

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期： 2021年 5月 7 日**

**计算机科学与技术学院**

目 录

[1 基于顺序存储结构的线性表实现 3](#_Toc458159879)

[1.1 问题描述 3](#_Toc458159880)

[1.2 系统设计 3](#_Toc458159882)

[1.3 系统实现 6](#_Toc458159883)

[1.4 系统测试 1](#_Toc458159884)1

[1.5 实验小结 1](#_Toc458159884)4

[2 基于链式存储结构的线性表实现 1](#_Toc458159885)5

[2.1 问题描述 1](#_Toc458159886)5

[2.2 系统设计 1](#_Toc458159887)5

[2.3 系统实现 1](#_Toc458159888)7

[2.4 系统测试 2](#_Toc458159889)1

[2.5 实验小结 2](#_Toc458159889)4

[3 基于二叉链表的二叉树实现 2](#_Toc458159890)5

[3.1 问题描述 2](#_Toc458159891)5

[3.2 系统设计 2](#_Toc458159892)5

[3.3 系统实现 2](#_Toc458159893)7

[3.4 系统测试 3](#_Toc458159893)5

[3.5 实验小结 3](#_Toc458159894)9

[4 基于邻接表图的二叉树实现 2](#_Toc458159895)

[4.1 问题描述 2](#_Toc458159896)

[4.2 系统设计 2](#_Toc458159897)

[4.3 系统实现 2](#_Toc458159898)

[4.4 实验小结 2](#_Toc458159899)

[参考文献 2](#_Toc458159900)

[附录A 基于顺序存储结构线性表实现的源程序 2](#_Toc458159901)5

[附录B 基于链式存储结构线性表实现的源程序 4](#_Toc458159902)4

[附录C 基于二叉链表二叉树实现的源程序 7](#_Toc458159903)5

[附录D 基于邻接表图实现的源程序 2](#_Toc458159904)

**1 基于顺序存储结构的线性表实现**

**1.1问题描述**

通过实验达到

⑴加深对线性表的概念、基本运算的理解；

⑵熟练掌握线性表的逻辑结构与物理结构的关系；

⑶物理结构采用顺序表,熟练掌握线性表的基本运算的实现。

**1.2系统设计**

**1.2.1整体结构设计**

实现顺序储存的线性表的基本运算操作。通过switch语句根据菜单系统上的提示内容输入数字进行对应线性表操作，包括：创建，销毁，清空，判空，求表长，获取元素，查找元素，获得前驱，获得后继，插入元素，删除元素，遍历线性表，数据读写文件，多线性表管理（增加一个新线性表，移除一个线性表，查找线性表）及相关附加功能：排序、判断子表、求两表的交集、并集、差集。

**1.2.2数据结构设计**

函数设计：

**⑴初始化表：函数名称是InitList(L)；**

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

**⑵销毁表：函数名称是DestroyList(L)**；

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

**⑶清空表：函数名称是ClearList(L)；**

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

**⑷判定空表：函数名称是ListEmpty(L)；**

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

**⑸求表长：函数名称是ListLength(L)；**

如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

**⑹获得元素：函数名称是GetElem(L,i,e)；**

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

**⑺查找元素：函数名称是LocateElem(L,e,compare())；**

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

**⑻获得前驱：函数名称是PriorElem(L,cur\_e,pre\_e)；**

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

**⑼获得后继：函数名称是NextElem(L,cur\_e,next\_e)；**

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

**⑽插入元素：函数名称是ListInsert(L,i,e)**；

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

**⑾删除元素：函数名称是ListDelete(L,i,e)；**

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

**⑿遍历表：函数名称是ListTraverse(L,visit())；**

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

1. **将数据元素写入文件：函数名称是SaveList(L,FileName[]）；**

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

1. **将文件数据写入线性表：函数名称是LoadList（L，FileName[]）；**

如果线性表L存在，就将FileName文件中的数据写入线性表中，返回OK。否则返回INFEASIBLE；

1. **添加线性表：函数名称是AddList（Lists，ListName[]）;**

只需要在Lists中增加一个名称为ListName的线性表,并返回OK，如果添加的线性表个数超过数组范围，返回ERROR。

1. **删除线性表：函数名称是RemoveList（Lists，ListName[]）；**

在Lists中删除一个名称为ListName的线性表；如果不存在该线性表，返回ERROR；删除成功返回OK。

1. **查找线性表：函数名称是LocateList（Lists，ListName[]）；**

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

**下面是附加功能：**

1. **对线性表进行升序排序：函数名称是UpList（L）；**

如果线性表存在，就对线性表进行升序排序并返回OK；否则返回INFEASIBLE.

1. **对线性表进行降序排列：函数名称是DownList（L）；**

如果线性表存在，就对线性表进行降序排列并返回OK，否则返回INFEASIBLE.

1. **判断子表：函数名称为SubList（L,S);**

如果L和S两个线性表都存在，判断S是不是L的子集；否则返回INFEASIBLE。

1. **求交集：函数名称是InterList（L，S,p）**

如果L，S，p三个线性表都存在，将L与S 两个顺序线性表的共同元素保存在p线性表中；否则返回INFEASIBLE。

1. **求两线性表的并集：函数名称是SumList（L,S,p）；**

如果L，S，p三个线性表都存在，将L与S 两个顺序线性表的元素保存在p线性表中，保存后p也为顺序线性表；否则返回INFEASIBLE。

1. **求差集；函数名称为DifferList（L,S,p）；**

如果L，S，p三个线性表都存在，将L与S 两个顺序线性表的不存在于另一个线性表中的元素保存在p线性表中，保存后p也为顺序线性表；否则返回INFEASIBLE。

**1.3系统实现**

函数实现：

**（1）初始化表：函数名称是InitList(L)；**

如果存在线性表，如果再构造线性表，会覆盖掉原先的元素，所以返回INFEASIBLE；

如果该线性表之前不存在，调用malloc函数给线性表分配空间，如果分配失败，就返回INFEASIBLE；如果分配成功，就对线性表进行初始化，length初始化为0，listsize初始化为前面已经定义的最大值。

**（2）销毁表：函数名称是DestroyList(L)；**

如果线性表不存在，无法进行销毁操作，返回INFEASIBLE；

如果线性表存在，首先清空线性表，即L.length=0;然后free函数释放线性表所分配的内存。重点：free掉以后一定要将指针指向NULL，不然野指针会出问题！

**（3）清空表：函数名称是ClearList(L)；**

如果线性表不存在，无法进行清空操作，就返回INFEASIBLE；

如果线性表存在，就将线性表清空，即将线性表的长度的长度变为0，即L.length=0;并返回OK以表示线性表被清空成功。

**（4）判定空表：函数名称是ListEmpty(L)；**

如果线性表不存在，无法进行判空操作，就返回INFEASIBLE；

如果线性表存在，判断线性表是否为空，如果L.length=0，那么线性表为空，返回TRUE，如果L.length不等于0，那么线性表不为空，返回FALSE。

**（5）求表长：函数名称是ListLength(L)；**

如果线性表不存在，无法进行求表长的操作，就返回INFEASIBLE；

如果线性表存在，就将L.length作为返回值将表长返回。

**（6）获得元素：函数名称是GetElem(L,i,e)；**

如果线性表不存在，无法进行获取元素的操作，返回INFEASIBLE；

如果线性表已存在，判断i值是否合法，如果i<=0或者i>L.length,线性表中只有L.length个元素，此时无法获取到第i个元素，返回ERROR； 如果i>0并且i<=L.length，那么就将elem数组中的第i个元素赋值给e并且返回OK以表示函数结束。

**（7）查找元素：函数名称是LocateElem(L,e,compare())；**

如果线性表不存在，无法进行查找元素的操作，返回INFEASIBLE；

如果线性表已存在，对线性表进行遍历，从第一个元素开始查找，如果有与e相等的元素，就直接返回该元素所处的位置，如果遍历结束还没有e元素，就返回0.

**（8）获得前驱：函数名称是PriorElem(L,cur\_e,pre\_e)；**

如果线性表不存在，无法进行查找元素的操作，返回INFEASIBLE；

如果线性表已存在，对线性表中具有前驱的元素进行遍历，从第一个具有前驱的元素开始查找，即第二个元素到最后一个元素，如果有与e相等的元素，就直接返回该元素的前驱所处的位置，如果遍历结束还没有e元素，就返回0.

1. **获得后继：函数名称是NextElem（L,E,Next）；**

如果线性表不存在，无法进行查找元素的操作，返回INFEASIBLE；

如果线性表已存在，对线性表中具有后继的元素进行遍历，从第一个具有后继的元素开始查找，即第一个元素到倒数第二个元素，如果有与e相等的元素，就直接返回该元素的后继所处的位置，如果遍历结束还没有e元素，就返回0.

1. **插入元素：函数名称是ListInsert(L,i,e)；**

如果线性表不存在，无法进行获取元素的操作，返回INFEASIBLE；

如果线性表已存在，判断i值是否合法，如果i<=0或者i>L.length+1,线性表中只有L.length个元素，插入的元素只能在第1到L.length+1的元素前面插入，其中L.length+1为在最后一个位置插入元素，此时i值不合法，返回ERROR；

如果i值合法，判断线性表是否已满，如果已满，调用realloc函数对线性表的内存重新分配，如果分配失败，返回OVERFLOW；

如果成功或者本身线性表未满，从第i个元素开始，之后的元素每个都后移一位然后将e元素插入第i位，最后使线性表长度加一更新值。

1. **删除元素：函数名称是ListDelete(L,i,e)；**

如果线性表不存在，无法进行删除元素的操作，返回INFEASIBLE；

如果线性表已存在，判断i值是否合法，如果i<=0或者i>L.length,线性表中只有L.length个元素，此时无法找到并删除第i个元素，返回ERROR；

如果i>0并且i<=L.length，那么就将elem数组中的第i个元素删除，删除操作为，从第i+1位值开始，到最后一位，都往前移一位以删除该值，并且使线性表长度减一，最后返回OK以表示函数结束。

1. **遍历表：函数名称是ListTraverse(L,visit())；**

如果线性表不存在，无法进行删除元素的操作，返回INFEASIBLE；

如果线性表已存在，遍历线性表，从第一个元素开始到最后一个，逐个输出，注意空格的输出最后一个元素后面不需要输出空格。

1. **将数据元素写入文件：函数名称是SaveList(L,FileName[]）；**

如果线性表不存在，无法进行元素读入文件的操作，返回INFEASIBLE； 如果线性表已存在，首先以“w”的方式打开名为filename的文件，如果打开失败，直接返回ERROR；

打开成功后，开始遍历线性表，从第一个元素开始，逐个向文件中写入数据，调用fprintf函数；遍历完成后关闭文件。

注意，每次写入文件时，数据与数据之间应该有间隔，不能将数据直接全部输入，不然读取文件的时候，无法读取到单个的数据。

1. **将文件数据写入线性表：函数名称是LoadList（L，FileName[]）；**

如果线性表不存在，无法进行文件元素写入线性表的操作，返回INFEASIBLE；

如果线性表已存在，首先以“r”的方式打开名为filename的文件，如果打开失败，直接返回ERROR；

打开成功后，开始遍历文件，从第一个元素开始，逐个向文件中写入数据，调用fscanf函数；然后调用ListInsert函数将读出的元素插入到线性表的末尾中，遍历完成后关闭文件。

1. **添加线性表：函数名称是AddList（Lists，ListName[]）;**

如果多线性表集里面的线性表个数已达到10，已无法再插入线性表，因为这个数组使静态的；

如果数组未满，就向多线性表中插入一个线性表，首先将ListName复制到新增的线性表的名称中去，while循环遍历到名称结尾进行赋值来复制。

调用InitList函数对最后一个线性表后面的数组元素中的线性表进行初始化，然后调用ListInsert函数向该线性表中插入元素，线性表加入成功后，多线性表的长度加一。

1. **删除线性表：函数名称是RemoveList（Lists，ListName[]）；**

首先遍历多线性表，查看是否有名称为ListName的线性表，采用while循环遍历查看线性表名称与指定名称是否相等，如果有不相等的元素，直接break表示不相等，如果两字符串都到达了结尾，说明名称相等，

然后进行删除线性表操作；即从该元素下一个开始，到最后一个，每个元素都前移一位，然后销毁最后一个线性表，并且使多线性表的长度减一。

1. **查找线性表：函数名称是LocateList（Lists，ListName[]）；**

遍历多线性表，查看是否有名称为ListName的线性表，采用while循环遍历查看线性表名称与指定名称是否相等，如果有不相等的元素，直接break表示不相等，如果两字符串都到达了结尾，说明名称相等，此时直接可返回该线性表的位置，如果遍历完整个多线性表没有相等的名称，就返回0以表示无。

**附加功能：**

1. **对线性表进行升序排序：函数名称是UpList（L）；**

如果线性表不存在，无法进行对元素排序的操作，返回INFEASIBLE；如果线性表已存在，采用冒泡排序法排序。

1. **对线性表进行降序排列：函数名称是DownList（L）；**

如果线性表不存在，无法进行对元素排序的操作，返回INFEASIBLE；如果线性表已存在，采用冒泡排序法排序。

1. **判断子表：函数名称为SubList（L,S);**

如果俩线性表有一个不存在，就无法进行比较操作，返回INFEASIBLE；

如果两线性表都存在，判断S是否为L的一个子集，遍历S中的元素，看是否在L这个线性表中存在，即可调用LocateList函数判断该元素是否在L这个线性表中，如果遍历过程中，有元素不在L中，直接返回ERROR以表示不是子集，如果遍历结束,S的元素在L 中都出现了，那么S是L的一个子集。

1. **求交集：函数名称是InterList（L，S,p）；**

如果俩线性表有一个不存在，就无法进行比较操作，返回INFEASIBLE；

如果两线性表都存在，遍历两个线性表，由于是顺序表，则可以通过比较得到相等的元素，

首先对L中的元素进行过滤，将比当前的S所拿出的元素小的元素都过滤掉，即i++，之后再过滤S中的元素，遇到相等的元素，就将该元素插入到p线性表中，调用ListInsert函数，直到遍历到其中一个线性表到达结尾结束。

1. **求两线性表的并集：函数名称是SumList（L,S,p）；**

如果俩线性表有一个不存在，就无法进行比较操作，返回INFEASIBLE；

如果两线性表都存在，开始求并集。俩顺序表的并集也是顺序表，

首先令i，j表示当前遍历到的L,S的线性表的位置，首先插入比较小的元素，先遍历L，插入比当前的S中的第j位小的元素，即i++；之后再遍历S，插入比当前的L中的第i位小的元素，即j++；如果遇到两线性表中相等的元素，插入该元素并且需要i++，j++；直到两个线性表都结束。

**（23）求差集；函数名称为DifferList（L,S,p）；**

如果俩线性表有一个不存在，就无法进行比较操作，返回INFEASIBLE；

如果两线性表都存在，开始求差集。俩顺序表的差集也是顺序表，

首先令i，j表示当前遍历到的L,S的线性表的位置，首先插入比较小的元素，先遍历L，插入比当前的S中的第j位小的元素，即i++；之后再遍历S，插入比当前的L中的第i位小的元素，即j++；如果遇到两线性表中相等的元素，不插入并且需要i++，j++；直到两个线性表都结束。

**1.4系统测试**

**1.4.1 系统测试目的**

通过运行每个程序功能来检测各个功能实现正确性，一方面检测正常数据是否能够正常处理，另一方面要检测异常数据处理并检验程序是否能够判断该数据不合法且给予提示，此过程程序能继续运行不受影响。

**1.4.2 系统测试结果**

表1-1

|  |  |  |  |
| --- | --- | --- | --- |
| 选择操作 | 输入 | 预计输出 | 实际输出 |
| 创建线性表 | 1 | 线性表创建成功 |  |
| 销毁线性表 | 2 | 线性表销毁成功 |  |
| 销毁线性表 | 2 | 线性表本身不存在，销毁失败 | Q~RYGF6}NH9(IQJX3P_U%85 |
| 创建线性表 | 1 | 线性表创建成功 |  |
| 清空线性表 | 3 | 线性表已清空 | Z{%(1FT[J3K~KE5XIH(FX(1 |
| 读入文件 | 14 | 文件数据读入线性表成功 | ~U29YGZRTC]LJWCR9RK6Y25 |
| 获取表长 | 5 | 线性表长度为13 | 6WPDFVIT]_CY]UV_ZI[[4CT |
| 获取元素 | 6 1 | 线性表第1个值是1 | HQ$)2AAL%ZB2AIKR~_17M9N |
| 遍历线性表 | 12 | 1 2 3 4 5 6 7 8 9 10 23 24 0 | )P2CW__66JZEUUR]4{30X}5 |
| 获取位置 | 7 10 | 元素10在线性表中的位置位10 | 06I@VBP(4T2%V8Q6V%DX[}B |
| 获得后继 | 9  9 | 线性表中元素9后继元素为10 | E}OSU0D[}$T9GRNZ%GA$RH3 |
| 获得前驱 | 8  9 | 线性表中元素9的前驱元素为8 | _242TZ@K85W6ZG0GRL)}CJF |
| 判空 | 4 | 线性表非空 | EIAK(Q8`A@@(K__G57M4TUL |
| 获取表长 | 5 | 线性表的长度为13 | V0U{47PYFEP(TT~Q0@`%5@X |
| 插入元素 | 10  5 11 | 插入成功 | T24WLD9F~S7RL~I_R7][Q)9 |
| 遍历 | 12 | 1 2 3 4 11 5 6 7 8 9 10 23 24 0 | Y]{HQP_XS[V}WOO2[L691DT |
| 删除元素 | 13  7 | 删除成功，删除元素为6！ | }A_QNW7802]{`BAPX7M`C0O |
| 遍历线性表 | 12 | 1 2 3 4 11 5 7 8 9 10 23 24 0 | MABKYEDQ4}9PGAI(PM1(FDI |
| 写入文件 | 13  data.txt | 写入文件成功 | P)9I2}L@H7ZKU1NB}DVHEWN |
| 添加线性表 | 15 | 给出多个线性表的遍历：  yi 1 6 9  Er 6 8 11 45  San 3 7 9 23 67 | G_TCP9F79QBW$2PQ%FOQE}B |
| 查找线性表 | 17  er | er线性表的位置为2 | }0TEXSBT4[7P[MGRZ${ZB0P |
| 查找线性表 | 17  www | 不存在该线性表 | O@`HU_YY4]NZ@JLRPYB4%NN |
| 线性表升序 | 18 | 升序后的线性表为： | _]HNN${[MUBWGH}~RHGE$TO |
| 降序排列 | 19 | 降序后的线性表：24 23 11 10 9 8 7 5 4 3 2 1 | L%0BP002MP`UX80])HNYPQ3 |
| 删除线性表 | 16  er | er线性表删除成功 | KB836(9}APRWAJVL]RCX}MC |
| 退出 | 0 | 欢迎下次再使用本系统 | %@]L{9K@5NPFEIB@2U8L1V1 |
| 添加线性表 | 15 | 给出多个线性表的遍历：  yi 5 8 9  Er 8 9 12 24  San 8 23 | 9J0WFIA}@GV7Y~Q]QEOL]US |
| 判断子集 | 20 | 二者不存在子集关系 | O_@NC{]Z`_@4W291_$KNI{A |
| 多线性表取交集 | 21 | 两线性表的交集为8 9 | L5Y6@FG52H8X~)B8%L}MX9K |
| 多线性表取差集 | 23 | 两线性表的差集为5 12 24 | W@Q%I`)RQ42G_{D]$54_3@C |
| 多线性表取并集 | 22 | 两线性表的并集是5 8 9 12 24 | R21MEG{L%YSH1SIQ5AT4O]N |
| 退出 | 0 | 欢迎下次再使用本系统 | %@]L{9K@5NPFEIB@2U8L1V1 |

**1.5实验小结**

本次实验是为了让我们对线性表有一个更深层次的理解，其中包含了一些对线性表的基本功能的实现，比如初始化线性表，删除插入元素等对单个线性表的操作以及对多线性表的操作。基础功能是我们必须要掌握的功能，我们应当熟练掌握，在实验过程中，我发现，在一些小细节上面需要多加注意，比如指针在初始时是否指为空，在释放过后是否指为空，如果不指为空，程序可能会出现不可避免的错误，在进行删除或者插入操作时，对指针的操作需要多加注意。另外一些length上的增减操作需要多加注意。

在附加功能中，我基于字符串的一些基础操作，想到了线性表的子集、交集、并集、差集等操作。这些功能不算复杂，也运用了一些基础功能的函数。所以，基础功能的函数是基础，我们需要熟练掌握，才能组合出更复杂的功能。

这次的线性表实验整体来说还是比较简单的，系统的功能并不复杂，代码也不是很复杂，但是也让我们第一次认识到了系统是如何实现的，以一种简单的方式开始实验，为以后打基础，以后还需要更多的努力。

**2 基于链式存储结构的线性表实现**

**2.1问题描述**

通过实验达到

⑴加深对线性表的概念、基本运算的理解；

⑵熟练掌握线性表的逻辑结构与物理结构的关系；

⑶物理结构采用单链表,熟练掌握线性表的基本运算的实现。

**2.2系统设计**

**2.2.1整体结构设计**

实现顺序储存的线性表的基本运算操作。通过switch语句根据菜单系统上的提示内容输入数字进行对应线性表操作，依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等12种基本运算，包括：创建，销毁，清空，判空，求表长，获取元素，查找元素，获得前驱，获得后继，插入元素，删除元素，遍历线性表，数据读写文件，及相关附加功能：多线性表管理（增加一个新线性表，移除一个线性表，查找线性表）、排序、逆置，归并。

**2.2.2数据结构设计**

线性表基本运算设计：

**(1)初始化表：函数名称是InitList(L)；**

初始条件是线性表L不存在；操作结果是构造一个空的线性表。

**(2)销毁表：函数名称是DestroyList(L)**；

初始条件是线性表L已存在；操作结果是销毁线性表L。

**(3)清空表：函数名称是ClearList(L)；**

初始条件是线性表L已存在；操作结果是将L重置为空表。

**(4)判定空表：函数名称是ListEmpty(L)；**

初始条件是线性表L已存在；操作结果是若L为空表则返回TRUE,否则返回FALSE。

**(5)求表长：函数名称是ListLength(L)；**

初始条件是线性表已存在；操作结果是返回L中数据元素的个数。

**(6)获得元素：函数名称是GetElem(L,i,e)；**

初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用e返回L中第i个数据元素的值。

**(7)查找元素：函数名称是LocateElem(L,e,compare())；**

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

**(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≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

**(11)删除元素：函数名称是ListDelete(L,i,e)；**

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

**(12)遍历表：函数名称是ListTraverse(L,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()。

**（13）将数据元素写入文件：函数名称是SaveList(L,FileName[]）；**

初始条件是线性表L存在并非空；操作结果：将线性表L的的元素写到FileName文件中。。

**（14）将文件数据写入线性表：函数名称是LoadList（L，FileName[]）；**

初始条件是线性表L；操作结果：将FileName文件中的数据写入线性表中。

**（15）添加线性表：函数名称是AddList（Lists，ListName[]）;**

初始条件是多线性表数组未满；操作结果：在多线性表中添加一个名称为listname的线性表。

**（16)删除线性表：函数名称是RemoveList（Lists，ListName[]）；**

初始条件是多线性表非空并且存在名称为listname的线性表；操作结果：删除多线性表中名称为listname的线性表。

**(17)查找线性表：函数名称是LocateList（Lists，ListName[]）；**

初始条件是多线性表非空并且存在名称为listname的线性表；操作结果：返回该线性表的位置序号。

**下面是附加功能：**

**(18)对线性表进行升序排序：函数名称是UpList（L）；**

初始条件为线性表存在；操作结果：线性表的顺序为升序。

**(19)对线性表进行降序排列：函数名称是DownList（L）；**

初始条件是线性表存在并非空；操作结果：线性表的顺序为降序。

1. **对线性表进行逆置：函数名称为ReverseList(L)；**

初始条件是线性表存在并非空；操作结果：线性表逆序排列。

1. **对两个线性表进行归并：函数名称为MergeList(L,b)；**

初始条件是多线性表中含有至少两个线性表并且为升序；操作结果：两个顺序线性表归并到第一个线性表中得到一个顺序线性表并且去除相同元素。

**2.3系统实现**

函数实现：

1. **InitList(LinkList &L):**

如果线性表存在，返回INFEASIBLE；

如果线性表不存在，对头结点分配空间；使L的next指针指空；返回OK。

**（2）DestroyList(LinkList &L)：**

如果线性表不存在，返回INFEASIBLE；

令a为当前要释放空间的结点，b为后继结点；

While循环释放当前结点a的空间，a，b结点不断后移，直到a为空；

最后对L赋值为空，防止野指针产生。

**（3）ClearList(LinkList &L)：**

如果线性表不存在，返回INFEASIBLE；

初始化a为首元结点，如果a为空，直接返回OK已清空；如果a不为空，令b为a的后继结点。

While循环，释放当前结点的空间，之后a，b两个结点都后移，直到a为空结束循环；

最后使头结点的next为空表示首元结点为空。

**（4）ListEmpty(LinkList L)**

如果线性表不存在，返回INFEASIBLE；

如果首元结点的next指针不为空，返回FASLE表示线性表非空；反之，返回OK，线性表为空。

**（5）ListLength(LinkList L)**

如果线性表不存在，返回INFEASIBLE；

初始化a为首元结点，len初始化为0。

While循环，判断条件为a是否为空，a空时结束循环；每次进入循环，len增一表示当前的结点含有元素，线性表长度加一；a指向下一节点再次进入循环。

返回len表示线性表的长度。

**（6）GetElem(LinkList L,int i,ElemType &e)**

如果线性表不存在，返回INFEASIBLE；

判断i值的合法性，若i<=0，位置序号的赋值不合法，返回ERROR；

初始化a为首元结点，len为当前元素所在位置；

While循环，判断条件为a是否为空，a为空时停止计数，线性表遍历结束；每次循环：len增一以表示当前结点所处的线性表位置；如果len＝i，用e储存该位置的元素并返回OK；a指向下一个节点后继续下一轮循环。

如果循环结束并未找到指定位置的值，表示i值输入的不合法，返回ERROR。

**（7）LocateElem(LinkList L,ElemType e)**

如果线性表不存在，返回INFEASIBLE；

初始化a为首元结点，i为0；while循环判断条件为a是否为空，即遍历结束；若a不为空，i++表示当前结点的位置，判断a->data是否等于e，如果相等，返回位置序号i，如果不相等，a指向下一个结点，进入下一个循环；

如果遍历结束没有相等的e，就返回ERROR表示不含有e元素。

**（8）PriorElem(LinkList L,ElemType e,ElemType &pre)**

如果线性表不存在，返回INFEASIBLE；

如果线性表为空或者只有一个首元结点，表示没有具有前驱元素的元素；返回ERROR；

初始化a为首元结点，b为第二个结点；即b是第一个有前驱的结点；

While循环，循环条件为b不为空；每次循环：判断b的元素是否为条件判断值，如果是，将b的前驱a的元素赋给e并返回ok；如果不是，a，b均指向各自的下一结点重新循环；

如果遍历结束无这样的b，返回ERROR表示该元素在线性表中无前驱。

**（9）NextElem(LinkList L,ElemType e,ElemType &next)**

如果线性表不存在，返回INFEASIBLE；

如果线性表为空或者只有一个首元结点，表示没有具有后继元素的元素；返回ERROR；

初始化a为首元结点；

While循环，循环条件为a的后继不为空；每次循环：判断a的元素是否为条件判断值，如果是，将a的后继a->next的元素赋给e并返回ok；如果不是，a指向各自的下一结点重新循环；

如果遍历结束无这样的a，返回ERROR表示该元素在线性表中无前驱。

**（10）ListInsert(LinkList &L,int i,ElemType e)**

如果线性表不存在，返回INFEASIBLE；

如果输入的插入位置小于1，必定不合法，返回ERROR；合法值为1-length+1；

初始化a为头结点，len为0；while循环，判断条件为a是否为空，若a不为空，进入循环；判断len是否等于i-1，插入到第i个元素之前也就是插入到第i-1个元素之后，若相等，即找到了第i-1个元素，此时为插入元素分配一个结点空间b，b->data=e，让b的next等于a的next，a的next指向b，完成插入，返回OK；若不想等，a指向下一个结点，len++；表示a的位置；

若遍历结束并为找到该位置，说明输入的i值不合法，返回ERROR；

1. **ListDelete(LinkList &L,int i,ElemType &e)**

如果线性表不存在，返回INFEASIBLE；

如果输入的删除位置小于1，必定不合法，返回ERROR；合法值为1-length；

初始化a为头结点，len为0；

while循环，判断条件为a->next是否为空，若a->next不为空，说明删除的位置合法，进入循环；判断len是否等于i-1，删除第i个元素，也就是让第i-1个元素的next记录第i+1个元素；如果相等，也就是删除该元素后面的元素，用e记录a->next->data，a->next指向下一个结点，然后释放a原本后面的结点的空间，并返回OK；如果不相等，a指向下一结点，len++表示a的位置；

如果遍历结束并没有这样的值，返回ERROR。

1. **ListTraverse(LinkList L)**

如果线性表不存在，返回INFEASIBLE；

初始化a为首元结点；

While循环，判断条件为a是否为空；输出a->data;判断a之后是否还有元素，如果有，输出空格，如果没有，不输出空格；a指向下一个结点，再次循环；

遍历完毕，返回ERROR。

1. **SaveList(LinkList L,char FileName[])**

如果线性表不存在，返回INFEASIBLE；

初始化a为首元结点，file为以“w”的方式打开，如果打开失败，返回ERROR；

While循环，判断a是否为空，fprintf函数将数据+一个空格写入文件；然后a指向下一个结点后继续循环；

遍历完毕后关闭文件指针并返回OK。

1. **LoadList(LinkList &L,char FileName[])**

如果L存在，调用ClearList函数清空L；如果L不存在，调用InitList函数初始化L；

初始化a为头结点，b为NULL；以“r”的方式打开文件，如果打开失败，返回ERROR；

While循环，判断条件是读取数据是否到达文件末尾；为b分配空间，数据域赋值从文件中读取的数据，指针域为空，a指针域更新为b，然后a，b指针均后移；直到达到文件末尾；

遍历结束，关闭文件，返回OK。

下面是附加功能：

1. **AddList(LISTS &lists,char ListName[])**

如果数组已满，无法再加入线性表，返回ERROR；

将要加入的线性表名字复制到多线性表最后一个线性表后面的线性表中去，调用InitList函数初始化线性表，然后向其中插入元素。

插入成功后，lists的长度增一，返回OK。

1. **RemoveList(LISTS &lists,char ListName[])**

初始化i为多线性表的长度，j为0；

For循环j从0到i-1，比较线性表的名称跟指定名称，如果相等，删除该位置的线性表，也就是从该位置的后一位开始每位都前移一个；

删除完毕，长度减一，返回OK；

如果遍历结束时未找到相同名称的，返回ERROR。

1. **LocateList(LISTS lists,char ListName[])**

初始化i为0；

For循环，i从0到length-1；比较线性表名称与指定名称，如果相等，直接返回位置i+1；

遍历结束如果没有与指定名称相等的线性表，返回ERROR。

1. **UpList(LinkList &L)**

如果L不存在，返回INFEASIBLE;

For循环，p从首元结点开始，到最后一个结点；q从当前的p结点开始，一直到最后一个结点，与p的数据域比较，找到最小值，然后交换最小值的数据域与当前节点的数据域；

遍历结束，返回ERROR。

1. **DownList(LinkList &L)**

如果L不存在，返回INFEASIBLE;

For循环，p从首元结点开始，到最后一个结点；q从当前的p结点开始，一直到最后一个结点，与p的数据域比较，找到最大值，然后交换最小值的数据域与当前节点的数据域；

遍历结束，返回ERROR。

1. **ReverseList(LinkList &L)**

如果L不存在，返回INFEASIBLE;

初始化now记录当前到达的结点的位置，pre记录逆置以后now该指向的结点 ；

While循环，判断条件为now是否为空，即是否遍历完毕；next记录当前结点now的下一个结点，对now的指针域进行更新，指向其前驱；完毕后pre，now均后移继续循环；

循环结束后，将头结点的指针域更新，指向pre；返回OK。

1. **MergeList(LinkList &L,LinkList b)**

如果要归并的两个线性表不存在，返回INFEASIBLE;

初始化a为第一个线性表的首元结点，b为第二个线性表的首元结点；

While循环，判断条件是a，b都不为空；进入循环后，比较a，b的数据域，如果a的数据域更小，就使a以及a的前驱pre后移；如果相等，b后移结点；如果b小a大，就在pre和a之前插入b结点；插入以后，b后移至第二个线性表的下一个结点，pre后移，再次循环；

循环完毕后，如果a为NULL，可能第二个线性表并未插入完毕，此时后面的数据都是较大的，所以直接让最后一个结点的指针域指向剩余结点即可，最后一个结点为pre，剩余结点以b开头；

归并完毕，返回OK。

**2.4系统测试**

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 实验输入 | 预期结果 | 实际结果 |
| 1 | 1 | 线性表创建成功 | $O7X7JHV[8K[XF`X~U6T(D6 |
| 2 | 2 | 线性表销毁成功 | )G}`I1~BXCW@4{)39D)NFLW |
| 3 | 3 | 线性表不存在 | ZB3Z8G55C4FQO3WA5WTQPMT |
| 4 | 1 | 线性表创建成功 | V(4KET9[SQL`AR6{PLP(LUD |
| 5 | 5 | 线性表长度为0 | Y3LQVU@]]}3@)E`QRA9_S`9 |
| 6 | 4 | 线性表为空 | RRKTJS7RU%K[UU]V9(WK%TJ |
| 7 | 6  1 | 输入的1值不合法 | 9G)5GESJ7@M52D(`UO)$%@I |
| 8 | 10  1 2 | 插入成功 | )5~MAMZ(20`_P69{~Q110NK |
| 9 | 10  2 23 | 插入成功 | 5PZ[Z8QYYI(3FJ[Z5]}F{L7 |
| 10 | 10  3 11 | 插入成功 | ZAZYOWJV`LTWX]IQ3O]8~T8 |
| 11 | 10  5 10 | 输入的5值不合法 | VA`BJSI6X6AULPDH_B2]6%0 |
| 12 | 10  4 10 | 插入成功 | XL3PN%5L$RN2K[$2LM5Z[W2 |
| 13 | 10  5 20 | 插入成功 | TI)}BLJYBKB$TVDRTHZ[}X1 |
| 14 | 10  5 20 | 插入成功 | 7}(QS~S11}{R{9IR6QT7@26 |
| 15 | 12 | 2 23 11 10 20 10 | DQ)IHF5H~Z`}965}[]MQE$Y |
| 16 | 11  6 | 删除成功，删除元素为10！ | JS}`PHTAI`ZIQTS$)O2RX0P |
| 17 | 12 | 2 23 11 10 20 | V0G$`OO~Z8]YX%Y7PHI7RTG |
| 18 | 10  6 99 | 插入成功 | VTYSZ37{(ZST_XBBO2EO@FV |
| 19 | 10  7 223 | 插入成功 | N_2VRZ9Q_R5TBH~KVO%GRJS |
| 20 | 12 | 2 23 11 10 20 99 223 | 0D%M{6PBXE%MX4P%A$YCI90 |
| 21 | 10  7 6 | 插入成功 | BXK58)$2]L@$UJE5{{@KCAJ |
| 22 | 12 | 2 23 11 10 99 6 223 | ]IQ1E8BBL9~9FYMWA6A8U2Y |
| 23 | 13  xxx.txt | 写入文件成功 | W~[)[`])(4O(PX{6BGA43E4 |
| 24 | 3 | 线性表已清空 | {IV{8[W)RYKSL_TX}}M)_1R |
| 25 | 14  xxx.txt | 文件数据写入线性表成功 | )%%ENOKZ)}F(86)4IMN%XA2 |
| 26 | 12 | 2 23 11 10 20 99 6 223 | }`LUB35W72E4@_B%S9Z}~YX |
| 27 | 5 | 线性表长度为8 | K%3Z0U}T6ECDRHG42}78JJI |
| 28 | 6  6 | 线性表第6个值为99 | @XYF172P`5_ZT5%@[6HJ$SU |
| 29 | 7  23 | 元素23在线性表中的位置为2 | NLG_H%L2{_RQVFH{BA%05WX |
| 30 | 7  88 | 线性表中不存在该元素 | ~~](RG~}~]67}NAW@9K9Y8S |
| 31 | 8  11 | 线性表中元素11的前驱元素为23 | JRGH%E5EW2C]RT3LSVAFP$9 |
| 32 | 8  89 | 没有前驱元素 | R]@8OVB}EA{`K)WRT0J(@IK |
| 33 | 9  6 | 线性表中元素6的后继元素为223 | @L(MI30NK2KU{~SLQAD[$QK |
| 34 | 9  223 | 没有后继元素 | T5[W)PU)0VD{NFWJW{{9XXS |
| 35 | 18 | 升序后的线性表：2 6 10 11 20 23 99 223 | 7WDN2VJ8EPE9P%826@N@S44 |
| 36 | 19 | 降序后的线性表：223 99 23 20 11 10 6 2 | W%EVAGVTKR()ASSTYUQL(}M |
| 37 | 20 | 逆置后的线性表：  2 6 10 11 20 23 99 223 | D{N8{P3%K)~YR82JY[7RTSE |
| 38 | 15 | 线性表的遍历 | GMWXB]~B_Q8L96((J{B5`ZN |
| 39 | 17  Er | Er线性表的位置为2 | 2)NISRY5[FC3I[J88A_9QVR |
| 40 | 17  San | San线性表的位置为3 | 0XN$4WY)Z2CJ}G1G6LHT3BC |
| 41 | 17  Si | 不存在该线性表 | {$HI54C89ZM1MHLLQ_8GLCV |
| 42 | 16  San | San线性表删除成功 | MN15MPB6UR`%S}KPEMJA8JS |
| 43 | 16  San | 删除失败 | )XISR9UGT}61AFTNQC3W)Z7 |
| 44 | 21 | 归并后的线性表为：1 2 3 4 5 6 7 9 11 | R8G[HONQXBTY2J6I]B7D}J8 |
| 45 | 0 | 欢迎下次使用本系统 | LYO_SC)~0)YEKUSYZVQ8P@K |

**2.5实验小结**

这次的实验和第一次实验难度相当，都不算很难，上次是基于顺序储存结构的线性表系统实现，这次是基于链式结构的线性表系统实现。

与上一个的实验的区别基本在与物理结构不同，上一个实验是对动态数组进行操作，这次是对动态链表进行操作，后者在插入、删除等方面的操作上面更简单，所需要的时间复杂度和空间复杂度都比较小，但是前者在查找指定位置元素以及初始化和销毁方面比较方便，两种结构都有各自的应用场景以及优缺点。

在链式储存中，需要注意的是前驱的记录和后继next的使用；此外还需要注意一些细节，最后循环结束时，尾结点是哪个指针；还需要注意的是NULL的使用，next野指针应该赋空。

而附加功能是在基于之前c语言对链表的学习结合了数据结构做的，这次的排序由于数据域较短，所以交换数据域，在数据域长的时候，交换结点是更方便的，以及归并，两个顺序表的归并还是顺序表，所以，要注意结点后移开始和结束的边界条件，这点容易出问题。

**3 基于二叉链表的二叉树实现**

## 3.1 实验目的

通过实验达到

⑴加深对二叉树的概念、基本运算的理解；

⑵熟练掌握二叉树的逻辑结构与物理结构的关系；

⑶以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。

## 3.2 二叉树基本运算定义

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

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

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

（2）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。

（3）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。

（4）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。

（5）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。

（6）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。

（7）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。

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

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

（9）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。

（10）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。

（11）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。

（12）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。

(13)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。

附加功能：

1. 保存文件：函数名称为SaveBiTree(T,FileName[])；初始条件是二叉树T存在，FileName是写入文件名；操作结果是将二叉树T以带空结点的先序遍历形式写入文件，最后以-1结束。
2. 文件写入二叉树：函数名称为LoadBiTree(T,FileName[])；初始条件是二叉树不存在，FileName是调用的文件的文件名；操作结果是创建一个二叉树。
3. 遍历森林：函数名称是TraveTrees(BTrees trees)；初始条件是森林中有树存在；操作结果是：对森林进行遍历，输出每棵树的先序遍历序列。
4. 插入一棵树：函数名称是InsertTree(BTrees &trees,BNode T,int pos);初始条件是森林未满；操作结果是：在森林中的指定位置pos插入树T；
5. 删除一棵树：函数名称为DeleteTree(BTrees &trees,int pos);初始条件是pos值合法；操作结果是：删除指定位置的树
6. 获取树的位置：函数名称为GetPos(BTrees trees,char s[]);初始条件是输入一个书的名字；操作结果是：如果有指定的树就返回他在森林中的位置，否则返回ERROR。
7. 选择单树操作：函数名称为OptionTree(BiTree &T);初始条件为选择的树的序号合法；操作结果是对单树进行操作。
8. 合并两棵树：函数名称为 merge(BTrees &trees,int pos1,int pos2);初始条件为输入的两棵树的位置合法；操作结果是：将两棵树合并为一棵树并且得到一个新的结点存储两个节点的信息。（哈夫曼树）

**3.3 函数实现**

（1）创建二叉树：函数名称是CreateBiTree(T,definition)；

如果二叉树本身已经存在了，返回INFEASIBLE;

置一个空栈，并对根节点分配空间；开始遍历definition数组，建立二叉树；a表示当前要创造的结点；

For循环直到遍历到-1；判断当前遍历到的definition数组的关键字，0表示该节点为空。若不为0，首先判断该关键字是否跟前面的有重复，如果重复，直接返回ERROR；如果不重复，对a结点赋值并且进栈；判断下一个关键字是否为0，如果不为0，对b分配空间，为a左子树，然后a更新为左孩子结点重新循环。若当前该关键字为0，当前节点为空；如果栈不为空 ，即当前创造的并非最后一个结点 ；出栈，创造上个结点的右孩子；判断下一结点的关键字，如果不为0，右孩子不为空，对b分配空间，然后得到a的右子树，之后a指向右孩子开始创造右子树。

一个循环结束，遍历definition下一个点，继续循环；

创建成功的话，返回OK。

1. 清空二叉树：函数名称是ClearBiTree (T)；

如果T不存在，无法清空，返回INFEASIBLE;

初始化栈；中序遍历销毁结点。

While循环，结束条件为a为空并且栈为空。找到最左的结点，遍历时将经过的结点入栈；找到最左节点后令b记录a的右孩子，调用free函数释放a；然后对该节点的父节点的孩子结点指空；如果b不为空，开始销毁右子树；如果b为空并且栈不为空，出栈给a；如果b和栈都为空，a指向null。

最后将根节点指空。返回OK

1. 判定空二叉树：函数名称是BiTreeEmpty(T)；

如果T不为空，返回TRUE；

反之，返回FASLE。

1. 求二叉树深度：函数名称是BiTreeDepth(T)；

如果树不存在，返回INFEASIBLE;

初始化栈；初始化深度数组为0；初始化max为0，max记录最大深度；初始化根节点的深度为1；

While循环，到达最右边的结点的左孩子，结点遍历完毕；上一个节点的左孩子不存在，出栈一个结点即最近的右孩子给当前节点a；i记录当前结点的关键字；如果当前结点左孩子存在，左孩子的深度为双亲结点的深度值加一；如果当前结点右孩子存在，右孩子的深度为双亲结点的深度值加一；

a指向下一个结点，左孩子，继续循环。

返回max值表示最大深度。

1. 查找结点：函数名称是LocateNode(T,e)；

初始化栈，置栈为空；先序遍历查找结点；

While循环，遍历到最右边的结点；如果一个节点的左孩子为空，出栈一个结点 ；如果关键字吻合，返回该节点；如果右孩子存在，右孩子入栈；继续遍历左子树；

While循环结束，遍历结束没有指定的值，返回NULL。

1. 结点赋值：函数名称是Assign(T,e,value)

如果树不存在，返回INFEASIBLE;

初始化栈，置栈为空；初始化k[i]为0,表示关键字i还未出现；

While循环，遍历到最右结点；如果双亲节点的左孩子为空，出栈给a；关键字记录给i，k[i]=1标记表示该关键字出现过；判断关键字是否为所要的关键字，用b记录该节点；右孩子入栈；a指向左子树，遍历左子树；

如果b不为空，找到了要赋值的结点；如果该关键字与原关键字相同 ，直接赋值，返回OK；如果不同，查找该关键字是否与其他关键字重复，遍历k数组，如果该节点的关键字对应的标记结点的值为1并且关键字相同，则重复了，返回ERROR；For结束，未重复，赋值；返回OK；

如果b为空，返回ERROR。

1. 获得兄弟结点：函数名称是GetSibling(T,e)；

初始化栈，置栈为空；

判断根节点是否为所要查询的结点，如果根节点为所要找的结点，无兄弟结点 ，返回NULL；

While遍历，采用先序遍历；如果双亲节点的左孩子为空，出栈给a；如果a的左孩子存在，判断左孩子的关键字是否符合；如果符合，返回双亲的右孩子；如果a的右孩子存在，判断右孩子的关键字是否符合；如果符合，返回双亲的左孩子；a指向左子树继续循环；

循环结束，未找到指定节点，返回NULL。

1. 插入结点：函数名称是InsertNode(T,e,LR,c)；

对要插入的结点初始化并赋值b，初始化栈，置栈为空；

While遍历，采用先序遍历；如果双亲节点的左孩子为空，出栈给a；判断当前节点的关键字与所要插入的节点关键字是否相同，如果相同，关键字重复，返回ERROR；判断指定的关键字与当前节点的关键字是否相同，如果相同，用p记录插入的结点的双亲结点；右孩子入栈，a指向左孩子，继续循环；

如果LR为-1，插入根节点，即将原树的根节点作为b的右子树，更新根节点，返回OK；

如果p不为空，指定节点存在，如果LR为0，插入在左孩子处，原来的节点的孩子记录在插入结点的右孩子；如果为1，插入在右孩子处，原来的节点的孩子记录在插入结点的右孩子；

如果p为空，未找到指定的结点，返回ERROR。

1. 删除结点：函数名称是DeleteNode(T,e)；

如果树不存在，返回INFEASIBLE；

初始化栈，置栈为空；初始化pa数组，记录双亲结点；

While循环，先序遍历，遍历到最右边的结点；

判断当前节点关键字是否等于指定关键字；如果相等，找到要删除的结点，用b记录该节点；如果左右孩子都存在，将右孩子放置在左孩子的最右结点的右孩子处，则初始化栈为空，a指向左子树找到最右结点，继续循环找左子树的最右结点；如果孩子数小于2，不需要找到左孩子的最右结点，直接break；如果不相等，记录该节点的孩子的双亲结点；右孩子入栈，指向左子树，如果左子树为空，出栈；

循环结束， 最右结点未判断，判断最右结点是否符合，如果符合，将该节点赋给b；

如果b不为空，要删除的结点存在，用i进行关键字记录，通过双亲数组找到要删除的结点的双亲结点pre；如果pre为空，删除的是根结点，如果b的左孩子不存在，根节点直接更新为右孩子；如果左孩子存在但右孩子不存在，根节点直接更新为左孩子；如果左右孩子都存在，将b的右孩子存放在a的右孩子处，根节点更新为左孩子；释放b的空间，返回OK；

如果pre不为空，如果b等于pre的左孩子，要删除为左结点；如果b的左孩子不存在，或右孩子不存在，直接删除b结点；如果左右孩子都存在，将b的右孩子存放在a的右孩子处，再删除b结点；释放b的空间，返回OK；如果b等于pre的右孩子，要删除为左结点；如果b的左孩子不存在，或右孩子不存在，直接删除b结点；如果左右孩子都存在，将b的右孩子存放在a的右孩子处，再删除b结点；释放b的空间，返回OK；

如果b为空，则未找到要删除的指定节点，返回ERROR。

1. 前序遍历：函数名称是PreOrderTraverse(T,Visit())；

如果树不存在 ，返回INFEASIBLE；

初始化栈，置栈为空；初始化a指向根节点；

While循环遍历，结束条件为a为空并且栈为空；每一次循环，如果a为空，出栈一个给a，调用visit函数访问该节点，如果a的右孩子不为空，右孩子入栈；a指向左子树，继续循环。

循环结束，遍历完毕，返回OK。

1. 中序遍历：函数名称是InOrderTraverse(T,Visit))；

如果树不存在 ，返回INFEASIBLE；

初始化栈，置栈为空；初始化a指向根节点；

While循环遍历，结束条件为a为空且栈为空；循环中，while循环找到以a为根节点的树的最左的结点，过程中经过但未访问的结点入栈保存，a一直指向左孩子；出栈一个结点给a，记录当前遍历的子树的最左结点；对a进行访问；判断a的右子树是否存在，如果不存在，该节点的左子树与根节点已遍历，右子树不存在，向上遍历双亲结点，一直出栈给a一个结点并访问，直到a存在右孩子或者栈为空，a指向右子树，继续while循环遍历右子树；

循环结束，遍历完毕，返回OK。

1. 后序遍历：函数名称是PostOrderTraverse(T,Visit))；

如果树不存在 ，返回INFEASIBLE；

初始化栈，置栈为空；初始化a指向根节点；

如果a不为空，对树进行遍历，do-while循环进行遍历；找到以a为根节点的树的最左节点，过程中经过的结点入栈保存；置pre为空，表示上一个访问的结点，置flag标志为1；while循环将栈顶元素置给a，如果a的右孩子不等于pre，该节点的右孩子未被遍历，遍历右子树，a指向右子树，flag置为0，跳出循环；如果a的右孩子等于pre，左右孩子均已被遍历，然后遍历该节点，更新pre为a，继续循环，直到栈为空或者flag标志为0；

直到栈为空，遍历结束；返回OK。

1. 按层遍历：函数名称是LevelOrderTraverse(T,Visit))；

bro数组记录兄弟结点，下一个兄弟结点；对该数组初始化为空；初始化a指向根节点；

While循环直到a为空结束；置k为0表示该层尚未遍历结点；b指向a，while循环遍历这一层；对b进行访问，对b的孩子进行兄弟节点的保存：

如果k为0并且b的左孩子存在，更新k为b的左孩子的关键字，如果b左孩子存在但k不为0，对关键字为k的兄弟结点记录为b的左孩子；如果k为0并且b的右孩子存在，更新k为b的左孩子的关键字，如果b右孩子存在但k不为0，对关键字为k的兄弟结点记录为b的右孩子；

b更新为b的兄虎结点，继续while循环，直到b为空；

b指向该层的第一个结点，开始找下一层的第一个结点保存到a中继续循环；如果b的左孩子存在，b指向b的左孩子并跳出循环；如果b左孩子不存在但右孩子存在，b指向b的右孩子并跳出循环；如果两孩子都不存在，b指向自己的兄弟节点继续循环直到b为空；

a指向b更新为下一层的第一个结点；

While循环结束，遍历完毕，返回OK。

附加功能：

（13）保存文件：函数名称为SaveBiTree(T,FileName[])；

一“w”的方式打开名称为FileName，如果打开失败，返回ERROR；

While循环对树进行先序遍历，将a的数据域写入文件，如果a的右孩子存在，右孩子入栈；如果a的左孩子存在，a指向a的左孩子，如果a的左孩子为空并且栈不为空，将a的空孩子写入，出栈一个结点给a；如果a的左孩子不存在并且栈为空，写到最后一个结点，a指空，将左右空孩子写入；

最后写入-1；写入完毕；

关闭文件；返回OK。

（14）文件写入二叉树：函数名称为LoadBiTree(T,FileName[])；

以“r”的方式打开名称为FileName的文件，如果打开失败，返回ERROR；

读取文件中的第一个关键字存放在k中，如果k等于-1，返回OK；

如果k不等于-1，为根节点分配空间，初始化a指向根节点；

While循环对先序保存的文件进行先序创建二叉树，直到读取到关键字为-1结束；

如果关键字不为0，当前创立的结点不为空结点，对数据域进行赋值，读取下一个结点的值存放在i，如果i大于0，左孩子存在，当前结点入栈，更新要创建的当前结点a指向其左孩子；如果i不大于0，将null读取出来防止出错；k更新为下一关键字i；

如果关键字为0，当前创立的节点为空结点，读取下一结点存放在i，如果i不为0即为创建右孩子 ，右孩子创建完毕后更新当前创建结点；如果i等于0，右孩子也为0，即该节点为叶子结点；采用while循环，将最近的叶子结点都过滤掉，直到i不为0；如果i不等于-1，即要创造一个右孩子结点；更新k为下一个关键字；

创建完毕，关闭文件；返回OK。

1. 遍历森林：函数名称是TraveTrees(BTrees trees)；

初始化i为森林的树的数目，for循环遍历每棵树：

从j=0到j=i-1输出树的名字，并调用PreOrderTraverse输出先序遍历序列和InOrderTraverse输出中序遍历序列；

遍历完毕，返回OK。

1. 插入一棵树：函数名称是InsertTree(BTrees &trees,BNode T,int pos);

初始化length为森林中树的数目；

如果length大于等于10，森林已满，无法插入；

如果pos小于1或者大于length+1，输入的位置不合法，返回ERROR；

之后将pos到最后一颗树后移，在pos位置插入树T；

森林的树的数目加一；

返回OK。

1. 删除一棵树：函数名称为DeleteTree(BTrees &trees,int pos);

初始化length为森门中树的数目；

如果pos小于0或者大于length，输入的位置不合法，返回ERROR；

将第pos个开始，到最后一个，树前移一位，最后释放最后一颗树的空间并使森林中树的数目减一；

返回OK。

1. 获取树的位置：函数名称为GetPos(BTrees trees,char s[]);

初始化length为森林中树的数目；

遍历森林，从i=0到length-1；如果名称相同，返回位置i+1；

遍历完毕，未找到，返回ERROR。

1. 选择单树操作：函数名称为OptionTree(BiTree &T);

输入要选择的树的位置序号，如果序号小于1或者大于森林的树的数目，返回ERROR；如果输入值合法，对该位置的树进行多项操作；

返回OK。

1. 合并两棵树：函数名称为 merge(BTrees &trees,int pos1,int pos2);

初始化length为森林中树的数目；

如果pos1和pos2的值小于1或者大于length或两值相等，输入值不合法，返回ERROR；

创建一个新的结点，左孩子指向pos1的根节点，右孩子指向pos2的根节点，并在数据域中储存两节点的相关信息，可为权重之和；pos1位置的根节点更新为这个新的结点；

返回OK。

**3.4 系统调试**

|  |  |  |  |
| --- | --- | --- | --- |
| 选择操作 | 输入 | 预计输出 | 实际输出 |
| 遍历森林 | 1 | 森林为空 | LJ$RH`2$M)]CNRGGJ10W$ER |
| 删除树 | 3 1 | 输入的值不合法 | Y8UKR`X4D(V[1N)8)2T$}YW |
| 合并两棵树 | 5 2 1 | 输入值不合法 | EF9O@TG$RD97LWEH2%86OPV |
| 插入树 | 2 1 shuyi | 插入成功 | }`5)TRQ7`S]$1V7%CK}60RT |
| 插入树 | 2 2  Shuer  Shusan | 插入成功 | LOZB)~]E0U_]XYKVV$32WK0 |
| 遍历多树 | 1 | 森林中有3棵树，遍历如下：  第1个树的名称为shuyi  第1个二叉树的遍历为：  先序遍历为：此树为空!  中序遍历为: 此树为空！  第2个树的名称为shuer  第2个二叉树的遍历为：  先序遍历为：此树为空!  中序遍历为: 此树为空！  第3个树的名称为shusan  第3个二叉树的遍历为：  先序遍历为：此树为空!  中序遍历为: 此树为空！ | TJM]3(RRW{T4O)(~OX1P$%3 |
| 树名获得位序 | 4 shu | 森林中不存在该名称的二叉树！ | @[MA40RAL71VULIOOAZ7{I0 |
| 树名获得位序 | 6 shuyi | 名称为shuyi的树的位置为1 | RXLOZW$`Y~9}M~K0D1W7DQB |
| 删除树 | 3 2 | 删除成功 | IMG_256 |
| 删除树 | 3 3 | 输入的值不合法 | IMG_256 |
| 进入单树操作 | 6 1 | 对单个树进行操作的菜单 | IMG_256 |
| 创建 | 1  1 a 2 b 0 null 0 null 3 c 4 d 0 null 0 null 5 e 0 null 0 null -1 | 创建完毕 | IMG_256 |
| 求深度 | 4 | 深度为3！ | IMG_256 |
| 查找结点 | 5 1 | 关键字为1的结点的名称为a! | _61PBC[L%@O[S(ZD[$VPUQH |
| 查找结点 | 5 9 | 未二叉树中没有关键字为9的树! | 1(23O[${XC@T]6)7$CGWVAN |
| 结点赋值 | 6 2 1 k | 关键字不合法！赋值失败！ | 75N%I@%SUQ9H]04{K~@Q0(W |
| 结点赋值 | 6 2 9 k | 赋值成功 | GY)~MJ{4_VE4SZNAVL4H5T1 |
| 删除结点 | 7 | 删除失败 | X)T]5}D6CC4N~TF)4@2](OB |
| 判空 | 3 | 树不为空 | IMG_256 |
| 获得兄弟结点 | 7 2 | 该关键字不合法或者其兄弟结点为空！ | IMG_256 |
| 获得兄弟结点 | 7 9 | 该关键字的兄弟结点的数据域为: 3,c | IMG_256 |
| 插入结点 | 7 3 0 8 r | 插入成功 | IMG_256 |
| 删除结点 | 9 | 删除成功 | IMG_256 |
| 先序遍历 | 10 | 对该树的先序遍历为： 1,a 9,k 3,c 8,r 5,e | IMG_256 |
| 删除结点 | 9 7 | 删除失败 | IMG_256 |
| 中序遍历 | 11 | 对该树的中序遍历为9,k 1,a 8,r 3,c 5,e | LN)33@Y`]WU_W5EY)C78{C7 |
| 后序遍历 | 12 | 对该树的后序遍历为 9,k 8,r 5,e 3,c 1,a | D(1)O4%0FD50_H]S]@0Q@{K |
| 按层遍历 | 13 | 对该树的按层遍历为1,a 9,k 3,c 8,r 5,e | SJW)E8XXB2L2WTTW]4{}@I2 |
| 文件写入 | 14 data.txt | 写入文件成功！ | TLUL9FT5Z$W44`@}{X3XW~W |
| 清空树 | 2 | 树已清空 | 9884{H5PIE`VAHBZV@@S)6A |
| 文件读入 | 15 data.txt | 二叉树写入成功！ | ]AI]~77$1CS3UZOU0)5(NHP |
| 先序遍历 | 10 | 对该树的先序遍历为 1,a 9,k 3,c 8,r 5,e | )Q$XY9~Q)O7O(Q_PUD766BA |
| 退出单表 | 0 | 多树操作界面 | XCFE[DEW5K]ZT98%B_4G5KM |
| 退出多表 | 0 | 欢迎下次再使用本系统！ | PV1%KU2FU2V]A52I)TD%(]M |

**3.5 实验小结**

这次的实验是在二叉树的各项功能上实现的系统，整体来说相较于前两个稍微复杂了一些，主要考察遍历等基础功能和多树的一些基本操作；整体来说，由于对树的操作不太熟悉，所以这个实验完成起来相较于前两个耗时更长；

在许多细节方面，二叉树中的细节也增加了不少，例如结点释放的时候的双亲结点的孩子指针应该指空，所以在遍历的时候用孩子结点做判断更简便；在不同的算法中，入栈的结点不相同，需要认真考虑；

在算法上面，用的算法并未使用递归，所以栈是一个很好的方法，通过入栈保存一些结点，达到遍历到底再向上回去的效果；许多算法也有很大的精简空间，后序遍历的算法，自己写了以后总是有一些边界情况不能满足要求，在又一次研究了书上的后序遍历的非递归算法以后，发现do-while循环能更好的考虑到边界情况，所以将while改成do-while更好一些，在算法上有了更进一步的认识；在其他的一些函数中也是如此，例如删除结点时，如何找到子树的最右结点，更新栈是一个很好的选择；

此外，多树操作中，合并树操作为哈夫曼树的灵感，可以使用该算法生成一个哈夫曼树，每次创造的新节点可以保存权重之和；

这次的实验整体上还是提高了难度，让我对算法有了更新的认识，当然还有一些更好复杂度更低更稳定的算法还需要学习。

附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

/\*---------page 10 on textbook ---------\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

typedef struct{ //线性表的集合类型定义

struct { char name[30];

SqList L;

} elem[10];

int length;

}LISTS;

LISTS Lists;

/\*-----page 19 on textbook ---------\*/

status InitList(SqList &L);//初始化一个线性表

status DestroyList(SqList &L);//销毁一个线性表

status ClearList(SqList &L);//清空一个线性表

status ListEmpty(SqList L);//判断一个线性表是否为空

int ListLength(SqList L);//看一个线性表的长度

status GetElem(SqList L,int i,ElemType &e);//取出第i个元素

status LocateElem(SqList L,ElemType e); //定位e元素的位置

status PriorElem(SqList L,ElemType cur,ElemType &pre\_e);//获得e元素的前驱

status NextElem(SqList L,ElemType cur,ElemType &next\_e);//获得e元素的后继

status ListInsert(SqList &L,int i,ElemType e);//插入e元素在第i个元素之前

status ListDelete(SqList &L,int i,ElemType &e);//删除第i个元素

status ListTrabverse(SqList L); //遍历线性表并输出

status SaveList(SqList L,char FileName[]);//将线性表的元素写入文件中

status LoadList(SqList &L,char FileName[]);//将文件中的数据元素写入线性表

status AddList(LISTS &Lists,char ListName[]);//添加一个线性表表

status RemoveList(LISTS &Lists,char ListName[]);//删除一个指定名称的线性表

int LocateList(LISTS Lists,char ListName[]);//定位一个指定名称的线性表

status UpList(SqList &L);//升序排列

status DownList(SqList &L);//降序排列

status SubList(SqList L,SqList s);//判断s是不是L的子集

status InterList(SqList L,SqList S,SqList &p);//求两线性表的交集

status SumList(SqList L,SqList S,SqList &p);//就两线性表的并集

status DifferList(SqList L,SqList S,SqList &p);//求两线性表的差集

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

int main(){

SqList L; int op=1;

L.elem=NULL;

int i,flag;

ElemType y,e,pre,next;

char name[30];

char filename[100],listname[100];

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

printf(" 13.SaveList 14.LoadList\n");

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

printf(" 17.LocateList 18.UpList\n");

printf(" 19.DownList 20.SubList\n");

printf(" 21.InterList 22.SumList\n");

printf(" 23.DifferList 0.exit\n");

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

while(op){

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

printf("请选择你的操作[0~23]:");

scanf("%d",&op);

switch(op){

case 1:

//printf("\n----IntiList功能待实现！\n");

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

else printf("线性表已存在，创建失败！\n");

getchar();getchar();

break;

case 2:

//printf("\n----DestroyList功能待实现！\n");

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

else printf("线性表本身不存在，销毁失败！\n");

getchar();getchar();

break;

case 3:

//printf("\n----ClearList功能待实现！\n");

if(ClearList(L)==OK) printf("线性表已清空!\n");

else printf("线性表不存在!\n");

getchar();getchar();

break;

case 4:

//printf("\n----ListEmpty功能待实现！\n");

if(ListEmpty(L)==TRUE) printf("线性表为空！\n");

else if(ListEmpty(L)==FALSE) printf("线性表非空！\n");

else printf("线性表不存在！\n");

getchar();getchar();

break;

case 5:

//printf("\n----ListLength功能待实现！\n");

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

else printf("线性表的长度为%d\n",ListLength(L));

getchar();getchar();

break;

case 6:

//printf("\n----GetElem功能待实现！\n");

if(!L.elem)

{

printf("线性表不存在！\n");

break;

}

printf("请输入获取元素的位序：");

scanf("%d",&i);

if(GetElem(L,i,e)==ERROR) printf("输入的%d值不合法！\n",i);

else printf("线性表第%d个值是%d!\n",i,e);

getchar();getchar();

break;

case 7:

//printf("\n----LocateElem功能待实现！\n");

if(!L.elem){

printf("不存在线性表！\n");

break;

}

if(LocateElem(L,e)==ERROR)

{

printf("线性表中不存在该元素!\n");

break;

}

printf("请输入要定位的元素：");

scanf("%d",&e);

printf("\n");

if(LocateElem(L,e)>0) printf("元素%d在线性表中的位置为%d!\n",e,LocateElem(L,e));

getchar();getchar();

break;

case 8:

//printf("\n----PriorElem功能待实现！\n");

if(!L.elem){

printf("线性表不存在！\n");

break;

}

printf("请输入元素以获得前驱：");

scanf("%d",&e);

printf("\n");

if(PriorElem(L,e,pre)==OK) printf("线性表中元素%d的前驱元素为%d\n",e,pre);

else if(PriorElem(L,e,pre)==ERROR) printf("没有前驱元素！\n");

getchar();getchar();

break;

case 9:

//printf("\n----NextElem功能待实现！\n");

if(!L.elem){

printf("线性表不存在！\n");

break;

}

printf("请输入元素以获得后继：");

scanf("%d",&e);

printf("\n");

if(NextElem(L,e,next)==OK) printf("线性表中元素%d的后继元素为%d\n",e,next);

else if(NextElem(L,e,next)==ERROR) printf("没有后继元素！\n");

getchar();getchar();

break;

case 10:

//printf("\n----ListInsert功能待实现！\n");

if(!L.elem){

printf("线性表不存在！\n");

break;

}

printf("请输入要插入的位置及元素：");

scanf("%d",&i);scanf("%d",&e);

printf("\n");

if(ListInsert(L,i,e)==OK) printf("插入成功！\n");

else if(ListInsert(L,i,e)==ERROR) printf("输入的值%d不合法！\n",i);

getchar();getchar();

break;

case 11:

//printf("\n----ListDelete功能待实现！\n");

if(!L.elem){

printf("线性表不存在！\n");

break;

}

printf("请输入要删除元素的位序：");

scanf("%d",&i);

printf("\n");

if(ListDelete(L,i,e)==OK) printf("删除成功，删除元素为%d!\n",e);

else if(ListDelete(L,i,e)==ERROR) printf("删除失败！输入值不合法!\n");

getchar();getchar();

break;

case 12:

//printf("\n----ListTrabverse功能待实现！\n");

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

else if(L.length==0) printf("线性表是空表！\n");

getchar();getchar();

break;

case 13:

//SaveList函数

if(!L.elem){

printf("线性表不存在！\n");

break;

}

printf("请输入文件名：");

scanf("%s",filename);

if(SaveList(L,filename)==OK) printf("\n写入文件成功！\n");

else if(SaveList(L,filename)==ERROR) printf("写入文件失败！\n");

getchar();getchar();

break;

case 14:

//LoadList函数

if(!L.elem){

printf("线性表不存在！\n");

break;

}

printf("请输入文件名：");

scanf("%s",filename);

if(LoadList(L,filename)==OK) printf("文件数据输入线性表成功！\n");

else if(LoadList(L,filename==ERROR)) printf("文件数据输入线性表失败！\n");

//else printf("线性表不存在！\n");

getchar();getchar();

break;

case 15:

//AddList函数

printf("请输入要添加线性表的个数：\n");

int n;

scanf("%d",&n);

while(n--){

printf("请输入要增加的线性表名称：\n");

scanf("%s",name);

AddList(Lists,name);

}

printf("线性表个数为%d\n",Lists.length);

printf("下面给出多个线性表的遍历：\n");

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

{

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

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

putchar('\n');

}

getchar();getchar();

break;

case 16:

//RemoveList函数

printf("请输入删除线性表的名称") ;

scanf("%s",listname);

if(RemoveList(Lists,listname)) printf("%s线性表删除成功!\n",listname);

else printf("删除失败！\n");

getchar();getchar();

break;

case 17:

//LocateList函数

printf("请输入所需查找的线性表名称");

scanf("%s",listname);

if(LocateList(Lists,listname)) printf("%s线性表的位置为%d!\n",listname,LocateList(Lists,listname));

else printf("不存在该线性表！\n");

break;

case 18:

if(UpList(L)==INFEASIBLE){

printf("线性表不存在！\n");

break;

}

printf("升序后的线性表为：\n");

ListTrabverse(L);

printf("\n");

break;

case 19:

if(DownList(L)==INFEASIBLE){

printf("线性表不存在!\n");

break;

}

printf("降序后的线性表为：\n");

ListTrabverse(L);

break;

case 20:

flag=SubList(Lists.elem[0].L,Lists.elem[1].L);

if(flag==INFEASIBLE){

printf("多线性表中不存在两个线性表!\n");

break;

}

if(flag==OK){

printf("%s是%s的一个子集！\n",Lists.elem[1].name,Lists.elem[0].name);

}

else{

flag=SubList(Lists.elem[1].L,Lists.elem[0].L);

if(flag==OK){

printf("%s是%s的一个子集！\n",Lists.elem[0].name,Lists.elem[1].name);

}

else{

printf("二者不存在子集关系！\n");

}

}

break;

case 21:

if(InterList(Lists.elem[0].L,Lists.elem[1].L,Lists.elem[2].L)==INFEASIBLE) printf("线性表不存在\n");

if(Lists.elem[2].L.length==0) printf("交集为空！");

else {

printf("两线性表的交集是：");

ListTrabverse(Lists.elem[2].L);

}

printf("\n");

break;

case 22:

if(SumList(Lists.elem[0].L,Lists.elem[1].L,Lists.elem[2].L)==INFEASIBLE) printf("线性表不存在\n");

if(Lists.elem[2].L.length==0) printf("并集为空！");

else {

printf("两线性表的并集是：");

ListTrabverse(Lists.elem[2].L);

}

printf("\n");

break;

case 23:

if(DifferList(Lists.elem[0].L,Lists.elem[1].L,Lists.elem[2].L)==INFEASIBLE) printf("线性表不存在\n");

if(Lists.elem[2].L.length==0) printf("差集为空！");

else {

printf("两线性表的差集是：");

ListTrabverse(Lists.elem[2].L);

}

printf("\n");

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

return 0;

}//end of main()

/\*--------page 23 on textbook --------------------\*/

status InitList(SqList &L)

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

{

if(L.elem) return INFEASIBLE;//存在线性表，如果再构造线性表，会覆盖掉原先的元素，所以返回INFEASIBLE

L.elem=(ElemType\*)malloc(sizeof(ElemType)\*LIST\_INIT\_SIZE); //给线性表分配空间

if(!L.elem) return INFEASIBLE;//如果分配失败，返回INFEASIBLE

L.length=0;//构造空的线性表，表长为0

L.listsize=LIST\_INIT\_SIZE;//设定最大长度

return OK;

}

status DestroyList(SqList& L)

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

{

if(!L.elem) return INFEASIBLE;//线性表不存在，返回INFEASIBLE

L.length=0;//让线性表为空

free(L.elem);//释放数据元素的空间

L.elem=NULL;//L.elem指针仍旧存在，成为了野指针，不符合释放空间的要求，释放后使其指向空指针

return OK;

}

status ClearList(SqList& L)

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

{

if(!L.elem) return INFEASIBLE;//线性表不存在，返回INFEASIBLE

L.length=0;//使线性表为空来删除所有元素

return OK;

}

status ListEmpty(SqList L)

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

{

if(!L.elem) return INFEASIBLE;//线性表不存在，返回INFEASIBLE

if(L.length==0) return TRUE;//线性表为空，返回1

else return FALSE;//线性表不为空，返回0

}

status ListLength(SqList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

if(!L.elem) return INFEASIBLE;//线性表不存在，返回INFEASIBLE

return L.length;//线性表存在，返回其长度值

}

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

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

{

if(!L.elem) return INFEASIBLE;//线性表不存在，返回INFEASIBLE

if(i>L.length||i<=0) return ERROR;//i的值不合法，元素的位置无负数并且有最大值

e=L.elem[i-1];//i值合法时，返回第i个元素

return OK;

}

status LocateElem(SqList L,ElemType e)

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

{

if(!L.elem) return INFEASIBLE;//线性表不存在，返回INFEASIBLE

int i;//用来表示元素在线性表中的逻辑位置

for(i=0;i<L.length;i++)//查找线性表中是否有元素与e相等

if(e==L.elem[i]) return i+1;//若有元素与e相等，直接返回逻辑位置的值

return ERROR;//没有与e相等的值，返回0

}

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

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

{

if(!L.elem) return INFEASIBLE;//如果线性表不存在，返回INFEASIBLE

if(e==L.elem[0]) return ERROR;//线性表中的第一个元素与e相等，但第一个元素没有直接前驱，返回ERROR

int i;//用来记录线性表元素的逻辑位置

for(i=1;i<L.length;i++)//查找有前驱的元素中是否有与e值相等的元素

if(e==L.elem[i]){//发现与e相等的元素

pre=L.elem[i-1];//记录前驱元素

return OK;

}

return ERROR;//未找到与e相等的元素

}

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

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

{

if(!L.elem) return INFEASIBLE;//如果线性表不存在，返回INFEASIBLE

int i;//i用来标记元素的逻辑位置

for(i=0;i<L.length-1;i++)//在有后驱的元素中查找是否有与e相等的元素 最后一个元素无后驱

if(e==L.elem[i]){//找到与e相等的元素

next=L.elem[i+1];//记录后驱元素

return OK;

}

return ERROR;//未找到有后驱且与e相等的元素

}

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

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

{

if(i<=0||i>L.length+1) return ERROR;//i的值不合法，无法插入

if(!L.elem) return INFEASIBLE;//线性表不存在，return INFEASIBLE

if(L.length>=L.listsize){//线性表已满

L.elem=(ElemType\*)realloc(L.elem,(L.listsize+LISTINCREMENT)\*(sizeof(ElemType)));//对线性表重新分配空间，增加线性表的空间

if(!L.elem) return OVERFLOW;//重新分配空间失败

L.listsize+=LISTINCREMENT;//线性表的最大表长增加

}

int j;

for(j=L.length;j>=i;j--)//从i个元素到最后一个元素都后移一位

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

L.elem[i-1]=e;//插入e到i前面，即变成第i位

L.length++;//线性表的表长增加

return OK;

}

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

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

{

if(!L.elem) return INFEASIBLE;//线性表不存在，返回INFEASIBLE

if(i>L.length||i<=0) return ERROR;//i值不合法，线性表不存在第i个元素

e=L.elem[i-1];//记录第i个元素

int j;

for(j=i-1;j<L.length-1;j++)//从第i+1个元素 到最后一个元素，前移一位

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

L.length--;//线性表的表长减一

return OK;

}

status ListTrabverse(SqList L)

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

{

if(!L.elem) return INFEASIBLE;//线性表不存在，返回INFEASIBLE

int i;

if(L.length>0){//线性表非空

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

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

printf("%d",L.elem[i]);//逐个输出元素

}

return OK;

}

status SaveList(SqList L,char FileName[])

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

{

if(!L.elem) return INFEASIBLE;//如果不存在线性表，返回INFEASIBLE

FILE \*file=fopen(FileName,"w");//以“写”的方式打开文件

if(!file) return ERROR;//打开文件失败

int i;

for(i=0;i<L.length;i++){//向文件中写入元素

fprintf(file,"%d ",L.elem[i]);//数字元素之间一定要有空格，不然从文件中读取的时候只能读出来一个数

}

fclose(file);//关闭文件

return OK;

}

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

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

{

if(!L.elem) return INFEASIBLE;

FILE \*file=fopen(FileName,"r");//以读的方式打开文件

if(!file) return ERROR;

ElemType e;//将读出的数据放入e所指的地方

while(fscanf(file,"%d",&e)!=EOF){//读取数据到文件末尾

if(L.length>=L.listsize){//如果文件的数据多于开辟的空间

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

if(!L.elem) return OVERFLOW;

L.listsize+=LISTINCREMENT;

}//重新分配空间

L.elem[L.length]=e;//向线性表写入数据

L.length++;//表长增加

}

fclose(file);//关闭文件

return OK;

}

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

// 只需要在Lists中增加一个名称为ListName的线性表

{

int i=Lists.length,j=0;//用i记录Lists中的数组长度，简便

if(i>=10) return ERROR;

while((\*(Lists.elem[i].name+j)=ListName[j])!='\0') j++;//将名称复制到Lists的数组中总后一个元素的后面

InitList(Lists.elem[i].L);//对增加的线性表初始化

Lists.length++;//线性表组的长度增加

printf("\n是否需要插入元素，插入输入1，不插入输入0：");

int y,e;

scanf("%d",&y);

if(y){

printf("\n请输入要插入的数据元素，以0结束：");

scanf("%d",&e);

while (e)//输入元素

{

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

scanf("%d",&e);

}

}

return OK;

}

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

// Lists中删除一个名称为ListName的线性表

{

int i,j;

char \*l;//用来指向线性表的名称，简便

for(i=0;i<Lists.length;i++)//遍历查看是否有名称为ListName的

{

l=Lists.elem[i].name;

j=0;

while(\*(l+j)==ListName[j]&&\*(l+j)&&ListName[j]) j++;//检测是否名称一样

if(!(\*(l+j))&&!ListName[j]){//找到名称为ListName的线性表

for(j=i;j<Lists.length-1;j++)//删除该线性表

Lists.elem[j]=Lists.elem[j+1];//该线性表后面的线性表都前移一位

Lists.length--;//线性表组的数量减一

Lists.elem[j].L.elem=NULL;//对数组的最后一个元素进行释放

return OK;

}

}

return ERROR;//未找到名称为ListName的线性表

}

int LocateList(LISTS Lists,char ListName[])

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

{

int i,j;

char \*l;

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

l=Lists.elem[i].name;//记录线性表的名称，更简便

j=0;

while(\*(l+j)==ListName[j]&&\*(l+j)&&ListName[j]) j++;//找与目标线性表名称相同的线性表

if(!\*(l+j)&&!ListName[j]){//找到指定名称的线性表

return i+1;//返回所找线性表的逻辑位置

}

}

return 0;//未找到指定线性表

}

//升序

status UpList(SqList &L)

{

if(!L.elem) return INFEASIBLE;

int i,j;

ElemType t;

for(i=0;i<L.length;i++)//排序

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

if(L.elem[i]>L.elem[j])

{

t=L.elem[i];

L.elem[i]=L.elem[j];

L.elem[j]=t;

}

return OK;

}

//降序

status DownList(SqList &L)

{

if(!L.elem) return INFEASIBLE;

int i,j;

ElemType t;

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

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

if(L.elem[i]<L.elem[j])

{

t=L.elem[i];

L.elem[i]=L.elem[j];

L.elem[j]=t;

}

return OK;

}

//判断子表,判断s是否为L 的子集

status SubList(SqList L,SqList s)

{

if(!s.elem||!L.elem) return INFEASIBLE;

int i,flag,j;

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

{

flag=0;//先假设该元素在另一个线性表中不存在

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

if(L.elem[j]==s.elem[i]){//判断子集

flag=1; //该元素存在于另一个线性表中

break;

}

if(!flag) return ERROR;//有元素不相等，则不是子集关系

}

return OK;

}

//求俩顺序表的交,L,S的交集保存在p中

int InterList(SqList L,SqList S,SqList &p)//求交集

{

if(!L.elem||!S.elem||!p.elem) return INFEASIBLE;

int i=0,j=0,k=0;

p.length=0;//清空储存表

while(i<L.length&&j<S.length){

while(L.elem[i]<S.elem[j]) i++;//过滤不相等的元素

while(L.elem[i]>S.elem[j]) j++;//过滤不相等的元素

if(i<L.length&&j<S.length&&L.elem[i]==S.elem[j]){

ListInsert(p,k+1,L.elem[i]);//相等的插入

k++;i++;j++;//记得此时需要改变i，j，不然会死循环

}

}

if(!k) return ERROR;//无交集

return OK;

}

//求两顺序线性表的并

status SumList(SqList L,SqList S,SqList &p)

{

if(!L.elem||!S.elem||!p.elem) return INFEASIBLE;

int i=0,j=0;

p.length=0;

while(i<L.length&&j<S.length)

{

while(i<L.length&&j<S.length&&L.elem[i]<S.elem[j]){

ListInsert(p,p.length+1,L.elem[i]);

i++;

}

while(j<S.length&&i<L.length&&L.elem[i]>S.elem[j])

{

ListInsert(p,p.length+1,S.elem[j]);

j++;

}

while(j<S.length&&i<L.length&&L.elem[i]==S.elem[j])

{

ListInsert(p,p.length+1,S.elem[j]);

j++;i++;

}

}

if(i<L.length)

while(i<L.length)

{

ListInsert(p,p.length+1,L.elem[i]);

i++;

}

if(j<S.length)

while(j<S.length){

ListInsert(p,p.length+1,S.elem[j]);

j++;

}

return OK;

}

status DifferList(SqList L,SqList S,SqList &p)

//求两线性表的差集

{

if(!L.elem||!S.elem||!p.elem) return INFEASIBLE;

int i=0,j=0;

p.length=0;

while(i<L.length&&j<S.length)

{

while(i<L.length&&j<S.length&&L.elem[i]<S.elem[j]){

ListInsert(p,p.length+1,L.elem[i]);

i++;

}

while(j<S.length&&i<L.length&&L.elem[i]>S.elem[j])

{

ListInsert(p,p.length+1,S.elem[j]);

j++;

}

while(j<S.length&&i<L.length&&L.elem[i]==S.elem[j])

{

j++;i++;

}

}

if(i<L.length)

while(i<L.length)

{

ListInsert(p,p.length+1,L.elem[i]);

i++;

}

if(j<S.length)

while(j<S.length){

ListInsert(p,p.length+1,S.elem[j]);

j++;

}

return OK;

}

**附录B 基于链式存储结构线性表实现的源程序**

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef int ElemType;

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

ElemType data;

struct LNode \*next;

}LNode,\*LinkList;

typedef struct{ //线性表的集合类型定义

struct { char name[30];

LinkList L;

} elem[10];

int length;

}LISTS;

status InitList(LinkList &L);//初始化一个线性表

status DestroyList(LinkList &L);//销毁一个线性表

status ClearList(LinkList &L);//清空一个线性表

status ListEmpty(LinkList L);//对一个线性表判空

int ListLength(LinkList L);//求线性表的表长

status GetElem(LinkList L,int i,ElemType &e);//获取线性表的指定位置的元素

status LocateElem(LinkList L,ElemType e);//对指定元素在线性表中进行定位

status PriorElem(LinkList L,ElemType e,ElemType &pre);//获取指定元素的前驱

status NextElem(LinkList L,ElemType e,ElemType &next);//获取指定元素的后继

status ListInsert(LinkList &L,int i,ElemType e);//在指定位置插入一个元素

status ListDelete(LinkList &L,int i,ElemType &e);//删除指定位置的元素

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

status SaveList(LinkList L,char FileName[]);//将线性表数据写入文件

status LoadList(LinkList &L,char FileName[]);//将文件中的数据写入线性表

status AddList(LISTS &lists,char ListName[]);//向多线性表中插入线性表

status RemoveList(LISTS &lists,char ListName[]);//删除指定名称的线性表

int LocateList(LISTS lists,char ListName[]);//定位线性表的位置

status UpList(LinkList &L);//升序排列线性表

status DownList(LinkList &L);//降序排列线性表

status ReverseList(LinkList &L);//逆置线性表

status MergeList(LinkList &L,LinkList b);//合并两个顺序线性表，顺序为升序

LinkList L=NULL;

ElemType e,pre,next;

int i;

char filename[1000],listname[30];

LISTS lists;

int main()

{

int flag,op=1;

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

printf(" 13.SaveList 14.LoadList\n");

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

printf(" 17.LocateList 18.UpList\n");

printf(" 19.DownList 20.ReverseList\n");

printf(" 21.MergeList 0.exit\n");

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

while(op){

printf("请选择你的操作[0~21]:");

scanf("%d",&op);

switch(op){

case 1:

flag=InitList(L);

if(flag==OK) printf("线性表创建成功！\n");

else printf("线性表已存在，创建失败！\n");

getchar();getchar();

break;

case 2:

flag=DestroyList(L);

if(flag==OK) printf("线性表销毁成功！\n");

else printf("线性表本身不存在，销毁失败！\n");

getchar();getchar();

break;

case 3:

flag=ClearList(L);

if(flag==OK) printf("线性表已清空!\n");

else printf("线性表不存在!\n");

getchar();getchar();

break;

case 4:

flag=ListEmpty(L);

if(flag==TRUE) printf("线性表为空！\n");

else if(flag==FALSE) printf("线性表非空！\n");

else printf("线性表不存在！\n");

getchar();getchar();

break;

case 5:

flag=ListLength(L);

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

else printf("线性表的长度为%d\n",flag);

getchar();getchar();

break;

case 6:

if(!L)

{

printf("线性表不存在！\n");

break;

}

printf("请输入获取元素的位序：");

scanf("%d",&i);

flag=GetElem(L,i,e);

if(flag==ERROR) printf("输入的%d值不合法！\n",i);

else printf("线性表第%d个值是%d!\n",i,e);

getchar();getchar();

break;

case 7:

if(!L){

printf("不存在线性表！\n");

break;

}

printf("请输入要定位的元素：");

scanf("%d",&e);

flag=LocateElem(L,e);

printf("\n");

if(flag==ERROR) printf("线性表中不存在该元素!\n");

else printf("元素%d在线性表中的位置为%d!\n",e,flag);

getchar();getchar();

break;

case 8:

if(!L){

printf("线性表不存在！\n");

break;

}

printf("请输入元素以获得前驱：");

scanf("%d",&e);

printf("\n");

flag=PriorElem(L,e,pre);

if(flag==OK) printf("线性表中元素%d的前驱元素为%d\n",e,pre);

else if(flag==ERROR) printf("没有前驱元素！\n");

getchar();getchar();

break;

case 9:

if(!L){

printf("线性表不存在！\n");

break;

}

printf("请输入元素以获得后继：");

scanf("%d",&e);

printf("\n");

flag=NextElem(L,e,next);

if(flag==OK) printf("线性表中元素%d的后继元素为%d\n",e,next);

else if(flag==ERROR) printf("没有后继元素！\n");

getchar();getchar();

break;

case 10:

if(!L){

printf("线性表不存在！\n");

break;

}

printf("请输入要插入的位置及元素：");

scanf("%d",&i);scanf("%d",&e);

printf("\n");

flag=ListInsert(L,i,e);

if(flag==OK) printf("插入成功！\n");

else if(flag==ERROR) printf("输入的值%d不合法！\n",i);

getchar();getchar();

break;

case 11:

if(!L){

printf("线性表不存在！\n");

break;

}

printf("请输入要删除元素的位序：");

scanf("%d",&i);

printf("\n");

flag=ListDelete(L,i,e);

if(flag==OK) printf("删除成功，删除元素为%d!\n",e);

else if(flag==ERROR) printf("删除失败！输入值不合法!\n");

getchar();getchar();

break;

case 12:

flag=ListTraverse(L);

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

else if(L->next==NULL) printf("线性表是空表！\n");

getchar();getchar();

break;

case 13:

if(!L){

printf("线性表不存在！\n");

break;

}

printf("请输入文件名：");

scanf("%s",filename);

flag=SaveList(L,filename);

if(flag==OK) printf("\n写入文件成功！\n");

else if(flag==ERROR) printf("写入文件失败！\n");

getchar();getchar();

break;

case 14:

printf("请输入文件名：");

scanf("%s",filename);

flag=LoadList(L,filename);

if(flag==OK) printf("文件数据输入线性表成功！\n");

else if(flag==ERROR) printf("文件数据输入线性表失败！\n");

getchar();getchar();

break;

case 15:

//AddList函数

printf("请输入要添加线性表的个数：\n");

int n;

scanf("%d",&n);

while(n--){

printf("请输入要增加的线性表名称：\n");

scanf("%s",listname);

flag=AddList(lists,listname);

if(flag==OK) printf("线性表插入成功!\n");

else printf("线性表插入失败!\n");

}

printf("线性表个数为%d\n",lists.length);

printf("下面给出多个线性表的遍历：\n");

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

{

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

flag=ListTraverse(lists.elem[n].L);

if(lists.elem[n].L->next==NULL) printf("该线性表为空表!");

putchar('\n');

}

getchar();getchar();

break;

case 16:

//RemoveList函数

printf("请输入删除线性表的名称：") ;

scanf("%s",listname);

if(RemoveList(lists,listname)) printf("%s线性表删除成功!\n",listname);

else printf("删除失败！\n");

getchar();getchar();

break;

case 17:

//LocateList函数

printf("请输入所需查找的线性表名称:");

scanf("%s",listname);

flag=LocateList(lists,listname);

if(flag!=ERROR) printf("%s线性表的位置为%d!\n",listname,flag);

else printf("不存在该线性表！\n");

break;

case 18:

flag=UpList(L);

if(flag==INFEASIBLE){

printf("线性表不存在！\n");

break;

}

printf("升序后的线性表为：\n");

ListTraverse(L);

printf("\n");

break;

case 19:

flag=DownList(L);

if(flag==INFEASIBLE){

printf("线性表不存在!\n");

break;

}

printf("降序后的线性表为：\n");

ListTraverse(L);

printf("\n");

break;

case 20:

flag=ReverseList(L);

if(flag==INFEASIBLE){

printf("线性表不存在!\n");

break;

}

printf("逆置后的线性表为：");

ListTraverse(L);

printf("\n");

break;

case 21:

printf("请确保多线性表中有两个线性表再进行操作!\n");

flag=MergeList(lists.elem[0].L,lists.elem[1].L);

if(flag==INFEASIBLE){

printf("多线性表中不存在两个线性表!\n");

break;

}

UpList(lists.elem[0].L);

UpList(lists.elem[1].L);

printf("归并后的线性表为：");

ListTraverse(lists.elem[0].L);

printf("\n");

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}

status InitList(LinkList &L)

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

{

if(L) return INFEASIBLE;//如果线性表已存在，无法再初始化

L=(LinkList)malloc(sizeof(LNode));//分配头节点空间

L->next=NULL;//指针赋空

return OK;

}

status DestroyList(LinkList &L)

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

{

if(!L) return INFEASIBLE;

LinkList a,b;

a=L,b=L->next;//a为当前要释放空间的结点，b为后继结点

while(a){//a到达末尾时完毕

free(a);//释放当前空间

a=b;//对下一结点进行操作

if(a)//a不为空时才有后继结点

b=a->next;

}

L=NULL;

return OK;

}

status ClearList(LinkList &L)

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

{

if(!L) return INFEASIBLE;

LinkList a,b;

a=L->next;//从首元结点开始删除，头结点不删除

if(!a) return OK;//链表本身为空，直接返回

b=a->next;//b记录后继节点

while(a){//a不为空时才删除

free(a);

a=b;//对下一个结点操作

if(a)//a不为空才有后继结点

b=a->next;

}

L->next=NULL;//对头结点的next赋空

return OK;

}

status ListEmpty(LinkList L)

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

{

if(!L) return INFEASIBLE;

if(L->next) return FALSE;//首元结点为空，链表为空

else return TRUE;

}

int ListLength(LinkList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

if(!L) return INFEASIBLE;

LinkList a;

a=L->next;//a为当前结点

int len=0;//开始还没计数时赋为0

while(a){//a不为空时才记录当前结点为一个元素

len++;//当前结点有元素，长度加一

a=a->next;//判断下一结点

}

return len;

}

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

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

{

if(!L) return INFEASIBLE;

if(i<=0) return ERROR;

LinkList a;

int len=0;//当前所在位置的元素位置

a=L->next;

while(a){//a不为空时才计数

len++;//计数当前元素的位置

if(len==i){//找到要求的位置

e=a->data;

return OK;

}

a=a->next;

}

return ERROR;

}

status LocateElem(LinkList L,ElemType e)

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

{

if(!L) return INFEASIBLE;

int i=0;

LinkList a=L->next;

while(a){

i++;//记录当前的元素的位置序号

if(a->data==e) return i;//找到元素返回位置

a=a->next;

}

return ERROR;

}

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

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

{

if(!L) return INFEASIBLE;

if(!L->next) return ERROR;

LinkList a,b;

a=L->next;//a为首元结点

if(!a->next) return ERROR;//首元结点无前驱，若首元结点之后为空，就没有含有前驱的结点

b=a->next;//第一个有前驱的结点

while(b){

if(b->data==e){//找到指定元素

pre=a->data;//获取前驱元素

return OK;

}

a=b;b=a->next;//当前结点与前驱结点均后移

}

return ERROR;

}

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

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

{

if(!L) return INFEASIBLE;

if(!L->next) return ERROR;//线性表表为空

LinkList a=L->next;//第一个具有后继的结点

while(a->next){//最后一个具有后继的结点为后继不为空的结点

if(a->data==e){

next=a->next->data;//记录后继元素

return OK;

}

a=a->next;

}

return ERROR;

}

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

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

{

if(!L) return INFEASIBLE;

if(i<=0) return ERROR;//i值不合法

LinkList a=L;

int len=0;//当前结点的位置序号

while(a){//插入到第i个元素之前也就是插入到第i-1个元素之后，一直遍历到最后一个元素，最多插入到最后一个元素之后

if(len==i-1){//找到第i-1个元素，在该元素之后插入元素

LinkList b;//为插入的结点分配空间和位置

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

b->data=e;//插入元素赋值

b->next=a->next;

a->next=b;//插入b在a之后

return OK;

}

a=a->next;len++;

}

return ERROR;//遍历完毕，i值不合法

}

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

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

{

if(!L) return INFEASIBLE;

if(i<=0) return ERROR;//i值不合法

LinkList a=L,b;

int len=0;//当前到达的结点的位置序号

while(a){//删除第i个元素，也就是让第i-1个元素的next记录第i+1个元素

if(len==i-1){//到达第i-1个元素的位置

b=a->next;//记录要删除的结点，第i个元素

if(!b) return ERROR;//要删除的结点为空，i值不合法

e=b->data;//记录要删除的元素

a->next=b->next;//删除结点

free(b);//释放该节点的空间

b=NULL;//野指针赋空

return OK;

}

len++;a=a->next;

}

return ERROR;

}

status ListTraverse(LinkList L)

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

{

if(!L) return INFEASIBLE;

LinkList a=L->next;

while(a){

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

a=a->next;

if(a) printf(" ");//最后一个元素后面不输出空格

}

return OK;

}

status SaveList(LinkList L,char FileName[])

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

{

if(!L) return INFEASIBLE;

LinkList a=L->next;

FILE \*file=fopen(FileName,"w");//以“写”的方式打开文件

if(!file) return ERROR;//打开文件失败

while(a){

fprintf(file,"%d ",a->data);

a=a->next;

}

fclose(file);//关闭文件!!!

return OK;

}

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

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

{

if(L) ClearList(L);

else InitList(L);

LinkList a=L,b=NULL;

FILE \*file=fopen(FileName,"r");//以读的方式打开文件

if(!file) return ERROR;

ElemType e;//将读出的数据放入e所指的地方

while(fscanf(file,"%d",&e)!=EOF){//读取数据到文件末尾

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

b->next=NULL;

b->data=e;

a->next=b;a=b;b=NULL;//在线性表的末尾插入结点，a表示最后一个元素的结点

}

fclose(file);

return OK;

}

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

//向多线性表中添加一个指定名称的线性表

{

int i=lists.length,j=1,flag;

if(i>=10) return ERROR;//超过数组边界

strcpy(lists.elem[i].name,ListName);//名字赋值

InitList(lists.elem[i].L);//初始化线性表

printf("是否要向线性表中插入元素，若要插入元素输入1，否则输入0：");

scanf("%d",&flag);

if(flag){

printf("\n请输入要插入的元素并且以0结尾\n");

scanf("%d",&flag);

while(flag){

ListInsert(lists.elem[i].L,j,flag);//插入元素

j++;

scanf("%d",&flag);

}

}

lists.length++;//多线性表的长度增加

return OK;

}

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

{

int i=lists.length,j,k;

if(i==0) return ERROR;

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

if(strcmp(lists.elem[j].name,ListName)==0){//名字相同

for(k=j;k<i-1;k++) lists.elem[k]=lists.elem[k+1];//删除该线性表

DestroyList(lists.elem[k].L);

//lists.elem[k].L=NULL;//销毁该线性表

lists.length--;//多线性表减少一个线性表

return OK;

}

}

return ERROR;

}

int LocateList(LISTS lists,char ListName[])

{

int i;

for(i=0;i<lists.length;i++)//i+1为当前到达的线性表的位置序号

{

if(strcmp(lists.elem[i].name,ListName)==0) return i+1;//找到指定线性表

}

return ERROR;

}

status UpList(LinkList &L)

{

if(!L)

return INFEASIBLE;

LinkList p,q,s;

int temp;

for(p=L->next;p->next!=NULL;p=p->next)

{

s=p;

for(q=p->next;q;q=q->next)//找到当前元素之后的最小值

{

if(q->data<s->data)

s=q;

}

if(s!=p)//数据少，交换数据域

{

temp=p->data;

p->data=s->data;

s->data=temp;

}

}

return OK;

}

status DownList(LinkList &L)

{

if(!L) return INFEASIBLE;

UpList(L);//升序

ReverseList(L);//逆置

return OK;

}

status ReverseList(LinkList &L)

{

if(!L)

return INFEASIBLE;

LinkList pre,now,next;

now=L->next;//now记录当前到达的结点的位置

pre=NULL;//pre记录逆置以后now该指向的结点

while(now){//一直到最后一个结点也逆置成功

next=now->next;//记录当前结点的下一结点

now->next=pre;//更改当前节点的下一结点，逆置

pre=now;

now=next;//当前结点与前驱结点均后移 ，对下一个结点进行操作

}

L->next=pre;//逆置结束后，pre指向最后一个元素结点，逆置后应该为首元结点

return OK;

}

status MergeList(LinkList &L,LinkList L2)

//将两个升序线性表归并为一个线性表，并且删除两个线性表中相同的值

{

if(!L||!L2)

return INFEASIBLE;

LinkList a=L->next,pre=L,temp,b=L2->next;

while(a&&b)//未到达两线性表结尾

{

while(a!=NULL&&a->data<b->data){//a的元素小，一直向后走

pre=a;a=pre->next;

}//a元素更大以后才准备插入b元素在a之前

while(a!=NULL&&b!=NULL&&a->data==b->data){//两元素相等，不插入

b=b->next;

}

if(a&&b&&a->data>b->data){//避免跳出第一个循环是相等的情况

pre->next=b;temp=b->next;b->next=a;pre=b;//将b元素插入在a之前，也就是pre之后

b=temp;//temp记录了第二个线性表中的下一个元素，并对b进行更新

}

}

if(!a) pre->next=b;//因为a达到结尾而跳出了循环，所以b有可能没插入完，此时b元素一定是大的，将第一个表的结尾指向b即可，最后一个结点时pre

return OK;

}

**附录C 基于二叉树的源程序实现**

#include<stdio.h>

#include<malloc.h>

#include<string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int KeyType;

typedef struct {

KeyType key;

char others[20];

} TElemType; //二叉树结点类型定义

typedef struct BiTNode{ //二叉链表结点的定义

TElemType data;

struct BiTNode \*lchild,\*rchild;

} BiTNode, \*BiTree;

typedef struct {

BiTree t;

char name[10];

}BNode;

typedef struct{

BNode t[10];

int len;

}BTrees;

void visit(BiTree T);//访问结点

status CreateBiTree(BiTree &T,TElemType definition[]);//创建二叉树

status ClearBiTree(BiTree &T);//清空二叉树

status BiTreeEmpty(BiTree T);//判空

int BiTreeDepth(BiTree T);//求深度

BiTNode\* LocateNode(BiTree T,KeyType e);//定位结点

status Assign(BiTree &T,KeyType e,TElemType value);//赋值结点

BiTNode\* GetSibling(BiTree T,KeyType e);//获取兄弟结点

status InsertNode(BiTree &T,KeyType e,int LR,TElemType c);//插入结点

status DeleteNode(BiTree &T,KeyType e);//删除结点

status PreOrderTraverse(BiTree T,void (\*visit)(BiTree));//先序遍历

status InOrderTraverse(BiTree T,void (\*visit)(BiTree));//中序遍历

status PostOrderTraverse(BiTree T,void (\*visit)(BiTree));//后序遍历

status LevelOrderTraverse(BiTree T,void (\*visit)(BiTree));//按层遍历

status SaveBiTree(BiTree T, char FileName[]);//保存至文件

status LoadBiTree(BiTree &T, char FileName[]);//文件创建树

status TraveTrees(BTrees trees);//遍历森林

status InsertTree(BTrees &trees,BNode T,int pos);//插入一棵树

status DeleteTree(BTrees &trees,int pos);//删除一棵树

int GetPos(BTrees trees,char s[]);//获取树的位置

status OptionTree(BiTree &T);//选择单树操作

status merge(BTrees &trees,int pos1,int pos2);//合并两棵树，哈夫曼

BTrees trees;

int pos,pos1,pos2;

TElemType definition[100];

int main()

{

trees.len=0;

BNode T;

T.t=NULL;

int i,n,flag,y;

char name[100];

printf(" Menu for Trees \n");

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

printf(" 1. 对森林遍历 2. 插入几棵树\n");

printf(" 3. 删除一棵树 4. 获取树的位置\n");

printf(" 5. 合并两棵树 6. 单树操作\n");

printf(" 0.exit \n");

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

int op=1;

while(op)

{

printf("请选择你要进行的操作[0~6]：");

scanf("%d",&op);

switch(op)

{

case 1:

if(trees.len==0)

printf("森林为空！\n");

else{

printf("森林中有%d棵树，遍历如下：\n",trees.len);

TraveTrees(trees);

printf("\n\n");

}

getchar();getchar();

break;

case 2:

printf("请输入要插入的树的个数：");

scanf("%d",&n);

y=trees.len;

for(i=1+y;i<=n+y;i++){

printf("请输入插入的第%d颗树的名字：",i);

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

flag=InsertTree(trees,T,i);

if(flag==OK) printf("插入成功！\n");

else printf("插入失败！森林已满或输入值不合法！\n");

}

getchar();getchar();

break;

case 3:

printf("请输入要删除的树的位置：");

scanf("%d",&pos);

flag=DeleteTree(trees,pos);

if(flag==ERROR) printf("输入的值不合法！\n");

else printf("删除成功！\n");

getchar();getchar();break;

case 4:

printf("请输入要查找的树的名称：");

scanf("%s",name);

flag=GetPos(trees,name);

if(flag==ERROR) printf("森林中不存在该名称的二叉树！\n");

else printf("名称为%s的树的位置为%d\n",name,flag);

getchar();getchar();

break;

case 5:

printf("请输入要合并的两个树的位置：");

scanf("%d",&pos1);scanf("%d",&pos2);

flag=merge(trees,pos1,pos2);

if(flag==ERROR) printf("输入法不合法！\n");

else {

printf("合并成功！\n");

printf("合并后的树为：\n");

printf("先序遍历为：");

flag=PreOrderTraverse(trees.t[pos1-1].t,visit);

if(flag==INFEASIBLE) printf("此树为空!");

printf("\n");

printf("中序遍历为: ");

flag=InOrderTraverse(trees.t[pos1-1].t,visit);

if(flag==INFEASIBLE) printf("此树为空！");

printf("\n");

}

getchar();getchar();break;

case 6:

printf("请输入要操作的单树在森林中的序号：");

scanf("%d",&pos);

if(pos<=0||pos>trees.len) printf("输入的值不合法！\n");

else{

flag=OptionTree(trees.t[pos-1].t);

if(flag==OK) printf("对单树操作完毕！\n");

printf(" Menu for Trees \n");

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

printf(" 1. 对森林遍历 2. 插入几棵树\n");

printf(" 3. 删除一棵树 4. 获取树的位置\n");

printf(" 5. 合并两棵树 6. 单树操作\n");

printf(" 0.exit \n");

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

}

case 0:

getchar();getchar();

break;

}

}

printf("欢迎下次使用本系统！\n");

return OK;

}

status OptionTree(BiTree &T)

//对单个二叉树进行操作

{

int i,flag;

KeyType k;

TElemType value;

BiTree a=NULL;

char filename[100];

printf(" \n Menu for Linear Table \n");

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

printf(" 1. CreateBiTree 2. ClearBiTree\n");

printf(" 3. BiTreeEmpty 4. BiTreeDepth\n");

printf(" 5. LocateNode 6. Assign\n");

printf(" 7. GetSibling 8. InsertNode\n");

printf(" 9. DeleteNode 10.PreOrderTraverse\n");

printf(" 11.InOrderTraverse 12. PostOrderTraverse\n");

printf(" 13.LevelOrderTraverse 14.SaveBiTree\n");

printf(" 15.LoadBiTree 0.exit\n");

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

int op=1;

while(op)

{

printf("请输入要进行的操作[0-15]:");

scanf("%d",&op);

switch(op)

{

case 1:

printf("请输入一组先序遍历得到的数组，空结点用0，null表示，最后以-1结束\n");

i=0;

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

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

scanf("%s",definition[i].others);

i++;

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

}

flag=CreateBiTree(T,definition);

if(flag==INFEASIBLE) printf("树已存在！不可再创建！\n");

else printf("创建完毕!\n");

getchar();getchar();break;

case 2:

flag=ClearBiTree(T);

if(flag==INFEASIBLE) printf("树不存在，无法清空！\n");

else printf("树已清空!\n");

getchar();getchar();

break;

case 3:

flag=BiTreeEmpty(T);

if(flag==TRUE) printf("树为空！\n");

else printf("树不为空！\n");

getchar();getchar();break;

case 4:

flag=BiTreeDepth(T);

if(flag==INFEASIBLE) printf("树为空！深度为0！\n");

else printf("该树的深度为%d\n",flag);

getchar();getchar();break;

case 5:

printf("请输入要查询的结点的关键字：");

scanf("%d",&i);

a=LocateNode(T,i);

if(!a) printf("二叉树中没有关键字为%d的树!\n",i);

else printf("关键字为%d的结点的名称为%s!\n",i,a->data.others);

getchar();getchar();break;

case 6:

printf("请输入要赋值的结点的关键字：");

scanf("%d",&i);

printf("请输入要赋值的关键字和名称：");

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

flag=Assign(T,i,value);

if(flag==INFEASIBLE) printf("树不存在！无法赋值！\n");

else if(flag==ERROR) printf("关键字不合法！赋值失败！\n");

else printf("赋值成功！\n");

getchar();getchar();break;

case 7:

printf("请输入关键字以获取兄弟结点：");

scanf("%d",&i);

a=NULL;

a=GetSibling(T,i);

if(a==NULL) printf("该关键字不合法或者其兄弟结点为空！\n");

else printf("该关键字的兄弟结点的数据域为: %d,%s\n ",a->data.key,a->data.others);

getchar();getchar();break;

case 8:

printf("请输入要插入的结点位置的关键字：");

scanf("%d",&k);

printf("请输入插入位置，根节点输入-1，左孩子输入0，右孩子输入1：");

scanf("%d",&i);

printf("请输入要插入的结点的数据域,关键字与字符串：");

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

flag=InsertNode(T,k,i,value);

if(flag==ERROR) printf("插入失败！\n");

else{

printf("插入成功！\n给出新树的遍历：\n");

printf("先序遍历为： ");

PreOrderTraverse(T,visit);

printf("\n");

printf("中序遍历为： ");

InOrderTraverse(T,visit);

printf("\n");

}

getchar();getchar();break;

case 9:

printf("请输入要删除的结点的关键字：");

scanf("%d",&k);

flag=DeleteNode(T,k);

if(flag==INFEASIBLE) printf("树为空，无法删除！\n");

else if(flag==ERROR) printf("结点删除失败！\n");

else{

printf("结点删除成功！\n给出新树的遍历：\n");

printf("先序遍历为： ");

flag=PreOrderTraverse(T,visit);

if(flag==INFEASIBLE) printf("树为空！");

printf("\n");

printf("中序遍历为： ");

flag=InOrderTraverse(T,visit);

if(flag==INFEASIBLE) printf("树为空！");

printf("\n");

}

getchar();getchar();break;

case 10:

printf("对该树的先序遍历为：");

flag=PreOrderTraverse(T,visit);

if(flag==INFEASIBLE) printf("树为空！");

printf("\n");

getchar();getchar();break;

case 11:

printf("对该树的中序遍历为： ");

flag=InOrderTraverse(T,visit);

if(flag==INFEASIBLE) printf("树为空！");

printf("\n");

getchar();getchar();break;

case 12:

printf("对该树的后序遍历为： ");

flag=PostOrderTraverse(T,visit);

if(flag==INFEASIBLE) printf("树为空！");

printf("\n");

getchar();getchar();break;

case 13:

printf("对该树的按层遍历为： ");

flag=LevelOrderTraverse(T,visit);

if(flag==INFEASIBLE) printf("树为空！");

printf("\n");

getchar();getchar();break;

case 14:

printf("请输入文件名：");

scanf("%s",filename);

flag=SaveBiTree(T,filename);

if(flag==INFEASIBLE) printf("树为空！写入失败！\n");

else printf("写入文件成功！\n");

getchar();getchar();break;

case 15:

printf("请输入文件名：");

scanf("%s",filename);

flag=LoadBiTree(T,filename);

if(flag==INFEASIBLE) printf("树已存在！写入失败！\n");

else printf("二叉树写入成功！\n");

getchar();getchar();break;

case 0:

getchar();

break;

}

}

}

void visit(BiTree T)

//访问结点

{

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

}

status TraveTrees(BTrees trees)

//遍历森林

{

int i=trees.len,j,flag;

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

printf("第%d个树的名称为%s\n",j+1,trees.t[j].name);

printf("第%d个二叉树的遍历为：\n",j+1);

printf("先序遍历为：");

flag=PreOrderTraverse(trees.t[j].t,visit);

if(flag==INFEASIBLE) printf("此树为空!");

printf("\n");

printf("中序遍历为: ");

flag=InOrderTraverse(trees.t[j].t,visit);

if(flag==INFEASIBLE) printf("此树为空！");

printf("\n");

}

return OK;

}

status InsertTree(BTrees &trees,BNode T,int pos)

//插入一个二叉树

{

int length=trees.len;

if(length>=10) return ERROR;

if(pos>length+1) return ERROR;

int i;

for(i=length-1;i>=pos-1;i--){

trees.t[i+1]=trees.t[i];

}

trees.t[pos-1]=T;

trees.len++;

return OK;

}

status DeleteTree(BTrees &trees,int pos)

//删除一个树

{

int length=trees.len;

if(pos<0||pos>length) return ERROR;

int i;

for(i=pos-1;i<length;i++){

trees.t[i]=trees.t[i+1];

}

trees.len--;

free(trees.t[length-1].t);

trees.t[length-1].t=NULL;

return OK;

}

int GetPos(BTrees trees,char s[])

//获取某个树的位置

{

int length=trees.len;

int i;

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

if(strcmp(s,trees.t[i].name)==0) return i+1;

}

return ERROR;

}

status merge(BTrees &trees,int pos1,int pos2)

//合并两个二叉树为一个

{

int length=trees.len;

if(pos1<=0||pos1>length) return ERROR;

if(pos2<=0||pos2>length) return ERROR;

if(pos1==pos2) return ERROR;

BiTree a=NULL;

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

a->lchild=NULL;a->rchild=NULL;

printf("请输入合并后的根节点的关键字与字符：");

scanf("%d %s",&a->data.key,a->data.others);

a->lchild=trees.t[pos1-1].t;

a->rchild=trees.t[pos2-1].t;

trees.t[pos1-1].t=a;

//DeleteTree(trees,pos2);

return OK;

}

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

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

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

{

if(T) return INFEASIBLE;//T本身已经存在了，无法再创造

int i,j,top=0;//top表示栈有多少个元素

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

T->lchild=NULL;T->rchild=NULL; //初始化根节点

BiTree a=T,b,s[100];//初始化栈s，a表示当前要创造的结点

for(i=0;definition[i].key!=-1;i++){//-1为结束点

if(definition[i].key>0)//0表示该节点为空，当前不为空

{

for(j=i-1;j>=0;j--){//查找关键字是否重复

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

}

a->data=definition[i];s[top++]=a;//对当前结点进行赋值并且入栈

if(definition[i+1].key>0){//左孩子不为空

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

b->lchild=NULL;b->rchild=NULL; //左孩子初始化

a->lchild=b;//a指向左孩子

a=b;b=NULL; //开始创造左子树 ,b保持null

}

}

else if(definition[i].key==0){//当前节点为空

if(top){//如果栈不为空 ，即当前创造的并非最后一个结点

a=s[--top];//出栈，创造上个结点的右孩子

if(definition[i+1].key>0){//右孩子不为空

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

b->lchild=NULL;b->rchild=NULL;//初始化右孩子

a->rchild=b;//右孩子初始化

a=b;b=NULL;//a指向右孩子，开始创造右孩子

}

}

}

}

return OK;

}

status ClearBiTree(BiTree &T)

//将二叉树设置成空，并删除所有结点，释放结点空间

{

if(!T) return INFEASIBLE;//T不存在，无法清空

int top=0;//栈为空

BiTree s[100],a=T,b;//初始化栈

if(a==NULL) return OK;

while(a||top){//销毁结点操作

while(a->lchild){

s[top++]=a;

a=a->lchild;

}

b=a->rchild;

free(a);

if(top){

a=s[top-1];a->lchild=NULL;

}

if(b) a=b;

else if(top) a=s[--top];

else a=NULL;

}

T=NULL;//根节点也被销毁了

return OK;

}

status BiTreeEmpty(BiTree T)

//判断树是否为空

{

if(T) return FALSE;

else return TRUE;

}

int BiTreeDepth(BiTree T)

//求二叉树T的深度

{

if(!T) return INFEASIBLE;//树不存在

BiTree s[100],a=T,pre;//初始化栈

int depth[100],k,top=0,i,max=0;

for(k=0;k<100;k++) depth[k]=0;//初始化每个结点的深度值为0

if(a) depth[T->data.key]=1,max=1;//初始化根节点的深度

while(a||top){//到达最右边的结点的左孩子，结点遍历完毕

if(!a&&top) a=s[--top];//上一个节点的左孩子不存在，出栈一个结点即最近的右孩子给当前节点

i=a->data.key;//记录当前结点的关键字

if(a->lchild){//当前结点左孩子存在

k=a->lchild->data.key;

depth[k]=depth[i]+1;//左孩子的深度为双亲结点的深度值加一

if(depth[k]>max) max=depth[k];//更新最大深度值max

}

if(a->rchild){//当前节点的右孩子存在

k=a->rchild->data.key;//记录关键字

depth[k]=depth[i]+1;//右孩子的深度值为双亲的深度值加一

if(depth[k]>max) max=depth[k];//更新最大深度值max

s[top++]=a->rchild;//右孩子入栈保存，先序遍历

}

a=a->lchild;//a指向下一个结点，左孩子

}

return max;

}

BiTNode\* LocateNode(BiTree T,KeyType e)

//查找结点

{

BiTree a=T,s[100];//初始化栈

int top=0;//栈为空

while(a||top){//遍历到最右边的结点的左空孩子

if(!a) a=s[--top];//上一个节点的左孩子为空，出栈一个结点

if(a->data.key==e) return a;//找到要的结点

if(a->rchild) s[top++]=a->rchild;//右孩子入栈

a=a->lchild;//继续遍历左子树

}//while完毕

return NULL;//遍历结束没有指定的值

}

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

//实现结点赋值。

{

if(!T) return INFEASIBLE;//树不存在

BiTree a=T,s[100],b=NULL;//初始化栈

int top=0,i,k[100];//栈为空

for(i=0;i<100;i++) k[i]=0;//初始化k[i],表示关键字i还未出现

while(a||top){//遍历到最右结点的左空孩子

if(!a) a=s[--top];//双亲节点的左孩子为空，出栈

i=a->data.key;//关键字记录

k[i]=1;//该关键字出现过

if(a->data.key==e){//找到要赋值的结点

b=a;

}

if(a->rchild) s[top++]=a->rchild;//右孩子入栈

a=a->lchild;//a指向左子树，遍历左子树

}//while完毕，所有关键字均已做记录

if(b){//找到了要赋值的结点

if(value.key==e){//该关键字与原关键字相同

b->data=value;//赋值

return OK;

}

for(i=0;i<100;i++)//查找该关键字是否与其他关键字重复

if(k[i]&&i==value.key) return ERROR;//与其他关键字重复了

b->data=value;//未重复，赋值

return OK;

}

return ERROR;//未找到指定结点

}

BiTNode\* GetSibling(BiTree T,KeyType e)

//实现获得兄弟结点

{

BiTree a=T,s[100];//初始化栈

int top=0;//栈为空

if(a->data.key==e) return NULL;//根节点为所要找的结点，无兄弟结点

while(a||top){//遍历

if(!a) a=s[--top];

if(a->lchild){//a的左孩子存在

if(a->lchild->data.key==e)//看左孩子的关键字是否符合

return a->rchild;//符合返回双亲的右孩子

}

if(a->rchild){//右孩子存在

if(a->rchild->data.key==e)//看右孩子的关键字是否符合

return a->lchild;//符合返回双亲的左孩子

s[top++]=a->rchild;//右孩子入栈，先序遍历

}

a=a->lchild;//继续遍历左子树

}

return NULL;//未找到指定节点，返回NULL

}

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

//插入结点。

{

BiTree a=T,b,p=NULL;

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

b->data=c;b->lchild=NULL;b->rchild=NULL;//要插入的结点初始化并赋值

BiTree s[100];//初始化栈

int top=0,i;//栈为空

while(a||top){//遍历

if(!a) a=s[--top];

i=a->data.key;//关键字记录

if(b->data.key==i) return ERROR;//关键字重复，返回ERROR

if(a->data.key==e) p=a;//记录插入的结点的双亲结点

if(a->rchild) s[top++]=a->rchild;

a=a->lchild;

}

if(LR==-1){//插入根节点

b->rchild=T;//树作为新的根节点的右子树

T=b;//更新根节点

return OK;

}

if(p){

if(LR==0){//插入在左孩子处

b->rchild=p->lchild;//原来的节点的孩子记录在插入结点的右孩子

p->lchild=b;

return OK;

}

if(LR==1){//插入再右孩子处

b->rchild=p->rchild;

p->rchild=b;

return OK;

}

}

return ERROR;//未找到指定的结点

}

status DeleteNode(BiTree &T,KeyType e)

//删除结点。

{

if(!T) return INFEASIBLE;//树不存在

BiTree a=T,s[100],b=NULL,pa[100],pre=NULL;//初始化栈

int top=0,i;//栈为空

for(i=0;i<100;i++) pa[i]=NULL;//记录双亲 结点

while(a->lchild||a->rchild||top){//遍历到最右边的结点

if(a->data.key==e){//找到要删除的结点

b=a;//记录

if(b->lchild&&b->rchild){//左右孩子都存在，将右孩子放置在左孩子的最右结点的右孩子处

a=b->lchild;//a指向左子树找到最右结点

top=0;//初始化栈为空

}

else break;//孩子数小于2，不需要找到左孩子的最右结点

}

if(a->lchild){//左孩子存在

i=a->lchild->data.key;

pa[i]=a;//对左孩子记录双亲结点

}

if(a->rchild){//右孩子存在

i=a->rchild->data.key;

pa[i]=a;//对右孩子进行双亲记录

s[top++]=a->rchild;//右孩子入栈

}

if(a->lchild) a=a->lchild;//左子树存在，遍历左子树

else if(top) a=s[--top];//左子树不存在，出栈一个结点

}

if(a->data.key==e) b=a;

if(b){//要删除的结点存在

i=b->data.key;//关键字记录

pre=pa[i];//找到双亲

if(!pre){//删除的是根结点

if(!b->lchild) T=b->rchild;//左孩子不存在，直接更新为右孩子

else if(!b->rchild) T=b->lchild;//更新为左孩子

else{//左右孩子都存在

a->rchild=b->rchild;//左孩子的最右结点记录右子树

T=b->lchild;//更新根节点

}

free(b);//释放要删除的节点的空间

return OK;

}

if(pre->lchild==b){//要删除为左结点

if(!b->lchild) pre->lchild=b->rchild;//记录右孩子

else if(!b->rchild) pre->lchild=b->lchild;//记录左孩子

else{//左右孩子都存在

pre->lchild=b->lchild;//更新为左孩子

a->rchild=b->rchild;//右孩子插入左子树最右节点的右孩子

}

free(b);

return OK;

}

else{//要删除右节点

if(!b->lchild) pre->rchild=b->rchild;//记录右子树

else if(!b->rchild) pre->rchild=b->lchild;//记录左子树

else{//左右子树都存在

pre->rchild=b->lchild;

a->rchild=b->rchild;

}

free(b);

return OK;

}

}

return ERROR;

}

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

//先序遍历二叉树T

{

if(!T) return INFEASIBLE;//树不存在

BiTree a=T,s[100];//初始化栈

int top=0;//栈为空

while(a||top){

if(!a) a=s[--top];

visit(a);//输出该节点

if(a->rchild) s[top++]=a->rchild;//记录右子树

a=a->lchild;//遍历左子树

}

return OK;

}

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

//中序遍历二叉树T

{

if(!T) return INFEASIBLE;//树不存在

BiTree a=T,s[100];//初始化栈

int top=0,flag=0;//栈为空

while(a||top){

while(a){//找到最左的结点

s[top++]=a;//经过但未访问的结点入栈保存

a=a->lchild;

}

a=s[--top];//记录当前遍历的子树的最左结点

visit(a);//访问 当前相对根结点

while(!a->rchild&&top){//该节点的左子树与根节点已遍历，右子树不存在，向上遍历双亲结点

a=s[--top];

visit(a);

}

a=a->rchild;//右子树存在，遍历右子树

}

return OK;

}

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

//后序遍历二叉树T

{

if(!T) return INFEASIBLE;//树不存在

BiTree a=T,s[100],pre=NULL;//初始化栈

int top=0,flag;//栈为空

if(a){//根节点不为空

do{

while(a){//找到最左结点

s[top++]=a;

a=a->lchild;

}

pre=NULL;flag=1;//上一个遍历的结点pre为空

while(top&&flag){//找到

a=s[top-1];//定位当前可能要遍历的结点

if(a->rchild!=pre){//该节点的右孩子未被遍历，遍历右子树，

a=a->rchild;

flag=0;//结束循环

}

else{//左右孩子均已被遍历，该节点遍历

visit(a);

pre=a;//更新上一个被遍历的结点

top--;//被遍历后出栈

}

}

}while(top);//栈为空即根节点已经被遍历完毕，整个树被遍历完毕

}

return OK;

}

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

//按层遍历二叉树T

{

if(!T) return INFEASIBLE;//树不存在

BiTree bro[100],a=T,b=NULL;//bro数组记录兄弟结点，下一个兄弟结点

int i,k;

for(i=0;i<100;i++) bro[i]=NULL;//初始化

while(a){//a为每一层中最左边的结点

k=0;

b=a;//第一个结点赋值

while(b){//遍历这一层

i=b->data.key;//关键字赋值

visit(b);//对这一层的结点赋值

//对下一层的结点进行兄弟结点的记录

if(b->lchild&&k==0) k=b->lchild->data.key;//当前的结点为下一层的第一个结点，记录关键字

else if(b->lchild){//左孩子不是第一个结点

bro[k]=b->lchild;//左边的结点赋值兄弟结点

k=b->lchild->data.key;//更新关键字为当前节点

}

if(b->rchild&&k==0) k=b->rchild->data.key;//右孩子为第一个结点，记录关键字

else if(b->rchild){//右孩子不是第一个结点，

bro[k]=b->rchild;//对左边的结点记录兄弟结点

k=b->rchild->data.key;//更新关键字，为当前节点

}

b=bro[i];//遍历下一个兄弟节点

}

b=a;

while(b){//遍历找到下一层的第一个结点

if(b->lchild){//当前结点左孩子存在

b=b->lchild;//下一结点

break;//跳出循环

}

if(b->rchild){//右孩子为下一层第一个结点

b=b->rchild;

break;

}

i=b->data.key;

b=bro[i];//这个节点的孩子不是第一个结点，找下一个

}

a=b;//a向下一层遍历，更新为下一层的第一个结点

}

return OK;

}

status SaveBiTree(BiTree T, char FileName[])

//将二叉树的结点数据写入到文件FileName中

{

if(!T) return INFEASIBLE;//树不存在

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

if(!file) return ERROR;

BiTree a=T,s[100];

int top=0,k=0;

char t[10]="null";

while(a){//先序遍历写入文件

fprintf(file,"%d %s ",a->data.key,a->data.others);//空格格式！

if(a->rchild) s[top++]=a->rchild;//右孩子入栈

if(a->lchild) a=a->lchild;//遍历左子树

else if(top){//左子树不存在

fprintf(file,"%d %s ",k,t);//写入空孩子结点

if(!a->rchild) fprintf(file,"%d %s ",k,t);//右孩子为空时也写入

a=s[--top];//出栈写下一个结点

}

else{//写到最后一个结点

a=NULL;

fprintf(file,"%d %s ",k,t);

fprintf(file,"%d %s ",k,t);//将左右孩子为空补上

}

}

k=-1;

fprintf(file,"%d %s",k,s);//输入-1表示二叉树写入完毕

fclose(file);//关闭文件

return OK;

}

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

//读入文件FileName的结点数据，创建二叉树

{

if(T) return INFEASIBLE;//树存在

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

if(!file) return ERROR;

BiTree a,b=NULL,p[100];

int k,i,top=0;

char s[10];

fscanf(file,"%d",&k);

if(k==-1) return OK;

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

T->lchild=NULL;T->rchild=NULL;

a=T;

while(k!=-1){//-1终止

if(k){//当前创立的结点不为空结点

a->data.key=k;//赋值数据域

fscanf(file,"%s",a->data.others);//数据域

fscanf(file,"%d",&i);//读取下一个结点的值

if(i>0){//大于0也就是左孩子存在

p[top++]=a;//当前结点入栈

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

b->lchild=NULL;b->rchild=NULL;

a->lchild=b;a=b;b=NULL;//更新要创建的当前结点

}

else fscanf(file,"%s",s);//读取null

k=i;//更新当前数据域

}

else{//当前创立的节点为空结点

fscanf(file,"%d",&i);//读取下一结点

if(i>0){//不为0即为创建右孩子

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

b->lchild=NULL;b->rchild=NULL;

a->rchild=b;a=b;b=NULL;//右孩子创建完毕后更新当前创建结点

}

else if(i==0){//右孩子也为0

while(i==0){//将最近的叶子结点都过滤掉

if(top) a=p[--top];

fscanf(file,"%s",s);

fscanf(file,"%d",&i);

}

if(i!=-1){//不是最后一个结点的话，就是要创建一个结点的右孩子了

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

b->lchild=NULL;b->rchild=NULL;

a->rchild=b;a=b;b=NULL;

}

}

k=i;//下一个结点的值更新为i

}

}

fclose(file);

return OK;

}