc->nextarc; 189
            free(s); 190
            flag1 = 1; 191
        } 192
        p = p->nextarc; 193
    } 194
    p = G.vertices[j].firstarc; 195
    if(G.vertices[j].firstarc->adjvex == i) { 197
        ArcNode * s = G.vertices[j].firstarc;
        G.vertices[j].firstarc = G.vertices[j].firstarc->nextarc; 199
        free(s); 200
        flag2 = 1; 201
    } 202
```

# 华中科技大学课程实验报告

```
 }else while(p){  
    if(p->nextarc && p->nextarc->adjvex == i) {  
        ArcNode * s = p->nextarc;  
        p->nextarc = p->nextarc->nextarc;  
        free(s);  
        flag2 = 1;  
    }  
    p = p->nextarc;  
}  
  
G.arcnum--;  
if(flag1&&flag2) return OK;  
else return ERROR;  
}  
  
void visit(VertexType v) {  
    printf("(%d, %s)", v.key, v.others);  
}  
  
void VoidVisit(VertexType v) {  
    return;  
}  
  
void dfs(ALGraph &G, int record[], int i, void (*visit)(VertexType)){  
    visit(G.vertices[i].data);  
    record[i] = 1;  
    ArcNode * p = G.vertices[i].firstarc;  
    while(p){  
        if(record[p->adjvex]==0){  
            dfs(G,record,p->adjvex,visit);  
        }  
        p = p->nextarc;  
    }  
}  
  
/*对图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次*/  
status DFSTraverse(ALGraph &G, void (*visit)(VertexType))  
{  
    int record[MAX_VERTEX_NUM];  
    memset(record, 0, sizeof(record));  
    int i=0;
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
while(i<G.vexnum){  
    if(record[i]==0) {dfs(G, record, i, visit);printf("\n");}  
    i++;  
}  
return OK;  
}  
  
int Degree(VNode v){  
    ArcNode * p = v.firstarc;  
    int k = 0;  
    while (p) {  
        k++;  
        p = p->nextarc;  
    }  
    return k;  
}  
  
/*对图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次*/  
status BFSTraverse(ALGraph &G,void (*visit)(VertexType))  
{  
    int record[MAX_VERTEX_NUM];  
    memset(record, 0, sizeof(record));  
    std::queue<int> q;  
    for(int i=0; i<G.vexnum; i++){  
        if(record[i]==0){  
            q.push(i); // 开始  
            record[i] = 1;  
            while(!q.empty()) {  
                visit(G.vertices[q.front()].data);  
                ArcNode * p = G.vertices[q.front()].firstarc;  
                q.pop();  
                while (p) {  
                    if(!record[p->adjvex]) q.push(p->adjvex);  
                    record[p->adjvex] = 1;  
                    p = p->nextarc;  
                }  
            }  
        }  
        else;  
    }  
    return 0;  
}
```

# 华中科技大学课程实验报告

```
}

/*将图的数据写入到文件FileName中*/
status SaveGraph(ALGraph G, char FileName[])
{
    FILE * fp = fopen(FileName, "w");
    if(!fp) return ERROR;
    fprintf(fp, "%d %d\n", G.vexnum, G.arcnum);
    for(int i=0; i<G.vexnum; i++){
        fprintf(fp, "%d %s\n", G.vertices[i].data.key, G.vertices[i].data.others);
        fprintf(fp, "%d ", Degree(G.vertices[i]));
        ArcNode * p = G.vertices[i].firstarc;
        while (p) {
            fprintf(fp, "%d ", p->adjvex);
            p = p->nextarc;
        }
        fprintf(fp, "\n");
    }
    fclose(fp);
    return OK;
}

/*读入文件FileName的图数据，创建图的邻接表*/
status LoadGraph(ALGraph &G, char FileName[])
{
    FILE * fp = fopen(FileName, "r");
    if(!fp) return ERROR;
    fscanf(fp, "%d %d", &G.vexnum, &G.arcnum);
    for(int i=0; i<G.vexnum; i++){
        fscanf(fp, "%d %s", &G.vertices[i].data.key, G.vertices[i].data.others);
        int d;
        fscanf(fp, "%d", &d);
        if(d==0) {G.vertices[i].firstarc = NULL; continue;}
        else{
            G.vertices[i].firstarc = (ArcNode*) malloc(sizeof(ArcNode));
            fscanf(fp, "%d", &G.vertices[i].firstarc->adjvex);
            ArcNode * p = G.vertices[i].firstarc;
            for(int j=1; j<d; j++){
                p->nextarc = (ArcNode*) malloc(sizeof(ArcNode));
                p = p->nextarc;
            }
        }
    }
}
```

# 华中科技大学课程实验报告

```
fscanf(fp, "%d", &p->nextarc->adjvex); 326
    p = p->nextarc; 327
}
p->nextarc = NULL; 328
}
}
fclose(fp); 329
return OK; 330
}

/*
根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR 331
如果有相同的关键字，返回ERROR。此题允许通过增加其它函数辅助实现本关任务 332
*/
status CreateGraph(ALGraph &G, VertexType V[], KeyType VR[] [2]) { 333
    if(V[0].key == -1) return ERROR; 334
    for(int i=0; V[i].key != -1; i++) 335
        for (int j=i+1; V[j].key != -1; j++) 336
            if(V[i].key == V[j].key) return ERROR;
    memset(&G.vertices, 0, sizeof(G.vertices)); 337
    G.kind = UDG; 338
    G.arcnum = 0; 339
    G.vexnum = 0; 340
    int i=0; 341
    while(V[i].key != -1) { 342
        G.vertices[i].data = V[i]; G.vexnum++; i++; 343
    } 344
    if(G.vexnum>MAX_VERTEX_NUM) return ERROR; 345
    i=0;
    while(VR[i][0] != -1) { 346
        G.arcnum++; 347
        int k=0, j=0;
        while(G.vertices[j].data.key != VR[i][0]){
            j++; 348
            if(j >= G.vexnum){ 349
                return ERROR; 350
            }
        } 351
        while(G.vertices[k].data.key != VR[i][1]){
            k++; 352
            if(k >= G.vexnum){ 353
                return ERROR; 354
            }
        } 355
    } 356
}
```

# 华中科技大学课程实验报告

```
        return ERROR;
    }
}

if(G.vertices[j].firstarc != NULL){
    ArcNode * p;
    p = (ArcNode *) malloc(sizeof(ArcNode));
    p->adjvex = k;
    p->nextarc = G.vertices[j].firstarc;
    G.vertices[j].firstarc = p;
}
else {
    G.vertices[j].firstarc = (ArcNode * ) malloc(sizeof(ArcNode));
    G.vertices[j].firstarc->adjvex = k;
    G.vertices[j].firstarc->nextarc = NULL;
}

if(G.vertices[k].firstarc != NULL){
    ArcNode * p;
    p = (ArcNode *) malloc(sizeof(ArcNode));
    p->adjvex = j;
    p->nextarc = G.vertices[k].firstarc;
    G.vertices[k].firstarc = p;
}
else {
    G.vertices[k].firstarc = (ArcNode * ) malloc(sizeof(ArcNode));
    G.vertices[k].firstarc->adjvex = j;
    G.vertices[k].firstarc->nextarc = NULL;
}

i++;
}

return OK;
}

/*销毁无向图G,删除G的全部顶点和边*/
status DestroyGraph(ALGraph &G) {
    for(int i=0; i<G.vexnum; i++) {
        ArcNode * p = G.vertices[i].firstarc;
        while(p){
            ArcNode * q = p;
            p = p->nextarc;
            free(q);
        }
    }
}
```

# 华中科技大学课程实验报告

```
        }
    }
    memset(&G, 0, sizeof(G));
    return OK;
}

int LocateGraph(GRAPHS Graphs, char name[]) {
    int k = 0;
    while(k<Graphs.length){
        if(strcmp(Graphs.elem[k].name, name)==0) return k;
        k++;
    }
    return -1;
}

unsigned int Targetdfs(ALGraph &G, int record[], int i, int target){
    record[i] = 1;
    ArcNode * p = G.vertices[i].firstarc;
    unsigned int min = UINT_MAX;
    while(p){
        if(record[p->adjvex]==0){
            if(p->adjvex == target) return 1;
            else min = fmin(Targetdfs(G,record,p->adjvex,target), min);
        }
        p = p->nextarc;
    }
    record[i] = 0;
    return min==UINT_MAX? min : min + 1;
}

unsigned int ShortestPathLength(ALGraph G, KeyType u,KeyType v){
    int i = LocateVex(G, u);
    int j = LocateVex(G, v);
    int record[MAX_VERTEX_NUM];
    memset(record, 0, sizeof(record));
    if(u==v) return 0;
    return Targetdfs(G, record, i, j);
}
```

# 华中科技大学课程实验报告

```
int ConnectedComponentsNums(ALGraph G){  
    int record[MAX_VERTEX_NUM];  
    int k = 0;  
    memset(record, 0, sizeof(record));  
    int i = 0;  
    while(i<G.vexnum){  
        if(record[i]==0) {dfs(G, record, i, VoidVisit); k++;}  
        i++;  
    }  
    return k;  
}  
  
std::vector<VertexType> VerticesSetLessThanK(ALGraph G, KeyType u, unsigned int k)  
{  
    std::vector<VertexType> vec;  
    for(int i=0; i<G.vexnum; i++){  
        if(G.vertices[i].data.key==u) continue;  
        if(ShortestPathLength(G,u,G.vertices[i].data.key) < k) {  
            // 操作结果是返回与顶点v距离**小于**k的顶点集合  
            vec.push_back(G.vertices[i].data);  
        }  
    }  
    return vec;  
}  
  
int main(void)  
{  
    for(int i=0;i<5;i++) std::cout<<std::endl;  
    std::cout<<"*****图****实****验*****"<<std::endl;  
    for(int i=0;i<5;i++) std::cout<<std::endl;  
    int op=1, identity=0;  
    GRAPHS Graphs;  
    while(op){  
        if(Graphs.length==0) {  
            printf("图管理表为空, 请先添加无向图! \n");  
            op = 17;  
        }  
        else  
        {  
            printf("\n\n");  
            printf("          Menu for Graph          \n");  
        }  
    }  
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
printf("-----\n");
printf("基础功能: \n");
printf("1. CreatGraph           2. DestroyGraph\n");
printf("3. LocateVex            4. PutVex\n");
printf("5. FirstAdjVex          6. NextAdjVex \n");
printf("7. InsertVex            8. DeleteVex\n");
printf("9. InsertArc             10. DeleteArc\n");
printf("11. DFSTraverse          12. BFSTraverse\n");
printf("附加功能: \n");
printf("13. VerticesSetLessThanK   14. ShortestPathLength\n");
printf("15. ConnectedComponentsNums \n");
printf("多无向图管理: \n");
printf("16. RemoveGraph          17. AddGraph\n");
printf("18. SwitchGraph          19. LocateGraph\n");
printf("20. SaveGraphToFile       21. LoadGraphFromFile(DestroyGraph will
be called)\n");
printf("22. ShowGraphs           \n退出: \n0. Exit\n");
printf("-----\n");
printf("    活动无向图: %s\n", Graphs.elem[identy].name);
printf("    请选择你的操作[0~22]:\n");
scanf("%d", &op);
}

ALGraph & G = Graphs.elem[identy].G;
switch (op) {
case 1:{
    int vexnum=0, arcnum=0;
    VertexType V[MAX_VERTEX_NUM];
    KeyType Arcs[MAX_VERTEX_NUM][2];
    if(G.vexnum) {printf("活动无向图已有顶点! \n");getchar();getchar();break;}
    // 退出switch
    /* VERTICES */
    printf("请输入顶点定义: \n");
    while(1){
        scanf("%d %s", &V[vexnum].key, V[vexnum].others);
        if(V[vexnum].key == -1) break;
        vexnum++;
    }
    /* ARCS */
    printf("请输入边定义: \n");
    while(1){
        scanf("%d %d", &Arcs[arcnum][0], &Arcs[arcnum][1]);
        if(Arcs[arcnum][0] == -1) break;
        arcnum++;
    }
}
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
    if(Arcs[arcnum][0] == -1) break;
    arcnum++;
}
if(vexnum==0) printf("图中无顶点! \n");
/* END */
switch(CreateGraph(G, V, Arcs)){
    case ERROR:
    {
        printf("创建无向图失败, 请检查数据。 \n");
        break;
    }
    case OK:
    {
        printf("创建无向图成功\n");
        break;
    }
}
getchar();getchar();
break;
}
case 2:{549
    DestroyGraph(G);
    printf("图销毁成功! \n");
    getchar();getchar();
    break;
}
case 3:{555
    KeyType u;
    printf("请输入要查找的关键字: \n");
    scanf("%d", &u);
    int i = LocateVex(G, u);
    if(i<0){
        printf("未能查找到顶点! \n");
    }else{562
        printf("关键字%d的位序是%d, 内容是%s\n", u,i,G.vertices[i].data.others);
    }
    getchar();getchar();
    break;
}
case 4:{568
    KeyType u;
```

# 华中科技大学课程实验报告

```
VertexType value; 570
printf("请输入要赋值的顶点关键字: \n");
scanf("%d", &u); 571
printf("请输入要赋值关键字: \n");
scanf("%d", &value.key); 572
printf("请输入要赋值内容: \n");
scanf("%s", value.others); 573
if(PutVex(G, u, value)==ERROR){ 574
    printf("赋值失败! \n");
} else{ 575
    printf("赋值成功! \n");
}
getchar();getchar(); 576
break; 577
}
case 5:{ 578
    KeyType u;
    printf("请输入目标关键字: \n"); 579
    scanf("%d", &u);
    int i = FirstAdjVex(G, u); 580
    if(i==-1) printf("查找失败! \n");
    else printf("关键字%d的第一邻接顶点是key:%d,others:%s\n",u,G.vertices[i]. 581
        data.key,G.vertices[i].data.others);
    getchar();getchar(); 582
    break; 583
}
case 6:{ 584
    KeyType v,w;
    printf("查找v的邻接顶点中w的下一邻接点, 请输入v, w:\n"); 585
    scanf("%d %d",&v,&w);
    int i = NextAdjVex(G, v, w); 586
    if(i==-1) printf("查找失败! \n");
    else printf("关键字%d的下一邻接顶点是key:%d,others:%s\n",w,G.vertices[i]. 587
        data.key,G.vertices[i].data.others);
    getchar();getchar(); 588
    break; 589
}
case 7:{ 590
    VertexType v;
    printf("请输入要插入的顶点关键字: \n");
    scanf("%d",&v.key); 591
    592
    593
}
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
printf("请输入要插入的顶点内容: \n");
scanf("%s",v.others);
if(InsertVex(G, v) == OK){
    printf("插入顶点成功! \n");
}
else{
{
    printf("插入顶点失败! \n");
}
getchar();getchar();
break;
}
case 8:{619
    int key;
    printf("请输入要删除的顶点关键字: \n");
    scanf("%d",&key);
    if(DeleteVex(G, key) == OK){
        printf("删除顶点成功! \n");
    }
    else{
{
    printf("删除顶点失败! \n");
}
getchar();getchar();
break;
}
case 9:{634
    KeyType v,w;
    printf("插入边<v,w>, 请输入v, w:\n");
    scanf("%d %d",&v,&w);
    if(InsertArc(G, v, w)==OK){
        printf("插入边成功! \n");
    }else{
        printf("插入边失败! \n");
}
getchar();getchar();
break;
}
case 10:{646
    KeyType v,w;
    printf("删除边<v,w>, 请输入v, w:\n");
647
648
649
```

# 华中科技大学课程实验报告

```
scanf("%d %d", &v, &w);
if(DeleteArc(G, v, w)==OK){
    printf("删除边成功! \n");
}else{
    printf("删除边失败! \n");
}
getchar();getchar();
break;
}

case 11:{
    DFSTraverse(G, &visit);
    getchar();getchar();
    break;
}
case 12:{
    BFSTraverse(G, &visit);
    getchar();getchar();
    break;
}
case 13:{
    int u,k;
    printf("请输入关键字和距离: \n");
    scanf("%d %d", &u, &k);
    if(LocateVex(G, u)==-1) {printf("操作失败! \n"); getchar();getchar();break
    ;}
    std::vector<VertexType> set = VerticesSetLessThanK(G, u, k);
    for(auto p : set){
        visit(p);
    }
    getchar();getchar();
    break;
}
case 14:{
    int u,v;
    printf("请输入关键字u,v: \n");
    scanf("%d %d", &u, &v);
    if(LocateVex(G, u)==-1 || LocateVex(G, v)==-1){
        printf("求距离失败! \n");
        getchar();
        getchar();
        break;
    }
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
    }
    unsigned int spl = ShortestPathLength(G, u, v);
    if(spl == UINT_MAX) printf("INFINITE\n");
    else printf("\n%d\n", spl);
    getchar();getchar();
    break;
}
case 15:{

    int ccn = ConnectedComponentsNums(G);
    printf("\n%d\n", ccn);
    getchar();getchar();
    break;
}
case 16:{

    printf("请输入要移除的图名称: \n");
    char name[30];
    scanf("%s", name);
    int RmNum = LocateGraph(Graphs, name);
    if(RmNum<0) {
        printf("找不到图%s", name);
        getchar();getchar();
        break;
    }
    char nowName[30];
    strcpy(nowName, Graphs.elem[identy].name);
    for (int i = RmNum; i < Graphs.length; i++) {
        Graphs.elem[i] = Graphs.elem[i+1];
    }
    Graphs.length--;
    int newidentity;
    if((newidentity = LocateGraph(Graphs, nowName)) >= 0)
        identy = newidentity;
    else
        /*Do Nothing.*/
    getchar();getchar();
    break;
}
case 17:{

    char name[20];
    printf("请输入要添加的图名称: \n");
}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
scanf("%s", name);                                731
if(LocateGraph(Graphs, name)!=-1) {                732
    printf("无向图的名称不能重复！添加失败。\\n");      733
    getchar();getchar();                            734
    break;                                         735
}else{
    strcpy(Graphs.elem[Graphs.length].name, name); 737
    Graphs.length++;                             738
    printf("添加图 %s 成功! \\n", Graphs.elem[Graphs.length-1].name); 739
    getchar();getchar();                            740
    break;                                         741
}
}                                                 742
743
case 18:{                                       744
    char name[30];                                745
    int j;                                         746
    printf("请输入要切换的图名称: \\n");           747
    scanf("%s",name);                            748
    if((j=LocateGraph(Graphs,name))>=0) {
        identy = j;
        printf("切换成功! \\n");                  751
    }
}                                                 752
753
else{
    printf("切换失败! \\n");                      754
}
getchar();getchar();                            756
break;                                         757
}                                                 758
759
case 19:{                                       760
    char name[30];
    int j;
    printf("请输入要查找的图名称: \\n");
    scanf("%s",name);
    if((j=LocateGraph(Graphs,name))>=0){
        printf("图%s在第%d位, 有%d个顶点, %d条边\\n",name,j,Graphs.elem[j].G.
            vexnum,Graphs.elem[j].G.arcnum);
    }
}                                                 766
767
else{
    printf("查找失败! \\n");
}
getchar();getchar();                            769
break;                                         770
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
}

case 20:{

    char name[128];

    printf("保存到文件, 请输入文件名: \n");
    scanf("%s",name);

    if(SaveGraph(G, name)==OK)

    {

        printf("\n保存成功! \n");
    }

    else

    {

        printf("\n保存失败! \n");
    }

    getchar();getchar();

    break;

}

case 21:{

    char name[128];

    printf("从文件加载, 请输入文件名: \n");
    scanf("%s",name);

    if(LoadGraph(G, name)==OK)

    {

        printf("\n加载成功! \n");
    }

    else

    {

        printf("\n加载失败! \n");
    }

    getchar();getchar();

    break;

}

case 22:

{

    printf("管理表中的图: ");

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

        printf("\tGraph%d : %s",i,Graphs.elem[i].name);

    }

    printf("\n");

    getchar();getchar();

    break;

}
```

# 华 中 科 技 大 学 课 程 实 验 报 告

```
    default:  
        break;  
    } // end switch  
} // end while  
printf("欢迎下次再使用本系统! \n"); // op==0  
}
```

812  
813  
814  
815  
816  
817