

**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 12月 31 日**

**计算机科学与技术学院**

目 录

[1 基于顺序存储结构的线性表实现 3](#_Toc533958721)

[1.1 问题描述 3](#_Toc533958722)

[1.2 系统设计 4](#_Toc533958723)

[1.3 系统实现 16](#_Toc533958727)

[1.4 实验小结 28](#_Toc533958730)

[2 基于链式存储结构的线性表实现 30](#_Toc533958731)

[2.1 问题描述 30](#_Toc533958732)

[2.2 系统设计 31](#_Toc533958733)

[2.3 系统实现 46](#_Toc533958737)

[2.4 实验小结 58](#_Toc533958740)

[3 基于二叉链表的二叉树实现 60](#_Toc533958741)

[3.1 问题描述 60](#_Toc533958742)

[3.2 系统设计 62](#_Toc533958743)

[3.3 系统实现 71](#_Toc533958747)

[3.4 实验小结 91](#_Toc533958750)

[4 基于邻接表的无向图实现 93](#_Toc533958751)

[4.1 问题描述 93](#_Toc533958752)

[4.2 系统设计 94](#_Toc533958753)

[4.3 系统实现 100](#_Toc533958757)

[4.4 实验小结 113](#_Toc533958760)

[参考文献 114](#_Toc533958761)

# 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！ |

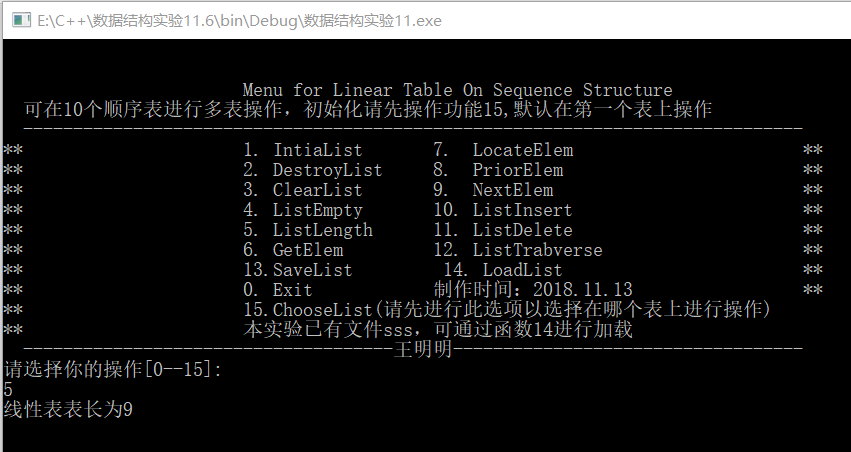
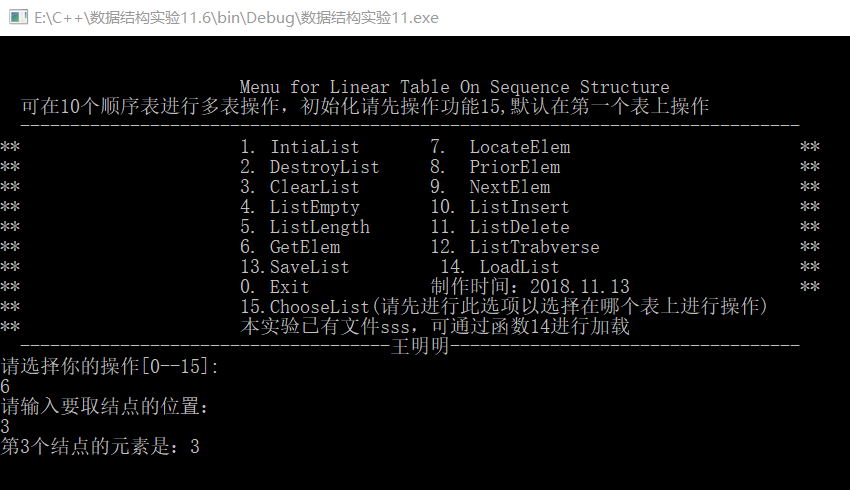


图1-17 ListLength测试截图

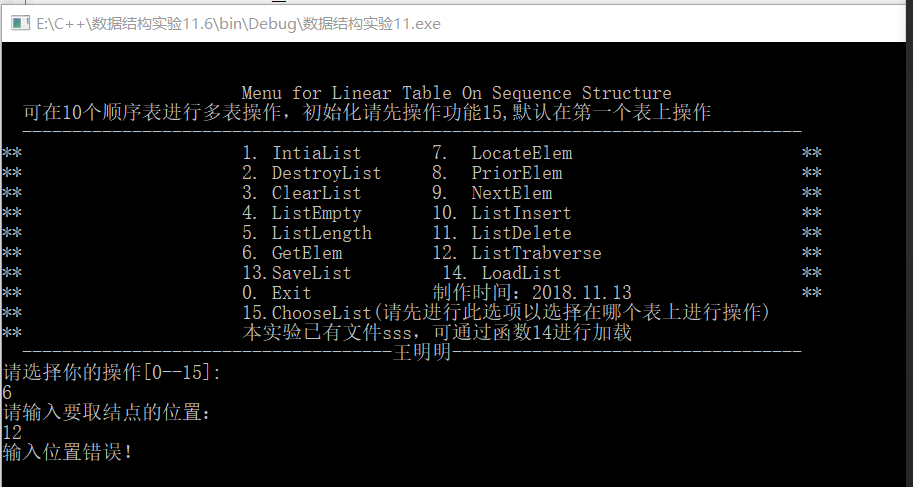
(3) GetElem测试

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



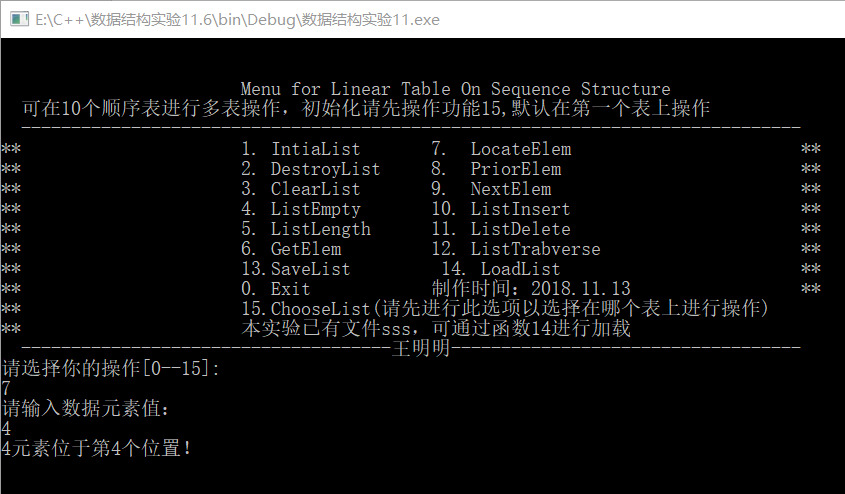
(b)输入位置错误时运行截图

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

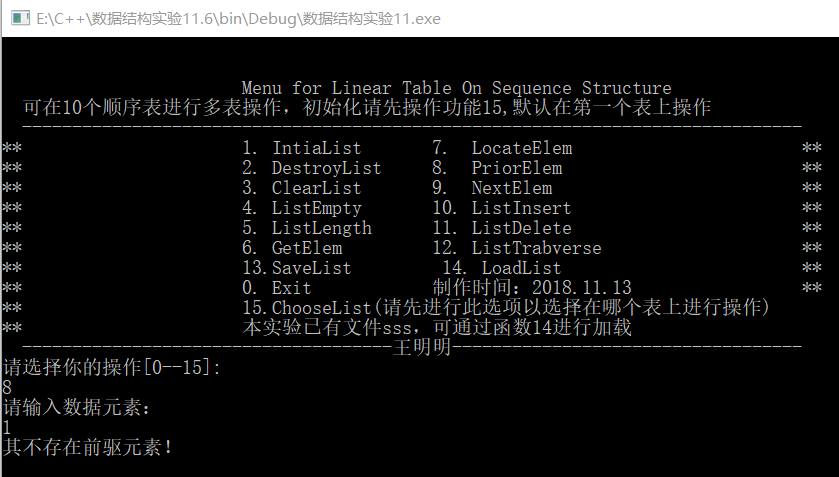
(5) PriorElem测试

表1-5 PriorElem测试用例表

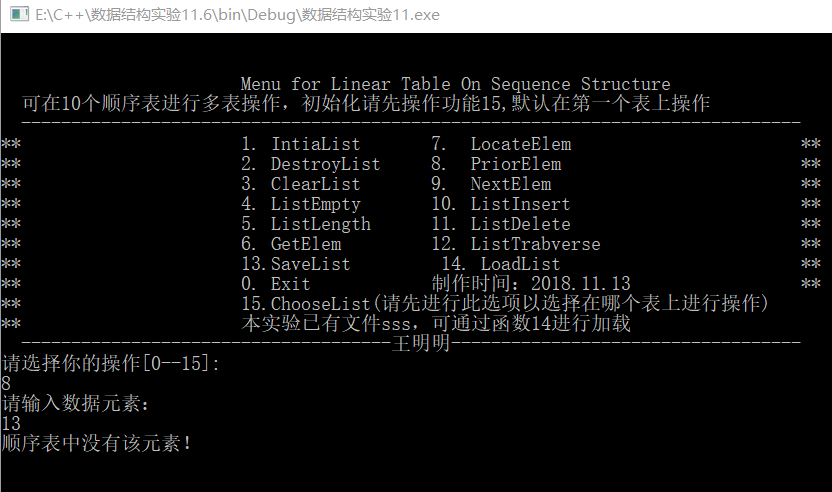
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选8  输入元素2 | 其前驱元素为：1 | 其前驱元素为：1 |
| sss | 界面选8  输入元素1 | 其不存在前驱元素！ | 其不存在前驱元素！ |
| sss | 界面选8  输入元素13 | 顺序表中没有该元素！ | 顺序表中没有该元素！ |
| 若表不存在 | 界面选8 | 线性表不存在！ | 线性表不存在！ |



(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



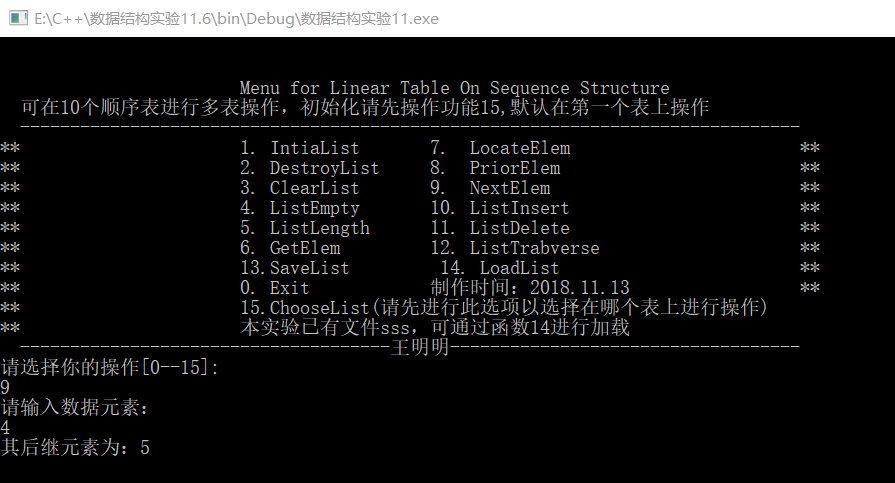
(c)输入元素不在顺序表中时

图1-20 LocateElem测试截图

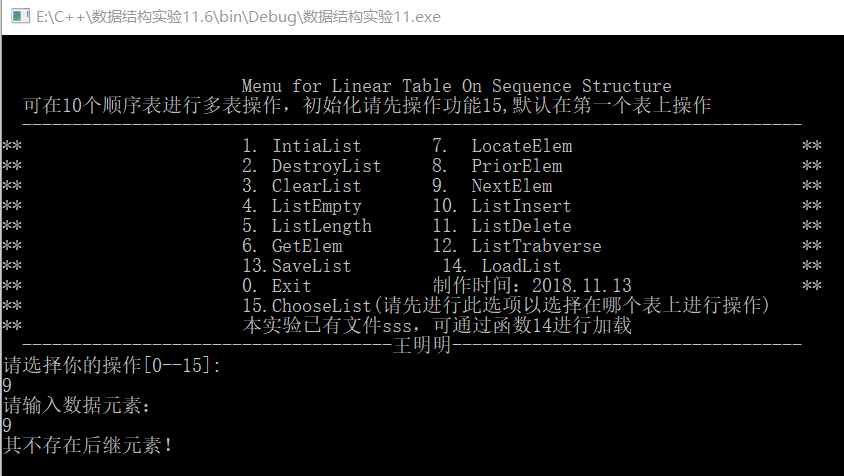
(6) NextElem测试

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



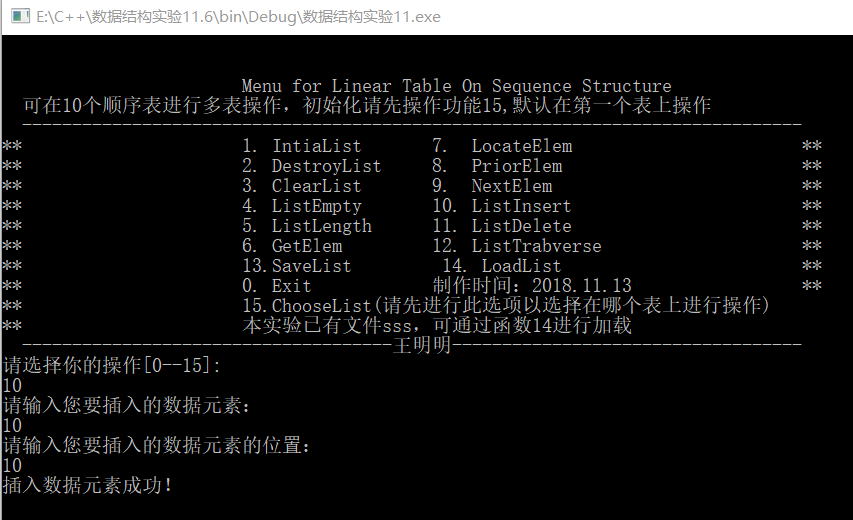
(c)输入元素不存在

图1-21 NextElem测试截图

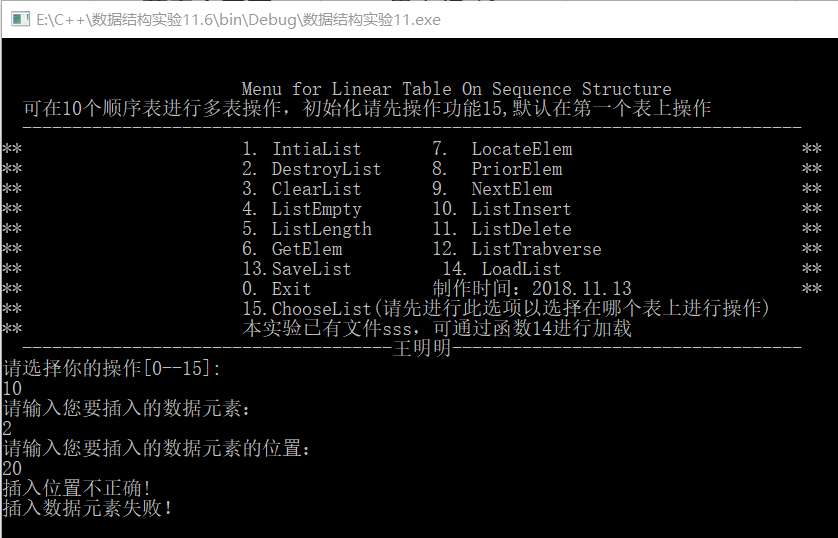
(7) ListInsert测试

表1-7 ListInsert测试用例表

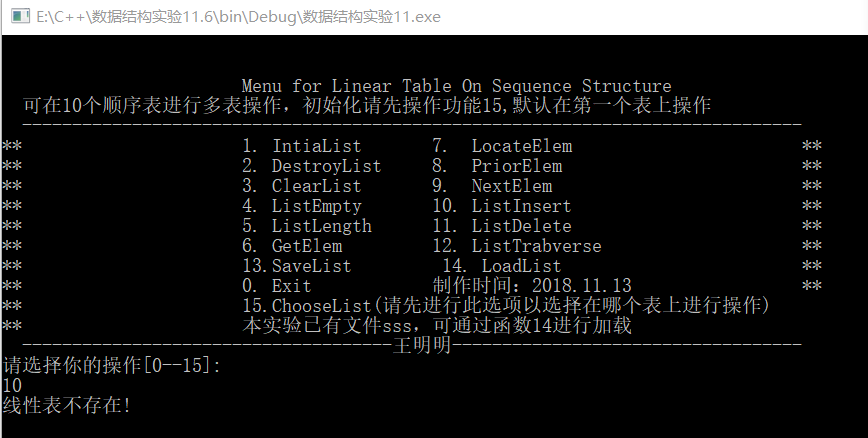
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选10  输入元素10  输入位置10 | 插入数据元素成功！ | 插入数据元素成功！ |
| sss | 界面选10  输入元素2  插入位置20 | 插入位置不正确！  插入数据元素失败！ | 插入位置不正确！  插入数据元素失败！ |
| 若表不存在 | 界面选10 | 线性表不存在！ | 线性表不存在！ |



(a)插入位置正确时截图



(b)插入元素位置不正确时截图



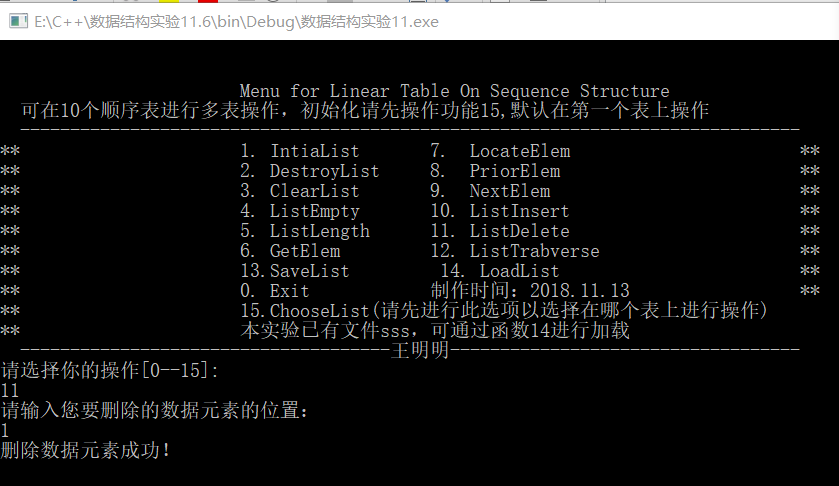
(c)线性表不存在时截图

图1-22 ListInsert测试截图

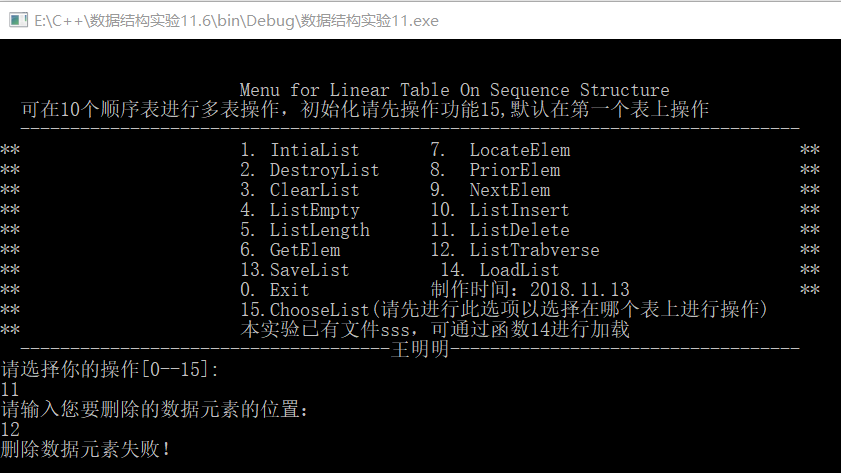
(8) ListDelete测试

表1-8 ListDelete测试用例表

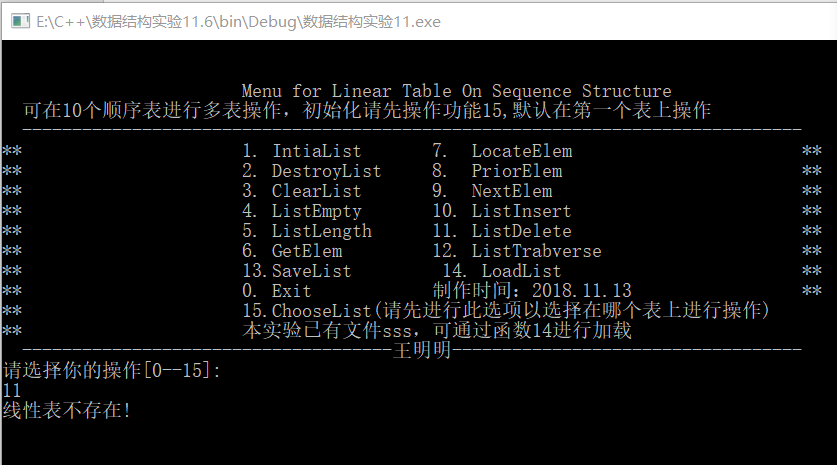
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 选11、输入1 | 删除元素成功！ | 删除元素成功！ |
| sss | 选11、输入12 | 删除元素失败！ | 删除元素失败！ |
| 若表不存在 | 选11 | 线性表不存在！ | 线性表不存在！ |



(a)正常删除元素运行截图



(b) 删除位置不正确运行截图



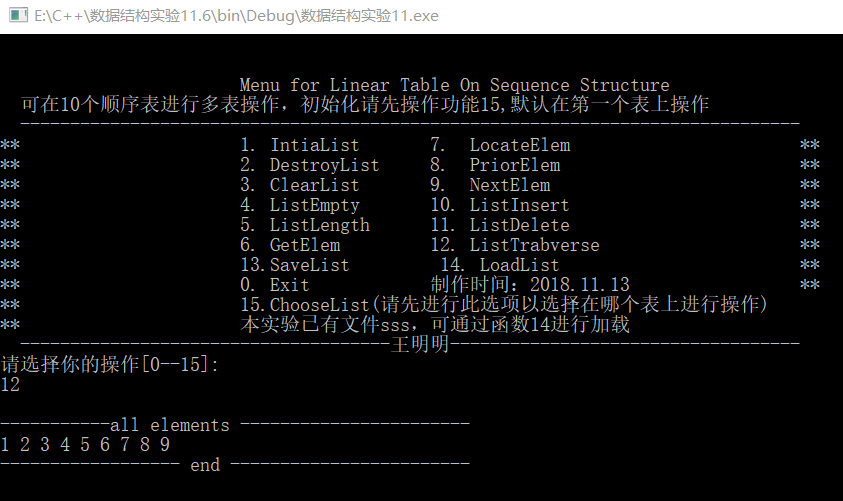
(c)线性表不存在运行截图

图1-23 ListDelete测试截图

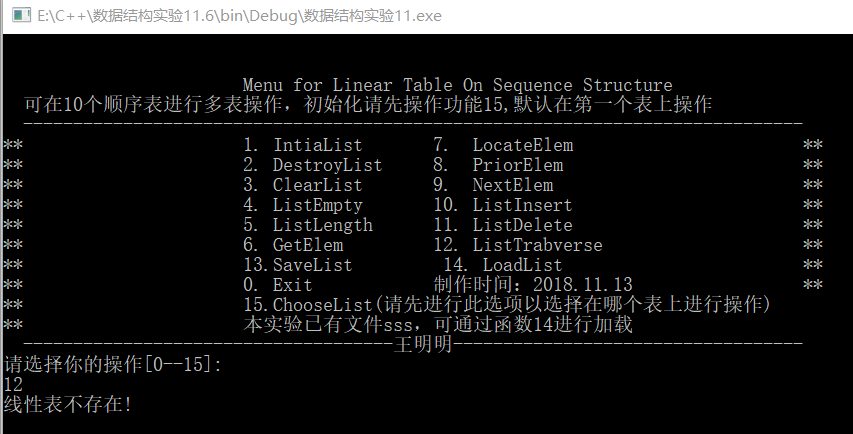
(9) ListTrabverse测试

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！ |

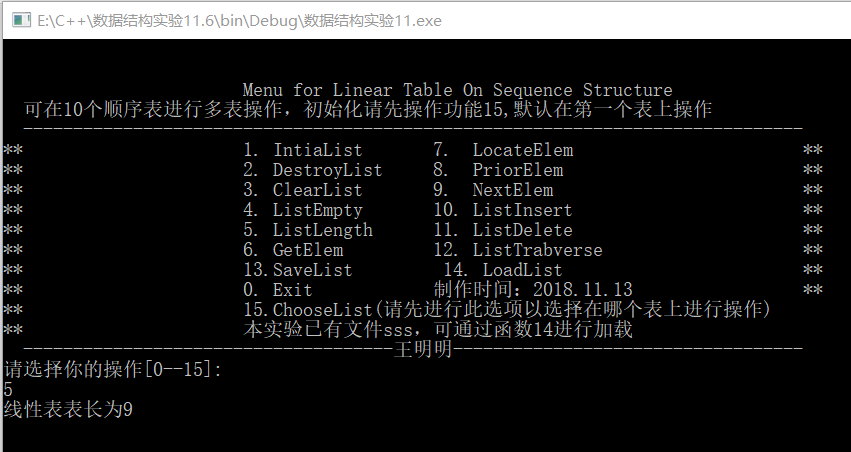
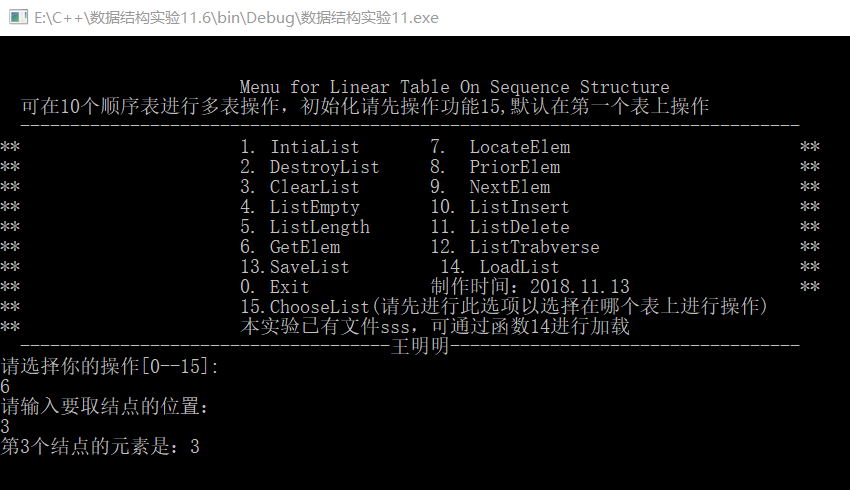


图2-17 ListLength测试截图

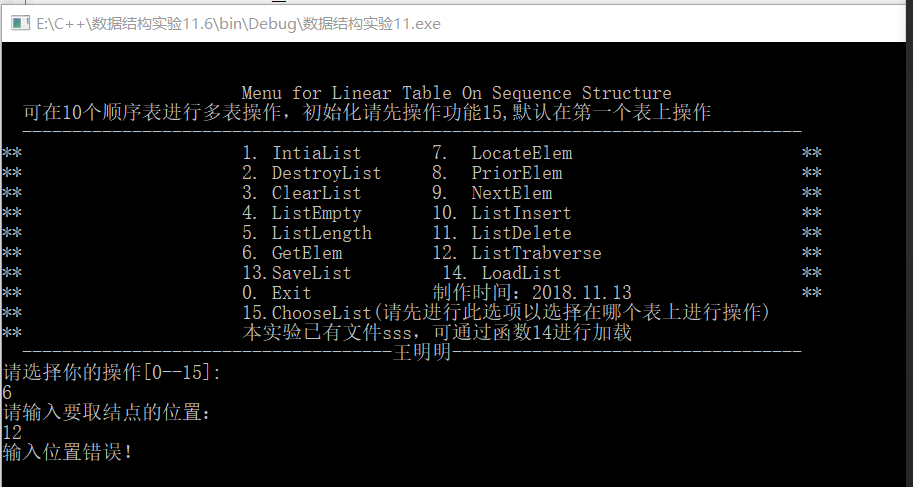
(3) GetElem测试

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



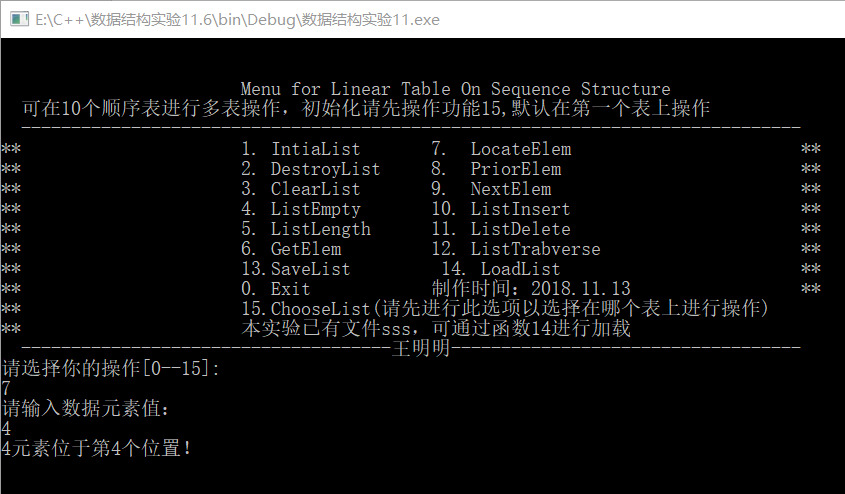
(b)输入位置错误时运行截图

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

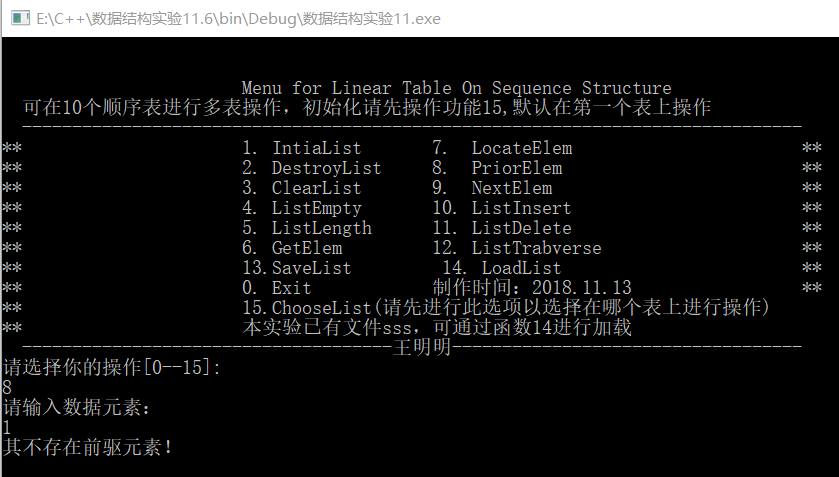
(5) PriorElem测试

表2-5 PriorElem测试用例表

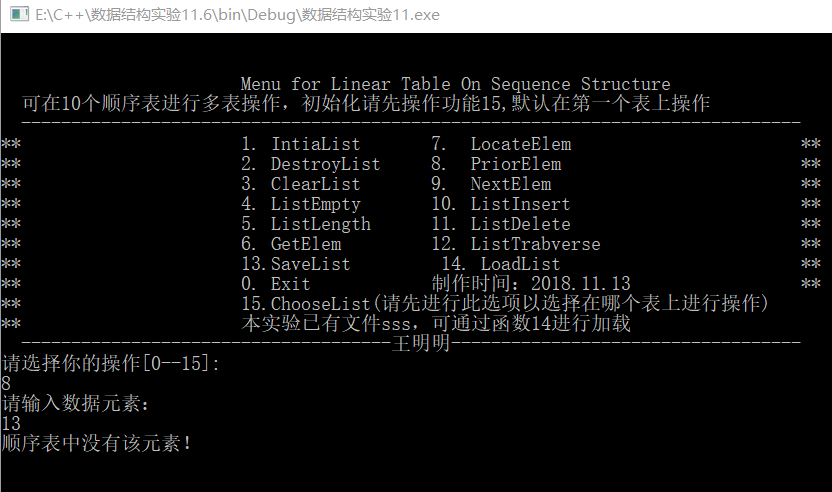
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选8  输入元素2 | 其前驱元素为：1 | 其前驱元素为：1 |
| sss | 界面选8  输入元素1 | 其不存在前驱元素！ | 其不存在前驱元素！ |
| sss | 界面选8  输入元素13 | 顺序表中没有该元素！ | 顺序表中没有该元素！ |
| 若表不存在 | 界面选8 | 线性表不存在！ | 线性表不存在！ |



(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



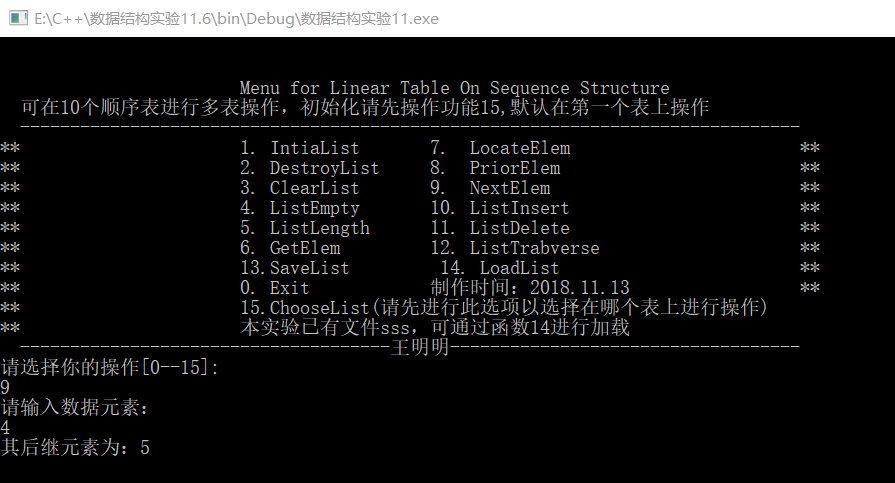
(c)输入元素不在顺序表中时

图2-20 LocateElem测试截图

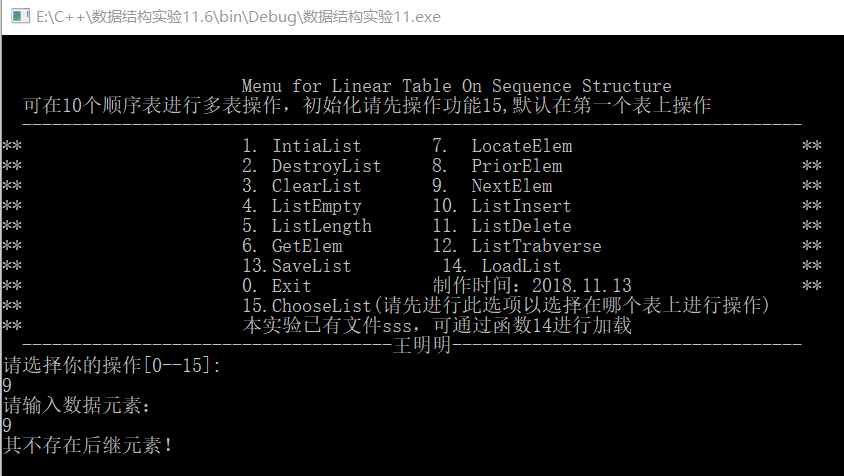
(6) NextElem测试

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



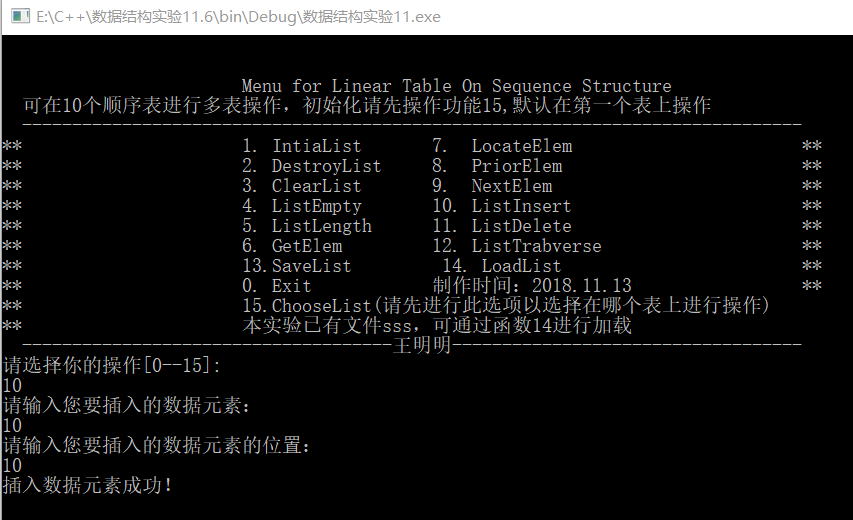
(c)输入元素不存在

图2-21 NextElem测试截图

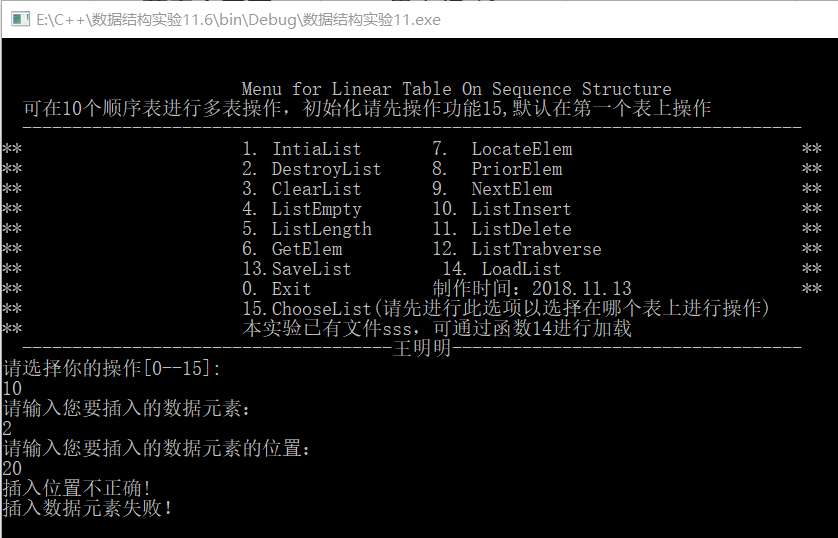
(7) ListInsert测试

表2-7 ListInsert测试用例表

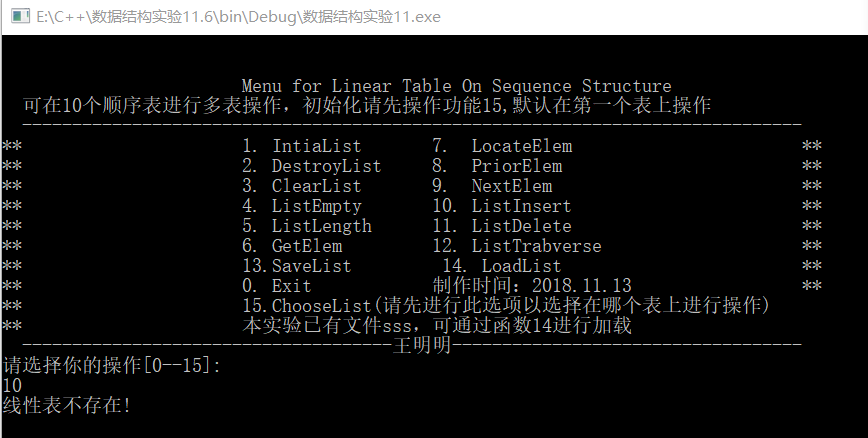
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 界面选10  输入元素10  输入位置10 | 插入数据元素成功！ | 插入数据元素成功！ |
| sss | 界面选10  输入元素2  插入位置20 | 插入位置不正确！  插入数据元素失败！ | 插入位置不正确！  插入数据元素失败！ |
| 若表不存在 | 界面选10 | 线性表不存在！ | 线性表不存在！ |



(a)插入位置正确时截图



(b)插入元素位置不正确时截图



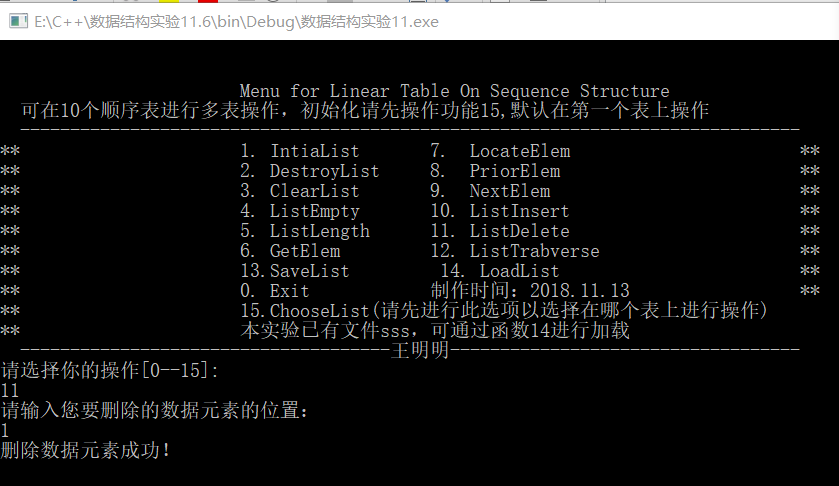
(c)线性表不存在时截图

图2-22 ListInsert测试截图

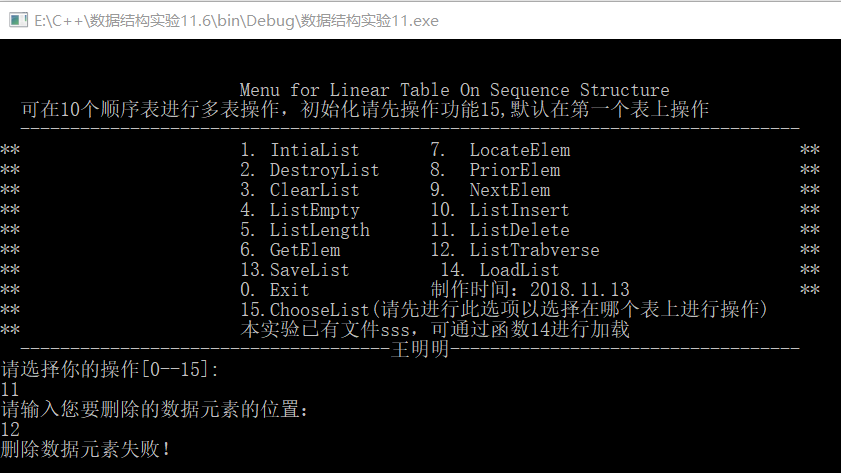
(8) ListDelete测试

表2-8 ListDelete测试用例表

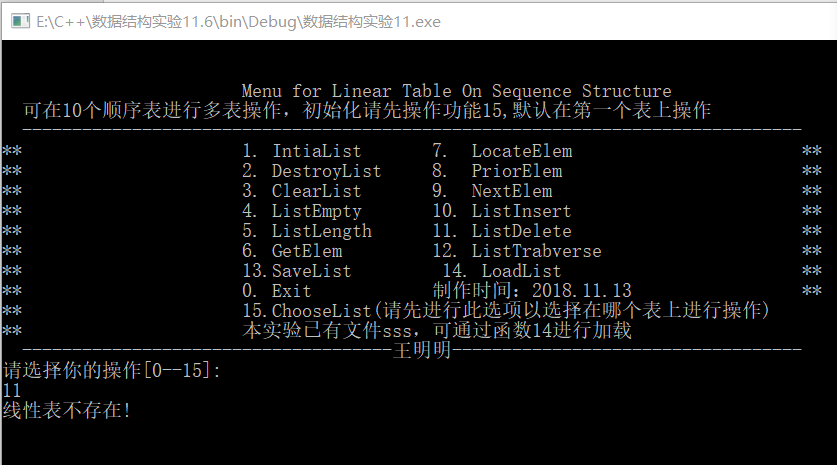
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| sss | 选11、输入1 | 删除元素成功！ | 删除元素成功！ |
| sss | 选11、输入12 | 删除元素失败！ | 删除元素失败！ |
| 若表不存在 | 选11 | 线性表不存在！ | 线性表不存在！ |



(a)正常删除元素运行截图



(b) 删除位置不正确运行截图



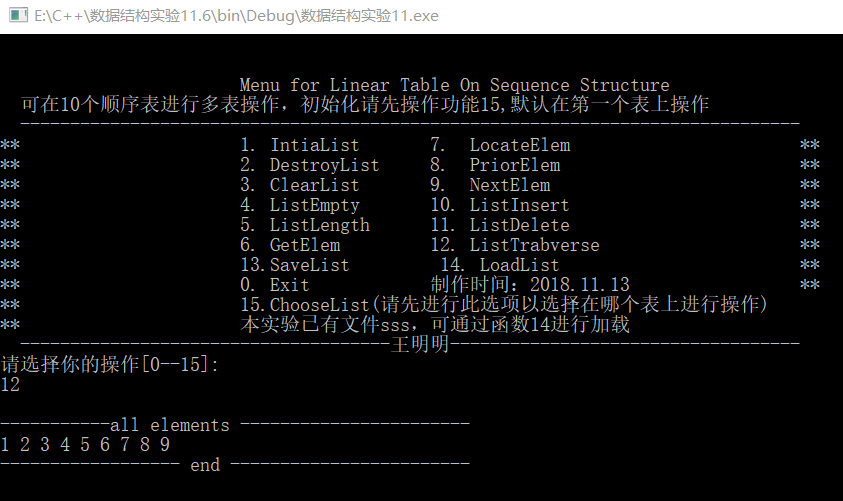
(c)线性表不存在运行截图

图2-23 ListDelete测试截图

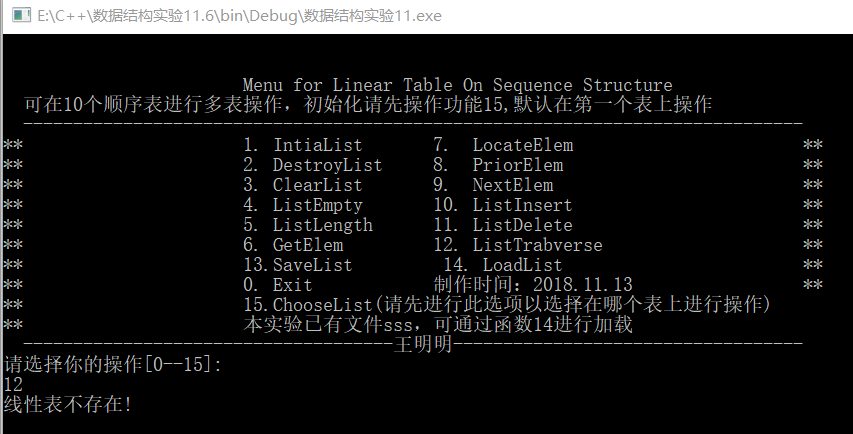
(9) ListTrabverse测试

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。

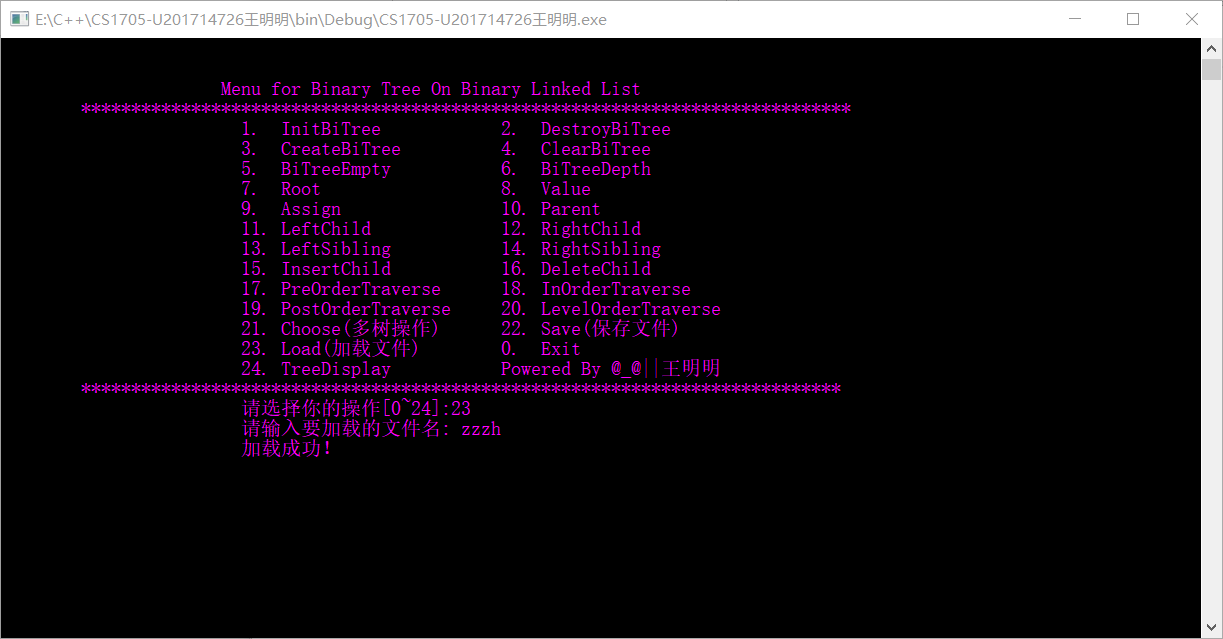
表3-1 测试用例T的节点值与key

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

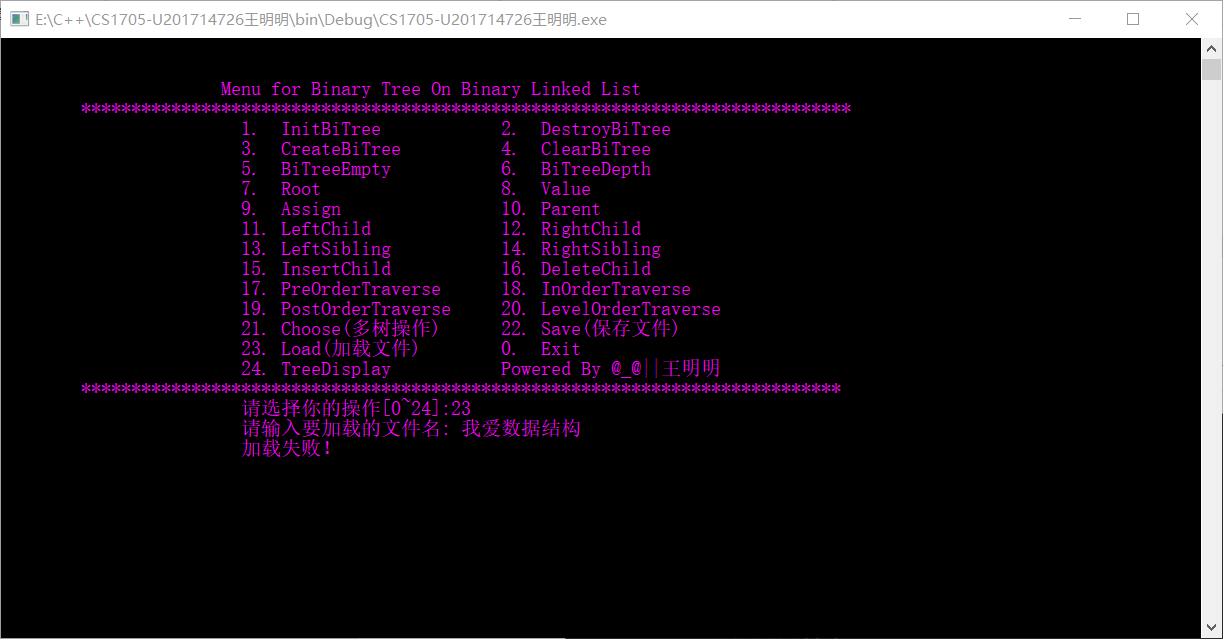
(1)Load函数测试

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



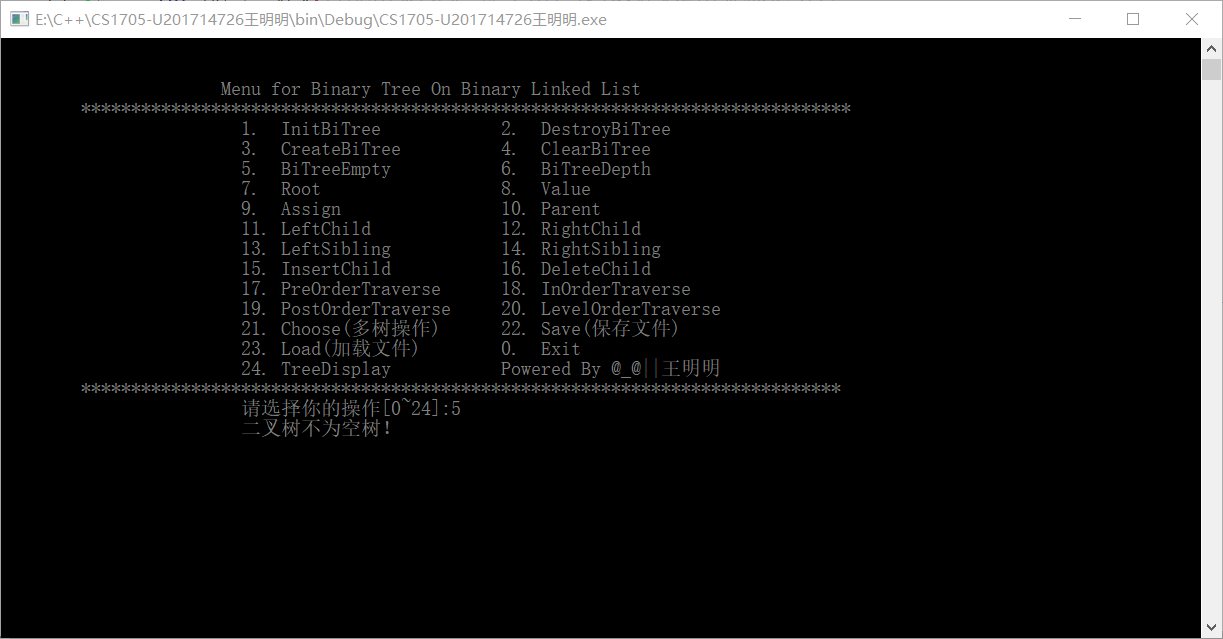
(b)文件不存在时Load测试截图

图3-7 Load测试运行截图

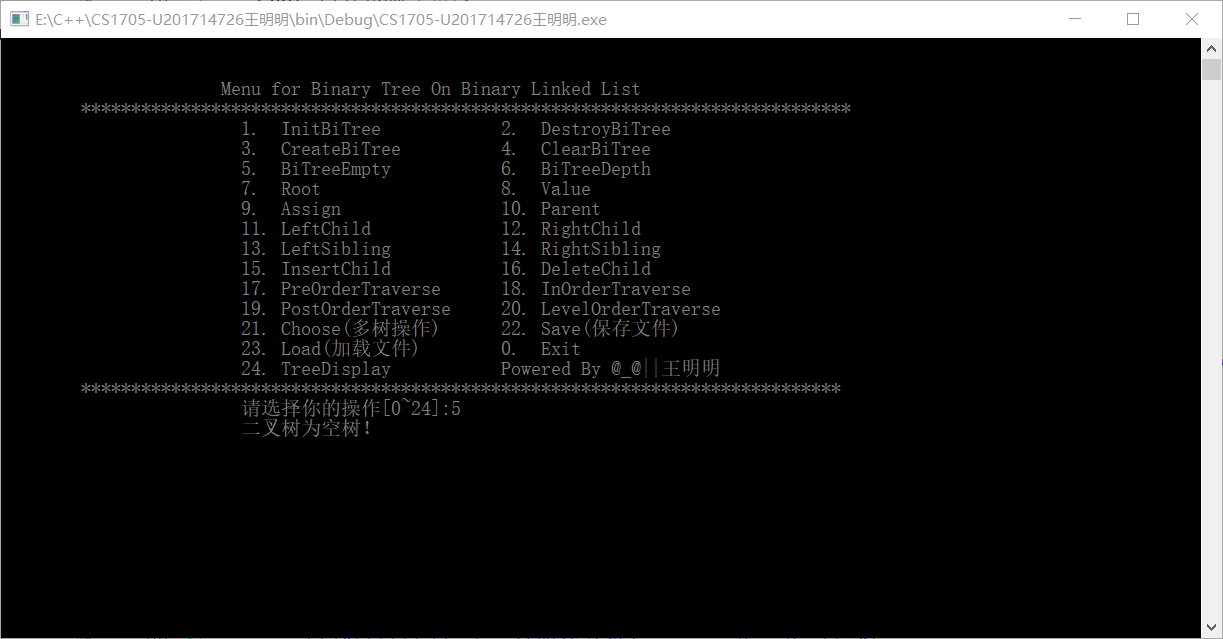
(2) BiTreeEmpty测试

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



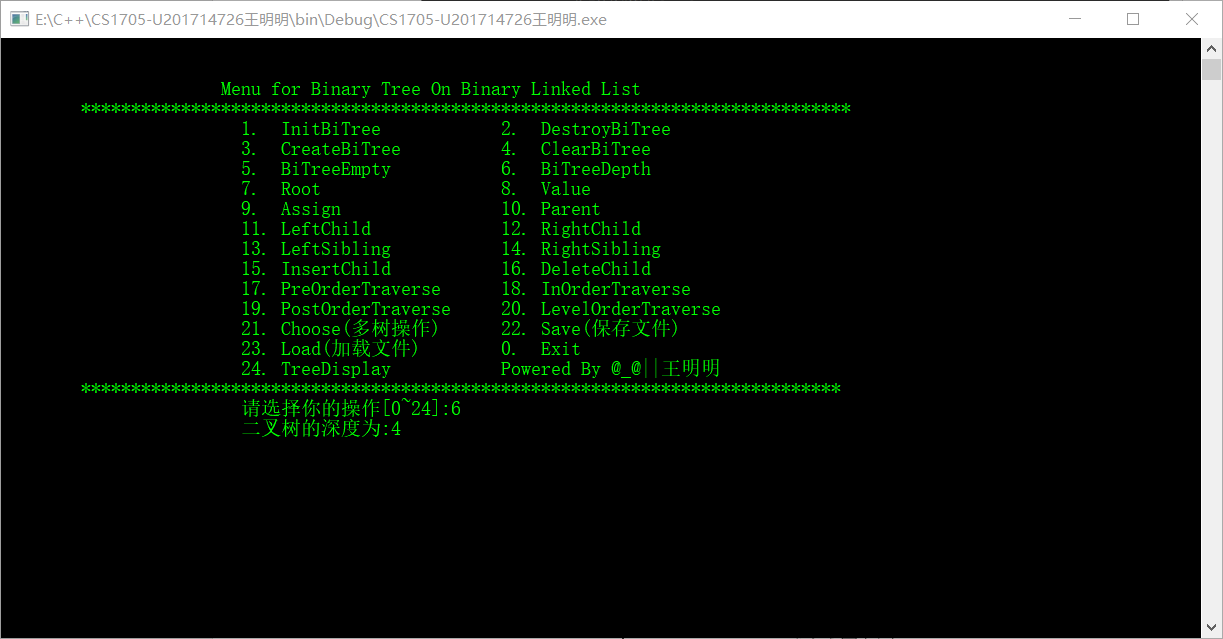
(b)二叉树为空时截图

图3-8 BiTreeEmpty测试截图

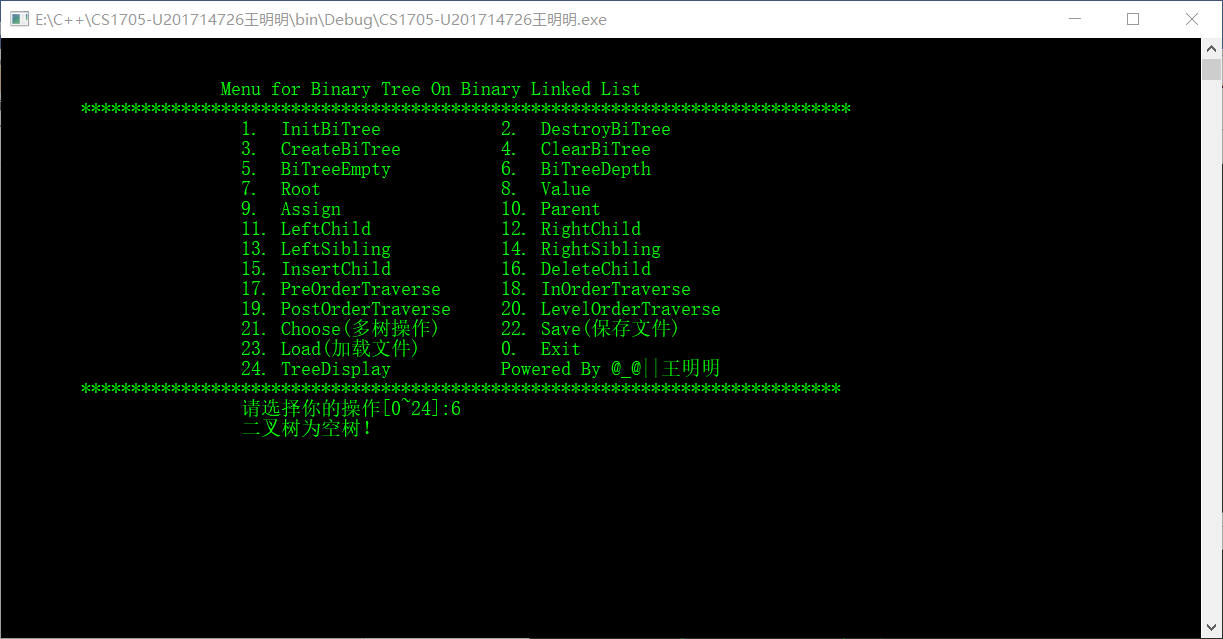
(3) BiTreeDepth测试

表3-4 BiTreeDepth测试用例表

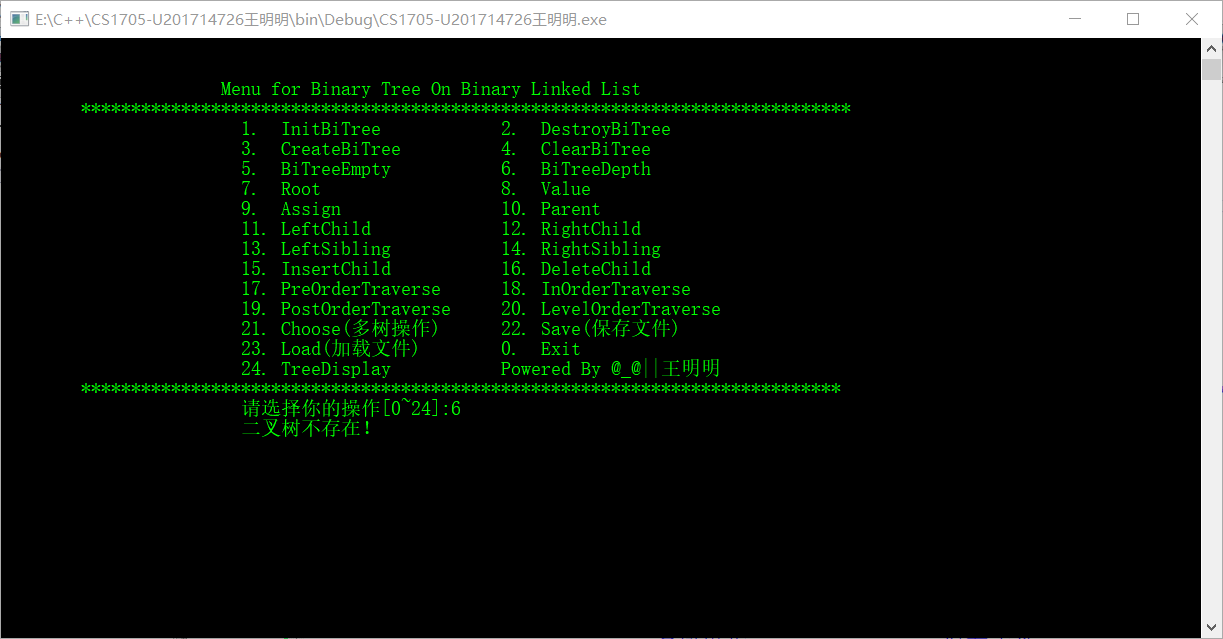
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 界面输入 | 理论结果 | 测试结果 |
| zzzh | 6 | 二叉树深度为：4 | 二叉树深度为4 |
| 空树 | 6 | 二叉树为空树！ | 二叉树为空树！ |
| 二叉树不存在 | 6 | 二叉树不存在！ | 二叉树不存在！ |



(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

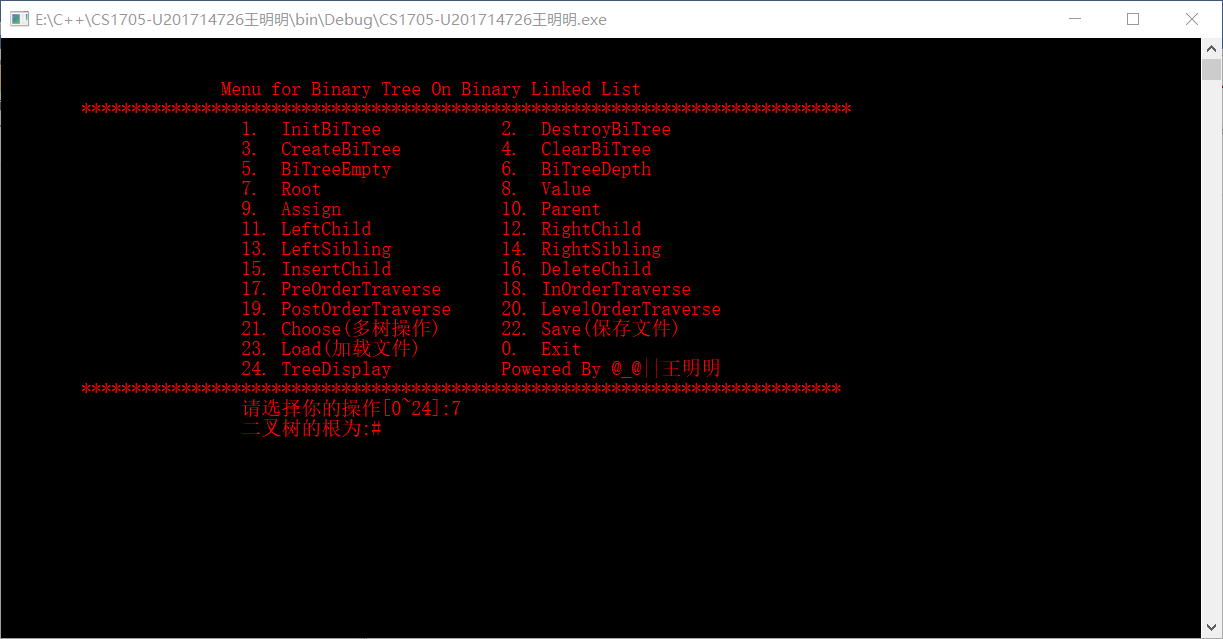
(4) Root测试

表3-5 Root测试用例表

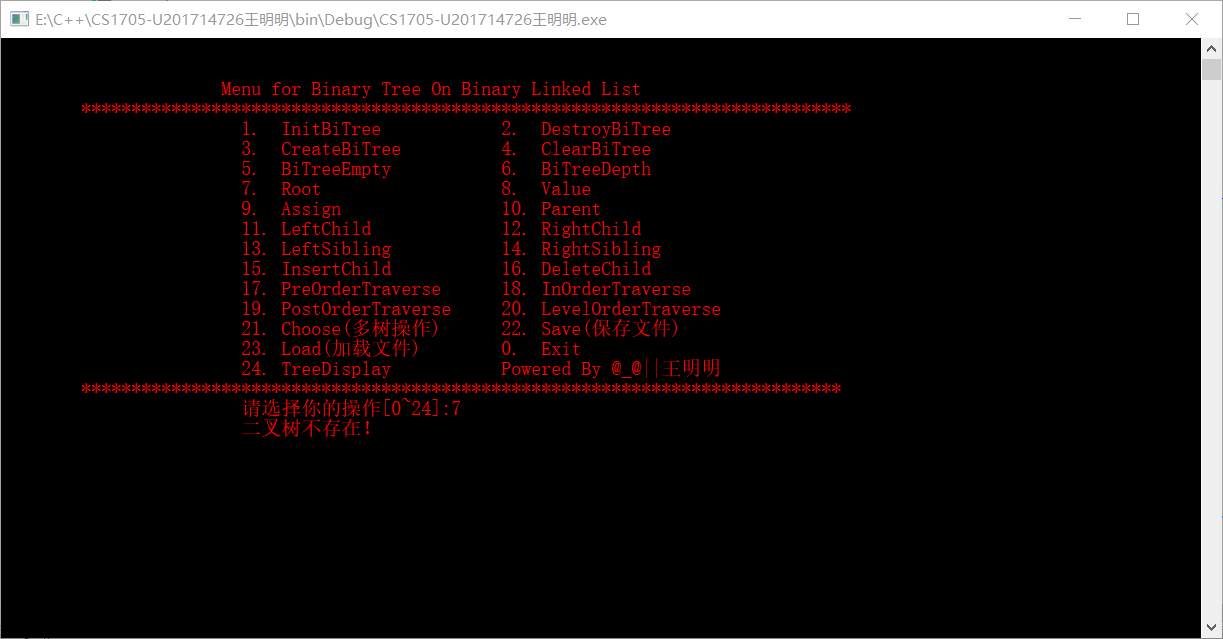
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 界面输入 | 理论结果 | 测试结果 |
| zzzh | 7 | 二叉树的根为：a | 二叉树的根为：a |
| 空树 | 7 | 二叉树的根为：# | 二叉树的根为：# |
| 二叉树不存在 | 7 | 二叉树不存在！ | 二叉树不存在！ |



(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

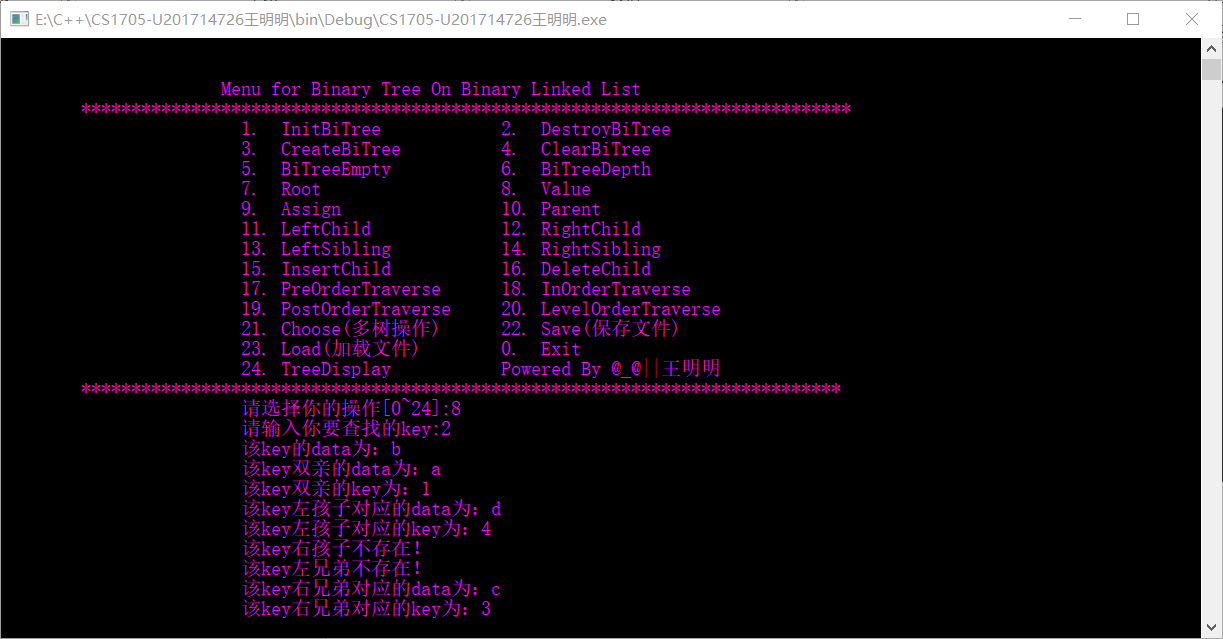
(5) Value测试

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



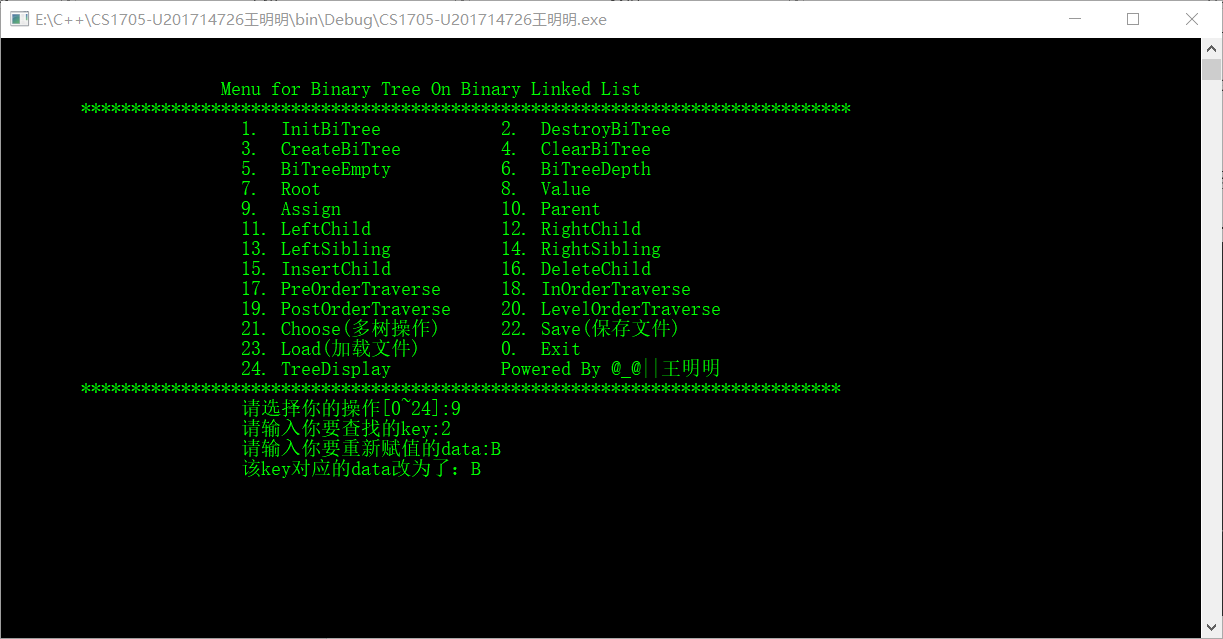
(b)对测试用例zzzh取key为2时截图

图3-11 Value测试截图

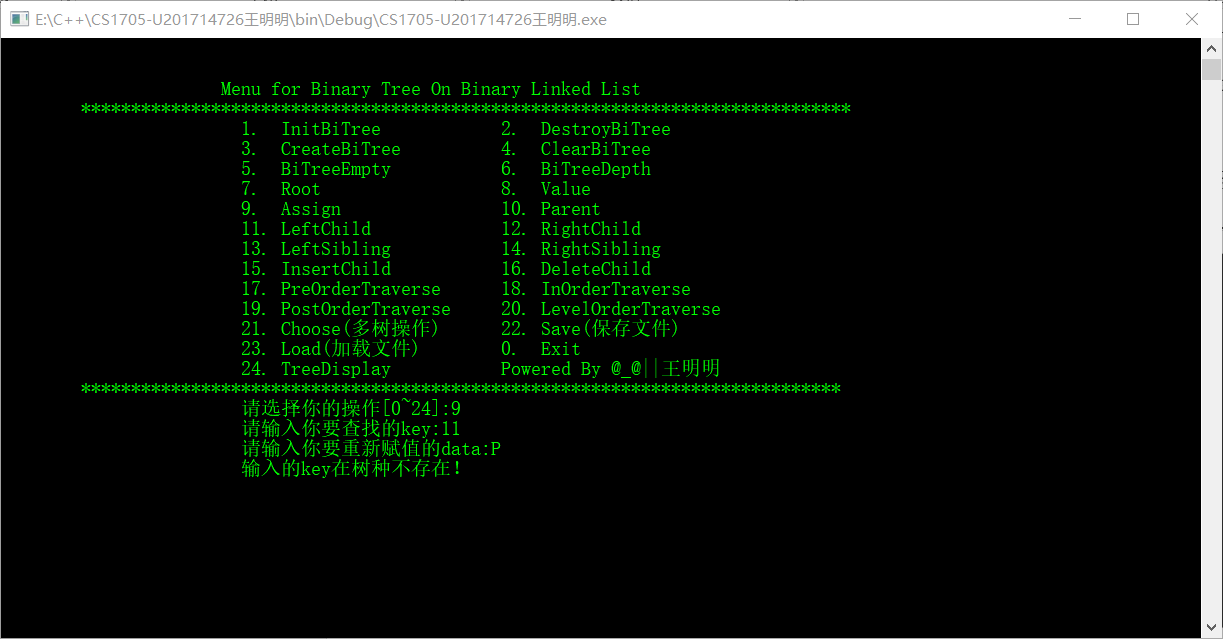
(6) Assign测试

表3-7 Assign测试用例表

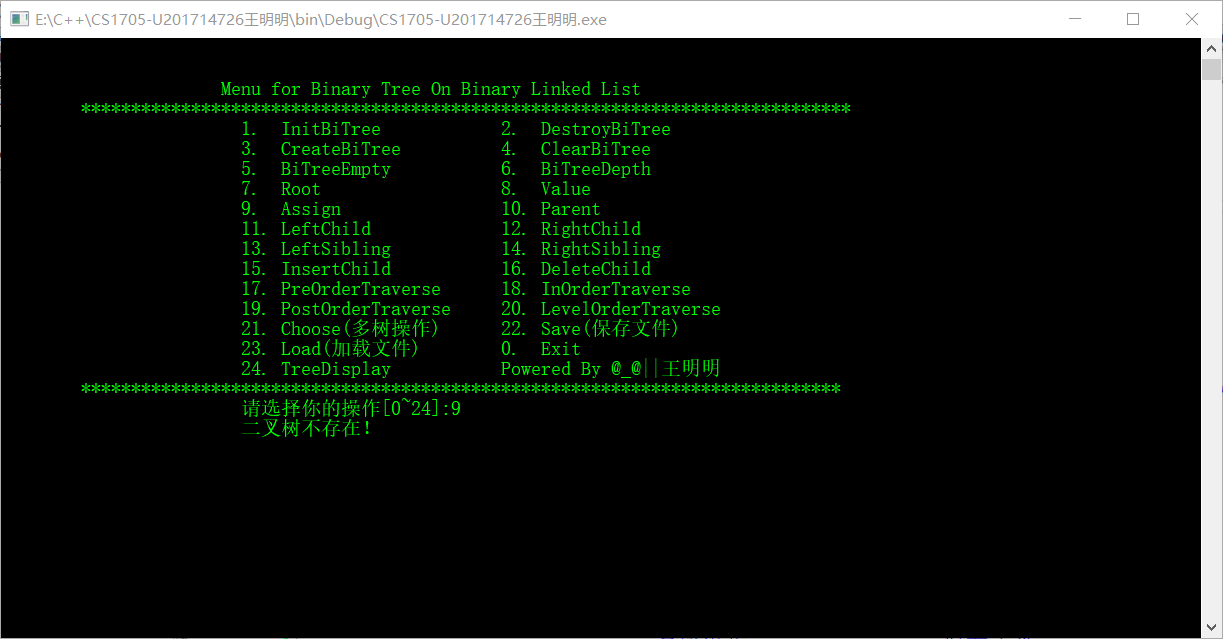
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选9  输入要查找的key:2  输入要重新赋值的data:B | 该key对应的data改为了：B | 该key对应的data改为了：B |
| 空树 | 主界面选9  输入要查找的key:11  输入要重新赋值的data:P | 输入的key在树中不存在！ | 输入的key在树中不存在！ |
| 二叉树不存在 | 主界面选9 | 二叉树不存在！ | 二叉树不存在！ |



(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



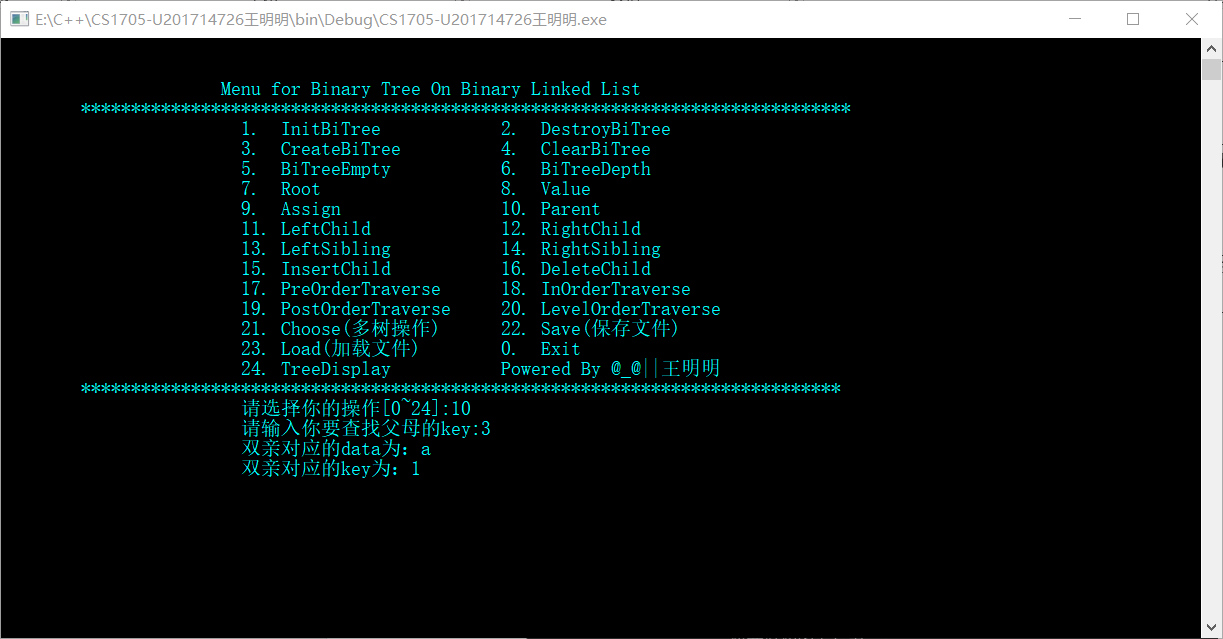
(c)二叉树不存在时截图

图3-12 Assign测试截图

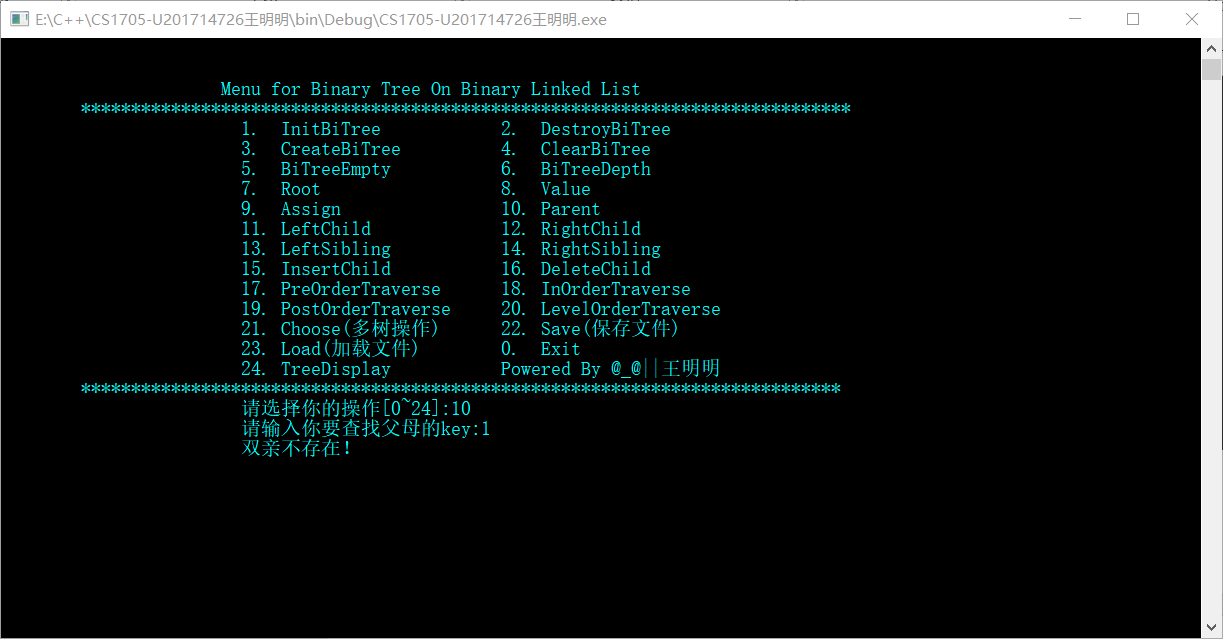
(7) Parent测试

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



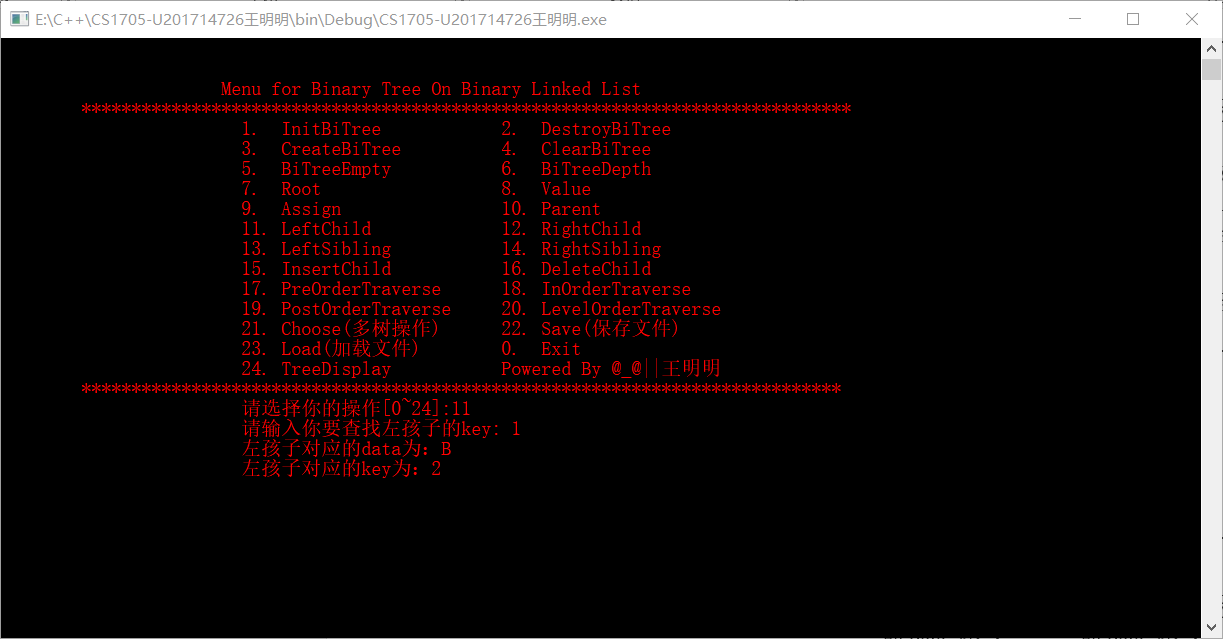
(b)zzzh用例下根节点情况

图3-13 Parent测试截图

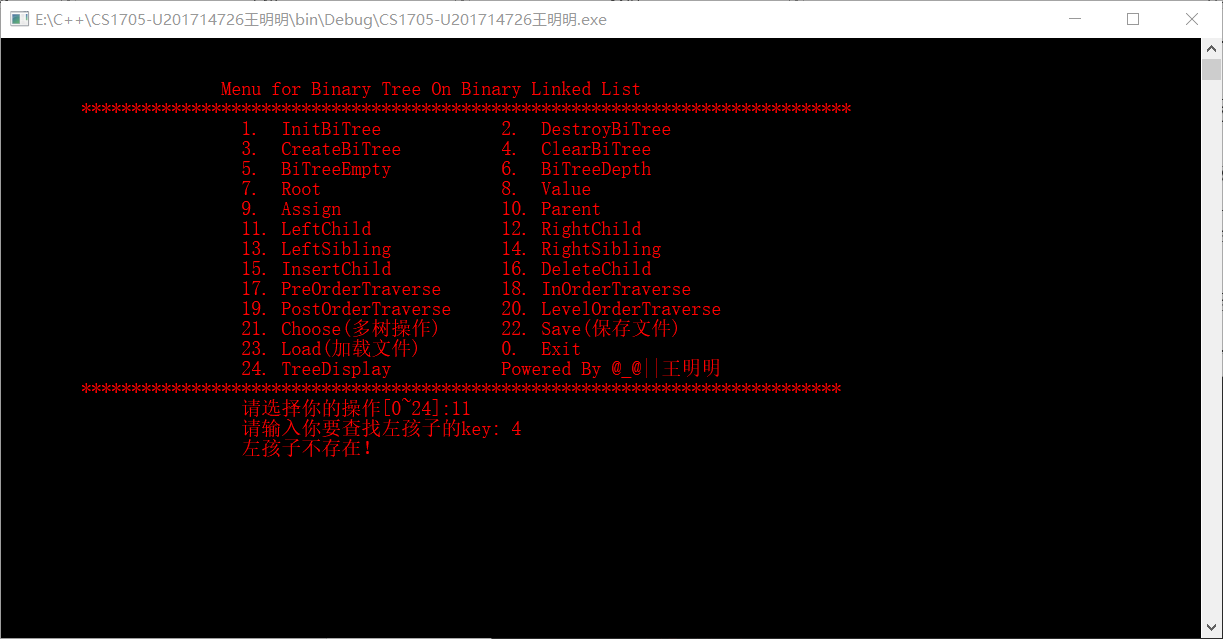
(8) LeftChild测试

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



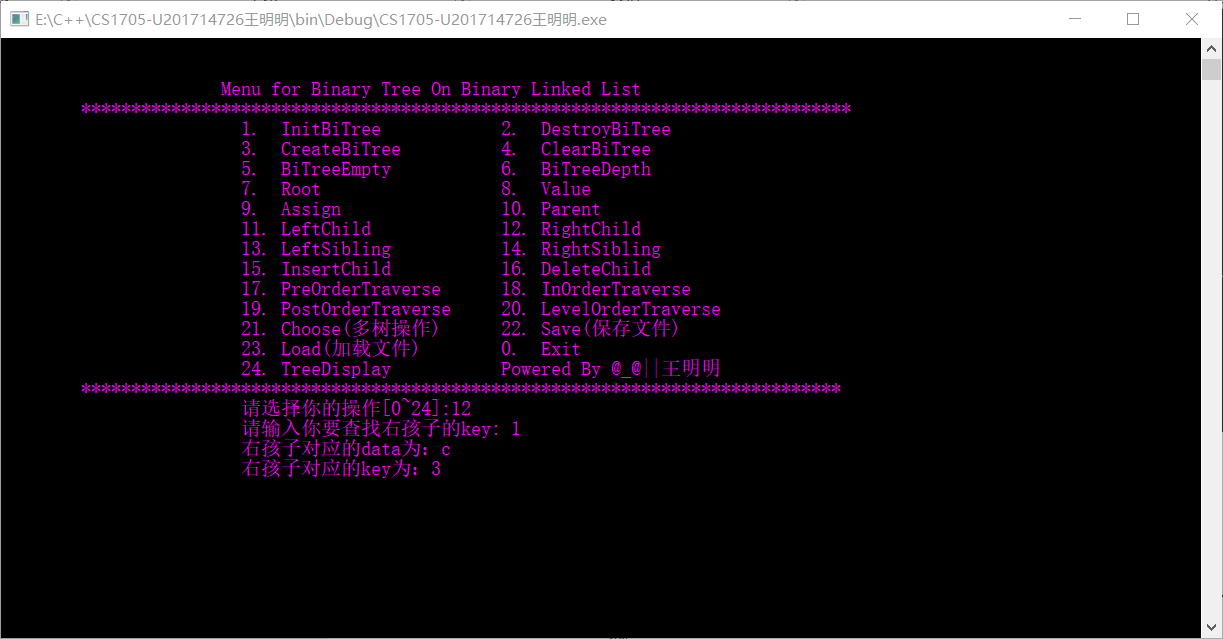
(b)zzzh用例下key为4时截图

图3-14 LeftChild测试截图

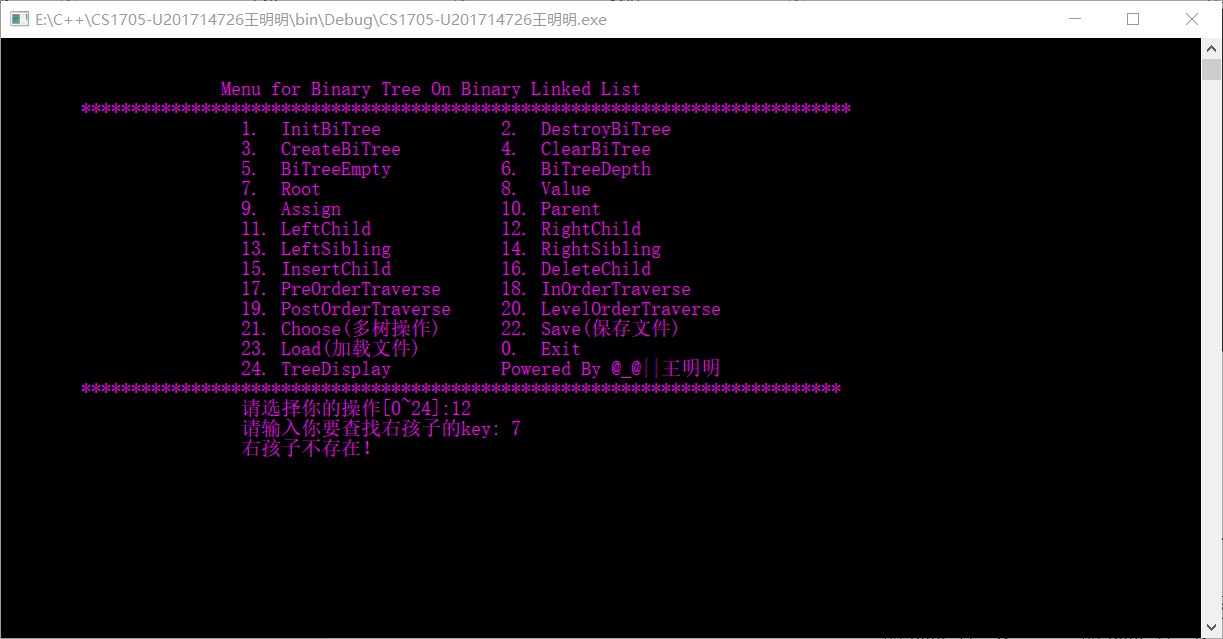
(9) RightChild测试

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

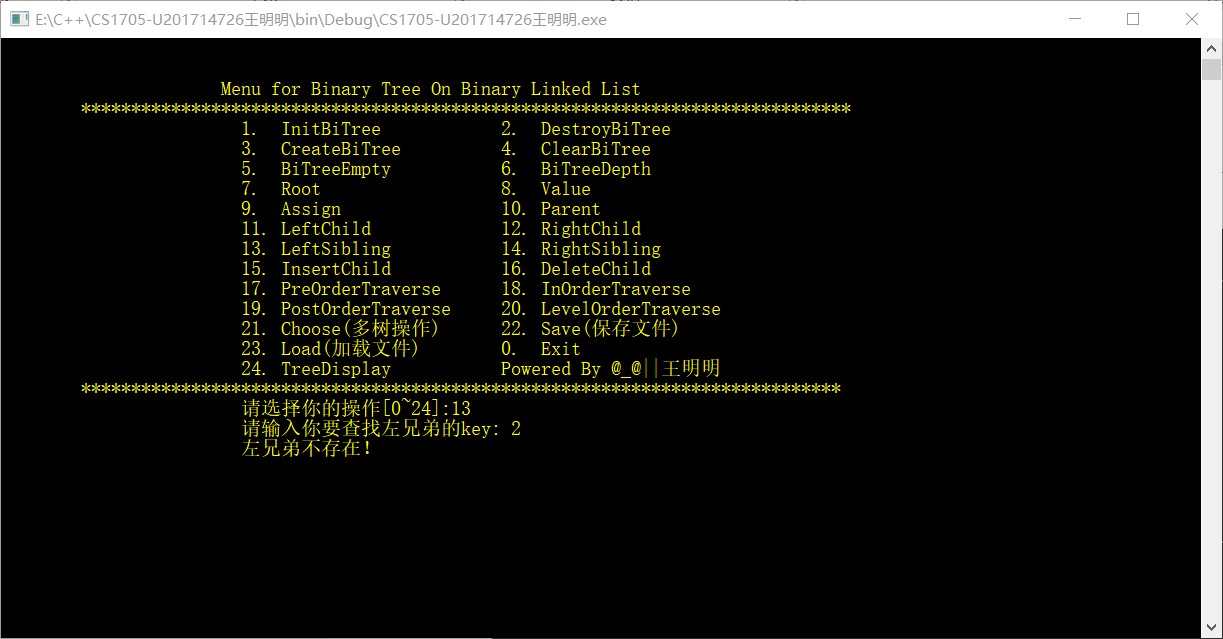
(10) LeftSibling测试

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



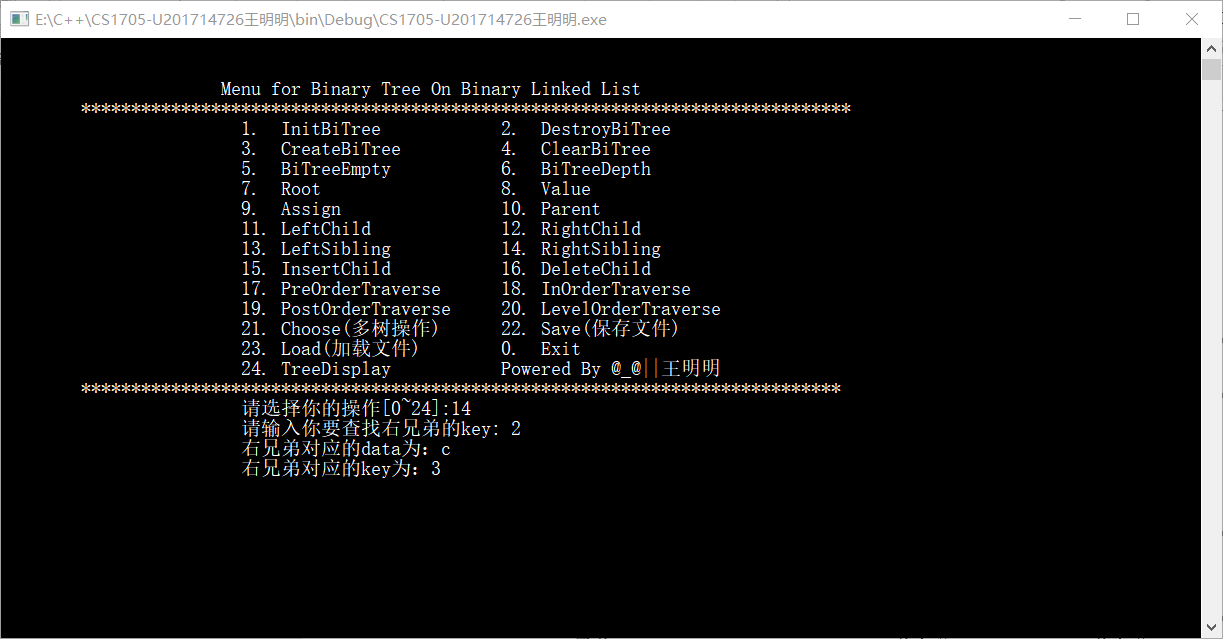
(b)zzzh用例下key为2时截图

图3-16 LeftSibling测试截图

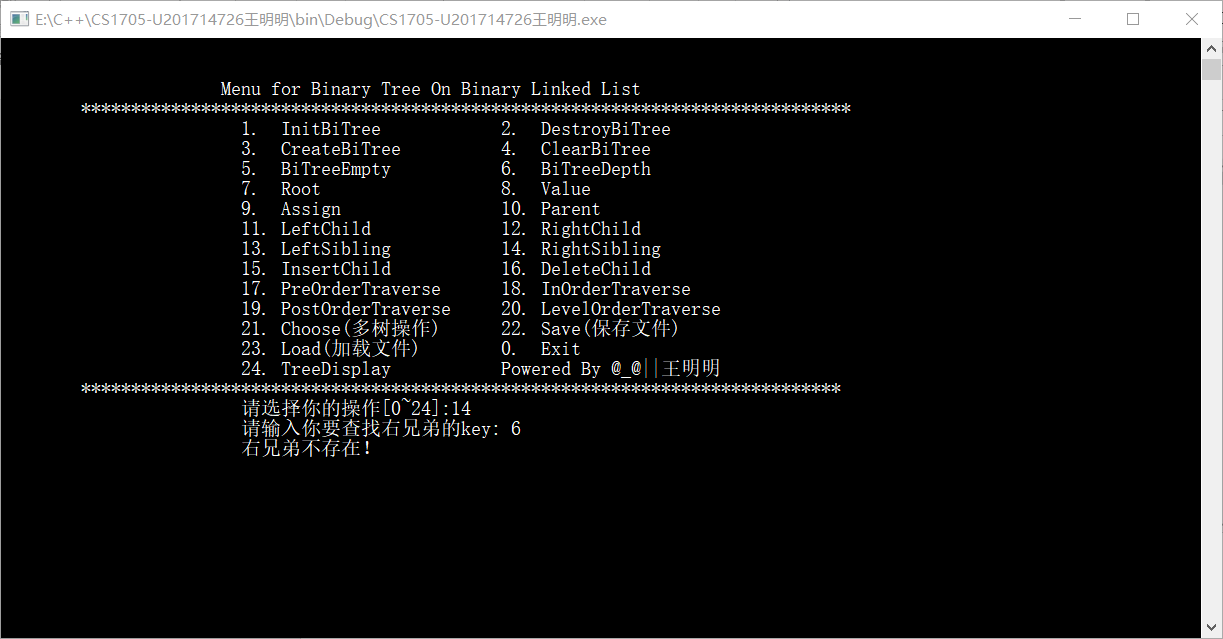
(11) RightSibling测试

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！ |

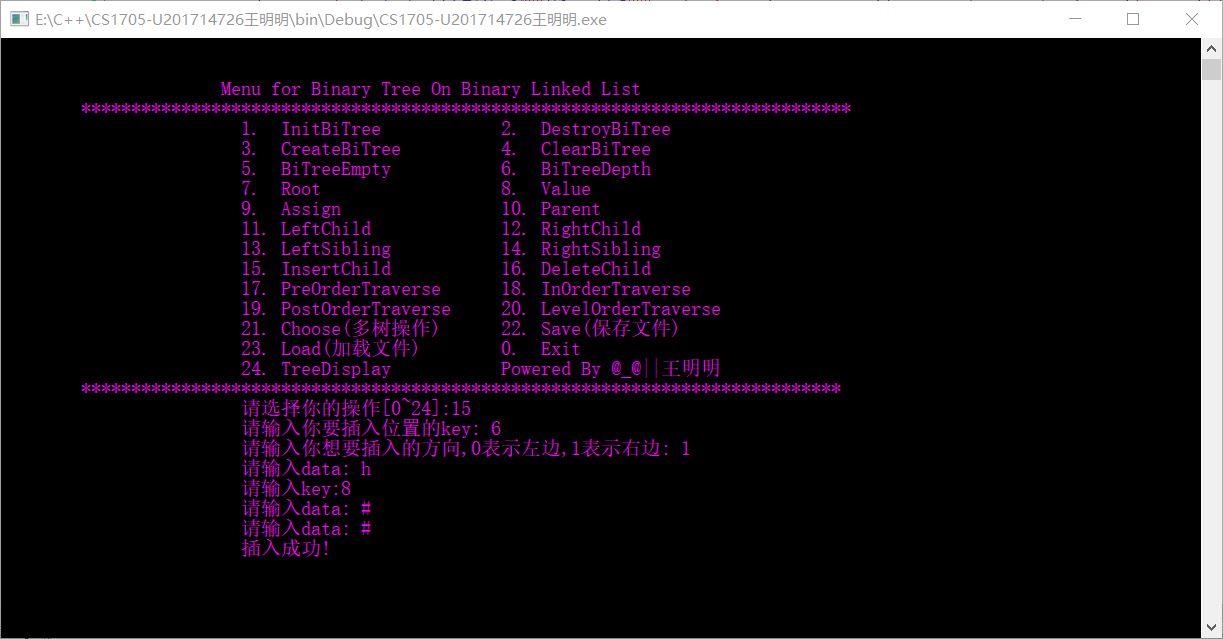


图3-18 InsertChild测试截图

(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！ |

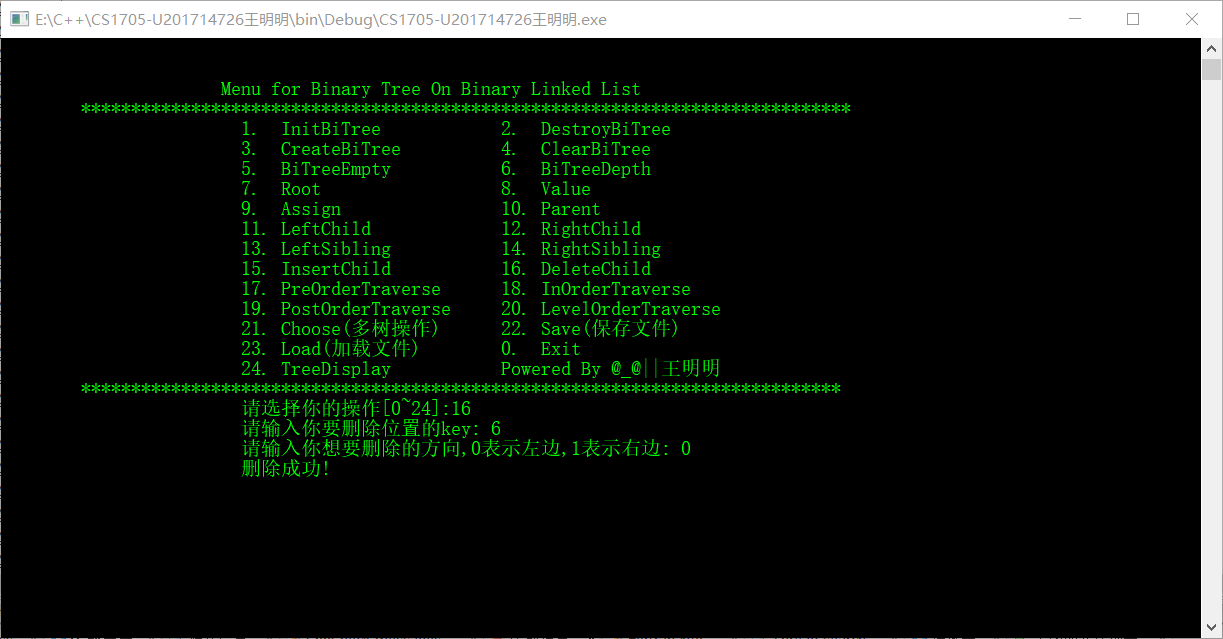


图3-19 DeleteChild测试截图

(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！ |

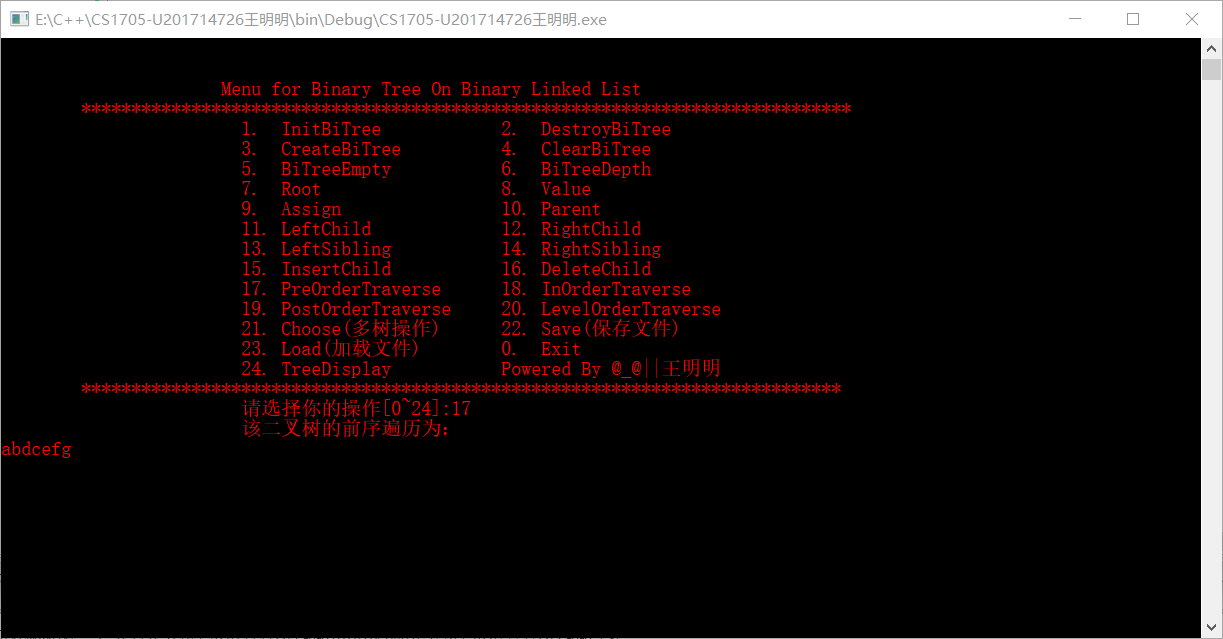


图3-20 PreOrderTraverse测试截图

(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！ |

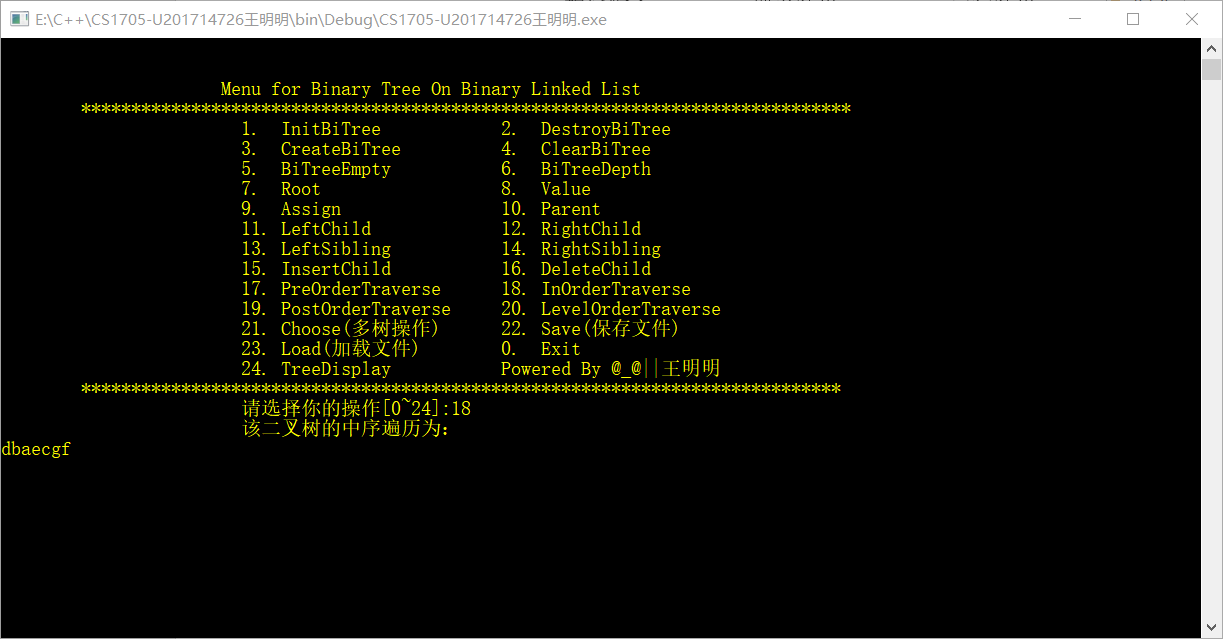


图3-21 InOrderTraverse测试截图

(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！ |

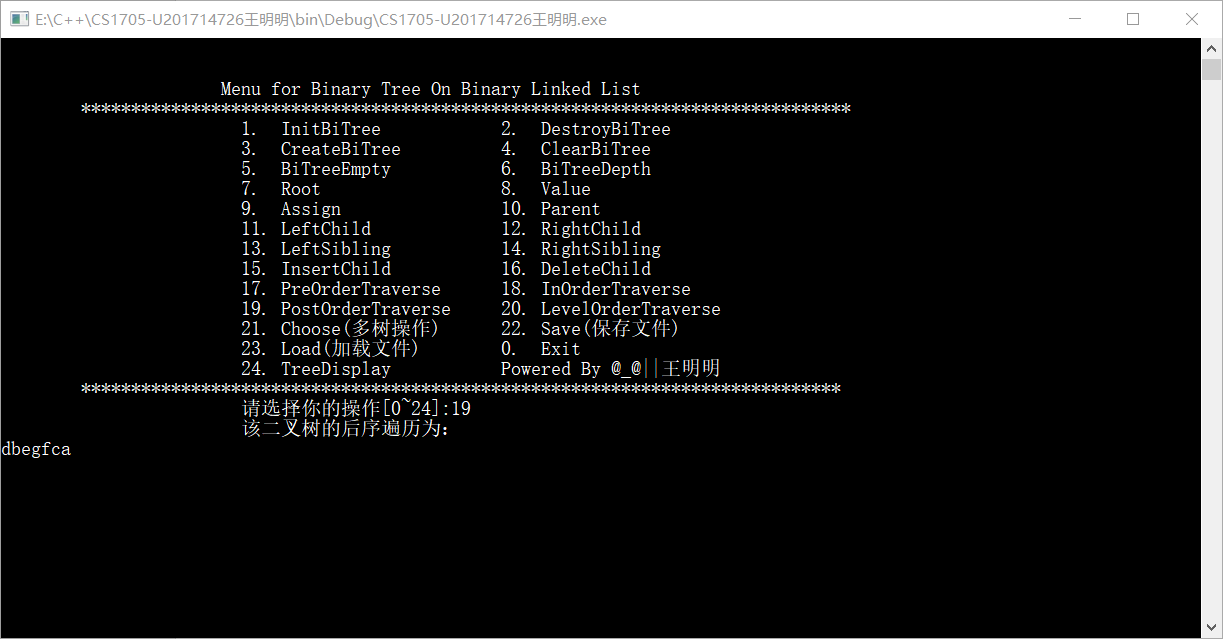


图3-22 PostOrderTraverse测试截图

(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

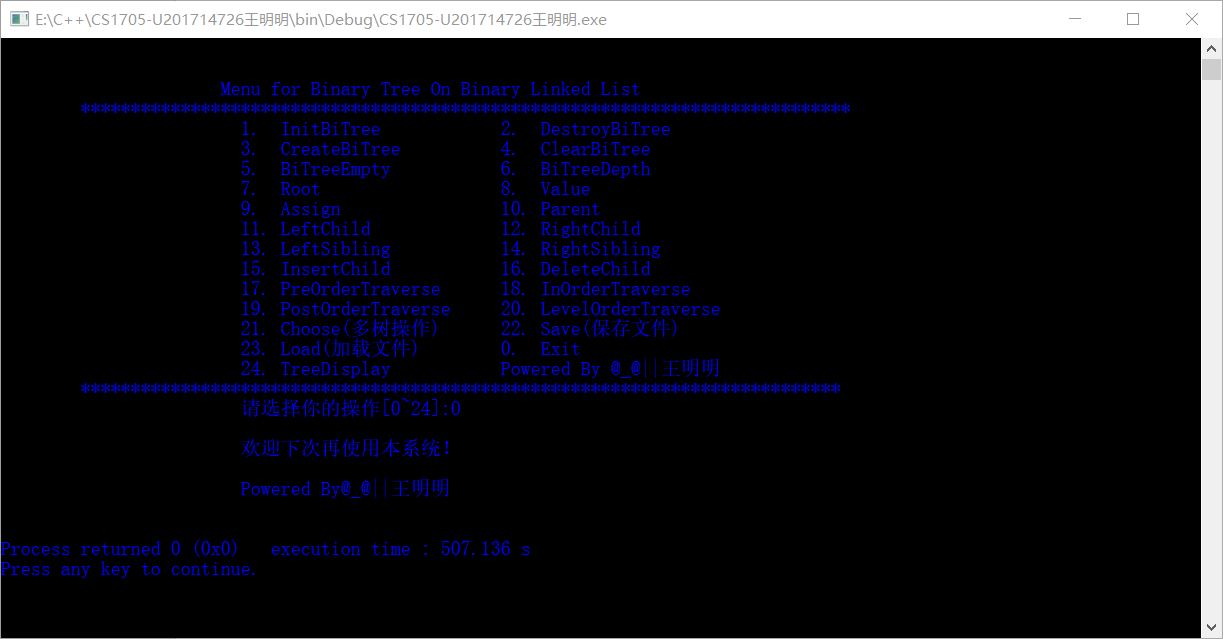


图3-24 Exit运行截图

## 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。

# 4 基于邻接表的无向图实现

## 4.1 问题描述

物理结构为邻接表，创建一个简易菜单的功能演示界面。依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等13种基本运算，并给出了相应的操作提示。也可选择以文件的形式进行存储和加载，整个系统在主程序中完成函数调用。

创建图、销毁图、查找顶点、获得顶点值和顶点赋值等13种基本运算的具体运算功能定义如下：

⑴创建图：函数名称是CreateCraph(&G,V,VR)；初始条件是V是图的顶点集，VR是图的关系集；操作结果是按V和VR的定义构造图G。

⑵销毁图：树函数名称是DestroyBiTree(T)；初始条件图G已存在；操作结果是销毁图G。

⑶查找顶点：函数名称是LocateVex(G,u)；初始条件是图G存在，u和G中的顶点具有相同特征；操作结果是若u在图G中存在，返回顶点u的位置信息，否则返回其它信息。

⑷获得顶点值：函数名称是GetVex (G,v)；初始条件是图G存在，v是G中的某个顶点；操作结果是返回v的值。

⑸顶点赋值：函数名称是PutVex (G,v,value)；初始条件是图G存在，v是G中的某个顶点；操作结果是对v赋值value。

⑹获得第一邻接点：函数名称是FirstAdjVex(&G, v)；初始条件是图G存在，v是G的一个顶点；操作结果是返回v的第一个邻接顶点，如果v没有邻接顶点，返回“空”。

⑺获得下一邻接点：函数名称是NextAdjVex(&G, v, w)；初始条件是图G存在，v是G的一个顶点,w是v的邻接顶点；操作结果是返回v的（相对于w）下一个邻接顶点，如果w是最后一个邻接顶点，返回“空”。

⑻插入顶点：函数名称是InsertVex(&G,v)；初始条件是图G存在，v和G中的顶点具有相同特征；操作结果是在图G中增加新顶点v。

⑼删除顶点：函数名称是DeleteVex(&G,v)；初始条件是图G存在，v是G的一个顶点；操作结果是在图G中删除顶点v和与v相关的弧。

⑽插入弧：函数名称是InsertArc(&G,v,w)；初始条件是图G存在，v、w是G的顶点；操作结果是在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>。

⑾删除弧：函数名称是DeleteArc(&G,v,w)；初始条件是图G存在，v、w是G的顶点；操作结果是在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>。

⑿深度优先搜索遍历：函数名称是DFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

⒀广深度优先搜索遍历：函数名称是BFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

实验目的为：

通过实验达到(1)加深对图的概念、基本运算的理解；(2)熟练掌握图的逻辑结构与物理结构的关系；(3)以邻接表作为物理结构，熟练掌握图基本运算的实现。

## 4.2 系统设计

**4.2.1系统总体设计**

物理结构为邻接表，创建一个简易菜单的功能演示界面。依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等13种基本运算，并给出了相应的操作提示。也可选择以文件的形式进行存储和加载，整个系统在主程序中完成函数调用，以及程序的退出。

**4.2.2有关常量和类型定义**

数据元素类型的定义：

typedef int status;

typedef char VertexType[MAX\_NAME];

typedef int InfoType;

typedef int QElemType;

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 MAX\_NUM 10

#define MAX\_VERTEX\_NUM 20

**4.2.3算法设计**

(1)

函数名称：CreateGraph(G)

操作结果：构造图G

算法思路：采用邻接表作为物理结构，先输入顶点数和边数，再依次输入各顶点并赋值，最后将按边尾和边首表示的边输入并创建完无向图。

(2)

函数名称：DestroyGraph(G)

初始条件：图G已存在

操作结果：销毁图G

算法思路：遍历图，依次释放存储数据元素的空间，最后将图的几个顶点数、弧数和种类均置为零，完成销毁。



图4-1 DestroyGraph(G)流程图

(3)

函数名称：LocateVex(G,u)

初始条件：图G存在，u和G中的顶点具有相同特征

操作结果：若u在图G中存在，返回顶点u的位置信息，否则返回其它信息

算法思路：遍历图，当图中顶点的值与所给信息的值相同时返回该顶点的key，完成定位。

(4)

函数名称：GetVex (G,v)

初始条件：图G存在，v是G中的某个顶点

操作结果：返回v的值

算法思路：返回该key所指向的顶点的值。



图4-2 GetVex (G,v)流程图

(5)

函数名称：PutVex (G,v,value)

初始条件：图G存在，v是G中的某个顶点

操作结果：对v赋值value

算法思路：调用LocateVex(G,u)函数，并将找到的顶点的值修改为所给值。

(6)

函数名称：FirstAdjVex(&G, v)

初始条件：图G存在，v是G的一个顶点

操作结果：返回v的第一个邻接顶点，如果v没有邻接顶点，返回“空”

算法思路：调用LocateVex(G,u)函数，并返回找到的顶点的第一个邻接顶点。



图4-3 FirstAdjVex(&G, v)

(7)

函数名称：NextAdjVex(&G, v, w)

初始条件：图G存在，v是G的一个顶点,w是v的邻接顶点

操作结果：返回v的（相对于w）下一个邻接顶点，如果w是最后一个邻接顶点，返回“空”

算法思路：调用LocateVex(G,u)函数，并返回找到的顶点相对所给的邻接顶点的下一个邻接顶点。

(8)

函数名称：InsertVex(&G,v)

初始条件：图G存在，v和G中的顶点具有相同特征

操作结果：在图G中增加新顶点v

算法思路：输入该顶点的值，插入新结点并对其赋值。



图4-4 InsertVex(&G,v)流程图

(9)

函数名称：DeleteVex(&G,v)

初始条件：图G存在，v是G的一个顶点

操作结果：在图G中删除顶点v和与v相关的弧

算法思路：调用LocateVex(G,u)函数，找到该顶点，删除与该顶点相关的所有边以及该顶点。

(10)

函数名称：InsertArc(&G,v,w)

初始条件：图G存在，v、w是G的顶点

操作结果：在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>

算法思路：调用LocateVex(G,u)函数，找到相应顶点，增添对应的弧<v,w>以及 <w,v>。

(11)

函数名称：DeleteArc(&G,v,w)

初始条件：是图G存在，v、w是G的顶点

操作结果：在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>

算法思路：调用LocateVex(G,u)函数，找到相应顶点，删除对应的弧<v,w>以及 <w,v>。

(12)

函数名称：DFSTraverse(G,visit())

初始条件：图G存在

操作结果：进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次

算法思路：首先，访问出发顶点v并作访问标记；然后，依次从v的未访问过的邻接顶点w出发，进行深度优先遍历。

(13)

函数名称：BFSTraverse(G,visit())

初始条件：图G存在

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

算法思路：首先，访问出发顶点v并作访问标记；然后，依次访问v的所有未访问过的邻接顶点w，再依次广度优先遍历w的所有未访问过的邻接顶点，直到出发顶点v的所有可达顶点均被访问过为止。

## 4.3 系统实现

**4.3.1程序源代码**

见《附录D 基于邻接表图实现的源程序》。

**4.3.2系统测试**

本系统界面简洁舒适，如图4-1所示，挑选LocateVex, GetVex, PutVex, FirstAdjVex, NextAdjVex, InsertVex, DeleteVex, InsertArc, DeleteArc, DFSTraverse, BFSTraverse这些重要功能进行测试。

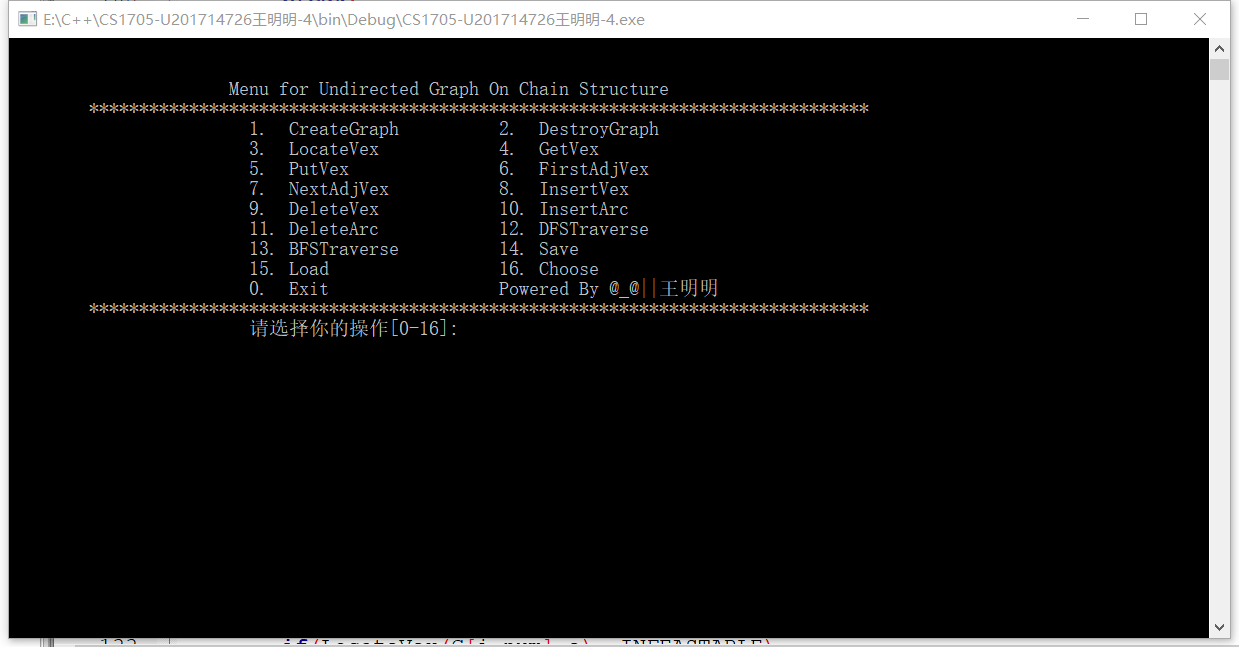


图4-5 程序简易界面截图

测试用例G=(V,E)以及图不存在的情况共两种情况。附有文件zzzh，节点集为V，边集为E。V={v1,v2,v3,v4,v5},E={(v1,v2),(v1,v3),(v2,v3),(v3,v4),(v3,v5),(v4,v5)}。

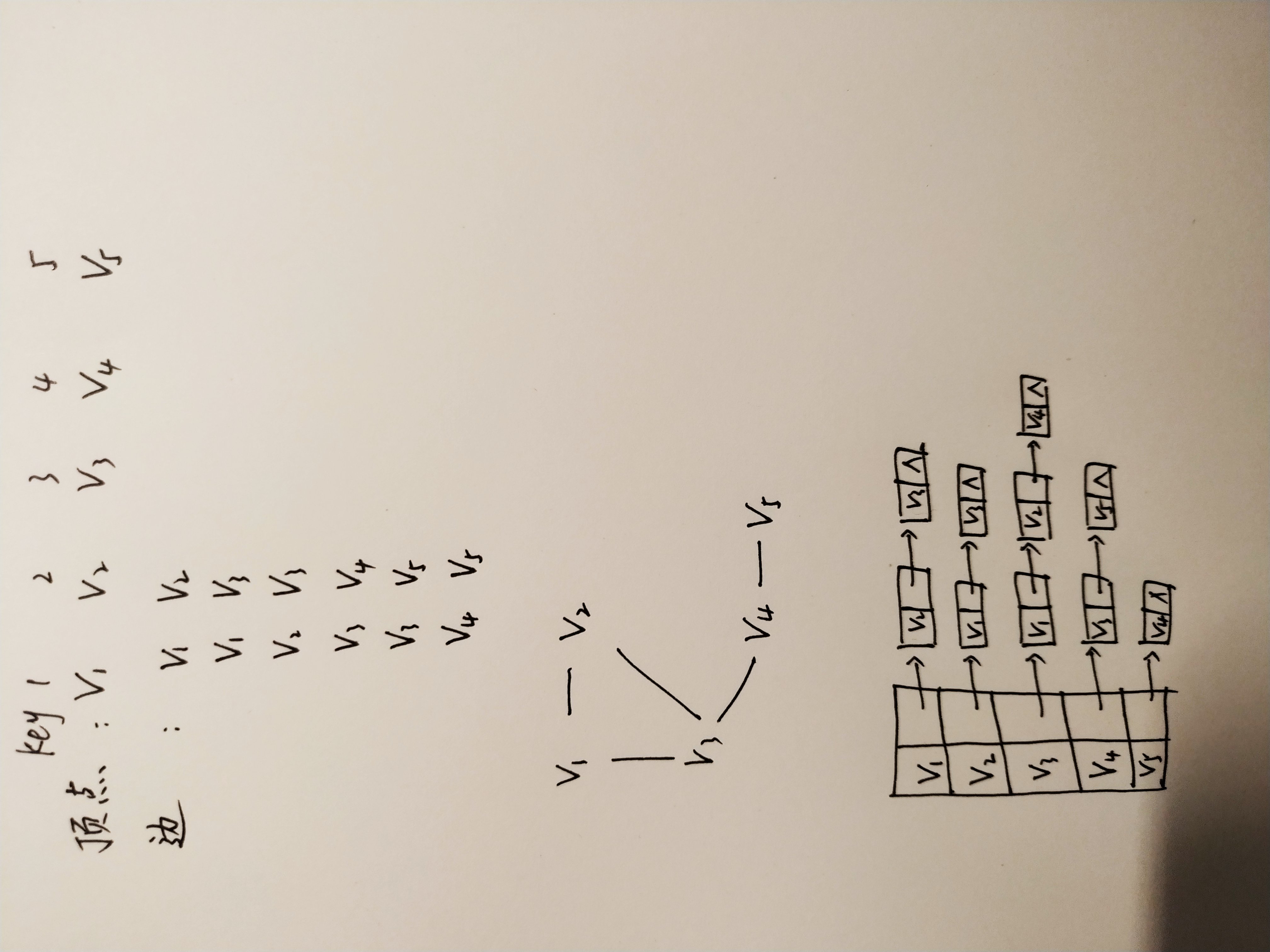


图4-6 测试用例zzzh

(1) LocateVex测试 (如表4-1所示)

表4-1 LocateVex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3  输入要查找的顶点的值：v2 | 该顶点的key为2 | 该顶点的key为2 |
| 若图不存在 | 主界面选3 | 图不存在！” | 图不存在！ |

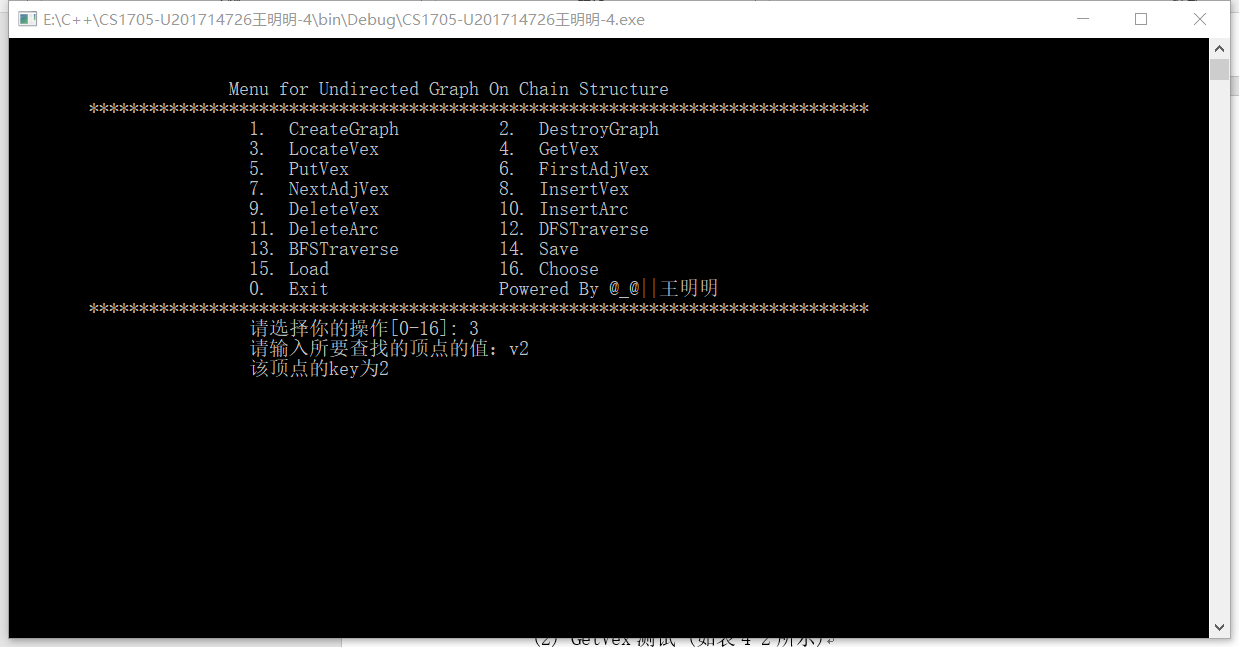


图4-7 LocateVex测试截图

(2) GetVex测试 (如表4-2所示)

表4-2 GetVex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4  输入要获取顶点的key：2 | 该顶点为：v2 | 该顶点为：v2 |
| 若图不存在 | 主界面选4 | 图不存在！ | 图不存在！ |

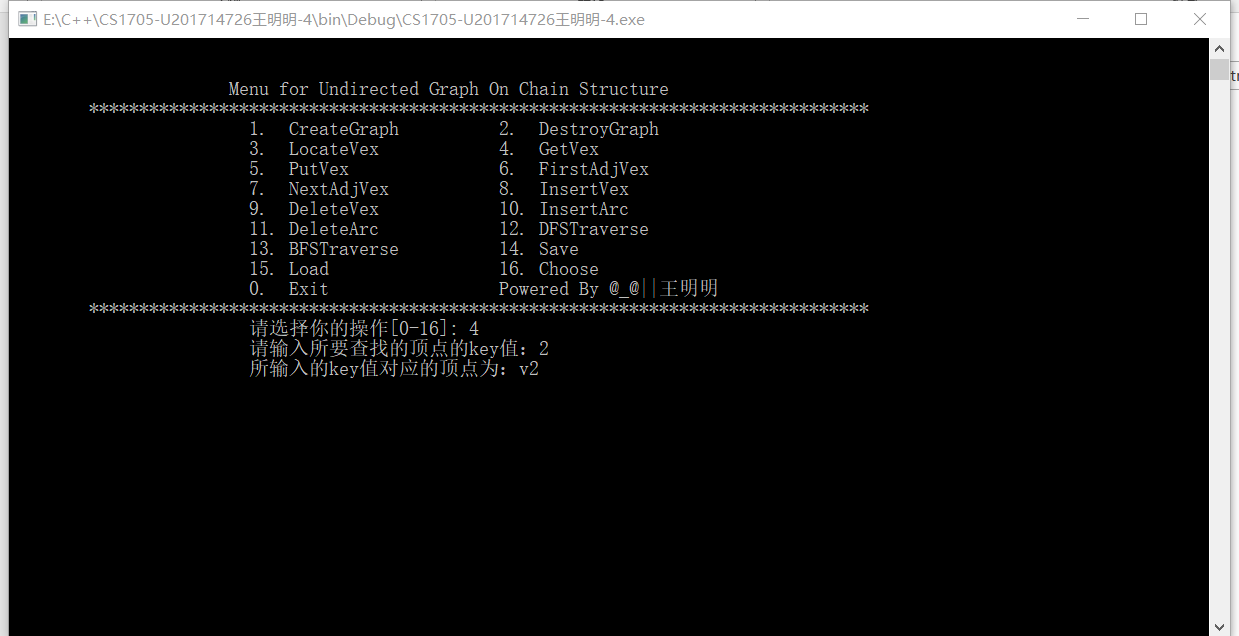
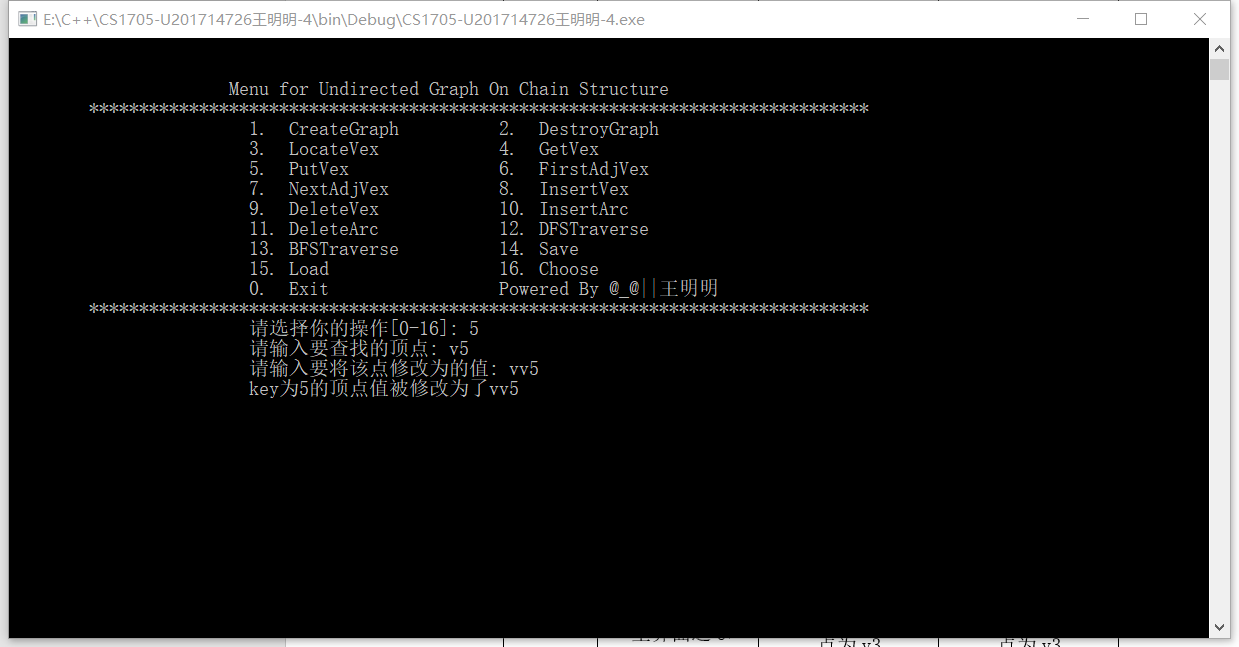


图4-8 GetVex测试截图

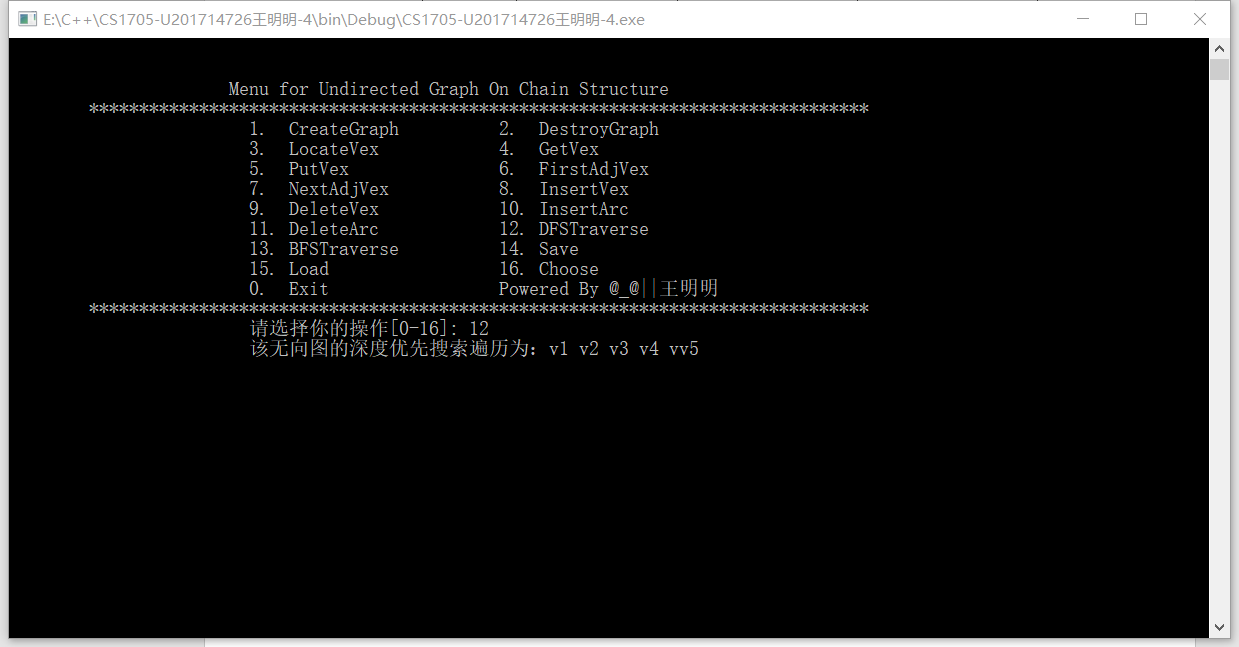
(3) PutVex测试 (如表4-3所示)

表4-3 PutVex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5  请输入要查找的顶点的值：v5 输入要将该点修改的值：vv5 | key为5的顶点值被修改为了vv5 | key为5的顶点值被修改为了vv5 |
| 若图不存在 | 主界面选5 | 图不存在！ | 图不存在！ |



(a)修改成功时截图



(b)进行遍历，验证已被修改

图4-9 PutVex测试截图

(4) FirstAdjVex测试 (如表4-4所示)

表4-4 FirstAdjVex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6  输入要查找的顶点的key：1 | 该顶点首个邻接顶点为v2 该顶点首个邻接顶点的key为2 | 该顶点首个邻接顶点为v2 该顶点首个邻接顶点的key为2 |
| 若图不存在 | 主界面选6 | 图不存在！ | 图不存在！ |

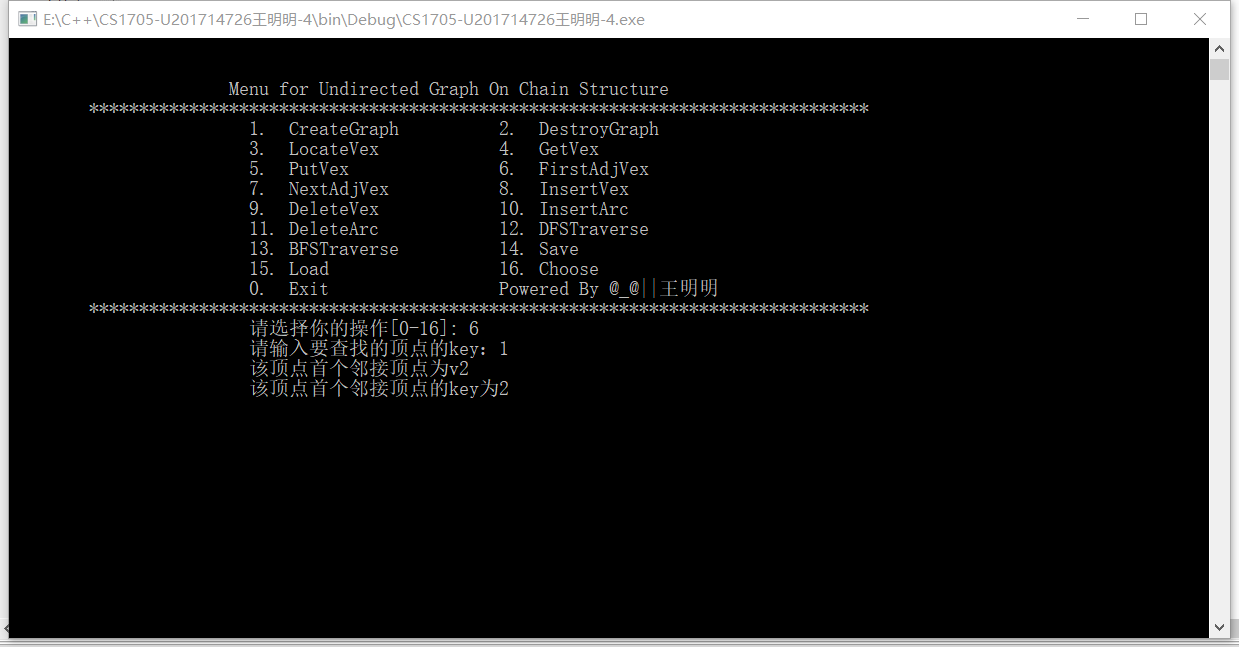


图4-10 FirstAdjVex测试截图

(5) NextAdjVex测试 (如表4-5所示)

表4-5 NextAdjVex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7  输入要查找的顶点v的key：2 输入和其相对的顶点w的key：3 | w是v的最后一个邻接顶点！ | W是v的最后一个邻接顶点！ |
| 若图不存在 | 主界面选7 | 图不存在！ | 图不存在！ |

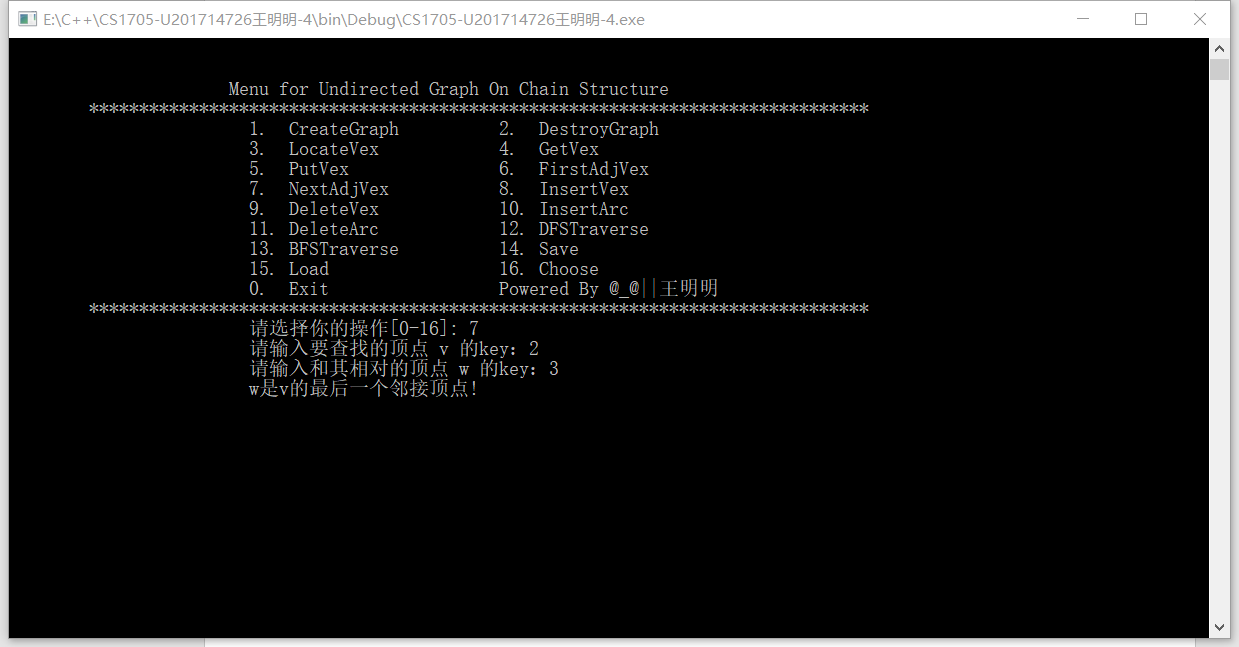
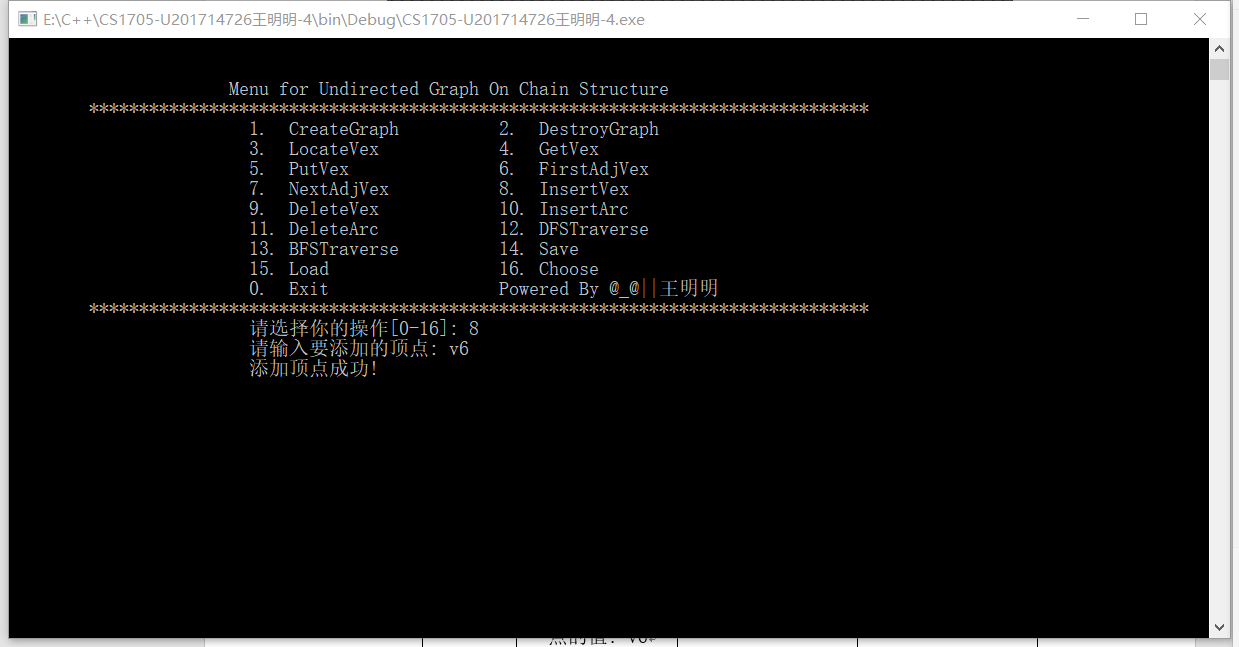


图4-11 NextAdjVex测试截图

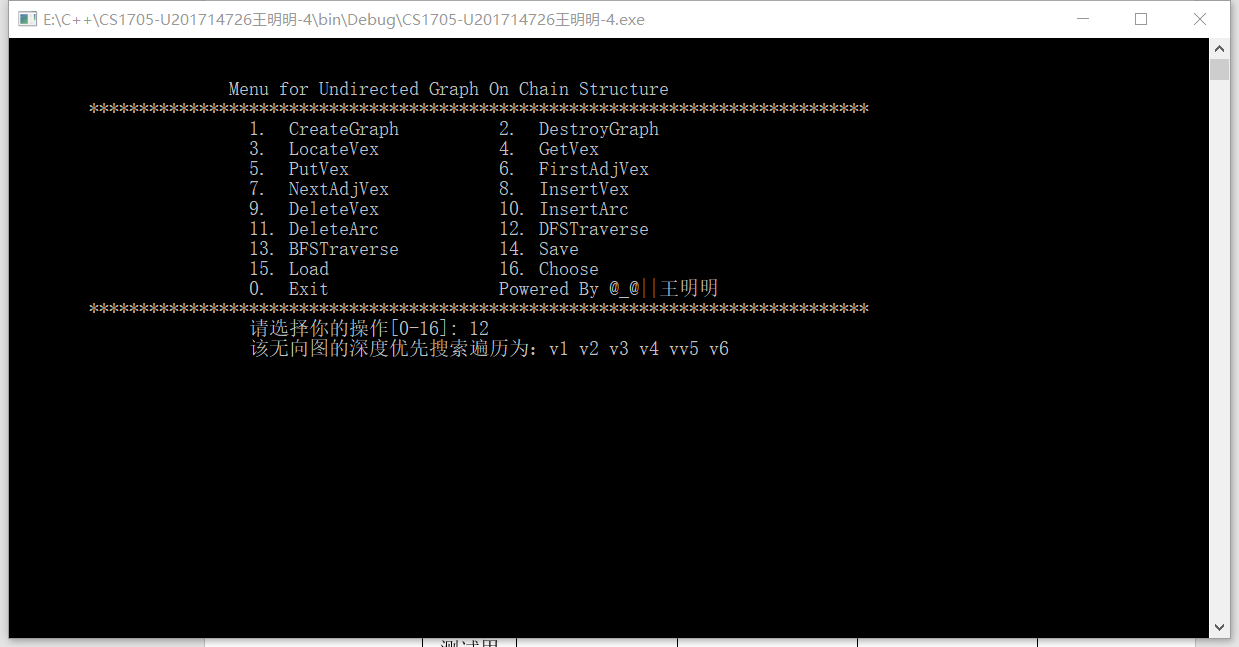
(6) InsertVex测试 (如表4-6所示)

表4-6 InsertVex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8  输入要添加的顶点的值:v6 | 添加顶点成功！ | 添加顶点成功！ |
| 若图不存在 | 主界面选8 | 图不存在！ | 图不存在！ |



(a)添加顶点成功时截图



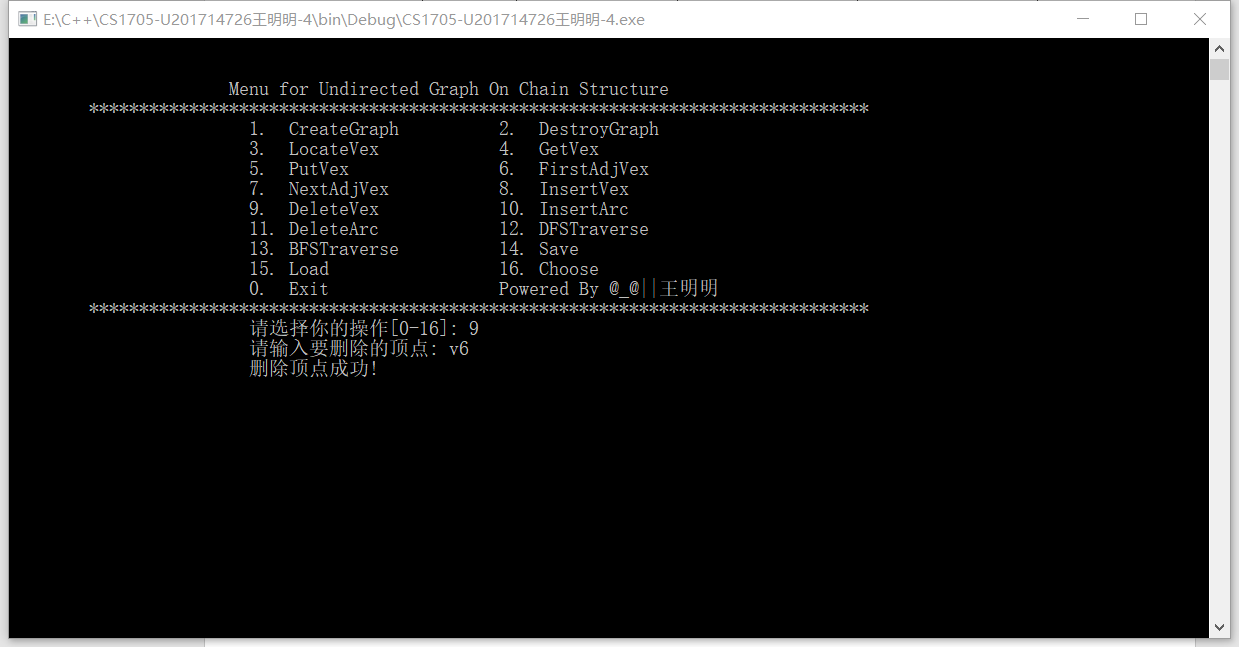
(b)验证顶点添加成功

图4-12 InsertVex测试截图

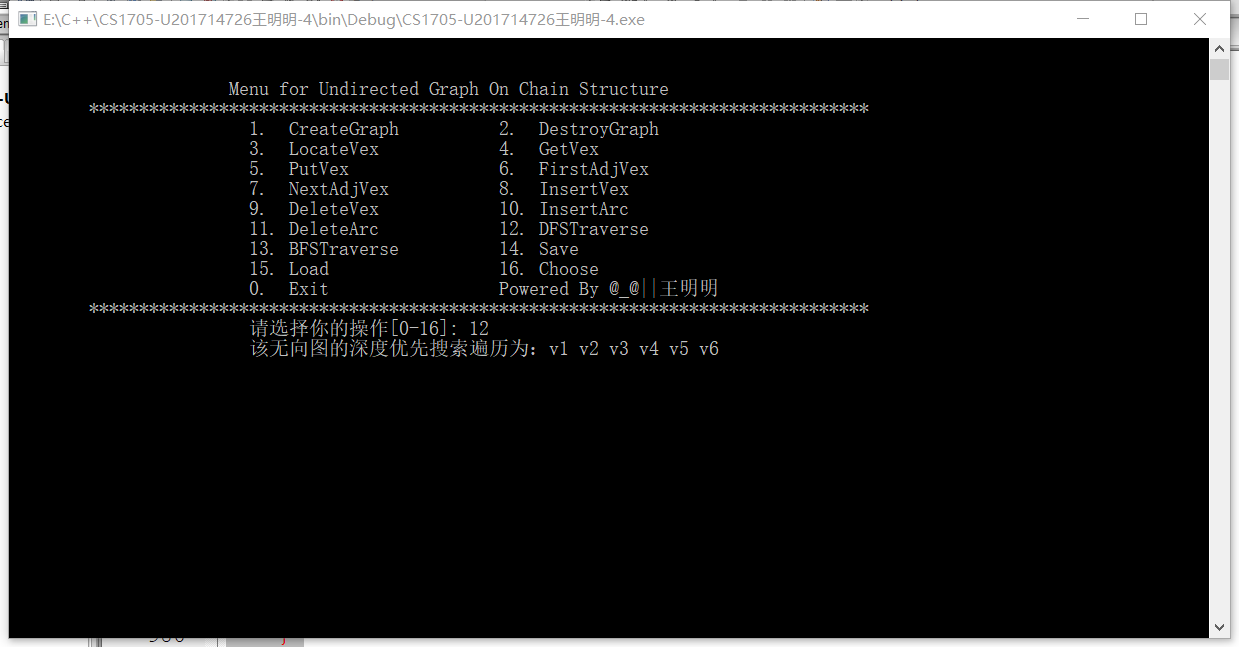
(7) DeleteVex测试 (如表4-7所示)

表4-7 DeleteVex测试用例表

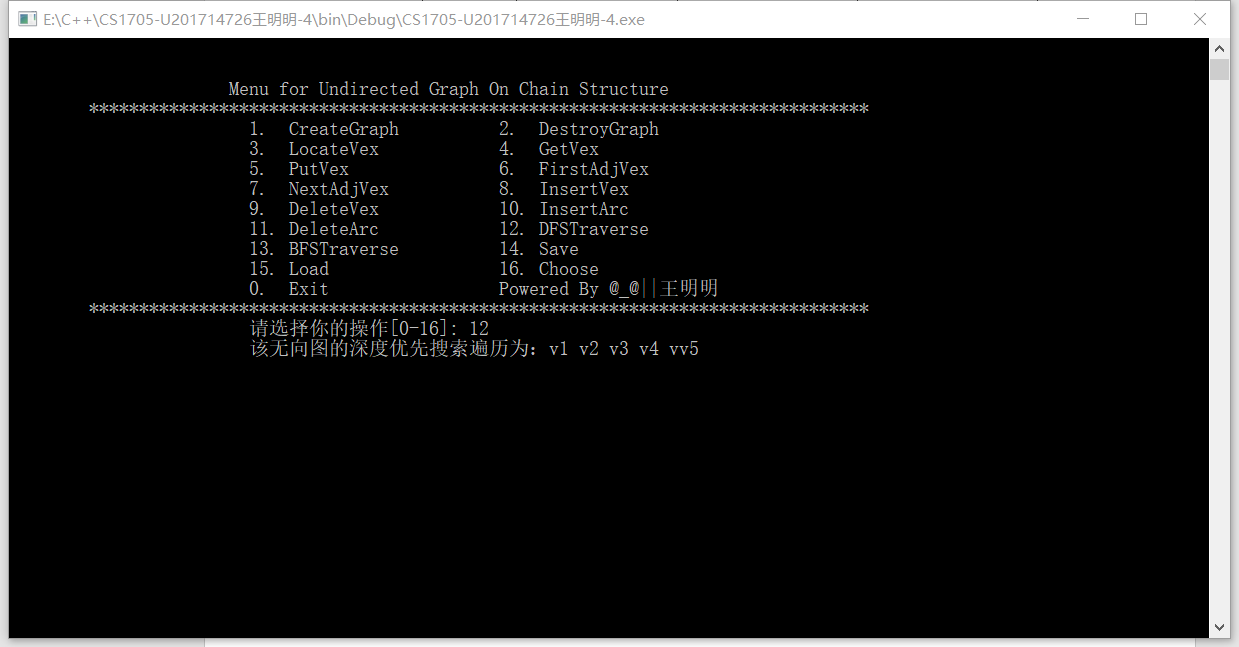
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选9  输入要删除的顶点的值: v6 | 删除顶点成功！ | 删除顶点成功！ |
| 若图不存在 | 主界面选9 | 图不存在！ | 图不存在！ |



(a)删除顶点成功时截图



(b)插入之后截图



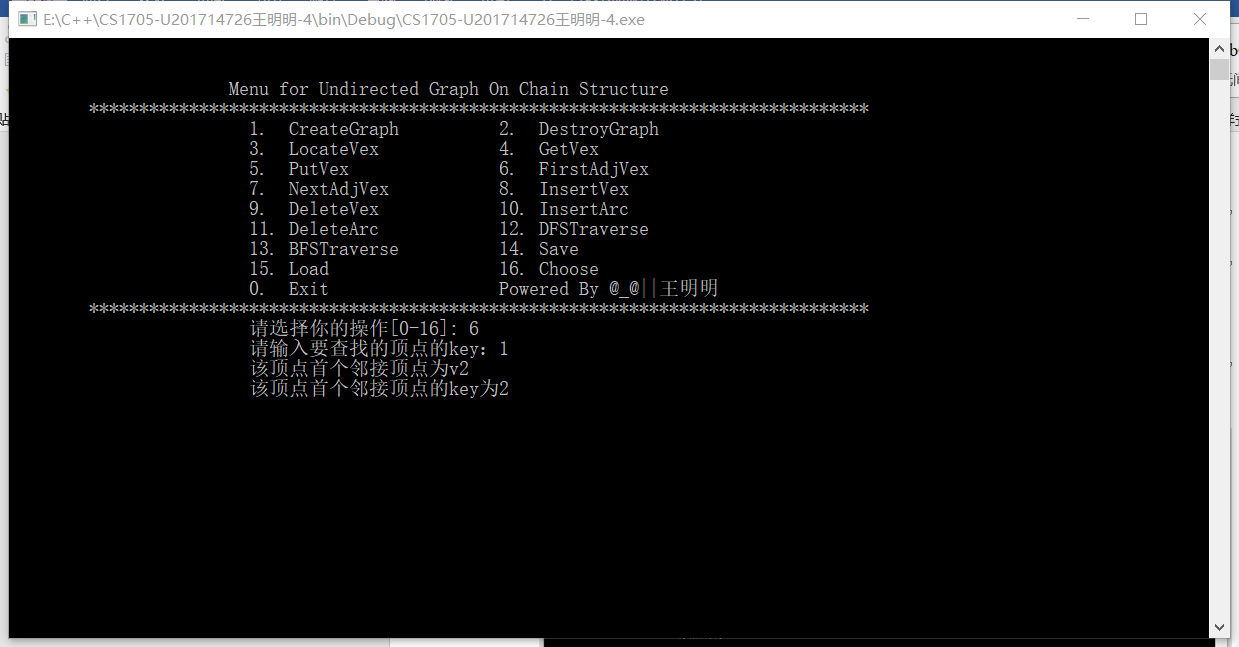
(c)验证删除顶点成功

图4-13 DeleteVex测试截图

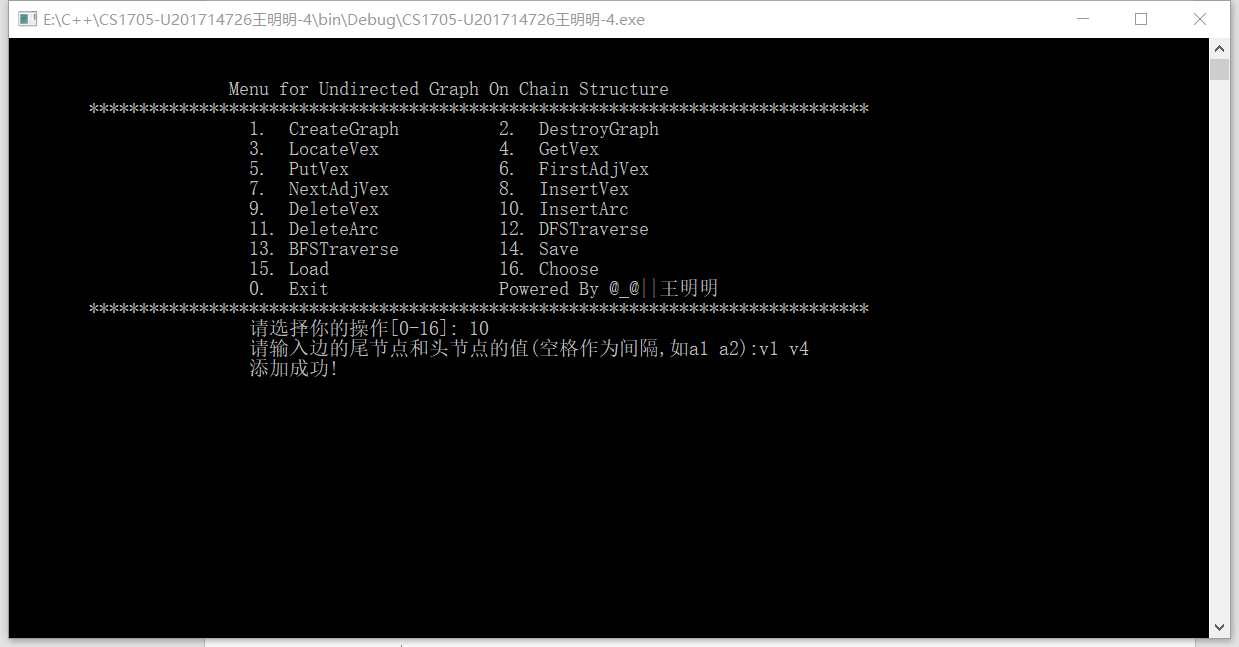
(8) InsertArc测试 (如表4-8所示)

表4-8 InsertArc测试用例表

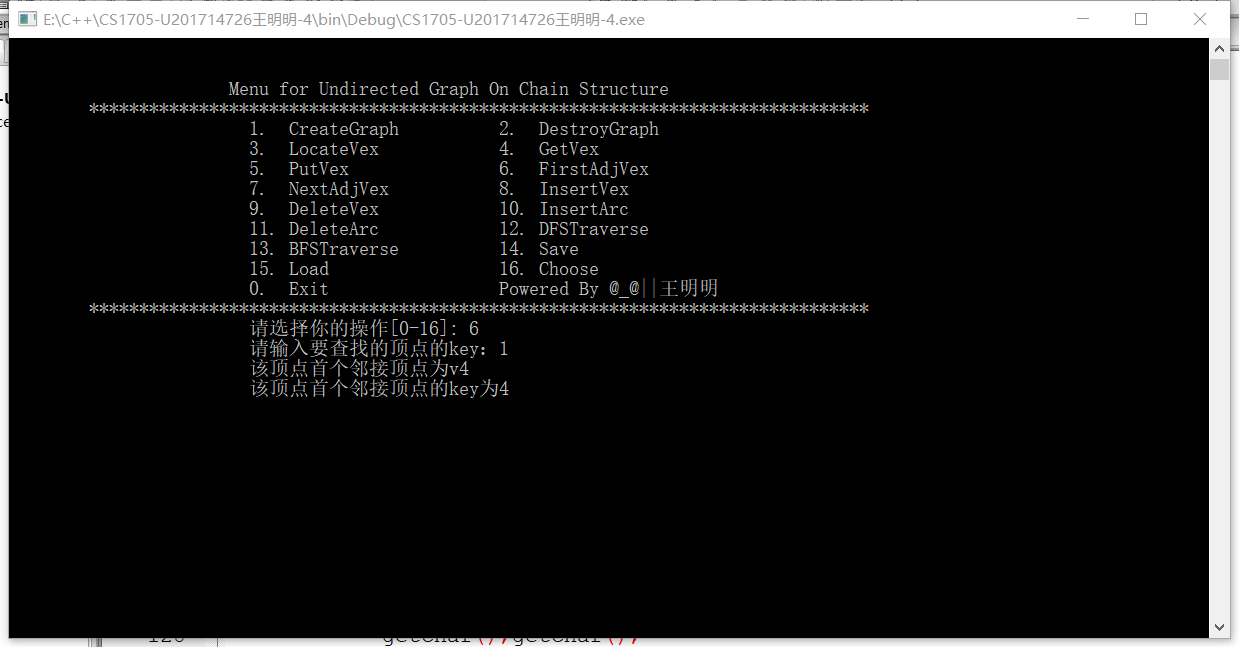
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| zzzh | 主界面选10  输入边的尾和头的值: v1 v4 | 添加成功! | 添加成功! |
| 若图不存在 | 主界面选10 | 图不存在！ | 图不存在！ |



(a)插入边之前v1的首个邻接顶点



(b)添加边成功时截图



(c)添加边后v1的邻接顶点变为v4

图4-14 InsertArc测试截图

(9) DeleteArc测试 (如表4-9所示)

表4-9 DeleteArc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  输入边的尾和头的值: v1 v4 | 删除成功！ | 删除成功！ |
| 若图不存在 | 主界面选11 | 图不存在！ | 图不存在！ |

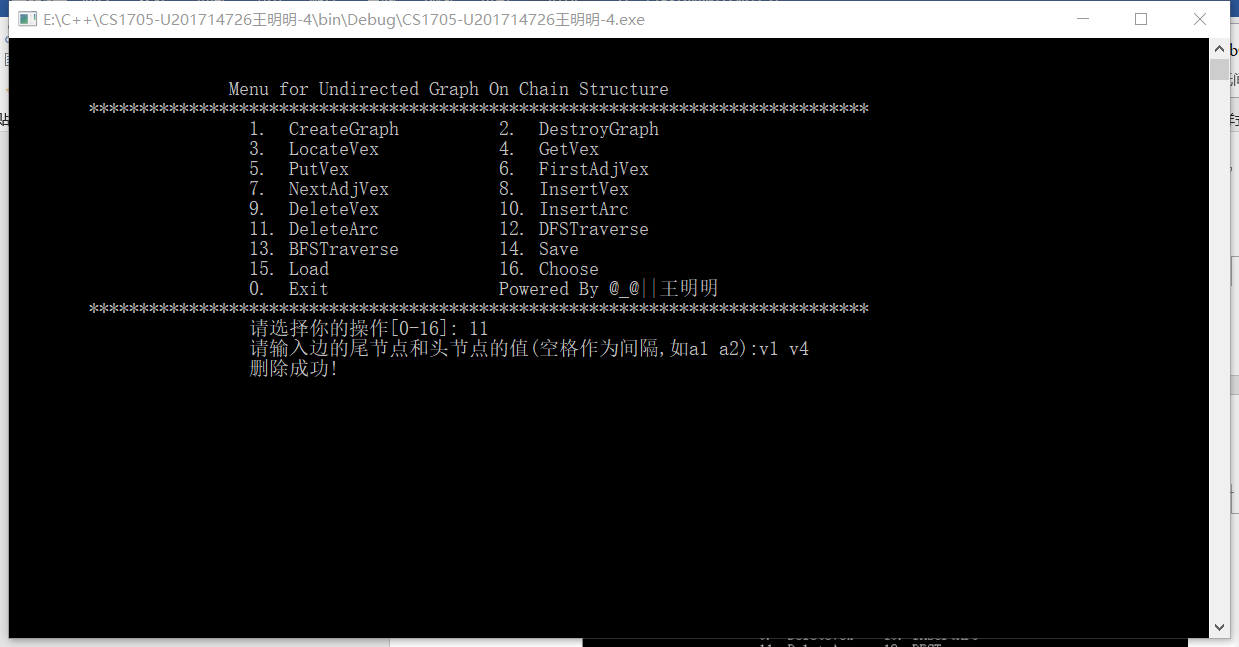


图4-15 DeleteArc测试截图

(10) DFSTraverse测试 (如表4-10所示)

表4-10 DFSTraverse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 | 该图的深度优先搜索遍历为：v1 v2 v3 v4 v5 | 该图的深度优先搜索遍历为：v1 v2 v3 v4 v5 |
| 若图不存在 | 主界面选12 | 图不存在！ | 图不存在！ |



图4-16 DFSTraverse测试截图

(11) BFSTraverse测试 (如表4-11所示)

表4-11 BFSTraverse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13 | 该图的广度优先搜索遍历为：v1 v2 v3 v4 v5 | 该图的广度优先搜索遍历为：v1 v2 v3 v4 v5 |
| 若图不存在 | 主界面选13 | 图不存在！ | 图不存在！ |

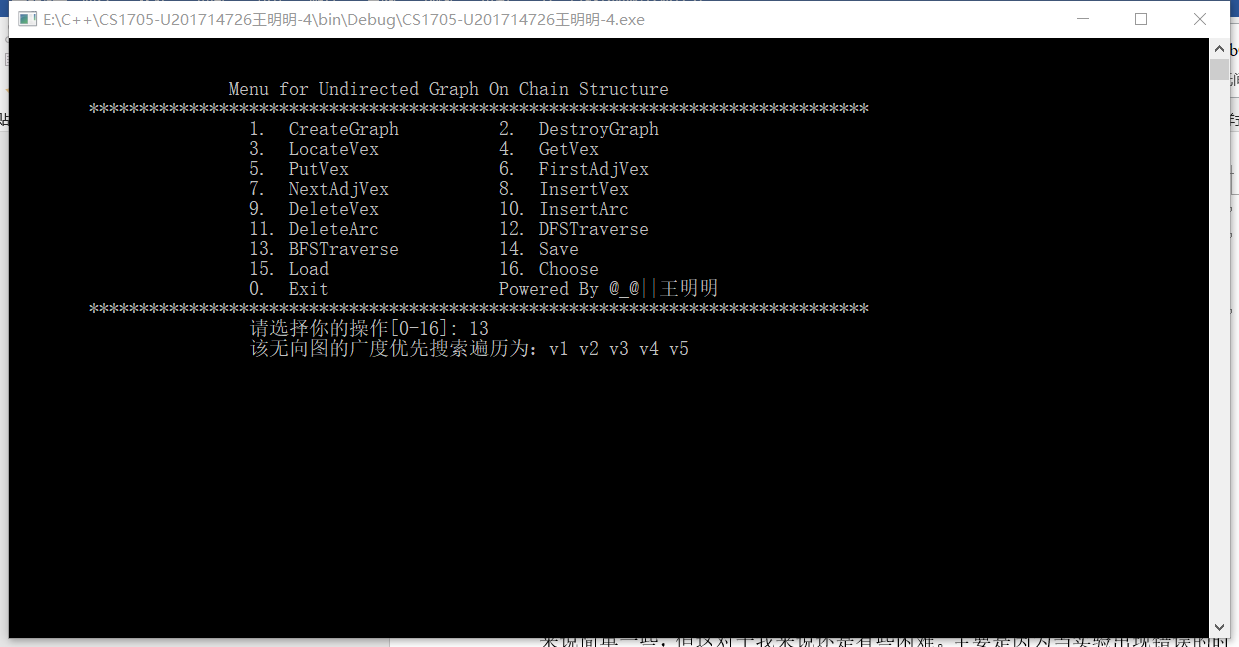
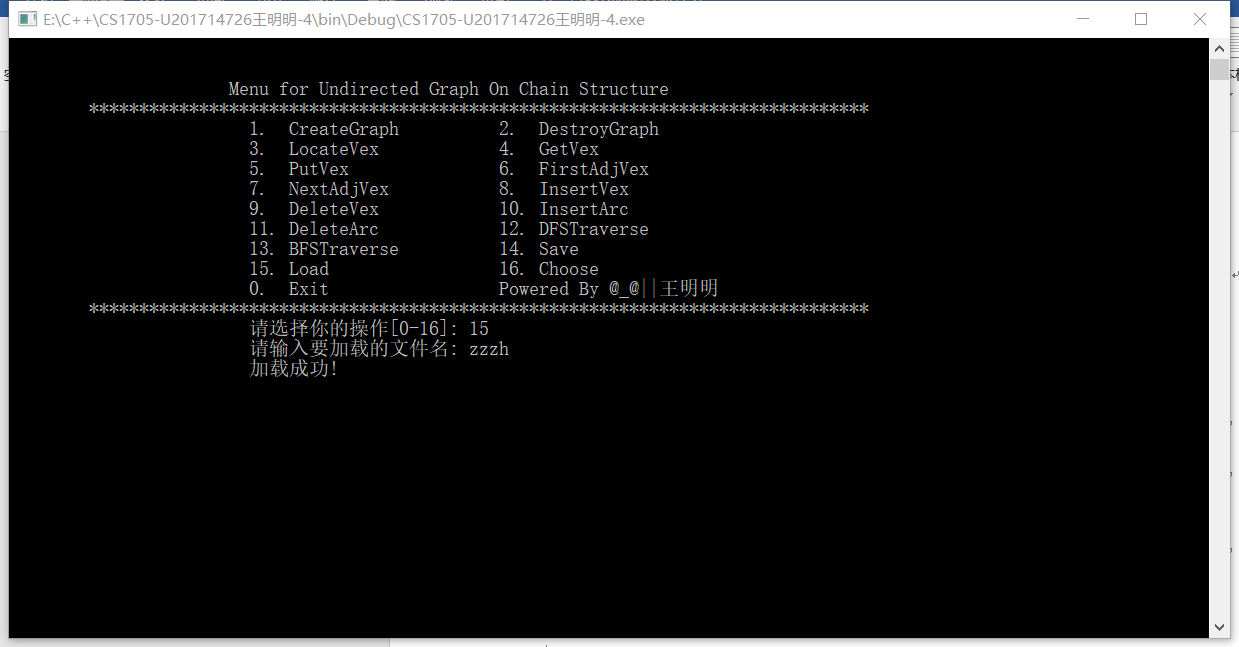


图4-17 BFSTraverse测试截图

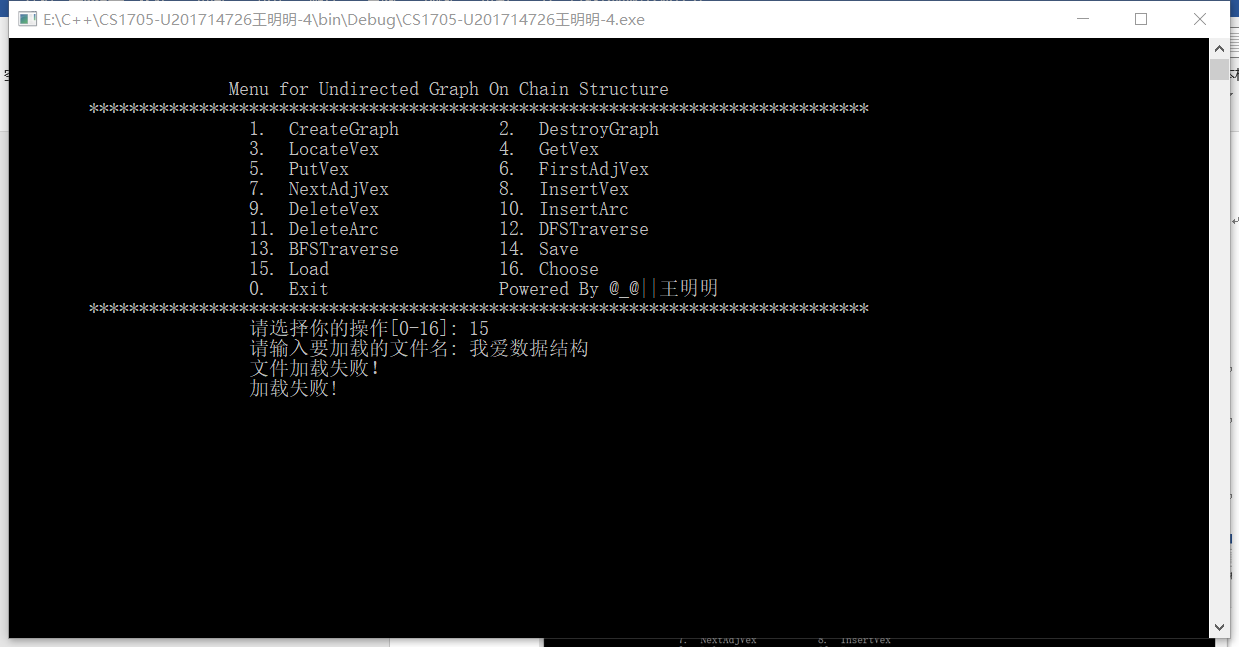
(12)Load测试

表4-12Load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15 | 加载成功！ | 加载成功！ |
| 我爱数据结构 | 主界面选15 | 文件加载失败！  加载失败！ | 文件加载失败！  加载失败！ |



(a)文件加载成功时截图



(b)文件加载失败截图

图4-18 Load函数测试截图

## 4.4 实验小结

本次实验加深了对图的概念、基本运算的理解，熟练了掌握图的逻辑结构与物理结构的关系，熟练了掌握图基本运算的实现。

本次实验我完成的是邻接表形式的无向图，相对于有向图、无向网、有向网来说简单一些，但这对于我来说还是有些困难。主要是因为当实验出现错误的时候可能一个晚上也找不出来错误。我在本次实验中有很多不足之处，比如不能清楚直观的表现顶点之间边的关系，只能求顶点的邻接顶点来确定是否删除或插入边成功。

实验四花费了我很长时间，因为临近考试，时间很紧张，但是也使我收获颇丰。短暂的数据结构实验课程告一段落，期待寒假数据结构的课设题目，愿我可以在0和1的世界里找到乐趣。

# 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 |  |  |  |  |  |  |  |