華中科技大學

课 程 实 验 报 告

课程名称： 数据结构实验

专业班级 CS2205

学　　号 U202215510

姓　　名 徐新陬

指导教师 王雄

报告日期 2022 年 6 月 10 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 1
1.3　系统实现 1
1.4　系统测试 5
1.5　实验小结 7
2　基于邻接表的图实现 9
2.1　问题描述 9
2.2　系统设计 9
2.3　系统实现 10
2.4　系统测试 17
2.5　实验小结 20
3　课程的收获和建议 21
3.1　基于顺序存储结构的线性表实现 21
3.2　基于链式存储结构的线性表实现 22
3.3　基于二叉链表的二叉树实现 22
3.4　基于邻接表的图实现 22
参考文献 24
附录 A 基于顺序存储结构线性表实现的源程序 25
附录 B 基于链式存储结构线性表实现的源程序 53
附录 C 基于二叉链表二叉树实现的源程序 83
附录 D 基于邻接表图实现的源程序 122

1 基于顺序存储结构的线性表实现

1.1 问题描述

采用顺序表作为线性表的物理结构，实现 1.2 小节的运算。其中 ElemType 为数据元素的类型名，具体含义可自行定义，其它有关类型和常量的定义和引用详见文献 [1] 的 p10。

构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。以达到解决以下问题的目的：（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，熟练掌握顺序表基本运算的实现。

1.2 系统设计

线性表是一种基本的数据结构，它由多个元素组成，元素之间存在线性关系，即元素之间的顺序是固定的。而基于顺序存储结构的线性表实现是一种静态数据结构，其大小在创建时就已经确定，元素之间存在物理上的连续关系，本次实验旨在探究线性表的基本原理和应用场景，通过实验来深入理解线性表的操作方法和性能特点。

该系统通过编程实现顺序表的基本操作和额外操作，构造了一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示，使用者可以通过基于顺序存储结构的线性表实现一些基础的函数功能，并通过不同函数和功能的组合来实现对数据的处理。。

1.3 系统实现

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

（1）初始化表：函数名称是 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；
首先判断 L 是否为 NULL，若为 NULL 则通过 malloc 函数分配一定空间，反之若则返回 INFEASIBLE。

（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；
首先判断 L 是否为 NULL，若 L 不为 NULL 则通过 free(L.elem)，并且使 L.length=0，反之则返回 INFEASIBLE。

（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；
首先判断 L 是否为 NULL，若 L 不为 NULL 则使 L.length=0，反之则返回 INFEASIBLE。

（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；
首先判断 L 是否为 NULL，若 L 不为 NULL 则进行判断 L.length 是否为 0，若为 0 则返回 TRUE，反之则返回 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；
首先判断 L 是否为 NULL，若 L 不为 NULL 则返回 L.length，反之则返回 INFEASIBLE。

（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；
首先判断 L 是否为 NULL，若 L 不为 NULL，首先判断 i>L.length||i<1，若 i 为此范围则返回 ERROR，反之则令 e=L.elem[i-1]，返回 OK。

（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；
首先判断 L 是否为 NULL，若 L 不为 NULL, 则 for(i=0;i<=L.length;i++)，同时进行判断 if(e==L.elem[i])，如果判断通过则 return i，反之 return 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；
首先判断 L 是否为 NULL，若 L 不为 NULL, 则 for(i=1;i<L.length;i++)，查找元素 e 的位置，若找到了，则令 pre=L.elem[i-1]，并且返回 OK。反之则返回 ERROR。

（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；
首先判断 L 是否为 NULL，若 L 不为 NULL, 则 for(i=0;i<L.length-1;i++)，查找元素 e 的位置，若找到了，则令 next=L.elem[i+1]，并且返回 OK。反之则返回 ERROR。

（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。
首先判断 L 是否为 NULL，若 L 不为 NULL, 则看是否 L.length+1>L.listsize，如果成立，则 realloc 重新分配空间，判断完以后，进行判断 if(i<1||i>L.length+1)，如果 i 的值合法，则从 i 往后的数组元素依次后移，最后令 L.elem[i-1]=e，L.length++ 并返回 OK。

（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 返回的值；
首先判断 L 是否为 NULL，若 L 不为 NULL, 则与插入元素同理，令 i 之后的元素依次前移一格，并且 L.length–, 返回 OK.

（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()。
首先判断 L 是否为 NULL，若 L 不为 NULL，并且表的长度合理，则 for(i=0;i<L.length-1;i++){ printf("%d ",L.elem[i]); } 并返回 OK.

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；
首先判断 L 是否为 NULL，若 L 不为 NULL，则通过动态规划的方法求最大连续子数组和，首先初始化 dp[0]=0，然后 dp[i]=max(dp[i-1]+num[i],num[i])，res=max(dp[i],res) 对 i 从 1 到 L.length 迭代，最后返回 res.

（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；
首先判断 L 是否为 NULL，若 L 不为 NULL，则采用暴力累加的方法求和为 k 的子数组个数，先选取 i 为子数组首元素下标，然后考虑以 i 为起点的所有子数组，若存在符合要求的子数组，则 res++

```
for(i=0;i<L.length;i++){
sum=0;
for(j=i;j<L.length;j++){
sum=sum+L.elem[j];
if(sum==k)
res++;
} }
```

（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；
首先判断 L 是否为 NULL，若 L 不为 NULL，则采用冒泡排序法进行顺序表的排序

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

```
for(j=i;j<L.length;j++){
if(L.elem[j]<L.elem[i]){
temp=L.elem[j];
L.elem[j]=L.elem[i];
L.elem[i]=temp;

}
}
```

（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。

对于 SaveList 首先判断 L 是否为 NULL，若 L 不为 NULL,FILE* fp=fopen(FileName,"w+") 并且从下标 0 到 L.length-1 逐步写入，最后 fclose（fp）。

对于 LoadList 首先判断 L 是否为 NULL，若 L 为 NULL，则 fp=fopen(FileName,"r+"); 随后 while((ch=getc(fp))!=EOF) 输入字符，然后记录输入的字符数 i，最后令 L.length=i, fclose（fp）并 return OK。

（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。

首先 j=InitList(Lists.elem[Lists.length].L) 来添加空线性表，然后 strcpy(Lists.elem[Lists.length].name,ListName) 来输入表名，最后 Lists.length++。删除时通过查找找到相应线性表下标 i，令该下标后依次前移一格，在查找线性表时与查找元素相同，for(i=0;i<=Lists.length;i++)，同时进行判断，若判断通过则返回相应下标地址

1.4 系统测试

各个函数的正常和异常的测试用例及测试结果如下图，若使用正确，菜单则将给出正确反馈，若使用错误，菜单则会指出错误原因，并要求使用者重新输入

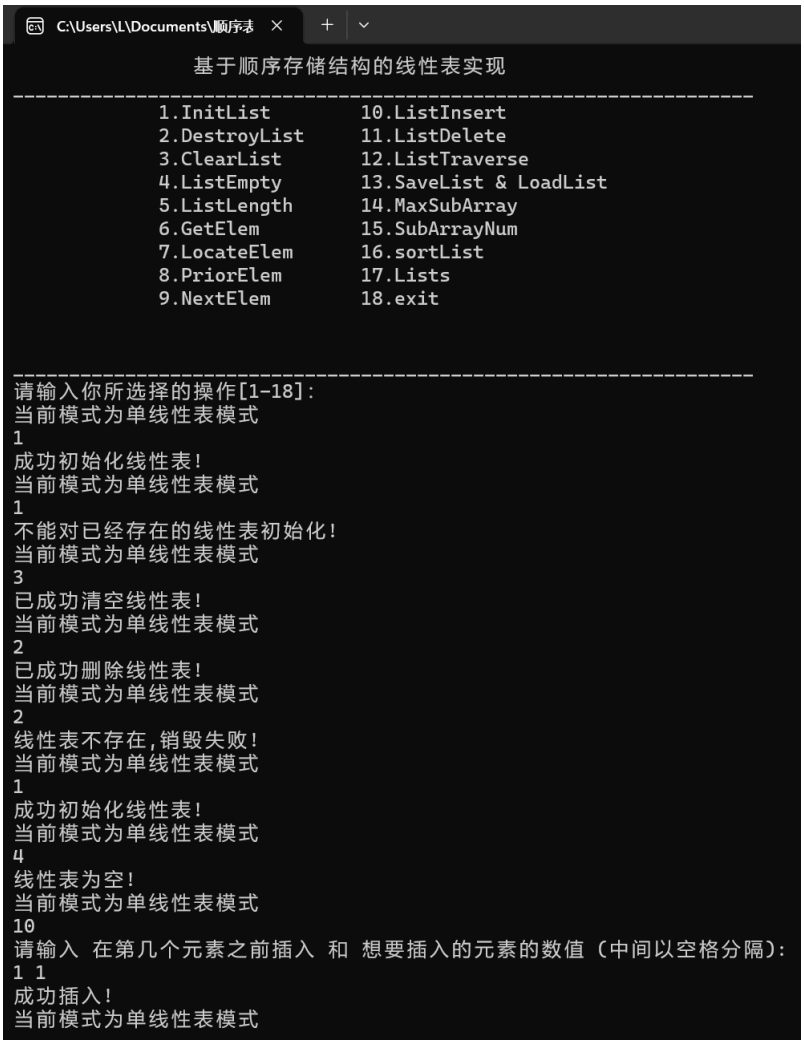

图 1-1　基于顺序存储结构的线性表实现系统测试 1

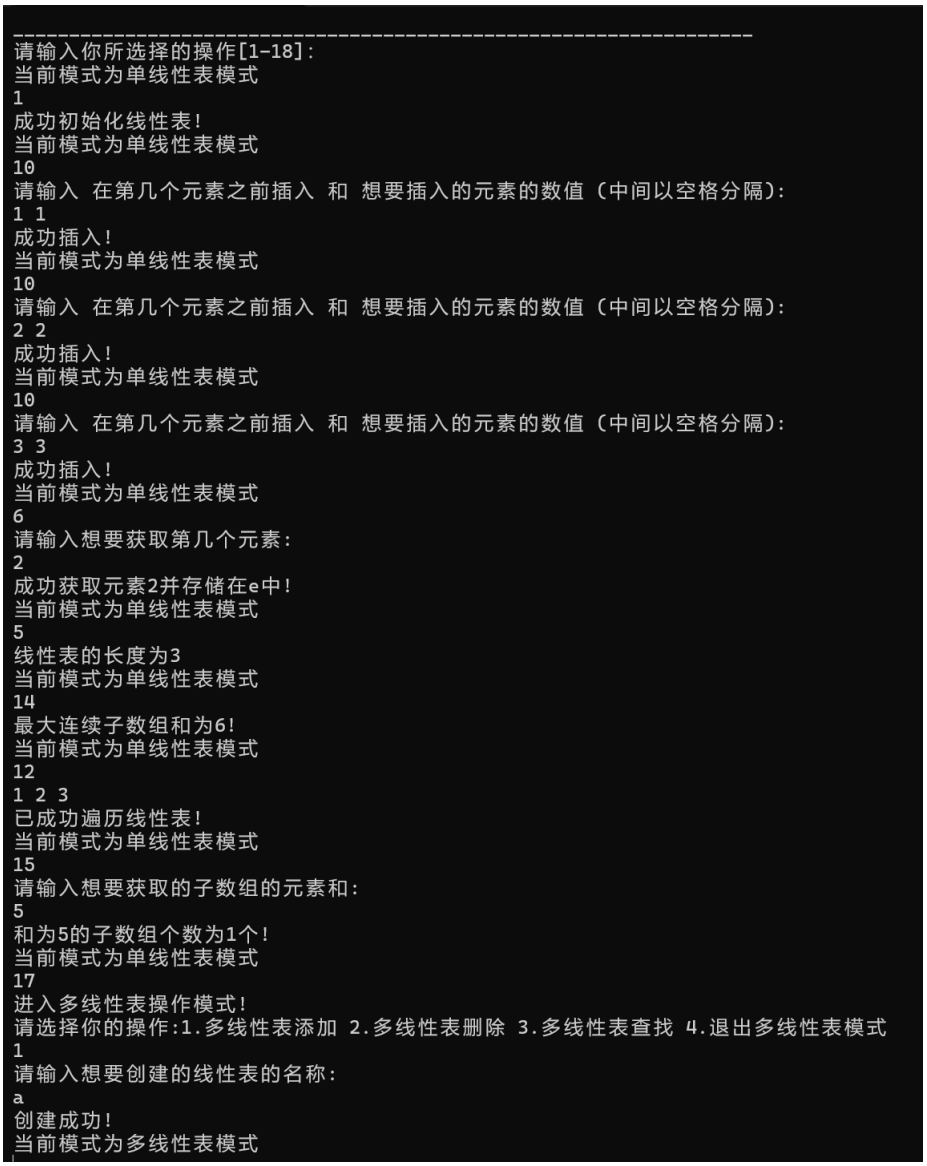

图 1-2　基于顺序存储结构的线性表实现系统测试 2

1.5　实验小结

在实现顺序表的操作时，我碰到了几个个调试中的典型错误，其一是初始化线性表的时候，可能会出现重复初始化的问题，解决这个问题的关键是在注意初始化线性表函数 InitList(L) 的判定条件，即一定要判断 L 是否为空表，此外，该问题的另一个难点是与 DestroyList(L) 和 ClearList(L) 函数相匹配，在 DestroyList(L) 最后一定要使 L=NULL，而在 ClearList(L) 则不能使 L=NULL。

其二是关于数组下标越界的问题，如在 ListInsert(L,i,e) 和 ListDelete(L,i,e) 中，涉及到了线性表内元素个数的改变，而由于数组本身长度一定的特性，解决该问题通常需要特判或者 realloc 重新分配空间，此外这个问题在程序中很难被

发现，因为下标越界的情况在运行时才会出现，而且错误信息往往比较模糊，很难定位问题所在。这边要求代码具有健壮性，对于某些处于越界边缘的操作数据，可以通过特判识别出，并单独进行特殊处理，在程序中添加下标合法性判断，确保下标始终处于合法范围内。这种方法需要在程序中添加额外的代码，但是可以有效地避免下标越界的问题。

其三是关于多线性表的操作问题，为实现可以灵活对于多线性表中的每个线性表进行操作，在程序中专门设置了一个全局变量 whe 来判断是否进入多线性表操作模式，若 whe=1，即系统进入多线性表模式，此时每次调用函数时都会增加一个选项，即对何线性表进行操作，以此来解决多线性表的操作需求。

这些通过实践得到的经验和教训，使我深刻认识到编写代码时应该注意细节，尽可能地避免潜在的问题，以提高程序的可靠性和可维护性，也要求我注意程序的正确性，健壮性和可读性。同时，也学会了如何使用基于顺序存储结构的线性表实现来应对不同的问题，对我的编程能力有了很大的提升。

2 基于邻接表的图实现

2.1 问题描述

采用邻接表作为图的物理结构，实现 4.2 节的基本运算，可任选无向图、有向图、无向网和有向网这四种图中的一种实现。其中 ElemType 为数据元素的类型名，具体含义可自行定义，但要求顶点类型为结构型，至少包含二个部分，一个是能唯一标识一个顶点的关键字（类似于学号或职工号），另一个是其它部分。其它有关类型和常量的定义和引用详见文献 [1] 的 p10。

构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示，通过实验达到：（1）加深对图的概念、基本运算的理解；（2）熟练掌握图的逻辑结构与物理结构的关系；（3）以邻接表作为物理结构，熟练掌握图基本运算的实现。

2.2 系统设计

图的整体系统结构设计和数据结构设计主要包括以下几个方面：

首先要确定图的抽象数据类型，在实验中，需要定义一个抽象数据类型，用于表示图。通常情况下，图由顶点、边和权值等要素组成，需要定义相应的操作和函数，如创建图、添加顶点、添加边、删除顶点、删除边、遍历图等。

接下来需要设计图的数据结构，需要选择合适的数据结构来表示图。通常情况下，图可以使用邻接表、邻接矩阵、逆邻接表等数据结构来表示。每种数据结构有其适用的场景和特点，需要根据实际情况进行选择。在本次实验中，选择用领接表来表示图。

还需设计图的算法，以实现一些基本的算法，如深度优先搜索、广度优先搜索、最短路径算法等。这些算法可以对图进行遍历和搜索，从而实现对图进行综合操作和分析。

最后在编写完成后，还要进行调试和排错，在实验中，可能会遇到各种各样错误，如堆栈错误、编译错误、逻辑错误等。需要进行调试和排错，以解决问题并确保图最后输出结果的正确性和可靠性。

2.3 系统实现

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

（1）创建图：函数名称是 CreateCraph(G,V,VR)；初始条件是 V 是图的顶点集，VR 是图的关系集；操作结果是按 V 和 VR 的定义构造图 G；
首先判断输入图的数据是否合法，比如顶点关键字是否有重复，若有重复，则直接返回 ERROR，然后记录输入的个数，G.vexnum=vexnum;G.arcnum=arcnum; 如果输入的顶点数大于 size，则返回 ERROR，再 copy 头结点数据，最后建立指针的连接

```
for(i=0;i<arcnum;i++){
j=LocateVex(G,VR[i][0]);
p=new ArcNode;
p->adjvex=LocateVex(G,VR[i][1]);
p->nextarc=G.vertices[j].firstarc;
G.vertices[j].firstarc=p;
k=LocateVex(G,VR[i][1]);
q=new ArcNode;
q->adjvex=LocateVex(G,VR[i][0]);
q->nextarc=G.vertices[k].firstarc;
G.vertices[k].firstarc=q;
}
```

并返回 OK

（2）销毁图：函数名称是 DestroyGraph(G)；初始条件图 G 已存在；操作结果是销毁图 G；
从头结点开始，依次释放弧节点即

```
for(i=0;i<G.vexnum;i++){
if(G.vertices[i].firstarc!=NULL){
```

```
p=G.vertices[i].firstarc;
while(p->nextarc!=NULL){
ArcNode* q=p->nextarc;
free(p);
p=q;
}
free(p);
G.vertices[i].firstarc=NULL;
}
```

最后返回 OK

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

```
for(i=0;i<G.vexnum;i++){
if(u==G.vertices[i].data.key) return i;
}
```

若成功则返回下标 i, 反之返回-1

（4）顶点赋值：函数名称是 PutVex (G,u,value)；初始条件是图 G 存在，u 是和 G 中顶点关键字类型相同的给定值；操作结果是对关键字为 u 的顶点赋值 value；逐步搜索给定的 u 值，若找到则通过 strcpy 将所要修改的信息进行添加

```
for(i=0;i<G.vexnum;i++){
if(u==G.vertices[i].data.key){
G.vertices[i].data.key=value.key;
strcpy(G.vertices[i].data.others,value.others);
return OK;
}
}
```

（5）获得第一邻接点：函数名称是 FirstAdjVex(G, u)；初始条件是图 G 存在，

u 是 G 中顶点的位序；操作结果是返回 u 对应顶点的第一个邻接顶点位序，如果 u 的顶点没有邻接顶点，否则返回其它表示“不存在”的信息；
首先判断头结点后是否有弧结点，若有，则记录下 num 并返回 num
if(G.vertices[i].firstarc==NULL) return -1;
num=G.vertices[i].firstarc->adjvex;

（6）获得下一邻接点：函数名称是 NextAdjVex(G, v, w)；初始条件是图 G 存在，v 和 w 是 G 中两个顶点的位序，v 对应 G 的一个顶点,w 对应 v 的邻接顶点；操作结果是返回 v 的（相对于 w）下一个邻接顶点的位序，如果 w 是最后一个邻接顶点，返回其它表示“不存在”的信息；
在获得第一邻接点的前提下，延链表不断进行寻找，直到找到指定的值，反之返回-1.
while(p->nextarc!=NULL){
if(p->adjvex==LocateVex(G,w)){
return p->nextarc->adjvex;
}
p=p->nextarc;
}

（7）插入顶点：函数名称是 InsertVex(G,v)；初始条件是图 G 存在，v 和 G 中的顶点具有相同特征；操作结果是在图 G 中增加新顶点 v。（在这里也保持顶点关键字的唯一性）；
在判断 v 与先前顶点不重复的前提下，在头结点数组中插入顶点即可。
G.vertices[G.vexnum].data.key=v.key;
G.vertices[G.vexnum].firstarc=NULL;
strcpy(G.vertices[G.vexnum].data.others,v.others);
G.vexnum++;
最后返回 OK

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

顶点以及相关的弧；
首先找到以 v 为关键字的头结点，依次寻找另一个与 v 相连的结点，并删除相应的弧结点, 最后删除以 v 为关键字的头结点。

（9）插入弧：函数名称是 InsertArc(G,v,w)；初始条件是图 G 存在，v、w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中增加弧 <v,w>，如果图 G 是无向图，还需要增加 <w,v>；
首先判断相应的边的顶点是否存在，若不存在则直接返回 ERROR，反之则在 v,w 两个头结点后插入新的弧结点

```
ArcNode* p=(ArcNode*)malloc(sizeof(ArcNode));
p->adjvex=vex2;
p->nextarc=G.vertices[vex1].firstarc;
G.vertices[vex1].firstarc=p;
ArcNode* q=(ArcNode*)malloc(sizeof(ArcNode));
q->adjvex=vex1;
q->nextarc=G.vertices[vex2].firstarc;
G.vertices[vex2].firstarc=q;
G.arcnum++;
```

最后返回 OK

（10）删除弧：函数名称是 DeleteArc(G,v,w)；初始条件是图 G 存在，v、w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中删除弧 <v,w>，如果图 G 是无向图，还需要删除 <w,v>；
分别从 v,w 中头结点中向后寻找 <w,v>, 进行单链表的删除操作即可，若没找到则返回 ERROR, 特别要注意对第一个结点的特判

（11）深度优先搜索遍历：函数名称是 DFSTraverse(G,visit())；初始条件是图 G 存在；操作结果是图 G 进行深度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问一次，且仅访问一次；
使用 mark 数组对该结点是否被访问过进行标记，若该节点已经被访问过，则标记为 1，反之则为 0，递归调用 dfs 函数并将已经访问过的结点在 mark 中标记为

1 即可。

```
for(v=0;v<G.vexnum;v++) mark[v]=0;//初始化
for(v=0;v<G.vexnum;v++){
if(mark[v]==0) DFS(G,v,visit);
}

void DFS(ALGraph G,int v,void (*visit)(VertexType))
{
mark[v]=1;
visit(G.vertices[v].data);
int w;

for(w=FirstAdjVex(G,G.vertices[v].data.key);w>=0;w=NextAdjVex(G,G.vertices[v].
data.key,G.vertices[w].data.key)){
if(mark[w]==0) DFS(G,w,visit);
}
```

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

由于广度优先遍历可以通过队列实现，即某个节点出队后，将与之相连的节点入队，由于先进先出的性质，故可以实现广度优先搜索

```
for(v=0;v<G.vexnum;++v){
if(mark[v]==0){
mark[v]=1;
visit(G.vertices[v].data);
EnQueue(Q,G.vertices[v].data);
while(QueueEmpty(Q)!=0){
DeQueue(Q,u,G);

for(w=FirstAdjVex(G,G.vertices[u].data.key);w>=0;w=NextAdjVex(G,G.vertices[u].
```

```
data.key,G.vertices[w].data.key)){
if(mark[w]==0){
visit(G.vertices[w].data);
mark[w]=1;
EnQueue(Q,G.vertices[w].data);
}
}
}
}
}
```

附加功能：

（1）距离小于 k 的顶点集合：函数名称是 VerticesSetLessThanK(G,v,k)，初始条件是图 G 存在；操作结果是返回与顶点 v 距离小于 k 的顶点集合；
将图由领接表转化为领接矩阵，然后通过 FLOYD 算法算出矩阵中每行小于 k 的值并进行输出即可

```
void Floyd(ALGraph G)
{ //表示从 j 点到 k 点的最短路径，其中的 i 为中间的中转点
for (int i = 1; i <= G.vexnum; ++i) //i 为中转在最外层
for (int j = 1; j <= G.vexnum; ++j)
for (int k = 1; k <= G.vexnum; ++k)
arcmat[j][k] = min(arcmat[j][k], arcmat[j][i] + arcmat[i][k]);
}
```

（2）顶点间最短路径和长度：函数名称是 ShortestPathLength(G,v,w); 初始条件是图 G 存在；操作结果是返回顶点 v 与顶点 w 的最短路径的长度；
由上，通过 floyd 算法计算出直接输出 arcmat[v][w] 即可

（3）图的连通分量：函数名称是 ConnectedComponentsNums(G)，初始条件是图 G 存在；操作结果是返回图 G 的所有连通分量的个数；

通过 DFS 搜索，经过几次 dfs 循环即说明有几个联通分类, 用 cnt 计数并返回 cnt

```
for(v=0;v<G.vexnum;v++) mark[v]=0;//初始化
for(v=0;v<G.vexnum;v++){
if(mark[v]==0){
DFS0(G,v);
cnt++;
}
}
return cnt;
```

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

对于 SaveList 首先判断 G 是否为 NULL，若 G 不为 NULL,FILE* fp=fopen(FileName,”w+”) 并且从头结点下标 0 到 G.vexnum-1, 弧结点从头到尾逐步写入，最后 fclose（fp）。

对于 LoadList 首先判断 G 是否为 NULL，若 G 为 NULL，则 fp=fopen(FileName,”r+”); 随后与建立图时一样，通过与 CreateCraph(G,V,VR) 一样的方式生成图，最后 fclose（fp）并 return OK。

（5）实现多个图管理：设计相应的数据结构管理多个图的查找、添加、移除等功能。

首先 j=InitList(Lists.elem[Lists.length].G) 来添加空图，然后 strcpy(Lists.elem[Lists.length].name,ListName) 来输入表名，最后 Lists.length++。删除时通过查找找到相应线性表下标 i，令该下标后依次前移一格，在查找线性表时与查找元素相同，for(i=0;i<=Lists.length;i++)，同时进行判断，若判断通过则返回相应下标地址

2.4 系统测试

各个函数的正常和异常的测试用例及测试结果如下图，若使用正确，菜单则将给出正确反馈，若使用错误，菜单则会指出错误原因，并要求使用者重新输入

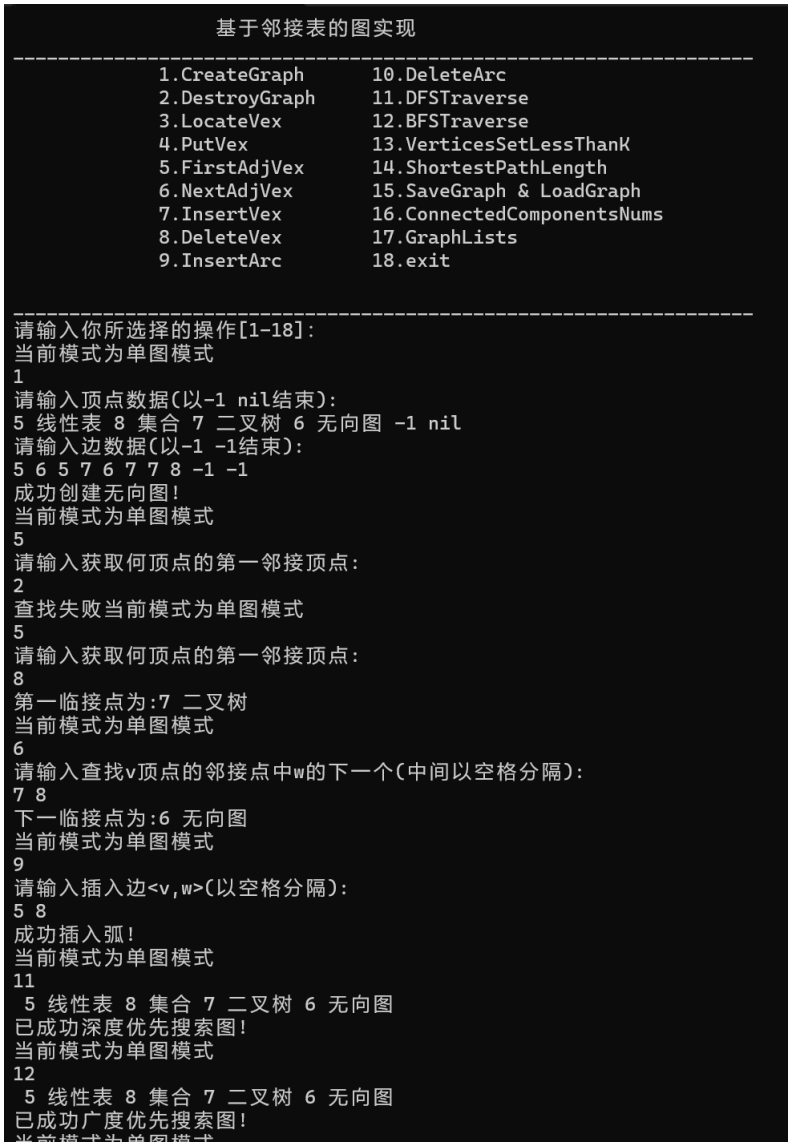

图 2-1 基于领接表的图实现系统测试 1

```
16
图的连通分量有1个!
当前模式为单图模式
10
请输入想要删除的弧<v,w>:
5 6
成功删除!
当前模式为单图模式
10
请输入想要删除的弧<v,w>:
6 7
成功删除!
当前模式为单图模式
16
图的连通分量有2个!
当前模式为单图模式
14
请输入顶点v和顶点w(以空格分隔):
5 8

顶点v到w的最短路径为2!
当前模式为单图模式
14
请输入顶点v和顶点w(以空格分隔):
6 7
两顶点间不连通!
当前模式为单图模式
3
请输入顶点关键字:
6
该顶点在G中的位序为3!
当前模式为单图模式
2
已成功销毁图!
当前模式为单图模式
2
图不存在,销毁失败!
当前模式为单图模式
1
请输入顶点数据(以-1 nil结束):
-1 nil
请输入边数据(以-1 -1结束):
-1 -1
输入的顶点或弧不正确!
当前模式为单图模式
```

图 2-2　基于领接表的图实现系统测试 2

```
请输入顶点数据(以-1 nil结束):
5 线性表 8 集合 7 二叉树 6 无向图 -1 nil
请输入边数据(以-1 -1结束):
5 6 5 7 6 7 7 8 -1 -1
成功创建无向图!
当前模式为单图模式

11
 5 线性表 7 二叉树 8 集合 6 无向图
已成功深度优先搜索图!
当前模式为单图模式
12
 5 线性表 7 二叉树 6 无向图 8 集合
已成功广度优先搜索图!
当前模式为单图模式
13
请输入获取与顶点v路径小于k的顶点中的v,k(以空格分隔):
5 1
与顶点v距离小于k的顶点集合为: 5 线性表
已成功获取与顶点v路径小于k的顶点集合!
当前模式为单图模式
14
请输入顶点v和顶点w(以空格分隔):
5 8

顶点v到w的最短路径为2!
当前模式为单图模式
15
请选择你的操作:1.将图写入文件 2.将文件中的数据读入图 3.退出该功能
1
请输入文件名
a
写入成功!
当前模式为单图模式

2
已成功销毁图!
当前模式为单图模式
15
请选择你的操作:1.将图写入文件 2.将文件中的数据读入图 3.退出该功能
2
请输入文件名
a
读取成功!
当前模式为单图模式
12
 5 线性表 7 二叉树 6 无向图 8 集合
已成功广度优先搜索图!
当前模式为单图模式
```

图 2-3　基于领接表的图实现系统测试 3

2.5 实验小结

在进行数据结构实验，探究图及其应用的过程中，可能会遇到一些典型错误，以下是我在图的实验中遇到的错误以及解决方法：

忘记初始化：在图的实现中，经常会忘记初始化图的数据结构。这可能会导致程序在运行时出现未定义的行为。解决方法是在创建图时，确保对所有数据结构进行正确的初始化，同时也要记得加入初始化图的时候正确的判断条件。

内存泄漏：在使用动态分配的内存来存储图的数据结构时，可能会导致内存泄漏。这通常是由于忘记释放动态分配的内存或释放已释放的内存引起的。解决方法是使用内存调试工具来查找问题，并确保在不再需要内存时释放它。

指针错误：在实现图的数据结构时，可能会遇到指针错误，例如指针未初始化或指针越界。这些错误可能导致程序崩溃或或产生未定义的行为。解决方法是确保指针在使用之前被正确初始化，并在某些特殊情况确保访问数组时不会超出其边界。

算法错误：在实现图算法时，可能会犯一些基本的算法错误，例如在迭代和递归图时遗漏顶点或边。这些错误可能导致算法无法正确地运行。解决方法是仔细审查算法的实现，并确保所有必要的操作都已正确执行。

测试不充分：在测试算法时，可能会测试不充分，导致无法发现错误。解决方法是尝试使用全面的测试用例，确保代码的健壮性，包括边界情况和异常情况，以确保算法的正确性。

在实现图的数据结构和算法时，需要仔细考虑每个细节，并确保代码的正确性和可靠性。同时，也要注重进行全面的测试和调试。

3 课程的收获和建议

数据结构是一门非常重要的计算机科学基础知识课程。通过学习数据结构以及进行相应的数据结构实验，我掌握了各种不同的数据存储和组织方式的有关知识，以及它们的特点和应用场合。

通过课程可以了解线性表、栈、队列、串、数组、树、图等基本数据结构，以及它们的常见操作和实现方法。我还学习了如何根据实际需求选择合适的数据结构，以及如何优化数据结构的性能，以提高程序的效率，解决相应的任务。

通过学习数据结构以及进行相应的数据结构实验，我掌握了解决算法问题的基本思路和方法。我学会了如何分析问题，设计算法，并选择合适的数据结构来实现算法。这不仅提高了我的编程能力，也提高了我的逻辑思考和解决问题的能力。

学习数据结构也让我对计算机科学的认识更加深入。数据结构是计算机科学中非常重要的基础知识，是计算机科学的核心内容之一。通过进行数据结构实验，我了解了计算机科学的基本概念和原理，感到了进行计算机科学研究的乐趣，增强了进行计算机科学探索的动力，为未来的学习和工作打下了坚实的基础。

学习数据结构以及进行数据结构实验是一项非常有益的过程。它不仅提高了我的编程能力和逻辑思考能力，也让我对计算机科学的认识更加深入。我相信这些收获将对我的未来的学习和发展产生积极的影响。

3.1 基于顺序存储结构的线性表实现

从数据结构实验——基于顺序存储结构的线性表实现中，我取得了很多方面的收获，首先我掌握了线性表的基本操作和实现方法，并了解了线性表的应用场景，从实验不断编写代码的过程中，我遇到了各种问题，如语法错误、逻辑错误等。通过自己的思考和查找资料，我逐渐解决了这些问题，这让我感受到了问题解决的快乐，也培养了我解决问题的能力。此外，一些难度比较高的问题，我也通过团队协作的方式进行了解决，在该过程中我也培养了团队合作的能力。

关于课程的建议，我认为可以进一步加强对顺序表的理解和应用能力培养，在实训中加入一些与实际应用相关联的训练，以便更好地理解有关算法。

3.2 基于链式存储结构的线性表实现

从数据结构实验——基于链式存储结构的线性表实现中，我取得了许多方面的收获，例如我深入理解了单链表的数据结构，包括节点、指针、头指针等概念，以及单链表的常用操作和函数，如创建单链表、添加节点、删除节点、遍历单链表等。此外，通过编写代码，我掌握了单链表的实现方法，提高了编程能力和数据结构实现的能力。在实验中，我了解了单链表在程序设计中的应用场景，如队列、栈等。这让我意识到基于链式存储结构的线性表作为一种基本的数据结构，在现实生活中有着广泛的应用。

在不断地编写和调试过程中，我遇到了各种问题，但相应的，我也提高了自己解决问题的能力，同时培养了良好的编程习惯，意识到需要注意编写规范、易读、易维护的代码，遵循命名规范、缩进、注释等规范。提高了代码质量和编程能力。

对于课程的建议，我认为可以增加一些更复杂的问题，以使更加深入地了解单链表的应用场景。

3.3 基于二叉链表的二叉树实现

从数据结构实验——基于二叉链表的二叉树实现中，可以获得很多收获，首先我深入理解了二叉树的数据结构，二叉树是一种非线性数据结构，它包含节点和分支，分支又可以分成左右子树。通过实验，可以深入理解二叉树的定义、特性和有关的算法操作，从而为后续的应用奠定基础。由于二叉树在计算机科学中有着广泛的应用，例如在文件系统管理、编译原理、图形学等领域都有重要的应用。通过课程，可以了解二叉树的应用场景，并探索如何在实际应用中使用二叉树解决问题。同时，在实验任务的完成过程中，我也培养了问题解决的能力和团队合作的能力。

关于课程的相关建议，我认为可以增加扩展有关树的内容，增加对于树的相关计算算法和有关应用的介绍和编写。

3.4 基于邻接表的图实现

从数据结构实验——图及其应用中，可以获得很多收获，首先，图是一种非线性数据结构，属于多对多的结构，它包含节点和边，节点和边之间可以有多条

链接。通过该实验课程，可以深入理解图的定义、特性和操作，从而为后续的应用奠定基础。此外，我还掌握了图的实现方法，包括图的节点定义和图的创建、深度优先遍历和广度优先遍历、搜索等操作，并且提高了编程能力和数据结构实现的能力。

图在计算机科学中有着广泛的应用，例如在社交网络分析、路由算法、图像处理等领域都有重要的应用。通过实验，我了解了图的应用场景，并探索如何在实际应用中使用图解决问题。并且在实验过程中，不断培养问题解决的能力和团队合作的能力。

关于课程的相关建议，我认为可以增加一些更复杂的应用题，比如 TSP 问题和最短路问题等，以便更加深入地了解图的应用场景。

参考文献

[1] 严蔚敏, 吴伟民. 数据结构 (C 语言版)[M]. 北京: 清华大学出版社, 2012.

[2] Larry Nyhoff. ADTs, Data Structures, and Problem Solving with C++. Second Edition,Calvin College,2005

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

[4] 严蔚敏等. 数据结构题集（C 语言版）. 清华大学出版社

附录 A 基于顺序存储结构线性表实现的源程序

```
/* Linear Table On Sequence Structure */
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>

/*———page 10 on textbook ———*/
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASTABLE -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; //线性表集合的定义 Lists

const int inf = 0x3f3f3f3f; //无穷大的定义

int whe=0;//作为是否进入多线性表模式的判断

int Listnum;

char Listname[30];


ElemType max(ElemType a,ElemType b)
{
if(a>=b) return a;
else return b;
}


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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(L.elem) return INFEASIBLE;
L.elem=(ElemType *)malloc(LIST_INIT_SIZE*sizeof(ElemType));
if(!L.elem) exit(OVERFLOW);
L.length=0;
```

```
L.listsize=LIST_INIT_SIZE;
return OK;
/********** End **********/
}


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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(L.elem){
free(L.elem);
L.elem=NULL;
L.length=0;
return OK;
}
else return INFEASIBLE;

/********** End **********/
}


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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L.elem) return INFEASIBLE;
```

```
else{
L.length=0;
return OK;
}
/********** End **********/
}


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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L.elem) return INFEASIBLE;
else{
if(L.length==0) return TRUE;
else return FALSE;
}

/********** End **********/
}


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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L.elem) return INFEASIBLE;
return L.length;
```

```
/********** End **********/
}


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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L.elem) return INFEASIBLE;
if(i>L.length||i<1) return ERROR;
e=L.elem[i-1];
return OK;

/********** End **********/
}


int LocateElem(SqList L,ElemType e)
/ 如果线性表 L 存在，查找元素 e 在线性表 L 中的位置序号并返回该序号；如果
e 不存在，返回 0；当线性表 L 不存在时，返回 INFEASIBLE（即-1）。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L.elem) return INFEASIBLE;
int i;
for(i=1;i<=L.length;i++){
if(e==L.elem[i-1]) return i;
}
```

```
return 0;

/********** End **********/
}


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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L.elem) return INFEASIBLE;
int i;
for(i=1;i<L.length;i++){
if(e==L.elem[i]){
pre=L.elem[i-1];
return OK;
}
}
return ERROR;
/********** End **********/
}


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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
```

```
if(!L.elem) return INFEASIBLE;
int i;
for(i=0;i<L.length-1;i++){
if(e==L.elem[i]){
next=L.elem[i+1];
return OK;
}
}
return ERROR;

/********** End **********/
}


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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int j;
if(!L.elem) return INFEASIBLE;
while(L.length+1>L.listsize){
L.elem=(ElemType*)realloc(L.elem,sizeof(L.elem)+LISTINCREMENT*sizeof(ElemType));
L.listsize=L.listsize+LISTINCREMENT;
}
//扩大 listsize
if(i<1||i>L.length+1) return ERROR;
for(j=L.length;j>=i;j–){
L.elem[j]=L.elem[j-1];
}
```

```
L.elem[i-1]=e;
L.length++;
return OK;
/********** End **********/
}


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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L.elem) return INFEASIBLE;
if(i<1||i>L.length) return ERROR;
e=L.elem[i-1];
int j;
for(j=i-1;j<L.length-1;j++){
L.elem[j]=L.elem[j+1];
}
L.length–;
return OK;
/********** End **********/
}


status ListTraverse(SqList L)
/ 如果线性表 L 存在，依次显示线性表中的元素，每个元素间空一格，返回 OK；如果线性表 L 不存在，返回 INFEASIBLE。
{
// 请在这里补充代码，完成本关任务
```

```
/********** Begin *********/
if(!L.elem) return INFEASIBLE;
int i=0;
if(L.length==0||L.length<0) return OK;
for(i=0;i<L.length-1;i++){
printf("%d ",L.elem[i]);
}
printf("%d",L.elem[i]);
return OK;
/********** End **********/
}
```

```
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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L.elem) return INFEASIBLE;
FILE* fp;
int i=0;
fp=fopen(FileName,"w+");
for(i=0;i<L.length;i++)
fprintf(fp,"%d ",L.elem[i]);
fclose(fp);
return OK;

/********** End **********/
}
```

```
status LoadList(SqList &L,char FileName[])
/ 如果线性表 L 不存在，将 FileName 文件中的数据读入到线性表 L 中，返回 OK，
否则返回 INFEASIBLE。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(L.elem) return INFEASIBLE;
L.elem=(ElemType *)malloc(LIST_INIT_SIZE*sizeof(ElemType));
L.length=0;
L.listsize=LIST_INIT_SIZE;
FILE* fp;
fp=fopen(FileName,”r+”);
int i=0,ch,bi,num=0;
while((ch=getc(fp))!=EOF){
//putchar(ch);
if(ch!=’ ’){
num=num*10+(ch-’0’);
}
if(ch==’ ’){
L.elem[i]=num;
i++;
num=0;
}
}
L.length=i;
fclose(fp);
return OK;
/********** End **********/
}
```

```
status AddList(LISTS &Lists,char ListName[])
/ 只需要在 Lists 中增加一个名称为 ListName 的空线性表，线性表数据又后台测试程序插入。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int j;
Lists.elem[Lists.length].L.elem=NULL;
j=InitList(Lists.elem[Lists.length].L);
if(j==OK){
//导入表名
strcpy(Lists.elem[Lists.length].name,ListName);
Lists.length++;
return OK;
}
return INFEASIBLE;
/********** End **********/
}


status RemoveList(LISTS &Lists,char ListName[])
/ Lists 中删除一个名称为 ListName 的线性表
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i,j;
for(i=0;i<Lists.length;i++){
if(strcmp(ListName,Lists.elem[i].name)==0){
for(j=i;j<Lists.length-1;j++){
Lists.elem[j]=Lists.elem[j+1];
```

```
}
Lists.length–;
return OK;
}
}
return INFEASIBLE;

/********** End **********/
}


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
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i,j;
for(i=0;i<Lists.length;i++){
if(strcmp(ListName,Lists.elem[i].name)==0){
return i+1;
}
}
return 0;

/********** End **********/
}


status MaxSubArray(SqList L)
{
```

```
//采用动态规划法求最大连续子数组和 O(n)
if(!L.elem) return INFEASIBLE;
if(L.length==0) return INFEASIBLE;
ElemType res=-inf,dp[L.length+1],num[L.length+1];
int i;
num[0]=0;
for(i=1;i<=L.length;i++){
num[i]=L.elem[i-1];
}
dp[0]=0;
for(i=1;i<=L.length;i++){
dp[i]=max(dp[i-1]+num[i],num[i]);
res=max(dp[i],res);
}
return res;
}


status SubArrayNum(SqList L,int k)
{
//采用暴力累加的方式求和为 k 的子数组个数 O(n^2)
if(!L.elem) return INFEASIBLE;
if(L.length==0) return INFEASIBLE;
ElemType res=0,sum;
int i,j;
for(i=0;i<L.length;i++){
sum=0;
for(j=i;j<L.length;j++){
sum=sum+L.elem[j];
if(sum==k){
res++;
```

```
}
}
}
return res;
}


status sortList(SqList &L)
{
//采用冒泡排序法
if(!L.elem) return INFEASIBLE;
int i,j;
ElemType temp;
for(i=0;i<L.length;i++){
for(j=i;j<L.length;j++){
if(L.elem[j]<L.elem[i]){
temp=L.elem[j];
L.elem[j]=L.elem[i];
L.elem[i]=temp;
}
}
}
return OK;
}


int main()
{
int choice;
ElemType e,pre,next;
SqList L;
```

```
L.elem=NULL;
LISTS Lists;
Lists.length=0;
while(1){
printf(
" 基于顺序存储结构的线性表实现"
"                                                                                "
" 1.InitList 10.ListInsert"
" 2.DestroyList 11.ListDelete"
" 3.ClearList 12.ListTraverse"
" 4.ListEmpty 13.SaveList & LoadList"
" 5.ListLength 14.MaxSubArray"
" 6.GetElem 15.SubArrayNum"
" 7.LocateElem 16.sortList"
" 8.PriorElem 17.Lists"
" 9.NextElem 18.exit"
" "
" "
"                                                                                "
n"
printf(" 请输入你所选择的操作 [1-18]: ");//菜单目录
if(whe==0){
printf(" 当前模式为单线性表模式");
}else{
printf(" 当前模式为多线性表模式");
}
scanf("%d",&choice);
switch(choice)//选择菜单
{
case 0: {
printf(" 无效的操作！输入的数字应在 1 18 之间，请重新输入!");
```

```
break;
}
case 1: {
int j=InitList(L);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=InitList(Lists.elem[Listnum-1].L);
//if(j==OK) printf(" 成功初始化线性表!");
//else printf(" 不能对已经存在的线性表初始化!");
}else{
printf(" 输入的名称有误或不存在!");break;
}
}
if (j==INFEASIBLE) printf(" 不能对已经存在的线性表初始化!");
if(j==OK) printf(" 成功初始化线性表!");
break;
}
case 2:{
int j=DestroyList(L);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=DestroyList(Lists.elem[Listnum-1].L);
}else{
printf(" 输入的名称有误或不存在!");
}
```

```
}
if(j==INFEASIBLE) printf(” 线性表不存在, 销毁失败!”);
if(j==OK) printf(” 已成功删除线性表!”);
break;
}
case 3:{
int j=ClearList(L);
if(whe==1){
printf(” 请输入想要操作的线性表的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ClearList(Lists.elem[Listnum-1].L);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(j==INFEASIBLE) printf(” 线性表不存在, 清空失败!”);
if(j==OK) printf(” 已成功清空线性表!”);
break;
}
case 4:{
int j=ListEmpty(L);
if(whe==1){
printf(” 请输入想要操作的线性表的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ListEmpty(Lists.elem[Listnum-1].L);
}else{
printf(” 输入的名称有误或不存在!”);
```

```
}
}
if(j==INFEASIBLE) printf(” 线性表不存在, 判空失败!”);
if(j==TRUE) printf(” 线性表为空!”);
if(j==FALSE) printf(” 线性表不为空!”);
break;
}
case 5:{
int j=ListLength(L);
if(whe==1){
printf(” 请输入想要操作的线性表的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ListLength(Lists.elem[Listnum-1].L);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(j==INFEASIBLE) printf(” 线性表不存在, 获取长度失败!”);
else printf(” 线性表的长度为%d”,j);
break;
}
case 6:{
int i;
printf(” 请输入想要获取第几个元素:”);
scanf(”%d”,&i);
int j=GetElem(L,i,e);
if(whe==1){
printf(” 请输入想要操作的线性表的名称:”);
scanf(”%s”,Listname);
```

```
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=GetElem(Lists.elem[Listnum-1].L,i,e);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 线性表不存在, 获取失败!");
if(j==OK) printf(" 成功获取元素%d 并存储在 e 中!",e);
if(j==ERROR) printf(" 输入的位置值不合法!");
break;
}
case 7:{
int e1;
printf(" 请输入想要查找的元素的数值:");
scanf("%d",&e1);
int j=LocateElem(L,e1);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=LocateElem(Lists.elem[Listnum-1].L,e1);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 线性表不存在, 查找失败!");
if(j>0) printf(" 所要查找的元素位置序号为%d!",j);
if(j==0) printf(" 所要查找的元素不存在!");
break;
```

```
}
case 8:{
int e1;
printf(" 请输入想要获取何数值元素的前驱:");
scanf("%d",&e1);
int j=PriorElem(L,e1,pre);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=PriorElem(Lists.elem[Listnum-1].L,e1,pre);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 线性表不存在, 获取失败!");
if(j==OK) printf(" 成功获取元素前驱%d 并存储在 pre 中!",pre);
if(j==ERROR) printf(" 该元素没有前驱!");
break;
}
case 9:{
int e1;
printf(" 请输入想要获取何数值元素的后继:");
scanf("%d",&e1);
int j=NextElem(L,e1,next);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
```

```
j=NextElem(Lists.elem[Listnum-1].L,e1,next);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 线性表不存在, 获取失败!");
if(j==OK) printf(" 成功获取元素后继%d 并存储在 next 中!",next);
if(j==ERROR) printf(" 该元素没有后继!");
break;
}
case 10:{
int i,e1;
printf(" 请输入在第几个元素之前插入和想要插入的元素的数值 (中间以空格分隔):");
scanf("%d %d",&i,&e1);
int j=ListInsert(L,i,e1);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ListInsert(Lists.elem[Listnum-1].L,i,e1);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 线性表不存在, 插入失败!");
if(j==OK) printf(" 成功插入!");
if(j==ERROR) printf(" 插入位置不正确!");
break;
}
```

```
case 11:{
int i;
printf(" 请输入删除第几个元素:");
scanf("%d",&i);
int j=ListDelete(L,i,e);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ListDelete(Lists.elem[Listnum-1].L,i,e);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 线性表不存在, 删除失败!");
if(j==OK) printf(" 成功删除并将元素值%d 保存在 e 中!",e);
if(j==ERROR) printf(" 删除位置不正确!");
break;
}
case 12:{
int j=ListTraverse(L);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ListTraverse(Lists.elem[Listnum-1].L);
}else{
printf(" 输入的名称有误或不存在!");
}
```

```
}
if(j==INFEASIBLE) printf(" 线性表不存在, 遍历失败!");
if(j==OK) printf(" 已成功遍历线性表!");
break;
}
case 13:{
char ListName[100];
int key;
file:
printf(" 请选择你的操作:1. 将线性表写入文件 2. 将文件中的数据读入线性表 3. 退出该功能");
scanf("%d",&key);
printf(" 请输入文件名");
fflush(stdin);
int file;
fflush(stdin);
scanf("%[ ^ ]",ListName);
switch(key)
{
case 0:{
printf(" 无效的操作！ 输入的数字应在 1 3 之间，请重新输入!");
goto file;
break;
}
case 1:{
int j=SaveList(L,ListName);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
```

```
j=SaveList(Lists.elem[Listnum-1].L,ListName);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 线性表不存在, 写入失败!");
if(j==OK) printf(" 写入成功!");
break;
}
case 2:{
int j=LoadList(L,ListName);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=LoadList(Lists.elem[Listnum-1].L,ListName);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 线性表已经存在, 读取失败!");
if(j==OK) printf(" 读取成功!");
break;
}
case 3:break;
default:{
printf(" 无效的操作！ 输入的数字应在 1 3 之间， 请重新输入!");
goto file;
break;
}
```

```
}
break;
}
case 14:{
int j=MaxSubArray(L);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=MaxSubArray(Lists.elem[Listnum-1].L);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if (j==INFEASIBLE) printf(" 线性表不存在或为空表，求最大连续子数组和失败!");
else printf(" 最大连续子数组和为%d!",j);
break;
}
case 15:{
int k;
printf(" 请输入想要获取的子数组的元素和:");
scanf("%d",&k);
int j=SubArrayNum(L,k);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=SubArrayNum(Lists.elem[Listnum-1].L,k);
```

```
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 线性表不存在或为空表, 获取失败!");
else printf(" 和为%d 的子数组个数为%d 个!",k,j);
break;
}
case 16:{
int j=sortList(L);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=sortList(Lists.elem[Listnum-1].L);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if (j==INFEASIBLE) printf(" 线性表不存在，排序失败!");
if(j==OK) printf(" 已成功从小到大排序线性表中元素!");
break;
}
case 17:{
printf(" 进入多线性表操作模式!");
whe=1;
printf(" 请选择你的操作:1. 多线性表添加 2. 多线性表删除 3. 多线性表查找 4. 退出多线性表模式");
int key;
scanf("%d",&key);
```

```
switch(key)
{
case 1:{
printf(" 请输入想要创建的线性表的名称:");
scanf("%s",Listname);
int j=AddList(Lists,Listname);
if(j==INFEASIBLE) printf(" 线性表不为空, 创建失败!");
if(j==OK) printf(" 创建成功!");
break;
}
case 2:{
printf(" 请输入想要删除的线性表的名称:");
scanf("%s",Listname);
int j=RemoveList(Lists,Listname);
if(j==INFEASIBLE) printf(" 线性表不存在, 删除失败!");
if(j==OK) printf(" 删除成功!");
break;
}
case 3:{
printf(" 请输入想要查找的线性表的名称:");
scanf("%s",Listname);
int j=LocateList(Lists,Listname);
if(j==0) printf(" 不存在该名称的线性表!");
else printf(" 该线性表的序号为%d!",j);
break;
}
case 4:{
whe=0;
break;
}
default:{
```

```
printf(" 无效的操作！输入的数字应在 1-4 之间，请重新输入!");
break;
}
}
break;
}
case 18:{
printf(" 已成功退出程序, 欢迎下次再使用本系统!");
return 0;
}
default:{
printf(" 无效的操作！输入的数字应在 1-18 之间，请重新输入!");
break;
}
}
fflush(stdin);
system("pause");
system("cls");
}
}
```

附录 B 基于链式存储结构线性表实现的源程序

```
#include <string.h>
#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 INFEASTABLE -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;
LISTS Lists;

const int inf = 0x3f3f3f3f; //无穷大的定义
int whe=0;//作为是否进入多线性表模式的判断
int Listnum;
char Listname[30];
int a[100];


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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(L) return INFEASIBLE;
L=(LinkList)malloc(sizeof(LNode));
L->next=NULL;
return OK;

/********** End **********/
}

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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(L){
LinkList current=L;
```

```
while (current!=NULL) {
LinkList next=current->next;
free(current);
current=next;
}
L=NULL;
return OK;
}
else return INFEASIBLE;


/********** End **********/
}


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 INFEA-
SIBLE。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(L){
LinkList current=L->next;
while (current!=NULL) {
LinkList next=current->next;
free(current);
current=next;
}
L->next=NULL;
return OK;
}
```

```
else return INFEASIBLE;

/********** End **********/
}

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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L) return INFEASIBLE;
if(L->next==NULL) return TRUE;
else return FALSE;
/********** End **********/
}

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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L) return INFEASIBLE;
int length=0;
while(L->next!=NULL){
L=L->next;
length++;
}
return length;
/********** End **********/
}
```

```
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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L) return INFEASIBLE;
int length=0,j;
LinkList p=L;
while(p->next!=NULL){
p=p->next;
length++;
}
if(i<1||i>length) return ERROR;
p=L;
for(j=1;j<=i;j++){
p=p->next;
}
e=p->data;
return OK;
/********** End **********/
}

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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L) return INFEASIBLE;
```

```
int num=1;
LinkList p=L->next;
while(p!=NULL){
if(p->data==e){
return num;
}
p=p->next;
num++;
}
return ERROR;
/********** End **********/
}

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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L) return INFEASIBLE;
if(L->next==NULL) return ERROR;
LinkList p=L->next,q=L;
if(p->data==e) return ERROR;
while(p!=NULL){
if(p->data==e){
pre=q->data;
return OK;
}
p=p->next;
q=q->next;
}
```

```
return ERROR;
/********** End **********/
}

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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L) return INFEASIBLE;
if(L->next==NULL) return ERROR;//空表的特判
LinkList p=L->next;
while(p->next!=NULL){
if(p->data==e){
next=p->next->data;
return OK;
}
p=p->next;
}
return ERROR;

/********** End **********/
}

status ListInsert(LinkList &L,int i,ElemType e)
// 如果线性表 L 存在，将元素 e 插入到线性表 L 的第 i 个元素之前，返回 OK；当插入位置不正确时，返回 ERROR；如果线性表 L 不存在，返回 INFEASIBLE。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
```

```
if(!L) return INFEASIBLE;
int length=0;
LinkList p=L;
while(p->next!=NULL){
p=p->next;
length++;
}
if(i<1||i>length+1) return ERROR;
int j;
p=L;
for(j=2;j<=i;j++){
p=p->next;
}
//插入新节点
LinkList temp;
temp=(LinkList)malloc(sizeof(LNode));
temp->data=e;
temp->next=p->next;
p->next=temp;
return OK;

/********** End **********/
}

status ListDelete(LinkList &L,int i,ElemType &e)
// 如果线性表 L 存在，删除线性表 L 的第 i 个元素，并保存在 e 中，返回 OK；当删除位置不正确时，返回 ERROR；如果线性表 L 不存在，返回 INFEASIBLE。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L) return INFEASIBLE;
```

```
int length=0;
LinkList p=L,q=L;
while(p->next!=NULL){
p=p->next;
length++;
}
if(i<1||i>length) return ERROR;
int j;
p=L->next;
for(j=2;j<=i;j++){
p=p->next;
q=q->next;
}
//删除节点
q->next=p->next;
e=p->data;
free(p);
return OK;

/********** End **********/
}

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。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(!L) return INFEASIBLE;
if(L->next==NULL) return OK;
while(L->next->next!=NULL){
```

```
L=L->next;
printf("%d ",L->data);
}
L=L->next;
printf("%d",L->data);
return OK;

/********** End **********/
}

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。
{
// 请在这里补充代码，完成本关任务
/********** Begin 1 *********/
if(!L) return INFEASIBLE;
FILE* fp;
fp=fopen(FileName,"w+");
LinkList p=L->next;
while(p!=NULL){
fprintf(fp,"%d ",p->data);
p=p->next;
}
fclose(fp);
return OK;

/********** End 1 **********/
}

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

```
// 如果线性表 L 不存在，将 FileName 文件中的数据读入到线性表 L 中，返回 OK，
否则返回 INFEASIBLE。
{
// 请在这里补充代码，完成本关任务
/********** Begin 2 *********/
if(L) return INFEASIBLE;
int data;
L=(LinkList)malloc(sizeof(LNode));
LinkList p=L;
FILE* fp;
fp=fopen(FileName,”r+”);
LinkList temp;
while((fscanf(fp,”%d”,&data))!=EOF){
temp=(LinkList)malloc(sizeof(LNode));
p->next=temp;
temp->data=data;
p=p->next;
}
p->next=NULL;
fclose(fp);
return OK;

/********** End 2 **********/
}

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
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
```

```
int i,j;
for(i=0;i<Lists.length;i++){
if(strcmp(ListName,Lists.elem[i].name)==0){
return i+1;
}
}
return 0;

/********** End **********/
}

int reverseList(LinkList &L)
{//使用类似于头插法的方法来翻转链表
if(!L) return INFEASIBLE;//线性表不存在则返回 INFEASIBLE
LinkList p=L->next;
L->next=NULL;
LinkList q;
while(p!=NULL){
q=p;//用 q 记录 p 的位置
p=p->next;//p 寻找下一个元素
q->next=L->next;
L->next=q;
}
return OK;
}

int RemoveNthFromEnd(LinkList &L,int n)
{//删除倒数第 n 个节点
if(!L) return INFEASIBLE;//线性表不存在则返回 INFEASIBLE
int count=0,i;//用 count 来获取链表的长度
LinkList p=L->next;
```

```
LinkList q=L->next;//采用双指针
while(p!=NULL){
count++;
p=p->next;
}
p=L;
if(n>count||n<1) return ERROR;
for(i=1;i<=count-n;i++){
p=p->next;
q=q->next;//p 指到待删除节点的前一位置,q 指到待删除节点
}
p->next=q->next;
q->next=NULL;
free(q);
return OK;
}

int sortList(LinkList &L)
{//将线性表由小到大排序
if(!L) return INFEASIBLE;//线性表不存在则返回 INFEASIBLE
int i=0,lena=0;
LinkList p=L->next;
while(p!=NULL){
a[i]=p->data;
p=p->next;
lena++;
i++;
}
//采用冒泡排序法 O(n^2)
int j;
ElemType temp;
```

```
for(i=0;i<lena;i++){
for(j=i;j<lena;j++){
if(a[j]<a[i]){
temp=a[j];
a[j]=a[i];
a[i]=temp;
}
}
}
p=L->next;
i=0;
while(p!=NULL){
p->data=a[i];
p=p->next;
i++;
}
return OK;
}

status AddList(LISTS &Lists,char ListName[])
// 只需要在 Lists 中增加一个名称为 ListName 的空线性表，线性表数据又后台测试程序插入。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int j;
Lists.elem[Lists.length].L=NULL;
j=InitList(Lists.elem[Lists.length].L);
if(j==OK){
//导入表名
strcpy(Lists.elem[Lists.length].name,ListName);
```

```
Lists.length++;
return OK;
}
return INFEASIBLE;
/********** End **********/
}

status RemoveList(LISTS &Lists,char ListName[])
// Lists 中删除一个名称为 ListName 的线性表
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i,j;
for(i=0;i<Lists.length;i++){
if(strcmp(ListName,Lists.elem[i].name)==0){
for(j=i;j<Lists.length-1;j++){
Lists.elem[j]=Lists.elem[j+1];
}
Lists.length–;
return OK;
}
}
return INFEASIBLE;

/********** End **********/
}


int main()
{
int choice;
```

```
ElemType e,pre,next;
LinkList L;
L=NULL;
while(1){
printf(
" 基于链式存储结构的线性表实现"
"                                                                                "
" 1.InitList 10.ListInsert"
" 2.DestroyList 11.ListDelete"
" 3.ClearList 12.ListTraverse"
" 4.ListEmpty 13.SaveList & LoadList"
" 5.ListLength 14.reverseList"
" 6.GetElem 15.RemoveNthFromEnd"
" 7.LocateElem 16.sortList"
" 8.PriorElem 17.Lists"
" 9.NextElem 18.exit"
" "
" "
"                                                                                "
" 请输入你所选择的操作 [1-18]: ");
if(whe==0){
printf(" 当前模式为单线性表模式");
}else{
printf(" 当前模式为多线性表模式");
}
scanf("%d",&choice);
switch(choice)//选择菜单
{
case 0: {
printf(" 无效的操作！输入的数字应在 1 18 之间，请重新输入!");
break;
```

```
}
case 1: {
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=InitList(Lists.elem[Listnum-1].L);
//if(j==OK) printf(" 成功初始化线性表!");
//else printf(" 不能对已经存在的线性表初始化!");
}else{
printf(" 输入的名称有误或不存在!");break;
}
}
if(whe==0){
j=InitList(L);
}
if(j==INFEASIBLE) printf(" 不能对已经存在的线性表初始化!");
if(j==OK) printf(" 成功初始化线性表!");
break;
}
case 2:{
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=DestroyList(Lists.elem[Listnum-1].L);
}else{
```

```
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=DestroyList(L);
}
if(j==INFEASIBLE) printf(" 线性表不存在, 销毁失败!");
if(j==OK) printf(" 已成功删除线性表!");
break;
}
case 3:{
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ClearList(Lists.elem[Listnum-1].L);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=ClearList(L);
}
if(j==INFEASIBLE) printf(" 线性表不存在, 清空失败!");
if(j==OK) printf(" 已成功清空线性表!");
break;
}
case 4:{
int j;
```

```
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ListEmpty(Lists.elem[Listnum-1].L);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=ListEmpty(L);
}
if(j==INFEASIBLE) printf(" 线性表不存在, 判空失败!");
if(j==TRUE) printf(" 线性表为空!");
if(j==FALSE) printf(" 线性表不为空!");
break;
}
case 5:{
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ListLength(Lists.elem[Listnum-1].L);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
```

```
j=ListLength(L);
}
if(j==INFEASIBLE) printf(" 线性表不存在, 获取长度失败!");
else printf(" 线性表的长度为%d",j);
break;
}
case 6:{
int i;
printf(" 请输入想要获取第几个元素:");
scanf("%d",&i);
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=GetElem(Lists.elem[Listnum-1].L,i,e);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=GetElem(L,i,e);
}
if(j==INFEASIBLE) printf(" 线性表不存在, 获取失败!");
if(j==OK) printf(" 成功获取元素%d 并存储在 e 中!",e);
if(j==ERROR) printf(" 输入的位置值不合法!");
break;
}
case 7:{
int e1;
```

```
printf(" 请输入想要查找的元素的数值:");
scanf("%d",&e1);
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=LocateElem(Lists.elem[Listnum-1].L,e1);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=LocateElem(L,e1);
}
if(j==INFEASIBLE) printf(" 线性表不存在, 查找失败!");
if(j>0) printf(" 所要查找的元素位置序号为%d!",j);
if(j==0) printf(" 所要查找的元素不存在!");
break;
}
case 8:{
int e1;
printf(" 请输入想要获取何数值元素的前驱:");
scanf("%d",&e1);
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
```

```
j=PriorElem(Lists.elem[Listnum-1].L,e1,pre);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=PriorElem(L,e1,pre);
}
if(j==INFEASIBLE) printf(" 线性表不存在, 获取失败!");
if(j==OK) printf(" 成功获取元素前驱%d 并存储在 pre 中!",pre);
if(j==ERROR) printf(" 该元素没有前驱或该元素不存在!");
break;
}
case 9:{
int e1;
printf(" 请输入想要获取何数值元素的后继:");
scanf("%d",&e1);
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=NextElem(Lists.elem[Listnum-1].L,e1,next);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=NextElem(L,e1,next);
}
```

```
if(j==INFEASIBLE) printf(" 线性表不存在, 获取失败!");
if(j==OK) printf(" 成功获取元素后继%d 并存储在 next 中!",next);
if(j==ERROR) printf(" 该元素没有后继或该元素不存在!");
break;
}
case 10:{
int i,e1;
printf(" 请输入在第几个元素之前插入和想要插入的元素的数值 (中间以空格分隔):");
scanf("%d %d",&i,&e1);
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ListInsert(Lists.elem[Listnum-1].L,i,e1);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=ListInsert(L,i,e1);
}
if(j==INFEASIBLE) printf(" 线性表不存在, 插入失败!");
if(j==OK) printf(" 成功插入!");
if(j==ERROR) printf(" 插入位置不正确!");
break;
}
case 11:{
int i;
```

```
printf(" 请输入删除第几个元素:");
scanf("%d",&i);
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ListDelete(Lists.elem[Listnum-1].L,i,e);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=ListDelete(L,i,e);
}
if(j==INFEASIBLE) printf(" 线性表不存在, 删除失败!");
if(j==OK) printf(" 成功删除并将元素值%d 保存在 e 中!",e);
if(j==ERROR) printf(" 删除位置不正确!");
break;
}
case 12:{
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ListTraverse(Lists.elem[Listnum-1].L);
}else{
printf(" 输入的名称有误或不存在!");
```

```
}
}
if(whe==0){
j=ListTraverse(L);
}
if(j==INFEASIBLE) printf(" 线性表不存在, 遍历失败!");
if(j==OK) printf(" 已成功遍历线性表!");
break;
}
case 13:{
char ListName[100];
int key;
file:
printf(" 请选择你的操作:1. 将线性表写入文件 2. 将文件中的数据读入线性表 3. 退出该功能");
scanf("%d",&key);
printf(" 请输入文件名");
fflush(stdin);
scanf("%[^]",ListName);
switch(key)
{
case 0:{
printf(" 无效的操作！输入的数字应在 1-3 之间，请重新输入!");
goto file;
break;
}
case 1:{
int j=SaveList(L,ListName);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
```

```
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=SaveList(Lists.elem[Listnum-1].L,ListName);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 线性表不存在, 写入失败!");
if(j==OK) printf(" 写入成功!");
break;
}
case 2:{
int j=LoadList(L,ListName);
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=LoadList(Lists.elem[Listnum-1].L,ListName);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 线性表已经存在, 读取失败!");
if(j==OK) printf(" 读取成功!");
break;
}
case 3:break;
default:{
printf(" 无效的操作！ 输入的数字应在 1-3 之间，请重新输入!");
goto file;
```

```
break;
}
}
break;
}
case 14:{
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=reverseList(Lists.elem[Listnum-1].L);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=reverseList(L);
}
if (j==INFEASIBLE) printf(" 线性表不存在或为空表, 无法翻转!");
else printf(" 链表已成功翻转!",j);
break;
}
case 15:{
int k;
printf(" 请输入想要删除倒数第几个节点:");
scanf("%d",&k);
int j;
if(whe==1){
printf(" 请输入想要操作的线性表的名称:");
```

```
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=RemoveNthFromEnd(Lists.elem[Listnum-1].L,k);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(whe==0){
j=RemoveNthFromEnd(L,k);
}
if(j==INFEASIBLE) printf(” 线性表不存在或为空表, 删除失败!”);
if(j==ERROR) printf(”n 的值非法!”);
if(j==OK) printf(” 已成功删除!”,k,j);
break;
}
case 16:{
int j;
if(whe==1){
printf(” 请输入想要操作的线性表的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=sortList(Lists.elem[Listnum-1].L);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(whe==0){
j=sortList(L);
}
```

```
if (j==INFEASIBLE) printf(” 线性表不存在，排序失败!”);
if(j==OK) printf(” 已成功从小到大排序线性表中元素!”);
break;
}
case 17:{
printf(” 进入多线性表操作模式!”);
whe=1;
printf(” 请选择你的操作:1. 多线性表添加 2. 多线性表删除 3. 多线性表查找 4. 退出多线性表模式”);
int key;
scanf(”%d”,&key);
switch(key)
{
case 1:{
printf(” 请输入想要创建的线性表的名称:”);
scanf(”%s”,Listname);
int j=AddList(Lists,Listname);
if(j==INFEASIBLE) printf(” 线性表不为空, 创建失败!”);
if(j==OK) printf(” 创建成功!”);
break;
}
case 2:{
printf(” 请输入想要删除的线性表的名称:”);
scanf(”%s”,Listname);
int j=RemoveList(Lists,Listname);
if(j==INFEASIBLE) printf(” 线性表不存在, 删除失败!”);
if(j==OK) printf(” 删除成功!”);
break;
}
case 3:{
printf(” 请输入想要查找的线性表的名称:”);
```

```
scanf(”%s”,Listname);
int j=LocateList(Lists,Listname);
if(j==0) printf(” 不存在该名称的线性表!”);
else printf(” 该线性表的序号为%d!”,j);
break;
}
case 4:{
whe=0;
break;
}
default:{
printf(” 无效的操作！输入的数字应在 1-4 之间，请重新输入!”);
break;
}
}
break;
}
case 18:{
printf(” 已成功退出程序, 欢迎下次再使用本系统!”);
return 0;
}
default:{
printf(” 无效的操作！输入的数字应在 1-18 之间，请重新输入!”);
break;
}
}
fflush(stdin);
system(”pause”);
system(”cls”);
}
}
```

附录 C 基于二叉链表二叉树实现的源程序

```
#include <string.h>
#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 INFEASTABLE -1
#define OVERFLOW -2

typedef int ElemType;
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 QueueNode//队列节点
{
BiTree data;
```

```
QueueNode* next;
}QueueNode, * pQueueNode;

typedef struct Queue//队列
{
pQueueNode front,rear;
}Queue, * pQueue;

typedef struct{ //线性表的集合类型定义
struct { char name[30];
BiTree T;
}elem[10];
int length;
}LISTS;
LISTS Lists;//森林体系

int creatcnt=-1;//1 a 0 null 0 null -1 null
int judgecreat=0;
int judgedestroy=0;
int whe=0;
int Listnum;
char Listname[30];
TElemType definition[100];
int num=0;
int ndepth=0;

status CreateBiTree(BiTree &T,TElemType definition[])
/* 根据带空枝的二叉树先根遍历序列 definition 构造一棵二叉树，将根节点指针
赋值给 T 并返回 OK，
如果有相同的关键字，返回 ERROR。此题允许通过增加其它函数辅助实现本关
任务 */
```

```
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
//判断是否有重复的序号
if(T!=NULL) return INFEASIBLE;
int i,j,len;
if(judgecreat==0){
for(len=0;definition[len].key!=-1;len++);
//printf("%d",len);
for(i=0;i<len-1;i++){
for(j=i+1;j<len;j++){
if(definition[i].key!=0&&definition[i].key==definition[j].key){
return ERROR;
}
}
}
judgecreat=1;
}

creatcnt++;
int data0;
data0=definition[creatcnt].key;
if(data0==0){
T=NULL;
}
if(data0!=0&&data0!=-1){
T=(BiTree)malloc(sizeof(BiTNode));
T->lchild=NULL;
T->rchild=NULL;
T->data.key = data0;
strcpy(T->data.others,definition[creatcnt].others);//strcpy
```

```
CreateBiTree(T->lchild,definition);
CreateBiTree(T->rchild,definition);
}

return OK;
/********** End **********/
}

status DestroyBiTree(BiTree &T)
{
if(T){
judgedestroy=1;
DestroyBiTree(T->lchild);
DestroyBiTree(T->rchild);
free(T);
T=NULL;
}
if(judgedestroy==0) return INFEASIBLE;
else 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
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(T){
ClearBiTree(T->lchild);
ClearBiTree(T->rchild);
free(T);
T=NULL;
```

```
}
return OK;
/********** End **********/
}

status BiTreeEmpty(BiTree T)
{
if (T == NULL) return TRUE;
else return FALSE;
}

int BiTreeDepth(BiTree T)
//求二叉树 T 的深度
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int m, n;
if (T == NULL)
return 0; //如果是空树，深度为 0，递归结束
else
{
m = BiTreeDepth(T->lchild); //递归计算左子树的深度记为 m
n = BiTreeDepth(T->rchild); //递归计算右子树的深度记为 n
if (m > n)
return (m + 1); //二叉树的深度为 m 与 n 的较大者加 1
else
return (n + 1);
}

/********** End **********/
}
```

```
BiTNode* LocateNode(BiTree T,KeyType e)
//查找结点
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
//优先考虑左边, 然后再考虑右边
if (T == NULL) {
return NULL;
}
if (T->data.key == e) {
return T;
}
BiTree node = LocateNode(T->lchild, e);
if (node != NULL) {
// 左子树中找到了
return node;
}
node = LocateNode(T->rchild, e);
if (node != NULL) {
return node;
}
else {
return NULL;
}
/********** End **********/
}

status Assign(BiTree &T,KeyType e,TElemType value)
//实现结点赋值。此题允许通过增加其它函数辅助实现本关任务
{
```

```
// 请在这里补充代码，完成本关任务
/********** Begin *********/
BiTNode *p=LocateNode(T,e);
if(p==NULL){
return ERROR;
}
BiTNode *q=LocateNode(T,value.key);
if(q!=NULL&&e!=value.key){//需要注意保持二叉树中关键字的唯一性
return ERROR;
}
p->data.key=value.key;
int i;
for(i=0;i<20;i++){
p->data.others[i]=value.others[i];
}
return OK;
/********** End **********/
}

BiTNode* GetParent(BiTree T,KeyType e)
//获取父亲结点
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
//优先考虑左边, 然后再考虑右边
if (T == NULL ) {
return NULL;
}
if(T->lchild!=NULL){
if (T->lchild->data.key == e){
return T;
```

```
}
}
if(T->rchild!=NULL){
if (T->rchild->data.key == e){
return T;
}
}
BiTree node = GetParent(T->lchild, e);
if (node != NULL) {
// 左子树中找到了
return node;
}
node = GetParent(T->rchild, e);
if (node != NULL) {
return node;
}
else {
return NULL;
}
/********** End **********/
}

BiTNode* GetSibling(BiTree T,KeyType e)
//实现获得兄弟结点
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
BiTNode* p=GetParent(T,e);
if(p==NULL) return ERROR;
if(p->lchild==NULL||p->rchild==NULL) return ERROR;
if(p->lchild->data.key==e) return p->rchild;
```

```
if(p->rchild->data.key==e) return p->lchild;
else return ERROR;
/********** End **********/
}

status InsertNode(BiTree &T,KeyType e,int LR,TElemType c)
//插入结点。此题允许通过增加其它函数辅助实现本关任务
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
//LR=-1 时的特判
if(LR==-1){
BiTree q=(BiTree)malloc(sizeof(BiTNode));
q->lchild=NULL;
q->rchild=T;
T=q;
T->data.key=c.key;
strcpy(T->data.others,c.others);//strcpy
return OK;
}
//否则则开始查找结点
BiTree p=LocateNode(T,e);
if(p==NULL) return ERROR;//没有找到的情况
BiTree q=LocateNode(T,c.key);
if(q!=NULL) return ERROR;//关键字有重复的情况
if(LR==0){
BiTree m=(BiTree)malloc(sizeof(BiTNode));
m->data.key=c.key;
strcpy(m->data.others,c.others);//复制数据
BiTree n=p->lchild;
p->lchild=m;
```

```
m->lchild=NULL;
m->rchild=n;
return OK;
}
if(LR==1){
BiTree m=(BiTree)malloc(sizeof(BiTNode));
m->data.key=c.key;
strcpy(m->data.others,c.others);//复制数据
BiTree n=p->rchild;
p->rchild=m;
m->rchild=n;
m->lchild=NULL;
return OK;
}
return ERROR;
/********** End **********/
}

status DeleteNode(BiTree &T,KeyType e)
//删除结点。此题允许通过增加其它函数辅助实现本关任务
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
BiTree p=LocateNode(T,e);
if(p==NULL) return ERROR;//没有找到的情况
if(p->lchild==NULL&&p->rchild==NULL){
//度为 0 的情况
BiTree q=GetParent(T,e);
if(q==NULL){
free(T);
T=NULL;
```

```
return OK;
}
if(q!=NULL){
if(q->lchild->data.key==p->data.key){
free(p);
q->lchild=NULL;
}
else if(q->rchild->data.key==p->data.key){
free(p);
q->rchild=NULL;//判断 p 是 q 的左右哪个节点
}
return OK;
}
}
if(p->lchild!=NULL&&p->rchild==NULL){
//p 左节点不为空
BiTree q=GetParent(T,e);
if(q==NULL){//p 为根节点
T=p->lchild;
free(p);
return OK;
}
if(q!=NULL){
if(q->lchild->data.key==p->data.key){
q->lchild=p->lchild;
free(p);//p 是 q 的左节点
}
else if(q->rchild->data.key==p->data.key){
q->rchild=p->lchild;
free(p);//p 是 q 的右节点
}
```

```
return OK;
}
}
if(p->rchild!=NULL&&p->lchild==NULL){
//p 右节点不为空
BiTree q=GetParent(T,e);
if(q==NULL){//p 为根节点
T=p->rchild;
free(p);
return OK;
}
if(q!=NULL){
if(q->lchild->data.key==p->data.key){
q->lchild=p->rchild;
free(p);//p 是 q 的左节点
}
else if(q->rchild->data.key==p->data.key){
q->rchild=p->rchild;
free(p);//p 是 q 的右节点
}
return OK;
}
}
if(p->rchild!=NULL&&p->lchild!=NULL){
//度为 2 的情况
BiTree m=p->lchild;//用 m 去指向左子树的最右节点
BiTree q=GetParent(T,e);
if(q==NULL){//p 为根节点
while(m->rchild!=NULL){
m=m->rchild;
}
```

```
T=p->lchild;
m->rchild=p->rchild;
free(p);
return OK;
}
}
return ERROR;
/********** End **********/
}

void visit(BiTree T)
{//visit 访问二叉树每个结点的数据
printf(" %d,%s",T->data.key,T->data.others);
}

status PreOrderTraverse(BiTree T,void (*visit)(BiTree))
//先序遍历二叉树 T
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(T==NULL){//判出条件
return OK;
}
visit(T);
PreOrderTraverse(T->lchild,visit);
PreOrderTraverse(T->rchild,visit);
return OK;
/********** End **********/
}

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

```
//中序遍历二叉树 T
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(T==NULL){//判出条件
return OK;
}
InOrderTraverse(T->lchild,visit);
visit(T);
InOrderTraverse(T->rchild,visit);
return OK;
/********** End **********/
}

status PostOrderTraverse(BiTree T,void (*visit)(BiTree))
//后序遍历二叉树 T
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(T==NULL){//判出条件
return OK;
}
PostOrderTraverse(T->lchild,visit);
PostOrderTraverse(T->rchild,visit);
visit(T);
return OK;
/********** End **********/
}

pQueue initQ(pQueue pq)//建立只有头结点的队列
{
```

```
pq->front = (pQueueNode)malloc(sizeof(QueueNode));
pq->front->next = NULL;//队列的 front 和 rear 的 next 初始化为空
pq->rear = pq->front;
return pq;
}

void enqueue(pQueue pq, BiTree t)//把二叉树的数据取出放入队列
{
pQueueNode pNew = new QueueNode;
pNew->data = t;//二叉树的数据存入队列
pNew->next = NULL;
pq->rear->next = pNew;//尾插法建立连接
pq->rear = pNew;//rear 更新
}

BiTree dequeue(pQueue pq)//出队：删除队列第一个元素
{
pQueueNode pTemp= (pQueueNode)malloc(sizeof(QueueNode));
pTemp = pq->front->next;
if (pTemp->next == NULL)//只剩下 1 个节点（不含队列空的头结点）
{
pq->rear = pq->front;
}
else{
pq->front->next = pTemp->next;//front+1（从指向第 1 个非空节点改为指向第 2 个节点）
}
BiTree x;
x= pTemp->data;//x 为队列第一个元素的 data
free(pTemp);
return x;
```

```
}

status LevelOrderTraverse(BiTree T,void (*visit)(BiTree))
//按层遍历二叉树 T
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
//用队列实现非递归的层序遍历
pQueue pq= (pQueue)malloc(sizeof(Queue));
pq = initQ(pq);
enqueue(pq,T);//取出二叉树的根节点，子节点存入队列
while (pq->rear != pq->front)//当队列不为空
{
BiTree x = dequeue(pq);//x 用于输出队列弹出元素的数据
visit(x);
if (x->lchild!=NULL)
{
enqueue(pq, x->lchild);//递归左节点
}
if (x->rchild!=NULL)
{
enqueue(pq, x->rchild);//递归右节点
}
}
return OK;
/********** End **********/
}

int LocateList(LISTS Lists,char ListName[])
// 在 Lists 中查找一个名称为 ListName 的二叉树，成功返回逻辑序号，否则返回
0
```

```
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i,j;
for(i=0;i<Lists.length;i++){
if(strcmp(ListName,Lists.elem[i].name)==0){
return i+1;
}
}
return 0;

/********** End **********/
}

status AddList(LISTS &Lists,char ListName[])
// 只需要在 Lists 中增加一个名称为 ListName 的空线性表，线性表数据又后台测试程序插入。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
Lists.elem[Lists.length].T=NULL;
int j,i=0;
creatcnt=-1;
judgecreat=0;//做必要的初始化
printf(" 请输入数据 (以-1 null 结束):");
do {
scanf("%d%s",&definition[i].key,definition[i].others);
} while (definition[i++].key!=-1);
j=CreateBiTree(Lists.elem[Lists.length].T,definition);
if(j==OK){
//导入表名
```

```
strcpy(Lists.elem[Lists.length].name,ListName);
Lists.length++;
return OK;
}
return INFEASIBLE;
/********** End **********/
}

status RemoveList(LISTS &Lists,char ListName[])
// Lists 中删除一个名称为 ListName 的线性表
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i,j;
for(i=0;i<Lists.length;i++){
if(strcmp(ListName,Lists.elem[i].name)==0){
for(j=i;j<Lists.length-1;j++){
Lists.elem[j]=Lists.elem[j+1];//覆盖删除
}
Lists.length–;
return OK;
}
}
return INFEASIBLE;

/********** End **********/
}

status SaveBiTree(BiTree T, char FileName[])
//将二叉树的结点数据写入到文件 FileName 中
{
```

```
// 请在这里补充代码，完成本关任务
/********** Begin 1 *********/
static FILE* fp=fopen(FileName,”w+”);
static int idx = 0;
int i=idx++;
if(T==NULL){//判出条件
fprintf(fp,”0 ”);//空节点用 0 代替
//printf(”0 ”);
return OK;
}
fprintf(fp,”%d ”,T->data.key);
//printf(”%d ”,T->data.key);
fprintf(fp,”%s ”,T->data.others);
//printf(”%s ”,T->data.others);
SaveBiTree(T->lchild,FileName);
SaveBiTree(T->rchild,FileName);
if(i==0) fclose(fp);
return OK;//通过递归的方式输入先序
/********** End 1 **********/
}

status LoadBiTree(BiTree &T, char FileName[])
//读入文件 FileName 的结点数据，创建二叉树
{
// 请在这里补充代码，完成本关任务
/********** Begin 2 *********/
static FILE* fp=fopen(FileName,”r+”);
static int idx = 0;
int i=idx++;
int num;
fscanf(fp,”%d”,&num);
```

```
if(num==0){
T=NULL;
}else{
T=(BiTree)malloc(sizeof(BiTNode));
T->data.key=num; //给新节点数据域赋值, 生成根节点
fscanf(fp,"%s",T->data.others);
LoadBiTree(T->lchild,FileName); //构造左子树过程
LoadBiTree(T->rchild,FileName); //构造右子树过程
}
if(i==0) fclose(fp);
return OK;
/********** End 2 **********/
}

int MaxPathSum(BiTree T)//测试数据 1 a 0 null 3 c 6 f 0 null 4 d 0 null 0 null 11
x 0 null 0 null -1 null 输出 15
{//采用递归的方式来计算二叉树根节点到叶子结点的最大路径
int lenleft,lenright;
if(T==NULL) return 0;//空节点为判出条件
int num;
num=T->data.key;
lenleft=MaxPathSum(T->lchild);
lenright=MaxPathSum(T->rchild);
if(lenleft>=lenright) return (num+lenleft);//左大取左，右大取右
else return (num+lenright);
}

BiTree InvertTree(BiTree &T)//测试数据同上, 前序输出 1 a 3 c 11 x 6 f 4 d
{//采用递归的方式来翻转二叉树
if(T==NULL) return NULL;//判出条件
BiTree tmp=T->lchild;
```

```
T->lchild=InvertTree(T->rchild);
T->rchild=InvertTree(tmp);
return T;
}

int DepthAssitant(BiTree T,int currentdepth,KeyType e)
{//求指定结点深度的辅助函数
if(T==NULL) return 0;//判出条件
currentdepth++;
if(T->data.key==e){
ndepth=currentdepth;
return currentdepth;
}
if(T->lchild!=NULL) currentdepth=DepthAssitant(T->lchild,currentdepth,e);
if(T->rchild!=NULL) currentdepth=DepthAssitant(T->rchild,currentdepth,e);
return (currentdepth-1);
}

int NodeDepth(BiTree T,KeyType e)
{//求指定结点的深度
int depth=0;
ndepth=0;
depth=DepthAssitant(T,depth,e);
depth=ndepth;
return depth;
}

int LowestCommonAncestor(BiTree T,KeyType e1,KeyType e2)//测试数据同上, 输入 11 4，输出 3
{//寻找某两个节点的最近公共祖先
BiTree tmp;
```

```
if(NodeDepth(T,e1)==0||NodeDepth(T,e2)==0) return ERROR;//没有找到 e1 e2 的情
况
if(NodeDepth(T,e1)==1) return e1;
if(NodeDepth(T,e2)==1) return e2;//其中某个结点为根节点的情况
while(NodeDepth(T,e1)!=NodeDepth(T,e2)){
if(NodeDepth(T,e1)>NodeDepth(T,e2)){
tmp=GetParent(T,e1);
e1=tmp->data.key;
}
if(NodeDepth(T,e1)<NodeDepth(T,e2)){
tmp=GetParent(T,e2);
e2=tmp->data.key;
}
}
if(e1==e2) return e1;//找到的情况
//同层时还没找到, 则同时向上找
while(NodeDepth(T,e1)!=1){
if(e1==e2) return e1;
tmp=GetParent(T,e1);
e1=tmp->data.key;
tmp=GetParent(T,e2);
e2=tmp->data.key;
}
return e1;
}


int main()
{
BiTree T=NULL,TC;
Lists.length=0;
```

```
TElemType e;
int choice;
while(1){
printf(
" 基于二叉链表的二叉树实现"
"————————————————————————————————————————"
" 1.CreateBiTree 10.DeleteNode"
" 2.DestroyBiTree 11.PreOrderTraverse"
" 3.ClearBiTree 12.InOrderTraverse"
" 4.BiTreeEmpty 13.PostOrderTraverse"
" 5.BiTreeDepth 14.LevelOrderTraverse"
" 6.LocateNode 15.SaveBiTree & LoadBiTree"
" 7.Assign 16.MaxPathSum"
" 8.GetSibling 17.LowestCommonAncestor"
" 9.InsertNode 18.InvertTree "
" 20.exit 19.TreeLists"
" "
"————————————————————————————————————————"
" 请输入你所选择的操作 [1-20]： ");
if(whe==0){
printf(" 当前模式为单二叉树模式");
}else{
printf(" 当前模式为多二叉树模式");
}
scanf("%d",&choice);
switch(choice)//选择菜单
{
case 0: {
printf(" 无效的操作！输入的数字应在 1 20 之间，请重新输入!");
break;
}
```

```
case 1: {
int j,i=0;
creatcnt=-1;
judgecreat=0;//做必要的初始化
printf(" 请输入数据 (以-1 null 结束):");
do {
scanf("%d %s",&definition[i].key,definition[i].others);
} while (definition[i++].key!=-1);
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=CreateBiTree(Lists.elem[Listnum-1].T,definition);
}else{
printf(" 输入的名称有误或不存在!");break;
}
}
if(whe==0){
j=CreateBiTree(T,definition);
}
if(j==INFEASIBLE) printf(" 不能对已经存在的二叉树初始化!");
if(j==ERROR) printf(" 关键字不唯一!");
if(j==OK) printf(" 成功初始化二叉树!");
break;
}
case 2:{
int j;
judgedestroy=0;
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
```

```
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=DestroyBiTree(Lists.elem[Listnum-1].T);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(whe==0){
j=DestroyBiTree(T);
}
if(j==INFEASIBLE) printf(” 二叉树不存在, 销毁失败!”);
if(j==OK) printf(” 已成功删除二叉树!”);
break;
}
case 3:{
int j;
if(whe==1){
printf(” 请输入想要操作的二叉树的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ClearBiTree(Lists.elem[Listnum-1].T);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(whe==0){
j=ClearBiTree(T);
}
if(j==INFEASIBLE) printf(” 二叉树不存在, 清空失败!”);
```

```
if(j==OK) printf(” 已成功清空二叉树!”);
break;
}
case 4:{
int j;
if(whe==1){
printf(” 请输入想要操作的二叉树的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=BiTreeEmpty(Lists.elem[Listnum-1].T);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(whe==0){
j=BiTreeEmpty(T);
}
if(j==INFEASIBLE) printf(” 二叉树不存在, 判空失败!”);
if(j==TRUE) printf(” 二叉树为空!”);
if(j==FALSE) printf(” 二叉树不为空!”);
break;
}
case 5:{
int j;
if(whe==1){
printf(” 请输入想要操作的二叉树的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=BiTreeDepth(Lists.elem[Listnum-1].T);
```

```
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=BiTreeDepth(T);
}
if(j==INFEASIBLE) printf(" 二叉树不存在, 获取深度失败!");
else printf(" 二叉树的深度为%d",j);
break;
}
case 6:{
int i;
printf(" 请输入想要获取元素的内容:");
scanf("%d",&i);
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
TC=LocateNode(Lists.elem[Listnum-1].T,i);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
TC=LocateNode(T,i);
}
//if(j==INFEASIBLE) printf(" 二叉树不存在, 获取失败!");
if(TC) printf(" 成功获取元素结点%s 并存储在 TC 中!",TC->data.others);
if(TC==NULL) printf(" 未查找到该元素!");
```

```
break;
}
case 7:{
int e1;
printf(" 请输入想要被替换的结点关键字:");
scanf("%d",&e1);
TElemType value;
printf(" 请输入想要将结点关键字替换为什么:");
scanf("%d %s",&value.key,value.others);
int j;
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=Assign(Lists.elem[Listnum-1].T,e1,value);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=Assign(T,e1,value);
}
if(j==INFEASIBLE) printf(" 二叉树不存在, 赋值失败!");
if(j==OK) printf(" 结点赋值成功!");
if(j==ERROR) printf(" 所要查找的元素不存在或关键字不唯一!");
break;
}
case 8:{
int e1;
printf(" 请输入想要获取何关键字的兄弟结点:");
```

```
scanf(”%d”,&e1);
int j;
if(whe==1){
printf(” 请输入想要操作的二叉树的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
TC=GetSibling(Lists.elem[Listnum-1].T,e1);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(whe==0){
TC=GetSibling(T,e1);
}
//if(j==INFEASIBLE) printf(” 二叉树不存在, 获取失败!”);
if(TC) printf(” 成功获取元素兄弟结点%d %s!”,TC->data.key,TC->data.others);
if(TC==NULL) printf(” 该元素没有兄弟结点或该元素不存在!”);
break;
}
case 9:{
int e1,LR;
TElemType c;
printf(” 请输入想要在何关键字处插入插入的操作为-1 或者 0 或者 1 时为根节点或左孩子或右孩子插入何值:”);
scanf(”%d %d %d %s”,&e1,&LR,&c.key,c.others);
int j;
if(whe==1){
printf(” 请输入想要操作的二叉树的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
```

```
if(Listnum!=0){
j=InsertNode(Lists.elem[Listnum-1].T,e1,LR,c);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=InsertNode(T,e1,LR,c);
}
if(j==INFEASIBLE) printf(" 二叉树不存在, 获取失败!");
if(j==OK) printf(" 成功插入元素!");
if(j==ERROR) printf(" 插入失败!");
break;
}
case 10:{
int e1;
printf(" 请输入想要删除的关键字:");
scanf("%d",&e1);
int j;
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=DeleteNode(Lists.elem[Listnum-1].T,e1);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=DeleteNode(T,e1);
```

```
}
if(j==INFEASIBLE) printf(" 二叉树不存在, 插入失败!");
if(j==OK) printf(" 成功删除!");
if(j==ERROR) printf(" 删除失败!");
break;
}
case 11:{
int j;
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=PreOrderTraverse(Lists.elem[Listnum-1].T,visit);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=PreOrderTraverse(T,visit);
}
if(j==INFEASIBLE) printf(" 二叉树不存在, 遍历失败!");
if(j==OK) printf(" 已成功先序遍历二叉树!");
//if(j==ERROR) printf(" 删除位置不正确!");
break;
}
case 12:{
int j;
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
```

```
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=InOrderTraverse(Lists.elem[Listnum-1].T,visit);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=InOrderTraverse(T,visit);
}
if(j==INFEASIBLE) printf(" 二叉树不存在, 遍历失败!");
if(j==OK) printf(" 已成功中序遍历二叉树!");
break;
}
case 13:{
int j;
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=PostOrderTraverse(Lists.elem[Listnum-1].T,visit);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=PostOrderTraverse(T,visit);
}
if(j==INFEASIBLE) printf(" 二叉树不存在, 遍历失败!");
if(j==OK) printf(" 已成功后序遍历二叉树!");
```

```
break;
}
case 14:{
int j;
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=LevelOrderTraverse(Lists.elem[Listnum-1].T,visit);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=LevelOrderTraverse(T,visit);
}
if(j==INFEASIBLE) printf(" 二叉树不存在, 遍历失败!");
if(j==OK) printf(" 已成功层序遍历二叉树!");
break;
}
case 15:{
char ListName[100];
int key;
file:
printf(" 请选择你的操作:1. 将二叉树写入文件 2. 将文件中的数据读入二叉树 3. 退出该功能");
scanf("%d",&key);
printf(" 请输入文件名");
fflush(stdin);
scanf("%[^]",ListName);
```

```
switch(key)
{
case 0:{
printf(" 无效的操作！输入的数字应在 1-3 之间，请重新输入!");
goto file;
break;
}
case 1:{
int j=SaveBiTree(T,ListName);
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=SaveBiTree(Lists.elem[Listnum-1].T,ListName);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 二叉树不存在, 写入失败!");
if(j==OK) printf(" 写入成功!");
break;
}
case 2:{
int j=LoadBiTree(T,ListName);
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=LoadBiTree(Lists.elem[Listnum-1].T,ListName);
```

```
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 二叉树已经存在, 读取失败!");
if(j==OK) printf(" 读取成功!");
break;
}
case 3:break;
default:{
printf(" 无效的操作！ 输入的数字应在 1 3 之间，请重新输入!");
goto file;
break;
}
}
break;
}
case 16:{
int j;
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=MaxPathSum(Lists.elem[Listnum-1].T);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=MaxPathSum(T);
```

```
}
if (j) printf(" 最长路径为%d!",j);
else printf(" 求最长路径失败!");
break;
}
case 17:{
int e1,e2;
printf(" 请输入想要查找哪两个顶点的最近公共祖先:");
scanf("%d %d",&e1,&e2);
int j;
if(whe==1){
printf(" 请输入想要操作的二叉树的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=LowestCommonAncestor(Lists.elem[Listnum-1].T,e1,e2);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=LowestCommonAncestor(T,e1,e2);
}
//if(j==INFEASIBLE) printf(" 线性表不存在或为空表, 删除失败!");
if(j==ERROR) printf(" 至少有一个顶点不在二叉树中!");
if(j) printf(" 最近公共祖先的关键字为%d!",j);
break;
}
case 18:{
BiTree j;
if(whe==1){
```

```
printf(” 请输入想要操作的二叉树的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=InvertTree(Lists.elem[Listnum].T);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(whe==0){
j=InvertTree(T);
}
if(j==NULL) printf(” 二叉树翻转失败!”);
if(j) printf(” 已成功翻转二叉树!”);
break;
}
case 19:{
printf(” 进入多二叉树操作模式!”);
whe=1;
printf(” 请选择你的操作:1. 多二叉树添加 2. 多二叉树删除 3. 多二叉树查找 4. 退出多二叉树模式”);
int key;
scanf(”%d”,&key);
switch(key)
{
case 1:{
printf(” 请输入想要创建的二叉树的名称:”);
scanf(”%s”,Listname);
int j=AddList(Lists,Listname);
if(j==INFEASIBLE) printf(” 二叉树不为空, 创建失败!”);
if(j==OK) printf(” 创建成功!”);
```

```
break;
}
case 2:{
printf(" 请输入想要删除的二叉树的名称:");
scanf("%s",Listname);
int j=RemoveList(Lists,Listname);
if(j==INFEASIBLE) printf(" 二叉树不存在, 删除失败!");
if(j==OK) printf(" 删除成功!");
break;
}
case 3:{
printf(" 请输入想要查找的二叉树的名称:");
scanf("%s",Listname);
int j=LocateList(Lists,Listname);
if(j==0) printf(" 不存在该名称的二叉树!");
else printf(" 该二叉树的序号为%d!",j);
break;
}
case 4:{
whe=0;
break;
}
default:{
printf(" 无效的操作！ 输入的数字应在 1-4 之间， 请重新输入!");
break;
}
}
break;
}
case 20:{
printf(" 已成功退出程序, 欢迎下次再使用本系统!");
```

```
return 0;
}
default:{
printf(” 无效的操作！输入的数字应在 1-20 之间，请重新输入!”);
break;
}
}
fflush(stdin);
system(”pause”);
system(”cls”);
}
return 0;
}
```

附录 D 基于邻接表图实现的源程序

```
/* 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 INFEASTABLE -1
#define OVERFLOW -2

#define MAX_VERTEX_NUM 20
typedef int status;
typedef int KeyType;
typedef enum {DG,DN,UDG,UDN} GraphKind;
typedef struct {
KeyType key;
char others[20];
} VertexType; //顶点类型定义


typedef struct ArcNode { //弧结点类型定义
int adjvex; //顶点位置编号
struct ArcNode *nextarc; //下一个表结点指针
} ArcNode;

typedef struct VNode{ //头结点及其数组类型定义
```

```
VertexType data; //顶点信息
ArcNode *firstarc; //指向第一条弧
} VNode,AdjList[MAX_VERTEX_NUM];

typedef struct { //邻接表的类型定义
AdjList vertices; //头结点数组
int vexnum,arcnum; //顶点数、弧数
GraphKind kind; //图的类型
} ALGraph;

typedef struct QueueNode//队列节点
{
VertexType data;
QueueNode* next;
}QueueNode, * pQueueNode;

typedef struct Queue//队列
{
pQueueNode front,rear;
}Queue, * pQueue;


typedef struct{ //线性表的集合类型定义
struct { char name[30];
ALGraph G;
}elem[10];
int length;
}LISTS;


LISTS Lists;//多图体系
```

```
int mark[MAX_VERTEX_NUM];
int arcmat[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
VertexType V[30];
KeyType VR[100][2];
const int inf = 0x3f3f3f3f; //无穷大的定义
int whe=0;
char Listname[30];
int Listnum;


int LocateVex(ALGraph G,KeyType u)
{//定位端点, 成功则返回下标 i, 反之返回-1;
int i;
for(i=0;i<G.vexnum;i++){
if(u==G.vertices[i].data.key) return i;
}
return INFEASIBLE;
}

pQueue InitQueue(pQueue pq)//建立只有头结点的队列
{
pq->front = (pQueueNode)malloc(sizeof(QueueNode));
pq->front->next = NULL;//队列的 front 和 rear 的 next 初始化为空
pq->rear = pq->front;
return pq;
}

void EnQueue(pQueue pq,VertexType t)//把图的数据取出放入队列
{
pQueueNode pNew = new QueueNode;
pNew->data = t;//图的数据存入队列
```

```
pNew->next = NULL;
pq->rear->next = pNew;//尾插法建立连接
pq->rear = pNew;//rear 更新
}

int DeQueue(pQueue pq,int &u,ALGraph G)//出队：删除队列第一个元素并存储在 u 中
{
pQueueNode pTemp= (pQueueNode)malloc(sizeof(QueueNode));
pTemp = pq->front->next;
if (pTemp->next == NULL)//只剩下 1 个节点（不含队列空的头结点）
{
pq->rear = pq->front;
}
else{
pq->front->next = pTemp->next;//front+1（从指向第 1 个非空节点改为指向第 2 个节点）
}
VertexType x;
x= pTemp->data;//x 为队列第一个元素的 data
free(pTemp);
u=LocateVex(G,x.key);
return OK;
}

int QueueEmpty(pQueue pq)
{
if(pq->rear == pq->front) return 0;
return 1;
}
```

```
status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2])
/* 根据 V 和 VR 构造图 T 并返回 OK，如果 V 和 VR 不正确，返回 ERROR
如果有相同的关键字，返回 ERROR。此题允许通过增加其它函数辅助实现本关
任务 */
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
//judge error
//空图返回 ERROR(什么破玩意)
int size=MAX_VERTEX_NUM;
if(V[0].key==-1) return ERROR;
int vexnum=0;
for(vexnum=0;V[vexnum].key!=-1;vexnum++);
int i,j,k;
for(i=0;i<vexnum-1;i++){
for(j=i+1;j<vexnum;j++){
if(V[i].key==V[j].key) return ERROR;//判重
}
}
int arcnum=0;
for(arcnum=0;VR[arcnum][0]!=-1;arcnum++);
G.vexnum=vexnum;
G.arcnum=arcnum;
while(vexnum>size){
//G.vertices=(VNode*)realloc(G.vertices,size+MAX_VERTEX_NUM);
//size=size+MAX_VERTEX_NUM;
return ERROR;
}
//copy 头结点数据
for(i=0;i<vexnum;i++){
G.vertices[i].data.key=V[i].key;
```

```
G.vertices[i].firstarc=NULL;
strcpy(G.vertices[i].data.others,V[i].others);
}
for(i=0;i<arcnum;i++){
if(LocateVex(G,VR[i][0])==-1||LocateVex(G,VR[i][1])==-1) return ERROR;//判存在
}
//建立指针的连接
ArcNode* p,*q;
for(i=0;i<arcnum;i++){
j=LocateVex(G,VR[i][0]);
p=new ArcNode;
p->adjvex=LocateVex(G,VR[i][1]);
p->nextarc=G.vertices[j].firstarc;
G.vertices[j].firstarc=p;
k=LocateVex(G,VR[i][1]);
q=new ArcNode;
q->adjvex=LocateVex(G,VR[i][0]);
q->nextarc=G.vertices[k].firstarc;
G.vertices[k].firstarc=q;
}
return OK;

/********** End **********/
}

status DestroyGraph(ALGraph &G)
/* 销毁无向图 G, 删除 G 的全部顶点和边 */
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i;
```

```
if(G.vexnum==0) return INFEASIBLE;
ArcNode* p;
for(i=0;i<G.vexnum;i++){
if(G.vertices[i].firstarc!=NULL){
p=G.vertices[i].firstarc;
while(p->nextarc!=NULL){
ArcNode* q=p->nextarc;
free(p);
p=q;
}
free(p);
G.vertices[i].firstarc=NULL;
}
//G.vertices[i].data.key=0;
}
G.vexnum=0;
G.arcnum=0;
return OK;
/********** End **********/
}

status PutVex(ALGraph &G,KeyType u,VertexType value)
//根据 u 在图 G 中查找顶点，查找成功将该顶点值修改成 value，返回 OK；
//如果查找失败或关键字不唯一，返回 ERROR
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i;
for(i=0;i<G.vexnum;i++){
if(value.key==G.vertices[i].data.key) return ERROR;
}
```

```
for(i=0;i<G.vexnum;i++){
if(u==G.vertices[i].data.key){
G.vertices[i].data.key=value.key;
strcpy(G.vertices[i].data.others,value.others);
return OK;
}
}
return ERROR;

/********** End **********/
}


int FirstAdjVex(ALGraph G,KeyType u)
//根据 u 在图 G 中查找顶点，查找成功返回顶点 u 的第一邻接顶点位序，否则返回-1；
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i,j,num;
i=LocateVex(G,u);
if(i==-1) return -1;
if(G.vertices[i].firstarc==NULL) return -1;
num=G.vertices[i].firstarc->adjvex;
return num;

/********** End **********/
}

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

```
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i,j,num;
i=LocateVex(G,v);
if(i==-1) return -1;
if(G.vertices[i].firstarc==NULL) return -1;
ArcNode* p=G.vertices[i].firstarc;
while(p->nextarc!=NULL){
if(p->adjvex==LocateVex(G,w)){
return p->nextarc->adjvex;
}
p=p->nextarc;
}
return -1;
/********** End **********/
}

status InsertVex(ALGraph &G,VertexType v)
//在图 G 中插入顶点 v，成功返回 OK, 否则返回 ERROR
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
if(G.vexnum>=MAX_VERTEX_NUM) return ERROR;
if(LocateVex(G,v.key)!=-1) return ERROR;
G.vertices[G.vexnum].data.key=v.key;
G.vertices[G.vexnum].firstarc=NULL;
strcpy(G.vertices[G.vexnum].data.others,v.others);
G.vexnum++;
```

```
return OK;
/********** End **********/
}

status DeleteVex(ALGraph &G,KeyType v)
//在图 G 中删除关键字 v 对应的顶点以及相关的弧，成功返回 OK, 否则返回 ER-
ROR
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i=0,j,num;
i=LocateVex(G,v);
if(i==-1) return -1;
if(G.vexnum==1) return -1;//删除后为空表报错
ArcNode* p=G.vertices[i].firstarc;
while(p!=NULL){
ArcNode* q=p->nextarc;
num=p->adjvex;
ArcNode* m=G.vertices[num].firstarc;
ArcNode* n=G.vertices[num].firstarc->nextarc;
if(m->adjvex==LocateVex(G,v)){
G.vertices[num].firstarc=m->nextarc;
free(m);
}
else{
while(m->nextarc!=NULL){
if(n->adjvex==LocateVex(G,v)){
m->nextarc=n->nextarc;
free(n);
break;
}
```

```
n=n->nextarc;
m=m->nextarc;
}
}
G.arcnum=G.arcnum-1;
free(p);
p=q;
}
G.vertices[i].firstarc=NULL;
for(j=i;j<G.vexnum;j++){
G.vertices[j].firstarc= G.vertices[j+1].firstarc;
G.vertices[j].data.key=G.vertices[j+1].data.key;
strcpy(G.vertices[j].data.others,G.vertices[j+1].data.others);
}
G.vexnum–;
for(j=0;j<G.vexnum;j++){
p=G.vertices[j].firstarc;
while(p!=NULL){
if(p->adjvex>i) p->adjvex–;
p=p->nextarc;
}
}
return OK;
/********** End **********/
}

status InsertArc(ALGraph &G,KeyType v,KeyType w)
//在图 G 中增加弧 <v,w>，成功返回 OK, 否则返回 ERROR
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
```

```
int vex1,vex2;
vex1=LocateVex(G,v);
vex2=LocateVex(G,w);
if(vex1==-1||vex2==-1) return ERROR;
ArcNode* test=G.vertices[vex1].firstarc;
while(test!=NULL){
if(test->adjvex==vex2) return ERROR;
test=test->nextarc;
}
ArcNode* p=(ArcNode*)malloc(sizeof(ArcNode));
p->adjvex=vex2;
p->nextarc=G.vertices[vex1].firstarc;
G.vertices[vex1].firstarc=p;
ArcNode* q=(ArcNode*)malloc(sizeof(ArcNode));
q->adjvex=vex1;
q->nextarc=G.vertices[vex2].firstarc;
G.vertices[vex2].firstarc=q;
G.arcnum++;
return OK;
/********** End **********/
}

status DeleteArc(ALGraph &G,KeyType v,KeyType w)
//在图 G 中删除弧 <v,w>，成功返回 OK, 否则返回 ERROR
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int vex1,vex2;
vex1=LocateVex(G,v);
vex2=LocateVex(G,w);
if(vex1==-1||vex2==-1) return ERROR;
```

```
ArcNode* p,*q;
p=G.vertices[vex1].firstarc;
if(p==NULL) return ERROR;
if(p->adjvex==vex2){
free(p);
G.vertices[vex1].firstarc=NULL;
goto next;
}
q=p->nextarc;
while(q!=NULL){
if(q->adjvex==vex2){
p->nextarc=q->nextarc;
free(q);
goto next;
}
p=p->nextarc;
q=q->nextarc;
}
return ERROR;
next:
p=G.vertices[vex2].firstarc;
if(p==NULL) return ERROR;
if(p->adjvex==vex1){
free(p);
G.vertices[vex2].firstarc=NULL;
G.arcnum–;
return OK;
}
q=p->nextarc;
while(q!=NULL){
if(q->adjvex==vex1){
```

```
p->nextarc=q->nextarc;
free(q);
G.arcnum–;
return OK;
}
p=p->nextarc;
q=q->nextarc;
}
return ERROR;
/********** End **********/
}

void visit(VertexType v)
{
printf(" %d %s",v.key,v.others);
}

void DFS(ALGraph G,int v,void (*visit)(VertexType))
{
mark[v]=1;
visit(G.vertices[v].data);
int w;
for(w=FirstAdjVex(G,G.vertices[v].data.key);w>=0;w=NextAdjVex(G,G.vertices[v].data.key,G.vertices[
if(mark[w]==0) DFS(G,w,visit);
}
}

status DFSTraverse(ALGraph &G,void (*visit)(VertexType))
//对图 G 进行深度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问
一次，且仅访问一次
{
```

```
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int v;
for(v=0;v<G.vexnum;v++) mark[v]=0;//初始化
for(v=0;v<G.vexnum;v++){
if(mark[v]==0) DFS(G,v,visit);
}
return OK;
/********** End **********/
}

status BFSTraverse(ALGraph &G,void (*visit)(VertexType))
//对图 G 进行广度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问
一次，且仅访问一次
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int v,u,w;
for(v=0;v<G.vexnum;++v) mark[v]=0;
pQueue Q=(pQueue)malloc(sizeof(Queue));
Q=InitQueue(Q);
for(v=0;v<G.vexnum;++v){
if(mark[v]==0){
mark[v]=1;
visit(G.vertices[v].data);
EnQueue(Q,G.vertices[v].data);
while(QueueEmpty(Q)!=0){
DeQueue(Q,u,G);
for(w=FirstAdjVex(G,G.vertices[u].data.key);w>=0;w=NextAdjVex(G,G.vertices[u].data.key,G.vertices[
if(mark[w]==0){
visit(G.vertices[w].data);
```

```
mark[w]=1;
EnQueue(Q,G.vertices[w].data);
}
}
}
}
}
return OK;
/********** End **********/
}

status SaveGraph(ALGraph G, char FileName[])
//将图的数据写入到文件 FileName 中
{
// 请在这里补充代码，完成本关任务
/********** Begin 1 *********/
FILE* fp=fopen(FileName,”w+”);
int i;
ArcNode* p;
for(i=0;i<G.vexnum;i++){
fprintf(fp,”%d ”,G.vertices[i].data.key);
fprintf(fp,”%s ”,G.vertices[i].data.others);
if(G.vertices[i].firstarc!=NULL){
p=G.vertices[i].firstarc;
while(p!=NULL){
fprintf(fp,”%d ”,p->adjvex);
p=p->nextarc;
}
}
fprintf(fp,”-1 ”);//末尾用-1 代替
//G.vertices[i].data.key=0;
```

```
}
fclose(fp);
return OK;
/********** End 1 **********/
}

status LoadGraph(ALGraph &G, char FileName[])
//读入文件 FileName 的图数据，创建图的邻接表
{
// 请在这里补充代码，完成本关任务
/********** Begin 2 *********/
FILE* fp=fopen(FileName,”r+”);
int headnum=0,i=0,arcnum=0;
ArcNode* p,*q;
while((fscanf(fp,”%d”,&headnum))!=EOF){//头结点创建完毕
G.vertices[i].data.key=headnum;
fscanf(fp,”%s”,G.vertices[i].data.others);
G.vexnum++;//读取头结点
//接下来读取弧结点
fscanf(fp,”%d”,&arcnum);
q=G.vertices[i].firstarc;
if(arcnum==-1){
G.vertices[i].firstarc=NULL;
i++;
continue;
}
if(arcnum!=-1){
p=new ArcNode;
p->adjvex=arcnum;
G.vertices[i].firstarc=p;
q=p;
```

```
}
fscanf(fp,”%d”,&arcnum);
while(arcnum!=-1){
p=new ArcNode;
p->adjvex=arcnum;
q->nextarc=p;
q=p;
G.arcnum++;
fscanf(fp,”%d”,&arcnum);//直到读到-1
}
p->nextarc=NULL;
i++;
}
fclose(fp);
return OK;
/********** End 2 **********/
}

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
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i,j;
for(i=0;i<Lists.length;i++){
if(strcmp(ListName,Lists.elem[i].name)==0){
return i+1;
}
}
return 0;
```

```
/********** End **********/
}

status AddList(LISTS &Lists,char ListName[])
// 只需要在 Lists 中增加一个名称为 ListName 的空线性表，线性表数据又后台测试程序插入。
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i=0,j;
printf(" 请输入顶点数据 (以-1 nil 结束):");
do {
scanf("%d%s",&V[i].key,V[i].others);
} while(V[i++].key!=-1);
i=0;
printf(" 请输入边数据 (以-1 -1 结束):");
do {
scanf("%d%d",&VR[i][0],&VR[i][1]);
} while(VR[i++][0]!=-1);
j=CreateGraph(Lists.elem[Lists.length].G,V,VR);
if(j==OK){
//导入表名
strcpy(Lists.elem[Lists.length].name,ListName);
Lists.length++;
return OK;
}
return INFEASIBLE;
/********** End **********/
}
```

```
status RemoveList(LISTS &Lists,char ListName[])
// Lists 中删除一个名称为 ListName 的线性表
{
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int i,j;
for(i=0;i<Lists.length;i++){
if(strcmp(ListName,Lists.elem[i].name)==0){
for(j=i;j<Lists.length-1;j++){
Lists.elem[j]=Lists.elem[j+1];
}
Lists.length–;
return OK;
}
}
return INFEASIBLE;

/********** End **********/
}

void table_convert_matrix(ALGraph G) { // 邻接表转化为邻接矩阵
int i,j;
for(i = 1; i <= G.vexnum; i++) {
for(j = 1; j <= G.vexnum; j++) {
arcmat[i][j] = inf; // 初始化邻接矩阵
}
}
ArcNode *p;
for(i = 1; i <= G.vexnum; i++) { //依次遍历各顶点表结点为头的边链表
p = G.vertices[i-1].firstarc; // 取出顶点 i 的第一条出边
while(p) { //遍历边链表
```

```
arcmat[i][p->adjvex+1] = 1;
p = p -> nextarc; // 取出下一条出边
}
}
for(i=1;i<=G.vexnum; i++){
arcmat[i][i] = 0;//自身到自身的距离为 0;
}
}

int min(int a,int b)
{//求两数的最小值
if(a<=b) return a;
else return b;
}

void Floyd(ALGraph G)
{ //表示从 j 点到 k 点的最短路径，其中的 i 为中间的中转点
for (int i = 1; i <= G.vexnum; ++i) //i 为中转在最外层
for (int j = 1; j <= G.vexnum; ++j)
for (int k = 1; k <= G.vexnum; ++k)
arcmat[j][k] = min(arcmat[j][k], arcmat[j][i] + arcmat[i][k]);
}

int VerticesSetLessThanK(ALGraph G,int v,int k)
{//求距离小于 k 的顶点集合
int i;
table_convert_matrix(G);
Floyd(G);//Floyd 算法
int v0=LocateVex(G,v)+1;//下标不统一, 故 +1
printf(" 与顶点 v 距离小于 k 的顶点集合为:");
for(i=1;i<=G.vexnum;i++){
```

```
if(arcmat[v0][i]<k){
printf(" %d %s",G.vertices[i-1].data.key,G.vertices[i-1].data.others);
}
}
return OK;
}

int ShortestPathLength(ALGraph G,int v,int w)
{//求两顶点间的最短路径函数
table_convert_matrix(G);
Floyd(G);//Floyd 算法
int v0=LocateVex(G,v)+1;//下标不统一, 故 +1
int w0=LocateVex(G,w)+1;//下标不统一, 故 +1
return arcmat[v0][w0];
}

void DFS0(ALGraph G,int v)
{//深度优先搜索辅助求图的连通分量个数函数
mark[v]=1;
//visit(G.vertices[v].data);
int w;
for(w=FirstAdjVex(G,G.vertices[v].data.key);w>=0;w=NextAdjVex(G,G.vertices[v].data.key,G.vertices[
if(mark[w]==0) DFS0(G,w);
}
}

int ConnectedComponentsNums(ALGraph G)
{//求图的连通分量个数函数
int v,cnt=0;
for(v=0;v<G.vexnum;v++) mark[v]=0;//初始化
for(v=0;v<G.vexnum;v++){
```

```
if(mark[v]==0){
DFS0(G,v);
cnt++;
}
}
return cnt;
}


int main()//头歌中测试集 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7
8 -1 -1
{
ALGraph G;
G.vexnum=0;
G.arcnum=0;
Lists.length=0;
int choice;
while(1){
printf(
" 基于邻接表的图实现"
"                                                                                "
" 1.CreateGraph 10.DeleteArc"
" 2.DestroyGraph 11.DFSTraverse"
" 3.LocateVex 12.BFSTraverse"
" 4.PutVex 13.VerticesSetLessThanK"
" 5.FirstAdjVex 14.ShortestPathLength"
" 6.NextAdjVex 15.SaveGraph & LoadGraph"
" 7.InsertVex 16.ConnectedComponentsNums"
" 8.DeleteVex 17.GraphLists"
" 9.InsertArc 18.exit "
" "
```

```
”                                                                                ”
” 请输入你所选择的操作 [1-18]：”);
if(whe==0){
printf(” 当前模式为单图模式”);
}else{
printf(” 当前模式为多图模式”);
}
scanf(”%d”,&choice);
switch(choice)//选择菜单
{
case 0: {
printf(” 无效的操作！输入的数字应在 1-18 之间，请重新输入!”);
break;
}
case 1: {
int i=0,j;
if(G.vexnum!=0){
printf(” 不能对已经存在的图初始化!”);
break;
}
printf(” 请输入顶点数据 (以-1 nil 结束):”);
do {
scanf(”%d%s”,&V[i].key,V[i].others);
} while(V[i++].key!=-1);
i=0;
printf(” 请输入边数据 (以-1 -1 结束):”);
do {
scanf(”%d%d”,&VR[i][0],&VR[i][1]);
} while(VR[i++][0]!=-1);
if(whe==1){
printf(” 请输入想要操作的图的名称:”);
```

```
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=CreateGraph(Lists.elem[Listnum-1].G,V,VR);
}else{
printf(” 输入的名称有误或不存在!”);break;
}
}
if(whe==0){
j=CreateGraph(G,V,VR);
}
if(j==ERROR) printf(” 输入的顶点或弧不正确!”);
if(j==OK) printf(” 成功创建无向图!”);
break;
}
case 2:{
int j;
if(whe==1){
printf(” 请输入想要操作的图的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=DestroyGraph(Lists.elem[Listnum-1].G);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(whe==0){
j=DestroyGraph(G);
}
if(j==INFEASIBLE) printf(” 图不存在, 销毁失败!”);
```

```
if(j==OK) printf(” 已成功销毁图!”);
break;
}
case 3:{
int j;
int u;
printf(” 请输入顶点关键字:”);
scanf(”%d”,&u);
if(whe==1){
printf(” 请输入想要操作的图的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=LocateVex(Lists.elem[Listnum-1].G,u);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(whe==0){
j=LocateVex(G,u);
}
if(j==INFEASIBLE) printf(” 该顶点不存在!”);
if(j>=0) printf(” 该顶点在 G 中的位序为%d!”,j);
break;
}
case 4:{
int j,u;
VertexType value;
printf(” 请输入想要赋值的顶点关键字:”);
scanf(”%d”,&u);
printf(” 请输入想要赋的值:”);
```

```
scanf(”%d %s”,&value.key,value.others);
if(whe==1){
printf(” 请输入想要操作的图的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=PutVex(Lists.elem[Listnum-1].G,u,value);
}else{
printf(” 输入的名称有误或不存在!”);
}
}
if(whe==0){
j=PutVex(G,u,value);
}
if(j==ERROR) printf(” 查找失败或关键字不唯一!”);
if(j==OK) printf(” 赋值成功!”);
break;
}
case 5:{
int j;
int u;
printf(” 请输入获取何顶点的第一邻接顶点:”);
scanf(”%d”,&u);
if(whe==1){
printf(” 请输入想要操作的图的名称:”);
scanf(”%s”,Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=FirstAdjVex(Lists.elem[Listnum-1].G,u);
}else{
printf(” 输入的名称有误或不存在!”);
```

```
}
}
if(whe==0){
j=FirstAdjVex(G,u);
}
if(j!=-1) printf(" 第一临接点为:%d %s",G.vertices[j].data.key,G.vertices[j].data.others);
else printf(" 查找失败");
break;
}
case 6:{
int j;
int v,w;
printf(" 请输入查找 v 顶点的邻接点中 w 的下一个 (中间以空格分隔):");
scanf("%d %d",&v,&w);
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=NextAdjVex(Lists.elem[Listnum-1].G,v,w);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=NextAdjVex(G,v,w);
}
if(j!=-1) printf(" 下一临接点为:%d %s",G.vertices[j].data.key,G.vertices[j].data.others);
else printf(" 无下一邻接顶点或未查找到临接点");
break;
}
```

```
case 7:{
VertexType v;
printf(" 请输入想要添加的新顶点 v:");
scanf("%d %s",&v.key,v.others);
int j;
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=InsertVex(Lists.elem[Listnum-1].G,v);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=InsertVex(G,v);
}
if(j==OK) printf(" 顶点插入成功!");
if(j==ERROR) printf(" 所要插入的顶点关键字不唯一或顶点个数已满!");
break;
}
case 8:{
int e1;
printf(" 请输入想要删除何关键字的顶点:");
scanf("%d",&e1);
int j;
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
```

```
if(Listnum!=0){
j=DeleteVex(Lists.elem[Listnum-1].G,e1);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=DeleteVex(G,e1);
}
//if(j==INFEASIBLE) printf(" 二叉树不存在, 获取失败!");
if(j==OK) printf(" 已成功删除顶点!");
if(j==-1) printf(" 删除失败!");
break;
}
case 9:{
int v,w;
printf(" 请输入插入边 <v,w>(以空格分隔):");
scanf("%d %d",&v,&w);
int j;
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=InsertArc(Lists.elem[Listnum-1].G,v,w);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=InsertArc(G,v,w);
```

```
}
//if(j==INFEASIBLE) printf(" 二叉树不存在, 获取失败!");
if(j==OK) printf(" 成功插入弧!");
if(j==ERROR) printf(" 插入失败!");
break;
}
case 10:{
int v,w;
printf(" 请输入想要删除的弧 <v,w>:");
scanf("%d %d",&v,&w);
int j;
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=DeleteArc(Lists.elem[Listnum-1].G,v,w);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=DeleteArc(G,v,w);
}
//if(j==INFEASIBLE) printf(" 二叉树不存在, 插入失败!");
if(j==OK) printf(" 成功删除!");
if(j==ERROR) printf(" 删除失败!");
break;
}
case 11:{
int j;
```

```
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=DFSTraverse(Lists.elem[Listnum-1].G,visit);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=DFSTraverse(G,visit);
}
if(j==OK) printf(" 已成功深度优先搜索图!");
else printf(" 深度优先搜索失败!");
//if(j==ERROR) printf(" 删除位置不正确!");
break;
}
case 12:{
int j;
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=BFSTraverse(Lists.elem[Listnum-1].G,visit);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
```

```
j=BFSTraverse(G,visit);
}
if(j==OK) printf(" 已成功广度优先搜索图!");
else printf(" 广度优先搜索失败!");
break;
}
case 13:{
int j;
int v,k;
printf(" 请输入获取与顶点 v 路径小于 k 的顶点中的 v,k(以空格分隔):");
scanf("%d %d",&v,&k);
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=VerticesSetLessThanK(Lists.elem[Listnum-1].G,v,k);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=VerticesSetLessThanK(G,v,k);
}
if(j==OK) printf(" 已成功获取与顶点 v 路径小于 k 的顶点集合!");
else printf(" 获取集合失败!");
break;
}
case 14:{
int j;
int v,w;
```

```
printf(" 请输入顶点 v 和顶点 w(以空格分隔):");
scanf("%d %d",&v,&w);
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ShortestPathLength(Lists.elem[Listnum-1].G,v,w);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=ShortestPathLength(G,v,w);
}
if(j>0&&j<inf) printf(" 顶点 v 到 w 的最短路径为%d!",j);
if(j==inf) printf(" 两顶点间不连通!");
if(j==0) printf(" 两顶点为同一点!");
break;
}
case 15:{
char ListName[100];
int key;
file:
printf(" 请选择你的操作:1. 将图写入文件 2. 将文件中的数据读入图 3. 退出该功能");
scanf("%d",&key);
printf(" 请输入文件名");
fflush(stdin);
scanf("%[^]",ListName);
switch(key)
```

```
{
case 0:{
printf(" 无效的操作！输入的数字应在 1-3 之间，请重新输入!");
goto file;
break;
}
case 1:{
int j=SaveGraph(G,ListName);
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=SaveGraph(Lists.elem[Listnum-1].G,ListName);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 图不存在, 写入失败!");
if(j==OK) printf(" 写入成功!");
break;
}
case 2:{
int j=LoadGraph(G,ListName);
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=LoadGraph(Lists.elem[Listnum-1].G,ListName);
}else{
```

```
printf(" 输入的名称有误或不存在!");
}
}
if(j==INFEASIBLE) printf(" 图已经存在, 读取失败!");
if(j==OK) printf(" 读取成功!");
break;
}
case 3:break;
default:{
printf(" 无效的操作！ 输入的数字应在 1-3 之间，请重新输入!");
goto file;
break;
}
}
break;
}
case 16:{
int j;
if(whe==1){
printf(" 请输入想要操作的图的名称:");
scanf("%s",Listname);
Listnum=LocateList(Lists,Listname);
if(Listnum!=0){
j=ConnectedComponentsNums(Lists.elem[Listnum-1].G);
}else{
printf(" 输入的名称有误或不存在!");
}
}
if(whe==0){
j=ConnectedComponentsNums(G);
}
```

```
if (j) printf(" 图的连通分量有%d 个!",j);
else printf(" 求图连通分量失败!");
break;
}
case 17:{
printf(" 进入多图操作模式!");
whe=1;
printf(" 请选择你的操作:1. 多图添加 2. 多图删除 3. 多图查找 4. 退出多图模式");
int key;
scanf("%d",&key);
switch(key)
{
case 1:{
printf(" 请输入想要创建的图的名称:");
scanf("%s",Listname);
int j=AddList(Lists,Listname);
if(j==INFEASIBLE) printf(" 创建失败!");
if(j==OK) printf(" 创建成功!");
break;
}
case 2:{
printf(" 请输入想要删除的图的名称:");
scanf("%s",Listname);
int j=RemoveList(Lists,Listname);
if(j==INFEASIBLE) printf(" 图不存在, 删除失败!");
if(j==OK) printf(" 删除成功!");
break;
}
case 3:{
printf(" 请输入想要查找的图的名称:");
scanf("%s",Listname);
```

```
int j=LocateList(Lists,Listname);
if(j==0) printf(" 不存在该名称的图!");
else printf(" 该图的序号为%d!",j);
break;
}
case 4:{
whe=0;
break;
}
default:{
printf(" 无效的操作！输入的数字应在 1-4 之间，请重新输入!");
break;
}
}
break;
}
case 18:{
printf(" 已成功退出程序, 欢迎下次再使用本系统!");
return 0;
}
default:{
printf(" 无效的操作！输入的数字应在 1-20 之间，请重新输入!");
break;
}
}
system("pause");
system("cls");
}
return 0;
}
```