

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术201705**

**学 号： U201714726**

**姓 名： 王明明**

**指导教师： 李丹**

**报告日期： 2018年 11月 18 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](#_Toc532755140)

[1.1 问题描述 2](#_Toc532755141)

[1.2 系统设计 3](#_Toc532755142)

[1.3 系统实现 15](#_Toc532755146)

[1.4 实验小结 27](#_Toc532755149)

[2 基于链式存储结构的线性表实现 29](#_Toc532755150)

[2.1 问题描述 29](#_Toc532755151)

[2.2 系统设计 30](#_Toc532755152)

[2.3 系统实现 45](#_Toc532755156)

[2.4 实验小结 57](#_Toc532755159)

[3 基于二叉链表的二叉树实现 59](#_Toc532755160)

[3.1 问题描述 59](#_Toc532755161)

[3.2 系统设计 61](#_Toc532755162)

[3.3 系统实现 70](#_Toc532755166)

[3.4 实验小结 90](#_Toc532755169)

[参考文献 92](#_Toc532755170)

[附录A 基于顺序存储结构线性表实现的源程序 94](#_Toc532755171)

[附录B 基于链式存储结构线性表实现的源程序 109](#_Toc532755172)

[附录C 基于二叉链表的二叉树实现 125](#_Toc532755173)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

构造顺序表，呈现一个简易菜单的功能演示系统，该演示系统可选择实现多个线性表管理。需要在主程序中完成函数调用以及所需实参值和函数执行结果的输出。定义线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等函数，并给出适当的操作提示，并且可选择以文件的形式进行存储和加载。

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。

⑴初始化表：函数名称是InitaList(L)；初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。

⑵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。

⑶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。

⑷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。

⑸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。

⑹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。

⑺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。

⑻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。

⑼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。

⑽插入元素：函数名称是ListInsert(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

⑿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()。

本实验的目的是加深对线性表的概念、基本运算的理解。能够熟练掌握线性表的逻辑结构与物理结构的关系。物理结构采用顺序表,能够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**

本系统提供一个顺序存储的线性表，一个简易的菜单。

菜单可选择的操作有：初始化线性表、销毁表、清空表、判空表，求表长、得到某元素、查找元素、获得某元素的前驱、获得某元素的后继、插入元素、删除元素、遍历线性表、加载预置文件、多表之间切换等。

**1.2.2有关常量和类型定义**

数据元素类型的定义：

typedef int status;

typedef int ElemType;

有关常量的定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

#define MAX\_NUM 10

**1.2.3算法设计**

(1)

函数名称：InitaList(L)；

初始条件：线性表L不存在已存在；

操作结果：是构造一个空的线性表；

算法思路：先分配存储空间后，将表长设为0，再将线性表容量设为预定义的初始存储容量，图1-1 为InitaList(L)函数的流程图。



图1-1 InitaList(L)流程图

(2)

函数名称：DestroyList(L)；

初始条件：线性表L已存在；

操作结果：销毁线性表L；

算法思路：释放内存并将各数据设置为初值，图1-2 为DestroyList (L)的流程图。



图1-2 DestroyList (L)流程图

(3)

函数名称：ClearList(L)；

初始条件：线性表L已存在；

操作结果：将L重置为空表；

算法思路：将表长设为0即可，图1-3 为ClearList(L)的流程图。



图1-3 ClearList(L)流程图

(4)

函数名称：ListEmpty(L)；

初始条件：线性表L已存在；

操作结果：若L为空表则返回TRUE,否则返回FALSE；

算法思路：表长为0则为空表，否则不是空表，图1-4为 ListEmpty(L)的流程图。



图1-4 ListEmpty(L)流程图

(5)

函数名称：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；

算法思路：返回线性表表长的值，图1-5 为ListLength(L)的流程图。



图1-5 ListLength(L)流程图

(6)

函数名称：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；

算法思路：将线性表中第i个数据元素的值赋值给e，图1-6为 GetElem(L,i,e)的流程图。



图1-6 GetElem(L,i,e)流程图

(7)

函数名称：LocateElem(L,e,compare())；

初始条件：线性表已存在；

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

算法思路：先遍历顺序表，将线性表中的数据元素依次与e进行比较，返回该元素的位序，图1-7 为LocateElem(L,e,compare())的流程图。



图1-7 LocateElem(L,e,compare())流程图

(8)

函数名称：PriorElem(L,cur\_e,pre\_e)；

初始条件：线性表L已存在；

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

算法思路：首先遍历该顺序表。若找到该结点，并且该结点有前驱元素，则将前驱元素赋值给pre\_e。若未找到该结点，或者找到该结点但该结点不存在前驱元素，则返回FALSE。图1-8为 PriorElem(L,cur\_e,pre\_e)的流程图。



图1-8 PriorElem(L,cur\_e,pre\_e)流程图

(9)

函数名称：NextElem(L,cur\_e,next\_e)

初始条件：线性表L已存在；

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

算法思路：先遍历线性表，如果找到该结点并且该结点有后继元素，则将后继节点元素赋值给next\_e。若未找到该结点，或者找到该结点但该结点不存在后继元素，则返回FALSE。图1-9为 NextElem(L,cur\_e,next\_e)的流程图。



图1-9 NextElem(L,cur\_e,next\_e)流程图

(10)

函数名称：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。

算法思路：先遍历顺序表，若线性表L已存在且不为空，输入的i值不合法，则返回ERROR。若满足线性表L已存在且L非空，并且i的值合法，则在线性表的第i个位置之前插入新的数据元素e，返回OK。图1-10为 ListInsert(L,i,e)的流程图。



图1-10 ListInsert(L,i,e)流程图

(11)

函数名称：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；

设计思想：先遍历顺序表，如果线性表L已存在且非空，并且输入的i值不合法，则返回ERROR。若满足线性表L已存在且L非空，并且i的值合法，则删除线性表的第i个位置的数据元素，并用e返回其值，返回OK。图1-11 为ListDelete(L,i,e)的流程图。



图1-11 ListDelete(L,i,e)流程图

(12)

函数名称：ListTrabverse(L)；

初始条件：线性表L已存在；

操作结果：依次遍历L的每个数据元素；

设计思想：若线性表L存在，则遍历元素；否则返回ERROR。图1-12 为ListTrabverse(L)的流程图。



图1-12 ListTrabverse(L)流程图

(13)

函数名称：SaveList(L,filename)；

初始条件：线性表L已存在；

操作结果：将线性表L保存为文件形式；

算法思路：用fwrite保存为文件，图1-13 为SaveList(L,filename)的流程图。



图1-13 SaveList(L,filename)流程图

(14)

函数名称：LoadList (L);

初始条件：文件filename已存在；

操作结果：将线性表L以文件形式加载读取;

算法思路：用fread将文件读取顺序表，图1-14为LoadList (L)的流程图。



图1-14 LoadList (L)流程图

**1.3 系统实现**

**1.3.1程序源代码**

见《附录A 基于顺序存储结构线性表实现的源程序》。

**1.3.2系统测试**

程序采用简易界面，如图1-15所示，挑选ListEmpty, ListLength, GetElem, LocateElem, PriorElem, NextElem, ListInsert, ListDelete, ListTrabverse这些重要功能进行测试。

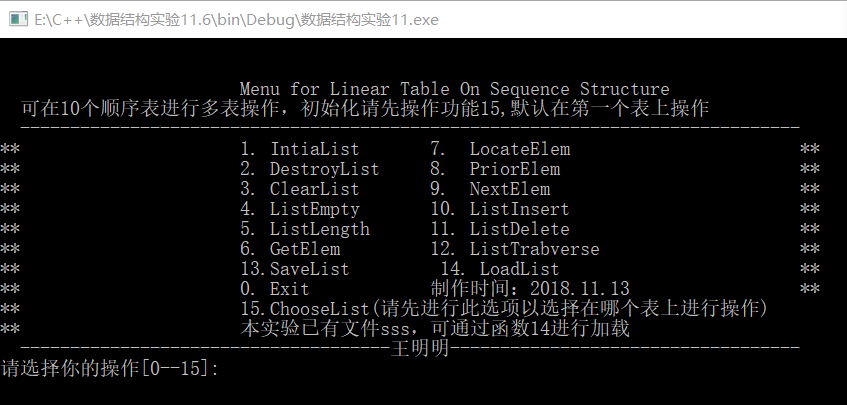


图1-15 程序简易界面截图

测试用例为：sss{1,2,3,4,5,6,7,8,9},null(空表)以及顺序表不存在。

(1)ListEmpty测试

表1-1 ListEmpty测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选4 | 线性表不是空表！ | 线性表不是空表！ |
| null | 界面选4 | 文件为空！ | 文件为空！ |
| 若表不存在 | 界面选4 | 线性表不存在！ | 线性表不存在！ |



图1-16 ListEmpty测试截图

(2) ListLength测试

表1-2 ListLength测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选5 | 线性表表长为9 | 线性表表长为9 |
| null | 界面选5 | 线性表表长为0 | 线性表表长为0 |
| 若表不存在 | 界面选5 | 线性表不存在！ | 线性表不存在！ |

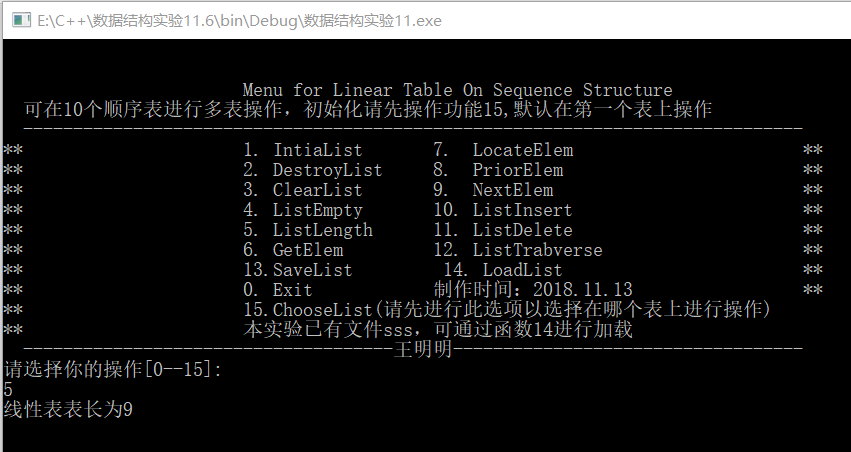
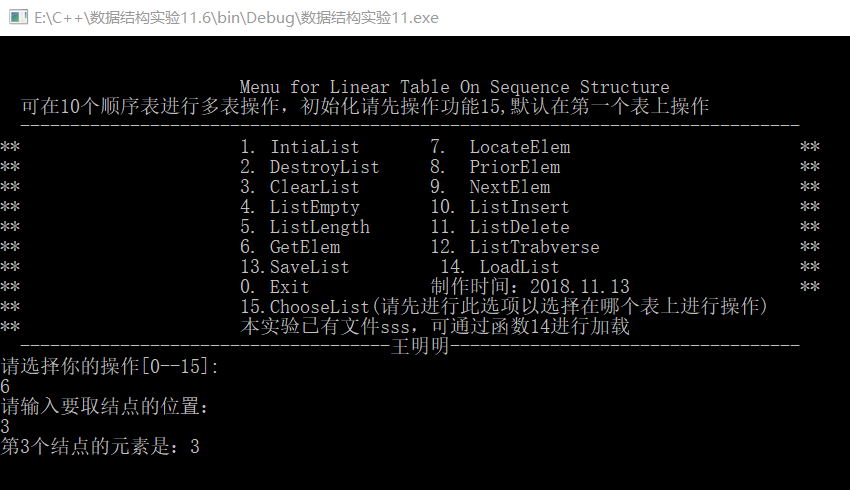


图1-17 ListLength测试截图

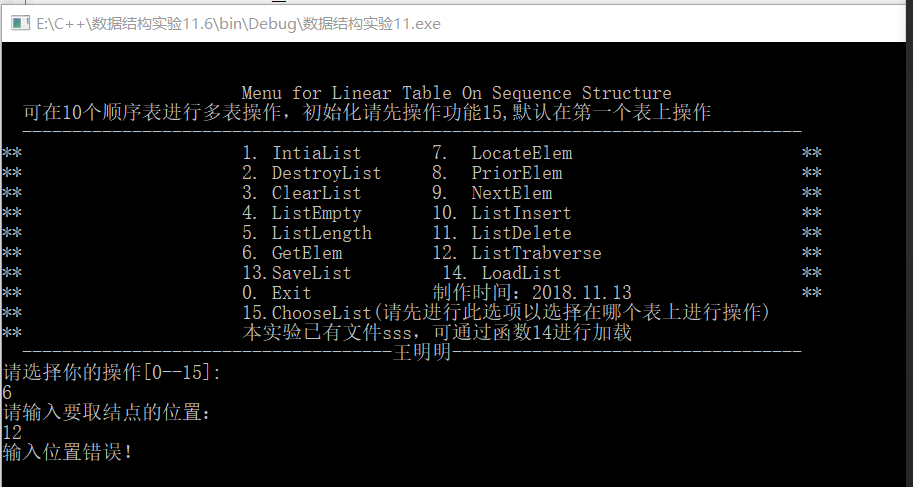
(3) GetElem测试

表1-3 GetElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选6  输入位置3 | 第3个节点的元素是：3 | 第3个节点的元素是：3 |
| sss | 界面选6  输入位置12 | 输入位置错误！ | 输入位置错误！ |
| 若表不存在 | 界面选6 | 线性表不存在！ | 线性表不存在！ |



(a)选取节点正确时运行截图



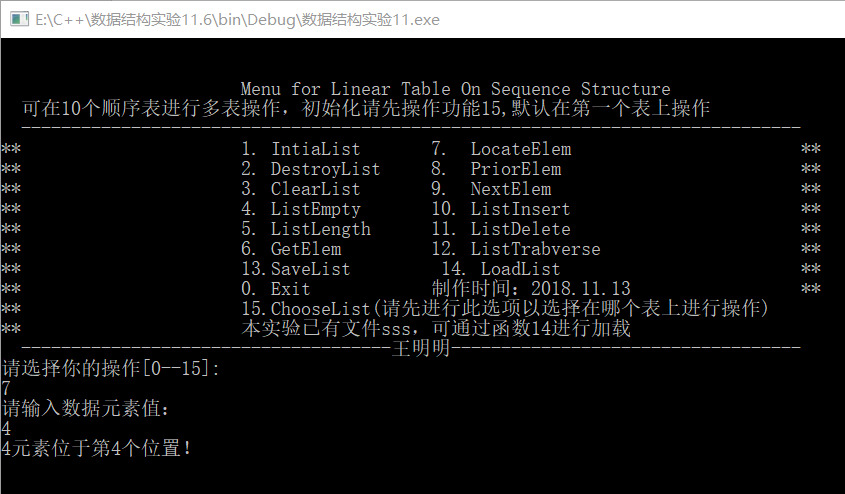
(b)输入位置错误时运行截图

图1-18 GetElem测试截图

(4) LocateElem测试

表1-4 LocateElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选7  输入元素4 | 4元素位于第4个位置 | 4元素位于第4个位置 |
| sss | 界面选7  输入元素20 | 该元素不存在！ | 该元素不存在！ |
| 若表不存在 | 界面选7 | 线性表不存在！ | 线性表不存在！ |



(a)元素位于顺序表中运行截图



(b)输入元素不在顺序表中运行截图

图1-19 LocateElem测试截图

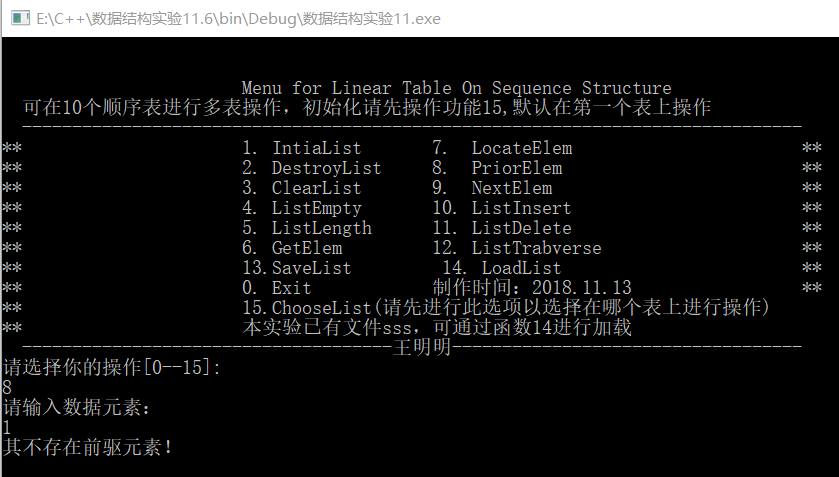
(5) PriorElem测试

表1-5 PriorElem测试用例表

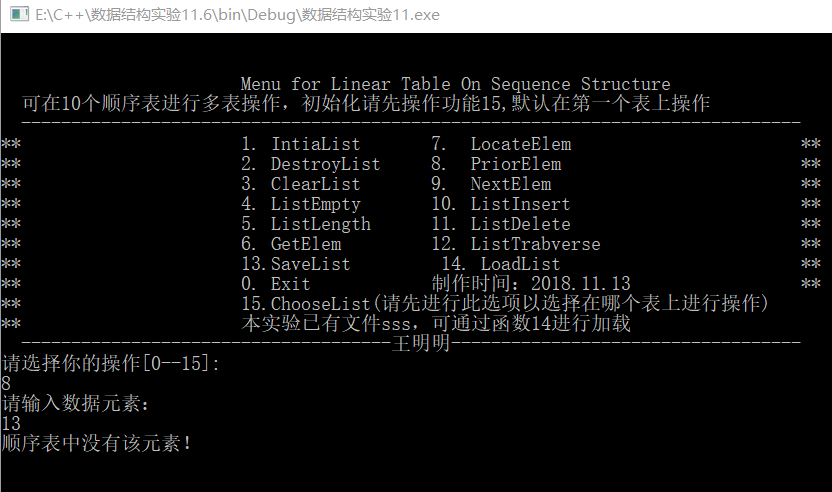
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选8  输入元素2 | 其前驱元素为：1 | 其前驱元素为：1 |
| sss | 界面选8  输入元素1 | 其不存在前驱元素！ | 其不存在前驱元素！ |
| sss | 界面选8  输入元素13 | 顺序表中没有该元素！ | 顺序表中没有该元素！ |
| 若表不存在 | 界面选8 | 线性表不存在！ | 线性表不存在！ |



(a)输入元素存在且不是第一个节点



(b)输入元素为第一个节点



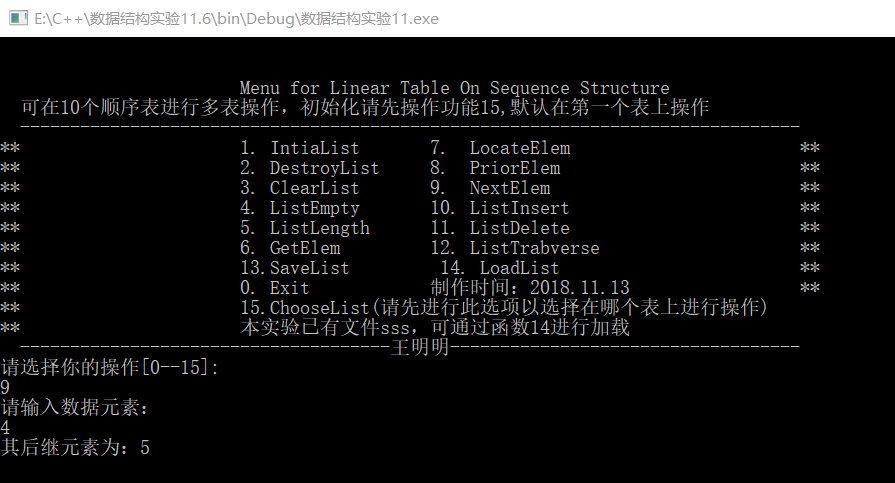
(c)输入元素不在顺序表中时

图1-20 LocateElem测试截图

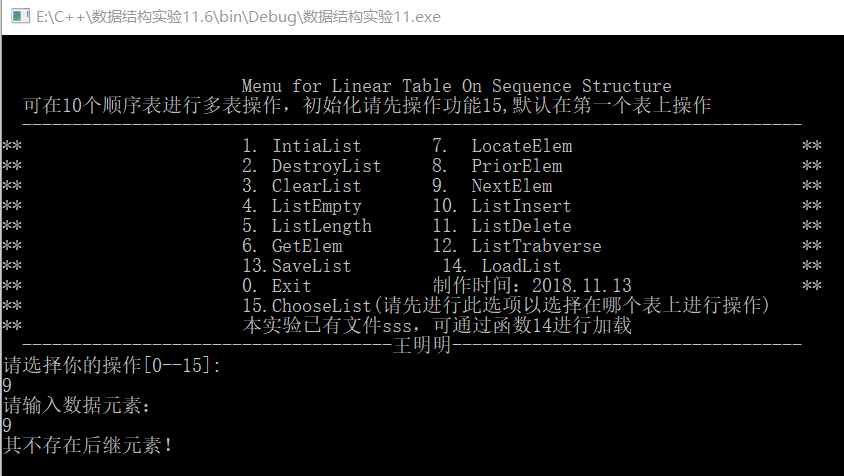
(6) NextElem测试

表1-6 NextElem测试用例表

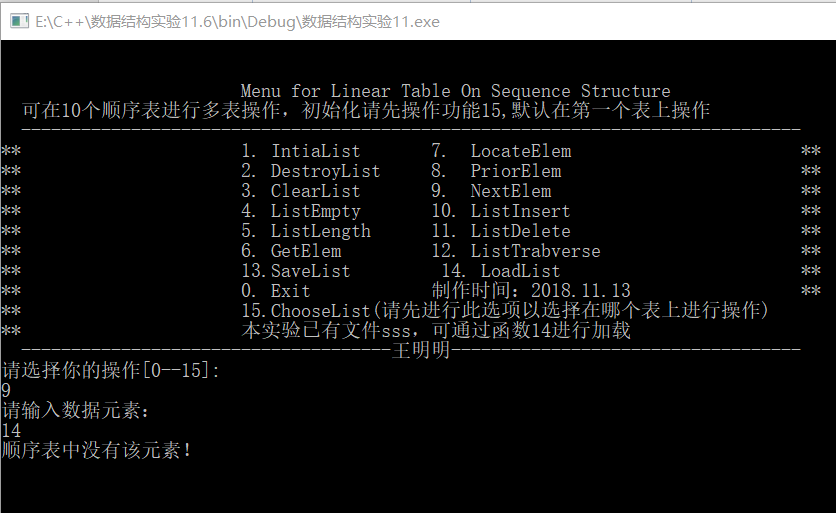
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选9  输入元素4 | 其后继元素为：5 | 其后继元素为：5 |
| sss | 界面选9  输入元素9 | 其不存在后继元素 | 其不存在后继元素 |
| sss | 界面选19  输入元素选14 | 顺序表中没有该元素！ | 顺序表中没有该元素！ |
| 若表不存在 | 界面选9 | 线性表不存在！ | 线性表不存在！ |



(a)输入元素存在且不是最后一个节点



(b)输入元素为最后一个节点



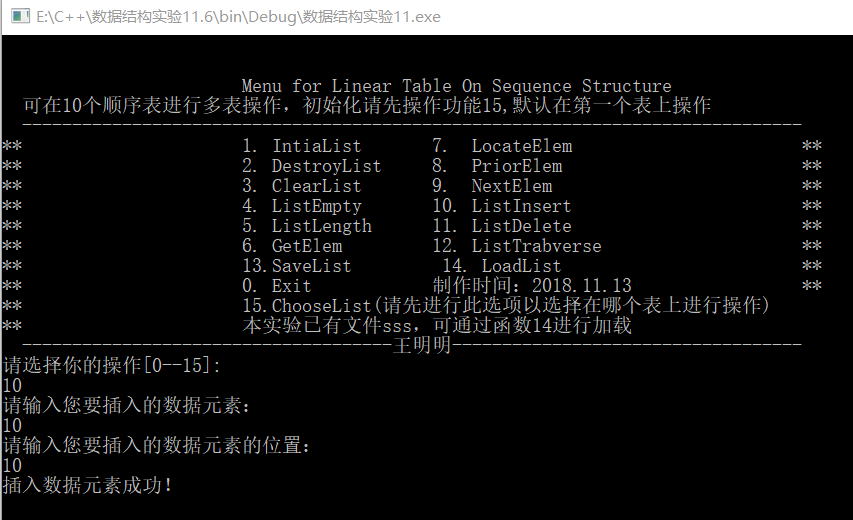
(c)输入元素不存在

图1-21 NextElem测试截图

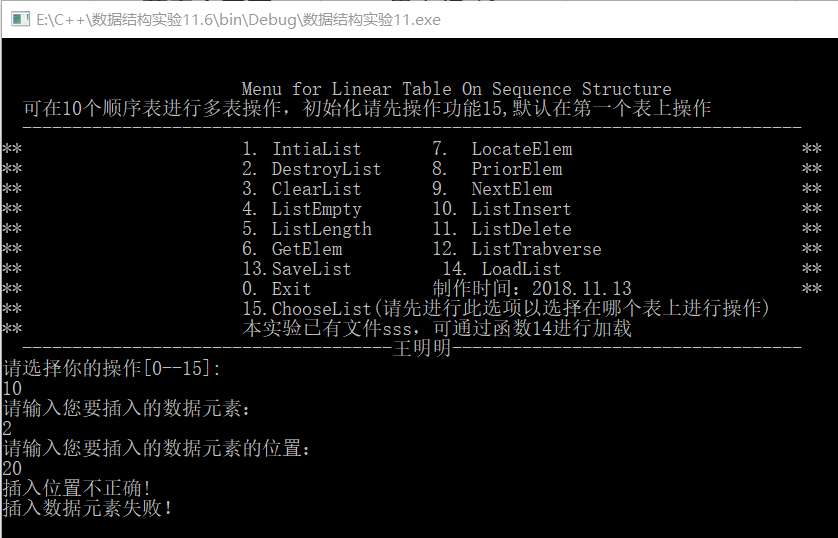
(7) ListInsert测试

表1-7 ListInsert测试用例表

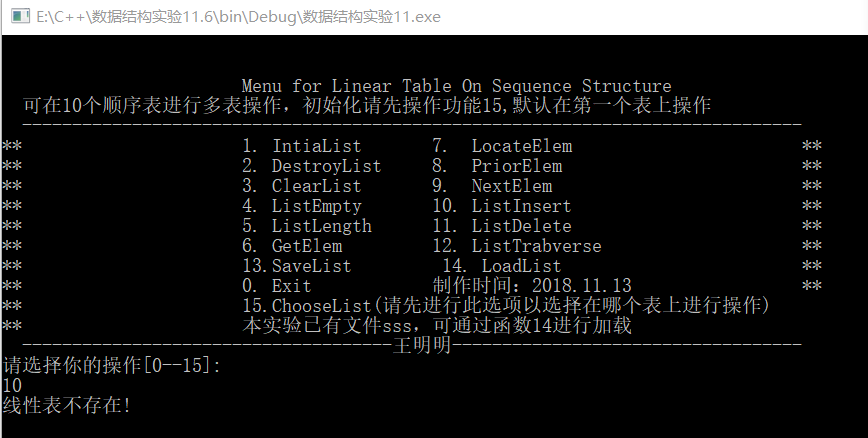
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选10  输入元素10  输入位置10 | 插入数据元素成功！ | 插入数据元素成功！ |
| sss | 界面选10  输入元素2  插入位置20 | 插入位置不正确！  插入数据元素失败！ | 插入位置不正确！  插入数据元素失败！ |
| 若表不存在 | 界面选10 | 线性表不存在！ | 线性表不存在！ |



(a)插入位置正确时截图



(b)插入元素位置不正确时截图



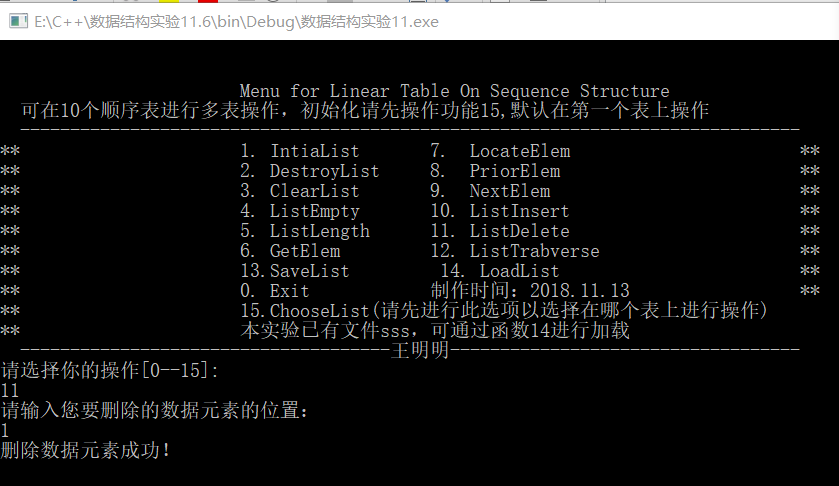
(c)线性表不存在时截图

图1-22 ListInsert测试截图

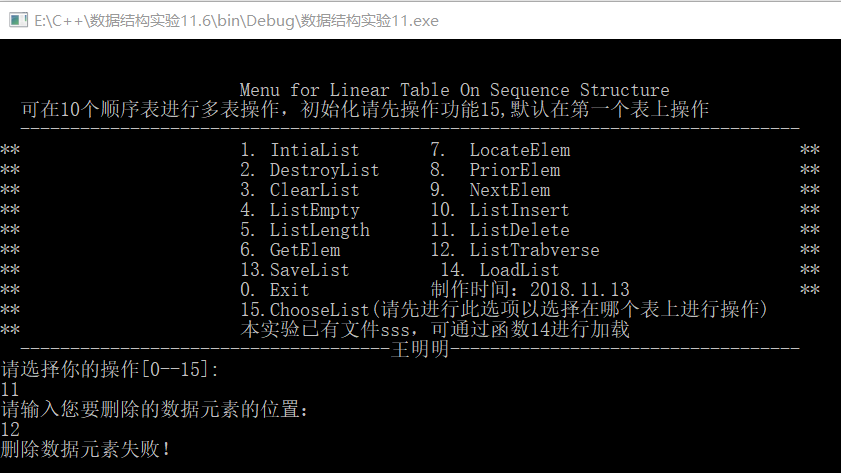
(8) ListDelete测试

表1-8 ListDelete测试用例表

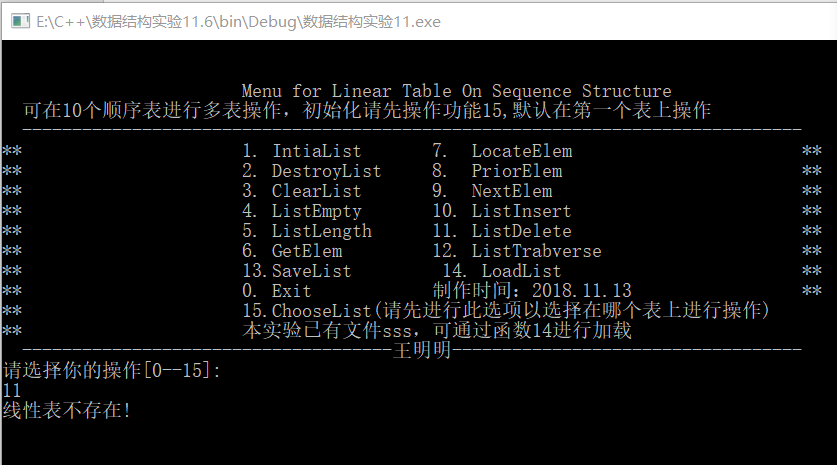
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 选11、输入1 | 删除元素成功！ | 删除元素成功！ |
| sss | 选11、输入12 | 删除元素失败！ | 删除元素失败！ |
| 若表不存在 | 选11 | 线性表不存在！ | 线性表不存在！ |



(a)正常删除元素运行截图



(b) 删除位置不正确运行截图



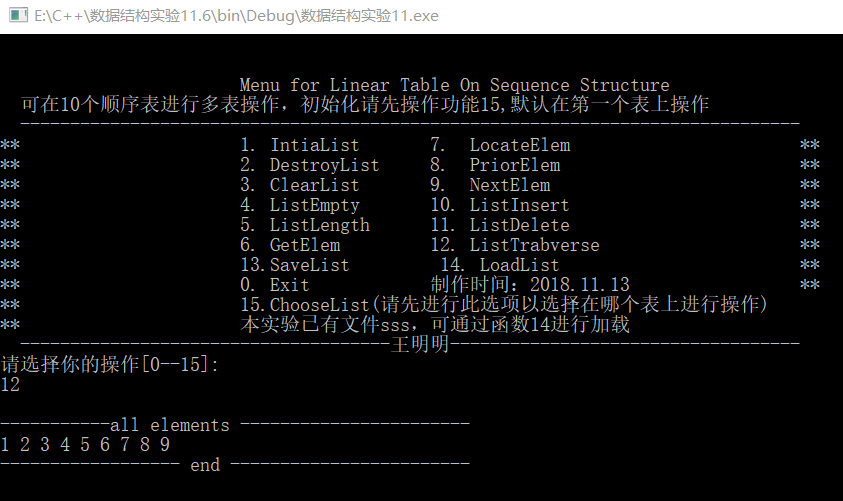
(c)线性表不存在运行截图

图1-23 ListDelete测试截图

(9) ListTrabverse测试

表1-9 ListTrabverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 测试结果 |
| sss | 主界面选12 | 1 2 3 4 5 6 7 8 9 | 1 2 3 4 5 6 7 8 9 |
| 不存在的表 | 主界面选12 | 线性表不存在 | 线性表不存在 |



(a)使用测试用例运行截图



(b)线性表不存在运行截图

图1-24 ListTrabverse测试截图

## 1.4 实验小结

本次实验主要内容是关于线性表的练习，由于实验之前老师已给出基础框架，只需对实验中要求的函数进行补充，这减小了我们的学习压力，更能突出对课程内容的考查与训练。

本次实验使我获益匪浅，加深了我对线性表的理解，也使我发现了自身很多的不足之处。首先，对C语言这一基本工具掌握不足，不能够熟练地使用这一语言，从而也导致了实验内容出现bug时无从下手。其次，做实验之前应该好好看书，书上有着插入，删除等函数的算法，而自己却没看，导致在第一次交予助教检查时便出了错。关于文件的保存和加载函数，自己掌握不够清楚，还要再看看C语言中的第十章文件。

在第一次提交之后，老师和助教给出了修改建议，比如流程图使用visio矢量图，我在老师的帮助下学会了如何使用visio做图。

本次实验锻炼了我们理论与实践结合的能力，更加深了我们对于数据结构这门课程的学习，从这次实验中，我学到了很多。最后，我由衷地感谢老师、助教和同学在本次实验中对我的帮助，帮助我解决实验中遇到的难题，谢谢！

# 

# 2 基于链式存储结构的线性表实现

**2.1 问题描述**

构造顺序表，呈现一个简易菜单的功能演示系统，该演示系统可选择实现多个线性表管理。需要在主程序中完成函数调用以及所需实参值和函数执行结果的输出。定义线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等函数，并给出适当的操作提示，并且可选择以文件的形式进行存储和加载。

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。

⑴初始化表：函数名称是InitaList(L)；初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。

⑵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。

⑶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。

⑷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。

⑸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。

⑹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。

⑺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。

⑻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。

⑼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。

⑽插入元素：函数名称是ListInsert(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

⑿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()。

实验目的是：通过实验加深对线性表的概念、基本运算的理解；能够熟练掌握线性表的逻辑结构与物理结构的关系；物理结构采用单链表,熟练掌握线性表的基本运算的实现。

## 2.2 系统设计

**2.2.1系统总体设计**

本系统提供一个顺序存储的线性表，一个简易的菜单。

菜单可选择的操作有：初始化线性表、销毁表、清空表、判空表，求表长、得到某元素、查找元素、获得某元素的前驱、获得某元素的后继、插入元素、删除元素、遍历线性表、加载预置文件、多表之间切换等。

**2.2.2有关常量和类型定义**

数据元素类型的定义：

typedef int status;

typedef int ElemType;

有关常量的定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

**2.2.3算法设计**

(1)

函数名称：InitaList(L)；

初始条件：线性表L已存在；

操作结果：构造一个空的线性表。

算法思路：分配存储空间，将表数据域初始化为0，再将表指针域初始化为空，图2-1 为InitaList(L)的流程图。



图2-1 InitaList(L)流程图

(2)

函数名称：DestroyList(L)；

初始条件：线性表L已存在；

操作结果：销毁线性表L。

算法思路：依次释放单链表每一个结点并将数据设置为初值，图2-2 为DestroyList (L)的流程图。



图2-2 DestroyList (L)流程图

(3)

函数名称：ClearList(L)；

初始条件：线性表L已存在

操作结果：将L重置为空表。

算法思路：依次释放单链表每一个结点并将自身指针域指向空，图2-3 为ClearList(L)的流程图。



图2-3 ClearList(L)流程图

(4)

函数名称：ListEmpty(L)；

初始条件：线性表L已存在

操作结果：若L为空表则返回TRUE,否则返回FALSE。

算法思路：L->next存在则非空，否则为空，图2-4为 ListEmpty(L)的流程图。



图2-4 ListEmpty(L)流程图

(5)

函数名称：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。

算法思路：遍历链表，用计数器i统计元素个数，图2-5 为ListLength(L)的流程图。



图2-5 ListLength(L)流程图

(6)

函数名称：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。

算法思路：遍历链表，用计数器j判断是否到第i个位置，如果到达则返回该元素，图2-6为 GetElem(L,i,e)的流程图。



图2-6 GetElem(L,i,e)流程图

(7)

函数名称：LocateElem(L,e,compare())；

初始条件：线性表已存在；

算法思路：遍历顺序表，用计数器j表示元素的位序，将与给定元素e满足相等关系的元素的位序j返回即可，图2-7 为LocateElem(L,e,compare())的流程图。



图2-7 LocateElem(L,e,compare())流程图

(8)

函数名称：PriorElem(L,cur\_e,pre\_e)；

初始条件：线性表L已存在；

算法思路：如果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。算法流程图为图2-8 PriorElem(L,cur\_e,pre\_e)。



图2-8 PriorElem(L,cur\_e,pre\_e)流程图

(9)

函数名称：NextElem(L,cur\_e,next\_e)；

初始条件：线性表L已存在；

算法思路：遍历该顺序表。若找到该结点，并且该结点有后继元素，则将前驱元素赋值给next\_e。如果未找到该结点，或者找到该结点但该结点不存在后继元素，则返回FALSE。算法流程图为图2-9 NextElem(L,cur\_e,next\_e)。



图2-9 NextElem(L,cur\_e,next\_e)流程图

(10)

函数名称：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。

算法思路：遍历链表，用计数器j判断是否到第i个位置，若到达则插入该元素。若遍历完链表后j仍小于i则返回FALSE。算法流程图为图2-10 ListInsert(L,i,e)。



图2-10 ListInsert(L,i,e)流程图

(11)

函数名称：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。

算法思路：遍历链表，用计数器j判断是否到第i个位置，若到达则删除该元素并用e返回其值同时释放该结点。若遍历完链表后j仍小于i则返回FALSE。算法流程图为图2-11 ListDelete(L,i,e)。



图2-11 ListDelete(L,i,e)流程图

(12)

函数名称：ListTrabverse(L)；

初始条件：线性表L已存在；

操作结果：依次遍历L的每个数据元素。

算法思路：如果线性表L存在，则遍历元素；否则返回ERROR。图2-12 为ListTrabverse(L)的流程图。



图2-12 ListTrabverse(L)流程图

(13)

函数名称：SaveList(L,filename)；

初始条件：线性表L已存在；

操作结果：将线性表L保存为文件形式。

算法思路：用fwrite保存为文件，图2-13为SaveList(L,filename)的流程图。



图2-13 SaveList(L,filename)流程图

(14)

函数名称：LoadList (L)；

初始条件：文件filename已存在；

操作结果：将线性表L以文件形式加载读取。

算法思路：用fread将文件读取顺序表，图2-14为 LoadList (L)的流程图。



图2-14 LoadList (L)流程图

## 2.3 系统实现

**2.3.1程序源代码**

见《附录B 基于链式线性存储结构实现的源程序》。

**2.3.2系统测试**

程序采用简易界面，如图1-15所示，挑选ListEmpty, ListLength, GetElem, LocateElem, PriorElem, NextElem, ListInsert, ListDelete, ListTrabverse这些重要功能进行测试。

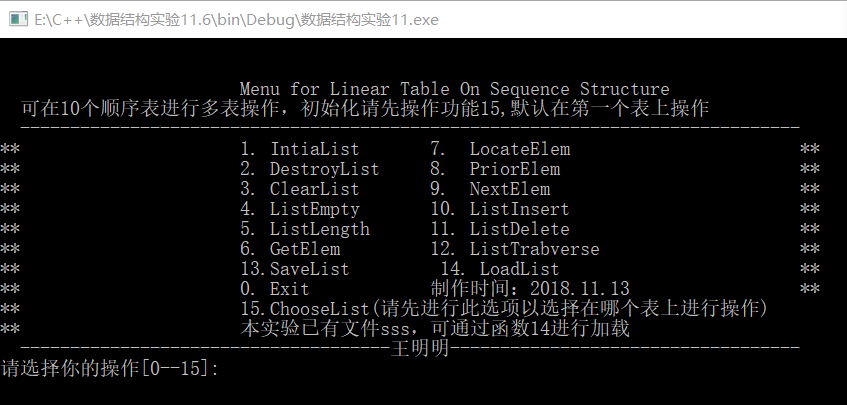


图2-15 程序简易界面截图

测试用例为：sss{1,2,3,4,5,6,7,8,9},null(空表)以及顺序表不存在。

(1)ListEmpty测试

表2-1 ListEmpty测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选4 | 线性表不是空表！ | 线性表不是空表！ |
| null | 界面选4 | 文件为空！ | 文件为空！ |
| 若表不存在 | 界面选4 | 线性表不存在！ | 线性表不存在！ |



图2-16 ListEmpty测试截图

(2) ListLength测试

表2-2 ListLength测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选5 | 线性表表长为9 | 线性表表长为9 |
| null | 界面选5 | 线性表表长为0 | 线性表表长为0 |
| 若表不存在 | 界面选5 | 线性表不存在！ | 线性表不存在！ |

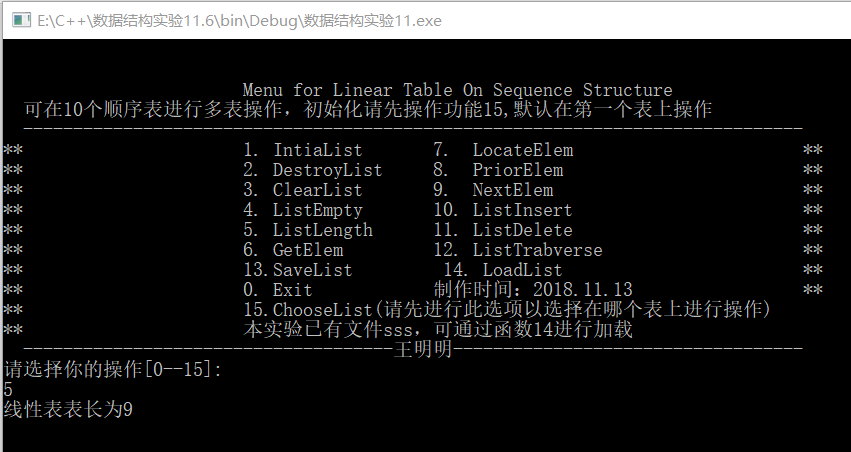
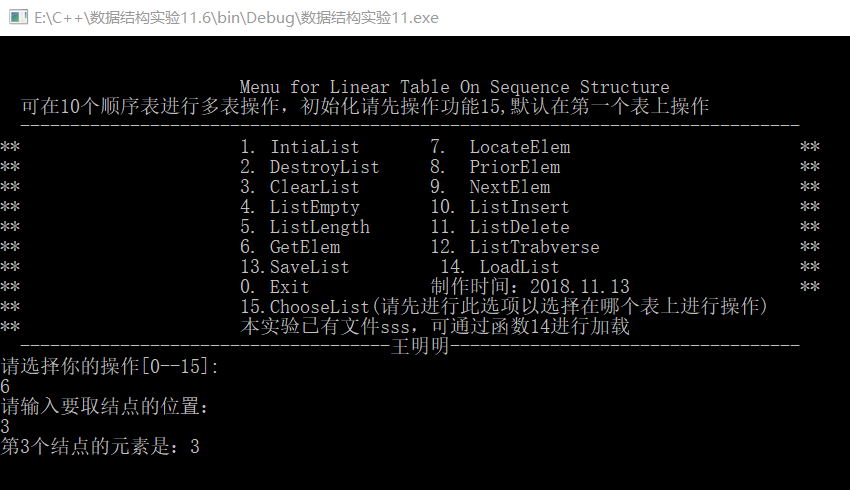


图2-17 ListLength测试截图

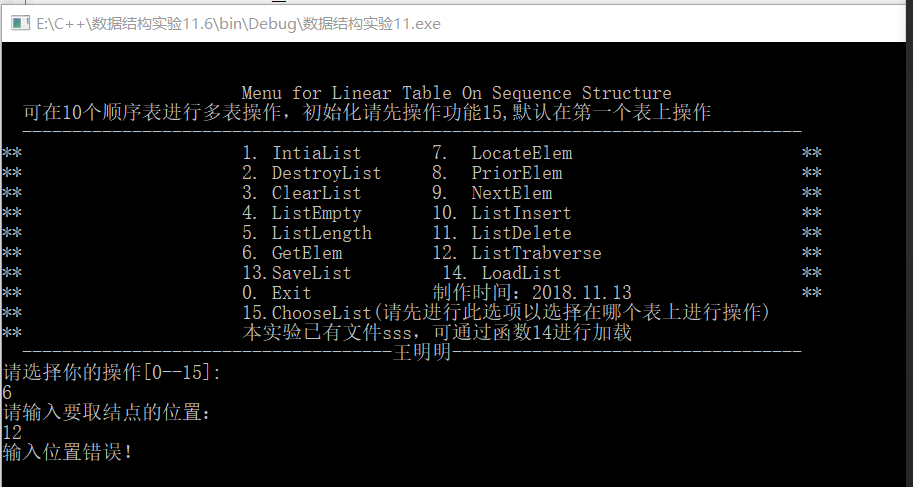
(3) GetElem测试

表2-3 GetElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选6  输入位置3 | 第3个节点的元素是：3 | 第3个节点的元素是：3 |
| sss | 界面选6  输入位置12 | 输入位置错误！ | 输入位置错误！ |
| 若表不存在 | 界面选6 | 线性表不存在！ | 线性表不存在！ |



(a)选取节点正确时运行截图



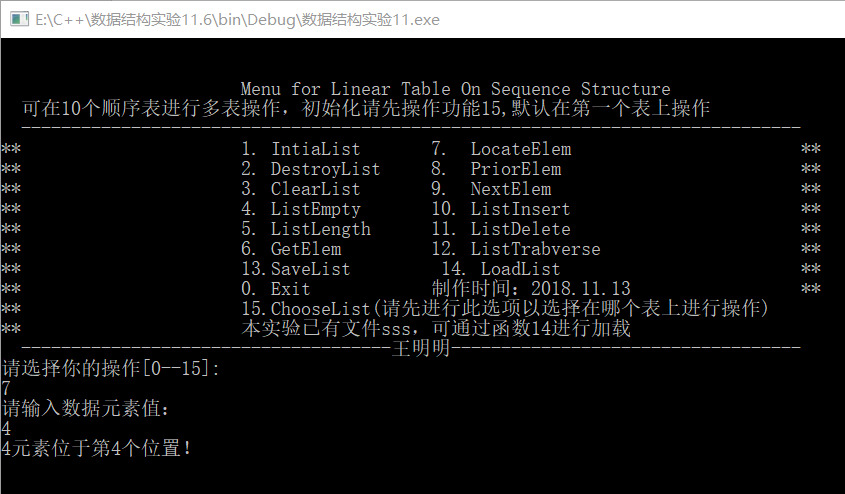
(b)输入位置错误时运行截图

图2-18 GetElem测试截图

(4) LocateElem测试

表2-4 LocateElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选7  输入元素4 | 4元素位于第4个位置 | 4元素位于第4个位置 |
| sss | 界面选7  输入元素20 | 该元素不存在！ | 该元素不存在！ |
| 若表不存在 | 界面选7 | 线性表不存在！ | 线性表不存在！ |



(a)元素位于顺序表中运行截图



(b)输入元素不在顺序表中运行截图

图2-19 LocateElem测试截图

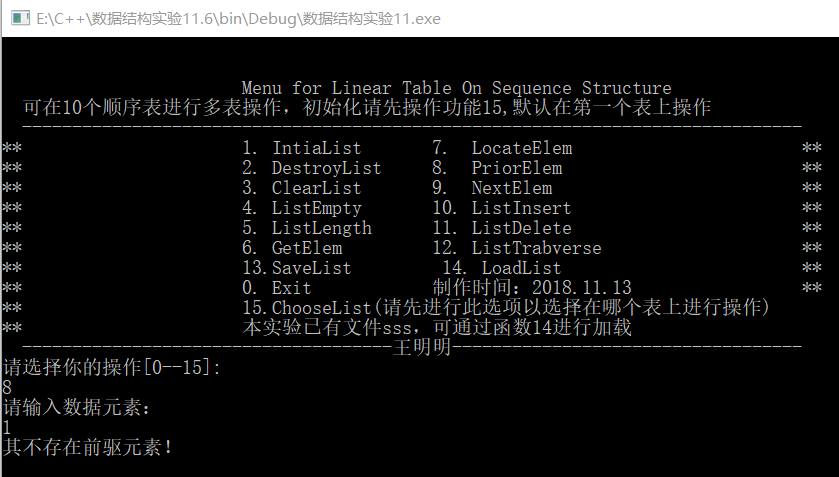
(5) PriorElem测试

表2-5 PriorElem测试用例表

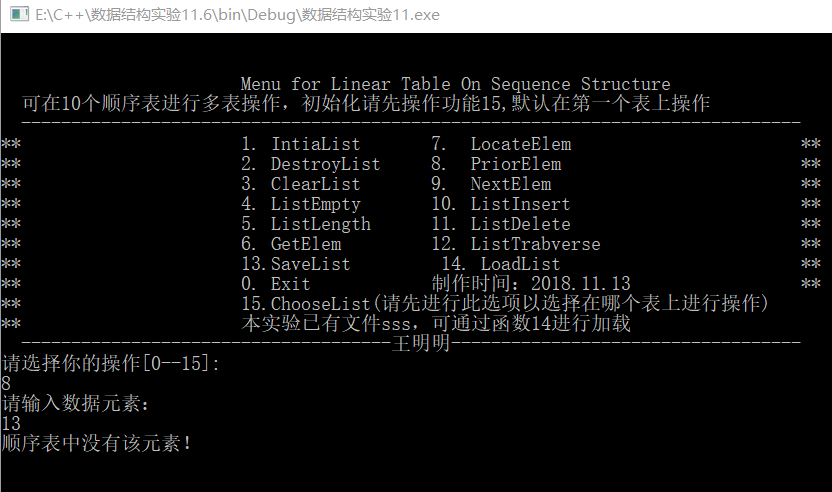
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选8  输入元素2 | 其前驱元素为：1 | 其前驱元素为：1 |
| sss | 界面选8  输入元素1 | 其不存在前驱元素！ | 其不存在前驱元素！ |
| sss | 界面选8  输入元素13 | 顺序表中没有该元素！ | 顺序表中没有该元素！ |
| 若表不存在 | 界面选8 | 线性表不存在！ | 线性表不存在！ |



(a)输入元素存在且不是第一个节点



(b)输入元素为第一个节点



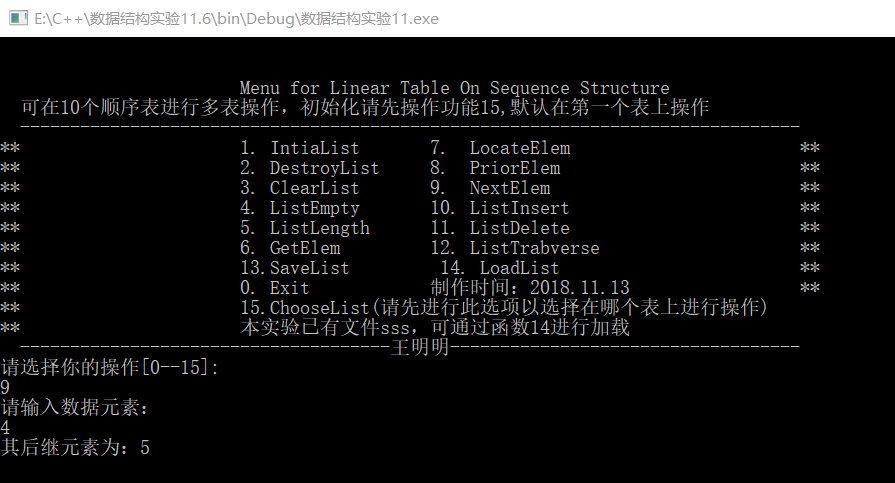
(c)输入元素不在顺序表中时

图2-20 LocateElem测试截图

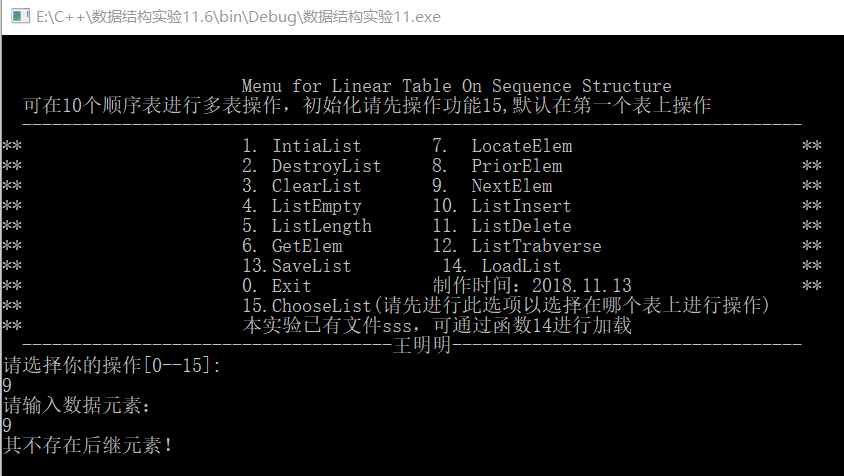
(6) NextElem测试

表2-6 NextElem测试用例表

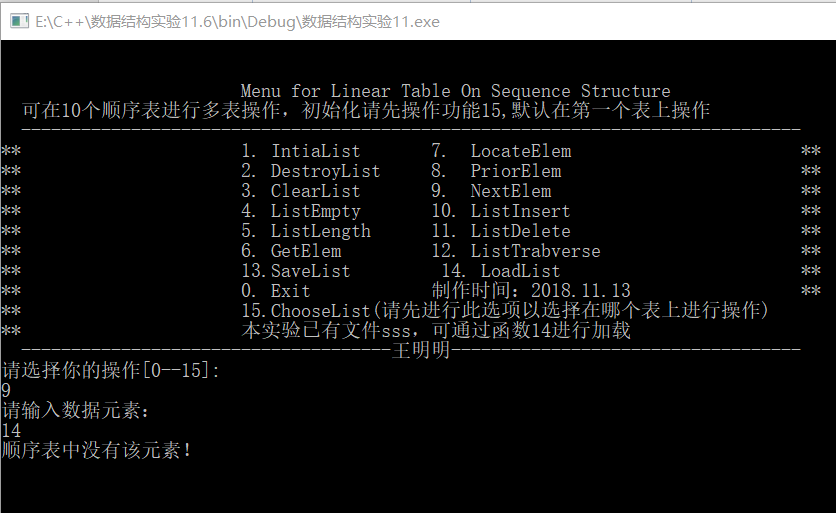
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选9  输入元素4 | 其后继元素为：5 | 其后继元素为：5 |
| sss | 界面选9  输入元素9 | 其不存在后继元素 | 其不存在后继元素 |
| sss | 界面选19  输入元素选14 | 顺序表中没有该元素！ | 顺序表中没有该元素！ |
| 若表不存在 | 界面选9 | 线性表不存在！ | 线性表不存在！ |



(a)输入元素存在且不是最后一个节点



(b)输入元素为最后一个节点



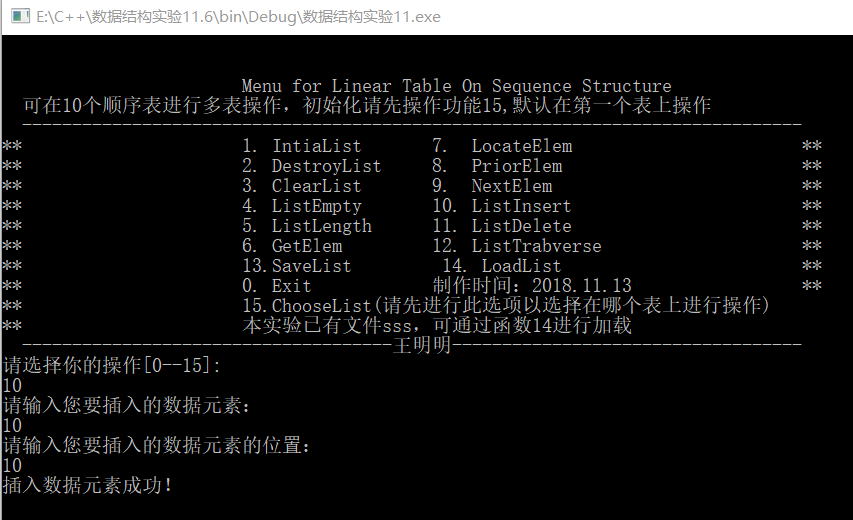
(c)输入元素不存在

图2-21 NextElem测试截图

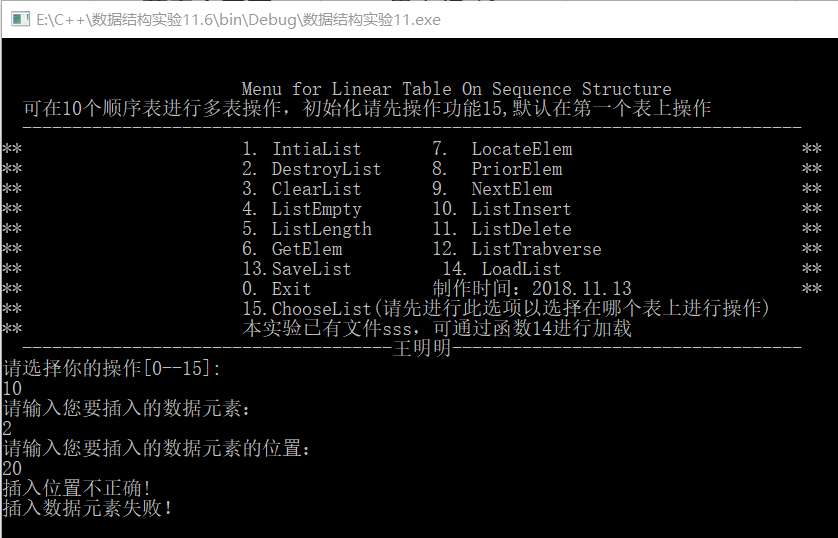
(7) ListInsert测试

表2-7 ListInsert测试用例表

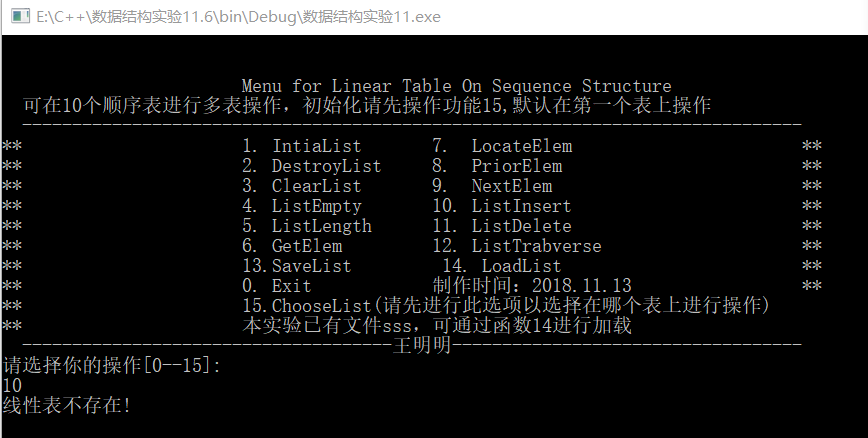
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选10  输入元素10  输入位置10 | 插入数据元素成功！ | 插入数据元素成功！ |
| sss | 界面选10  输入元素2  插入位置20 | 插入位置不正确！  插入数据元素失败！ | 插入位置不正确！  插入数据元素失败！ |
| 若表不存在 | 界面选10 | 线性表不存在！ | 线性表不存在！ |



(a)插入位置正确时截图



(b)插入元素位置不正确时截图



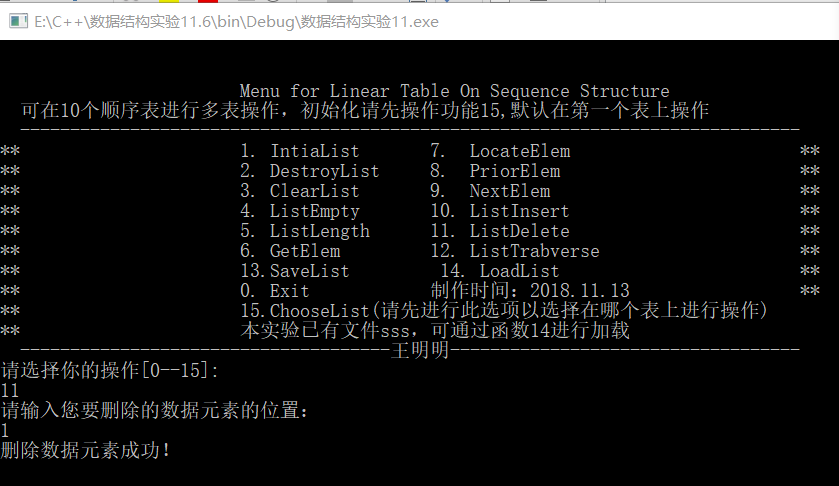
(c)线性表不存在时截图

图2-22 ListInsert测试截图

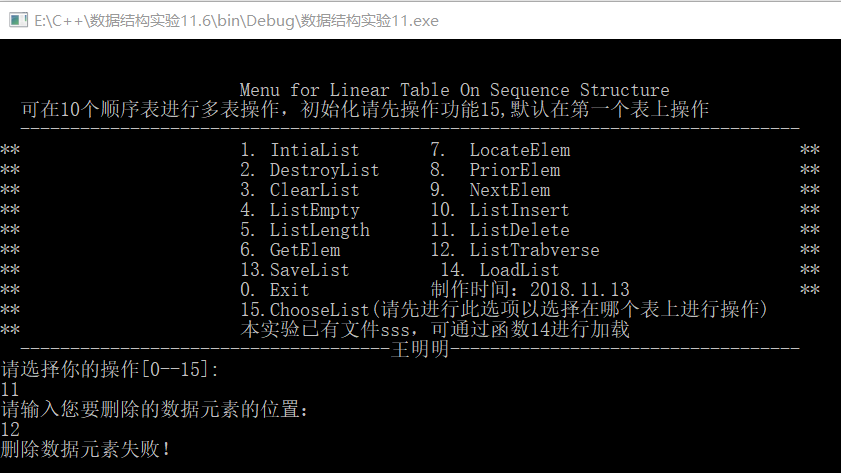
(8) ListDelete测试

表2-8 ListDelete测试用例表

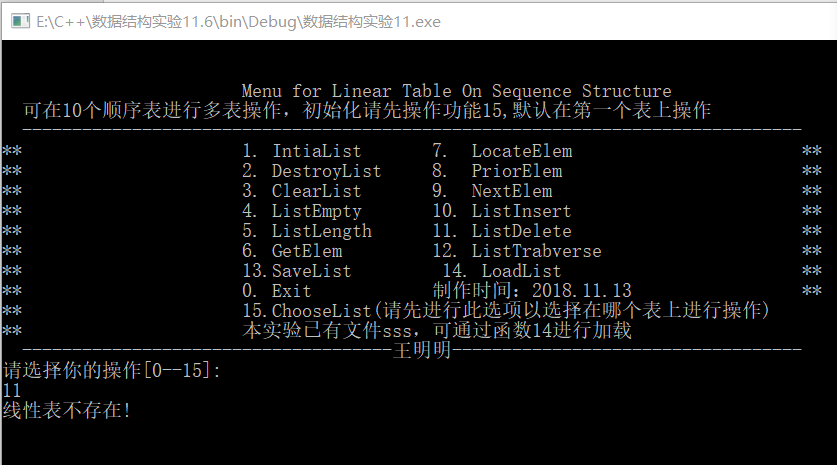
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 选11、输入1 | 删除元素成功！ | 删除元素成功！ |
| sss | 选11、输入12 | 删除元素失败！ | 删除元素失败！ |
| 若表不存在 | 选11 | 线性表不存在！ | 线性表不存在！ |



(a)正常删除元素运行截图



(b) 删除位置不正确运行截图



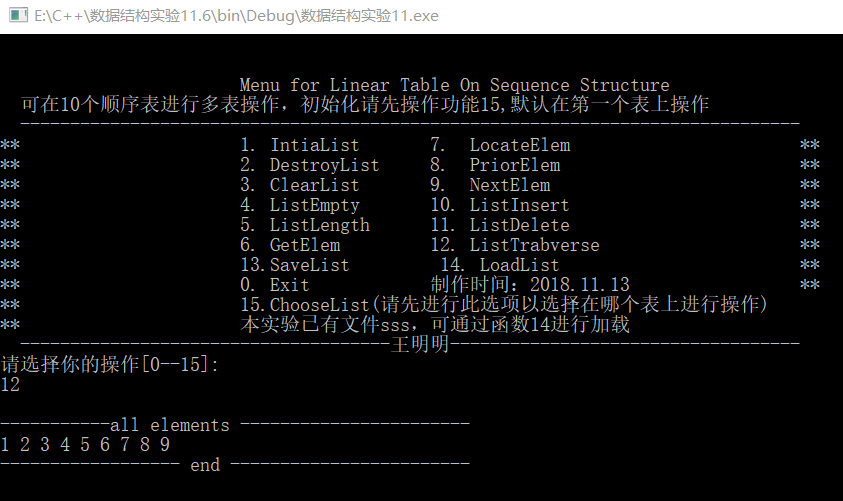
(c)线性表不存在运行截图

图2-23 ListDelete测试截图

(9) ListTrabverse测试

表2-9 ListTrabverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 测试结果 |
| sss | 主界面选12 | 1 2 3 4 5 6 7 8 9 | 1 2 3 4 5 6 7 8 9 |
| 不存在的表 | 主界面选12 | 线性表不存在 | 线性表不存在 |



(a)使用测试用例运行截图



(b)线性表不存在运行截图

图2-24 ListTrabverse测试截图

## 2.4 实验小结

因为第二次实验与第一次实验相比相差不大，仅仅是物理结构发生变化，仅需要做出相应的函数修改即可，因此减轻了一些负担。基本框架不变，从而突出了对单链表的训练，深了对线性表的概念、基本运算的理解，熟练了线性表的逻辑结构与物理结构的关系，采用了单链表的物理结构,熟练了线性表的基本运算的实现。

在第一次实验的基础上，我对函数进行修改，之后进行测试，最后完善，将注释部分进行完善。本次实验帮助我巩固了对线性表的操作，熟练了单链表的用法，收获很大。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

通过二叉链表作为二叉树的物理结构，创建一个简易的菜单演示系统。该系统定义了初始化二叉树、销毁二叉树、创建二叉树、清空二叉树、判定空二叉树、求二叉树深度、获得结点、插入子树、删除子树和遍历等运算对应的函数，并给出了相应的操作提示，并可选择以文件的形式进行存储和加载。

依据最小完备性和常用性相结合的原则，以函数形式定义了二叉树的初始化二叉树、销毁二叉树、创建二叉树、清空二叉树、判定空二叉树和求二叉树深度等20种基本运算，具体运算功能定义如下。

⑴初始化二叉树：函数名称是InitBiTree(T)；初始条件是二叉树T不存在；操作结果是构造空二叉树T。

⑵销毁二叉树：树函数名称是DestroyBiTree(T)；初始条件是二叉树T已存在；操作结果是销毁二叉树T。

⑶创建二叉树：函数名称是CreateBiTree(T,definition)；初始条件是definition 给出二叉树T的定义；操作结果是按definition构造二叉树T。

⑷清空二叉树：函数名称是ClearBiTree (T)；初始条件是二叉树T存在； 操作结果是将二叉树T清空。

⑸判定空二叉树：函数名称是BiTreeEmpty(T)；初始条件是二叉树T存在；操作结果是若T为空二叉树则返回TRUE，否则返回FALSE。

⑹求二叉树深度：函数名称是BiTreeDepth(T)；初始条件是二叉树T存在；操作结果是返回T的深度。

⑺获得根结点：函数名称是Root(T)；初始条件是二叉树T已存在；操作结果是返回T的根。

⑻获得结点：函数名称是Value(T,e)；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是返回e的值。

⑼结点赋值：函数名称是Assign(T,&e,value)；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是结点e赋值为value。

⑽获得双亲结点：函数名称是Parent(T,e) ；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是若e是T的非根结点，则返回它的双亲结点指针，否则返回NULL。

⑾获得左孩子结点：函数名称是LeftChild(T,e)；初始条件是二叉树T存在，e是T中某个节点；操作结果是返回e的左孩子结点指针。若e无左孩子，则返回NULL。

⑿获得右孩子结点：函数名称是RightChild(T,e)；初始条件是二叉树T已存在，e是T中某个结点；操作结果是返回e的右孩子结点指针。若e无右孩子，则返回NULL。

⒀获得左兄弟结点：函数名称是LeftSibling(T,e)；初始条件是二叉树T存在，e是T中某个结点；操作结果是返回e的左兄弟结点指针。若e是T的左孩子或者无左兄弟，则返回NULL。

⒁获得右兄弟结点：函数名称是RightSibling(T,e)；初始条件是二叉树T已存在，e是T中某个结点；操作结果是返回e的右兄弟结点指针。若e是T的右孩子或者无有兄弟，则返回NULL。

⒂插入子树：函数名称是InsertChild(T,p,LR,c)；初始条件是二叉树T存在，p指向T中的某个结点，LR为0或1，,非空二叉树c与T不相交且右子树为空；操作结果是根据LR为0或者1，插入c为T中p所指结点的左或右子树，p 所指结点的原有左子树或右子树则为c的右子树。

⒃删除子树：函数名称是DeleteChild(T.p.LR)；初始条件是二叉树T存在，p指向T中的某个结点，LR为0或1。 操作结果是根据LR为0或者1，删除c为T中p所指结点的左或右子树。

⒄前序遍历：函数名称是PreOrderTraverse(T,Visit())；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果：先序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒅中序遍历：函数名称是InOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是中序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒆后序遍历：函数名称是PostOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是后序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒇按层遍历：函数名称是LevelOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是层序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

实验目的为：

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

**3.2.1系统总体设计**

该系统提供一个物理结构为二叉链表的二叉树。

本简易演示系统可提供的操作：初始化二叉树、销毁二叉树、创建二叉树、清空二叉树、判定空二叉树、求二叉树深度、获得结点、插入子树、删除子树和遍历等24个操作。

**3.2.2有关常量和类型定义**

数据元素类型的定义：

typedef int status;

typedef char TElemType;

有关常量的定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

**3.2.3算法设计**

(1)

函数名称：InitBiTree(T)

初始条件：二叉树T不存在

操作结果：构造空二叉树T

设计思想：采用指针T指向二叉树的根结点，将T赋值为NULL完成二叉树的初始化。



图3-1 InitBiTree(T)

(2)

函数名称：DestroyBiTree(T)

初始条件：二叉树T已存在

操作结果：销毁二叉树T

设计思想： 采取递归的方式先销毁二叉树的左子树，再销毁二叉树的右子树，最后用free函数释放掉根结点对应的内存空间。



图3-2 DestroyBiTree(T)

(3)

函数名称：CreateBiTree(T)

初始条件：二叉树T已存在

操作结果：构造二叉树T

设计思想：传入根节点指针，使用递归算法，由用户自行输入每个根节点的信息

(4)

函数名称：ClearBiTree (T)

初始条件：二叉树T存在

操作结果：将二叉树T清空

设计思想：将根结点的左右孩子指针置为空。



图3-3 ClearBiTree (T)

(5)

函数名称：BiTreeEmpty(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

设计思想：判断指向根结点的结构类型指针T是否为空。

(6)

函数名称：BiTreeDepth(T)

初始条件：二叉树T存在

操作结果：返回T的深度

设计思想：从根节点出发，树的深度等于左右子树深度较大值加1，若左右子树均为空，则该树深度为1；若根节点为空，则深度为0



图3-4 BiTreeDepth(T)

(7)

函数名称：Root(T)

初始条件：二叉树T存在

操作结果：返回T的根

设计思想：将求二叉树根结点写成函数，传入头结点，若其头指针不为空，则返回该结点的data。

(8)

函数名称：Value(T,e)

初始条件：二叉树T已存在，e是T中的某个结点

操作结果：返回e的值

设计思想：通过递归先序遍历二叉树，即先比较根结点的key与给定是否一致，若一致，返回该结点的data，否则继续分别递归遍历其左子树和右子树。

(9)

函数名称：Assign(T,&e,value)

初始条件：二叉树T已存在，e是T中的某个结点

操作结果：结点e赋值为value

设计思想：根据输入的key获取到e结点，并对结点赋值。



图3-5 Assign(T,&e,value)

(10)

函数名称：Parent(T,e)

初始条件：二叉树T已存在，e是T中的某个结点

操作结果： 若e是T的非根结点，则返回它的双亲结点指针，否则返回NULL

设计思想：通过递归先序遍历二叉树，即先比较根结点的key与给定是否一致，若一致且该结点存在双亲，返回该结点的双亲，否则继续分别递归遍历其左子树和右子树。

(11)

函数名称：LeftChild(T,e)

初始条件： 二叉树T已存在，e是T中的某个结点

操作结果：返回e的左孩子结点指针。若e无左孩子，则返回NULL

设计思想：通过递归先序遍历二叉树，即先比较根结点的key与给定是否一致，若一致且该结点存在左孩子，返回该结点的左孩子，否则继续分别递归遍历其左子树和右子树。

(12)

函数名称：RightChild(T,e)

初始条件：二叉树T已存在，e是T中的某个结点

操作结果：返回e的右孩子结点指针。若e无右孩子，则返回NULL

设计思想：通过递归先序遍历二叉树，即先比较根结点的key与给定是否一致，若一致且该结点存在右孩子，返回该结点的右孩子，否则继续分别递归遍历其左子树和右子树。

(13)

函数名称：LeftSibling(T,e)

初始条件：二叉树T已存在，e是T中的某个结点

操作结果：返回e的左兄弟结点指针。若e是T的左孩子或者无左兄弟，则返回NULL

设计思想：通过递归先序遍历二叉树，即先比较根结点的key与给定是否一致，若一致且该结点存在左兄弟，返回该结点的左兄弟，否则继续分别递归遍历其左子树和右子树。

(14)

函数名称：RightSibling(T,e)

初始条件：二叉树T已存在，e是T中的某个结点

操作结果：返回e的右兄弟结点指针。若e是T的右孩子或者无有兄弟，则返回NULL

设计思想：通过递归先序遍历二叉树，即先比较根结点的key与给定是否一致，若一致且该结点存在右兄弟，返回该结点的右兄弟，否则继续分别递归遍历其左子树和右子树。

(15)

函数名称：InsertChild(T,p,LR,c)

初始条件：二叉树T存在，p指向T中的某个结点，LR为0或1，,非空二叉树c与T不相交且右子树为空

操作结果：根据LR为0或者1，插入c为T中p所指结点的左或右子树，p所指结点的原有左子树或右子树则为c的右子树

设计思想：通过之前的思想查找到要插入子树的结点，根据LR为0或者1，判断c为T中p所指结点的左或右子树，p所指结点的原有左子树或右子树则为c的右子树，再调用CreateBiTree(T)函数插入子树。

(16)

函数名称：DeleteChild(T.p.LR)

初始条件：二叉树T存在，p指向T中的某个结点，LR为0或1

操作结果：根据LR为0或者1，删除c为T中p所指结点的左或右子树

设计思想：通过之前的思想查找到要删除子树的结点，根据LR为0或者1，判断删除左或右子树，再调用DestroyBiTree(T)函数插入子树。

(17)

函数名称：PreOrderTraverse(T,Visit())

初始条件：二叉树T存在，Visit是对结点操作的应用函数

操作结果：先序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

设计思想：采用递归算法的思路，首先访问其根结点，对任一结点开始访问时都看作根结点，访问完根结点后若左孩子不为空，则按相同规则访问其左子树，访问完左子树后，再访问其右子树。

(18)

函数名称：InOrderTraverse(T,Visit())

初始条件：二叉树T存在，Visit是对结点操作的应用函数

操作结果：中序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

设计思想：采用递归算法的思路，首先访问其根结点的左子树，对任一结点开始访问时都看作根结点，访问完后若左孩子为空，则访问根结点，访问完根结点后，再访问其右子树。

(19)

函数名称：PostOrderTraverse(T,Visit())

初始条件：二叉树T存在，Visit是对结点操作的应用函数

操作结果：后序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

设计思想：采用递归算法的思路，首先访问其根结点的左子树，对任一结点开始访问时都看作根结点，访问完后若左孩子为空，则访问右子树，访问完右子树后，再访问其根结点。

(20)

函数名称：LevelOrderTraverse(T,Visit())

初始条件：二叉树T存在，Visit是对结点操作的应用函数

操作结果：层序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

设计思想：采用递归算法的思路，按层输入每一层的所有结点，最后先进先出，输出所有结点。

## 3.3 系统实现

**3.3.1程序源代码**

见《附录C 基于二叉链表二叉树实现的源程序》。

**3.3.2系统测试**

程序采用简易界面，如图3-1所示， BiTreeEmpty, BiTreeDepth, Root, Value, Assign, Parent, LeftChild, RightChild, LeftSibling, RightSibling, InsertChild, DeleteChild, PreOrderTraverse, InOrderTraverse, PostOrderTraverse, LevelOrderTra verse功能进行测试。

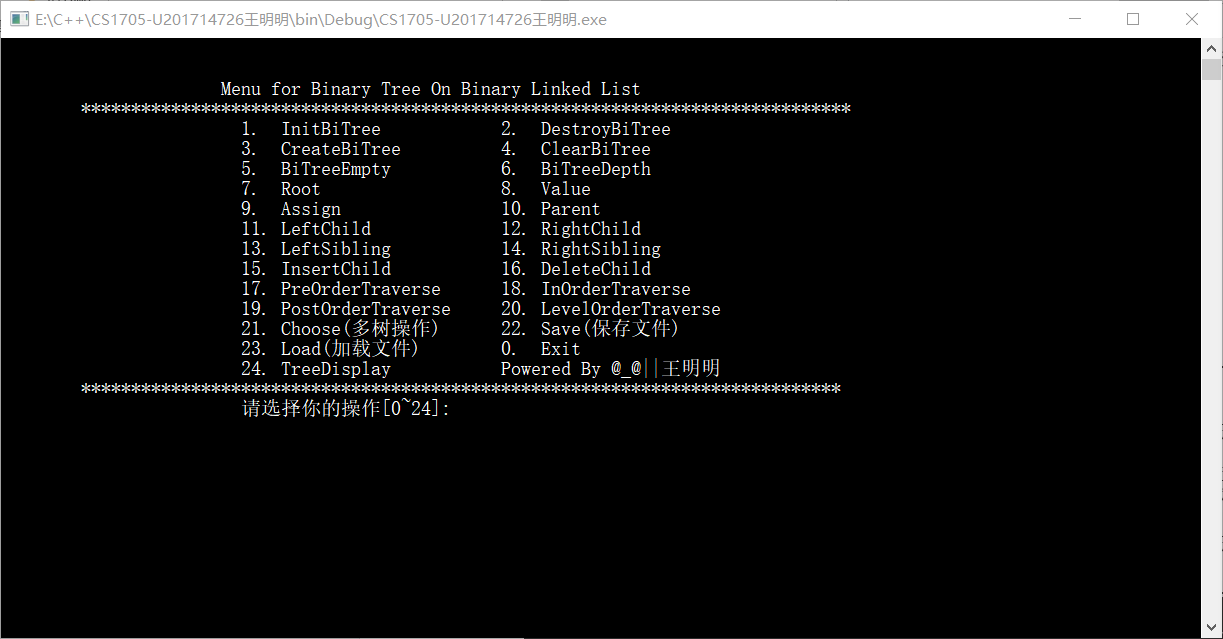


图3-6 程序简易界面截图

测试用例zzzh:{a,b,d,#,#,#,c,e,#,#,f,g,#,#,#}，二叉树不存在的情况共两个用例。

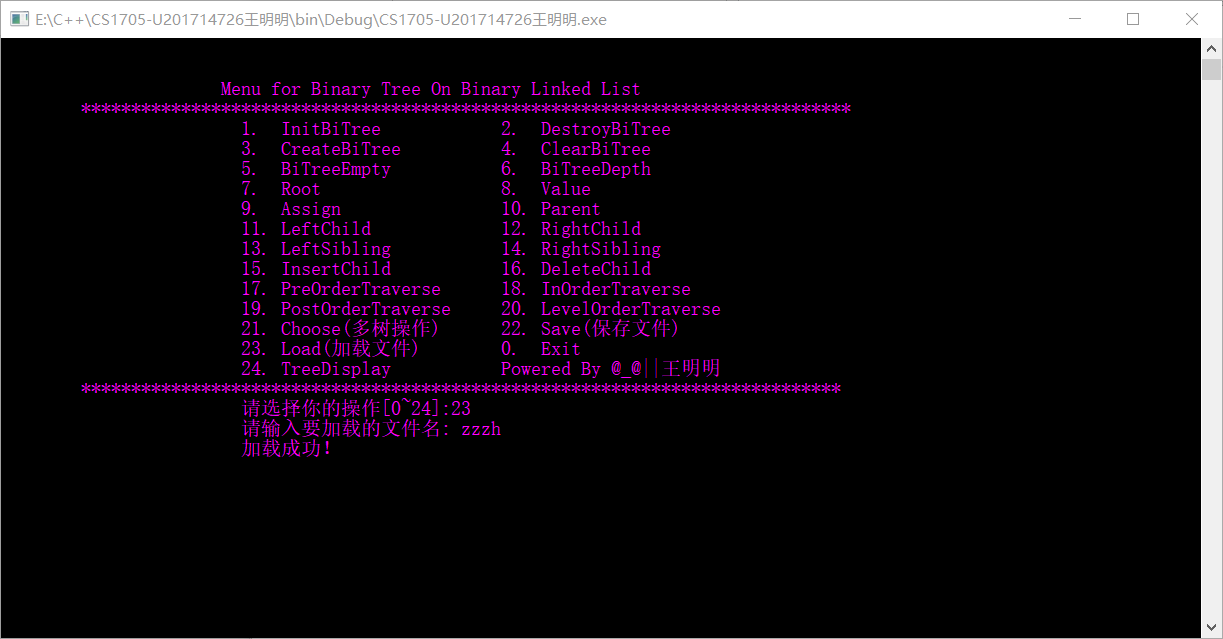
表3-1 测试用例T的节点值与key

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 节点值 | a | b | c | d | e | f | g |
| key | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

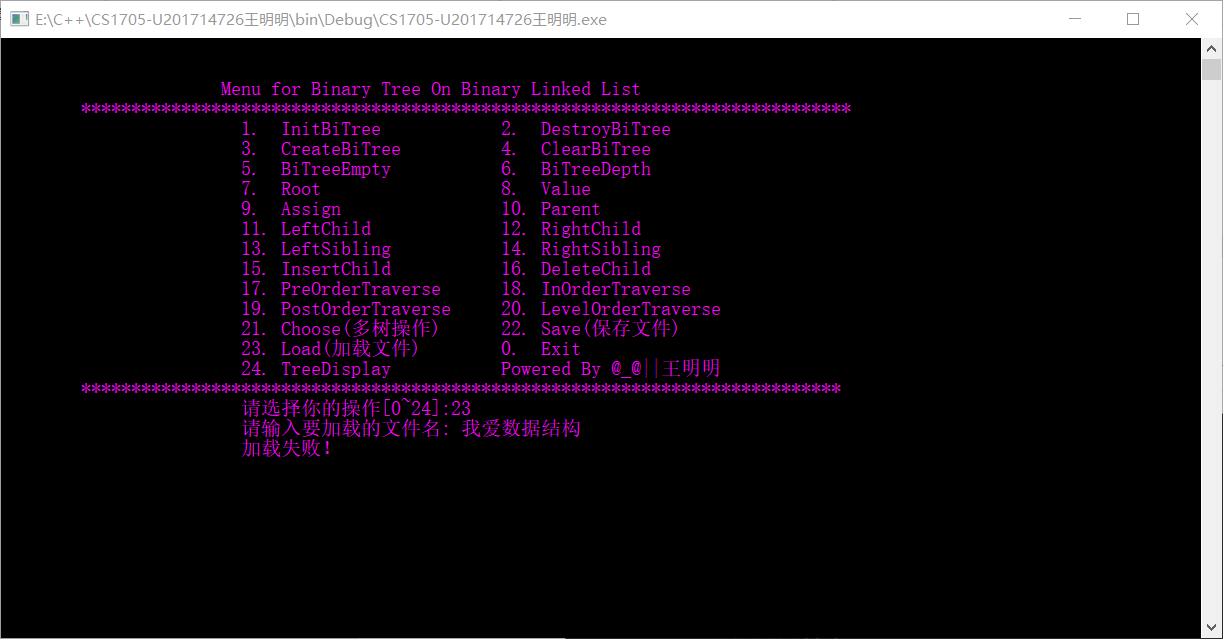
(1)Load函数测试

表3-2 Load测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 界面输入 | 理论结果 | 运行结果 |
| zzzh | 23 | 加载成功！ | 加载成功！ |
| 树不存在 | 23 | 加载失败！ | 加载失败！ |



(a)文件存在时Load测试截图



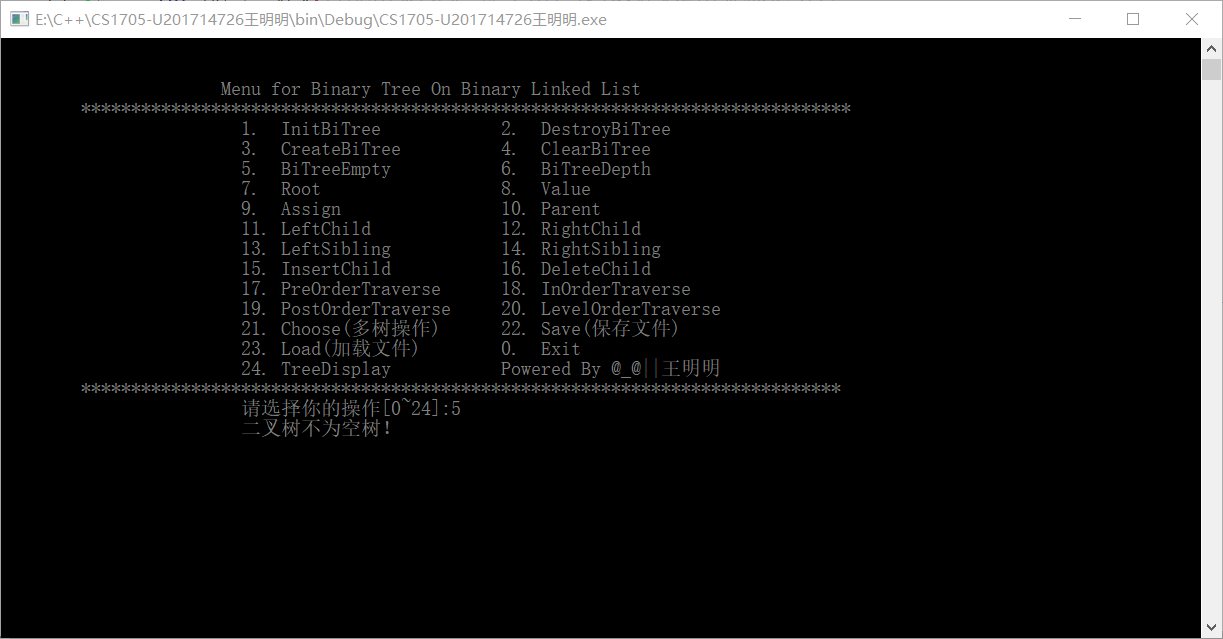
(b)文件不存在时Load测试截图

图3-7 Load测试运行截图

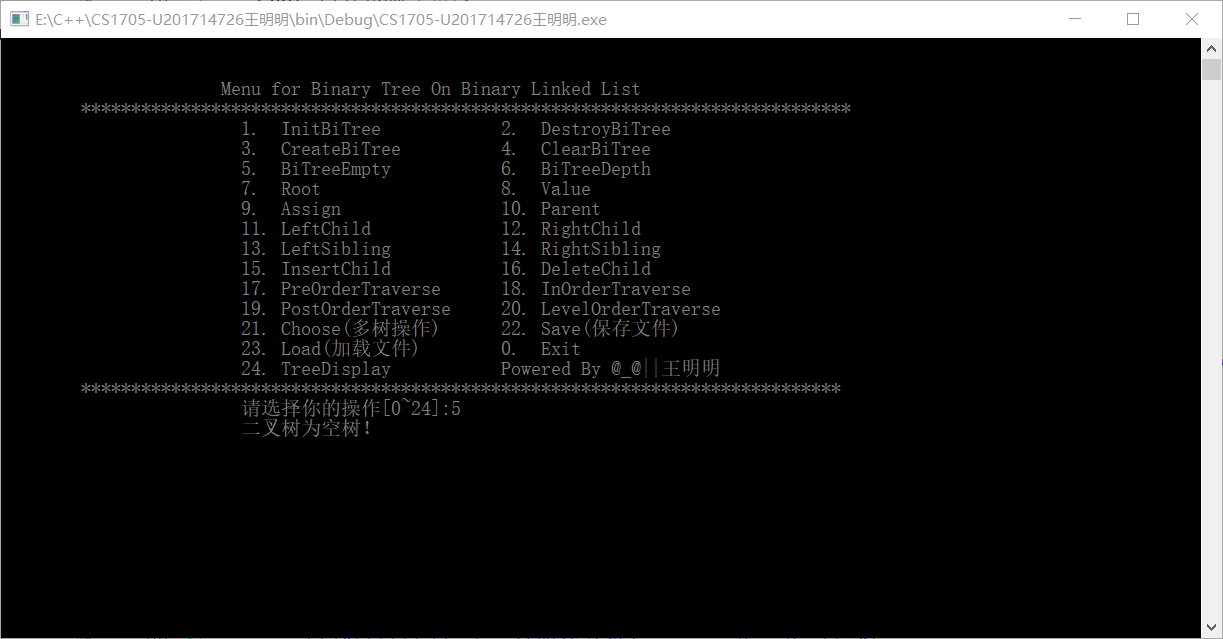
(2) BiTreeEmpty测试

表3-3 BiTreeEmpty测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 界面输入 | 理论结果 | 运行结果 |
| zzzh | 5 | 二叉树不为空树！ | 二叉树不为空树！ |
| 树不存在 | 5 | 二叉树为空树！ | 二叉树为空树！ |



(a)二叉树不为空时截图



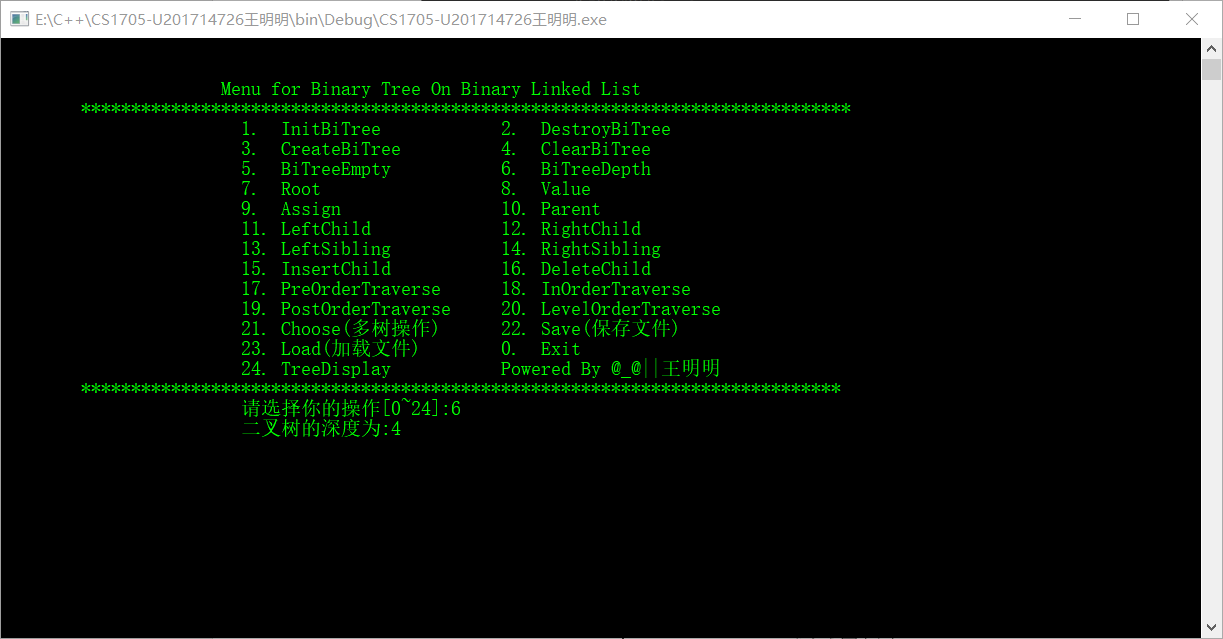
(b)二叉树为空时截图

图3-8 BiTreeEmpty测试截图

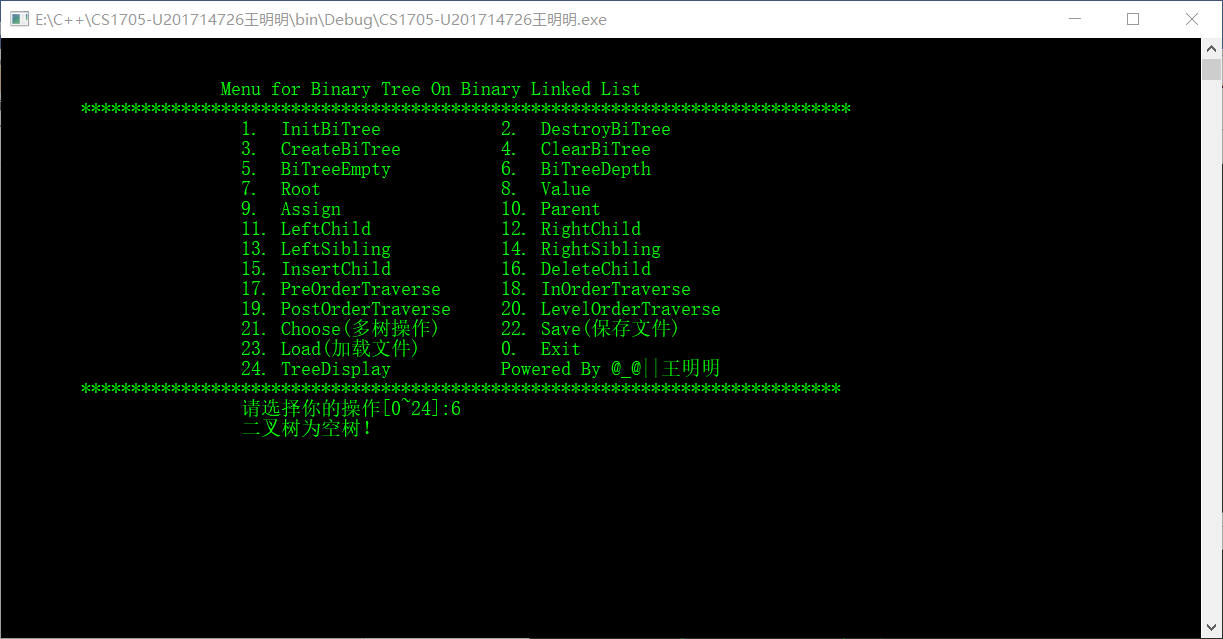
(3) BiTreeDepth测试

表3-4 BiTreeDepth测试用例表

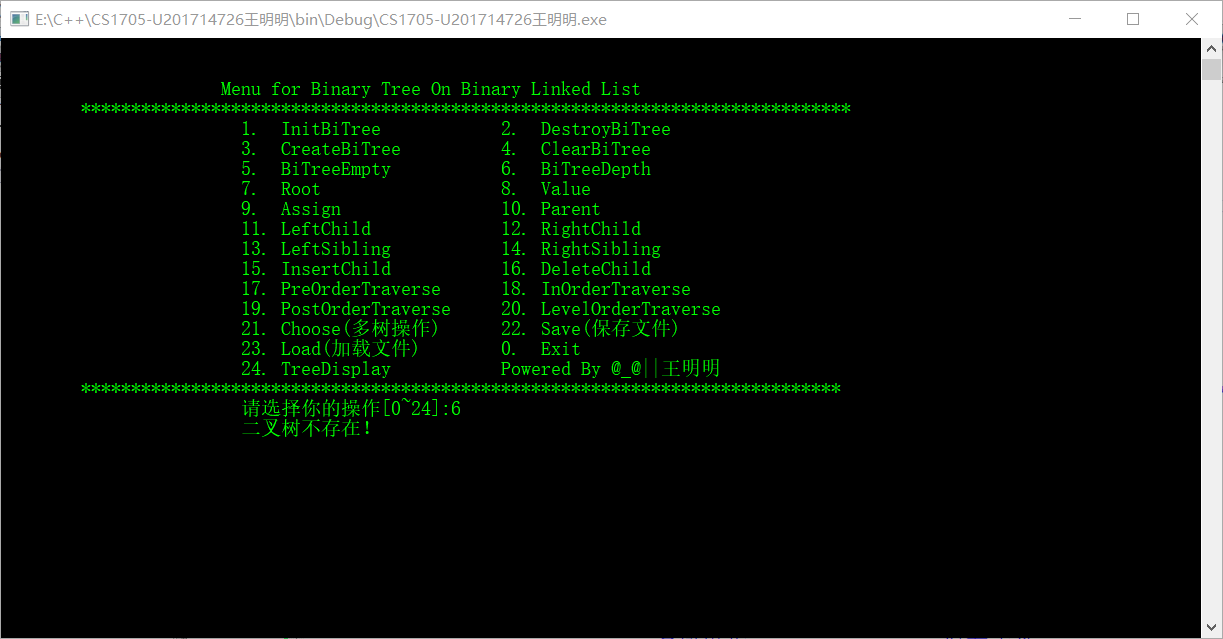
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 界面输入 | 理论结果 | 测试结果 |
| zzzh | 6 | 二叉树深度为：4 | 二叉树深度为4 |
| 空树 | 6 | 二叉树为空树！ | 二叉树为空树！ |
| 二叉树不存在 | 6 | 二叉树不存在！ | 二叉树不存在！ |



(a)二叉树存在且不为空时截图



(b)二叉树存在且为空时截图



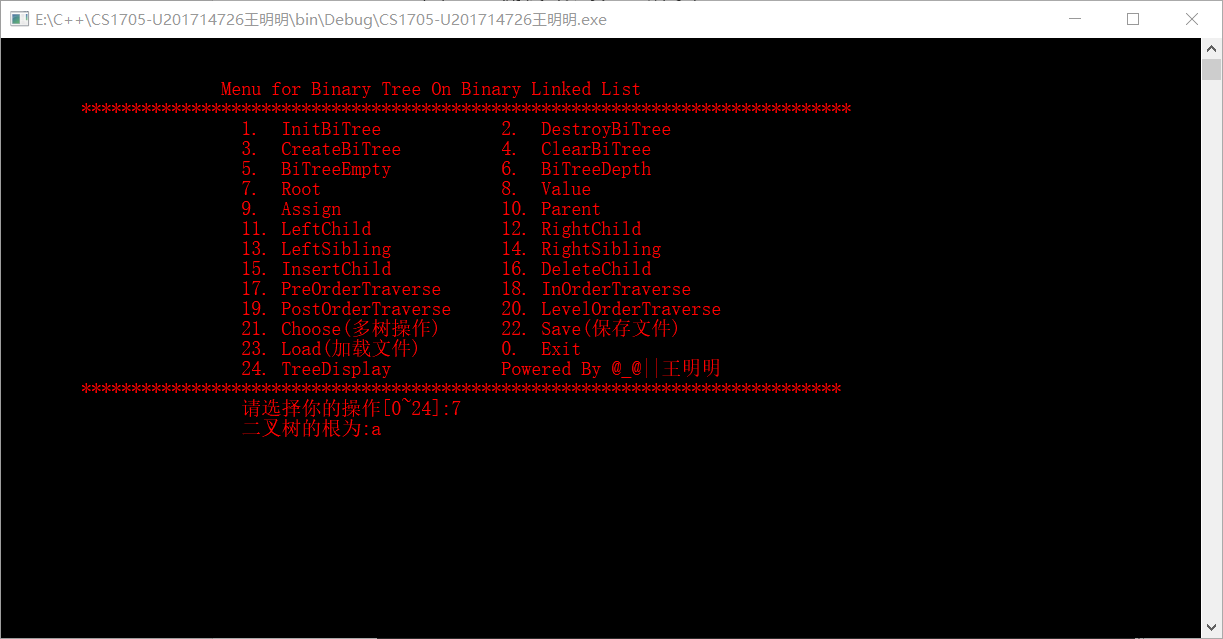
(c)二叉树不存在时截图

图3-9 BiTreeDepth测试截图

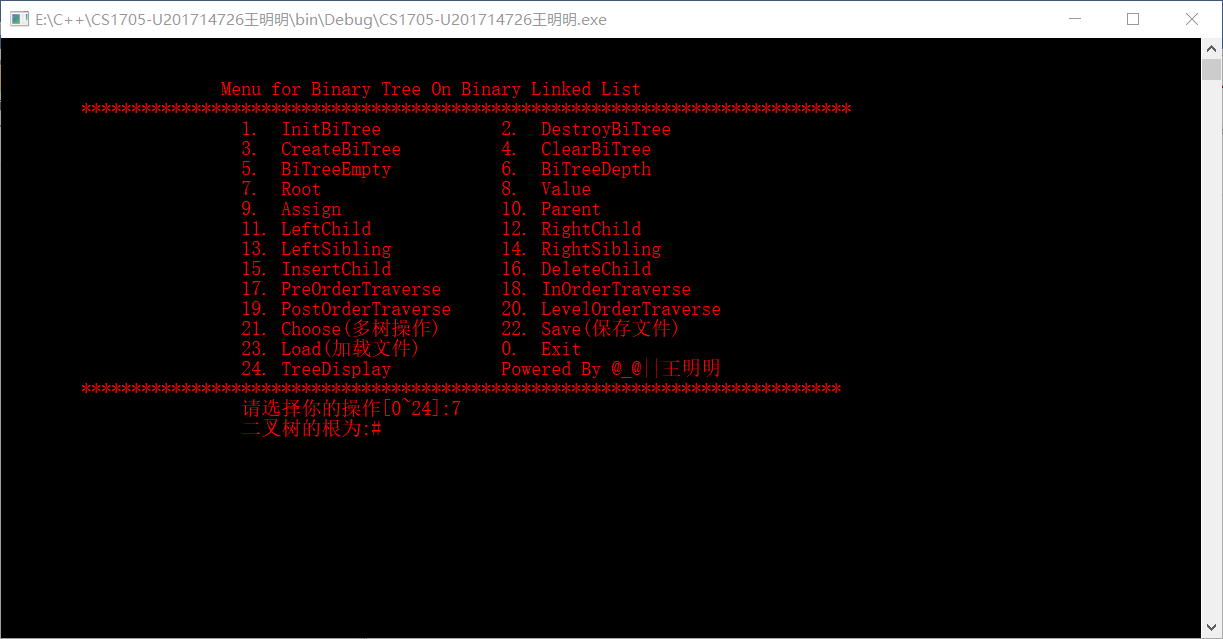
(4) Root测试

表3-5 Root测试用例表

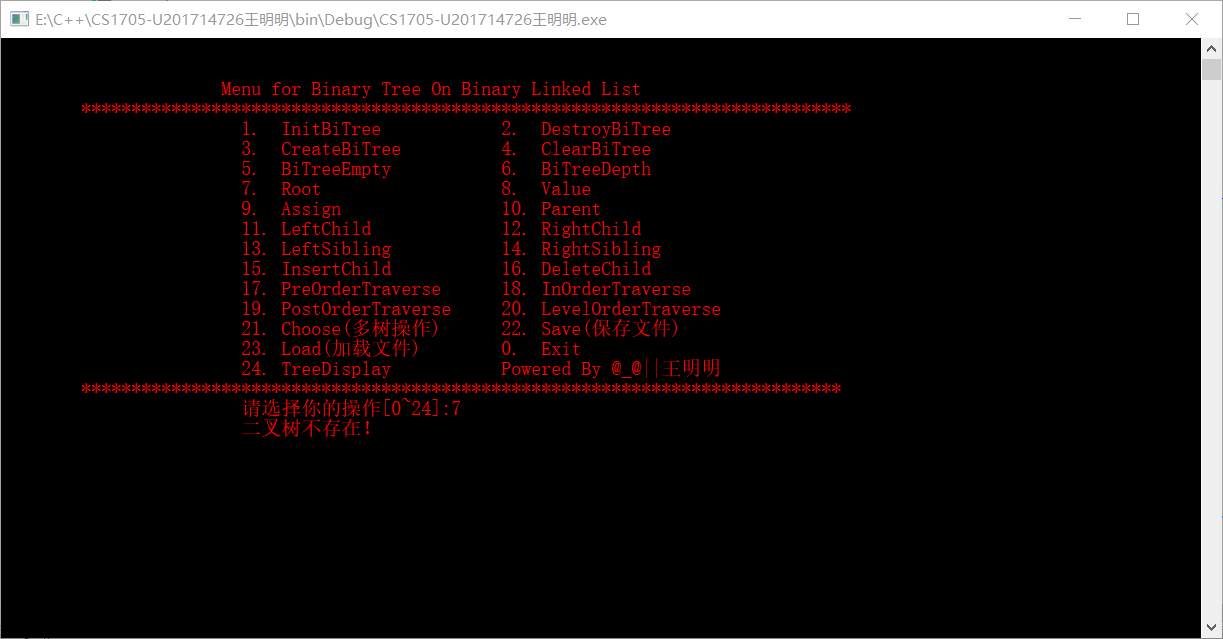
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 界面输入 | 理论结果 | 测试结果 |
| zzzh | 7 | 二叉树的根为：a | 二叉树的根为：a |
| 空树 | 7 | 二叉树的根为：# | 二叉树的根为：# |
| 二叉树不存在 | 7 | 二叉树不存在！ | 二叉树不存在！ |



(a)二叉树存在且不为空时截图



(b)二叉树存在且为空时截图



(c)二叉树不存在时截图

图3-10 Root测试截图

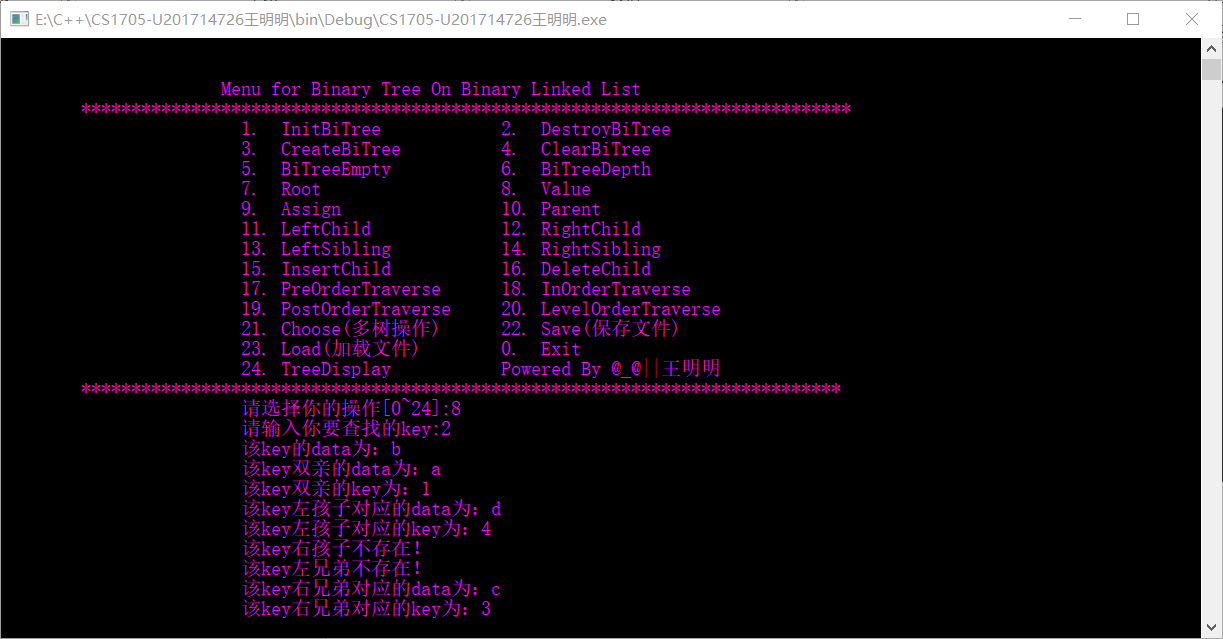
(5) Value测试

表3-6 Value测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 界面输入 | 理论结果 | 测试结果 |
| zzzh | 8  3 | 该key的data为：c  该key双亲的data为：a  该key双亲的key为：1  该key左孩子对应的data为：e  该key左孩子对应的key为：5  该key右孩子对应的data为：f  该key右孩子对应的key为：6  该key左兄弟对应的data为：b  该key左兄弟对应的key为：2  该key右兄弟不存在！ | 该key的data为：c  该key双亲的data为：a  该key双亲的key为：1  该key左孩子对应的data为：e  该key左孩子对应的key为：5  该key右孩子对应的data为：f  该key右孩子对应的key为：6  该key左兄弟对应的data为：b  该key左兄弟对应的key为：2  该key右兄弟不存在！ |
| zzzh | 8  2 | 该key的data为：b  该key双亲的data为：a  该key双亲的key为：1  该key左孩子对应的data为：d  该key左孩子对应的key为：4  该key右孩子不存在！  该key左兄弟不存在！  该key右兄弟对应的data为：c  该key右兄弟对应的key为：3 | 该key的data为：b  该key双亲的data为：a  该key双亲的key为：1  该key左孩子对应的data为：d  该key左孩子对应的key为：4  该key右孩子不存在！  该key左兄弟不存在！  该key右兄弟对应的data为：c  该key右兄弟对应的key为：3 |

![E:\Software\971278183\FileRecv\MobileFile\Image\HLM%Y$I745](X48@G9~RQ[F.png](data:image/png;base64,)

(a)对测试用例zzzh取key为3时截图



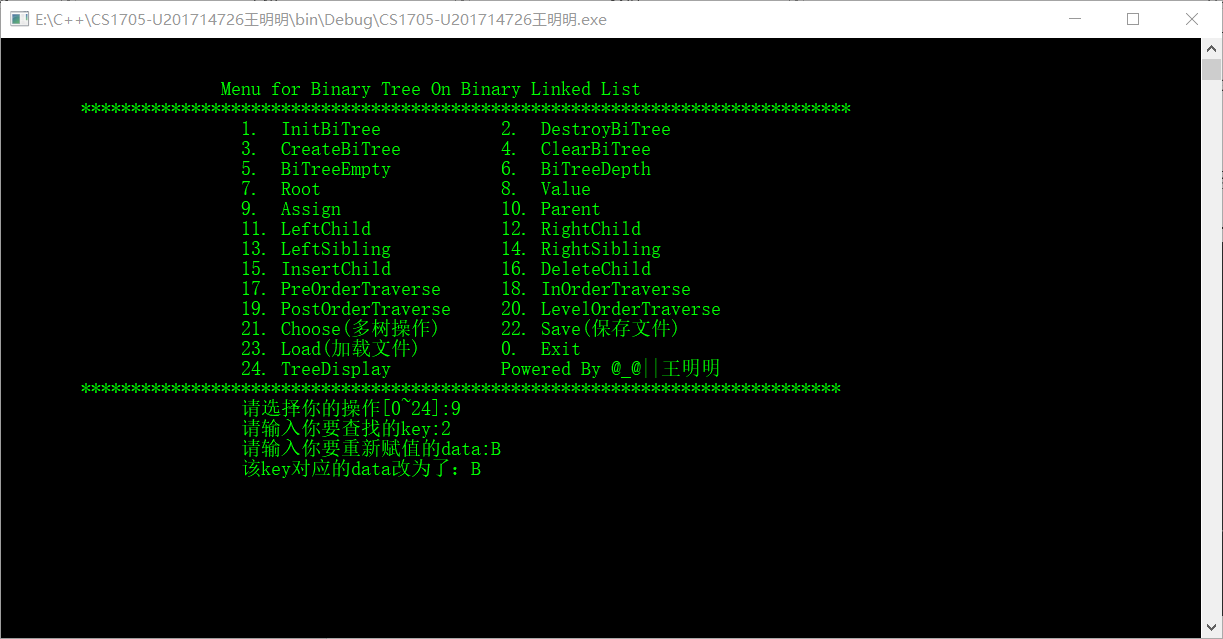
(b)对测试用例zzzh取key为2时截图

图3-11 Value测试截图

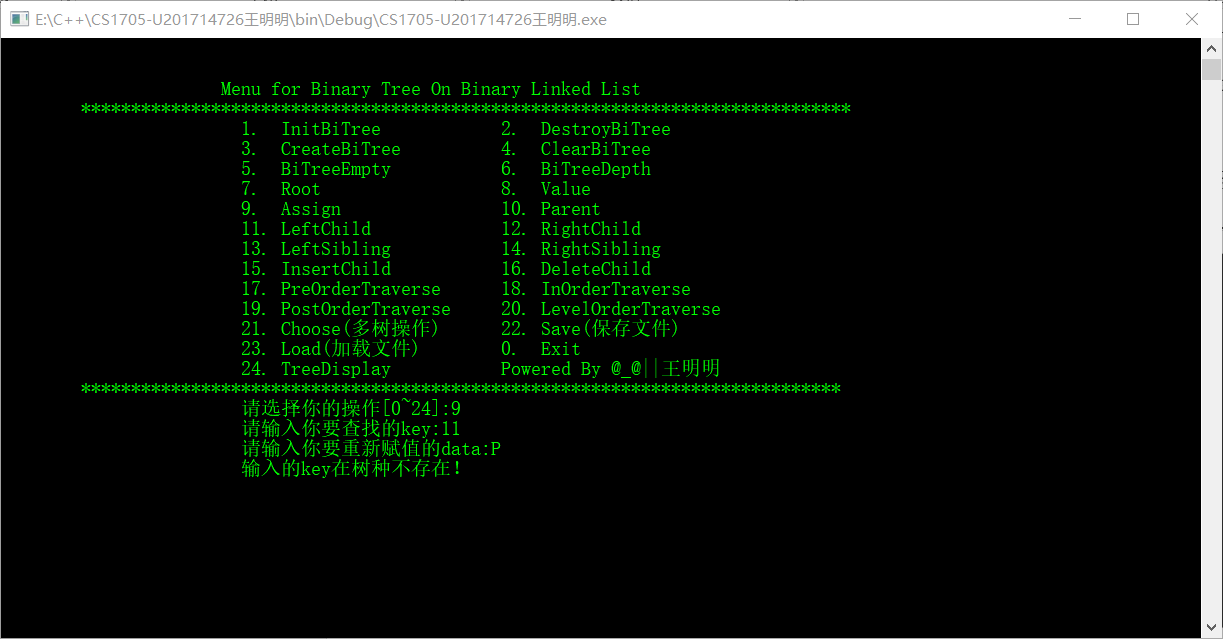
(6) Assign测试

表3-7 Assign测试用例表

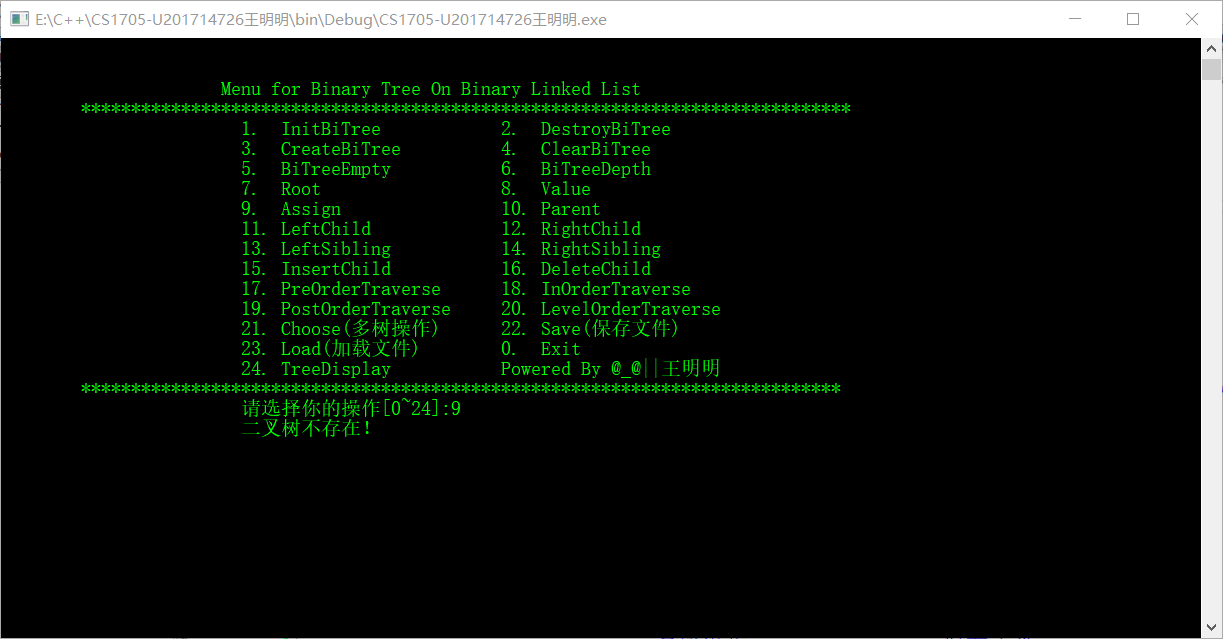
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选9  输入要查找的key:2  输入要重新赋值的data:B | 该key对应的data改为了：B | 该key对应的data改为了：B |
| 空树 | 主界面选9  输入要查找的key:11  输入要重新赋值的data:P | 输入的key在树中不存在！ | 输入的key在树中不存在！ |
| 二叉树不存在 | 主界面选9 | 二叉树不存在！ | 二叉树不存在！ |



(a)二叉树存在且为非空时截图



(b)二叉树存在且为空时截图



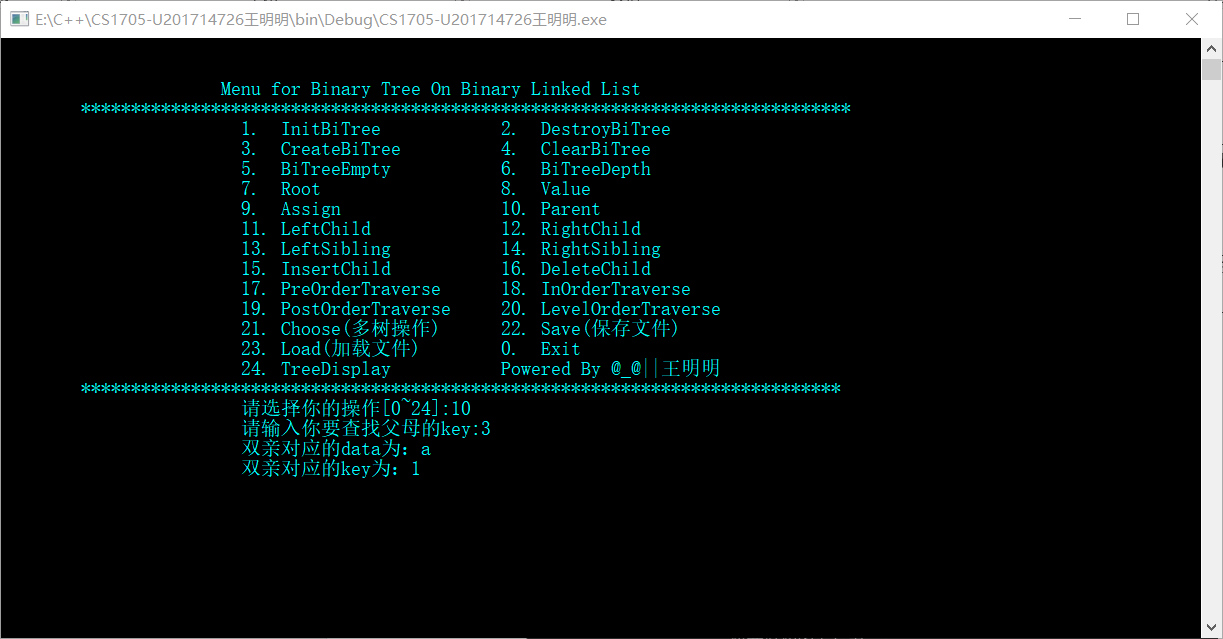
(c)二叉树不存在时截图

图3-12 Assign测试截图

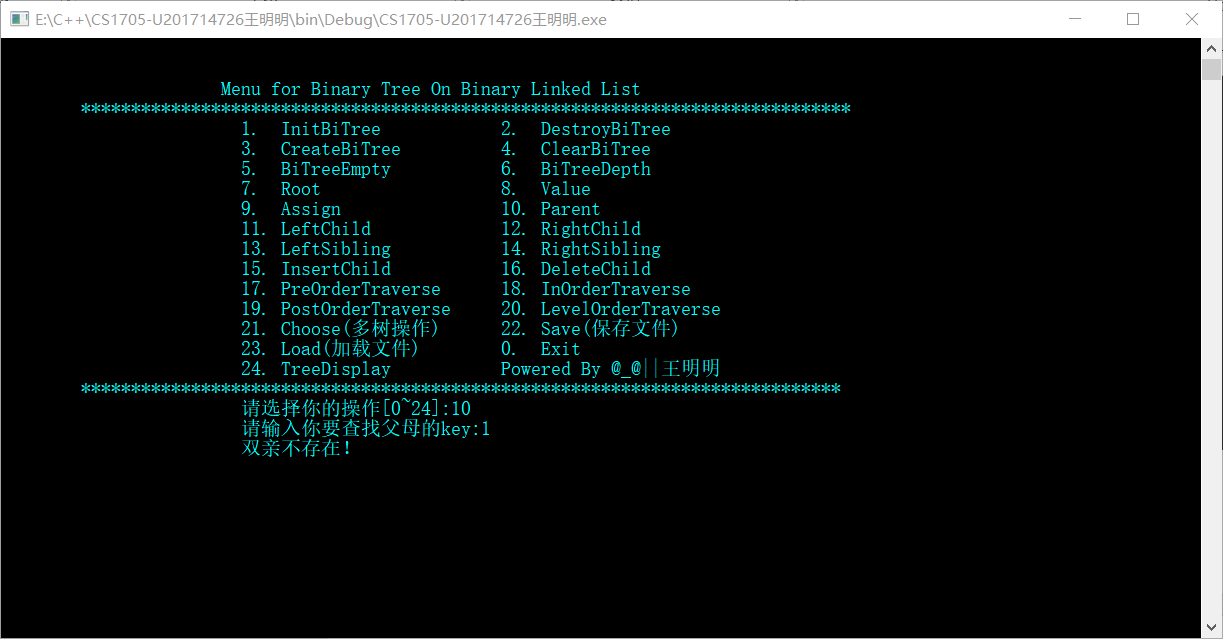
(7) Parent测试

表3-8 Parent测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选10  输入要查找的key:1 | 双亲不存在！ | 双亲不存在！ |
| zzzh | 主界面选10  输入要查找的key:3 | 双亲对应的data为：a 双亲对应的key为：1 | 双亲对应的data为：a 双亲对应的key为：1 |
| 二叉树不存在 | 主界面选10 | 二叉树不存在！ | 二叉树不存在！ |



(a)zzzh用例下key为3时的截图



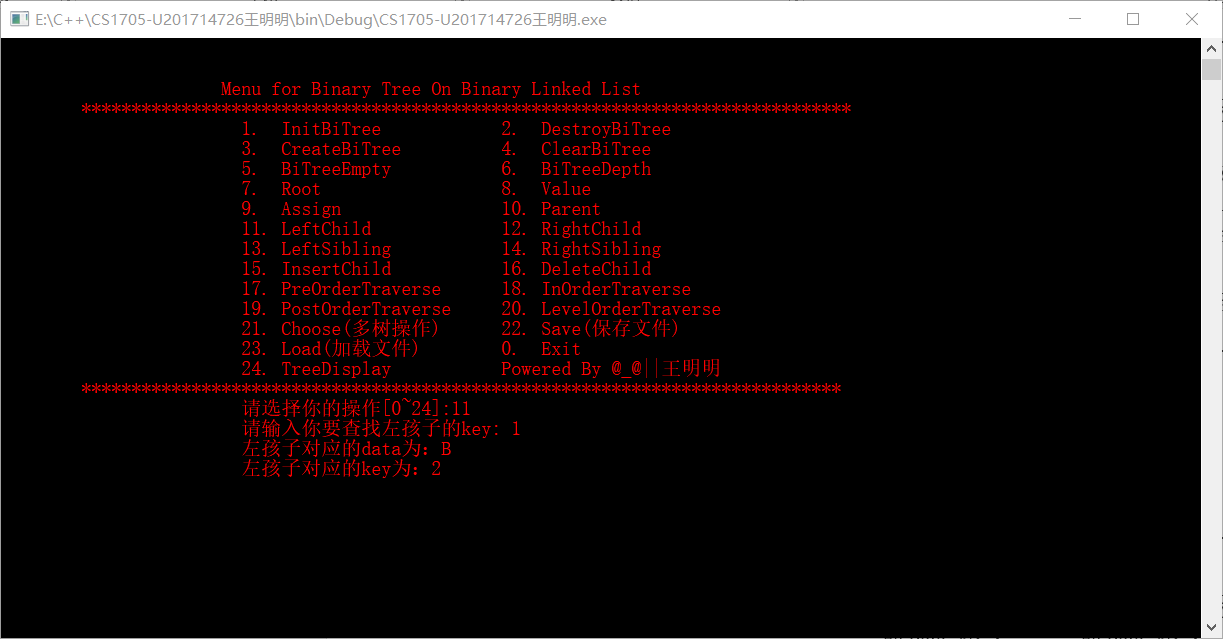
(b)zzzh用例下根节点情况

图3-13 Parent测试截图

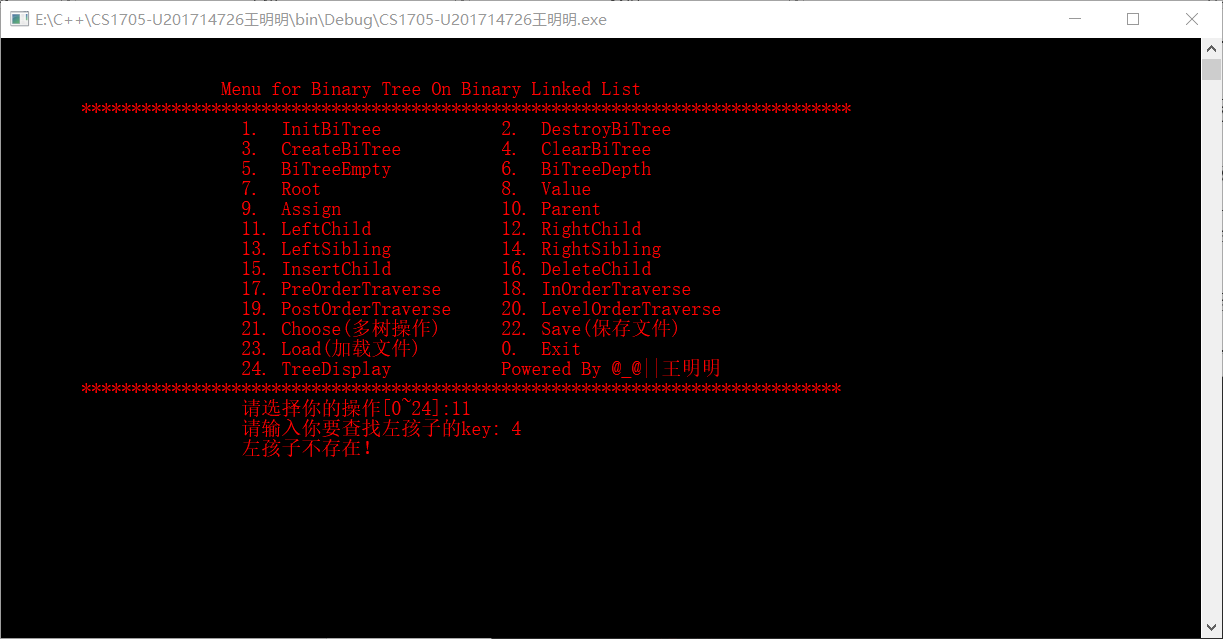
(8) LeftChild测试

表3-9 LeftChild测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选11  输入要查找的key:1 | 左孩子对应的data为：b 左孩子对应的key为：2 | 左孩子对应的data为：b 左孩子对应的key为：2 |
| zzzh | 主界面选11  输入要查找的key:4 | 左孩子不存在！ | 左孩子不存在！ |
| 二叉树树不存在 | 主界面选11 | 二叉树不存在！ | 二叉树不存在！ |



(a)zzzh用例下key为1时截图



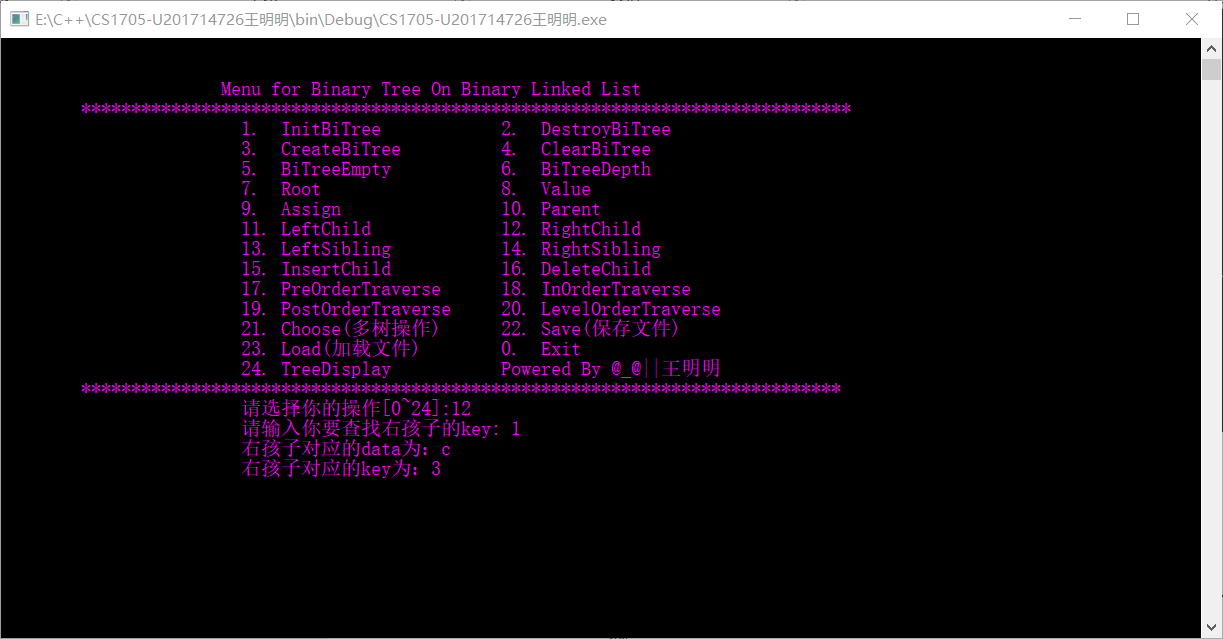
(b)zzzh用例下key为4时截图

图3-14 LeftChild测试截图

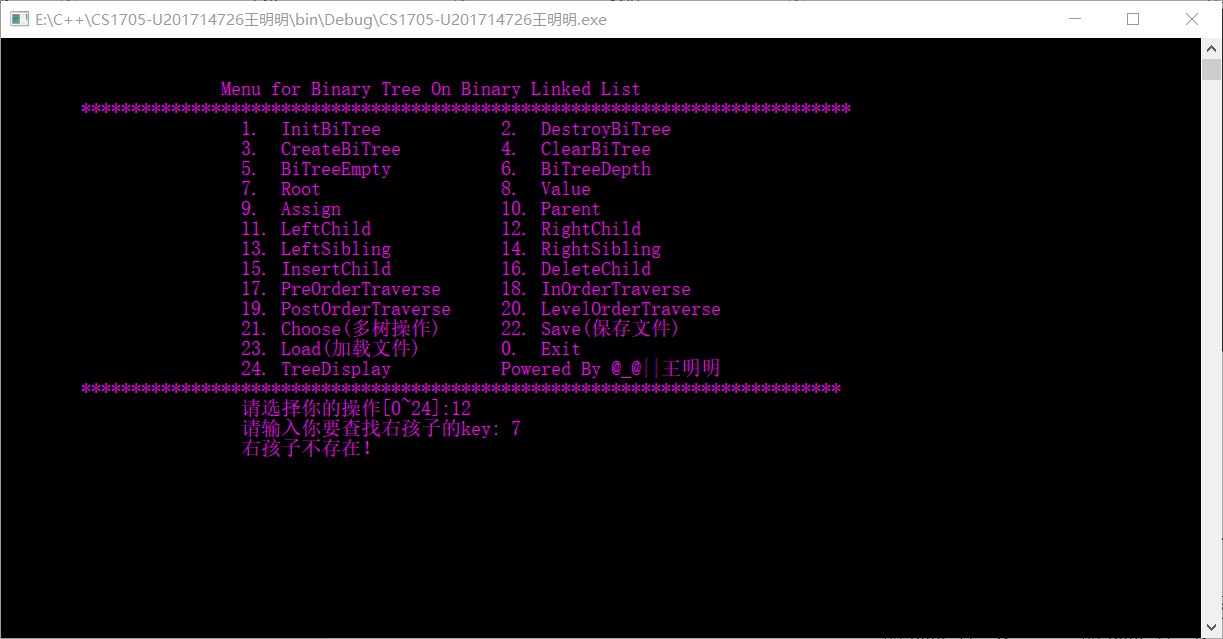
(9) RightChild测试

表3-10 RightChild测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选12  输入要查找的key:1 | 右孩子对应的data为：c 右孩子对应的key为：3 | 右孩子对应的data为：c 右孩子对应的key为：3 |
| zzzh | 主界面选12  输入要查找的key:7 | 右孩子不存在！ | 右孩子不存在！ |
| 二叉树不存在 | 主界面选12 | 二叉树不存在！ | 二叉树不存在！ |



(a)zzzh用例下key为1时截图



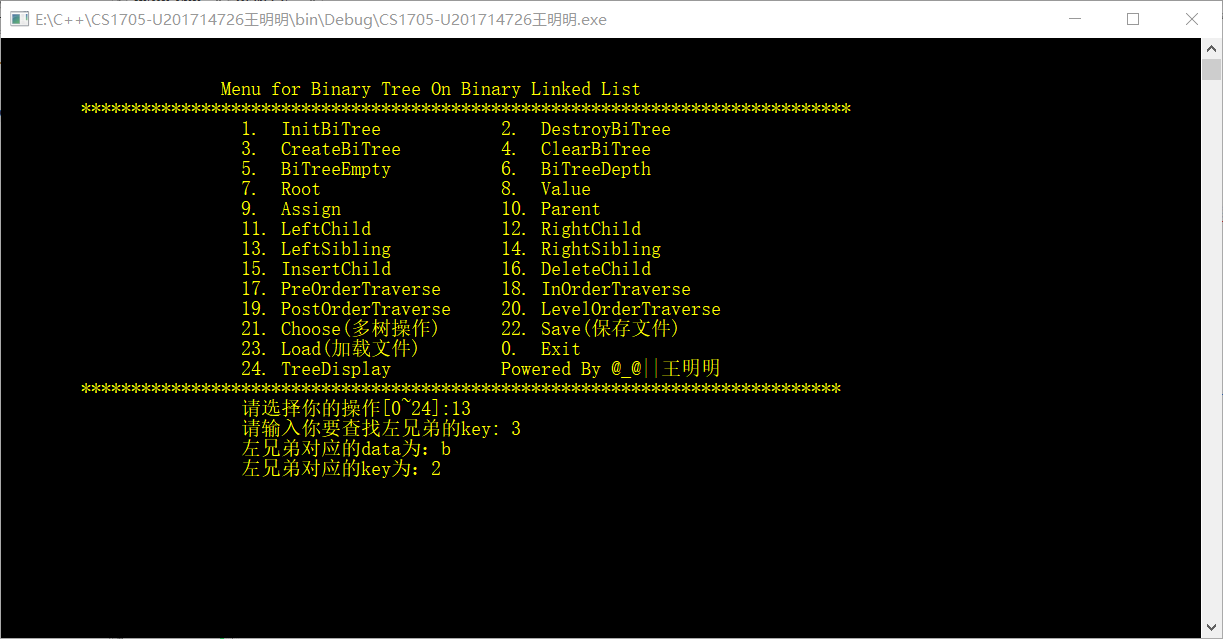
(b)zzzh用例下key为7时截图

图3-15 RightChild测试截图

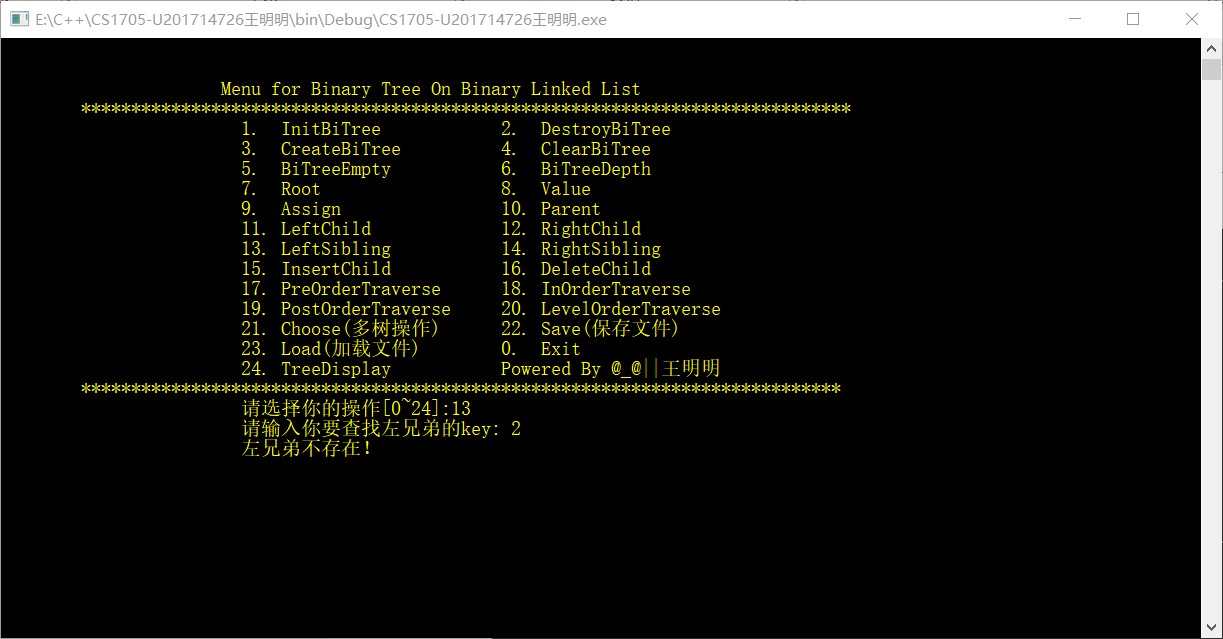
(10) LeftSibling测试

表3-11 LeftSibling测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| T1 | 主界面选13  输入要查找的key:3 | 左兄弟对应的data为：b 左兄弟对应的key为：2 | 左兄弟对应的data为：Bb 左兄弟对应的key为：2 |
| T1 | 主界面选13  输入要查找的key:2 | 左兄弟不存在！ | 左兄弟不存在！ |
| 二叉树不存在 | 主界面选11 | 二叉树不存在！ | 二叉树不存在！ |



(a)zzzh用例下key为3时截图



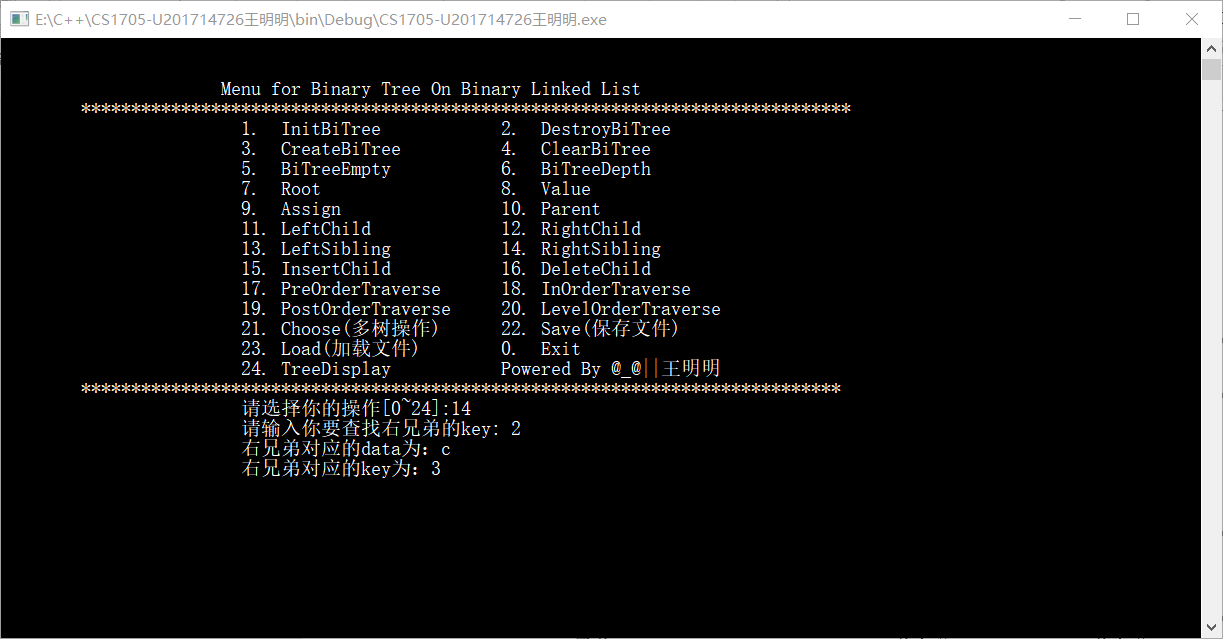
(b)zzzh用例下key为2时截图

图3-16 LeftSibling测试截图

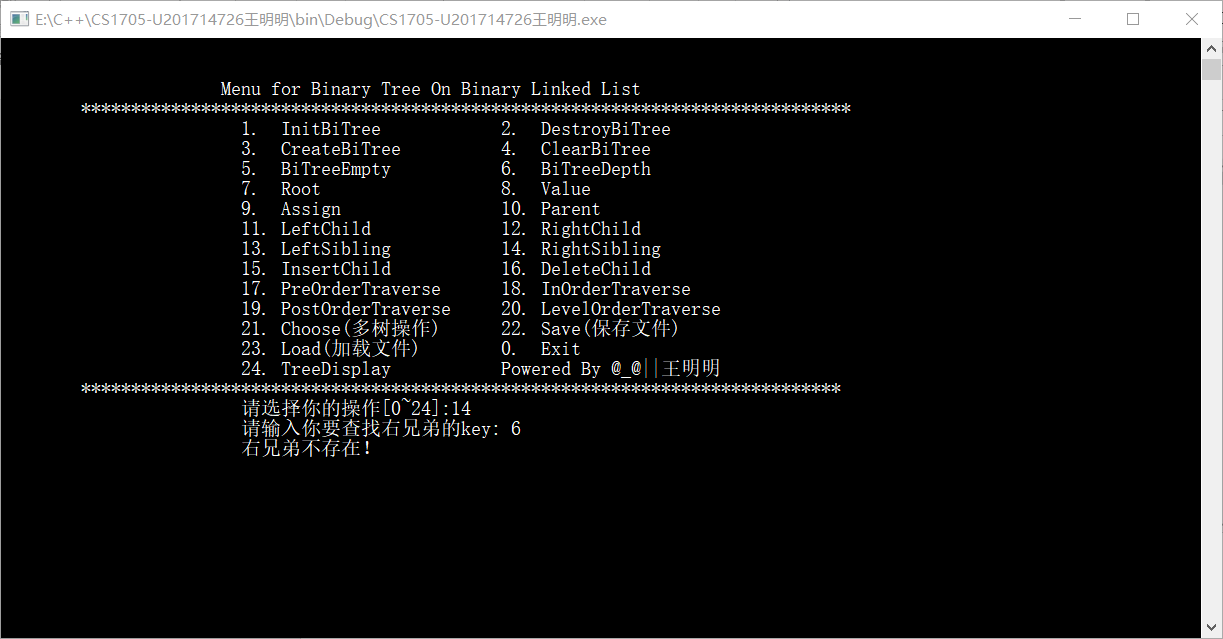
(11) RightSibling测试

表3-12 RightSibling测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选14  输入要查找的key:2 | 右兄弟对应的data为：c 右兄弟对应的key为：3 | 右兄弟对应的data为：c 右兄弟对应的key为：3 |
| zzzh | 主界面选14  输入要查找的key:6 | 右兄弟不存在！ | 右兄弟不存在！ |
| 二叉树不存在 | 主界面选14 | 二叉树不存在！ | 二叉树不存在！ |



(a)zzzh用例下key为2时截图



(b)zzzh用例下key为6时截图

图3-17 RightSibling测试截图

(12) InsertChild测试

表3-13 InsertChild测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| T1 | 主界面选15  输入要查找的key:6  方向LR：1  插入子树h## | 成功插入子树! | 成功插入子树! |
| 二叉树不存在 | 主界面选15 | 二叉树不存在！ | 二叉树不存在！ |

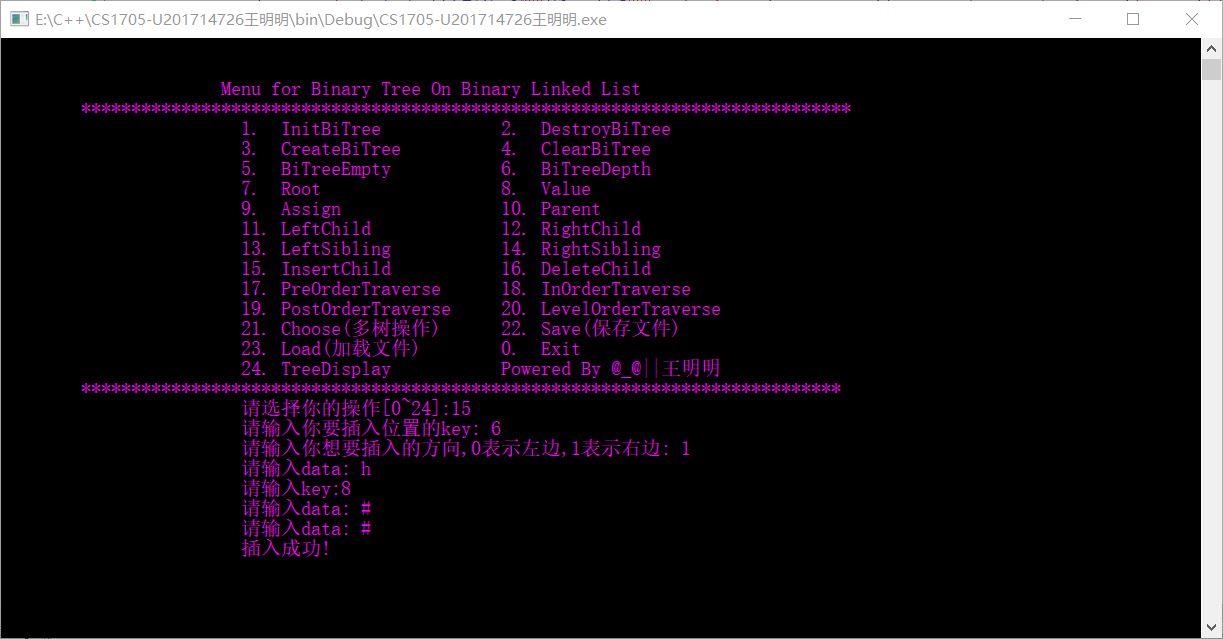


图3-18 InsertChild测试截图

(13) DeleteChild测试

表3-14 DeleteChild测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选16  输入要查找的key:6  方向LR：0 | 删除成功！ | 删除成功！ |
| 二叉树不存在 | 主界面选16 | 二叉树不存在！ | 二叉树不存在！ |

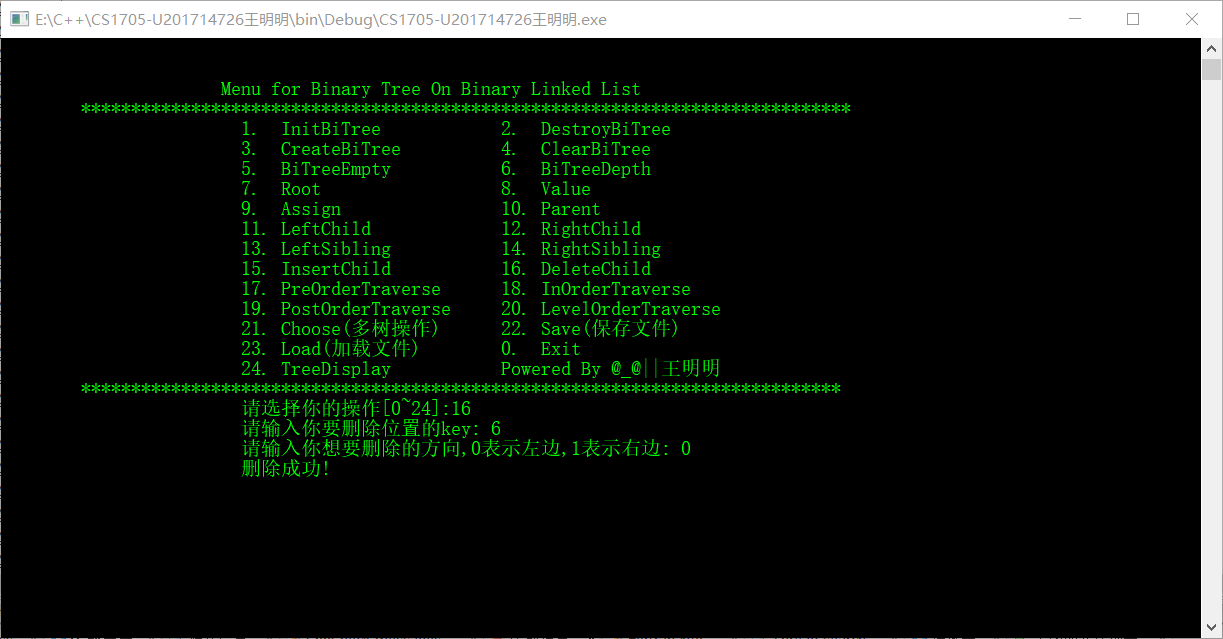


图3-19 DeleteChild测试截图

(14) PreOrderTraverse测试

表3-15 PreOrderTraverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| T1 | 主界面选17 | 该二叉树的前序遍历为：abdcefg | 该二叉树的前序遍历为：abdcefg |
| 二叉树不存在 | 主界面选17 | 二叉树不存在！ | 二叉树不存在！ |

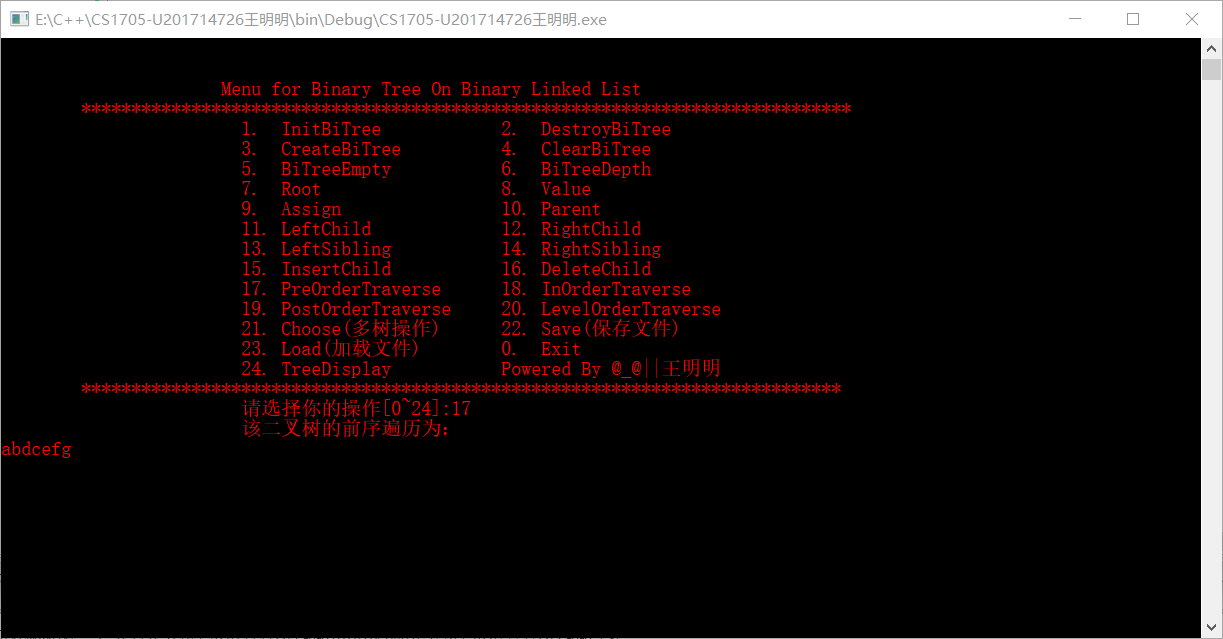


图3-20 PreOrderTraverse测试截图

(15) InOrderTraverse测试

表3-16 InOrderTraverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选18 | 该二叉树的中序遍历为：dbaecgf | 该二叉树的中序遍历为：dbaecgf |
| 二叉树不存在 | 主界面选18 | 二叉树不存在！ | 二叉树不存在！ |

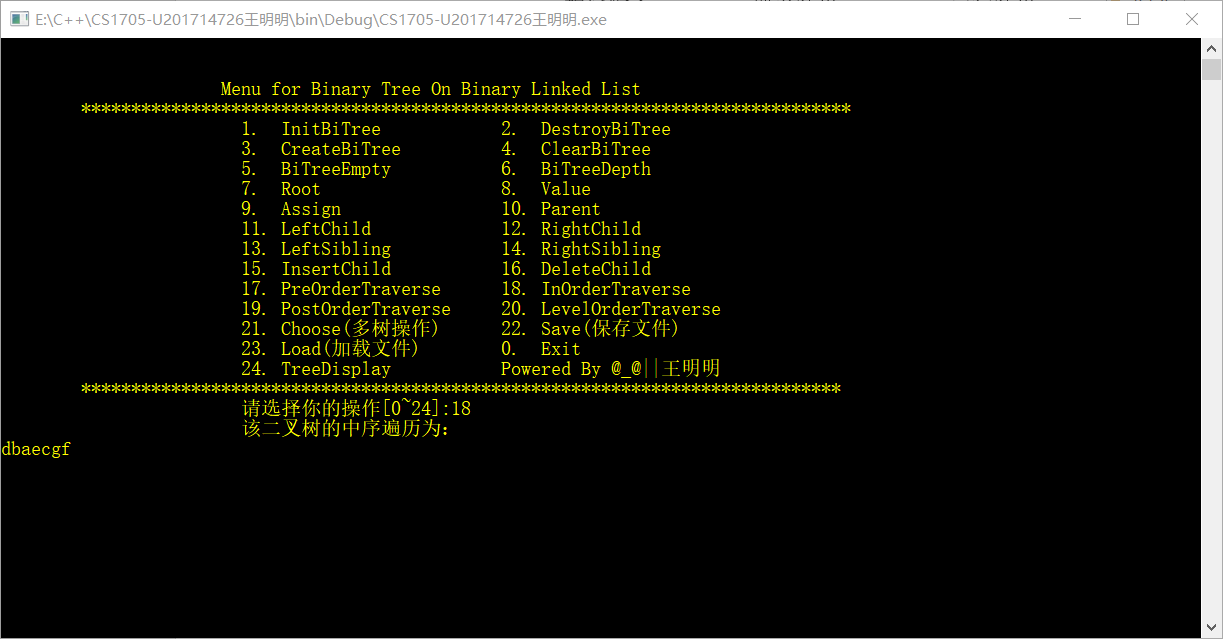


图3-21 InOrderTraverse测试截图

(16) PostOrderTraverse测试

表3-17 PostOrderTraverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选19 | 该二叉树的后序遍历为：dbegfca | 该二叉树的后序遍历为：dbegfca |
| 二叉树不存在 | 主界面选19 | 二叉树不存在！ | 二叉树不存在！ |

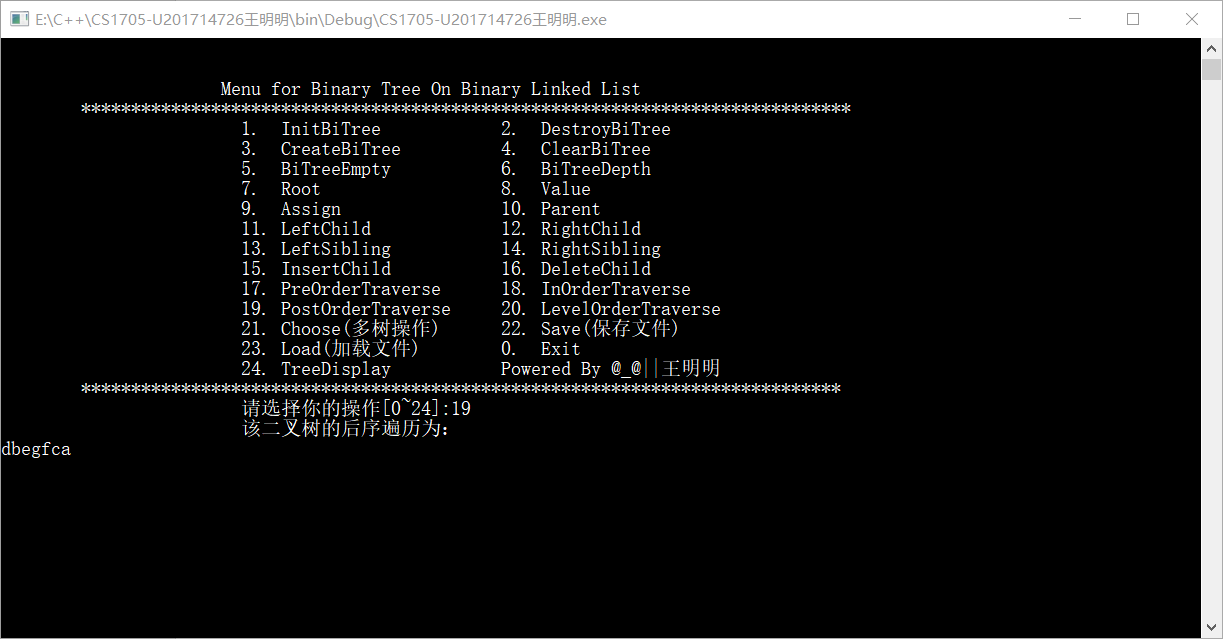


图3-22 PostOrderTraverse测试截图

(17) LevelOrderTraverse测试

表3-18 LevelOrderTraverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选20 | 该二叉树的层序遍历为：abcdefg | 该二叉树的层序遍历为：abcdefg |
| 二叉树树不存在 | 主界面选20 | 二叉树不存在！ | 二叉树不存在！ |

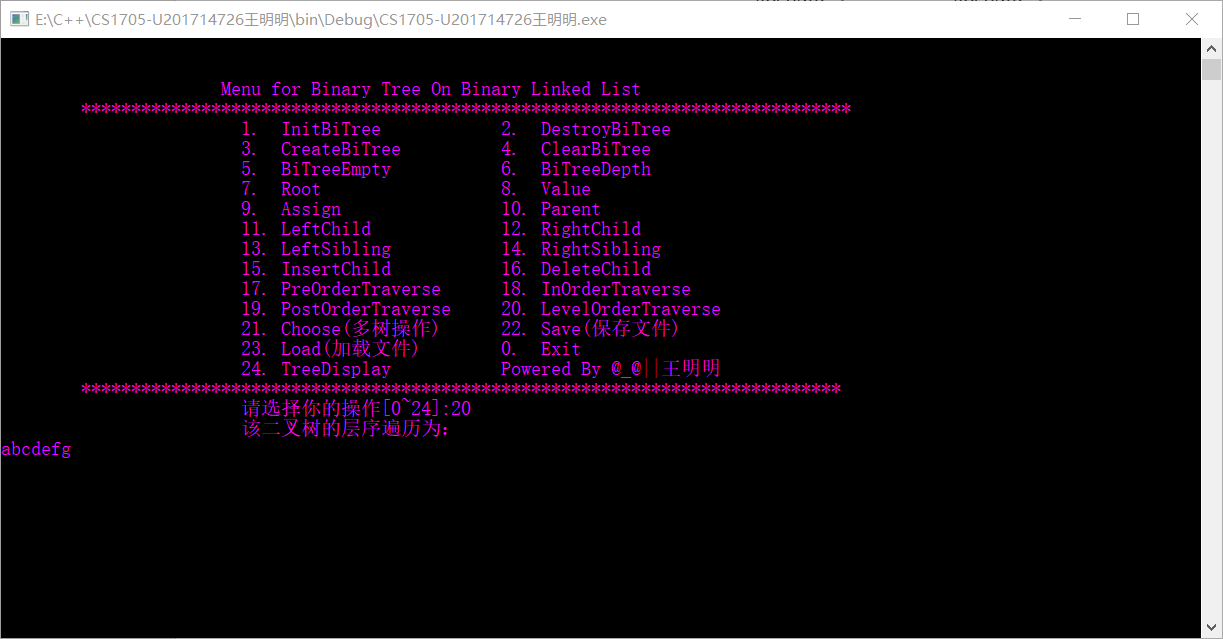


图3-23 LevelOrderTraverse测试截图

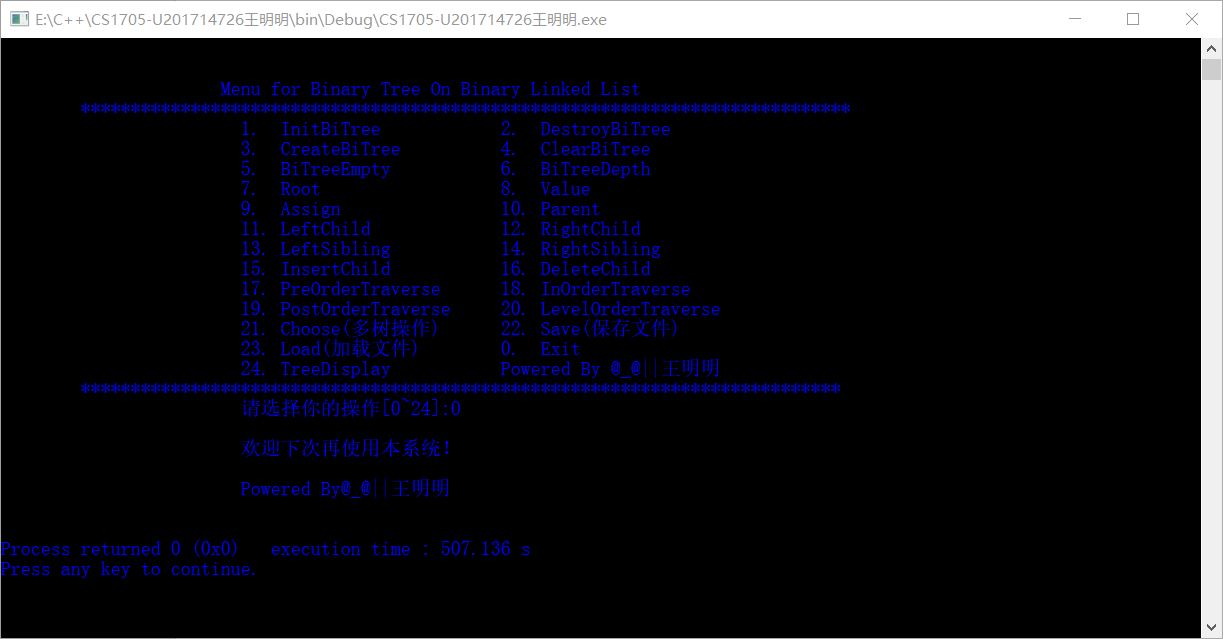


图3-24 Exit运行截图

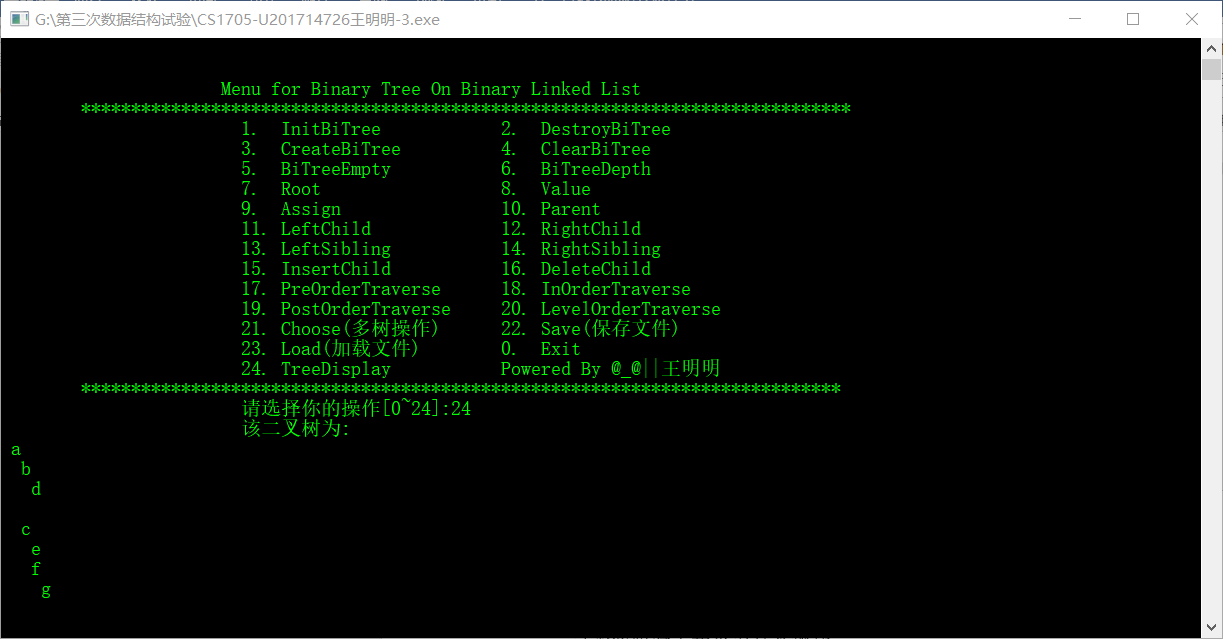


图3-25 TreeDisplay（第一部分为根节点和左子树、第二部分为右子树）

## 3.4 实验小结

这次实验对于我来说是个挑战。

在开始实验之前我就已经生病了，上呼吸道感染伴随低烧症状，六处口腔溃疡的口子，说不了话，很难进食。另外这次实验有20多个函数要完成，确实很有难度。在第一个星期里，我基本没看实验，因为真的没有思路，而且生病真的很难受，这也导致了我第二个星期没能按时给助教检查。第二个星期之后，病情好转，开始加班加点写实验三。

在本次实验过程中，我遇到很多困难，但在和同学的讨论和思考中得以解决。比如，我初始化树之后判断是否为空树，显示结果确是非空！后来仔细研究好久才发现：InitBiTree和BiTreeEmpty两个函数出现了错误，在初始化树的时候并没有考虑data域，结果是，只有根节点的树也会被认为是空树，刚初始化的树会被认为是非空树。于是，我在初始化的时候对data域进行了复制，将其设为空节点，用#表示，于是这个问题得以解决。还有一个就是Value函数，老师上课说要像查户口一般输入姓名出来所有的信息，比如身份证号、父母、配偶、孩子等等所有信息。我最初的时候没有考虑那么多，只是单纯的取出节点的值，后来改了很久，最终造就了Value函数——一个集合了10、11、12、13、14这几个函数功能的强大函数。

另外，在这次实验中留有遗憾，没有画流程图，因为我时间不够了。如果没有生病的话，我想我有充足的时间完成流程图的绘制。这是我本次实验的遗憾，我想我会在下次实验弥补。

最后，感谢同学老师助教的帮助，在我最弱的一两个星期里给予我支持和理解，谢谢。

# 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>

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

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define MAX\_NUM 10

typedef int status;

typedef int ElemType; //数据元素类型定义

/\*-------page 22 on textbook -------\*/

#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;

FILE \*fp;

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

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, status(\*compare)(ElemType,ElemType));

status compare(ElemType a, ElemType b);

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

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

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

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

status ListTrabverse(SqList L);

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

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

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

int main(void){

char filename[40];

int op=1;

int i;

int i\_num=1;

SqList L[MAX\_NUM];

for(i=0;i<MAX\_NUM;i++)

{

L[i].elem = NULL;

L[i].listsize = 0;

L[i].length = 0;

}

//上面的for循环是用来生成存储空间为空的线性表

ElemType e, cur\_e , pre\_e, next\_e;

while(op){

/\*\*

\*利用最简单的printf来制作简易的菜单，可供选择；

\*简洁美观的菜单有助于平复测试时的心情!!!

\*/

system("cls"); //用于清屏

printf("\n\n");

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

printf(" 可在%d个顺序表进行多表操作，初始化请先操作功能15,默认在第一个表上操作\n", MAX\_NUM);

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

printf("\*\*\t\t\t1. IntiaList 7. LocateElem\t\t\t\*\*\n");

printf("\*\*\t\t\t2. DestroyList 8. PriorElem\t\t\t\*\*\n");

printf("\*\*\t\t\t3. ClearList 9. NextElem \t\t\t\*\*\n");

printf("\*\*\t\t\t4. ListEmpty 10. ListInsert\t\t\t\*\*\n");

printf("\*\*\t\t\t5. ListLength 11. ListDelete\t\t\t\*\*\n");

printf("\*\*\t\t\t6. GetElem 12. ListTrabverse\t\t\t\*\*\n");

printf("\*\*\t\t\t13.SaveList 14. LoadList\t\t\t\*\*\n");

printf("\*\*\t\t\t0. Exit 制作时间：2018.11.13\t\t\t\*\*\n");

printf("\*\*\t\t\t15.ChooseList(请先进行此选项以选择在哪个表上进行操作)\n");

printf("\*\*\t\t\t本实验已有文件sss，可通过函数14进行加载\n");

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

printf("请选择你的操作[0--15]:\n");

scanf("%d",&op);//选择op的值,用于switch

switch(op){

case 1:

//第一种情况是初始化线性表

if(IntiaList(L[i\_num])==OK)

{

printf("请输入要保存的线性表名称\n");

scanf("%s", filename);

printf("线性表创建成功\n");

}

else printf("线性表创建失败！\n");

getchar();getchar();

break;

case 2:

//第二种情况是用来销毁线性表

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

if(DestroyList(L[i\_num])==OK)

{

printf("销毁线性表成功!\n");

}

else printf("销毁线性表失败！\n");

getchar();getchar();

break;

case 3:

//用于重置线性表

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

if(ClearList(L[i\_num])==OK)

{

printf("线性表重置成功！\n");

}

else printf("线性表重置失败！\n");

getchar();getchar();

break;

case 4:

//判断是否为空

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

if(ListEmpty(L[i\_num])==TRUE)

{

printf("文件为空！\n");

}

else printf("线性表不是空表！\n");

getchar();getchar();

break;

case 5:

//得到线性表长度

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("线性表表长为%d\n",ListLength(L[i\_num]));

getchar();getchar();

break;

case 6:

//得到某个元素

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入要取结点的位置：\n");

scanf("%d",&i);

if(GetElem(L[i\_num],i,&e)==OK)

printf("第%d个结点的元素是：%d\n",i,e);

else printf("输入位置错误！\n");

getchar();getchar();

break;

case 7:

//确定元素位置，容易出错

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入数据元素值：\n");

scanf("%d",&e);

if(i=LocateElem(L[i\_num],e,compare))

printf("%d元素位于第%d个位置！\n",e,i);

else printf("该元素不存在!\n");

getchar();getchar();

break;

case 8:

//求出前驱结点

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入数据元素：\n");

scanf("%d",&cur\_e);

PriorElem(L[i\_num],cur\_e,&pre\_e);

if(PriorElem(L[i\_num],cur\_e,&pre\_e)==OK)

printf("其前驱元素为：%d\n",pre\_e);

else if(PriorElem(L[i\_num],cur\_e,&pre\_e)==OVERFLOW)

printf("顺序表中没有该元素！\n");

else printf("其不存在前驱元素！\n");

getchar();getchar();

break;

case 9:

//求出后置节点

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入数据元素：\n");

scanf("%d",&cur\_e);

if(NextElem(L[i\_num],cur\_e,&next\_e)==OK)

printf("其后继元素为：%d\n",next\_e);

else if(NextElem(L[i\_num],cur\_e,&pre\_e)==FALSE)

printf("其不存在后继元素！\n");

else

{printf("顺序表中没有该元素！\n");}

getchar();getchar();

break;

case 10:

//插入元素

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入您要插入的数据元素：\n");

scanf("%d",&e);

printf("请输入您要插入的数据元素的位置：\n");

scanf("%d",&i);

if(ListInsert(L[i\_num],i,e)==OK)

printf("插入数据元素成功！\n");

else

printf("插入数据元素失败！\n");

getchar();getchar();

break;

case 11:

//删除元素

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入您要删除的数据元素的位置：\n");

scanf("%d",&i);

if(ListDelete(&L[i\_num],i,&e)==OK)

printf("删除数据元素成功！\n");

else

printf("删除数据元素失败！\n");

getchar();getchar();

break;

case 12:

//遍历线性表中的元素

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

if(!ListTrabverse(L[i\_num])) printf("线性表是空表！\n");

getchar();getchar();

break;

case 13:

//保存文件

if(L[i\_num].elem == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

if(SaveList(L[i\_num], filename)==OK)

printf("文件保存成功\n文件名为%s\n",filename);

break;

case 14:

//加载文件，需要输入需要加载的名称

printf("请输入要加载的文件名:\n ");

scanf("%s", filename);

if(LoadList(&L[i\_num], filename)==OK)

{

printf("文件加载成功\n");

}

break;

case 15:

//选择在哪个表进行操作

printf("请输入要在第几个表操作:\n ");

printf("\*只支持在%d个顺序表上操作\*\n",MAX\_NUM);

scanf("%d",&i\_num);

printf("正在对第%d个表进行操作\n",i\_num);

if((i\_num<1)||(i\_num>10))

{

printf("请选择正确范围！\n");

i\_num=1;

}

getchar(); getchar();

break;

break;

case 0:

//退出菜单，退出整个程序

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本辣鸡系统!\n");

}//end of main()

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：IntiaList

\*函数功能：构造一个空的线性表

\*注释：初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status IntiaList(SqList &L){

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

if(!L.elem) exit(OVERFLOW);//如果空间不足，创建失败

L.length=0;

L.listsize=LIST\_INIT\_SIZE;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：DestoryList

\*函数功能：销毁线性表

\*注释：初始条件是线性表L已存在；操作结果是销毁线性表L

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status DestroyList(SqList &L)

{

if(L.elem)

free(L.elem);

L.elem = NULL;

L.length = 0;

L.listsize = 0;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ClearList

\*函数功能：重置顺序表

\*注释：初始条件是线性表L已存在；操作结果是将L重置为空表。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ClearList(SqList &L)

{

L.length=0;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListEmpty

\*函数功能：判断线性表是否为空

\*注释：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。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ListEmpty(SqList L)

{

if(L.length==0)

{

return TRUE;

}

return FALSE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListLength

\*函数功能：求线性表的表长

\*注释：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。

\*返回值类型：int类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int ListLength(SqList L)

{

return L.length;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：GetElem

\*函数功能：得到某一个元素的值

\*注释：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

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

if(i<1||i>L.length)

{

return ERROR;

}

\*e = L.elem[i-1];//将得到的元素赋值给e

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：LocateElem

\*函数功能：查找元素

\*注释：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。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status LocateElem(SqList L,ElemType e,status(\*compare)(ElemType,ElemType))

{

int i;

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

{

if(compare(L.elem[i],e))

return ++i;

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：compare

\*函数功能：比较大小，服务于LocateList函数

\*注释：输入两个ElemType类型的值

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status compare(ElemType a, ElemType b)

{

if(a == b)

return TRUE;

else return FALSE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：PriorElem

\*函数功能：求元素的前驱

\*注释：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。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

int i;

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

{

if(L.elem[i]==cur\_e && i==0)

{

return ERROR;

}

else if(L.elem[i]== cur\_e)

{

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

return OK;

}

}

return OVERFLOW;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：NextElem

\*函数功能：求后继节点

\*输入输出：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。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

int i;

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

{

if(L.elem[i]==cur\_e)

{

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

return OK;

}

}

if(i==L.length-1 && (L.elem[i]!=cur\_e)) return OVERFLOW;

else return FALSE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListInsert

\*函数功能：插入元素

\*注释：初始条件是线性表L已存在且非空，1≤i≤ListLength(L)+1；

\* 操作结果是在L的第i个位置之前插入新的数据元素e

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

int \*p,\*q,\*newbase;

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

{

printf("插入位置不正确!\n");

return ERROR;

}

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

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

if(!newbase) exit(OVERFLOW);

L.elem = newbase;

L.listsize += LISTINCREMENT;

}

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

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

\*q=e;

++L.length;

return OK;

}

//这是课本上面的关于插入算法的实现

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListDelete

\*函数功能：删除元素

\*注释：初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；

\* 操作结果：删除L的第i个数据元素，用e返回其值。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

if(i<1||i>L->length)

return ERROR;//删除的位数不正确

int j;

\*e=L->elem[i-1];

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

L->elem[j] = L->elem[j + 1];

L->length--;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListTrabverse

\*函数功能：遍历顺序表

\*注释：输出顺序表的值

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ListTrabverse(SqList L){

int i;

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

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

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

return L.length;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：SaveList

\*函数功能：保存线性表

\*注释：将线性表保存，参考附录B，其中关于写入元素个数和长度的问题理解不够清楚

\*返回值类型：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status SaveList(SqList L, char\* filename)

{

int i = 0;

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

{

printf("文件保存失败\n");

return ERROR;

}

fprintf(fp, "%d ", L.length);//保存的时候，也将L的长度保存到了文件

fprintf(fp, "%d ", L.listsize);//将每个元素的大小也保存到了文件里

while (i < L.length)

fprintf(fp, "%d ", L.elem[i++]);//利用循环，将元素依次存进去

fclose(fp);//关闭文件

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：LoadList

\*函数功能：加载文件

\*注释：加载文件，以便功能的测试，文件名要正确

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status LoadList(SqList \*L, char \*filename)

{

int i = 0;

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

{

printf("文件加载失败\n");

return ERROR;

}

fscanf(fp, "%d ", &L->length);

fscanf(fp, "%d ", &L->listsize);

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

if (!L->elem) exit(OVERFLOW);

while (i < L->length)

fscanf(fp, "%d ", &L->elem[i++]);//利用循环，依次读出文件中的内容

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 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\_NUM 10

typedef int status;

typedef int ElemType; //数据元素类型定义

/\*-------page 22 on textbook -------\*/

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct LNode{

ElemType data;

struct LNode \*next;

}LNode, \*LinkList;

FILE \*fp;

status InitList(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);

int LocateElem(LinkList L, ElemType e, status(\*compare)(ElemType a, ElemType b));

status compare(ElemType a, ElemType b);

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

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

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

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

status ListTrabverse(LinkList L);

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

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

int main(){

char filename[40];

int op=1;

int i,i\_num=1;

LinkList L[MAX\_NUM];

for (i = 0; i<MAX\_NUM; i++)

{

L[i]=NULL;

}

ElemType e, cur\_e, pre\_e, next\_e;

while(op){

system("cls");

printf("\n\n");

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

printf(" 可在%d个顺序表进行多表操作，初始化请先操作功能15,默认在第一个表上操作\n", MAX\_NUM);

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

printf("\*\*\t\t\t1. InitList 7. LocateElem\t\t\t\*\*\n");

printf("\*\*\t\t\t2. DestroyList 8. PriorElem\t\t\t\*\*\n");

printf("\*\*\t\t\t3. ClearList 9. NextElem \t\t\t\*\*\n");

printf("\*\*\t\t\t4. ListEmpty 10. ListInsert\t\t\t\*\*\n");

printf("\*\*\t\t\t5. ListLength 11. ListDelete\t\t\t\*\*\n");

printf("\*\*\t\t\t6. GetElem 12. ListTrabverse\t\t\t\*\*\n");

printf("\*\*\t\t\t13.SaveList 14. LoadList\t\t\t\*\*\n");

printf("\*\*\t\t\t0. Exit 制作时间：2018.11.13\t\t\t\*\*\n");

printf("\*\*\t\t\t15.ChooseList(请先进行此选项以选择在哪个表上进行操作)\n");

printf("\*\*\t\t\t本实验已有文件sss，可通过函数14进行加载\n");

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

printf("请选择你的操作[0--15]:\n");

scanf("%d",&op);

switch(op)

{

case 1:

//第一种情况是初始化线性表

if(InitList(&L[i\_num])==OK)

{

printf("请输入要保存的线性表名称\n");

scanf("%s", filename);

printf("线性表创建成功\n");

}

else printf("线性表创建失败！\n");

getchar();getchar();

break;

case 2:

//第二种情况是用来销毁线性表

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

if(DestroyList(&L[i\_num])==OK)

{

printf("销毁线性表成功!\n");

}

else printf("销毁线性表失败！\n");

getchar();getchar();

break;

case 3:

//用于重置线性表

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

if(ClearList(&L[i\_num])==OK)

{

printf("线性表重置成功！\n");

}

else printf("线性表重置失败！\n");

getchar();getchar();

break;

case 4:

//判断是否为空

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

if(ListEmpty(L[i\_num])==TRUE)

{

printf("文件为空！\n");

}

else printf("线性表不是空表！\n");

getchar();getchar();

break;

case 5:

//得到线性表长度

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("线性表表长为%d\n",ListLength(L[i\_num]));

getchar();getchar();

break;

case 6:

//得到某个元素

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入要取结点的位置：\n");

scanf("%d",&i);

if(GetElem(L[i\_num],i,&e)==OK)

printf("第%d个结点的元素是：%d\n",i,e);

else printf("输入位置错误！\n");

getchar();getchar();

break;

case 7:

//printf("\n----LocateElem功能待实现！\n");

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入数据元素值：\n");

scanf("%d",&e);

if(i=LocateElem(L[i\_num],e,compare))

printf("%d元素位于第%d个位置！\n",e,i);

else printf("该元素不存在!\n");

getchar();getchar();

break;

case 8:

//求出前驱结点

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入数据元素：\n");

scanf("%d",&cur\_e);

PriorElem(L[i\_num],cur\_e,&pre\_e);

if(PriorElem(L[i\_num],cur\_e,&pre\_e)==OK)

printf("其前驱元素为：%d\n",pre\_e);

else if(PriorElem(L[i\_num],cur\_e,&pre\_e)==OVERFLOW)

printf("顺序表中没有该元素！\n");

else printf("其不存在前驱元素！\n");

getchar();getchar();

break;

case 9:

//求出后置节点

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入数据元素：\n");

scanf("%d",&cur\_e);

if(NextElem(L[i\_num],cur\_e,&next\_e)==OK)

printf("其后继元素为：%d\n",next\_e);

else if(NextElem(L[i\_num],cur\_e,&pre\_e)==ERROR)

printf("顺序表中没有该元素！\n");

else

{printf("其不存在后继元素！\n");}

getchar();getchar();

break;

case 10:

//插入元素

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入您要插入的数据元素：\n");

scanf("%d",&e);

printf("请输入您要插入的数据元素的位置：\n");

scanf("%d",&i);

if(ListInsert(&L[i\_num],i,e)==OK)

printf("插入数据元素成功！\n");

else

printf("插入数据元素失败！\n");

getchar();getchar();

break;

case 11:

//删除元素

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

printf("请输入您要删除的数据元素的位置：\n");

scanf("%d",&i);

if(ListDelete(&L[i\_num],i,&e)==OK)

printf("删除数据元素成功！\n");

else

printf("删除数据元素失败！\n");

getchar();getchar();

break;

case 12:

//遍历线性表中的元素

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

if(!ListTrabverse(L[i\_num])) printf("线性表是空表！\n");

getchar();getchar();

break;

case 13:

//保存文件

if(L[i\_num] == NULL)

{

printf("线性表不存在!\n");

getchar();getchar();

break;

}

if(SaveList(L[i\_num], filename)==OK)

printf("文件保存成功!文件名为%s\n",filename);

getchar();getchar();

break;

case 14:

//加载文件，需要输入需要加载的名称

InitList(&L[i\_num]);

printf("请输入要加载的文件名:\n ");

scanf("%s", filename);

if(LoadList(&L[i\_num], filename)==OK)

{

printf("文件加载成功\n");

}

getchar();getchar();

break;

case 15:

//选择在哪个表进行操作

printf("请输入要在第几个表操作:\n ");

printf("\*只支持在%d个顺序表上操作\*\n",MAX\_NUM);

scanf("%d",&i\_num);

printf("正在对第%d个表进行操作\n",i\_num);

if((i\_num<1)||(i\_num>10))

{

printf("请选择正确范围！\n");

i\_num=1;

}

getchar(); getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("\t\t欢迎下次再使用本系统！\n");

}//end of main()

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：IntiaList

\*函数功能：构造一个空的线性表

\*注释：初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status InitList(LinkList \*L)

{

\*L = (LinkList)malloc(sizeof(LNode));//动态分配

if(\*L == NULL)

{

exit(OVERFLOW);//如果没有足够的空间，创建失败

}

(\*L)->data = 0;

(\*L)->next = NULL;//创建带有表头节点的链表，表头节点的数据域值为0

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：DestoryList

\*函数功能：销毁线性表

\*注释：初始条件是线性表L已存在；操作结果是销毁线性表L

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status DestroyList(LinkList \*L)

{

LinkList p, q;//指针p,q

p = \*L;//将指针p指向表头节点

while(p)

{

q = p->next;//如果p不指向空，将q指向p的下一个节点

free(p);//然后释放p

p = q;//再将q所指向的节点赋给p

}

\*L = NULL;//最后指针L指向空

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ClearList

\*函数功能：重置顺序表

\*注释：初始条件是线性表L已存在；操作结果是将L重置为空表。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ClearList(LinkList \*L)

{

LinkList p, q;//创建两个指针p,q

p = (\*L)->next;//将p指向第一个节点

while(p)

{

q = p->next;//当p不指向空时，q指向p的下一个节点

free(p);//释放p

p = q;//将q指向的节点赋给p

}

(\*L)->next = NULL;//最后，将表头节点的指针域指向空

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListEmpty

\*函数功能：判断线性表是否为空

\*注释：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。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ListEmpty(LinkList L)

{

if(L->next)

{

return FALSE;

}

return TRUE;//如果表头节点的指针域指向空，那么返回TRUE

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListLength

\*函数功能：求线性表的表长

\*注释：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。

\*返回值类型：int类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int ListLength(LinkList L)

{

int i = 0;//i用来统计次数，即表长

LinkList p = L->next;//先将p指向表头节点的后一个节点，即第一个节点

while(p)

{

i++;

p = p->next;//如果p指向不为空，i的次数加一，p指向下一个节点

}

return i;//返回次数i，即表长

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：GetElem

\*函数功能：得到某一个元素的值

\*注释：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

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

int j = 1;

LinkList p;

p = L->next;//p指向表头节点后的第一个节点

while(p && j < i)

{

p = p->next;//循环用来找到i位置节点

++j;

}

if(!p || j>i)

{

return ERROR;//用来判断输入位置是否正确，空表等

}

\*e = p->data;//用e取节点的元素

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：LocateElem

\*函数功能：查找元素

\*注释：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。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int LocateElem(LinkList L, ElemType e, status(\*compare)(ElemType a, ElemType b))

{

int i = 0;

LinkList p = L->next;//p指向第一个节点

while(p)

{

i++;

if((\*compare)(p->data, e))//通过遍历法比较得到所要找的元素

return i;//此时，找到了元素所在位置

p = p->next;//没有找到时，p指向下一个节点，循环

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：compare

\*函数功能：比较大小，服务于LocateList函数

\*注释：输入两个ElemType类型的值

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status compare(ElemType a, ElemType b)

{

if(a == b)

return TRUE;//比较输入的两个元素的大小，一样大则为TRUE

else

return FALSE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：PriorElem

\*函数功能：求元素的前驱

\*注释：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。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

LinkList p = L->next;//p指向第一个节点

if(p->data==cur\_e) return ERROR;//如果第一个节点就是要找的元素，则没有前驱

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

{

p = p->next;//通过循环，将p指针指向所要找的元素的前一个节点

}

if(p->next == NULL)//如果此时p指针指向空，则意味着表中没有该元素

return OVERFLOW;

\*pre\_e = p->data;//用pre\_e取出p指向的节点的元素，即输入元素的前驱

return OK;

}

/\*status PriorElem(LinkList L, ElemType cur\_e, ElemType \*pre\_e)

{

LinkList p,q;

p = L;

q = L->next;

if(q->data==cur\_e) return ERROR;

while(q!= NULL && q->data != cur\_e)

{

p = q;

q = q->next;

}

if(q== NULL && p->data !=cur\_e)

return OVERFLOW;

else if(q== NULL && p->data ==cur\_e){

return ERROR;

}

else {

\*pre\_e = p->data;

return OK;

}

}\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：NextElem

\*函数功能：求后继节点

\*输入输出：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。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

LinkList p = L->next;//p指向第一个节点

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

{

p = p->next;//循环的方式找到所要找的元素的前一个节点

}

if(p->next == NULL && p->data != cur\_e)//此时p指针指向空，p指向节点的值不是输入的元素，那么没有输入的元素

return ERROR;

if(p->next == NULL && p->data == cur\_e)//此时p指针指向空，p指向节点的值是输入的元素，那么没有后继节点

return OVERFLOW;

\*next\_e = p->next->data;//剩余正常情况，用next\_e取出p所指向节点的下一个节点的值

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListInsert

\*函数功能：插入元素

\*注释：初始条件是线性表L已存在且非空，1≤i≤ListLength(L)+1；

\* 操作结果是在L的第i个位置之前插入新的数据元素e

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

int j = 1;

LinkList p, q;//用两个指针确定插入位置的前后节点

p = \*L;//p指向表头节点（考虑到表头插入）

while(p && j < i)

{

p = p->next;//扎到插入位置

++j;

}

if(!p || j > i)

{

return ERROR;

}

q = (LinkList)malloc(sizeof(LNode));//给插入元素分配空间

if(q == NULL)

{

exit(OVERFLOW);//没分配成功

}

q->data = e;

q->next = p->next;//这三行代码是用来将元素存进去，同时将指针指向问题解决

p->next = q;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListDelete

\*函数功能：删除元素

\*注释：初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；

\* 操作结果：删除L的第i个数据元素，用e返回其值。

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

int j = 1;

LinkList p ,q;

p = \*L;//p指向表头节点

while(p->next && j < i)

{

p = p->next;//找到删除位置，同时p的下一个节点不为空

++j;

}

if(!(p->next) || j>i)//此时，根据p指针的情况 或者 j 的值判断

return ERROR;//即要么是空表，要么是删除位置不对。两者均可认为是删除位置不对

q = p->next;

p->next = q->next;//这四行代码是删除节点，并用e取出删除节点的值

\*e = q->data;

free(q);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListTrabverse

\*函数功能：遍历顺序表

\*注释：输出顺序表的值

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ListTrabverse(LinkList L)

{

LinkList p = L->next;

if(!p)//此时为空表

return ERROR;

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

while(p)

{

printf("%d ",p->data);//循环，输出值

p = p->next;

}

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

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：SaveList

\*函数功能：保存线性表

\*注释：将线性表保存，参考附录B，其中关于写入元素个数和长度的问题理解不够清楚

\*返回值类型：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status SaveList(LinkList L, char\* filename)

{

LinkList p = L->next;//指针指向第一个节点

int listsize=LIST\_INIT\_SIZE;//最大表长

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

{

printf("\t\t\t文件保存失败！");//文件保存失败

return ERROR;

}

fprintf(fp, "%d ", ListLength(L));//将元素个数写入

fprintf(fp, "%d ", listsize);//将最大长度写入

while(p)

{

fprintf(fp, "%d ", p->data);//指针op不指向空，依次写入文件

p = p->next;

}

fclose(fp);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：LoadList

\*函数功能：加载文件

\*注释：加载文件，以便功能的测试，文件名要正确

\*返回值类型：status类型

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status LoadList(LinkList \*L, char \*filename)

{

int i = 1,length = 0,listsize;

ElemType e;

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

{

printf("文件加载失败!");

return ERROR;

}

fscanf(fp, "%d ", &length);

fscanf(fp, "%d ", &listsize);

fscanf(fp, "%d ", &e);

while(i<=length)

{

ListInsert(L,i,e);

fscanf(fp, "%d ", &e);

i++;

}

fclose(fp);

return OK;

}

# 附录C 基于二叉链表的二叉树实现

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <stack>

#include <queue>

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

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define MAX\_NUM 10

typedef int status;

typedef char TElemType;

//数据元素类型定义

/\*-------page 22 on textbook -------\*/

#define LIST\_INIT\_SIZE 100

#define LISTINCRE MENT 10

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

typedef struct BiTNode{

int key;//用key作为标记，便于查找节点

TElemType data;//char类型数据域

struct BiTNode \*lchild, \*rchild;//定义二叉链表的左孩子指针与右孩子指针

}BiTNode, \*BiTree;//BiTNode类型指针BiTree

typedef struct {

BiTree T;//创建二叉树用的指针T

char name[20];//用于保存树的名称

}LElemType;

typedef struct {

LElemType tree[20];//多个树进行操作

int length;

int listsize;

}SqList;

status InitBiTree(BiTree \*T);

status DestroyBiTree(BiTree \*T);

status CreateBiTree(BiTree \*T);

status ClearBiTree(BiTree T);

status BiTreeEmpty(BiTree T);

status BiTreeDepth(BiTree T);

char Root(BiTree T);

char Value(BiTree T, int e);

status Assign(BiTree T, int e, char ch);

BiTNode\* Parent(BiTree T, int e);

BiTNode\* LeftChild(BiTree T, int e);

BiTNode\* RightChild(BiTree T, int e);

BiTNode\* RightSibling(BiTree T, int e);

BiTNode\* LeftSibling(BiTree T, int e);

BiTree Find(BiTree T, int key);

status InsertChild(BiTree T, BiTree p, int LR, BiTree c);

status DeleteChild(BiTree T, BiTree p, int LR);

status visit(char e);

status PreOrderTraverse(BiTree T, status (\* visit)(char e));

status InOrderTraverse(BiTree T, status (\* visit)(char e));

status PostOrderTraverse(BiTree T, status (\* visit)(char e));

status LevelOrderTraverse(BiTree T, status (\* visit)(char e));

status TreeDisplay(BiTree T,int depth,status (\* visit)(char e));

status Save(BiTree T,FILE \*fp);

status Load(BiTree \*T,FILE \*fp);

int main(){

FILE \*fp;

char filename[30];

SqList L;//相当于用数组构建多树操作的框架

BiTree T1, T2;//两个用BiTree构建的BiTNode指针

int op=1, key;//op用来case,便于用户操作;key是用来标记节点

int i, num=1, LR;//num默认为1，即默认在第一棵树进行操作，LR用来表示方向

char ch;//用来接收data域的值

for(i=0;i<20;i++)

{

L.tree[i].T = NULL;//相当于创建20个定义中类型的树的指针，同时指空，即20个树；

}

while(op){

system("cls");//系统函数，用于清屏

system("color F");

printf("\n\n");

printf("\t\t Menu for Binary Tree On Binary Linked List \n");

printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t1. InitBiTree \t 2. DestroyBiTree\n");

printf("\t\t\t3. CreateBiTree \t 4. ClearBiTree\n");

printf("\t\t\t5. BiTreeEmpty \t 6. BiTreeDepth\n");

printf("\t\t\t7. Root \t 8. Value\n");

printf("\t\t\t9. Assign \t 10. Parent\n");

printf("\t\t\t11. LeftChild \t 12. RightChild\n");

printf("\t\t\t13. LeftSibling \t 14. RightSibling\n");

printf("\t\t\t15. InsertChild \t 16. DeleteChild\n");

printf("\t\t\t17. PreOrderTraverse 18. InOrderTraverse\n");

printf("\t\t\t19. PostOrderTraverse 20. LevelOrderTraverse\n");

printf("\t\t\t21. Choose(多树操作) 22. Save(保存文件)\n");

printf("\t\t\t23. Load(加载文件) 0. Exit\n");

printf("\t\t\t24. TreeDisplay Powered By @\_@||王明明\n");

printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t请选择你的操作[0~24]:");

scanf("%d",&op);

switch(op)

{

case 1:

system("color 3");

InitBiTree(&(L.tree[num-1].T));

printf("\t\t\t二叉树创建成功！\n");

getchar();getchar();

break;

case 2:

system("color 2");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

DestroyBiTree(&(L.tree[num-1].T));

printf("\t\t\t二叉树销毁成功！\n");

getchar();getchar();

break;

case 3:

system("color F");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

getchar();

printf("\t\t\t请用前序方式构建二叉树，#表示空结点！\n");

CreateBiTree(&L.tree[num-1].T);

printf("\t\t\t二叉树构造成功！\n");

getchar();getchar();

break;

case 4:

system("color D");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

if(ClearBiTree(L.tree[num-1].T) == 1)

printf("\t\t\t叉树清空成功！\n");

else

printf("\t\t\t二叉树清空失败！\n");

getchar();getchar();

break;

case 5:

system("color 8");

if (L.tree[num-1].T==NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

if(BiTreeEmpty(L.tree[num-1].T) == OK)

printf("\t\t\t二叉树为空树！\n");

else

printf("\t\t\t二叉树不为空树！\n");

getchar();getchar();

break;

case 6:

system("color A");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

if(BiTreeEmpty(L.tree[num-1].T) == OK)

printf("\t\t\t二叉树为空树！\n");

else

printf("\t\t\t二叉树的深度为:%d\n", BiTreeDepth(L.tree[num-1].T));

getchar();getchar();

break;

case 7:

system("color C");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t二叉树的根为:%c\n", Root(L.tree[num-1].T));

getchar();getchar();

break;

case 8:

system("color D");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t请输入你要查找的key:");

scanf("%d", &key);

if (Value(L.tree[num-1].T, key) != ERROR)

{

printf("\t\t\t该key的data为：%c\n", Value(L.tree[num-1].T, key));

if (Parent(L.tree[num-1].T, key) != NULL)

{

printf("\t\t\t该key双亲的data为：%c\n", Parent(L.tree[num-1].T, key)->data);

printf("\t\t\t该key双亲的key为：%d\n", Parent(L.tree[num-1].T, key)->key);

}

else{

printf("\t\t\t该key双亲不存在！\n");

}

if (LeftChild(L.tree[num-1].T, key) != NULL)

{

printf("\t\t\t该key左孩子对应的data为：%c\n", LeftChild(L.tree[num-1].T, key)->data);

printf("\t\t\t该key左孩子对应的key为：%d\n", LeftChild(L.tree[num-1].T, key)->key);

}

else{

printf("\t\t\t该key左孩子不存在！\n");

}

if (RightChild(L.tree[num-1].T, key) != NULL)

{

printf("\t\t\t该key右孩子对应的data为：%c\n", RightChild(L.tree[num-1].T, key)->data);

printf("\t\t\t该key右孩子对应的key为：%d\n", RightChild(L.tree[num-1].T, key)->key);

}

else{

printf("\t\t\t该key右孩子不存在！\n");

}

if (LeftSibling(L.tree[num-1].T, key) != NULL)

{

printf("\t\t\t该key左兄弟对应的data为：%c\n", LeftSibling(L.tree[num-1].T, key)->data);

printf("\t\t\t该key左兄弟对应的key为：%d\n", LeftSibling(L.tree[num-1].T, key)->key);

}

else{

printf("\t\t\t该key左兄弟不存在！\n");

}

if (RightSibling(L.tree[num-1].T, key) != NULL)

{

printf("\t\t\t该key右兄弟对应的data为：%c\n", RightSibling(L.tree[num-1].T, key)->data);

printf("\t\t\t该key右兄弟对应的key为：%d\n", RightSibling(L.tree[num-1].T, key)->key);

}

else{

printf("\t\t\t该key右兄弟不存在！\n");

}

}

else

printf("\t\t\t输入的key在树中不存在！\n");

getchar();getchar();

break;

case 9:

system("color A");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t请输入你要查找的key:");

scanf("%d", &key);

getchar();

printf("\t\t\t请输入你要重新赋值的data:");

scanf("%c", &ch);

if (Assign(L.tree[num-1].T, key, ch) != ERROR)

printf("\t\t\t该key对应的data改为了：%c\n", ch);

else

printf("\t\t\t输入的key在树种不存在！\n");

getchar();getchar();

break;

case 10:

system("color B");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t请输入你要查找父母的key:");

scanf("%d", &key);

if (Parent(L.tree[num-1].T, key) != NULL)

{

printf("\t\t\t双亲对应的data为：%c\n", Parent(L.tree[num-1].T, key)->data);

printf("\t\t\t双亲对应的key为：%d\n", Parent(L.tree[num-1].T, key)->key);

}

else

printf("\t\t\t双亲不存在！\n");

getchar();getchar();

break;

case 11:

system("color C");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t请输入你要查找左孩子的key: ");

scanf("%d", &key);

if (LeftChild(L.tree[num-1].T, key) != NULL)

{

printf("\t\t\t左孩子对应的data为：%c\n", LeftChild(L.tree[num-1].T, key)->data);

printf("\t\t\t左孩子对应的key为：%d\n", LeftChild(L.tree[num-1].T, key)->key);

}

else

printf("\t\t\t左孩子不存在！\n");

getchar();getchar();

break;

case 12:

system("color D");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t请输入你要查找右孩子的key: ");

scanf("%d", &key);

if (RightChild(L.tree[num-1].T, key) != NULL)

{

printf("\t\t\t右孩子对应的data为：%c\n", RightChild(L.tree[num-1].T, key)->data);

printf("\t\t\t右孩子对应的key为：%d\n", RightChild(L.tree[num-1].T, key)->key);

}

else

printf("\t\t\t右孩子不存在！\n");

getchar();getchar();

break;

case 13:

system("color E");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t请输入你要查找左兄弟的key: ");

scanf("%d", &key);

if (LeftSibling(L.tree[num-1].T, key) != NULL)

{

printf("\t\t\t左兄弟对应的data为：%c\n", LeftSibling(L.tree[num-1].T, key)->data);

printf("\t\t\t左兄弟对应的key为：%d\n", LeftSibling(L.tree[num-1].T, key)->key);

}

else

printf("\t\t\t左兄弟不存在！\n");

getchar();getchar();

break;

case 14:

system("color F");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t请输入你要查找右兄弟的key: ");

scanf("%d", &key);

if (RightSibling(L.tree[num-1].T, key) != NULL)

{

printf("\t\t\t右兄弟对应的data为：%c\n", RightSibling(L.tree[num-1].T, key)->data);

printf("\t\t\t右兄弟对应的key为：%d\n", RightSibling(L.tree[num-1].T, key)->key);

}

else

printf("\t\t\t右兄弟不存在！\n");

getchar();getchar();

break;

case 15:

system("color D");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t请输入你要插入位置的key: ");

scanf("%d", &key);

getchar();

T1 = Find(L.tree[num-1].T, key);

printf("\t\t\t请输入你想要插入的方向,0表示左边,1表示右边: ");

scanf("%d", &LR);

getchar();

CreateBiTree(&T2);

if (InsertChild(L.tree[num-1].T, T1, LR, T2) == OK)

{

printf("\t\t\t插入成功!\n");

getchar(); getchar();

}

else

{

printf("\t\t\t插入失败!\n");

getchar(); getchar();

}

break;

case 16:

system("color A");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t请输入你要删除位置的key: ");

scanf("%d", &key);

getchar();

T1 = Find(L.tree[num-1].T, key);

printf("\t\t\t请输入你想要删除的方向,0表示左边,1表示右边: ");

scanf("%d", &LR);

getchar();

if (DeleteChild(L.tree[num-1].T, T1, LR) == OK)

{

printf("\t\t\t删除成功!\n");

getchar(); getchar();

}

else

{

printf("\t\t\t删除失败!\n");

getchar(); getchar();

}

break;

case 17:

system("color C");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t该二叉树的前序遍历为：\n");

PreOrderTraverse(L.tree[num-1].T, visit);

getchar();getchar();

break;

case 18:

system("color E");

if (!L.tree[num-1].T)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t该二叉树的中序遍历为：\n");

InOrderTraverse(L.tree[num-1].T, visit);

getchar();getchar();

break;

case 19:

system("color F");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t该二叉树的后序遍历为：\n");

PostOrderTraverse(L.tree[num-1].T, visit);

getchar();getchar();

break;

case 20:

system("color D");

if (L.tree[num-1].T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

printf("\t\t\t该二叉树的层序遍历为：\n");

LevelOrderTraverse(L.tree[num-1].T, visit);

getchar();getchar();

break;

case 21:

system("color 2");

printf("\t\t\t在第几个树操作?只支持20个树进行操作: \n");

scanf("%d",&num);

if(num<1||num>20)

{

printf("\t\t\t请选择正确范围！\n");

num=1;

}

getchar(); getchar();

break;

case 22:

system("color D");

printf("\t\t\t请输入要保存的文件名: ");

scanf("%s", filename);

if((fp=fopen(filename,"w"))==NULL) printf("\t\t\t打开文件失败！\n");

else

{

if(Save(L.tree[num-1].T,fp)==OK)

printf("\t\t\t保存文件成功！\n");

else printf("\t\t\t保存文件失败！\n");

}

fclose(fp);

getchar();getchar();

break;

case 23:

system("color D");

printf("\t\t\t请输入要加载的文件名: ");

scanf("%s",filename);

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

printf("\t\t\t加载失败！\n");

else

{

if(Load(&L.tree[num-1].T,fp)==OK)

printf("\t\t\t加载成功！\n");

else printf("\t\t\t加载失败！\n");

}

fclose(fp);

getchar();getchar();

break;

case 24:

system("color A");

if (!L.tree[num-1].T)

{

printf("\t\t\t二叉树不存在！\n");

getchar();getchar();

break;

}

if(BiTreeEmpty(L.tree[num-1].T) == OK)

printf("\t\t\t二叉树为空树！\n");

else{

printf("\t\t\t该二叉树为:\n");

TreeDisplay((L.tree[num-1].T),1, visit);

}

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("\n");

printf("\t\t\t欢迎下次再使用本系统！\n\n");

system("color 9");

printf("\t\t\tPowered By@\_@||王明明\n\n");

//system("pause");

}//end of main()

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

/\*\*

\* 函数名称: InitBiTree

\* 初始条件：二叉树T不存在

\* 操作结果: 构造空树二叉树T

\*

\*/

status InitBiTree(BiTree \*T) {

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

(\*T)->lchild = NULL;

(\*T)->rchild = NULL;

(\*T)->data='#';//初始化二叉树,将左右指针指向空,data域设为#

return OK;

}

/\*\*

\* 函数名称: DestroyBiTree

\* 初始条件：二叉树T已存在

\* 操作结果: 销毁二叉树T

\*

\*/

status DestroyBiTree(BiTree \*T)

{

if (\*T) {

if ((\*T)->lchild)

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

if ((\*T)->rchild)

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

free(\*T);

(\*T) = NULL;//使用递归依次释放左子树、右子树、根节点指针

}

return OK;

}

/\*\*

\* 函数名称: CreateBiTree

\* 初始条件：二叉树T已存在

\* 操作结果: 创建二叉树

\*

\*/

status CreateBiTree(BiTree \*T)

{

char ch;

printf("\t\t\t请输入data: ");

scanf("%c", &ch);

getchar();

if(ch == '#')

{

\*T = NULL;

return 0;

}

else

{

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

(\*T)->data = ch;

printf("\t\t\t请输入key:");

scanf("%d", &(\*T)->key);

getchar();

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

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

}

return OK;

}

//取材于课本,依次输入节点值，并用key标记节点

/\*\*

\* 函数名称: ClearBiTree

\* 初始条件：二叉树T已初始化

\* 操作结果: 构造二叉树

\*

\*/

status ClearBiTree(BiTree T)

{

if(T)

{

T->lchild = NULL;

T->rchild = NULL;

T->data = '#';//清空二叉树,左右子树指针指向空,data域设置为#

}

return OK;

}

/\*\*

\* 函数名称: BiTreeEmpty

\* 初始条件：二叉树T已存在

\* 操作结果: 若T为空二叉树则返回TRUE，否则返回FALSE

\*

\*/

status BiTreeEmpty(BiTree T)

{

if(T->data=='#')//判断根节点的data域值是否为空

{

return OK;

}

else

{

return ERROR;

}

}

/\*\*

\* 函数名称: BiTreeDepth

\* 初始条件：二叉树T已存在

\* 操作结果: 返回T的深度

\*

\*/

status BiTreeDepth(BiTree T)

{

int depth = 0;

if(T)

{

int lchilddepth = BiTreeDepth(T->lchild);

int rchilddepth = BiTreeDepth(T->rchild);

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

}//使用递归,得到左右子树的深度,并比较大小,之后返回最大的深度

return depth;

}

/\*\*

\* 函数名称: Root

\* 初始条件：二叉树T已存在

\* 操作结果: 返回T的根

\*

\*/

char Root(BiTree T)

{

return T->data;//T的data域的值即根节点

}

/\*\*

\* 函数名称: visit

\* 操作结果: 打印字符e

\*/

status visit(char e)

{

printf("%c",e);//依次调用该函数，用来打印

}

/\*\*

\* 函数名称: Value

\* 初始条件：二叉树T已存在, e是T中的某个结点

\* 操作结果: 返回e的值

\*

\*/

char Value(BiTree T, int e)

{

if (!T) return ERROR; //若二叉树为空,返回ERROR

BiTNode \*st[10], \*p;

int top = 0; //置空栈

st[top++] = T;

while (top)

{

p = st[--top]; //先序遍历,弹出栈顶元素,判断是否有key的值与e相等

if (p->key == e)

{

return p->data;

}

else {

if(p->rchild!=NULL)

st[top++] = p->rchild;

if(p->lchild!=NULL)

st[top++] = p->lchild;

}

}

return ERROR;

}

/\*\*

\* 函数名称: Assign

\* 初始条件：二叉树T已存在, e是T中的某个结点

\* 操作结果: 结点e赋值为value

\*

\*/

status Assign(BiTree T, int e, char ch)

{

if (!T) return ERROR;

BiTNode \*st[10], \*p;

int top = 0;

st[top++] = T;

while (top)

{

p = st[--top];

if (p->key == e)

{

p->data =ch;//找到后进行复制,与Value函数相似

return OK;

}

else

{

if (p->rchild != NULL)

st[top++] = p->rchild;

if (p->lchild != NULL)

st[top++] = p->lchild;

}

}

return ERROR;

}

/\*\*

\* 函数名称: Parent

\* 初始条件：二叉树T已存在, e是T中的某个结点

\* 操作结果: 若e是T的非根结点，则返回它的双亲结点指针，否则返回NULL

\*

\*/

BiTNode\* Parent(BiTree T, int e)

{

BiTree T1;//利用遍历和递归依次寻找左孩子右孩子对应的e值,符合条件便返回指针

if (T)

{

if ((T->lchild!=NULL&&T->lchild->key == e) ||(T->rchild!=NULL&& T->rchild->key == e)) return T;

T1 = Parent(T->lchild, e);

if (T1 != NULL) return T1;

T1 = Parent(T->rchild, e);

if (T1 != NULL) return T1;

}

return NULL;

}

/\*\*

\* 函数名称: LeftChild

\* 初始条件：二叉树T已存在, e是T中的某个结点

\* 操作结果: 返回e的左孩子结点指针。若e无左孩子，则返回NULL

\*

\*/

BiTNode\* LeftChild(BiTree T, int e)

{

BiTree p;//利用递归找左孩子,思路简单

if (T)

{

if (T->key == e) return T->lchild;

p = LeftChild(T->lchild, e);

if (p != NULL) return p;

p = LeftChild(T->rchild, e);

if (p != NULL) return p;

}

return NULL;

}

/\*\*

\* 函数名称: RightChild

\* 初始条件：二叉树T已存在, e是T中的某个结点

\* 操作结果: 返回e的右孩子结点指针。若e无右孩子，则返回NULL

\*

\*/

BiTNode\* RightChild(BiTree T, int e)

{

BiTree p;//与上一个同理

if (T)

{

if (T->key == e) return T->rchild;

p = RightChild(T->lchild, e);

if (p != NULL) return p;

p = RightChild(T->rchild, e);

if (p != NULL) return p;

}

return NULL;

}

/\*\*

\* 函数名称: LeftSibling

\* 初始条件：二叉树T已存在, e是T中的某个结点

\* 操作结果: 返回e的左兄弟结点指针。若e是T的左孩子或者无左兄弟，则返回NULL

\*

\*/

BiTNode\* LeftSibling(BiTree T, int e)

{

BiTree p=NULL;

if (T)

{

if (T->rchild!=NULL&&T->rchild->key == e)//如果右孩子不为空并且节点的值符合,那么便返回左孩子的指针

return T->lchild;

p = LeftSibling(T->lchild, e);

if (p != NULL) return p;

p = LeftSibling(T->rchild, e);

if (p != NULL) return p;

}

return NULL;

}

/\*\*

\* 函数名称: RightSibling

\* 初始条件：二叉树T已存在, e是T中的某个结点

\* 操作结果: 返回e的左兄弟结点指针。若e是T的左孩子或者无左兄弟，则返回NULL回e的右兄弟结点指针。若e是T的右孩子或者无有兄弟，则返回NULL

\*

\*/

BiTNode\* RightSibling(BiTree T, int e)

{

BiTree p=NULL;

if (T)

{

if (T->lchild!=NULL&&T->lchild->key == e)//与上一个函数类似,最后依次递归遍历左子树右子树

return T->rchild;

p = RightSibling(T->lchild, e);

if (p != NULL) return p;

p = RightSibling(T->rchild, e);

if (p != NULL) return p;

}

return NULL;

}

BiTree Find(BiTree T, int key)

{

BiTree T1;

if (T == NULL) return NULL;

if (T->key == key) return T;

else

{

T1 = Find(T->lchild, key);

if (T1 != NULL) return T1;

T1 = Find(T->rchild, key);

if (T1 != NULL) return T1;

}

return NULL;

}

/\*\*

\* 函数名称: InsertChild

\* 初始条件：二叉树T存在，p指向T中的某个结点，LR为0或1

\* 操作结果: 根据LR为0或者1，插入c为T中p所指结点的左或右子树，p 所指结点的原有左子树或右子树则为c的右子树

\*

\*/

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

{

if (!T)

{

printf("\t\t\t二叉树不存在！");

return ERROR;

}

if (c->rchild != NULL)

{

printf("\t\t\t待插入二叉树的右子树不为空！");

return ERROR;

}

if (LR == 0)

{

c->rchild = p->lchild;//插入方向为左的情况

p->lchild = c;

}

else

{

c->rchild = p->rchild;//插入方向为右的情况

p->rchild = c;

}

return OK;

}

/\*\*

\* 函数名称: DeleteChild

\* 初始条件：二叉树T存在，p指向T中的某个结点，LR为0或1

\* 操作结果: 根据LR为0或者1，删除c为T中p所指结点的左或右子树

\*

\*/

status DeleteChild(BiTree T, BiTree p, int LR)

{

BiTree T1;

if (T == NULL)

{

printf("\t\t\t二叉树不存在！\n");

return ERROR;

}

if (LR == 0)//删除左子树

{

T1 = p->lchild;

p->lchild = NULL;

if(!DestroyBiTree(&T1)) return ERROR;

}

else//删除右子树

{

T1 = p->rchild;

p->rchild = NULL;

if (!DestroyBiTree(&T1)) return ERROR;

}

return OK;

}

/\*\*

\* 函数名称: PreOrderTraverse

\* 初始条件：二叉树T已存在

\* 操作结果: 先序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

\*

\*/

status PreOrderTraverse(BiTree T, status (\* visit)(char e))

{

if(T)

{

if(visit(T->data))//先序遍历，利用递归方式

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

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

return 1;

return 0;

}

else return 1;

}

/\*\*

\* 函数名称: InOrderTraverse

\* 初始条件：二叉树T已存在

\* 操作结果: 中序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

\*

\*/

status InOrderTraverse(BiTree T, status (\* visit)(char e))

{

if(T)

{

if(InOrderTraverse(T->lchild, visit))

if(visit(T->data))//中序遍历，利用递归方式

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

return 1;

return 0;

}

else return 1;

}

/\*\*

\* 函数名称: PostOrderTraverse

\* 初始条件：二叉树T已存在

\* 操作结果: 后序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

\*

\*/

status PostOrderTraverse(BiTree T, status (\* visit)(char e))

{

if(T)

{

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

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

if(visit(T->data))//后序遍历，利用递归方式

return 1;

return 0;

}

else return 1;

}

void level(BiTree T, int i)

{

if(T)

{

if(i == 1)

visit(T->data);//该函数用来辅助层序遍历

else

{

level(T->lchild, i-1);

level(T->rchild, i-1);

}

}

}

/\*\*

\* 函数名称: LevelOrderTraverse

\* 初始条件：二叉树T已存在

\* 操作结果: 层序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

\*

\*/

status LevelOrderTraverse(BiTree T, status (\* visit)(char e))

{

if(T)

{

int h = BiTreeDepth(T);//调用函数,得到深度

int i;

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

{

level(T, i);//对每一层进行访问

}

return OK;

}

else

return OK;

}

/\*\*

\* 函数名称: Save

\* 初始条件：二叉树T已存在

\* 操作结果: 保存二叉树为文件

\*

\*/

status Save(BiTree T,FILE \*fp)

{

int i = 0;

char ch = '#';

if(T)

{

fprintf(fp,"%d%c",T->key,T->data);//将key和data依次写入文件

if(Save(T->lchild,fp))

if(Save(T->rchild,fp)) return OK;

else return ERROR;

}

else

{

fprintf(fp,"%d%c",i,ch);//i和ch初始设置,意味着空节点的key为0,#代表空节点.

return OK;

}

}

/\*\*

\* 函数名称: Load

\* 操作结果: 从文件中读取二叉树

\*

\*/

status Load(BiTree \*T,FILE \*fp)

{

int i;//用i来表示key

char ch;//用ch来读取data域

if(feof(fp))

{

(\*T)=NULL;

return OK;

}

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

fscanf(fp,"%c",&ch);

if(ch=='#')

{

(\*T)=NULL;

}

else

{

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

(\*T)->key=i;

(\*T)->data=ch;

Load(&((\*T)->lchild),fp);

Load(&((\*T)->rchild),fp);

}//相当于依次读取之后创建为二叉树

return OK;

}

status TreeDisplay(BiTree T,int depth,status (\* visit)(char e))

{

if(!T)

{

printf("\n");

return OK;

}

int i=0;

for(; i<depth; i++)

printf(" ");

visit(T->data);

printf("\n");

if(T->lchild||T->rchild)

{

TreeDisplay(T->lchild,depth+1,visit);

TreeDisplay(T->rchild,depth+1,visit);

}

return OK;

}