

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 物联网工程1601**

**学 号： U201614887**

**姓 名： 林子涵**

**指导教师： 李国徽**

**报告日期： 2018年 1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 2](#_Toc458159879)

[1.1 问题描述 2](#_Toc458159880)

[1.2 系统设计 4](#_Toc458159882)

[1.3 系统实现 9](#_Toc458159883)

[1.4 实验小结 13](#_Toc458159884)

[2 基于链式存储结构的线性表实现 15](#_Toc458159885)

[2.1 问题描述 15](#_Toc458159886)

[2.2 系统设计 15](#_Toc458159887)

[2.3 系统实现 16](#_Toc458159888)

[2.4 实验小结 22](#_Toc458159889)

[3 基于二叉链表的二叉树实现 23](#_Toc458159890)

[3.1 问题描述 23](#_Toc458159891)

[3.2 系统设计 23](#_Toc458159892)

[3.3 系统实现 26](#_Toc458159893)

[3.4 实验小结 34](#_Toc458159894)

[4 基于二叉链表的二叉树实现 35](#_Toc458159895)

[4.1 问题描述 35](#_Toc458159896)

[4.2 系统设计 35](#_Toc458159897)

[4.3 系统实现 36](#_Toc458159898)

[4.4 实验小结 44](#_Toc458159899)

[参考文献 45](#_Toc458159900)

[附录A 基于顺序存储结构线性表实现的源程序 48](#_Toc458159901)

[附录B 基于链式存储结构线性表实现的源程序 66](#_Toc458159902)

[附录C 基于二叉链表二叉树实现的源程序 85](#_Toc458159903)

[附录D 基于邻接表图实现的源程序 111](#_Toc458159904)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

叙述实验中线性表的物理结构形式，如何用物理结构表示数据元素间的逻辑关系，可用图的方式直观表示物理结构，如图1-1所示。



图1-1 顺序表物理结构示意图

实验要完成的顺序表算法：

（1）InitaList(&L)

操作结果：构造一个空的线性表。

（2）DestroyList(&L)

初始条件：线性表L已存在。

操作结果：销毁线性表L。

（3）ClearList (&L)

初始条件：线性表L已存在。

操作结果：将L重置为空表。

（4）ListEmpty(L)

初始条件：线性表L已存在。

操作结果：若L为空表，则返回TRUE,否则返回FALSE。

（5）ListLength(L)

初始条件：线性表已存在。

操作结果：返回L中数据元素的个数。

（6）GetElem(L, i,&e)

初始条件：线性表已存在，1≤i≤ListLength(L)。

操作结果：用e返回L中第i个数据元素的值。

（7）LocateElem(L, e，compare（）)

初始条件：线性表已存在。

操作结果：返回L中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。

（8）PriorElem（L，cur\_e，&pre\_e）

初始条件：线性表L已存在。

操作结果：若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre\_e无定义。

（9）NextElem（L，cur\_e，&next\_e）

初始条件：线性表L已存在。

操作结果：若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败，next\_e无定义。

（10）ListInsert(&L,i,e)

初始条件：线性表L已存在且非空，1≤i≤ListLength(L)+1。

操作结果：在L的第i个位置之前插入新的数据元素e，L的长度加1

（11）ListDelete(&L,i,&e)

初始条件：线性表L已存在且非空，1≤i≤ListLength(L)。

操作结果：删除L的第i个数据元素，用e返回其值，L的长度减1.

（12）ListTraverse(L，visit（）)

初始条件：线性表L已存在。

操作结果：依次对L的每个数据元素调用函数visit()。一旦调用失败，则操作失败。

（13）ReadList(&L)

操作结果：读取线性表

（14）SaveList(L)

操作结果：保存线性表

## 1.1.1 实验目的

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

## 1.2 系统设计

## 1.2.1 系统总体设计

系统的总体架构：界面上采用简易菜单，通过switch函数进行功能的选择，进入相关功能函数执行相关操作。

## 1.2.2 有关常量和类型定义

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;

**1.2.3 算法设计**

（1）InitaList(&L)

设计：分配存储空间，并将length值设为0，listsize值设为预定义的初始存储容量。

复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

1. DestroyList(&L)

设计：释放内存，并让ele指向NULL，表长及存储容量置为0。

复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（3）ClearList (&L)

设计：将表长设为0。

复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（4）ListEmpty(L)

设计：读取表长length的值，为0则返回TRUE，否则FALSE。

复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（5）ListLength(L)

设计：返回L.length的值。

复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（6）GetElem(L, i,&e)

设计：将L.elem[i - 1]的值赋给e，并返回OK

复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（7）LocateElem(L, e，compare（）)

设计：遍历顺序表，将与给定元素e满足关系compare的元素的位序返回。

复杂度：时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. PriorElem（L，cur\_e，&pre\_e）

设计：从头遍历，若当前节点后继值为cur\_e，则将当前结点的元素赋给pre\_e，return OK。若未找到，则返回FALSE。流程图如下所示：



复杂度：时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. NextElem（L，cur\_e，&next\_e）

设计：从头遍历，若当前节点值为cur\_e且非表尾，则将后继结点的元素赋给pre\_e，return OK。若未找到，则返回FALSE。流程图如下：



复杂度：时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. ListInsert(&L,i,e)

设计：先判断i值是否符合要求，不符返回ERROR。i值合法，则检查空间大小，若空间不够，增加存储容量。然后将插入位置及之后的元素右移，最后插入e，表长加一。流程图如下所示：



复杂度：时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. ListDelete(&L,i,&e)

设计：先判断i值是否符合要求，不符返回ERROR。i值合法，则将被删除元素的值赋给e，然后将被删除元素之后的元素左移 ，最后表长减一。

流程图如下所示：



复杂度：时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. ListTraverse(L，visit（）)

设计：遍历顺序表，对每个元素调用函数指针visit所指向的函数，若调用失败则提示操作失败。

复杂度：时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. LoadFromFile(char\* filename ，SqList &L)

设计：若原L已存在，销毁，新建，再要求用户输入读取文件名，然后使用fread函数进行文件的读取。

复杂度：时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. SaveToFile(char\* filename ，SqList L)

设计：让用户输入所要保存到的文件名，然后使用fwrite函数将顺序表保存入文件中。

复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

## 1.3 系统实现

**基于数组的线性表结构定义：**

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

**函数分析：**

（1）InitaList(&L)

实现：通过malloc为L.elem分配存储空间，大小为初始存储容量，如果没有申请到则函数返回错误。将length值设为0，listsize值设为预定义的初始存储容量。返回OK

（2）DestroyList(&L)

实现：通过free释放L.elem，表长及存储容量置为0,返回OK。

（3）ClearList (&L)

实现：将表长设为0，listsize值设为初始存储容量。

（4）ListEmpty(L)

实现：对表未初始化情况进行异常判断。如果读取表长length的值，为0则返回TRUE，否则FALSE。

（5）ListLength(L)

实现：对表未初始化情况进行异常判断，返回L.length的值。

（6）GetElem(L, i,&e)

实现：对不合法的输入进行异常判断。将L.elem[i - 1]的值赋给e，并返回OK

（7）LocateElem(L, e，compare（）)

实现：从第一个元素遍历顺序表，将与给定元素e满足关系compare的元素（在此用==关系）的位序返回，如果未找到则返回0。

（8）PriorElem（L，cur\_e，&pre\_e）

实现：从头遍历，若当前节点为cur\_e，则将当前结点前一节点的元素赋给pre\_e，return OK。若未找到，则返回ERROR。：

（9）NextElem（L，cur\_e，&next\_e）

实现：从头遍历，若当前节点值为cur\_e且非表尾，则将后继结点的元素赋给pre\_e，return OK。若未找到，则返回FALSE。

（10）ListInsert(&L,i,e)

实现：先判断i值是否符合要求，不符返回ERROR。i值合法，则检查空间大小，若空间不够，使用realloc方法增加存储容量，并将listsize值加上相应值。然后将插入位置及之后的元素右移，最后插入e，表长加一。

（11）ListDelete(&L,i,&e)

实现：先判断i值是否符合要求，不符返回ERROR。i值合法，则将被删除元素的值赋给e，然后将被删除元素之后的元素左移 ，最后表长减一，返回OK。

（12）ListTraverse(L，visit（）)

实现：遍历顺序表，对每个元素调用函数指针visit（这里用printf代替）所指向的函数，返回表长

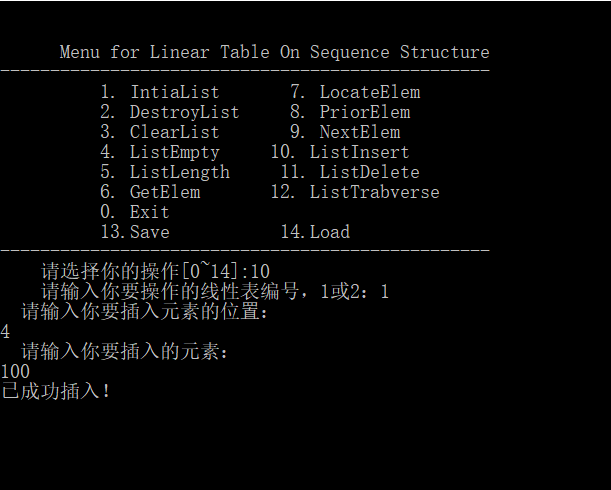
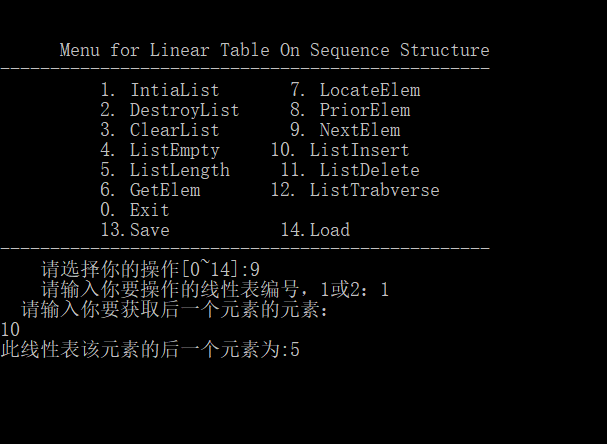
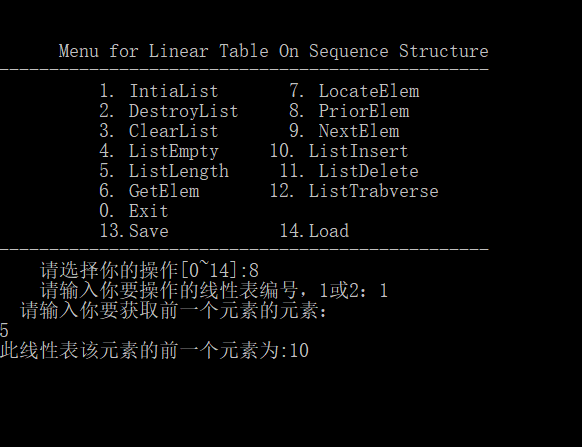
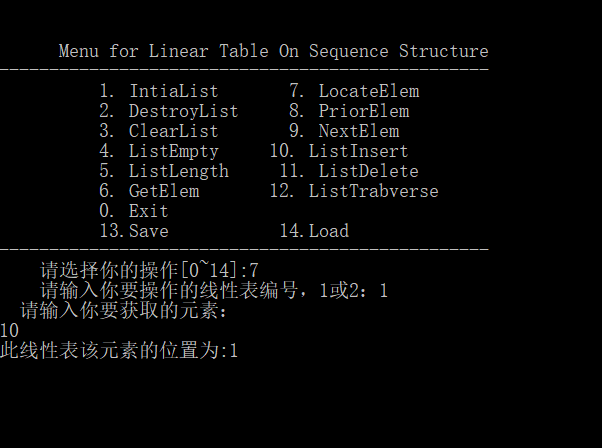
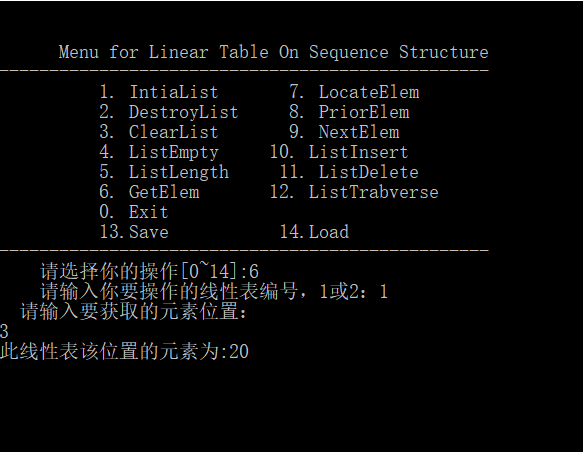
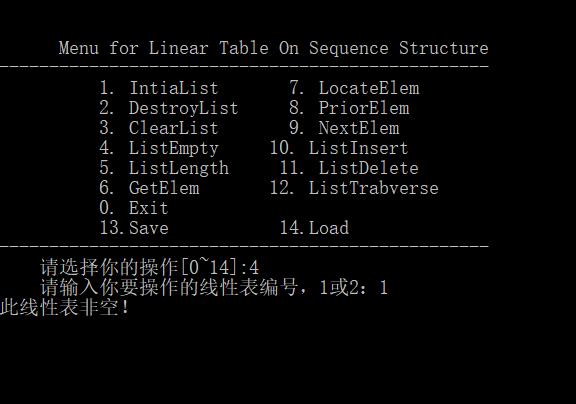
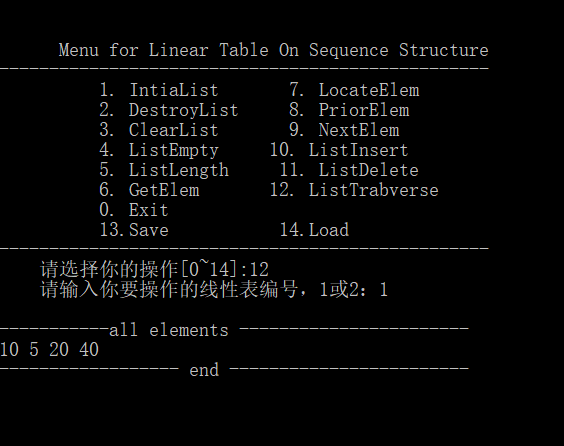
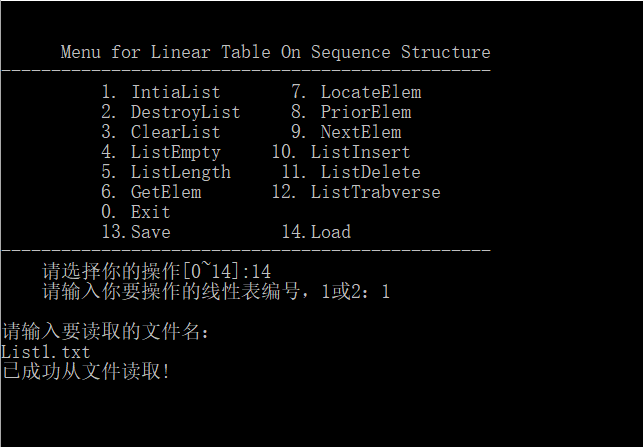
（13）LoadFromFile(char\* filename ，SqList &L)

实现：调用IntiaList初始化L，再用fopen打开指定文件名文件，然后使用fread函数进行文件的读取。

（14）SaveToFile(char\* filename ，SqList L)

实现：根据参数打开指定文件，然后使用fwrite函数将顺序表保存入文件中。

## 1.4测试



## 1.5 实验小结

本次实验加深了对线性表的概念、基本运算的理解，掌握了线性表的基本运算的实现。尤其加深了对于各种异常操作的处理。通过本次实验我了解了健壮可靠的函数接口对于数据存储的重要性，今后的学习过程中应当多从数据结构的角度分析如何进行数据的处理、存储以方便问题的解决，并要勤加练习达到熟能生巧的地步，还要多多拓展，掌握同一结构的各种实现方式。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

掌握基于链式存储结构，实现线性表的基本的、常见的运算

## 2.1.1 实验目的

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

## 2.2 系统设计

一共有14个基本函数，具体功能如下

（1）InitList(LinkList & L)

操作结果：构造一个空的链式表

（2）DestroyList(LinkList & L)

初始条件：链式表L已存在。

操作结果：销毁链式表L。

（3）ClearList (LinkList & L)

初始条件：链式表L已存在。

操作结果：将L重置为空链式表。

（4）ListEmpty(LinkList & L)

初始条件：链式表L已存在。

操作结果：若L为空链式表，则返回TRUE,否则返回FALSE.

（5）ListLength(LinkList & L)

初始条件：链式表已存在。

操作结果：返回L中数据元素的个数。

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

初始条件：单链表已存在，1≤i≤ListLength(L)。

操作结果：用e返回L中第i个结点的数据元素值。

（7）LocateElem(LinkList L,ElemType e)

初始条件：链式表已存在。

操作结果：用e返回L中第i个数据元素的值。

（8）PriorElem（LinkList L,ElemType cur\_e,ElemType & pre\_e）

初始条件：单链表L已存在。

操作结果：若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的

前驱，否则操作失败，pre\_e无定义。

（9）NextElem（LinkList L,ElemType cur\_e,ElemType & next\_e）

初始条件：单链表L已存在。

操作结果：若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它

的后继，否则操作失败，next\_e无定义。

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

初始条件：链式表L已存在且非空，1≤i≤ListLength(L)+1。

操作结果：在L的第i个结点之前插入新数据元素e的结点。

（11）ListDelete(LinkList & L,int i,ElemType \* e)

初始条件：链式表L已存在且非空，1≤i≤ListLength(L)。

操作结果：删除L第i个数据元素的结点，用e返回其结点数据元素的值。

（12）ListTraverse(LinkList L)

初始条件：链式表L已存在。

操作结果：依次对L的每个数据元素调用函数visit()。一旦调用失败，则操作失败。

（13）LoadFromFile(char \* filename,LinkList & L)

初始条件：文件夹中保存有链式表

操作结果：成功读取链式表L

（14）SaveToFile(char \* filename,LinkList L)

初始条件：已创建链式表L

操作结果：将链式表L保存在创建的文件夹中

## 2.3 系统实现

**基于链表的线性表结构定义：**

typedef struct LNode{ //顺序表（顺序结构）的定义

ElemType data;

struct LNode \*next;

}LNode, \*LinkList;

**函数分析:**

（1）InitList(LinkList & L)

创建空链式表L分三步：1.分配空间 2.将数据域初始化为零 3.将指针域初始化为0。

时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（2）DestroyList(LinkList & L)

销毁链式表分2步：1.将上一个链式表与下一个链式表连接 2.释放链式表

时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（3）ClearList (LinkList & L)

将链式表置空，先释放链式表的存储空间，再将上一个与下一个链式表连接，最后将自身指针域置空处理，由此达到重置为空表的目的。

时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（4）ListEmpty(LinkList &L)

判断链式表是否为空。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

（5）ListLength(LinkList & L)

遍历链表，利用计数器i统计链表元素个数。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

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

利用结点p遍历链式表L，再利用计数器j来判断是否到达所需位置i，如果到达，则返回该处元素，达到函数目的。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

（7）LocateElem(LinkList L,ElemType e)

利用p指针遍历链表，再运用compare函数进行比较，如果在遍历过程中找到元素e则返回此时计数器i，就是该元素在链表中的位置，由此实现函数功能。compare这里使用等于（=）操作

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

（8）PriorElem（LinkList L,ElemType cur\_e,ElemType & pre\_e）

参数为链表头指针L, 查找的元素cur\_e，用于返回其前驱的元素pre\_e的地址，Status 成功则返回OK，未找到返回ERROR，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。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

（9）NextElem（LinkList L,ElemType cur\_e,ElemType & next\_e）

参数为链表头指针L，Status 成功则返回OK，未找到返回ERROR，若cur\_e是L的数据元素，且不是第一个，则用next\_e返回它的后驱，否则操作失败，next\_e无定义。

时间复杂度T(n) = O(n)

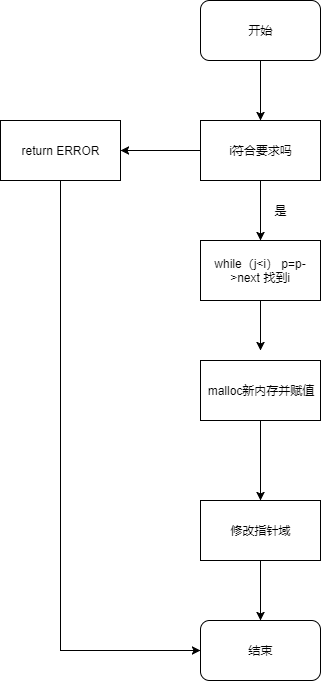
空间复杂度S(n) = O(1)

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

输入参数： 链表头指针地址&L, 插入位置i，插入元素e，在L的第i个结点之前插入新数据元素e的结点。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

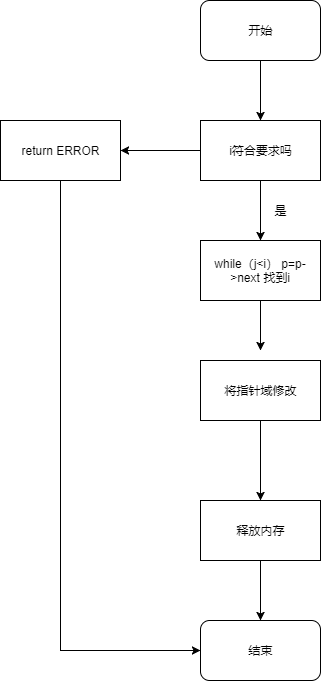


（11）ListDelete(LinkList & L,int i,ElemType \* e)

链表头指针L, 删除位置i，用于返回其值的元素e的地址，删除L第i个数据元素的结点，用e返回其结点数据元素的值。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)



（12）ListTraverse(LinkList L)

链表头指针L, 指向调用函数的函数指针visit，依次对L的每个数据元素调用函数visit()。一旦调用失败，则操作失败。visit()这里使用printf操作

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

（13）LoadFromFile(char \* filename,LinkList & L)读取链式表。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

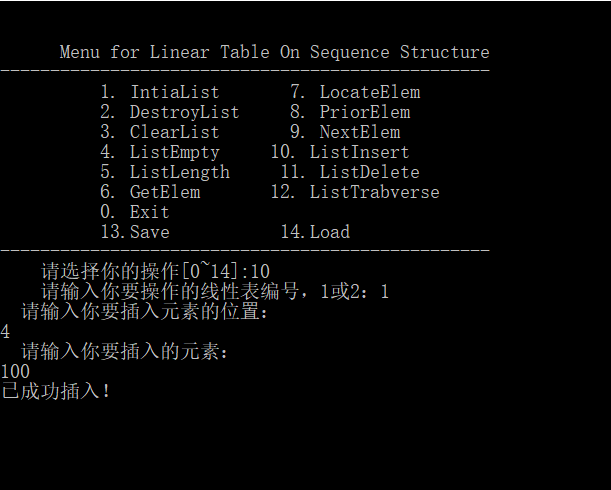
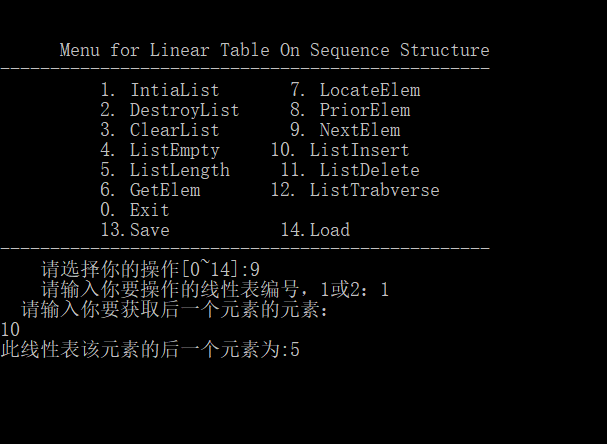
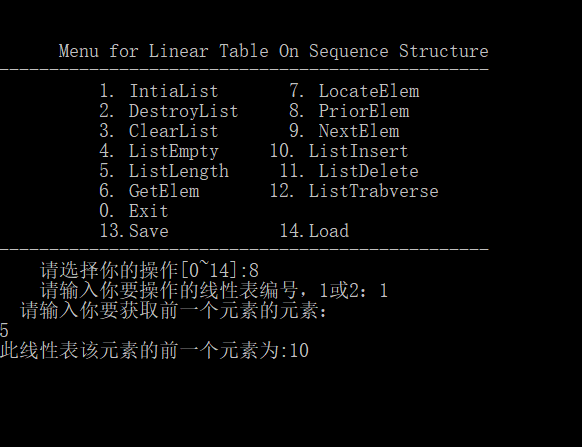
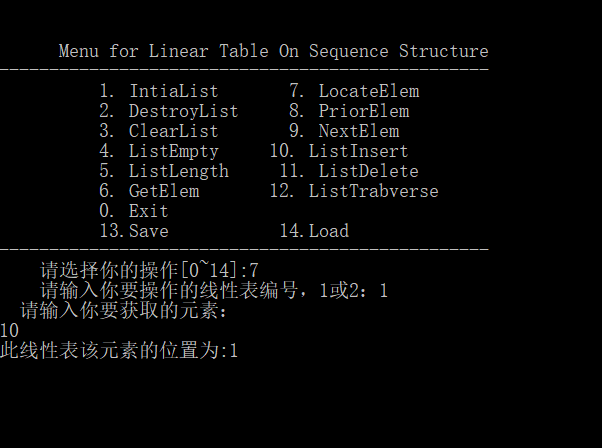
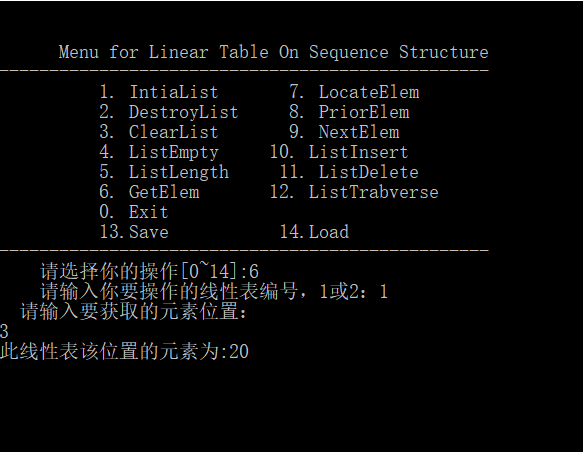
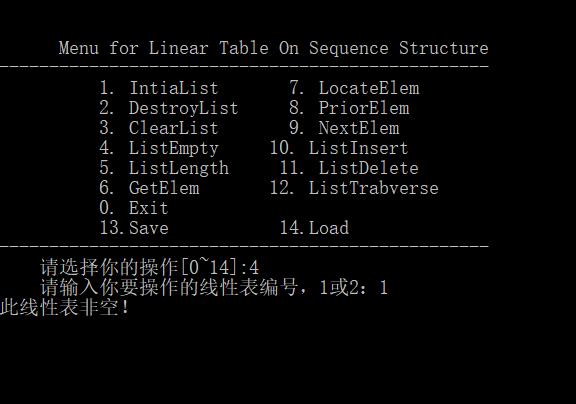
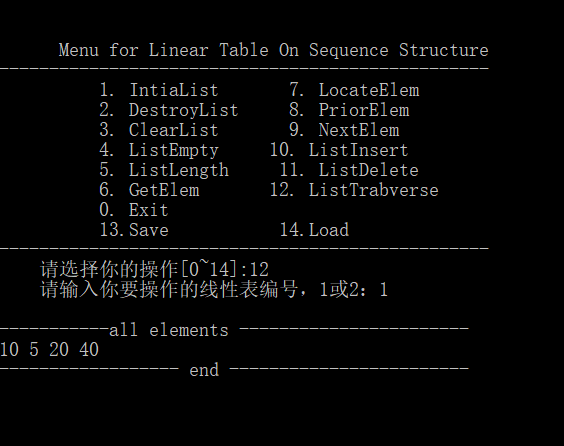
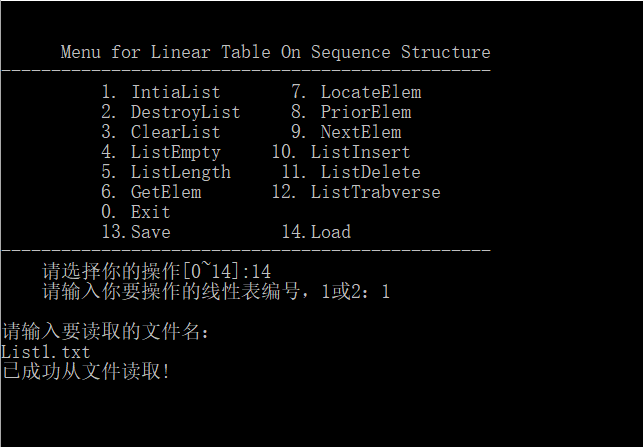
（14）SaveToFile(char \* filename,LinkList L)

保存链式表

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

## 2.4测试



## 2.5 实验小结

# 相对第一次上机感觉这次的内容要复杂一些，对链表的操作要比对数组的操作更具难度，但基本原理都是相通的，只需要对第一次实验的各个函数修改实现的物理方式，难度有所上升，但同时也能对链表再熟悉一下。这次上机还是获益匪浅的，学会用链式结构对线性表进行运算等操作，但自己的编程能力确实还有待加强，学得还不够深入使得每次上机都变得困难重重。以后要努力改进，争取做得更好。3 基于二叉链表的二叉树实现

## 3.1 问题描述

基于二叉链表实现二叉树的各种操作及常见运算。

## 3．1.2实验目的

通过实验达到⑴加深对二叉树的概念、基本运算的理解；⑵熟练掌握二叉树的逻辑结构与物理结构的关系；⑶以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。

## 3.2 系统设计

（1）Status InitBiTree(BiTree \* T);

初始条件：T不存在

操作结果：初始化T

（2）Status DestroyBiTree(BiTree \* T);

初始条件：T存在

操作结果：销毁树T

（3）void CreateBiTree(BiTree \* T);

初始条件：T存在

操作结果：以先序遍历序列创建整个树

（4）Status ClearBiTree(BiTree \* T);

初始条件：T存在

操作结果：清空T的所有节点

（5）Status BiTreeEmpty(BiTree T);

初始条件：二叉树T存在，

操作结果：若T为空二叉树，则返回TRUE,否则返回FALSE.

（6）Status BiTreeDepth(BiTree T);

初始条件：T存在

操作结果：返回二叉树T的深度

（7）BiTNode Root(BiTree T);

初始条件：T存在

操作结果：返回二叉树T的根

（8）Status Value(BiTree T, BiTree e);

初始条件：T存在且e为T的一个节点

操作结果：获得节点e的值

（9）Status Assign(BiTree T, BiTree e, ElemType value);

初始条件：T存在且e为T中的一个节点

操作结果：给节点e赋值为value

（10）ElemType Parent(BiTree T,ElemType e);

初始条件：T存在且e为T中一个节点。

操作结果：获得T中节点e的双亲结点

（11）ElemType LeftChild(BiTree T,ElemType e);

初始条件：T存在且e为T中一个节点。

操作结果：获得T中节点e的左孩子结点

（12）ElemType RightChild(BiTree T,ElemType e);

初始条件：T存在且e为T中一个节点。

操作结果：获得T中节点e的右孩子结点

（13）ElemType LeftSibling(BiTree T,ElemType e);

初始条件：T存在且e为T中一个节点。

操作结果：获得T中节点e的左兄弟结点

（14）ElemType RightSibling(BiTree T,ElemType e);

初始条件：T存在且e为T中一个节点。

操作结果：获得T中节点e的右兄弟结点

（15）Status InsertChild(BiTree T,BiTree p,int LR,BiTree c);

初始条件：T存在且p为T中的一个节点，LR为0或1

操作结果：将c插入T中p节点的相应位置

（16）Status DeleteChild(BiTree T,BiTree p,int LR);

初始条件：二叉树T存在且p为T内的一个节点

操作结果：删除子树

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

初始条件：二叉树T存在

操作结果：以先序遍历二叉树并用Visit访问每个节点

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

初始条件：二叉树T存在

操作结果：以中序遍历二叉树并用Visit访问每个节点

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

初始条件：二叉树T存在

操作结果：以后序遍历二叉树并用Visit访问每个节点

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

初始条件：二叉树T存在

操作结果：以层序遍历二叉树并用Visit访问每个节点

## 3.3 系统实现

**基于二叉链表的二叉树结构定义：**

typedef struct BiTNode{

ElemType data;

struct BiTNode \*lchild, \*rchild; //左右孩子指针

}BiTNode, \*BiTree;

**函数分析：**

（1）Status InitBiTree(BiTree \* T);

将根节点置空

时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（2）Status DestroyBiTree(BiTree \* T);

释放根节点对应的内存

时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

1. void CreateBiTree(BiTree \* T);

递归读取字符，如果读到$则将本节点置空，否则将本节点值设为此字符并继续递归下去。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. Status ClearBiTree(BiTree \* T);

递归对每个非空节点free掉其内存。

时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

1. Status BiTreeEmpty(BiTree T);

判断根节点是否为空，为空就返回TRUE，否则返回FALSE

时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

1. Status BiTreeDepth(BiTree T);

递归访问每个节点，每深一层则层数加1.

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. BiTNode Root(BiTree T);

返回根节点对应的结构体。

时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

1. Status Value(BiTree T, BiTree e);

返回节点e的值

时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

1. Status Assign(BiTree T, BiTree e, ElemType value);

对节点e赋值为value

时间复杂度T(n) = O(1)

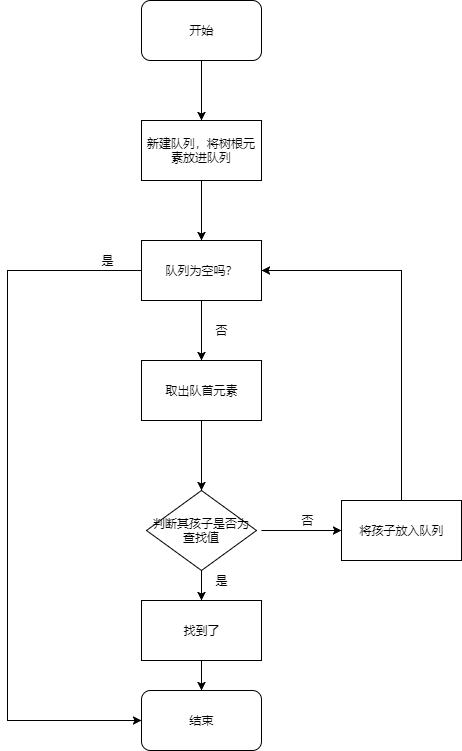
空间复杂度S(n) = O(1)

1. ElemType Parent(BiTree T,ElemType e);

返回节点e的双亲节点的数据。递归访问，如果下个节点为e则返回本节点数据

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)



1. ElemType LeftChild(BiTree T,ElemType e);

递归访问整个树，如果当前节点为且有左孩子的话就返回其左孩子。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. ElemType RightChild(BiTree T,ElemType e);

递归访问整个树，如果当前节点为且有右孩子的话就返回其右孩子。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. ElemType LeftSibling(BiTree T,ElemType e);

递归访问整个树，如果当前节点为且有左兄弟的话就返回其左兄弟。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. ElemType RightSibling(BiTree T,ElemType e);

递归访问整个树，如果当前节点为且有右兄弟的话就返回其右兄弟。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. Status InsertChild(BiTree T,BiTree p,int LR,BiTree c);

递归访问，在T中找到节点p，将p相应节点指针赋值为c

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. Status DeleteChild(BiTree T,BiTree p,int LR);

递归访问，在T中找到节点p，将p相应节点指针赋值为空

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

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

前序遍历递归访问，对每个节点用Visit输出

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

1. Status InOrderTrabverse(BiTree T,Status (\* Visit)(ElemType e));

中序遍历递归访问，对每个节点用Visit输出

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

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

后序遍历递归访问，对每个节点用Visit输出

时间复杂度T(n) = O(n)

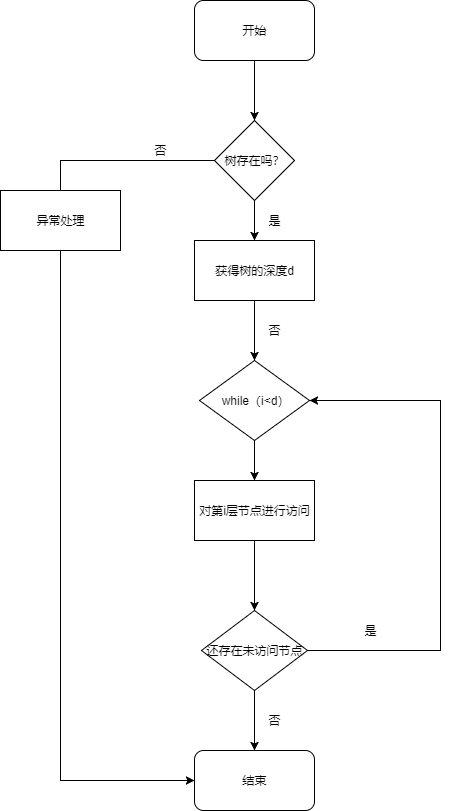
空间复杂度S(n) = O(1)

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

层序遍历递归访问，对每个节点用Visit输出

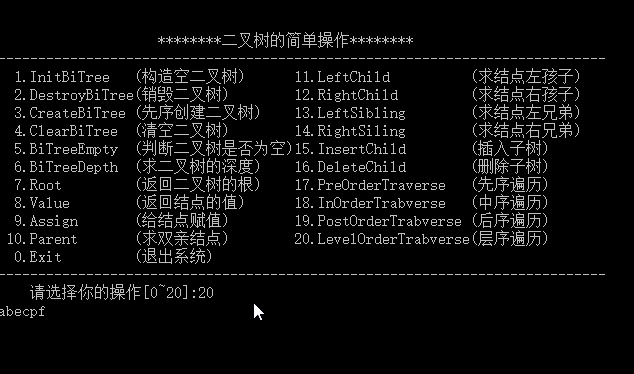
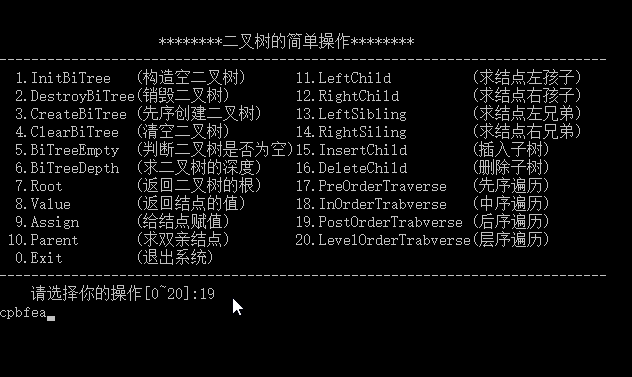
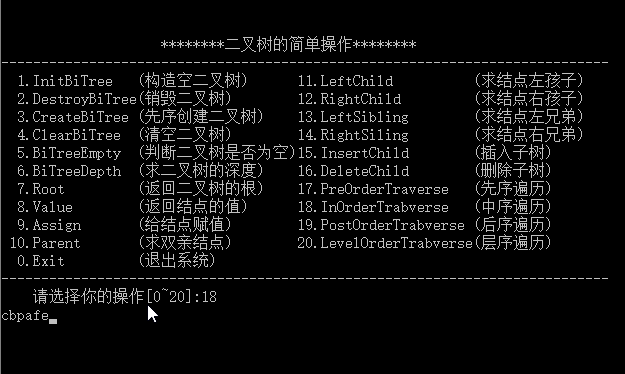
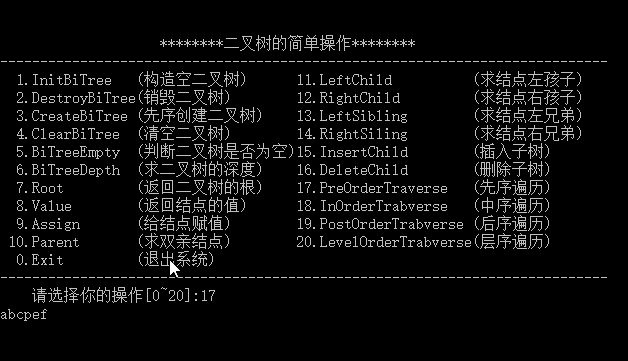
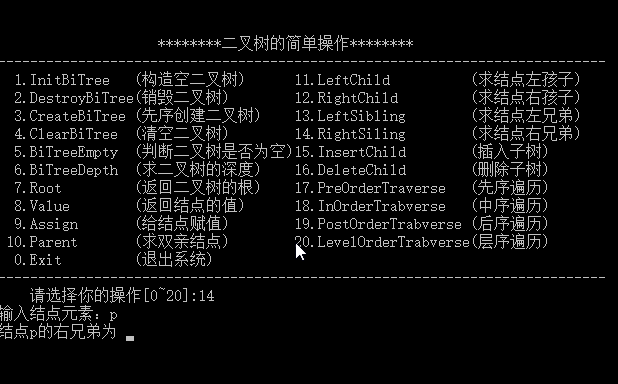
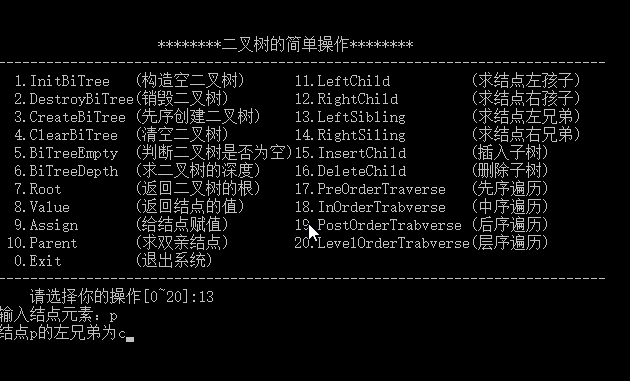
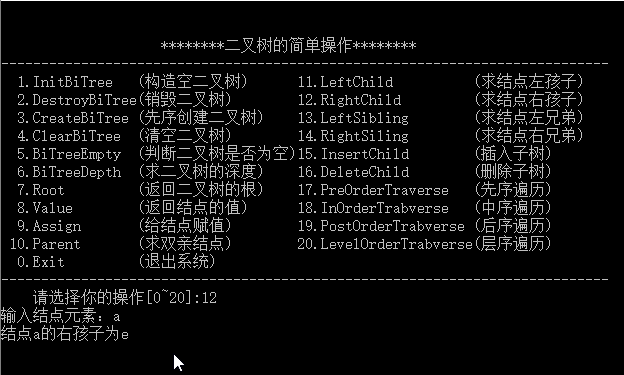
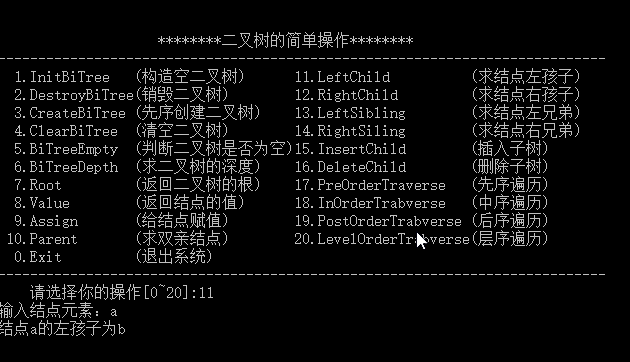
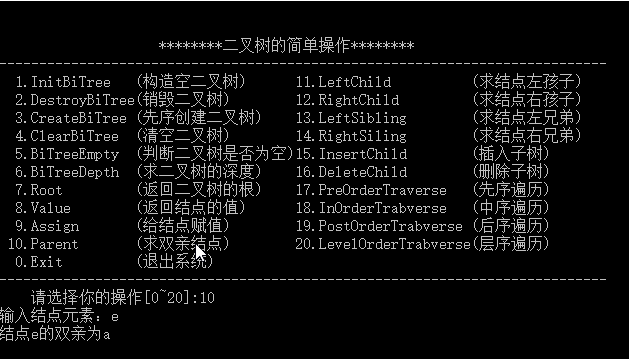
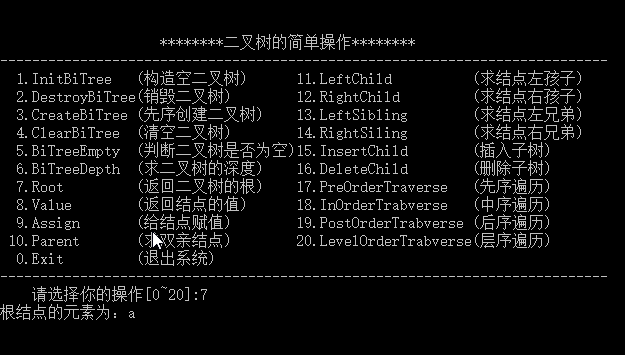
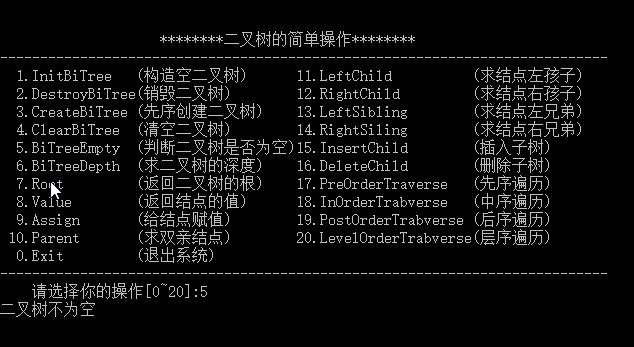
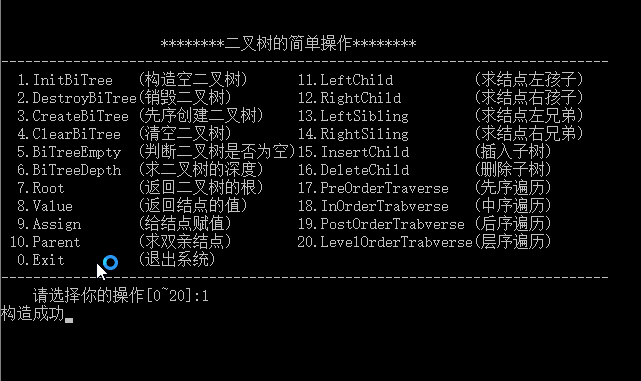
时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)



## 3.4测试

创建二叉树、输入abc##d##ef###、判断是否为空、求深度、返回根、给结点d赋值p、求e的双亲结点、求a左孩子、求a右孩子、求p左兄弟、求p右兄弟、先序遍历、中序遍历、后序遍历、层序遍历、清空二叉树、销毁二叉树、退出



## 3.5 实验小结

这次实验感觉较之前几次的都要难一些，所以花的时间也比较多，中间写先序、中序、后序遍历算法比较纠结，问过同学后发现用递归算法很简单，之后很快就搞定了。这次实验学会了如何创建二叉树以及对二叉树的各类简单操作，对递归的思想更加熟悉，对指针的运用更加熟练，收获很大，来日方长，在编程的路上还会遇到很多困难，都要努力的寻找办法去解决，这是成为一个好的程序员的必经之路！

# 4 基于邻接表的图实现

## 4.1 问题描述

基于邻接表实现图的各种操作及常见运算。

## 4.1.1实验目的

通过实验达到⑴加深对图的概念、基本运算的理解；⑵熟练掌握图的逻辑结构与物理结构的关系；⑶以邻接表作为物理结构，熟练掌握图基本运算的实现。

## 4.2 系统设计

⑴创建图： CreateGraph(&G,V,VR)；

初始条件：V是图的顶点集，VR是图的关系集；

操作结果：按V和VR的定义构造图G。

⑵销毁图：DestroyGraph(T)；

初始条件图G已存在；

操作结果是销毁图G。

⑶查找顶点： LocateVex(G,u)；

初始条件是图G存在，u和G中的顶点具有相同特征；

操作结果是若u在图G中存在，返回顶点u的位置信息，否则返回其它信息。

⑷获得顶点值： GetVex (G,v)；

初始条件是图G存在，v是G中的某个顶点；

操作结果是返回v的值。

⑸顶点赋值： PutVex (G,v,value)；

初始条件是图G存在，v是G中的某个顶点；

操作结果是对v赋值value。

⑹获得第一邻接点：FirstAdjVex(&G, v)；

初始条件是图G存在，v是G的一个顶点；

操作结果是返回v的第一个邻接顶点，如果v没有邻接顶点，返回“空”。

⑺获得下一邻接点： NextAdjVex(&G, v, w)；

初始条件是图G存在，v是G的一个顶点,w是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，如果w是最后一个邻接顶点，返回“空”。

⑻插入顶点： 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。

⑼删除顶点： DeleteVex(&G,v)；

初始条件是图G存在，v是G的一个顶点；

操作结果是在图G中删除顶点v和与v相关的弧。

⑽插入弧： InsertArc(&G,v,w)；

初始条件是图G存在，v、w是G的顶点；

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>。

⑾删除弧： DeleteArc(&G,v,w)；

初始条件是图G存在，v、w是G的顶点；

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>。

⑿深度优先搜索遍历： 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。

⒀广深度优先搜索遍历： BFSTraverse(G,visit())；

初始条件是图G存在；

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

## 4.3 系统实现

**邻接表结构定义：**

typedef struct ArcNode{

int adjvex; //该弧所指向的顶点位置

struct ArcNode \* nextarc; //指向下一条弧的指针

InfoType \*info; //该弧相关信息的指针

}ArcNode;

typedef struct VNode{

ArcNode \* firstarc; //指向第一条依附该顶点的弧的指针

VertexType data; //顶点信息

}VNode, AdjList[MAX\_VERTEX\_NUM];

typedef struct{

AdjList vertices;

int vexnum, arcnum; //图的当前顶点数和弧数

int kind; //图的种类标志

}ALGraph;

kind 取0、1、2、3，分别代表有向图、有向网、无向图、无向网。

**函数分析：**

1. CreateCraph(&G,V,VR)

函数名称：CreateGraph

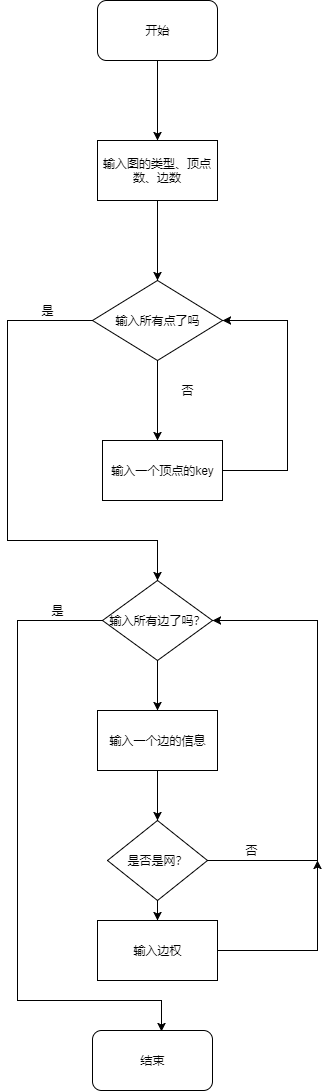
输入参数： 图G地址

返回值： Status 成功则返回OK

函数功能： 构造图G。

时间复杂度T(n) = O(n+e)

空间复杂度S(n) = O(n+e)



（2）DestroyCraph(&G)

函数名称：DestroyGraph

输入参数： 图G地址

返回值： Status 成功则返回OK

函数功能： 销毁图G。

时间复杂度T(n) = O(n+e)

空间复杂度S(n) = O(1)

（3）LocateVex(G,u)

函数名称：LocateVex

输入参数： 图G,所要查找的顶点的名称或值u

返回值： int

函数功能： 若u在图G中存在，返回顶点u的位置序号，否则返回-1。

时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

（4）FirstAdjVex(&G, v)

函数名称：FirstAdjVex

输入参数： 图G, G中某个顶点v

返回值： int

函数功能： 返回v的第一个邻接顶点的序号。若顶点在G中没有邻接顶点,则返回-1。

时间复杂度T(n) = O(n+e)

空间复杂度S(n) = O(1)

（5）NextAdjVex(&G, v, w)

函数名称：NextAdjVex

输入参数： 图G, G中某个顶点v, v的邻接顶点w

返回值： int

函数功能： 返回v的(相对于w的)下一个邻接顶点的序号，若w是v的最后一个邻接点,则返回-1。

时间复杂度T(n) = O(n+e)

空间复杂度S(n) = O(1)

（6）InsertVex(&G,v)

函数名称：InsertVex

输入参数： 图G地址, 新顶点v

返回值： Status

函数功能： 在图G中增加新顶点v。

时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（7）DeleteVex(&G,v)

函数名称：DeleteVex

输入参数： 图G地址, v是G中某个顶点

返回值： Status 成功返回OK

函数功能： 删除G中顶点v及其相关的弧。

时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（8）InsertArc(&G,v,w)

函数名称：InsertArc

输入参数： 图G地址, G中两个顶点v、w

返回值： Status 成功返回OK；

函数功能：在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>。

时间复杂度T(n) = O(n+e)

空间复杂度S(n) = O(1)

（9）DeleteArc(&G,v,w)

函数名称： DeleteArc

输入参数： 图G地址, G中两个顶点v、w

返回值： Status 成功返回OK；

函数功能： 在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>。

时间复杂度T(n) = O(n+e)

空间复杂度S(n) = O(1)

（10）DFSTraverse(G,visit())

函数名称： DFSTraverse

输入参数： 图G, 调用函数visit

返回值： Status 成功返回OK；

函数功能： 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。

时间复杂度T(n) = O(n+e)

空间复杂度S(n) = O(1)

（11）BFSTraverse(G,visit())

函数名称： BFSTraverse

输入参数： 图G, 调用函数visit

返回值： Status 成功返回OK；

函数功能： 按广度优先非递归遍历图G。

时间复杂度T(n) = O(n+e)

空间复杂度S(n) = O(1)

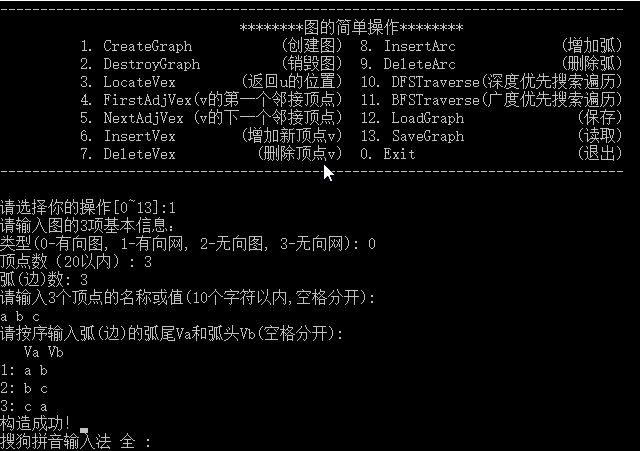
## 4.4测试

创建3个顶点abc的图

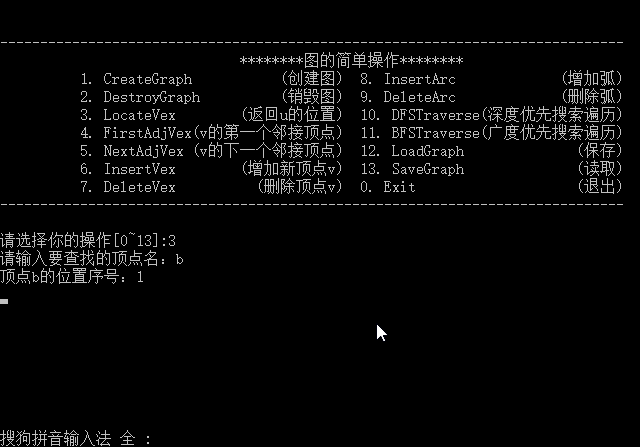
c

a

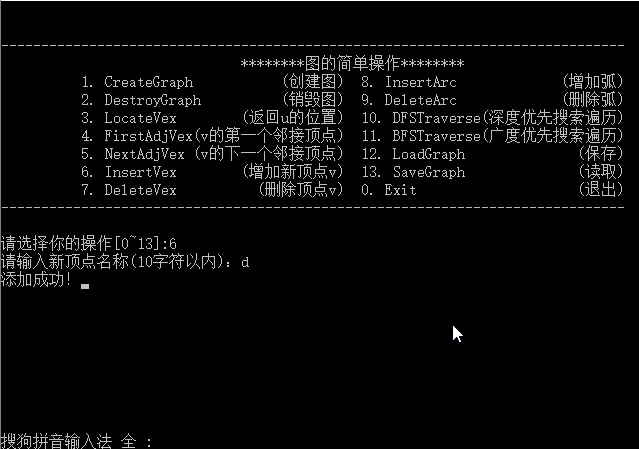
b



返回b的位置1



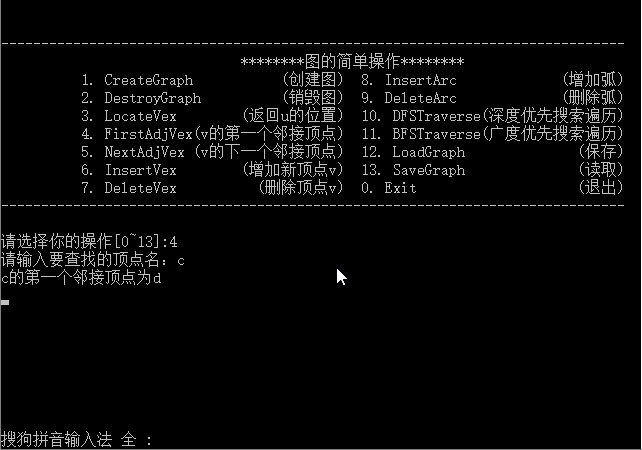
增加新顶点d



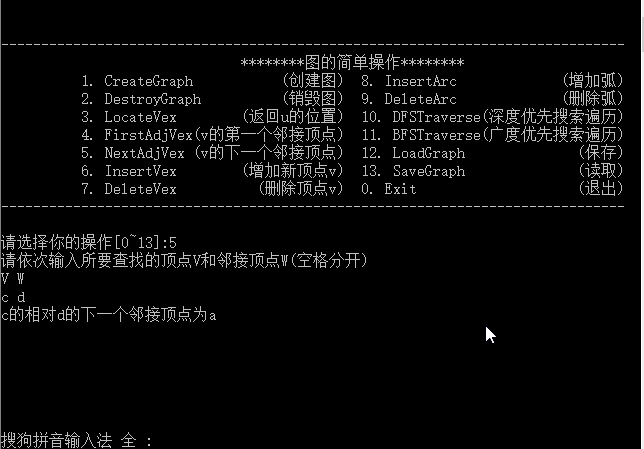
增加弧c d



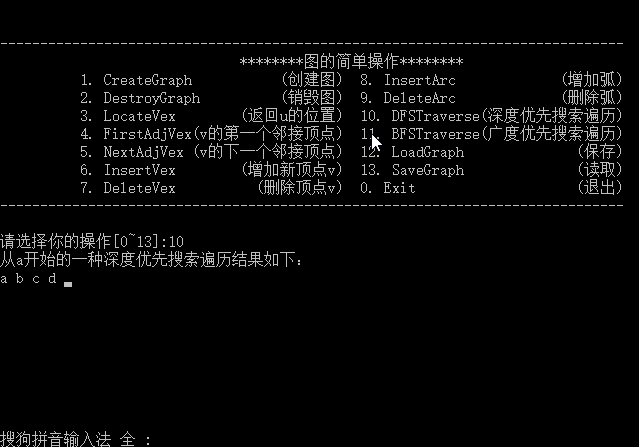
返回c的第一个邻接顶点d



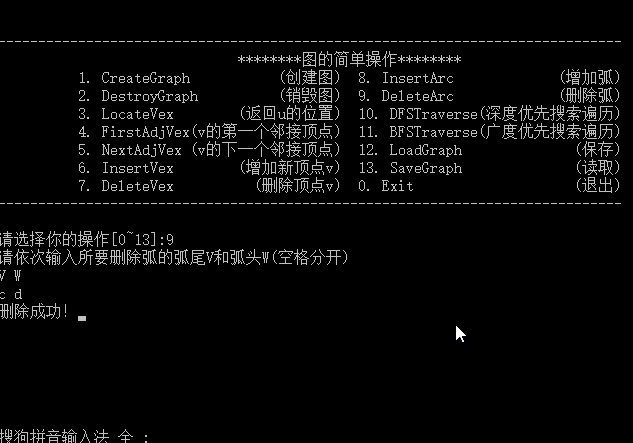
返回c相对d下一个邻接顶点a



遍历



删除弧c d



删除顶点d



## 4.5 实验小结

这次实验基本能够熟练掌握关于图的各项基本操作，个人认为这次实验核心在于用邻接表储存图的结构，以及深度、广度优先遍历图的方法的掌握。实验过程中遇到些小问题，都通过问同学一一解决了。数据结构的实验到此差不多快要结束了，希望自己能通过寒假的课设更加系统的增强自己在编程这方面的能力，将数据结构课上所学加以实践，编程之路还很漫长，还需更加努力！

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

[2] [Larry Nyhoff](http://www.calvin.edu/~nyhl/index.html). [ADTs, Data Structures, and Problem Solving with C++.](http://vig.prenhall.com/catalog/academic/product/0,1144,0131409093,00.html)Second Edition, [Calvin College](http://cs.calvin.edu/), 2005

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

指导教师评定意见

一、对实验报告的评语

|  |
| --- |
|  |

二、对实验报告评分

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 程序内容  (36.8分) | 程序规范  (9.2分) | 报告内容  (36.8分) | 报告规范  (9.2分) | 考勤  （8分） | 逾期扣分 | 合 计  (100分) |
| 得分 |  |  |  |  |  |  |  |

# 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>

#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;

status IntiaList(SqList & L);

status DestroyList(SqList & L);

status ClearList(SqList &L);

status ListEmpty(SqList L);

int ListLength(SqList L);

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

status LocateElem(SqList L,ElemType e); //简化过

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

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

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

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

status ListTrabverse(SqList L); //简化过

status LoadFromFile(char \* filename,SqList & L);

status SaveToFile(char \* filename,SqList L);

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

int main(){

SqList L1,L2; int op=1;

ElemType e,ans;

int i;

int option;

char fileName[20];

while(op){

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

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

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

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

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

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

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

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

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

printf(" 0. Exit\n");

printf(" 13.Save 14.Load\n");

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

printf(" 请选择你的操作[0~14]:");

scanf("%d",&op);

printf(" 请输入你要操作的线性表编号，1或2：");

scanf("%d",&option);

if(option==1){

switch(op){

case 1:

if(IntiaList(L1)==OK) printf("线性表1创建成功！\n");

else printf("线性表1创建失败！\n");

getchar();getchar();

break;

case 2:

if(DestroyList(L1)==OK) printf("线性表1删除成功！\n");

else printf("线性表1删除失败！\n");

getchar();getchar();

break;

case 3:

if(ClearList(L1)==OK) printf("线性表1清空成功！\n");

else printf("线性表1清空失败！\n");

getchar();getchar();

break;

case 4:

if(ListEmpty(L1)==TRUE) printf("此线性表为空！\n");

else printf("此线性表非空！\n");

getchar();getchar();

break;

case 5:

int len;

if((len=ListLength(L1))>=0) printf("此线性表为长度为%d！\n",len);

else printf("无此线性表！\n");

getchar();getchar();

break;

case 6:

int i; ElemType e;

printf(" 请输入要获取的元素位置：\n");

scanf("%d",&i);

if(GetElem(L1,i,e)) printf("此线性表该位置的元素为:%d\n",e);

else printf("输入有误！\n");

getchar();getchar();

break;

case 7:

int ans;

printf(" 请输入你要获取的元素：\n");

scanf("%d",&e);

if((ans=LocateElem(L1,e))) printf("此线性表该元素的位置为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 8:

printf(" 请输入你要获取前一个元素的元素：\n");

scanf("%d",&e);

if((PriorElem(L1,e,ans))) printf("此线性表该元素的前一个元素为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 9:

printf(" 请输入你要获取后一个元素的元素：\n");

scanf("%d",&e);

if((NextElem(L1,e,ans))) printf("此线性表该元素的后一个元素为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 10:

printf(" 请输入你要插入元素的位置：\n");

scanf("%d",&i);

if(!L1.elem) printf("操作不合法");

if(i<1||i>L1.length+1) printf("请输入合法的数据，在0~%d之间",L1.length+1);

else {

printf(" 请输入你要插入的元素：\n");

scanf("%d",&e);

if(ListInsert(L1,i,e)) printf("已成功插入！\n");

else printf("输入有误！\n");

}

getchar();getchar();

break;

case 11:

printf(" 请输入你要删除元素的位置：\n");

scanf("%d",&i);

if(ListDelete(L1,i,e)) printf("已成功删除！元素为：%d\n",e);

else printf("输入有误！\n");

getchar();getchar();

break;

case 12:

if(!ListTrabverse(L1)) printf("线性表是空表！\n");

getchar();getchar();

break;

case 13:

printf("\n请输入要保存的文件名：\n");

scanf("%s",fileName);

if(SaveToFile(fileName,L1)) printf("已成功保存至文件!");

else printf("保存失败！");

getchar();getchar();

break;

case 14:

printf("\n请输入要读取的文件名：\n");

scanf("%s",fileName);

if(LoadFromFile(fileName,L1)) printf("已成功从文件读取!");

else printf("读取失败！");

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of if

else if(option==2){

switch(op){

case 1:

if(IntiaList(L2)==OK) printf("线性表2创建成功！\n");

else printf("线性表2创建失败！\n");

getchar();getchar();

break;

case 2:

if(DestroyList(L2)==OK) printf("线性表2删除成功！\n");

else printf("线性表2删除失败！\n");

getchar();getchar();

break;

case 3:

if(ClearList(L2)==OK) printf("线性表2清空成功！\n");

else printf("线性表2清空失败！\n");

getchar();getchar();

break;

case 4:

if(ListEmpty(L2)==TRUE) printf("此线性表为空！\n");

else printf("此线性表非空！\n");

getchar();getchar();

break;

case 5:

int len;

if((len=ListLength(L2))>=0) printf("此线性表为长度为%d！\n",len);

else printf("无此线性表！\n");

getchar();getchar();

break;

case 6:

int i; ElemType e;

printf(" 请输入要获取的元素位置：\n");

scanf("%d",&i);

if(GetElem(L2,i,e)) printf("此线性表该位置的元素为:%d\n",e);

else printf("输入有误！\n");

getchar();getchar();

break;

case 7:

int ans;

printf(" 请输入你要获取的元素：\n");

scanf("%d",&e);

if((ans=LocateElem(L2,e))) printf("此线性表该元素的位置为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 8:

printf(" 请输入你要获取前一个元素的元素：\n");

scanf("%d",&e);

if(PriorElem(L2,e,ans)) printf("此线性表该元素的前一个元素为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 9:

printf(" 请输入你要获取后一个元素的元素：\n");

scanf("%d",&e);

if((NextElem(L2,e,ans))) printf("此线性表该元素的后一个元素为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 10:

printf(" 请输入你要插入元素的位置：\n");

scanf("%d",&i);

if(!L1.elem) printf("操作不合法");

if(i<1||i>L1.length+1) printf("请输入合法的数据，在0~%d之间",L1.length);

else{

printf(" 请输入你要插入的元素：\n");

scanf("%d",&e);

if(ListInsert(L2,i,e)) printf("已成功插入！\n");

else printf("输入有误！\n");

}

getchar();getchar();

break;

case 11:

printf(" 请输入你要删除元素的位置：\n");

scanf("%d",&i);

if(ListInsert(L2,i,e)) printf("已成功删除！元素为：%d\n",e);

else printf("输入有误！\n");

getchar();getchar();

break;

case 12:

if(!ListTrabverse(L2)) printf("线性表是空表！\n");

getchar();getchar();

break;

case 13:

printf("\n请输入要保存的文件名：\n");

scanf("%s",fileName);

if(SaveToFile(fileName,L2)) printf("已成功保存至文件!");

else printf("保存失败！");

getchar();getchar();

break;

case 14:

printf("\n请输入要读取的文件名：\n");

scanf("%s",fileName);

if(LoadFromFile(fileName,L2)) printf("已成功从文件读取!");

else printf("读取失败！");

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of if

else {

printf("输入有误！");

}

}//end of while

printf("欢迎下次再使用本系统！\n");

return 0;

}//end of main()

status IntiaList(SqList & L){

//构造一个空的线性表L

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

if(!L.elem) exit(OVERFLOW); //存储分配失败

L.length=0; //空表长度为0

L.listsize=LIST\_INIT\_SIZE; //初始存储容量

return OK;

}

status DestroyList(SqList & L){

if(!L.elem) return OK; //L不存在

free(L.elem); //释放内存

L.length=0; //空表长度置零

L.listsize=0; //初始容量置0

return OK;

}

status ClearList(SqList &L){

if(!L.elem) exit(ERROR);

L.length=0; //空表长度为0

L.listsize=LIST\_INIT\_SIZE; //初始存储容量

return OK;

}

status ListEmpty(SqList L){

if(!L.elem) exit(ERROR); //异常处理

if(L.length==0) return TRUE; //空表

else return FALSE; //非空

}

int ListLength(SqList L){

if(!L.elem) exit(ERROR); //异常处理

return L.length;

}

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

if(!L.elem) exit(ERROR); //异常处理

if(i>L.length) exit(OVERFLOW);

e=L.elem[i-1];

return OK;

}

status LocateElem(SqList L,ElemType e){

int i=1; //i的初值为第一个元素的位序

ElemType \*p=L.elem; //p的初值为第1个元素的存储位置

if(p==NULL) exit(ERROR);

while(i<=L.length&&((\*p++)!=e)) ++i;

if(i<=L.length) return i;

else return 0;

}

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

if(!L.elem) exit(ERROR);

for(int i=1;i<L.length;i++){ //从第二个开始遍历

if(L.elem[i]==cur){

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

return OK; //找到了

}

}

return ERROR; //未找到

}

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

if(!L.elem) exit(ERROR);

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

if(L.elem[i]==cur){

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

return OK; //找到了

}

}

return ERROR; //未找到

}

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

//在顺序线性表L中第i个位置之前插入新的元素e

//i的合法位置为1<=i<=ListLength\_Sq(L)+1

if(!L.elem) exit(ERROR);

if(i<1||i>L.length+1) return ERROR; //i值不合法

if(L.length>=L.listsize){ //当前存储空间已满，增加分配

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

if(!newbase) exit(OVERFLOW); //存储分配失败

L.elem=newbase; //新基址

L.listsize+=LISTINCREMENT; //增加存储容量

}

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

for(ElemType \*p=&(L.elem[L.length-1]);p>=q;--p) \*(p+1) =\*p; //插入位置及之后的元素右移

\*q =e; //插入e

++L.length; //表长增1

return OK;

}

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

if(!L.elem) exit(ERROR);

//在顺序线性表L中删除第i个元素，并用e返回其值

//i的合法值为1<=i<=ListLength\_Sq(L)

if((i<1)||(i>L.length)) return ERROR; //i值不合法

ElemType \*p=&(L.elem[i-1]); //p为被删除元素的位置

e=\*p; //被删除元素的值赋给e

ElemType \* q=L.elem+L.length-1; //表尾元素的位置

for(++p;p<=q;++p) \*(p-1)=\*p; //被删除元素之后的元素左移

--L.length; //表长减1

return OK;

}

status ListTrabverse(SqList L){

int i;

if(!L.elem) exit(ERROR);

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

for(i=0;i<L.length;i++) printf("%d ",L.elem[i]);

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

return L.length;

}

status LoadFromFile(char \* filename,SqList & L){

IntiaList(L);

FILE \*fp;

if ((fp=fopen(filename,"r"))==NULL){

printf("File open error\n ");

return ERROR;

}

while(fread(&L.elem[L.length],sizeof(ElemType),1,fp))

L.length++;

fclose(fp);

return OK;

}

status SaveToFile(char \* filename,SqList L){

FILE \*fp;

//写文件的方法

if ((fp=fopen(filename,"w"))==NULL){

printf("File open error\n ");

return ERROR;

}

fwrite(L.elem,sizeof(ElemType),L.length,fp);

fclose(fp);

return OK;

}

# 附录B 基于链式存储结构线性表实现的源程序

/\* Linear Table On Linked Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#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 LNode{ //顺序表（顺序结构）的定义

ElemType data;

struct LNode \*next;

}LNode, \*LinkList;

status IntiaList(LinkList & L);

status DestroyList(LinkList & L);

status ClearList(LinkList &L);

status ListEmpty(LinkList L);

int ListLength(LinkList L);

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

status LocateElem(LinkList L,ElemType e); //简化过

status PriorElem(LinkList L,ElemType cur,ElemType & pre\_e);

status NextElem(LinkList L,ElemType cur,ElemType & next\_e);

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

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

status ListTrabverse(LinkList L); //简化过

status LoadFromFile(char \* filename,LinkList & L);

status SaveToFile(char \* filename,LinkList L);

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

int main(){

LinkList L1,L2; int op=1;

ElemType e,ans;

int i;

int option;

char fileName[20];

while(op){

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

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

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

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

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

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

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

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

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

printf(" 0. Exit\n");

printf(" 13.Save 14.Load\n");

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

printf(" 请选择你的操作[0~14]:");

scanf("%d",&op);

printf(" 请输入你要操作的线性表编号，1或2：");

scanf("%d",&option);

if(option==1){

switch(op){

case 1:

if(IntiaList(L1)==OK) printf("线性表1创建成功！\n");

else printf("线性表1创建失败！\n");

getchar();getchar();

break;

case 2:

if(DestroyList(L1)==OK) printf("线性表1删除成功！\n");

else printf("线性表1删除失败！\n");

getchar();getchar();

break;

case 3:

if(ClearList(L1)==OK) printf("线性表1清空成功！\n");

else printf("线性表1清空失败！\n");

getchar();getchar();

break;

case 4:

if(ListEmpty(L1)==TRUE) printf("此线性表为空！\n");

else printf("此线性表非空！\n");

getchar();getchar();

break;

case 5:

int len;

if((len=ListLength(L1))>=0) printf("此线性表为长度为%d！\n",len);

else printf("无此线性表！\n");

getchar();getchar();

break;

case 6:

int i; ElemType e;

printf(" 请输入要获取的元素位置：\n");

scanf("%d",&i);

if(GetElem(L1,i,e)) printf("此线性表该位置的元素为:%d\n",e);

else printf("输入有误！\n");

getchar();getchar();

break;

case 7:

int ans;

printf(" 请输入你要获取的元素：\n");

scanf("%d",&e);

if((ans=LocateElem(L1,e))) printf("此线性表该元素的位置为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 8:

printf(" 请输入你要获取前一个元素的元素：\n");

scanf("%d",&e);

if((PriorElem(L1,e,ans))) printf("此线性表该元素的前一个元素为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 9:

printf(" 请输入你要获取后一个元素的元素：\n");

scanf("%d",&e);

if(NextElem(L1,e,ans)) printf("此线性表该元素的后一个元素为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 10:

printf(" 请输入你要插入元素的位置：\n");

scanf("%d",&i);

if(i<1||i>ListLength(L1)+1) printf("请输入合法的数据，在0~%d之间",ListLength(L1)+1);

else {

printf(" 请输入你要插入的元素：\n");

scanf("%d",&e);

if(ListInsert(L1,i,e)) printf("已成功插入！\n");

else printf("输入有误！\n");

}

getchar();getchar();

break;

case 11:

printf(" 请输入你要删除元素的位置：\n");

scanf("%d",&i);

if(ListDelete(L1,i,e)) printf("已成功删除！元素为：%d\n",e);

else printf("输入有误！\n");

getchar();getchar();

break;

case 12:

if(!ListTrabverse(L1)) printf("线性表是空表！\n");

getchar();getchar();

break;

case 13:

printf("\n请输入要保存的文件名：\n");

scanf("%s",fileName);

if(SaveToFile(fileName,L1)) printf("已成功保存至文件!");

else printf("保存失败！");

getchar();getchar();

break;

case 14:

printf("\n请输入要读取的文件名：\n");

scanf("%s",fileName);

if(LoadFromFile(fileName,L1)) printf("已成功从文件读取!");

else printf("读取失败！");

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of if

else if(option==2){

switch(op){

case 1:

if(IntiaList(L2)==OK) printf("线性表2创建成功！\n");

else printf("线性表2创建失败！\n");

getchar();getchar();

break;

case 2:

if(DestroyList(L2)==OK) printf("线性表2删除成功！\n");

else printf("线性表2删除失败！\n");

getchar();getchar();

break;

case 3:

if(ClearList(L2)==OK) printf("线性表2清空成功！\n");

else printf("线性表2清空失败！\n");

getchar();getchar();

break;

case 4:

if(ListEmpty(L2)==TRUE) printf("此线性表为空！\n");

else printf("此线性表非空！\n");

getchar();getchar();

break;

case 5:

int len;

if((len=ListLength(L2))>=0) printf("此线性表为长度为%d！\n",len);

else printf("无此线性表！\n");

getchar();getchar();

break;

case 6:

int i; ElemType e;

printf(" 请输入要获取的元素位置：\n");

scanf("%d",&i);

if(GetElem(L2,i,e)) printf("此线性表该位置的元素为:%d\n",e);

else printf("输入有误！\n");

getchar();getchar();

break;

case 7:

int ans;

printf(" 请输入你要获取的元素：\n");

scanf("%d",&e);

if((ans=LocateElem(L2,e))) printf("此线性表该元素的位置为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 8:

printf(" 请输入你要获取前一个元素的元素：\n");

scanf("%d",&e);

if(PriorElem(L2,e,ans)) printf("此线性表该元素的前一个元素为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 9:

printf(" 请输入你要获取后一个元素的元素：\n");

scanf("%d",&e);

if((NextElem(L2,e,ans))) printf("此线性表该元素的后一个元素为:%d\n",ans);

else printf("没有此元素！\n");

getchar();getchar();

break;

case 10:

printf(" 请输入你要插入元素的位置：\n");

scanf("%d",&i);

if(i<1||i>ListLength(L2)+1) printf("请输入合法的数据，在0~%d之间",ListLength(L2));

else{

printf(" 请输入你要插入的元素：\n");

scanf("%d",&e);

if(ListInsert(L2,i,e)) printf("已成功插入！\n");

else printf("输入有误！\n");

}

getchar();getchar();

break;

case 11:

printf(" 请输入你要删除元素的位置：\n");

scanf("%d",&i);

if(ListDelete(L2,i,e)) printf("已成功删除！元素为：%d\n",e);

else printf("输入有误！\n");

getchar();getchar();

break;

case 12:

if(!ListTrabverse(L2)) printf("线性表是空表！\n");

getchar();getchar();

break;

case 13:

printf("\n请输入要保存的文件名：\n");

scanf("%s",fileName);

if(SaveToFile(fileName,L2)) printf("已成功保存至文件!");

else printf("保存失败！");

getchar();getchar();

break;

case 14:

printf("\n请输入要读取的文件名：\n");

scanf("%s",fileName);

if(LoadFromFile(fileName,L2)) printf("已成功从文件读取!");

else printf("读取失败！");

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of if

else {

printf("输入有误！");

}

}//end of while

printf("欢迎下次再使用本系统！\n");

return 0;

}//end of main()

status IntiaList(LinkList & L){

//构造一个空的线性表L

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

if(!L) exit(OVERFLOW); //存储分配失败

L->data=0; //第一个元素置为0

L->next=NULL; //指针置空

return OK;

}

status DestroyList(LinkList & L){

if(L==NULL) return OK; //L不存在

LinkList p,q;

p=L;

while(p!=NULL){//释放内存

q=p->next;

free(p);

p=q;

}

L=NULL;//第头节点置空

return OK;

}

status ClearList(LinkList &L){

if(L->next==NULL) return OK; //L不存在元素节点

LinkList p,q;

p=L->next;

while(p!=NULL){//释放内存

q=p->next;

free(p);

p=q;

}

L->next=NULL;//第一个节点置空

return OK;

}

status ListEmpty(LinkList L){

if(L->next) return false; //存在元素节点

else return true; //不存在元素节点

}

int ListLength(LinkList L){

if(!L) exit(ERROR); //异常处理

int i=0;

LinkList p=L->next;//p指向第一个元素节点

while(p){

i++;

p=p->next;

}

return i;

}

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

int j=1;

LinkList p;

p=L->next;

while(j<i&&p){//依次比较，寻找插入位置

j++;

p=p->next;

}

if(j<i||!p) return ERROR;//异常处理

e=p->data;

return OK;

}

status LocateElem(LinkList L,ElemType e){

int i=0;

LinkList p=L->next;

while(p){

i++;//下标递增

if(p->data==e){//找到元素

return i;

}

p=p->next;

}

return 0;

}

status PriorElem(LinkList L,ElemType cur,ElemType & pre\_e){

LinkList p=L->next;

while(p->next!=NULL&&p->next->data!=cur){ //依次寻找

p=p->next;

}

if(p->next==NULL) return ERROR; //异常处理

pre\_e=p->data; //找到了

return OK;

}

status NextElem(LinkList L,ElemType cur,ElemType & next\_e){

LinkList p=L->next;

while(p->next!=NULL&&p->data!=cur){

p=p->next;

}

if(p->next==NULL) return ERROR; //未找到，异常处理

next\_e=p->next->data;

return OK;

}

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

LinkList p,newone;

int j=1;

p=L;

while(j<i&&p){ //依次寻找

j++;

p=p->next;

}

if(!p){

return ERROR; //异常处理

}

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

if(newone==NULL){

exit(OVERFLOW);

}

newone->data=e;

newone ->next= p->next; //新节点的后继为老节点的后继

p->next= newone; //老节点的后继为新节点

return OK;

}

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

LinkList p,q;

int j=1;

p=L;

while(j<i&&p!=NULL){

j++;

p=p->next;

}

if(p==NULL){

return ERROR; //第i个元素不存在

}

q=p->next;

p->next=q->next; //将p的后继设为q的后继

e=q->data; //读取q的数据

free(q); //释放内存

return OK;

}

status ListTrabverse(LinkList L){

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

LinkList p= L->next;

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

while(p!=NULL){

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

p=p->next;

}

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

return OK;

}

status LoadFromFile(char \* filename,LinkList & L){

IntiaList(L);

FILE \*fp;

if ((fp=fopen(filename,"r"))==NULL){

printf("File open error\n ");

return ERROR;

}

//L->next=(LinkList) malloc(sizeof(LNode));

LinkList p=L;

int temp;

while((fread(&temp,sizeof(ElemType),1,fp))){

p->next=(LinkList)malloc(sizeof(LNode));

p=p->next;

p->data=temp;

p->next=NULL;

}

fclose(fp);

return OK;

}

status SaveToFile(char \* filename,LinkList L){

FILE \*fp;

//写文件的方法

if ((fp=fopen(filename,"w"))==NULL){

printf("File open error\n ");

return ERROR;

}

LinkList p=L->next;

if(!p) return ERROR;

while(p){

fwrite(&(p->data),sizeof(ElemType),1,fp);

p=p->next;

}

fclose(fp);

return OK;

}

# 附录C 基于二叉链表二叉树实现的源程序

/\* Binary Tree On Double Linked Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status; /\*status是函数的类型，其值是函数结果状态代码\*/

typedef char ElemType; //数据元素类型定义

int depth, LR;

//数据元素类型定义

typedef struct BiTNode{

ElemType data;

struct BiTNode \*lchild, \*rchild; //左右孩子指针

}BiTNode, \*BiTree;

BiTree t1,t2;

typedef struct QNode //队列节点，包括一个数据域和一个指针域，数据域为二叉树的指针类型

{

BiTree data;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct

{

QueuePtr front,rear; //队头、队尾指针

}LinkQueue;

ElemType e, value, E;

BiTree e1, e2, e3, p;

FILE \*fp;

status InitQueue(LinkQueue \*Q); //构造一个空队列

status EnQueue(LinkQueue \*Q,BiTree e); //销毁队列Q，Q不再存在

status QueueEmpty(LinkQueue Q); //若栈Q为空队列，则返回TURE，否则返回FALSE

status DeQueue(LinkQueue \*Q,BiTree \*e); //删除队列Q的首元素,并且用e返回其值并返回OK；否则返回ERROR

BiTree GetPoint(BiTree T,ElemType s);

status InitBiTree(BiTree \* T); //构造空二叉树T

status DestroyBiTree(BiTree \* T); //二叉树T已存在,销毁二叉树T

void CreateBiTree(BiTree \* T,char \* s); //先序构造二叉树T

status ClearBiTree(BiTree \* T); //清空二叉树

bool BiTreeEmpty(BiTree T); //初始条件：二叉树T存在，若T为空二叉树，则返回TRUE,否则返回FALSE.

int BiTreeDepth(BiTree T); //求二叉树深度

BiTNode Root(BiTree T); //返回二叉树的根

ElemType Value(BiTree T, BiTree e); //返回二叉树结点的值

status Assign(BiTree T, BiTree e, ElemType value); //给二叉树结点赋值

ElemType Parent(BiTree T,ElemType e); //求双亲结点

ElemType LeftChild(BiTree T,ElemType e); //求左孩子结点

ElemType RightChild(BiTree T,ElemType e); //求右孩子结点

ElemType LeftSibling(BiTree T,ElemType e); //求左兄弟结点

ElemType RightSibling(BiTree T,ElemType e); //求右兄弟结点

status InsertChild(BiTree T,BiTree p,int LR,BiTree c);//插入子树

status DeleteChild(BiTree T,BiTree p,int LR); //删除子树

status PreOrderTraverse(BiTree T, status (\* Visit)(ElemType e)); //先序遍历二叉树

status InOrderTraverse(BiTree T,status (\* Visit)(ElemType e)); //中序遍历二叉树

status PostOrderTraverse(BiTree T,status (\* Visit)(ElemType e)); //后序遍历二叉树

status LevelOrderTraverse(BiTree T,status (\* Visit)(ElemType e)); //层序遍历二叉树

void level(BiTree T, int h, status (\* Visit)(ElemType e));

status Print(ElemType e);

int coutTheNode(BiTree T);

BiTree GetPoint(BiTree T,ElemType s); /\* 返回二叉树T中指向元素值为s的结点的指针。\*/

status SaveToFile(BiTree T); //保存到文件中

status LoadFromFile(BiTree \*T); //从文件中读取

status InitBiTree(BiTree \*T)

{

\*T = NULL;//指向根节点的指针初始化为空

return OK;

}

status DestroyBiTree(BiTree \*T)

{

free(T);//释放根节点的内存

return OK;

}

void CreateBiTree(BiTree \*T)

{

//按先序次序输入二叉树中结点的值(一个字符)，空格字符表示空树

char ch;

ch= getchar();

if(ch == '$')

{

\*T = NULL;

}

else

{

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

(\*T)->data = ch; //生成根节点

CreateBiTree(&((\*T)->lchild));//构造左子树

CreateBiTree(&((\*T)->rchild));//构造右子树

}

}

status ClearBiTree(BiTree \*T){

if((\*T)==NULL)

return OK;

else

{

ClearBiTree(&(\*T)->rchild);//对左子树递归清除

ClearBiTree(&(\*T)->lchild);//对右子树递归清除

free(\*T);

return OK;

}

}

bool BiTreeEmpty(BiTree T){

if(T == NULL){

return true;

}

else{

return false;

}

}

int BiTreeDepth(BiTree T){

int dept = 0;

if(T){

int lchilddepth = BiTreeDepth(T->lchild);

int rchilddepth = BiTreeDepth(T->rchild);

dept = lchilddepth >= rchilddepth ? (lchilddepth + 1) : (rchilddepth + 1);

}

return dept;

}

BiTNode Root(BiTree T){

return \*T;

}

ElemType Value(BiTree T, BiTree e){

return e->data;

}

status Assign(BiTree T, BiTree e, ElemType value){

e->data = value;

return OK;

}

ElemType Parent(BiTree T,ElemType e){

/\* 初始条件: 二叉树T存在,e是T中某个结点的元素 \*/

/\* 操作结果: 若e是T的非根结点,则返回它的双亲,否则返回＂空＂ \*/

LinkQueue q;

BiTree a;

if(T){ /\* 非空树 \*/

InitQueue(&q); /\* 初始化队列 \*/

EnQueue(&q,T); /\* 树根入队 \*/

while(!QueueEmpty(q)) {/\* 队不空 \*/

DeQueue(&q,&a); /\* 出队,队列元素赋给a \*/

if((a->lchild && a->lchild->data==e) || (a->rchild && a->rchild->data==e))

/\* 找到e(是其左或右孩子) \*/

return a->data; /\* 返回e的双亲的值 \*/

else {/\* 没找到e,则入队其左右孩子指针(如果非空) \*/

if(a->lchild)

EnQueue(&q,a->lchild);

if(a->rchild)

EnQueue(&q,a->rchild);

}

}

}

return ERROR; /\* 树空或没找到e \*/

}

ElemType LeftChild(BiTree T,ElemType e)

{

BiTree a;

if(T){

a=GetPoint(T,e);

if(a&&a->lchild)

return a->lchild->data;

}

return ERROR;

}

ElemType RightChild(BiTree T,ElemType e)

{

BiTree a;

if(T){

a=GetPoint(T,e);//得到指向e的指针

if(a&&a->rchild)

return a->rchild->data;

}

return ERROR;

}

ElemType LeftSibling(BiTree T,ElemType e)

{

ElemType a;

BiTree p;

if(T){

a=Parent(T,e);

p=GetPoint(T,a);//得到指向e父节点的指针

if(p->lchild&&p->rchild&&p->rchild->data==e)

return p->lchild->data;

}

return ERROR;

}

ElemType RightSibling(BiTree T,ElemType e)

{

ElemType a;

BiTree p;

if(T){

a=Parent(T,e);

p=GetPoint(T,a);//得到指向e又节点的指针

if(p->lchild&&p->rchild&&p->lchild->data==e)

return p->rchild->data;

}

return ERROR;

}

status InsertChild(BiTree T,BiTree \*p,int LR,BiTree c)

{

if(!c&&!T&&(LR==1||LR==0)) exit(OVERFLOW);

if(LR==0){ //添加左孩子节点

if((\*p)->lchild!=NULL) return ERROR;

(\*p)->lchild = c;

}

else{ //添加右孩子节点

if((\*p)->rchild!=NULL) return ERROR;

(\*p)->rchild = c;

}

return OK;

}

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

if(T){

if(!LR){ //删除左孩子节点

(\*p)->lchild=NULL;

}

else{ //删除右孩子节点

(\*p)->rchild=NULL;

}

return OK;

}

return ERROR;

}

status PreOrderTraverse(BiTree T, status (\* Visit)(ElemType e)){//先序遍历

if(T){//若二叉树不为空

if(Visit(T->data))

if(PreOrderTraverse(T->lchild, Visit))//先序遍历左子树

if(PreOrderTraverse(T->rchild, Visit)) //先序遍历右子树

return OK;

return 0;

}

else return OK;

}

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

if(T){

if(InOrderTraverse(T->lchild,Visit)) //中序遍历访问

if(Visit(T->data))

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

return 1;

return 0;

}else return 1;

}

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

if(T){

if(PostOrderTraverse(T->lchild,Visit)) //后序遍历访问

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

if(Visit(T->data))

return 1;

return 0;

}else return 1;

}

void level(BiTree T, int h, status (\* Visit)(ElemType e)){

if(T){

if(h==1)

Visit(T->data); //节点为指定值

else{

level(T->lchild,h-1, Visit); //节点深度小于指定值

level(T->rchild,h-1, Visit);

}

}

}

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

if(T){

int h=BiTreeDepth(T);

int i;

for(i=1;i<=h;i++){

level(T,i, Visit); //对深度为指定值的节点进行遍历访问

}

return OK;

}else return ERROR;

}

status Print(ElemType e){

printf("%c", e);

return OK;

}

status InitQueue(LinkQueue \*Q){

/\* 构造一个空队列Q \*/

(\*Q).front=(\*Q).rear=(QueuePtr)malloc(sizeof(QNode));

if(!(\*Q).front)

exit(OVERFLOW);

(\*Q).front->next=NULL;

return OK;

}

status EnQueue(LinkQueue \*Q,BiTree e){

/\* 插入元素e为Q的新的队尾元素 \*/

QueuePtr p=(QueuePtr)malloc(sizeof(QNode));

if(!p) /\* 存储分配失败 \*/

exit(OVERFLOW);

p->data=e;

p->next=NULL;

(\*Q).rear->next=p;

(\*Q).rear=p;

return OK;

}

status QueueEmpty(LinkQueue Q){ /\* 若Q为空队列,则返回TRUE,否则返回FALSE \*/

if(Q.front==Q.rear)

return TRUE;

else

return FALSE;

}

status DeQueue(LinkQueue \*Q,BiTree \*e){ /\* 若队列不空,删除Q的队头元素,用e返回其值,并返回OK,否则返回ERROR \*/

QueuePtr p;

if((\*Q).front==(\*Q).rear)

return ERROR;

p=(\*Q).front->next;

\*e=p->data;

(\*Q).front->next=p->next;

if((\*Q).rear==p)

(\*Q).rear=(\*Q).front;

free(p);

return OK;

}

BiTree GetPoint(BiTree T,ElemType s){ /\* 返回二叉树T中指向元素值为s的结点的指针。\*/

LinkQueue q;

BiTree a;

if(T) /\* 非空树 \*/

{

InitQueue(&q); /\* 初始化队列 \*/

EnQueue(&q,T); /\* 根结点入队 \*/

while(!QueueEmpty(q)) /\* 队不空 \*/

{

DeQueue(&q,&a); /\* 出队,队列元素赋给a \*/

if(a->data==s)

return a;

if(a->lchild) /\* 有左孩子 \*/

EnQueue(&q,a->lchild); /\* 入队左孩子 \*/

if(a->rchild) /\* 有右孩子 \*/

EnQueue(&q,a->rchild); /\* 入队右孩子 \*/

}

}

return NULL;

}

int coutTheNode(BiTree T){

int cnt=1;

if(T->lchild) cnt+=coutTheNode(T->lchild);

if(T->rchild) cnt+=coutTheNode(T->rchild);

return cnt;

}

status LoadFromFile(BiTree \*T)

{

//按先序次序输入二叉树中结点的值(一个字符)，空格字符表示空树

char ch;

fread(&ch,sizeof(ElemType),1,fp);

if(ch == '$')

{

\*T = NULL;

}

else

{

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

(\*T)->data = ch; //生成根节点

//printf("%c ",(\*T)->data);

LoadFromFile(&((\*T)->lchild));//构造左子树

LoadFromFile(&((\*T)->rchild));//构造右子树

}

}

status SaveToFile(BiTree T){

char temp='$';

if((T)!=NULL) {

printf("%c ",T->data);

fwrite(&((T)->data),sizeof(ElemType),1,fp);

}

if(T->lchild!=NULL) SaveToFile(((T)->lchild));//构造左子树

else fwrite(&temp,sizeof(ElemType),1,fp);

if(T->rchild!=NULL) SaveToFile(((T)->rchild));//构造右子树

else fwrite(&temp,sizeof(ElemType),1,fp);

}

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

int main(){

int op=1;

ElemType e,ans;

int i;

int option;

char fileName[20];

int lorr; char temp;

BiTree thisnode ;

BiTree newnode;

BiTree tempnode;

int stringleng;

while(op){

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

printf(" 二叉树操作菜单 \n");

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

printf(" 1. InitBiTree（初始化空二叉树） 11. LeftChild（求节点左孩子）\n");

printf(" 2. DestroyBiTree（销毁二叉树） 12. RightChild（求节点右孩子）\n");

printf(" 3. CreatBiTree（先序创建二叉树） 13. LeftSibling（求节点左兄弟）\n");

printf(" 4. ClearBiTree（清空二叉树） 14. RightSibling（求节点右兄弟）\n");

printf(" 5. BiTreeEmpty（判断二叉树是否为空） 15. InsertChild （插入子树）\n");

printf(" 6. BiTreeDepth（求二叉树深度） 16. DeleteChild （删除子树）\n");

printf(" 7. Root （返回二叉树的根） 17. PreOrderTraverse （先序遍历）\n");

printf(" 8. Value （返回节点的值） 18. inOrderTraverse （中序遍历）\n");

printf(" 9. Assign （给节点赋值） 19. PostOrderTraverse （后序遍历）\n");

printf(" 10. Parent （求父亲节点） 20. LevelOrderTraverse（层序子树）\n");

printf(" 0. Exit （退出系统）\n");

printf(" 21.Save （将二叉树保存到文件） 22.Load （从文件中读取二叉树） \n");

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

printf(" 请选择你的操作[0~20]:");

scanf("%d",&op);

switch(op){

case 1:

if(InitBiTree(&t1)==OK) printf("二叉树1创建成功！\n");

else printf("二叉树1创建失败！\n");

getchar();getchar();

break;

case 2:

if(DestroyBiTree(&t1)==OK) printf("二叉树1删除成功！\n");

else printf("二叉树1删除失败！\n");

getchar();getchar();

break;

case 3:

stringleng=0;

getchar();

printf("请输入你要建树的字符串：\n");

CreateBiTree(&t1);

//if(coutTheNode(t1)!=stringleng) {

// printf("输入字符串非法");

// DestroyBiTree(&t1);

//}

//else

printf("二叉树1构建成功！\n");

getchar();getchar();

break;

case 4:

if(ClearBiTree(&t1)==OK) printf("此二叉树已清空！\n");

else printf("清空出错！\n");

getchar();getchar();

break;

case 5:

if(BiTreeEmpty(t1)) printf("此二叉树为空！\n");

else printf("此二叉树非空\n");

getchar();getchar();

break;

case 6:

int ans;

if(t1!=NULL&&(ans=BiTreeDepth(t1))) printf("此二叉树深度为:%d\n",ans);

else printf("操作有误！\n");

getchar();getchar();

break;

case 7:

BiTNode rt;

if(t1!=NULL){ //当树存在时再取根

rt=Root(t1);

printf("此根节点的值为：%c\n",rt.data);

}

else printf("操作有误！\n");

getchar();getchar();

break;

case 8:

printf("请输入你要获取节点的值：\n");getchar();

scanf("%c",&temp);

thisnode = GetPoint(t1,temp); //得到值为temp1的节点

if(t1!=NULL&&thisnode!=NULL) printf("该节点元素为：%c\n",Value(t1,thisnode));

else printf("操作有误！\n");

getchar();getchar();

break;

case 9:

char temp1;

printf("请输入你要修改的节点的值：\n");getchar();

scanf("%c",&temp1);

printf("请输入修改后的值：\n");getchar();

scanf("%c",&temp);

thisnode =GetPoint(t1,temp1); //得到值为temp1的节点

if(t1!=NULL&&thisnode!=NULL) printf("修改成功%d\n",Assign(t1,thisnode,temp));

else printf("操作有误！\n");

getchar();getchar();

break;

case 10:

printf(" 请输入你要查询的元素：\n"); getchar();

scanf("%c",&temp);

if(t1!=NULL&&(ans=Parent(t1,temp))!=ERROR) printf("其双亲节点的值为：%c\n",ans);

else {

if(t1==NULL) printf("操作有误！\n");

else printf("未找到该节点或无双亲！\n");

}

getchar();getchar();

break;

case 11:

printf(" 请输入你要查询的元素：\n"); getchar();

scanf("%c",&temp);

if(t1!=NULL&&(ans=LeftChild(t1,temp))!=ERROR) printf("其左孩子节点的值为：%c\n",ans);

else {

if(t1==NULL) printf("操作有误！\n");

else printf("未找到该节点或无左孩子！\n");

}

getchar();getchar();

break;

case 12:

printf(" 请输入你要查询的元素：\n"); getchar();

scanf("%c",&temp);

if(t1!=NULL&&(ans=RightChild(t1,temp))!=ERROR) printf("其右孩子节点的值为：%c\n",ans);

else {

if(t1==NULL) printf("操作有误！\n");

else printf("未找到该节点或无右孩子！\n");

}

getchar();getchar();

break;

case 13:

printf(" 请输入你要查询的元素：\n"); getchar();

scanf("%c",&temp);

if(t1!=NULL&&(ans=LeftSibling(t1,temp))!=ERROR) printf("其左兄弟节点的值为：%c\n",ans);

else {

if(t1==NULL) printf("操作有误！\n");

else printf("未找到该节点或无左兄弟！\n");

}

getchar();getchar();

break;

case 14:

printf(" 请输入你要查询的元素：\n"); getchar();

scanf("%c",&temp);

if(t1!=NULL&&(ans=RightSibling(t1,temp))!=ERROR) printf("其右兄弟节点的值为：%c\n",ans);

else {

if(t1==NULL) printf("操作有误！\n");

else printf("未找到该节点或无右兄弟！\n");

}

getchar();getchar();

break;

case 15:

printf("请输入你想插到左还是右子树：（0左1右）\n");

scanf("%d",&lorr);

printf("请输入你要插入的节点：\n"); getchar();

scanf("%c",&temp);getchar();

printf("请输入你想插入的值：\n");

CreateBiTree(&t2);

tempnode=GetPoint(t1,temp);

if(lorr==0){

if(t1!=NULL&&t2!=NULL&&InsertChild(t1,&tempnode,lorr,t2)) printf("插入成功！\n");

else printf("操作有误!\n");

}

else{

if(t1!=NULL&&InsertChild(t1,&tempnode,lorr,t2)) printf("插入成功！\n");

else printf("操作有误!\n");

}

getchar();getchar();

break;

case 16:

printf("请输入你想删除左还是右子树：（0左1右）\n");

scanf("%d",&lorr);

printf("请输入进行删除操作的节点：\n");getchar();

scanf("%c",&temp);

thisnode = GetPoint(t1,temp);

printf("%c\n",thisnode->data);

if(lorr==0){

if(t1!=NULL&&thisnode!=NULL&&thisnode->lchild!=NULL&&DeleteChild(t1,&thisnode,lorr)) printf("删除成功！\n");

else printf("操作有误!\n");

}

else{

if(t1!=NULL&&tempnode!=0&&thisnode->rchild!=NULL&&DeleteChild(t1,&thisnode,lorr)) printf("删除成功！\n");

else printf("操作有误!\n");

}

getchar(); getchar();

break;

case 17:

if(t1!=NULL&&PreOrderTraverse(t1,Print)) printf("\n遍历成功！\n");

else printf("遍历失败或操作有误！\n");

getchar();getchar();

break;

case 18:

if(t1!=NULL&&InOrderTraverse(t1,Print)) printf("\n遍历成功！\n");

else printf("遍历失败或操作有误！\n");

getchar();getchar();

break;

case 19:

if(t1!=NULL&&PostOrderTraverse(t1,Print)) printf("\n遍历成功！\n");

else printf("遍历失败或操作有误！\n");

getchar();getchar();

break;

case 20:

if(t1!=NULL&&LevelOrderTraverse(t1,Print)) printf("\n遍历成功！\n");

else printf("遍历失败或操作有误！\n");

getchar();getchar();

break;

case 21:

printf("\n请输入要保存的文件名：\n");

scanf("%s",fileName);

if ((fp=fopen(fileName,"w"))==NULL){

printf("File open error\n ");

return ERROR;

}

SaveToFile(t1);

printf("已成功保存至文件!");

fclose(fp);

getchar();getchar();

break;

case 22:

printf("\n请输入要读取的文件名：\n");

scanf("%s",fileName);

if ((fp=fopen(fileName,"r"))==NULL){

printf("File open error\n ");

return ERROR;

}

else{

LoadFromFile(&t1);

printf("已成功从文件读取!");

}

fclose(fp);

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

return 0;

}//end of main()

# 附录D 基于邻接表图实现的源程序

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//函数结果状态代码

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

//Status是函数的类型，其值是函数结果状态代码

typedef int Status;

typedef int QElemType;

typedef struct QNode{

QElemType data;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct{

QueuePtr front,rear; //队头、队尾指针

}LinkQueue;

Status InitQueue(LinkQueue \*Q){

//构造一个空队列Q

(\*Q).front = (\*Q).rear = (QueuePtr)malloc(sizeof(QNode));

if(!(\*Q).front) exit(OVERFLOW);

(\*Q).front->next = NULL;

return OK;

}

Status QueueEmpty(LinkQueue Q){//若Q为空队列,则返回TRUE,否则返回FALSE

if(Q.front == Q.rear) return TRUE;

else return FALSE;

}

Status EnQueue(LinkQueue \*Q, QElemType e){//插入元素e为Q的新的队尾元素

QueuePtr p = (QueuePtr)malloc(sizeof(QNode));

if(!p) //存储分配失败

exit(OVERFLOW);

p->data = e;

p->next = NULL;

(\*Q).rear->next = p;

(\*Q).rear = p;

return OK;

}

Status DeQueue(LinkQueue \*Q, QElemType \*e){ //若队列不空,删除Q的队头元素,用e返回其值,并返回OK,否则返回ERROR

QueuePtr p;

if((\*Q).front == (\*Q).rear)

return ERROR;

p = (\*Q).front->next;

\*e = p->data;

(\*Q).front->next = p->next;

if((\*Q).rear == p)

(\*Q).rear = (\*Q).front;

free(p);

return OK;

}

typedef int Boolean;

#define MAX\_VERTEX\_NUM 20 //最大顶点数

#define MAX\_VERNAME\_LEN 10 //顶点名称最大长度

//数据元素类型定义

typedef char VertexType[MAX\_VERNAME\_LEN];

typedef int InfoType;

typedef struct ArcNode{

int adjvex; //该弧所指向的顶点位置

struct ArcNode \* nextarc; //指向下一条弧的指针

InfoType \*info; //该弧相关信息的指针

}ArcNode;

typedef struct VNode{

ArcNode \* firstarc; //指向第一条依附该顶点的弧的指针

VertexType data; //顶点信息

}VNode, AdjList[MAX\_VERTEX\_NUM];

typedef struct{

AdjList vertices;

int vexnum, arcnum; //图的当前顶点数和弧数

int kind; //图的种类标志

}ALGraph;

Status CreateGraph(ALGraph \* G);//按V和VR的定义构造图G

Status DestroyGraph(ALGraph \* G);//销毁图G

int LocateVex(ALGraph G, VertexType u);//若u在图G中存在，返回顶点u的位置序号，否则返回-1

int FirstAdjVex(ALGraph G, VertexType v);//返回v的第一个邻接顶点，如果v没有邻接顶点，返回空

int NextAdjVex(ALGraph G, VertexType v, VertexType w);//返回v的（相对于w）下一个邻接顶点，如果w是最后一个邻接顶点，返回空。

Status InsertVex(ALGraph \* G, VertexType v);//在图G中增加新顶点v

Status DeleteVex(ALGraph \* G, VertexType v);//在图G中删除顶点v和与v相关的弧

Status InsertArc(ALGraph \* G, VertexType v, VertexType w);//在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>

Status DeleteArc(ALGraph \* G, VertexType v, VertexType w);//在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>

Status DFSTraverse(ALGraph G, Status (\*visit)(VertexType a));//对图G进行深度优先搜索遍历

Status BFSTraverse(ALGraph G, Status (\*visit)(VertexType a));//对图G进行广度优先搜索遍历

int OutDegree(VNode \* i);//计算图G第i个定点的出度

Status SaveGraph(ALGraph G);//保存

Status LoadGraph(ALGraph \* G);//读取

Status Visit(VertexType a);//输出a

Boolean visited[MAX\_VERTEX\_NUM];

//访问标志数组

Status (\*VisitFunc)(VertexType a);

//函数变量

/\*返回值： Status 成功则返回OK

\*函数功能： 构造图G\*/

Status CreateGraph(ALGraph \* G)

{

int i,j,k;

int w; //权值W

VertexType va,vb;//弧尾Va，弧头Vb

ArcNode \*p;

printf("请输入图的3项基本信息：\n类型(0-有向图, 1-有向网, 2-无向图, 3-无向网): ");

//输图类型

scanf("%d",&(\*G).kind);

while ((\*G).kind < 0 || (\*G).kind > 3){

printf("输入值非法！请重输: ");

scanf("%d",&(\*G).kind);

}

//输顶点数

printf("顶点数（%d以内）: ", MAX\_VERTEX\_NUM);

scanf("%d", &(\*G).vexnum);

while ((\*G).vexnum < 0 || (\*G).vexnum > MAX\_VERTEX\_NUM){

printf("输入值非法！请重输: ");

scanf("%d",&(\*G).vexnum);

}

//输边数

printf("弧(边)数: ");

scanf("%d", &(\*G).arcnum);

printf("请输入%d个顶点的名称或值(%d个字符以内,空格分开):\n", (\*G).vexnum, MAX\_VERNAME\_LEN);

for (i = 0; i < (\*G).vexnum; i++) {//输入各顶点信息

getchar();

scanf("%s",(\*G).vertices[i].data);

(\*G).vertices[i].firstarc = NULL;

}

if ((\*G).kind % 2) {//1、3为网

printf("请按序输入弧(边)权值W、弧尾Va和弧头Vb(空格分开):\n W Va Vb\n");

}

else {//0、2为图

printf("请按序输入弧(边)的弧尾Va和弧头Vb(空格分开):\n Va Vb\n");

}

for (k = 0; k < (\*G).arcnum; k++) {//构造邻接表

printf("%d: ", k+1);

if ((\*G).kind % 2) {//网

scanf("%d%s%s", &w, va, vb);

}

else {//图

scanf("%s%s", va, vb);

}

i = LocateVex(\*G, va); //弧尾

j = LocateVex(\*G, vb); //弧头

if (i == -1 || j == -1){//顶点不存在

printf("顶点不存在！请重输！\n");

k--;

continue;

}

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

p->adjvex = j;

if ((\*G).kind % 2){//网

p->info = (int \*)malloc(sizeof(int));

\*(p->info) = w;

}

else {//图

p->info = NULL;

}

p->nextarc = (\*G).vertices[i].firstarc; //插在表头

(\*G).vertices[i].firstarc = p;

//无向图或无向网还需逆向的

if((\*G).kind >= 2){

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

p->adjvex = i;

if ((\*G).kind == 3) {//无向网

p->info = (int\*)malloc(sizeof(int));

\*(p->info) = w;

}

else {//无向图

p->info = NULL;

}

p->nextarc = (\*G).vertices[j].firstarc; //插在表头

(\*G).vertices[j].firstarc = p;

}

}

return OK;

}

/\*返回值： Status 成功则返回OK

\*函数功能： 销毁图G\*/

Status DestroyGraph(ALGraph \*G){

int i;

ArcNode \* p, \* q;

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

p=(\*G).vertices[i].firstarc;

while(p){

q = p->nextarc;

if((\*G).kind % 2) {//网

free(p->info);

}

free(p);

p = q;

}

}

(\*G).vexnum = 0;

(\*G).arcnum = 0;

(\*G).kind = -1;//以-1代表图不存在

return OK;

}

/\*返回值： int

\*函数功能： 若u在图G中存在，返回顶点u的位置序号，否则返回-1\*/

int LocateVex(ALGraph G, VertexType u)

{

int i;

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

if(strcmp(u,G.vertices[i].data) == 0){ //名称相同

return i;

}

}

return -1;//未找到则返回-1

}

/\*返回值： int

\*函数功能： 返回v的第一个邻接顶点的序号。若顶点在G中没有邻接顶点,则返回-1\*/

int FirstAdjVex(ALGraph G, VertexType v)

{

ArcNode \*p;

int v1;

v1 = LocateVex(G,v);//获取v在图G中的序号v1

if (v1 == -1){

return -2;

}

p = G.vertices[v1].firstarc;

if (p){

return p->adjvex;

}

else{

return -1;

}

}

/\*返回值： int

\*函数功能： 返回v的(相对于w的)下一个邻接顶点的序号，

\* 若w是v的最后一个邻接点,则返回-1\*/

int NextAdjVex(ALGraph G,VertexType v,VertexType w){

ArcNode \*p;

int v1, w1;

v1 = LocateVex(G,v); //获取v在图G中的序号v1

w1 = LocateVex(G,w); //获取w在图G中的序号w1

if (v1 == -1){//顶点不存在

return(-2);

}

if (w1 == -1){//邻接点不存在

return(-3);

}

p = G.vertices[v1].firstarc;

while (p && p->adjvex != w1){//指针p不空且所指表结点不是w

p = p->nextarc;

}

//没找到w或w是最后一个邻接点

if(!p || !p->nextarc){

return -1;

}

else{//返回v的(相对于w的)下一个邻接顶点的序号

return p->nextarc->adjvex;

}

}

/\*返回值： Status

\*函数功能： 在图G中增加新顶点v\*/

Status InsertVex(ALGraph \*G, VertexType v){

if((\*G).vexnum == MAX\_VERTEX\_NUM){//顶点数达最大值

return ERROR;

}

//构造新顶点

strcpy((\*G).vertices[(\*G).vexnum].data, v);

(\*G).vertices[(\*G).vexnum].firstarc=NULL;

(\*G).vexnum++; //图G的顶点数加1

return OK;

}

/\*返回值： Status 成功返回OK

\*函数功能： 删除G中顶点v及其相关的弧\*/

Status DeleteVex(ALGraph \*G, VertexType v){

int i, j;

ArcNode \*p, \*q;

j = LocateVex(\*G, v); //获取顶点v的序号

if(j == -1) {//v不是图G的顶点

return ERROR;

}

//删除以v为出度的弧或边

p = (\*G).vertices[j].firstarc;

while(p){

q = p;

p = p->nextarc;

if((\*G).kind%2) {//网

free(q->info);

}

free(q);

(\*G).arcnum--; //弧或边数减1

}

//重新排列顶点

(\*G).vexnum--; //顶点数减1

for(i = j; i < (\*G).vexnum; i++) //顶点v后面的顶点前移

{

(\*G).vertices[i] = (\*G).vertices[i+1];

}

//删除以v为入度的弧或边且修改弧所指顶点位置值

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

p = (\*G).vertices[i].firstarc; //指向第1条弧或边

while(p){ //有弧或边

if(p->adjvex == j){//该弧指向被删结点

if(p == (\*G).vertices[i].firstarc){ //待删结点是第1个结点

(\*G).vertices[i].firstarc = p->nextarc;

if((\*G).kind%2) {//网

free(p->info);

}

free(p);

p = (\*G).vertices[i].firstarc;

if((\*G).kind < 2){//有向图需再减一次

(\*G).arcnum--; //弧或边数减1

}

}

else{

q->nextarc = p->nextarc;

if((\*G).kind % 2) {//网

free(p->info);

}

free(p);

p = q->nextarc;

if((\*G).kind < 2){//有向需再减一次

(\*G).arcnum--; //弧或边数减1

}

}

}

else{//该弧不指向被删结点

if(p->adjvex > j){//该弧所指元素位置序号在删除顶点之后

p->adjvex--;//更改弧所指顶点位置序号

}

q = p;

p = p->nextarc;//分析下一条弧

}

}

}

return OK;

}

/\*返回值： Status 成功返回OK；

\*函数功能：在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>\*/

Status InsertArc(ALGraph \*G,VertexType v,VertexType w){

ArcNode \*p;

int w1, i, j;

i = LocateVex(\*G, v); //获取v位置序号

j = LocateVex(\*G, w); //获取w位置序号

if(i == -1 || j == -1){//未找到相应顶点

return ERROR;

}

//建立弧v-w

(\*G).arcnum++;

if((\*G).kind % 2){ //网要输权值

printf("请输入弧(边)%s->%s的权值: ", v, w);

scanf("%d", &w1);

}

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

p->adjvex = j;

if((\*G).kind % 2) {//网

p->info = (int \*)malloc(sizeof(int));

\*(p->info) = w1;

}

else{

p->info=NULL;

}

p->nextarc = (\*G).vertices[i].firstarc;

(\*G).vertices[i].firstarc = p;

//无向图需要再建立弧w-v

if((\*G).kind >= 2){

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

p->adjvex = i;

if((\*G).kind == 3) {//无向网

p->info = (int\*)malloc(sizeof(int));

\*(p->info) = w1;

}

else{

p->info = NULL;

}

p->nextarc = (\*G).vertices[j].firstarc;

(\*G).vertices[j].firstarc = p;

}

return OK;

}

/\*返回值： Status 成功返回OK；

\*函数功能： 在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>\*/

Status DeleteArc(ALGraph \* G,VertexType v,VertexType w){

ArcNode \*p, \*q;

int i, j;

i = LocateVex(\*G, v); //顶点v序号

j = LocateVex(\*G, w); //顶点w序号

if(i < 0 || j < 0){

return ERROR;

}

//删除弧<v,w>

p = (\*G).vertices[i].firstarc;

while(p && p->adjvex != j){ //p不空且所指之弧不是待删除弧<v,w>

//p指向下一条弧

q = p;

p = p->nextarc;

}

if(p && p->adjvex == j) {//找到弧<v,w>

if(p == (\*G).vertices[i].firstarc){ //p所指是第1条弧

(\*G).vertices[i].firstarc = p->nextarc;//指向下一条弧

}

else{//指向下一条弧

q->nextarc = p->nextarc;

}

if((\*G).kind % 2){//网要多释放权值

free(p->info);

}

free(p); //释放此结点

(\*G).arcnum--; //弧或边数减1

}

//无向图要删除对称弧<w,v>

if((\*G).kind >= 2){

p = (\*G).vertices[j].firstarc;

while(p && p->adjvex != i) {//p不空且所指之弧不是待删除弧<w,v>

q = p;//p指向下一条弧

p = p->nextarc;

}

if(p && p->adjvex == i){ //找到弧<w,v>

if(p == (\*G).vertices[j].firstarc){ //p所指是第1条弧

(\*G).vertices[j].firstarc=p->nextarc;//指向下一条弧

}

else{//指向下一条弧

q->nextarc=p->nextarc;

}

if((\*G).kind % 2){//网要多释放权值

free(p->info);

}

free(p); //释放此结点

}

}

return OK;

}

/\*返回值： 无

\*函数功能： 从第v个顶点出发递归地深度优先遍历图G。\*/

void DFS(ALGraph G,int v)

{

int w;

VertexType v1, w1;

strcpy(v1, G.vertices[v].data);

visited[v] = TRUE; //设置访问标志为TRUE(已访问)

VisitFunc(G.vertices[v].data); /\* 访问第v个顶点 \*/

for(w = FirstAdjVex(G,v1); w >= 0; w = NextAdjVex(G, v1, strcpy(w1, G.vertices[w].data))){

if(!visited[w]){

DFS(G,w); //对v的尚未访问的邻接点w递归调用DFS

}

}

}

/\*返回值： Status 成功返回OK；

\*函数功能： 对图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函

\* 数visit访问一次，且仅访问一次\*/

Status DFSTraverse(ALGraph G, Status (\*Visit)(VertexType a)){

int v;

VisitFunc = Visit; //使用全局变量VisitFunc,使DFS不必设函数指针参数

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

visited[v]=FALSE; //访问标志数组初始化

}

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

if(!visited[v]){

DFS(G,v); //对尚未访问的顶点调用DFS

}

}

return OK;

}

/\*返回值： Status 成功返回OK；

\*函数功能： 按广度优先非递归遍历图G。\*/

Status BFSTraverse(ALGraph G, Status (\*Visit)(VertexType a)){

int v, u, w;

VertexType u1, w1;

LinkQueue Q;

for(v = 0; v < G.vexnum; ++v) visited[v]=FALSE;

InitQueue(&Q); //置空的辅助队列Q

for(v = 0; v < G.vexnum; ++v)

if(!visited[v]) {//v尚未访问

visited[v] = TRUE;

Visit(G.vertices[v].data);

EnQueue(&Q, v); //v入队列

while(!QueueEmpty(Q)){

DeQueue(&Q, &u); //队头元素出队并置为u

strcpy(u1, G.vertices[u].data);

for(w = FirstAdjVex(G,u1); w >= 0; w = NextAdjVex(G, u1, strcpy(w1, G.vertices[w].data)))

if(!visited[w]){ //w为u的尚未访问的邻接顶点

visited[w] = TRUE;

Visit(G.vertices[w].data);

EnQueue(&Q,w); //w入队

}

}

}

return OK;

}

/\*返回值： int 图G的第i个顶点出度

\*函数功能： 计算图G的第i个顶点出度\*/

int OutDegree(VNode \* i){

int num = 0;

ArcNode \* p;

p = (\*i).firstarc;

while (p != NULL){

num++;

p = p->nextarc;

}

return num;

}

/\*返回值： Status

\*函数功能： 读取线性表。\*/

Status LoadGraph(ALGraph \* G){

int i, k = 0, num;

FILE \*fp;

ArcNode \*p;

char filename[30];

printf("请输入读取文件名: ");

scanf("%s", &filename);

if ((fp = fopen(filename,"r")) == NULL)

{

printf("文件打开失败！\n ");

return ERROR;

}

//读取图基本信息

fread(&G->kind, sizeof(int), 1, fp);

fread(&G->vexnum, sizeof(int), 1, fp);

fread(&G->arcnum, sizeof(int), 1, fp);

//读取邻接表

while(fread(G->vertices[k].data, sizeof(char), MAX\_VERTEX\_NUM, fp)){//有顶点

fread(&num, sizeof(int), 1, fp);

if (num == 0){

G->vertices[k].firstarc = NULL;

}

else{

G->vertices[k].firstarc = NULL;

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

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

fread(&(p->adjvex), sizeof(int), 1, fp);

if ((\*G).kind % 2){//网

p->info = (int \*)malloc(sizeof(int));

fread(p->info, sizeof(InfoType), 1, fp);

}

else {//图

p->info = NULL;

}

p->nextarc = (\*G).vertices[k].firstarc; //插在表头

(\*G).vertices[k].firstarc = p;

}

}

k++;

}

fclose(fp);

return OK;

}

/\*返回值： Status

\*函数功能： 保存线性表。\*/

Status SaveGraph(ALGraph G){

int i, num;

FILE \*fp;

char filename[30];

ArcNode \*p;

printf("请输入保存文件名： ");

scanf("%s",&filename);

//写文件的方法

if ((fp = fopen(filename, "w")) == NULL)

{

printf("文件打开失败\n ");

return ERROR;

}

//保存图基本信息

fwrite(&G.kind, sizeof(int), 1, fp);

fwrite(&G.vexnum, sizeof(int), 1, fp);

fwrite(&G.arcnum, sizeof(int), 1, fp);

//保存邻接表

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

fwrite(G.vertices[i].data, sizeof(char), MAX\_VERTEX\_NUM, fp);

num = OutDegree(&(G.vertices[i]));

fwrite(&num, sizeof(int), 1, fp);

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

fwrite(&p->adjvex, sizeof(int), 1, fp);

if (G.kind % 2){//网

fwrite(p->info, sizeof(int), 1, fp);

}

while ((p = p->nextarc) != NULL){

fwrite(&p->adjvex, sizeof(int), 1, fp);

if (G.kind % 2){//网

fwrite(p->info, sizeof(int), 1, fp);

}

}

}

}

fclose(fp);

return OK;

}

/\*返回值： Status 返回OK

\*函数功能： 输出a\*/

Status Visit(VertexType a)

{

printf("%s ",a);

return OK;

}

ALGraph g1;

int main(){

int op=1;

int i;

int option;

char s[30];

char temp[30];

//BiTree thisnode ;

//BiTree newnode;

//BiTree tempnode;

int stringleng;

int id;

while(op){

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

printf(" 图操作菜单 \n");

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

printf(" 1. CreateGraph（创建图） 9. DeleteVex（删除顶点）\n");

printf(" 2. DestroyGraph（销毁图） 10. InsertArc（插入弧）\n");

printf(" 3. LocateVex （查找顶点） 11. DeleteArc（删除弧）\n");

printf(" 4. GetVex （获得顶点值） \n");

printf(" 5. PutVex （顶点赋值） 12. DFSTraverse （深度优先搜索遍历）\n");

printf(" 6. FirstAdjVex（获得第一邻接点）13. BFSTraverse （广深度优先搜索遍历）\n");

printf(" 7. NextAdjVex （获得下一邻接点） \n");

printf(" 8. InsertVex （插入顶点） \n");

printf(" 0. Exit （退出系统）\n");

printf(" 14.Save （将图保存到文件） 15.Load （从文件中读取图） \n");

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

printf(" 请选择你的操作[0~15]:");

scanf("%d",&op);

switch(op){

case 1:

if(CreateGraph(&g1)==OK) printf("图1创建成功！\n");

else printf("二叉树1创建失败！\n");

getchar();getchar();

break;

case 2:

if(DestroyGraph(&g1)==OK) printf("图1删除成功！\n");

else printf("图1删除失败！\n");

getchar();getchar();

break;

case 3:

printf("请输入你要查找的顶点：\n");getchar();

scanf("%s",s);

id=LocateVex(g1,s);

if(id==-1){

printf("无此顶点\n");

}

else {

printf("此顶点编号为：%d\n",id);

}

getchar();getchar();

break;

case 4:

printf("请输入你要获得值的顶点：\n");getchar();

scanf("%s",s);

id=LocateVex(g1,s);

if(id==-1){

printf("无此顶点\n");

}

else {

printf("此顶点值为：%s\n",g1.vertices[id].data);

}

getchar();getchar();

break;

case 5:

printf("请输入你要赋值的顶点：\n");getchar();

scanf("%s",s);

printf("请输入新的值：\n");getchar();

scanf("%s",temp);

id=LocateVex(g1,s);

if(id==-1){

printf("无此顶点\n");

}

else {

strcpy(g1.vertices[id].data,temp);

printf("修改成功");

}

getchar();getchar();

break;

case 6:

printf("请输入你要操作的顶点：\n");getchar();

scanf("%s",s);

id=FirstAdjVex(g1,s);

if(id==-1){

printf("输入有误\n");

}

else if(id==-2){

printf("不存在邻接点\n");

}

else {

printf("第一邻接点为编号为%d\n",id);

}

getchar();getchar();

break;

case 7:

printf("请输入你要操作的顶点：\n");getchar();

scanf("%s",s);

printf("请输入一个它的邻接点：\n");getchar();

scanf("%s",temp);

id=NextAdjVex(g1,s,temp);

if(id==-1){

printf("无下一个邻接点\n");

}

else if(id==-2){

printf("该顶点不存在\n");

}

else if(id==-3){

printf("该邻接点不存在\n");

}

else{

printf("下一邻接点为编号为%d\n",id);

}

getchar();getchar();

break;

case 8:

printf("请输入你加入的顶点：\n");getchar();

scanf("%s",s);

id=InsertVex(&g1,s);

if(id){

printf("添加成功\n");

}

else {

printf("操作失败，点数太大\n");

}

getchar();getchar();

break;

case 9:

printf("请输入你删除的顶点：\n");getchar();

scanf("%s",s);

id=DeleteVex(&g1,s);

if(id){

printf("删除成功\n");

}

else {

printf("操作失败\n");

};

getchar();getchar();

break;

case 10:

printf("请输入第一个顶点：\n");getchar();

scanf("%s",s);

printf("请输入第二个顶点：\n");getchar();

scanf("%s",temp);

id=InsertArc(&g1,s,temp);

if(id){

printf("添加成功\n");

}

else {

printf("操作失败\n");

}

getchar();getchar();

break;

case 11:

printf("请输入第一个顶点：\n");getchar();

scanf("%s",s);

printf("请输入第二个顶点：\n");getchar();

scanf("%s",temp);

id=DeleteArc(&g1,s,temp);

if(id){

printf("删除成功\n");

}

else {

printf("操作失败\n");

}

getchar();getchar();

break;

case 12:

if(DFSTraverse(g1,Visit) ){

printf("遍历成功");

}

else {

printf("操作失败");

}

getchar();getchar();

break;

case 13:

if(BFSTraverse(g1,Visit) ){

printf("遍历成功");

}

else {

printf("操作失败");

}

getchar();getchar();

break;

case 14:

if(SaveGraph(g1)){

printf("保存成功");

}

else{

printf("操作失败");

}

getchar();getchar();

break;

case 15:

if(LoadGraph(&g1)){

printf("读取成功");

}

else{

printf("操作失败");

}

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

return 0;

}//end of main()