華中科技大學

课 程 实 验 报 告

课程名称： 数据结构实验

专业班级：_____

学　　号：_____

姓　　名：_____

指导教师：_____

报告日期：_____

计算机科学与技术学院

目 录

1 基于顺序存储结构的线性表实现

1.1 问题描述

线性表在物理内存中可以以顺序表的方式实现，即物理上存储位置相邻的两个元素是线性表中的相邻元素，且数据元素的前后关系不变。

本次实验主要完成线性表在物理内存中用顺序表的实现，和定义在其上的一系列算法。

实验要完成的顺序表算法：

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)；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 相等的数据元素的位序，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)，初始条件是线性表 L 已存在；操作结果是依次打印出 L 的每个数据元素。

实验目标：

通过实验达到

⑴ 加深对线性表的概念、基本运算的理解；

⑵ 熟练掌握线性表的逻辑结构与物理结构的关系；

⑶ 物理结构采用顺序表,熟练掌握线性表的基本运算的实现。

的目的。

1.2 系统设计

1.2.1 系统总体设计

图 1-1 系统总体设计示意图

系统具有一个功能菜单。在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

系统中，线性表结构体中含有一个 next 指针域，指向下一个线性表，相当于用链表的方式存储了多个线性表。在系统运行时，通过输入一个给定的 ID 来标识不同的线性表，并对它们进行操作。

系统开始运行时调用函数读取文件中的数据，并提供数据保存功能以实现线性表的文件形式保存。

该演示系统提供的操作有：表的初始化、销毁、清空、判空，求表长、获取数据元素、查找数据元素、获得前驱、获得后继、插入数据元素、删除数据元素、表的遍历、表的选择、数据保存。

在程序中实现消息处理和操作提示，包括数据的输入和输出，错误操作提示、程序的退出。

本系统采用 c++写成，但没有使用 class（因为 api 设计是面向过程样式的），但使用了常量引用一类的 c++特性。

1.2.2 公用头文件 common.h 中的有关定义

在实验中,我使用了一组公用的变量定义和类型定义,均放在 common.h 中，所有的源文件都要包含此头文件。此头文件的内容见附录 A。
可以看到，其中定义了几个返回值的status和一个ElemType。

而在SqList.h中，我定义了线性表的结构和所有的操作API。内容见附录 B。

线性表结构中的 next 指针域是为了管理多个线性表而设置的。

1.2.3 线性表算法的思想和设计

在算法设计中，函数参数的选择我参照了 c++的一些不言自明的标准：在不改变线性表内部结构，数据的情况下，我选择常量引用（const &）类型作为形参，其他时候采用引用类型、

（1）InitList(SqList & L)

设计：分配存储空间，并初始化表长为 0，表容量为 LIST_INIT_SIZE。

操作结果：构造一个空的线性表。

时间空间复杂度：O(1)

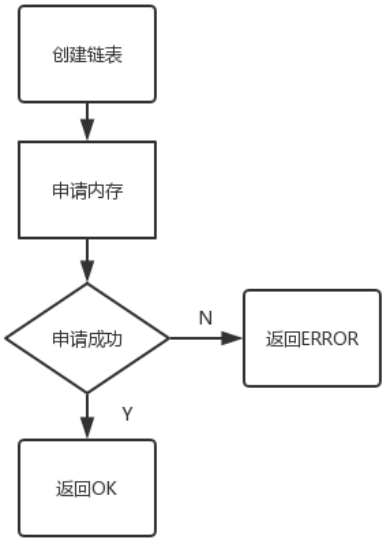

图 1-2 初始化表的流程图

（2）DestroyList(SqList & L)

设计：释放存储空间，每次操作当前线性表，销毁后当前线性表之后的线性表左移一个位序。

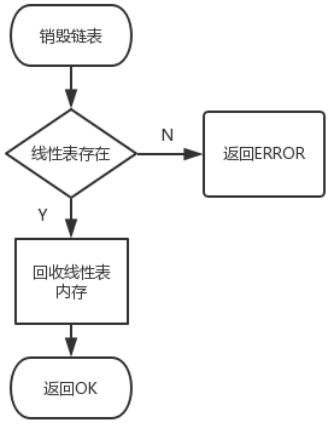

图 1-3 销毁表的流程图

操作结果：销毁线性表 L。

时间空间复杂度：O(1)

（3）ClearList(SqList & L)

设计：线性表 L 的长度赋值为 0

操作结果：将 L 重置为空表。

时间空间复杂度：O(1)

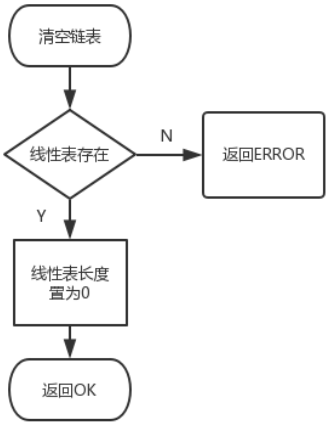

图 1-4 清空表的流程图

（4）ListEmpty(const SqList &L)

设计：根据表长判断表是否为空

操作结果：若 L 为空表，则返回 TRUE,否则返回 FALSE。

时间空间复杂度：O(1)

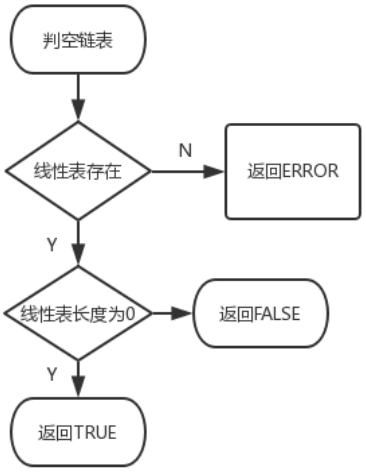

图 1-5 判空链表的流程图

（5）ListLength(const SqList &L)

设计：返回表长

操作结果：返回 L 中数据元素的个数。

时间空间复杂度：O(1)

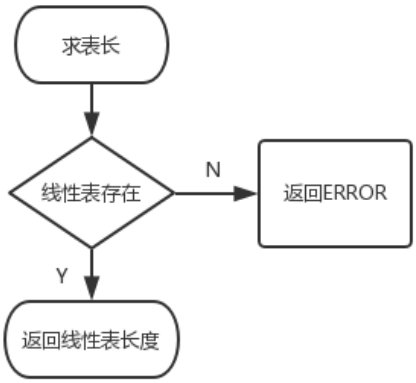

图 1-6 求表长的流程图

（6）GetElem(const SqList &L, int i, ElemType &e)

设计：根据位序找到第 i 个元素的地址并将其值赋值给指针 e 指向的元素

操作结果：用指针 e 指向的元素返回 L 中第 i 个数据元素的值。

时间空间复杂度：O(1)

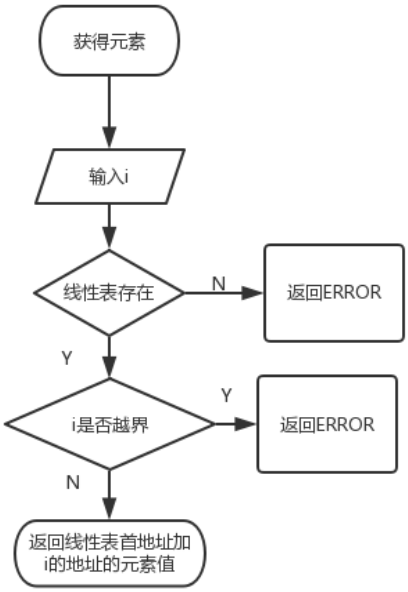

图 1-7 获得元素的流程图

（7）LocateElem(const SqList &L, const ElemType &e)

设计：遍历线性表找到第一个和元素 e 的相等的元素

操作结果：返回 L 中第 1 个与 e 相等的的数据元素的位序，若这样的数据元素不存在，则返回值为 0。

时间空间复杂度：O(1)

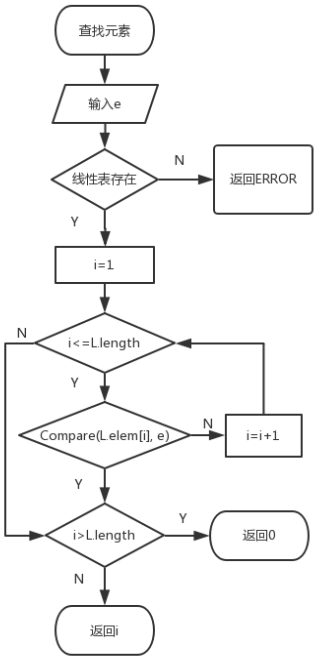

图 1-8 查找元素的流程图

（8）PriorElem(const SqList &L, const ElemType &cur, ElemType &pre_e)

设计：遍历线性表找到第一个和元素 cur 的相等的元素，如果其有前驱，用 pre_e 返回，函数返回 TRUE；否则函数返回 FALSE，pre_e 无意义

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

时间复杂度：O(n)

空间复杂度：O(1)

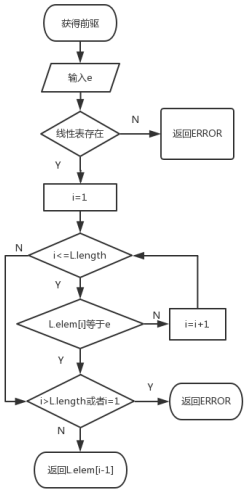

图 1-9 获得前驱的流程图

（9）NextElem（const SqList &L，const ElemType &cur_e，ElemType &next_e）

设计：遍历线性表找到第一个和元素 cur 的相等的元素，如果其有后继，用 next_e 返回，函数返回 TRUE；否则函数返回 FALSE，next_e 无意义

操作结果：若 cur 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。

时间复杂度：O(n)

空间复杂度：O(1)

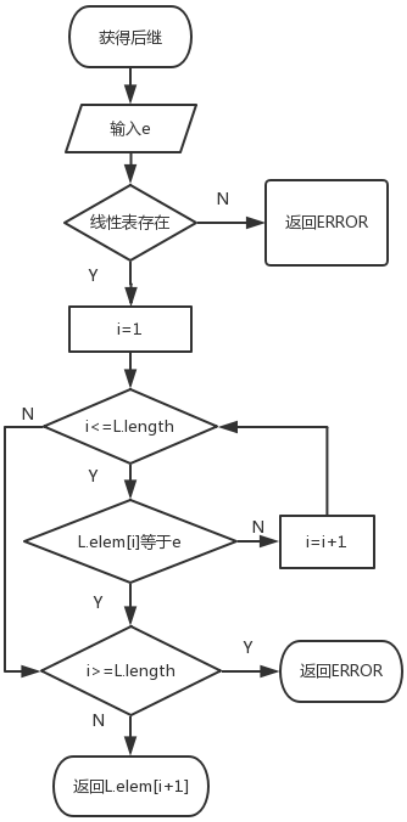

图 1-10 获得后继的流程图

（10）ListInsert(SqList * L, int i, ElemType e)

设计：如果线性表已满，重新分配存储空间。将线性表指针 L 指向的线性表第 i 个元素之后的元素都右移一个位序，之后将 e 插入第 i 个位序

操作结果：在 L 的第 i 个位置之前插入新的数据元素 e，L 的长度加 1

时间复杂度：O(n)

空间复杂度：O(1)

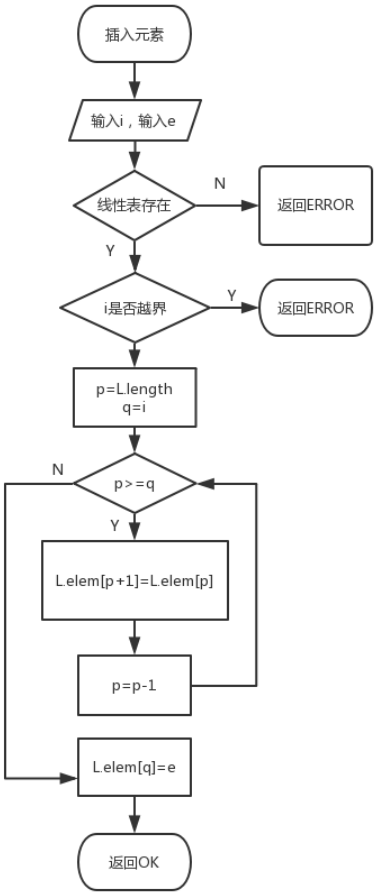

图 1-11 插入元素的流程图

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

设计：将第 i 个位序的值赋给指针 e 指向的变量，之后第 i 个位序之后的元素全部左移一个位序

操作结果：删除 L 的第 i 个数据元素，用 e 返回其值，L 的长度减 1.

时间复杂度：O(n)

空间复杂度：O(1)

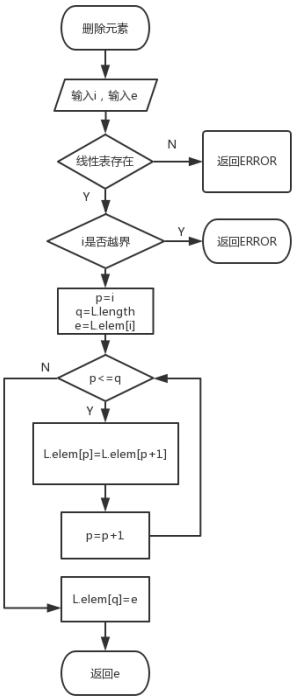

图 1-12 删除元素的流程图

（12）ListTraverse(const SqList &L)

设计：遍历并输出表 L 中的每个元素值，返回表长

操作结果：依次输出表 L 中的每个变量的值

时间复杂度：O(n)

空间复杂度：O(1)

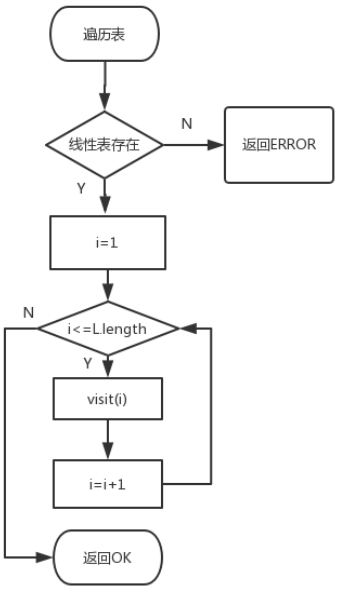

图 1-13 遍历表的流程图

（13）LoadData()

设计：此函数比较特殊，在 main.cpp 中定义，作用是读取文件中的线性表信息。

操作结果:从 SLDB 文件中读取所有线性表的数据。

（14）SaveData()

设计：此函数比较特殊，在 main.cpp 中定义，作用是存储文件中的线性表信息。

操作结果:将所有线性表的信息存储到文件中。

1.3 顺序表系统测试

1.3.1 顺序表演示系统实现说明

本演示系统包括一个循环，每次循环开始打印出演示菜单，菜单如图所示

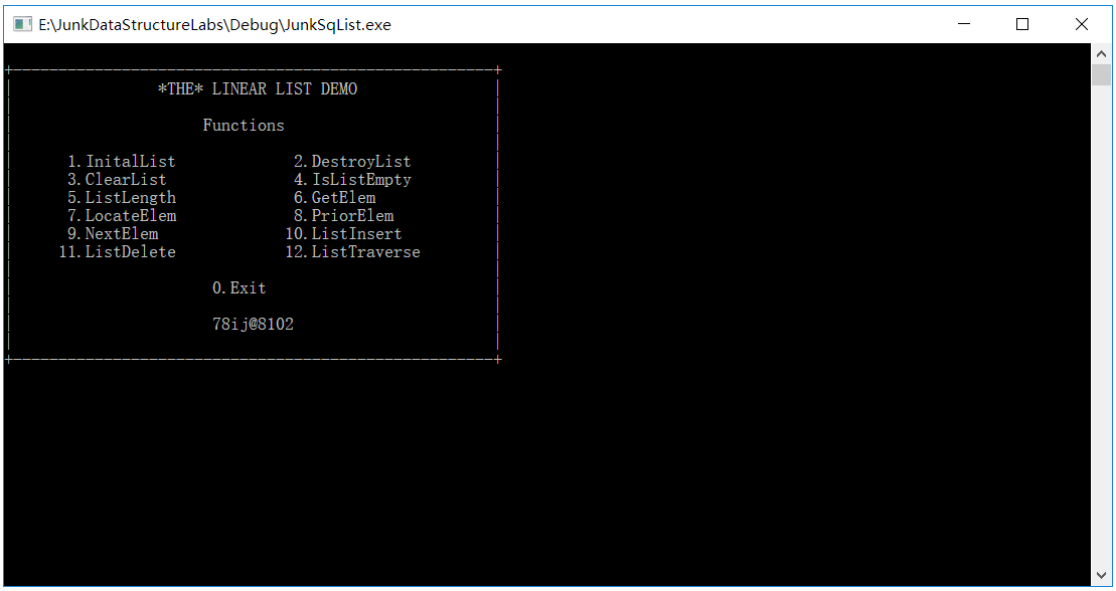

图 1-14 演示系统菜单

每次操作时要求输入本次操作的线性表编号：

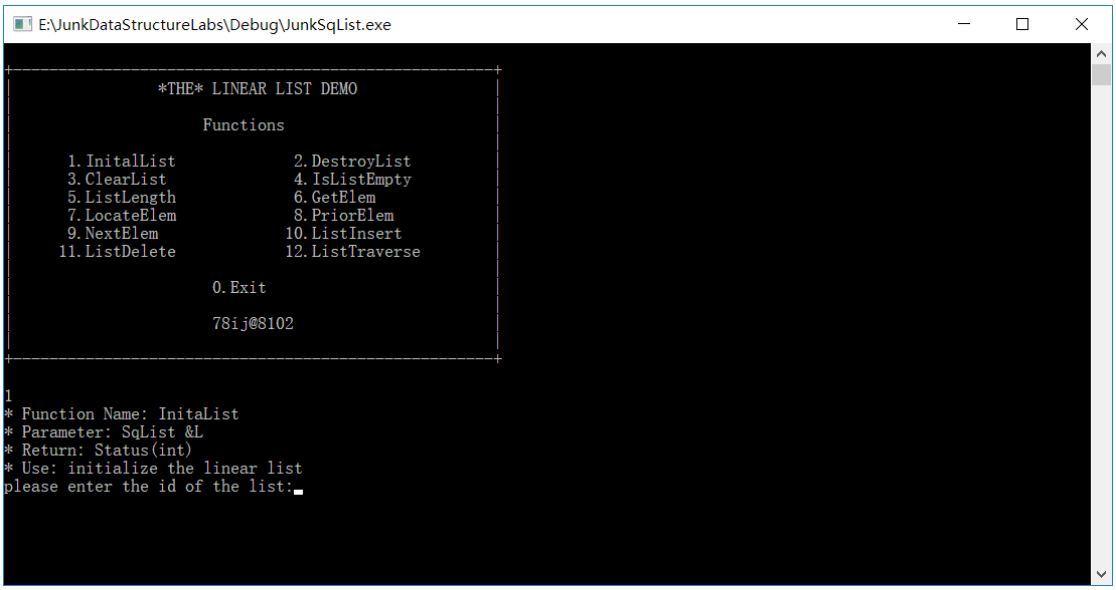

图 1-15 演示系统要求输入编号

每次打印出菜单之前，从文件中将所有线性表读取到内存中，每次进行完毕一个操作之后，将内存中的线性表都存到文件之中。

1.3.2 系统测试

下面，选取几个具有代表性的函数进行测试。

（1） 初始化表

表 1-1 初始化表的数据及结果

操作	输入	输出	是否正确
初始化表	无	创建成功	正确

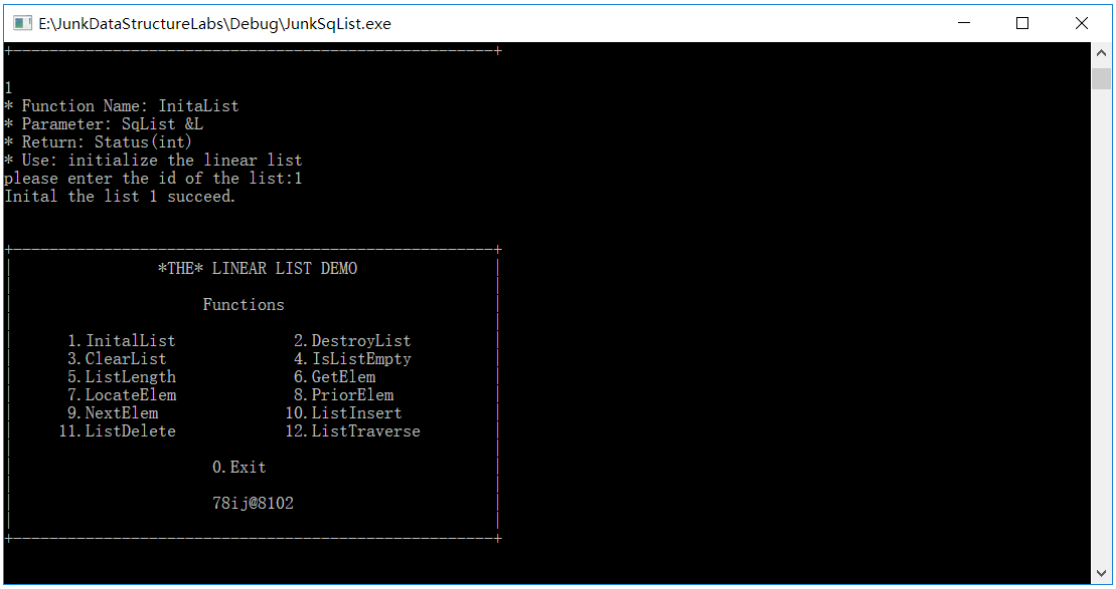

图 1-16 初始化表

（2） 销毁表

表 1-2 销毁表的数据及结果

操作	输入	输出	是否正确
销毁表	表存在	销毁成功	正确
销毁表	表不存在	销毁失败	正确

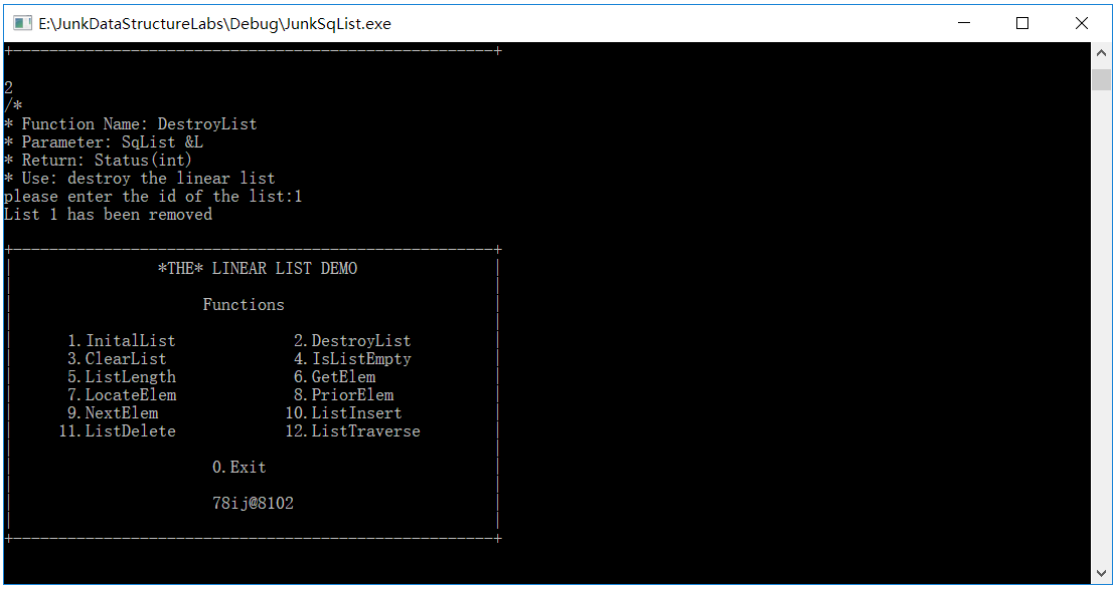

图 1-17 销毁表

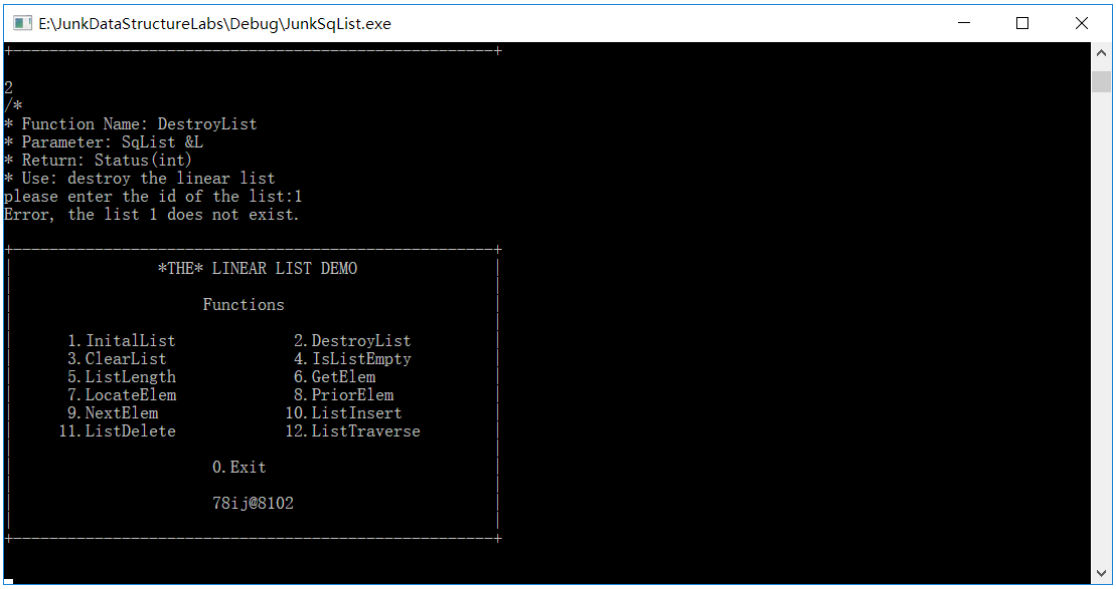

图 1-17 销毁表（2）

（3） 求表长

表 1-3 求表长的数据及结果

操作	输入	输出	是否正确
求表长	表长为 2 的表	2	正确
求表长	空表	0	正确

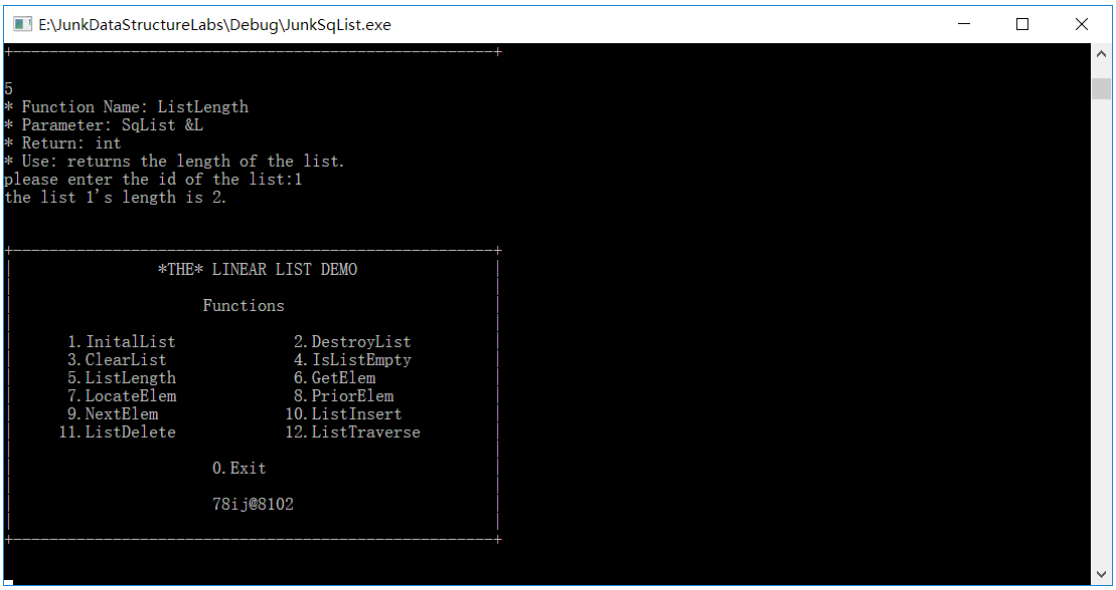

图 1-18 求表长

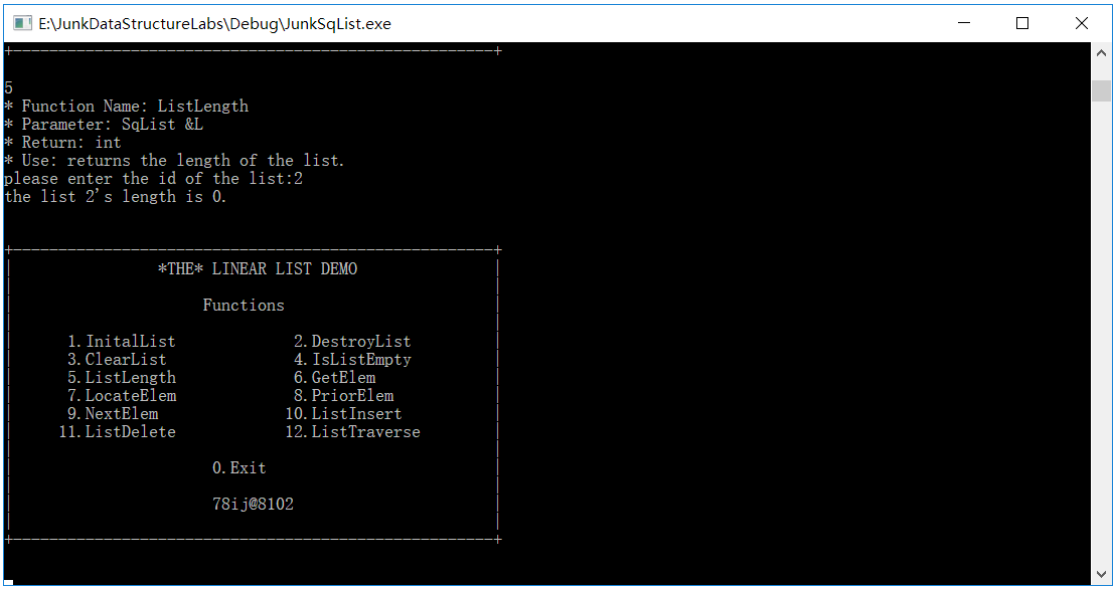

图 1-18 求表长（2）

（4） 插入元素

表 1-4 插入元素的数据及结果

操作	输入	输出	是否正确
插入元素	空表，第 1 号元素	插入成功	正确
插入元素	表长 1，第 4 号元素	插入失败	正确

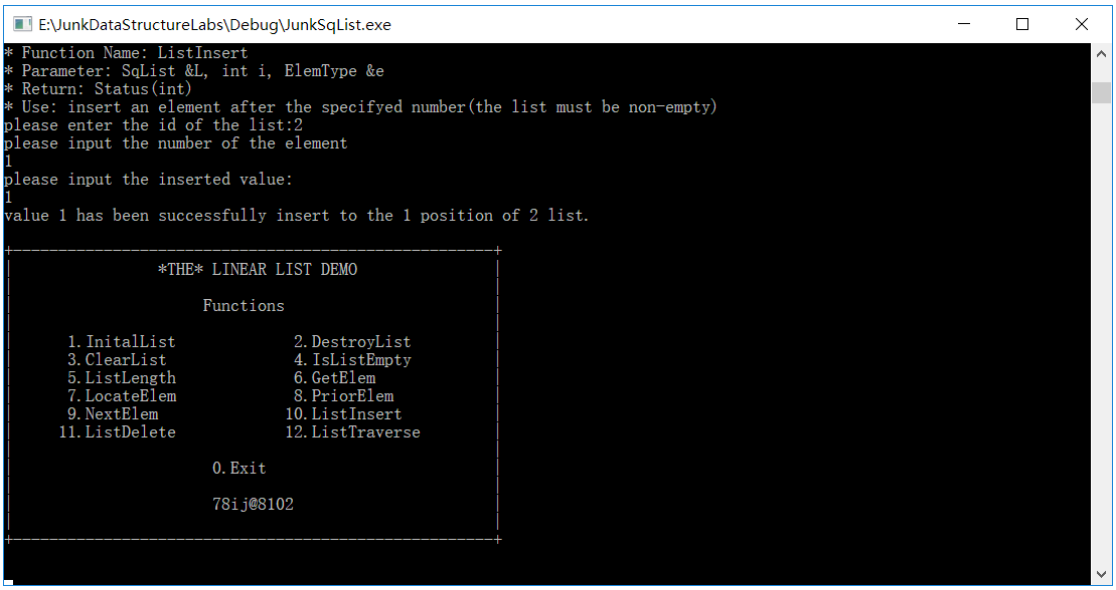

图 1-19 插入元素

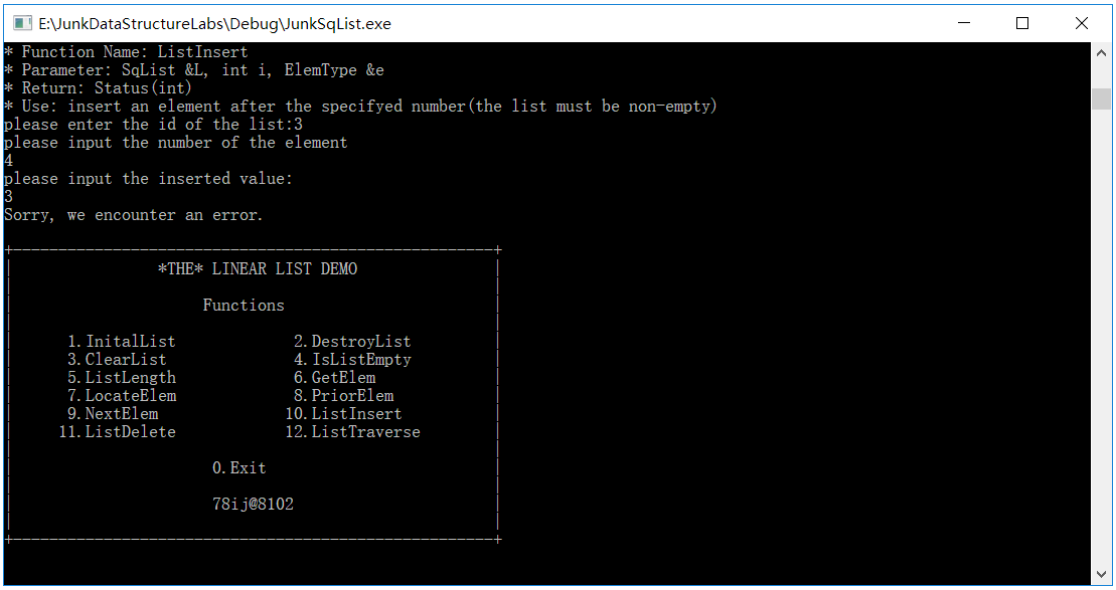

图 1-20 插入元素（2）

（5） 判断空表

表 1-5 初始化表的数据及结果

操作	输入	输出	是否正确
判断空表	空表	True	正确
判断空表	表不存在	Error	正确

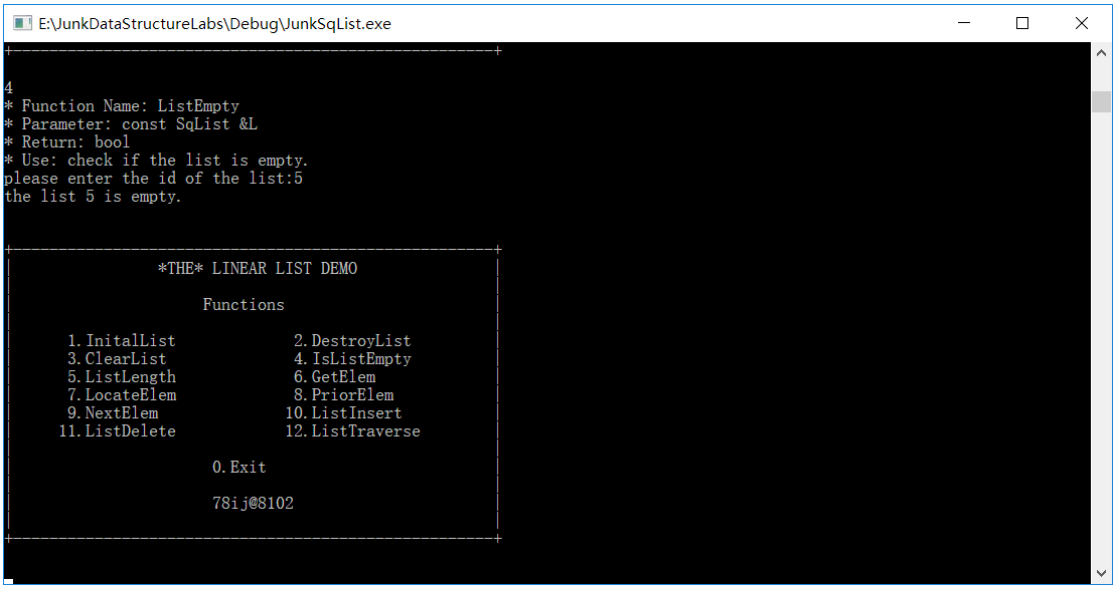

图 1-21 判断空表

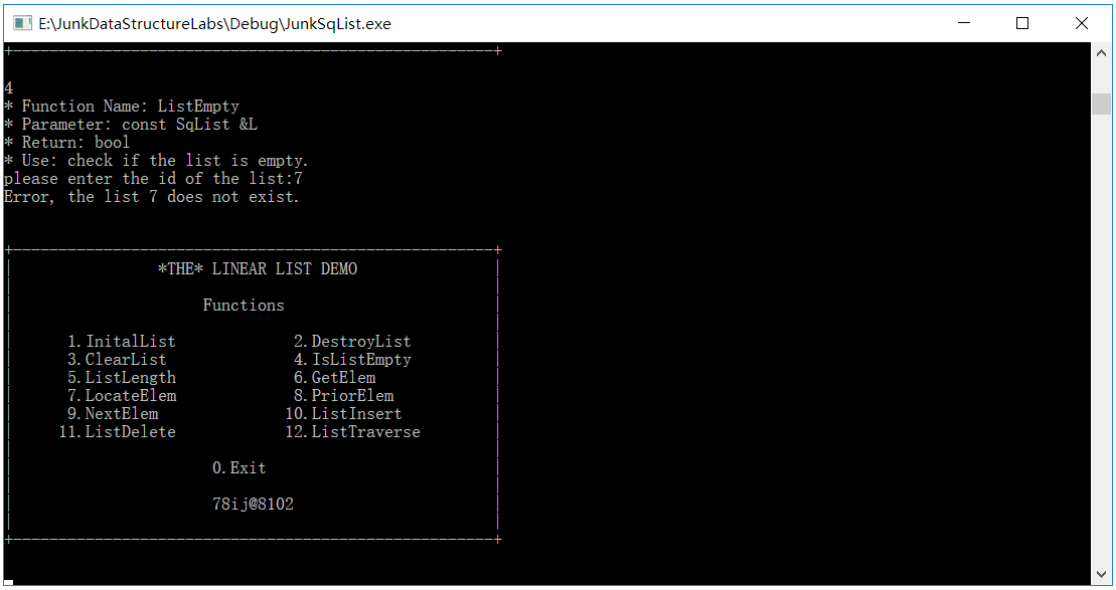

图 1-22 判断空表（2）

（6） 求前驱

表 1-6 求前驱的数据及结果

操作	输入	输出	是否正确
求前驱	表长为 4，第 4 号	第三号	正确
求前驱	表长为 4，第 6 号	错误	正确

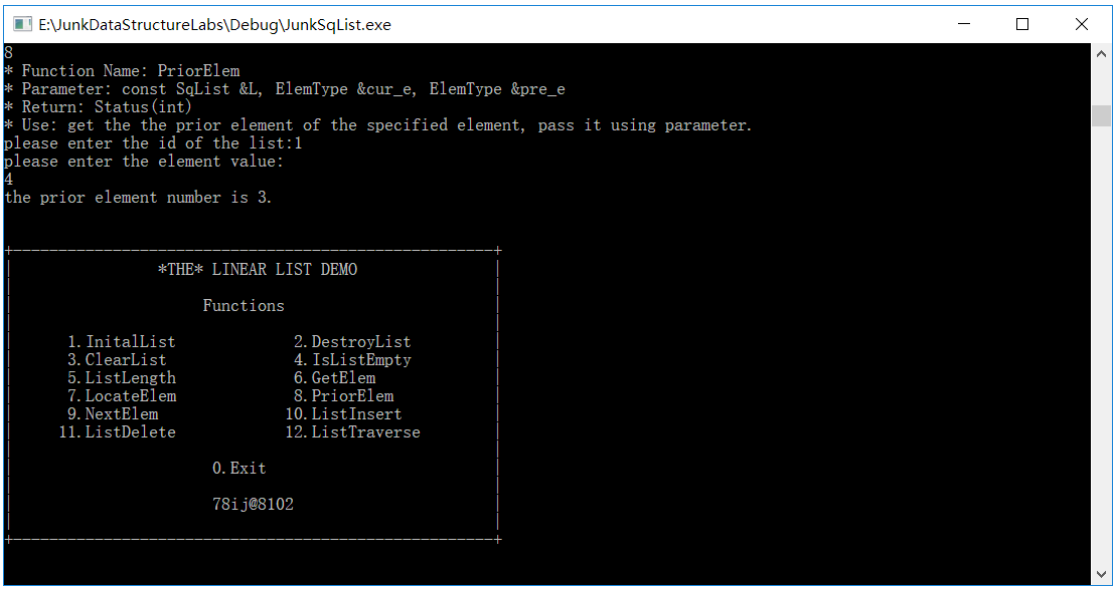

图 1-23 求前驱

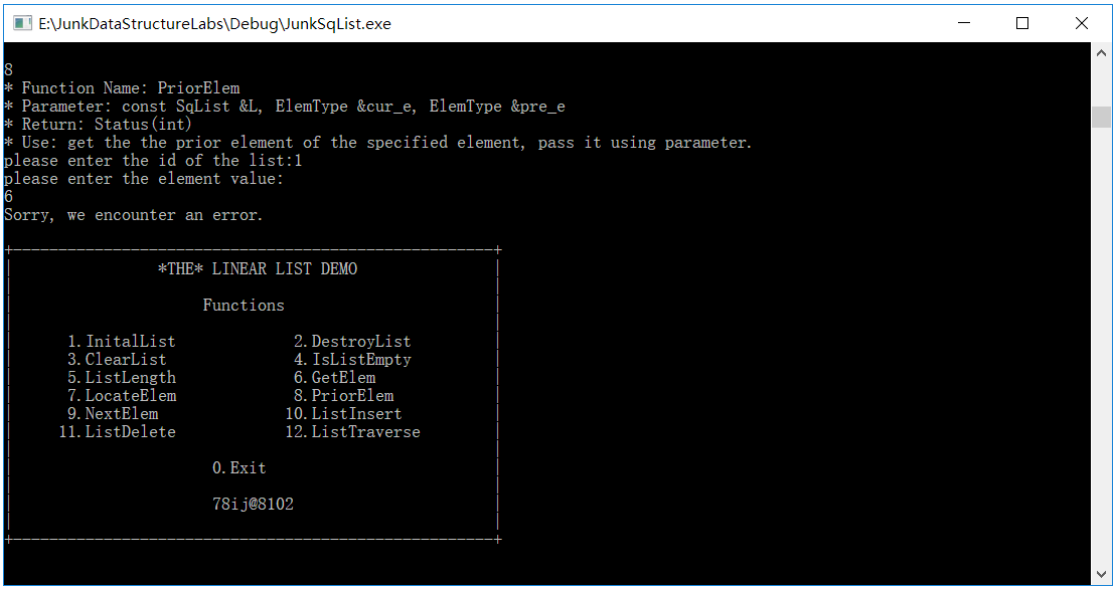

图 1-24 求前驱（2）

（7） 遍历表

表 1-7 遍历表的数据及结果

操作	输入	输出	是否正确
遍历表	空表	无	正确
遍历表	6 元素表	6 个元素	正确

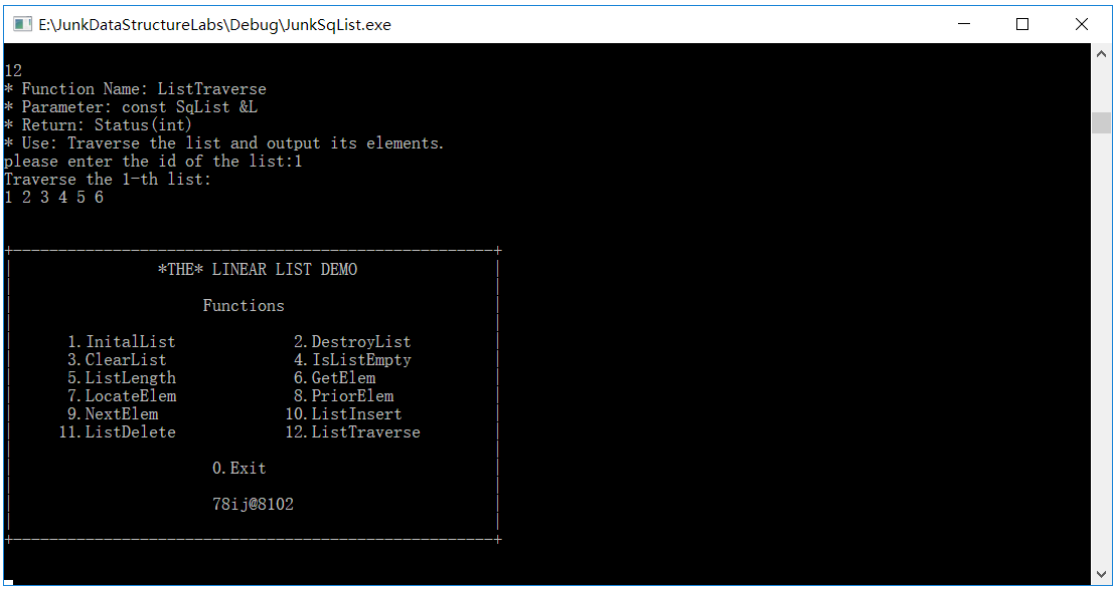

图 1-25 遍历表

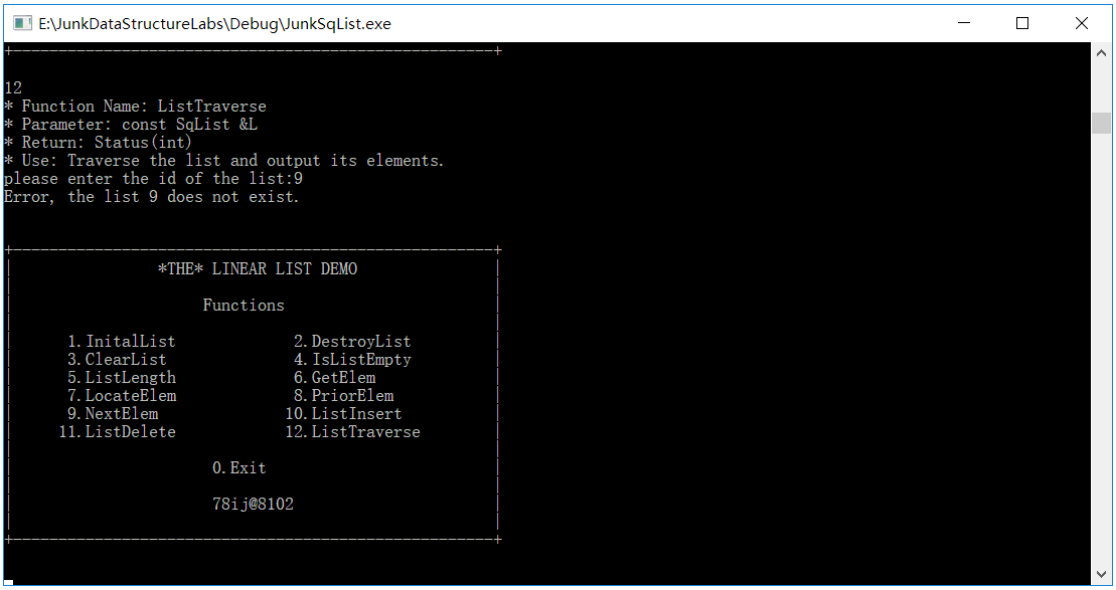

图 1-26 遍历表（2）

1.4 实验小结

本次实验中，我编写了线性表数据结构，其有关 API 和演示系统。在本次实验的过程中，我学习了如何编写程序使各个模块之间良好写作，在实验检查的过程中，也暴露出我编写程序容错性不够高，等错误。在今后的编程实践中，我将注意这些问题，并再接再厉，争取提高自己的水平。

1.5 附录 A common.h 代码清单

```
2    #ifndef COMMON_H
3    #define COMMON_H
4
5    #include <iostream>
6    #include <malloc.h>
7    #include <cstdlib>
8    #include <cstring>
9    #include <queue>
10
11   using std::cin;
12   using std::cout;
13   using std::endl;
14   using std::memset;
15   using std::queue;
16
17
18   //Page 10 on Textbook
19   #define OK                1
20   #define ERROR             0
21   #define INFEASTABLE -1
22   #define OVERFLOW       -2
23
24   typedef int status;
25   typedef int ElemType;
26
27
28   #endif //#ifndef COMMON_H
```

1.6 附录 B SqList.h 代码清单

```
#ifndef SqList_H
#define SqList_H

#include "common.h"

//page 22 on textbook
#define LIST_INIT_SIZE 100
#define LISTINCREMENT    10
#define LIST_MAX 500

//线性表的结构
typedef struct SqList {
        ElemType *head;
        int length;
        int listsize;
        int ListID = -1;
        SqList *next;
}SqList;


//page 19 on textbook
status IntiaList(SqList &L);
status DestroyList(SqList &L);
status ClearList(SqList &L);
bool   ListEmpty(const SqList &L);
int    ListLength(const SqList &L);
status GetElem(const SqList &L,int i, ElemType &e);
int    LocateElem(const SqList &L,const ElemType &e); //简化过
status PriorElem(const SqList &L,const ElemType &cur_e,ElemType &pre_e);
status NextElem(const SqList &L,const ElemType &cur_e,ElemType &next_e);
status ListInsert(SqList &L,int i,ElemType &e);
status ListDelete(SqList &L, int i, ElemType &e);
status ListTraverse(const SqList &L);   //简化过



#endif //SqList_H
```

2 基于链式存储结构的线性表实现

2.1 问题描述

线性表在物理内存中可以以链表的方式实现，即线性表的物理结构是链式结构，数据元素中含有一个指针域，此指针域指向下一个数据元素。

本次实验主要完成线性表在物理内存中用链式表的实现，和定义在其上的一系列算法。

实验要完成的链表算法：

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)；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 相等的数据元素的位序，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)，初始条件是线性表 L 已存在；操作结果是依次打印出 L 的每个数据元素。

实验目标：

通过实验达到

⑷ 加深对线性表的概念、基本运算的理解；

⑸ 熟练掌握线性表的逻辑结构与物理结构的关系；

⑹ 物理结构采用单链表,熟练掌握线性表的基本运算的实现。

的目的。

2.2 系统设计

2.2.1 系统总体设计

图 2-1 系统总体设计示意图

系统具有一个功能菜单。在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

系统中，线性表结构体中含有一个 next 指针域，指向下一个线性表，相当于用链表的方式存储了多个线性表。在系统运行时，通过输入一个给定的 ID 来标识不同的线性表，并对它们进行操作。

系统开始运行时调用函数读取文件中的数据，并提供数据保存功能以实现线性表的文件形式保存。

该演示系统提供的操作有：表的初始化、销毁、清空、判空，求表长、获取数据元素、查找数据元素、获得前驱、获得后继、插入数据元素、删除数据元素、表的遍历、表的选择、数据保存。

在程序中实现消息处理和操作提示，包括数据的输入和输出，错误操作提示、程序的退出。

本系统采用 c++写成，但没有使用 class（因为 api 设计是面向过程样式的），但使用了常量引用一类的 c++特性。

1.2.3 线性表算法的思想和设计

在算法设计中，函数参数的选择我参照了 c++的一些不言自明的标准：在不改变线性表内部结构，数据的情况下，我选择常量引用（const &）类型作为形参，其他时候采用引用类型、

（1）InitList(SqList & L)

设计：分配存储空间，并初始化表长为 0。

操作结果：构造一个空的线性表。

时间空间复杂度：0(1)

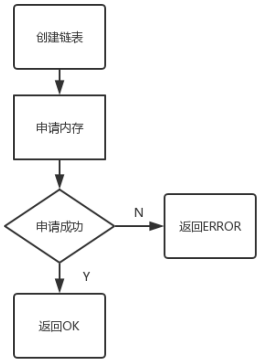

图 2-2 初始化表的流程图

（2）DestroyList(SqList & L)

设计：释放存储空间，每次操作当前线性表，销毁后当前线性表之后的线性表左移一个位序。

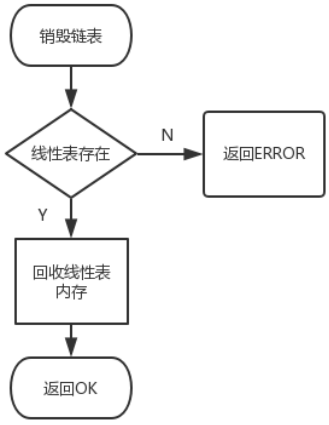

图 2-3 销毁表的流程图

操作结果：销毁线性表 L。

时间空间复杂度：O(1)

（3）ClearList(SqList & L)

设计：线性表 L 的长度赋值为 0

操作结果：将 L 重置为空表。

时间空间复杂度：O(1)

（4）ListEmpty(const SqList &L)

设计：根据表长判断表是否为空

操作结果：若 L 为空表，则返回 TRUE,否则返回 FALSE。

时间空间复杂度：O(1)

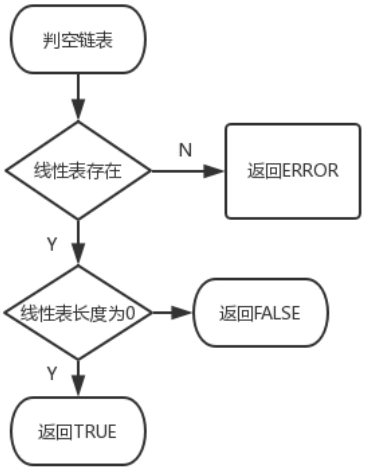

图 2-4 判空链表的流程图

（5）ListLength(const SqList &L)

设计：返回表长

操作结果：返回 L 中数据元素的个数。

时间空间复杂度：O(1)

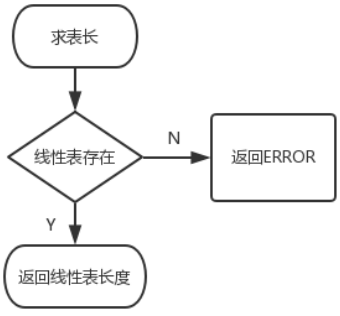

图 2-5 求表长的流程图

（6）GetElem(const SqList &L, int i, ElemType &e)

设计：遍历链表，找到元素 e 并将其值赋值给指针 e 指向的元素

操作结果：用指针 e 指向的元素返回 L 中第 i 个数据元素的值。

时间复杂度：O(n)

空间复杂度：O(1)

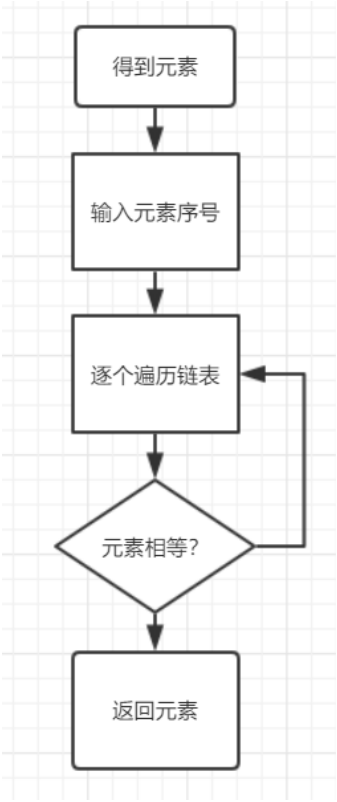

图 2-6 得到元素的流程图

（7）LocateElem(const SqList &L, const ElemType &e)

设计：遍历线性表找到第一个和元素 e 的相等的元素

操作结果：返回 L 中第 1 个与 e 相等的的数据元素的位序，若这样的数据元素不存在，则返回值为 0。

时间复杂度：O(n)

空间复杂度：O(1)

（8）PriorElem(const SqList &L, const ElemType &cur, ElemType &pre_e)

设计：遍历线性表找到第一个和元素 cur 的相等的元素，如果其有前驱，用 pre_e 返回，函数返回 TRUE；否则函数返回 FALSE，pre_e 无意义

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

时间复杂度：O(n)

空间复杂度：O(1)

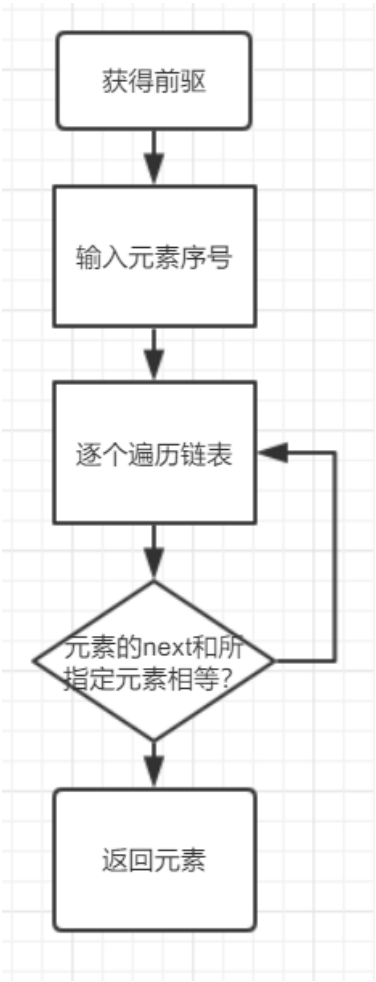

图 2-7 获得前驱的流程图

（9）NextElem（const SqList &L，const ElemType &cur_e，ElemType &next_e）

设计：遍历线性表找到第一个和元素 cur 的相等的元素，如果其有后继，用 next_e 返回，函数返回 TRUE；否则函数返回 FALSE，next_e 无意义

操作结果：若 cur 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。

时间复杂度：O(n)

空间复杂度：O(1)

（10）ListInsert(SqList * L, int i, ElemType e)

设计：将所给元素位序上的 next 指针指向所给元素 e，并将 e 的 next 指针指向下一个元素

操作结果：在 L 的第 i 个位置之前插入新的数据元素 e，L 的长度加 1

时间复杂度：O(1)

空间复杂度：O(1)

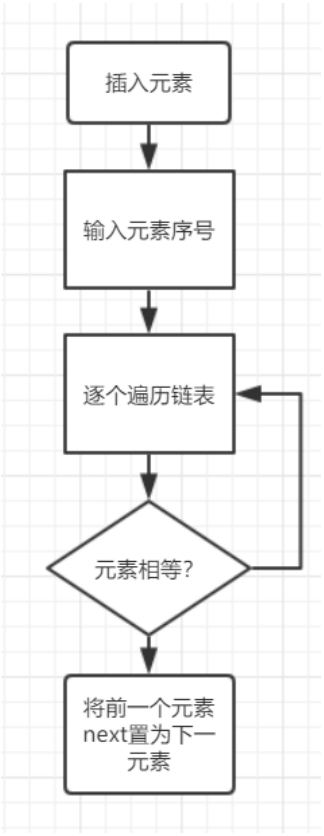

图 2-8 插入元素的流程图

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

设计：将 e 元素

操作结果：删除 L 的第 i 个数据元素，用 e 返回其值，L 的长度减 1.

时间复杂度：O(n)

空间复杂度：O(1)

（12）ListTraverse(const SqList &L)

设计：遍历并输出表 L 中的每个元素值，返回表长

操作结果：依次输出表 L 中的每个变量的值

时间复杂度：O(n)

空间复杂度：O(1)

（13）LoadData()

设计：此函数比较特殊，在 main.cpp 中定义，作用是读取文件中的线性表信息。

操作结果:从 SLDB 文件中读取所有线性表的数据。

（14）SaveData()

设计：此函数比较特殊，在 main.cpp 中定义，作用是存储文件中的线性表信息。

操作结果:将所有线性表的信息存储到文件中。

2.3 链表系统测试

2.3.1 链表演示系统实现说明

本演示系统包括一个循环，每次循环开始打印出演示菜单，菜单如图所示

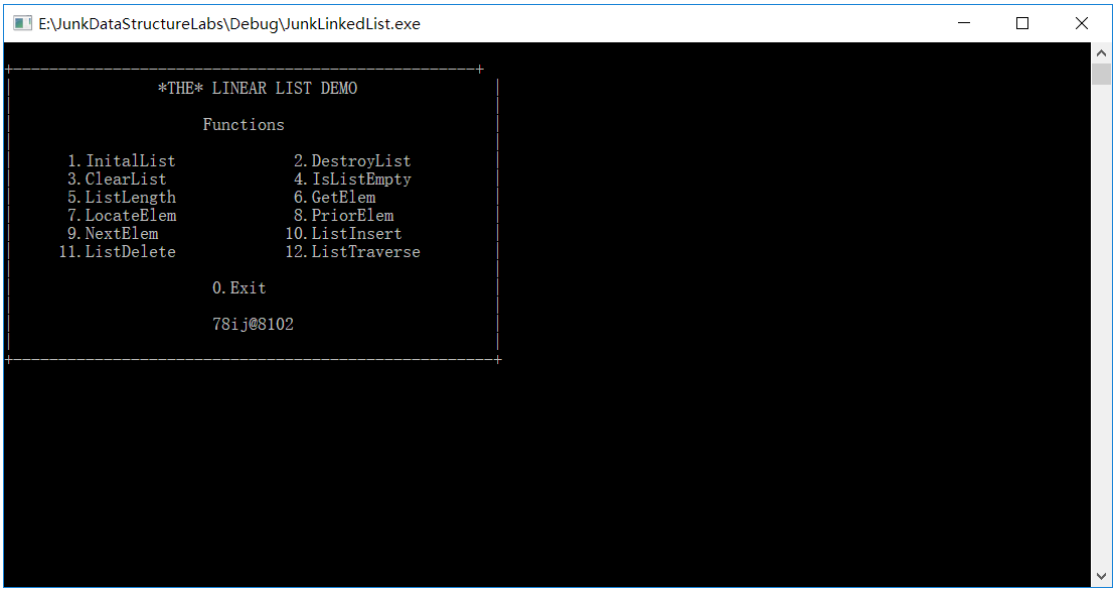

图 1-14 演示系统菜单

每次操作时要求输入本次操作的线性表编号：

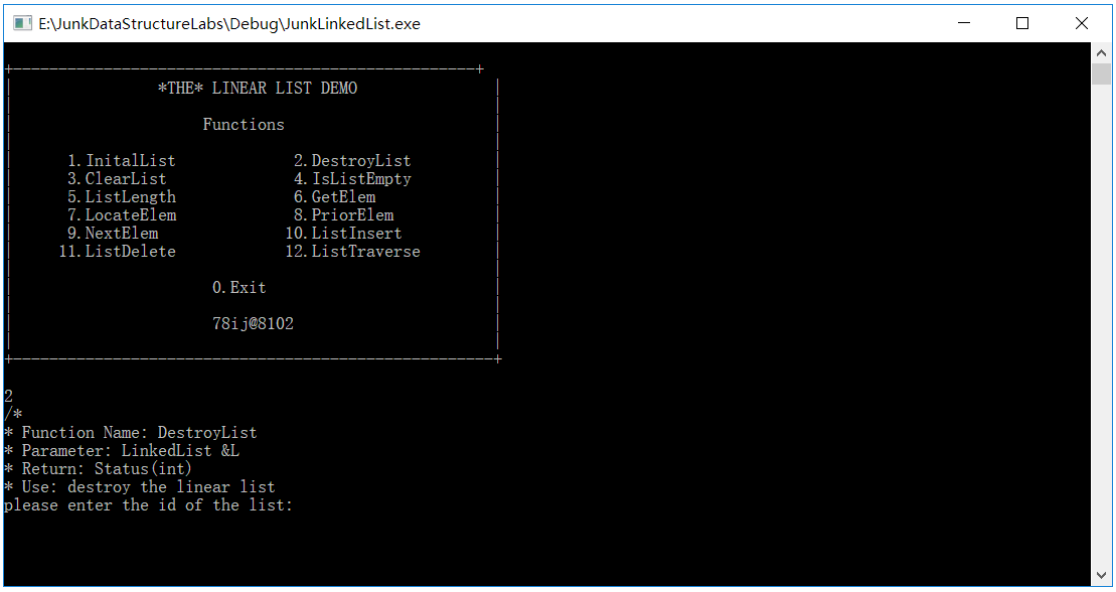

图 1-15 演示系统要求输入编号

每次打印出菜单之前，从文件中将所有线性表读取到内存中，每次进行完毕一个操作之后，将内存中的线性表都存到文件之中。

2.3.2 系统测试

下面，选取几个具有代表性的函数进行测试。

（8） 初始化表

表 1-1 初始化表的数据及结果

操作	输入	输出	是否正确
初始化表	无	创建成功	正确

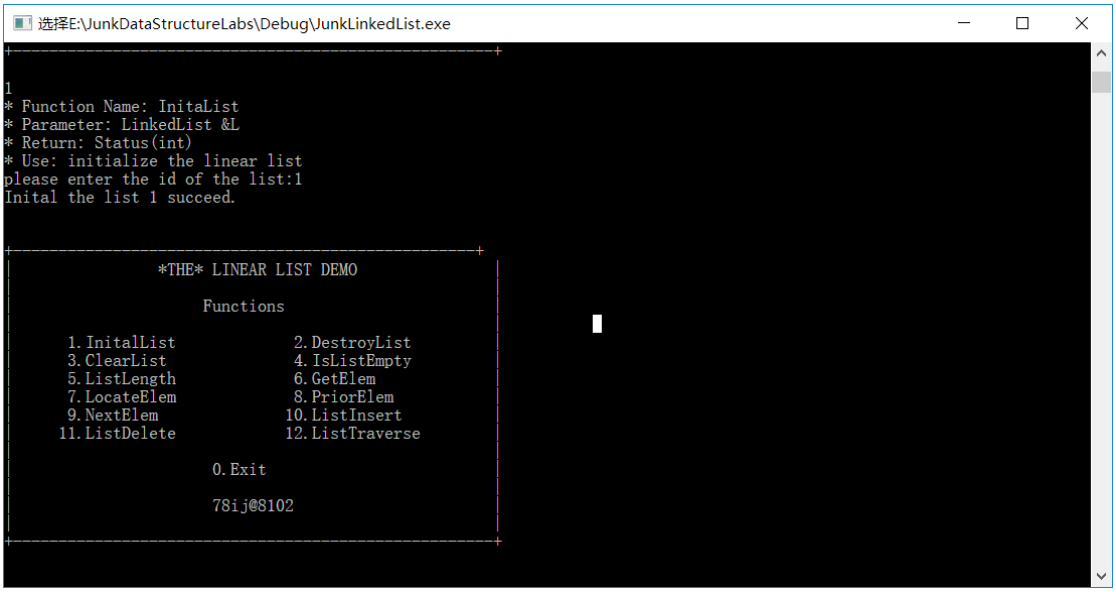

图 1-16 初始化表

（9） 销毁表

表 1-2 销毁表的数据及结果

操作	输入	输出	是否正确
销毁表	表存在	销毁成功	正确
销毁表	表不存在	销毁失败	正确

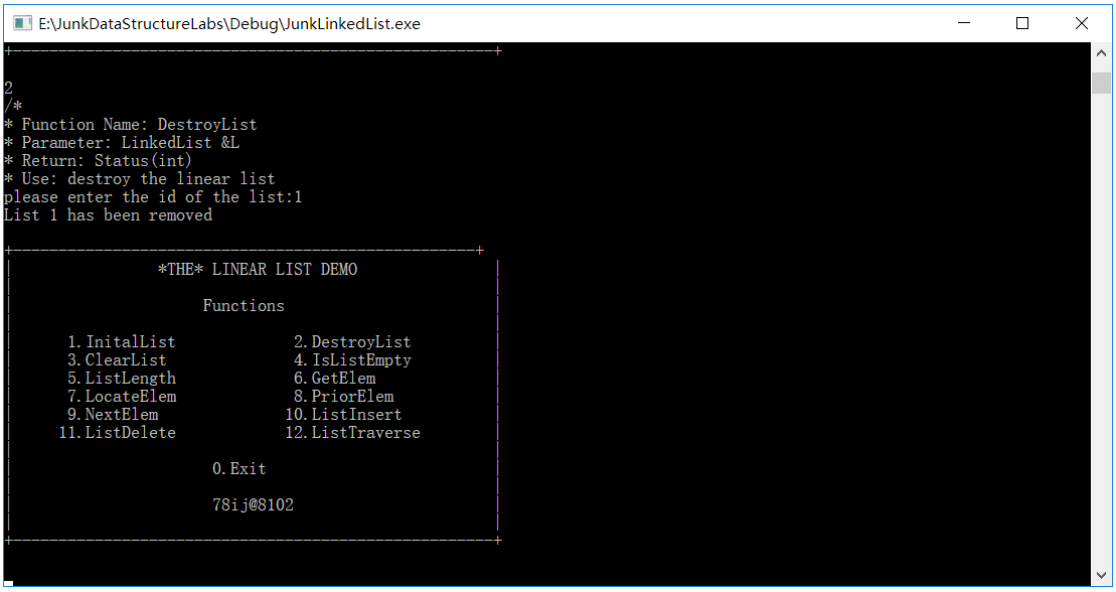

图 1-17 销毁表

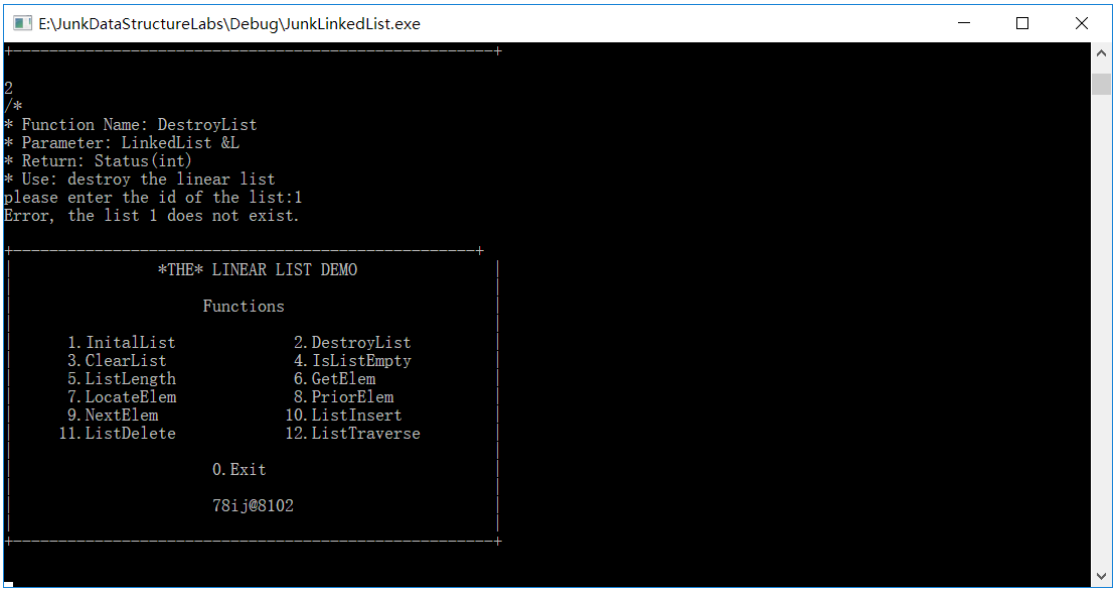

图 1-17 销毁表（2）

（10）求表长

表 1-3 求表长的数据及结果

操作	输入	输出	是否正确

求表长	表长为 5 的表	5	正确
求表长	空表	0	正确

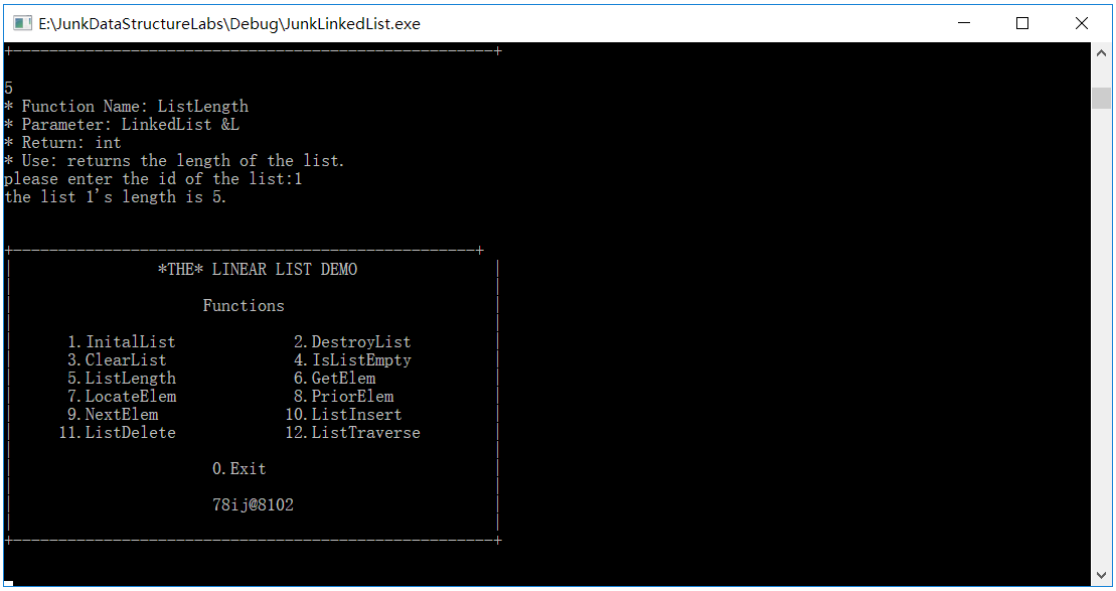

图 1-18 求表长

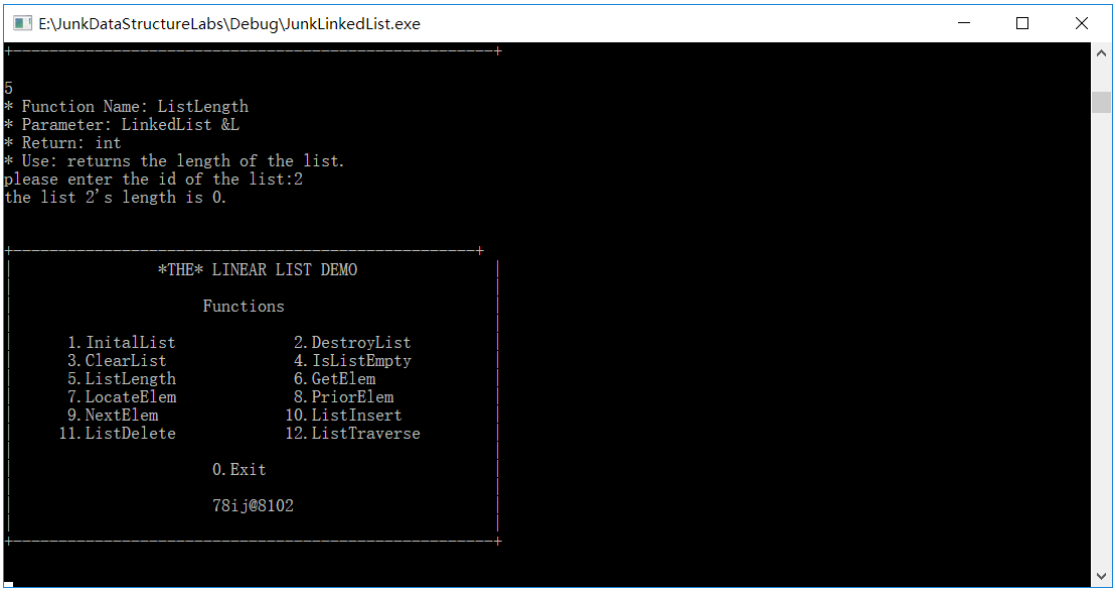

图 1-18 求表长（2）

（11）插入元素

表 1-4 插入元素的数据及结果

操作	输入	输出	是否正确
插入元素	空表，第 1 号元素	插入成功	正确
插入元素	表长 1，第 4 号元素	插入失败	正确

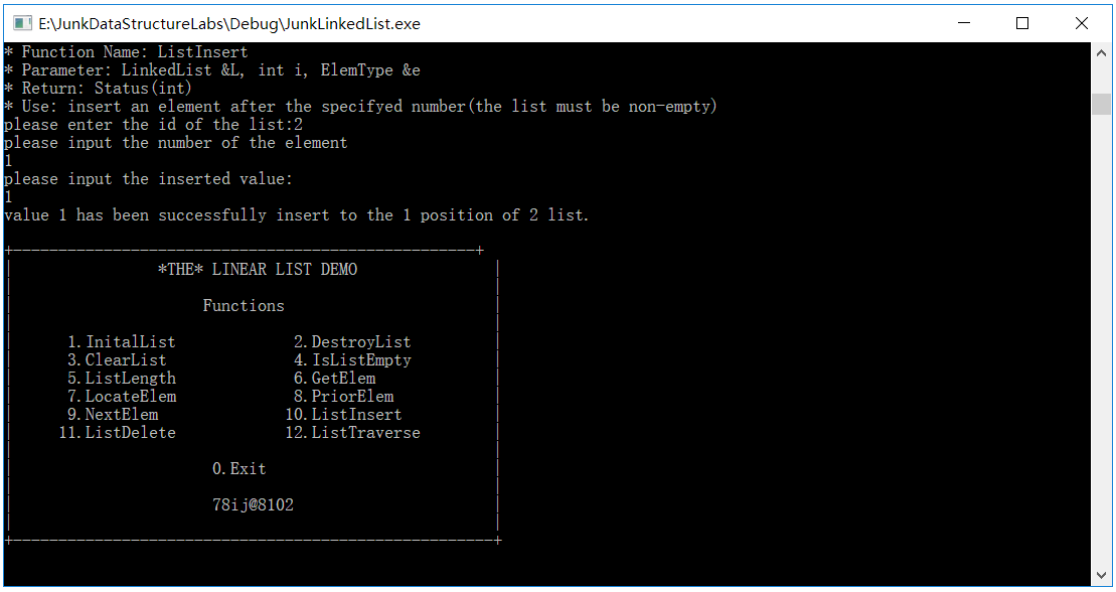

图 1-19 插入元素

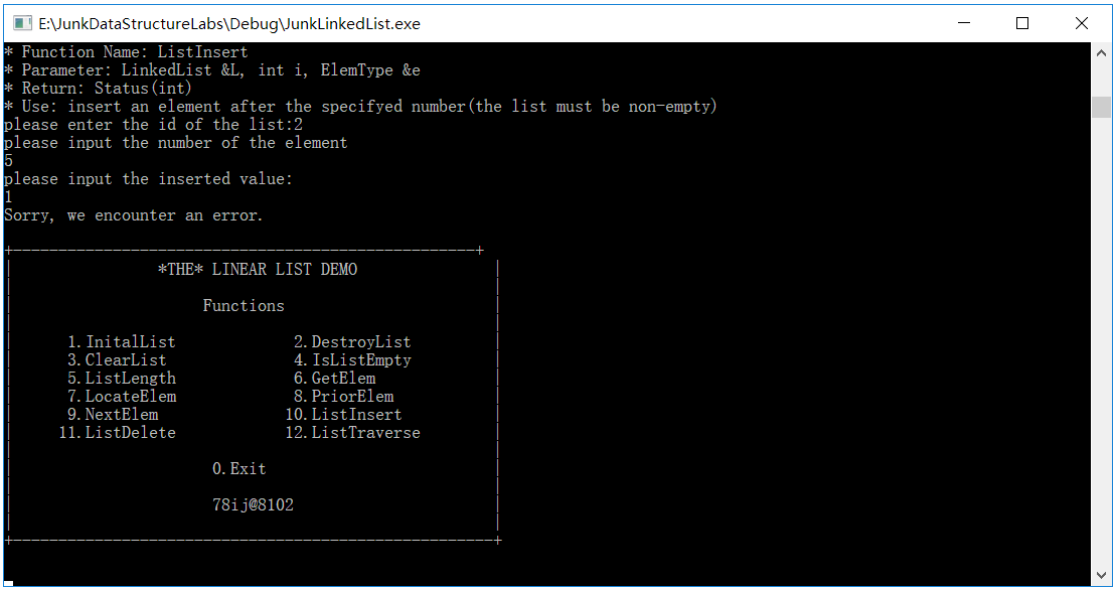

图 1-20 插入元素（2）

（12）求前驱

表 1-5 初始化表的数据及结果

操作	输入	输出	是否正确
求前驱	表 4 1,元素值为 4	1	正确
求前驱	表不存在	Error	正确

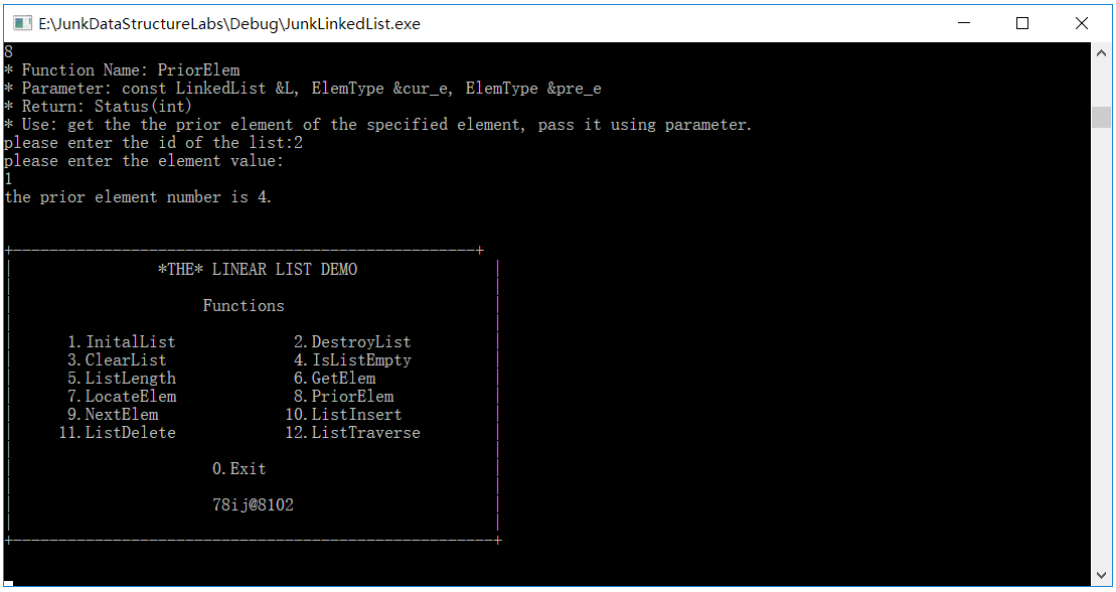

图 1-21 求前驱

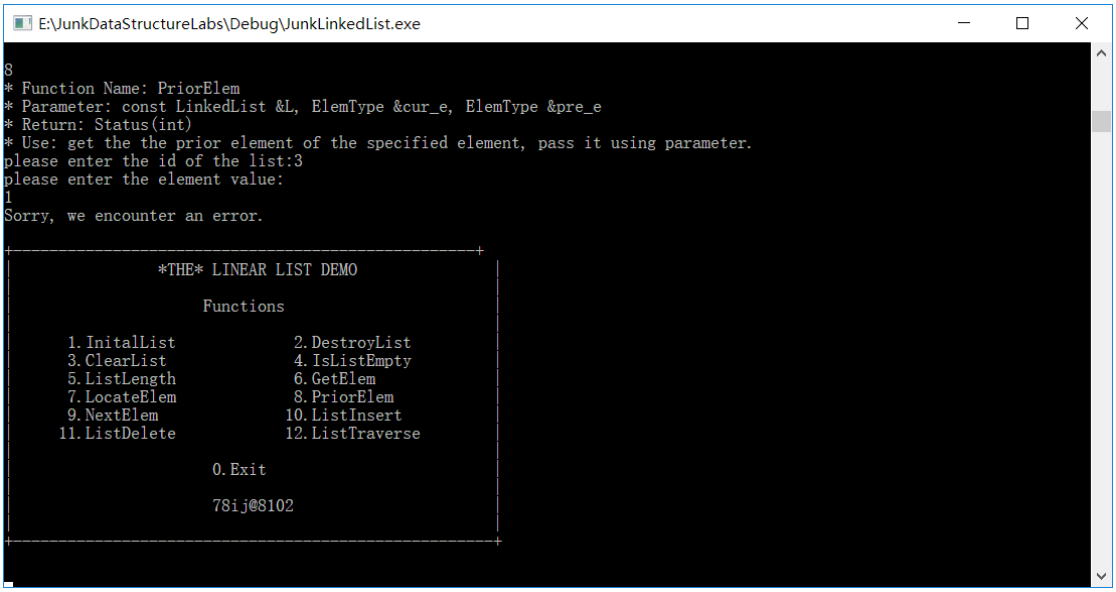

图 1-22 判断空表（2）

（13）遍历表

表 1-7 遍历表的数据及结果

操作	输入	输出	是否正确
遍历表	空表	无	正确
遍历表	6 元素表	6 个元素	正确

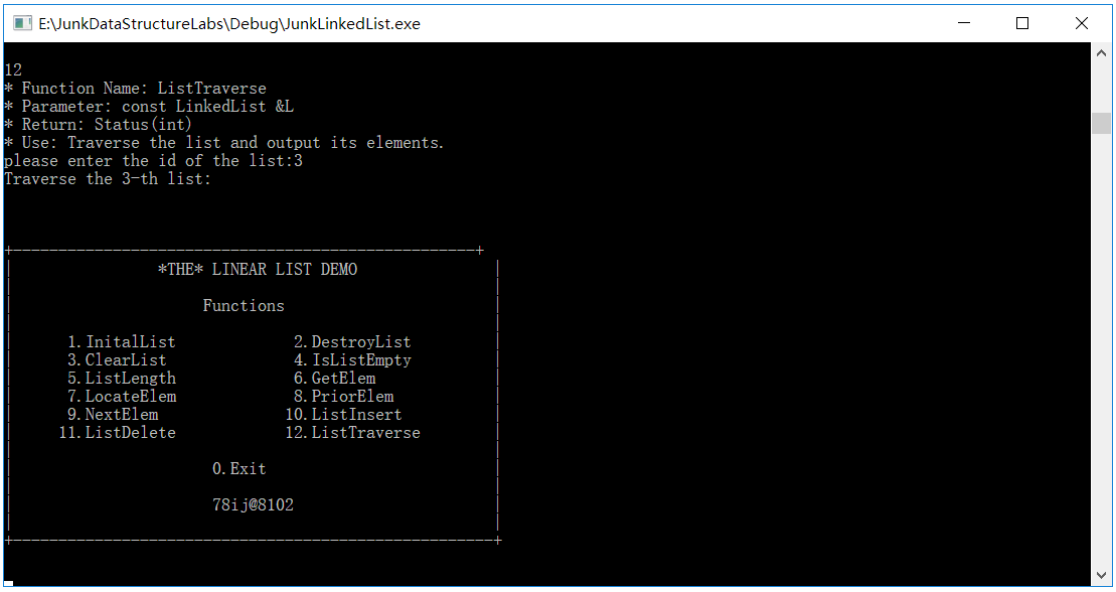

图 1-25 遍历表

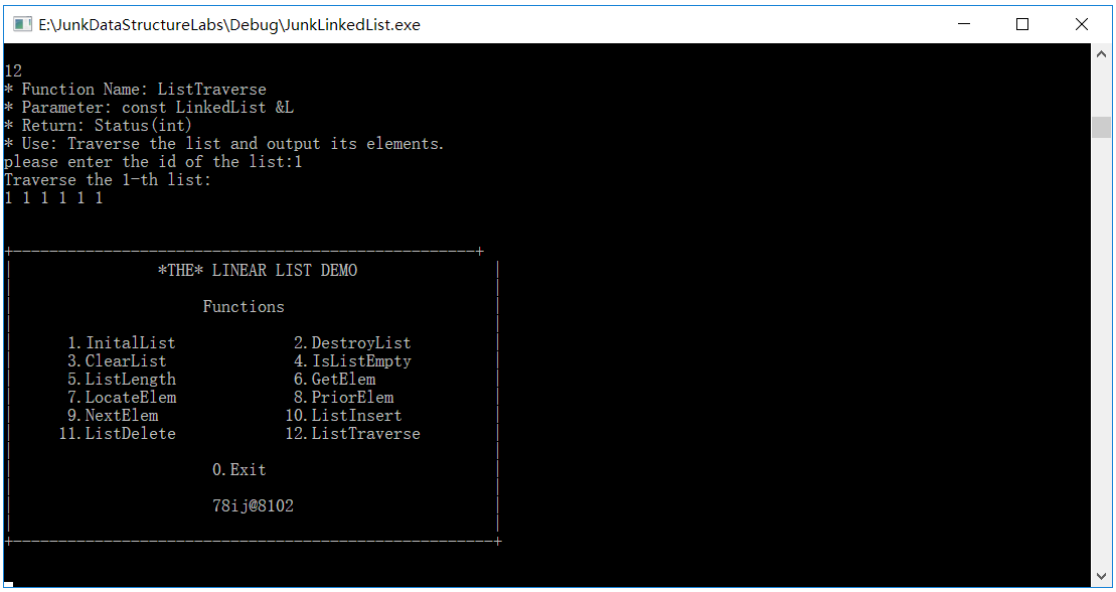

图 1-26 遍历表（2）

2.4 实验小结

本次实验各个方面都和上个实验非常相似，比如 API 接口，main 函数演示系统的结构等，不过在各个方面还是有差异，比如保存和加载数据的时候就有不小的差别，这些都是我们在编写程序过程中需要小心的方面。通过本次实验，我更了解了线性表的物理存储结构。

2.5 附录 A LinkedList.h 代码清单

```
/*
* AUTHOR: Jiamu Sun
* EMAIL: x739566858@outlook.com
* GITHUB: 78ij
*/

#include "common.h"

//带头结点的链表.
typedef struct LinkedListNode {
        ElemType data;
        LinkedListNode *next;
} LinkedListNode;

typedef struct LinkedList{
        int length;
        LinkedListNode *head;
        LinkedList *next;
        int ListID = -1;
} LinkedList;

status IntiaList(LinkedList &L);
status DestroyList(LinkedList &L);
status ClearList(LinkedList &L);
bool    ListEmpty(const LinkedList &L);
int     ListLength(const LinkedList &L);
status GetElem(const LinkedList &L, int i, ElemType &e);
int     LocateElem(const LinkedList &L, const ElemType &e); //简化过
status PriorElem(const LinkedList &L, const ElemType &cur_e, ElemType
&pre_e);
status NextElem(const LinkedList &L, const ElemType &cur_e, ElemType
&next_e);
status ListInsert(LinkedList &L, int i, ElemType &e);
status ListDelete(LinkedList &L, int i, ElemType &e);
status ListTraverse(const LinkedList &L);   //简化过
```

3 基于二叉链表的二叉树实现

3.1 问题描述

二叉树在物理内存中可以以二叉链表的方式实现，即二叉树的物理结构是链式结构，数据元素中含有两个指针域，分别指向当前节点的左儿子和右儿子。

本次实验主要完成二叉树在物理内存中用二叉链表的实现，和定义在其上的一系列算法。

实验要完成的二叉树算法：

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。

实验目标：

通过实验达到

⑺ 加深对二叉树的概念、基本运算的理解；

⑻ 熟练掌握二叉树的逻辑结构与物理结构的关系；

⑼ 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。

的目的。

3.2 系统设计

3.2.1 系统总体设计

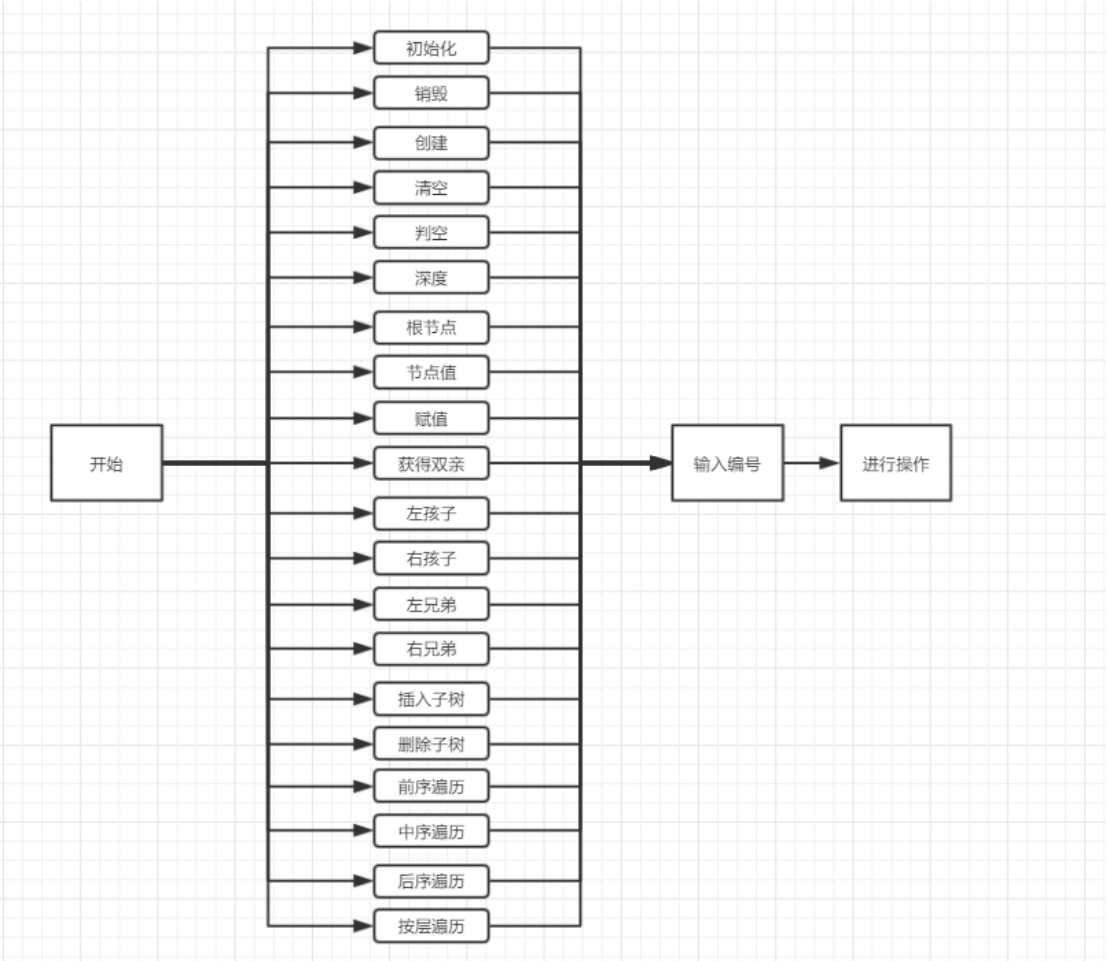

图 3-1 系统总体设计示意图

系统具有一个功能菜单。在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

系统中，树结构体中含有一个 next 指针域，指向下一个树，相当于用二叉树的方式存储了多个树。在系统运行时，通过输入一个给定的 ID 来标识不同的树，并对它们进行操作。

系统开始运行时调用函数读取文件中的数据，并提供数据保存功能以实现线性树的文件形式保存。

该演示系统提供的操作有：树的初始化、销毁、创建，清空、判空，求深度、获取根节点、任意节点、赋值、获得双亲，左右孩子、左右兄弟、插入子树、删除子树、树的前中后以及层先遍历。

在程序中实现消息处理和操作提示，包括数据的输入和输出，错误操作提示、程序的退出。

本系统采用 c++写成，但没有使用 class（因为 api 设计是面向过程样式的），但使用了常量引用一类的 c++特性。

3.2.3 二叉树算法的思想和设计

在算法设计中，函数参数的选择我参照了 c++的一些不言自明的标准：在不改变线性树内部结构，数据的情况下，我选择常量引用（const &）类型作为形参，其他时候采用引用类型。

另外，在二叉树数据结构实现过程中，数据元素定义为结构类型，由数据，索引和三个指针域组成。索引用来唯一标识二叉树中节点，必须定义为从 1 开始，递增步长为 1 的 int 类型变量。而三个指针域分别指向双亲节点，左孩子节点和右孩子节点。这些数据项在操作过程中会被维护。具体的定义参看附录。

（1）InitBiTree(BiTree &T)

设计：分配存储空间，并初始化树根为 NULL。

操作结果：构造一个空的二叉树。

时间空间复杂度：O(1)

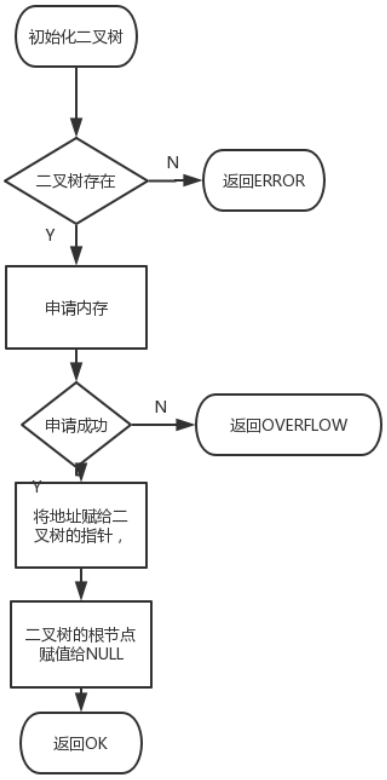

图 3-2 初始化树的流程图

（2）DestroyBiTree(BiTree &T)

设计：释放存储空间，每次操作当前二叉树

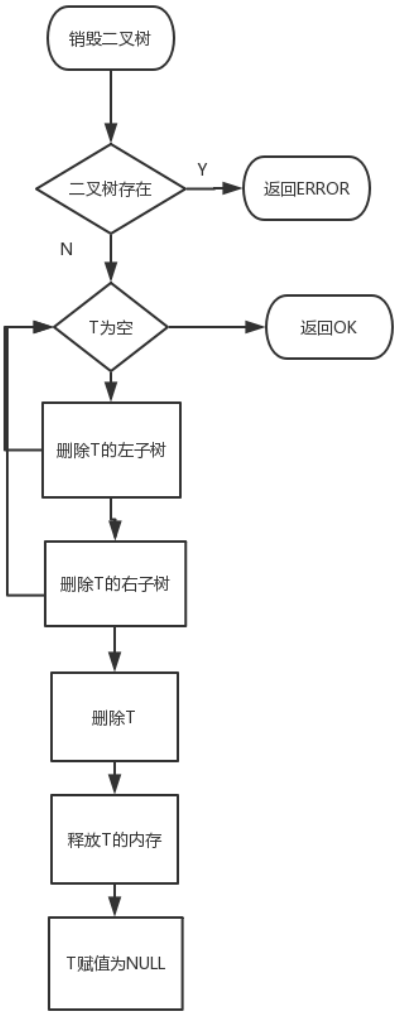

图 3-3 销毁二叉树的流程图

操作结果：销毁二叉树。

时间复杂度：O(n)

空间复杂度：O(1)

（3）CreateBiTree(BiTree &T, int length, int *preorder, int *inorder, ElemType * data)

设计：输入先序和中序遍历二叉树的索引以及中序遍历二叉树的数据，简历二叉树

操作结果：根据定义建立二叉树

时间复杂度：O(n)

空间复杂度:O(n)

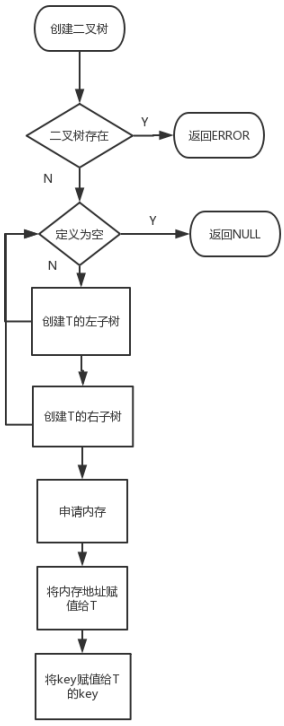

图 3-4 创建二叉树的流程图

（4）ClearBiTree(BiTree &T)

设计：递归释放二叉树节点，置指针为 NULL

操作结果：清空二叉树

时间复杂度：O(n)

空间复杂度：O(1)

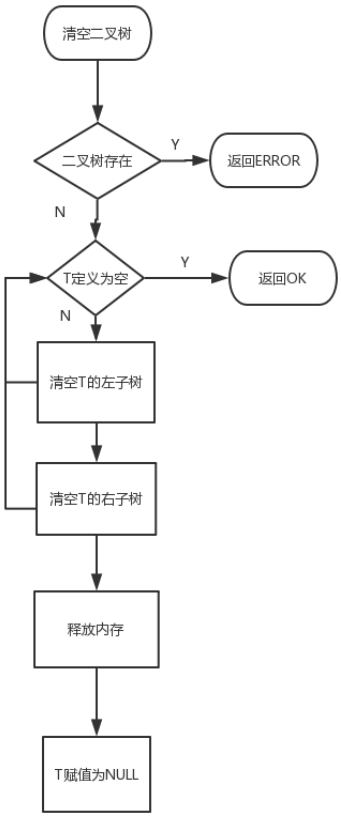

图 3-5 清空二叉树的流程图

（5）BiTreeEmpty(const BiTree &T)

设计：判断二叉树是否为空

操作结果：为空返回 true，否则返回 false

时间空间复杂度：O(1)

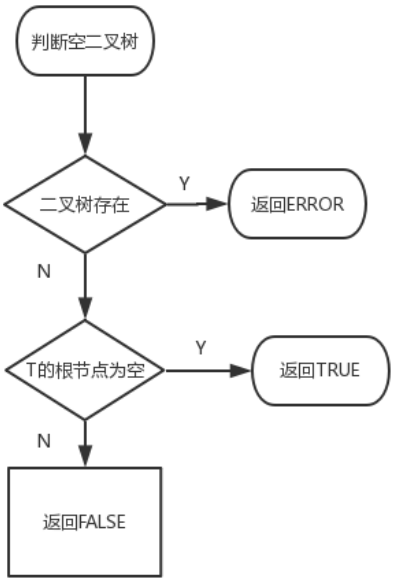

图 3-6 判断空树的流程图

（6）BiTreeDepth(const BiTree &T)

设计：递归的执行，返回二叉树的深度

操作结果：返回二叉树深度

时间复杂度：O(n)

空间复杂度：O(1)

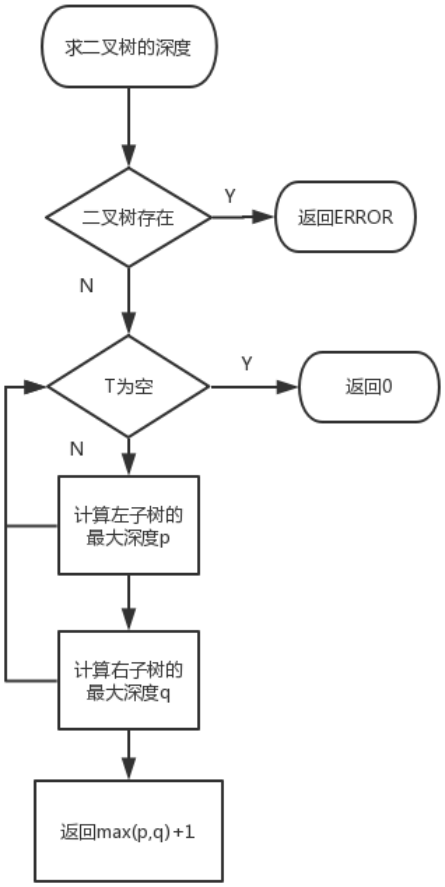

图 3-7 得到元素的流程图

（7）Root(const BiTree &T)

设计：返回根节点的值

操作结果：返回根节点的值

时间复杂度：O(1)

空间复杂度：O(1)

（8）Value(const BiTree &T,int index, ElemType &value)

设计：遍历二叉树找到节点的 index 等于形参中 index 的节点，并返回它的值。

操作结果：若 index 存在，返回其节点的值。如果不存在，返回 ERROR。

时间复杂度：O(n)

空间复杂度：O(1)

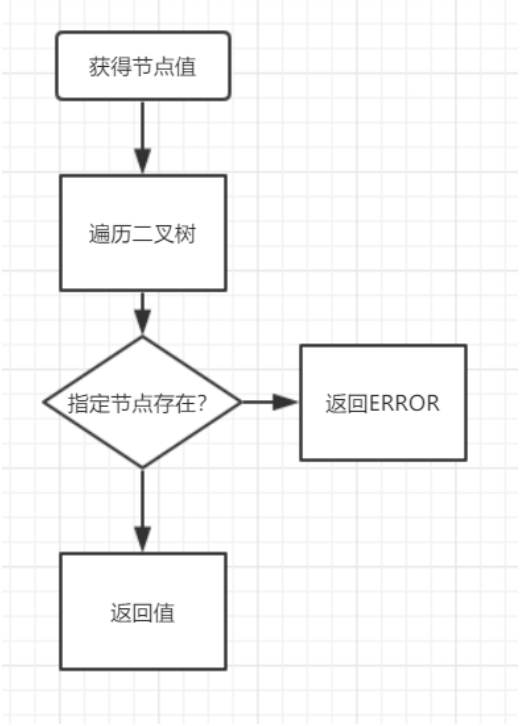

图 3-8 获得节点值的流程图

（9）Assign(BiTree &T,int index, ElemType &value)

设计：遍历二叉树找到节点的索引等于 index 的节点，并将传入的 value 赋给这个节点。

操作结果：若 index 存在，将 value 赋给这个节点，如果 index 不存在，返回 ERROR。

时间复杂度：O(n)

空间复杂度：O(1)

（10）Parent(const BiTree &T, int index,ElemType &value)

设计：找到指定节点的双亲节点，并返回其值。

操作结果：若指定节点的双亲节点存在，返回其 value，若节点或者其双亲节点不存在，返回 ERROR。

时间复杂度：O(n)

空间复杂度：O(1)

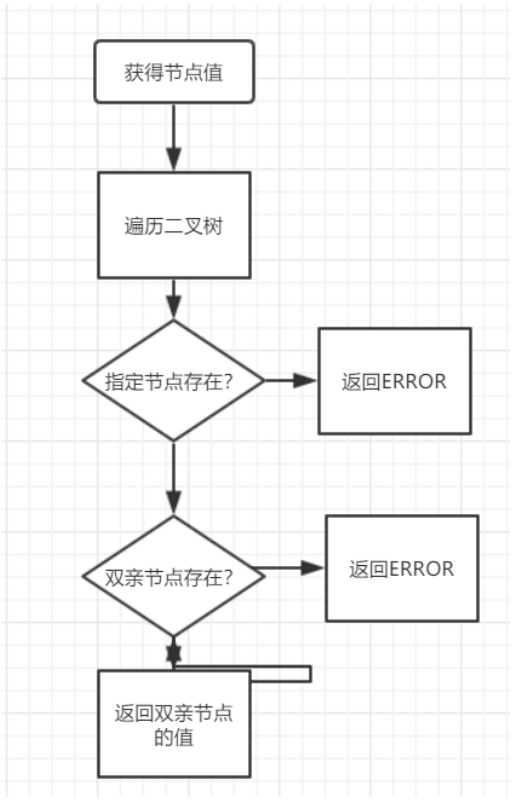

图 3-9 获得双亲节点的流程图

（11）LeftChild(const BiTree &T, int index);

设计：获取指定 index 节点的左孩子

操作结果：若左孩子存在，返回其值，否则返回 ERROR。

时间复杂度：O(n)

空间复杂度：O(1)

（12）RightChild(const BiTree &T, int index);

设计：获取指定 index 节点的右孩子

操作结果：若右孩子存在，返回其值，否则返回 ERROR。

时间复杂度：O(n)

空间复杂度：O(1)

（13）LeftSibling(const BiTree &T, int index);

设计：获取指定 index 节点的左兄弟

操作结果：若左兄弟存在，返回其值，否则返回 ERROR。

时间复杂度：O(n)

空间复杂度：O(1)

（14）RightSibling(const BiTree &T, int index);

设计：获取指定 index 节点的右兄弟

操作结果：若右兄弟存在，返回其值，否则返回 ERROR。

时间复杂度：O(n)

空间复杂度：O(1)

（15）InsertChild(BiTree &T, int index, int LR, BiTree &c)

设计：检查插入的树 c 的根的右孩子是否为空。如果不为空，返回 ERROR。否则将 c 的根节点连接到 T 指定节点的左或者右子树，T 的指定节点原有左或右子树则插入到 c 的根节点的右边。

操作结果：如果操作可行，则 c 被插入到 T 的指定节点的左或者右子树，T 的原有左或者右子树插入到 c 的根节点的右边。

时间复杂度：O(n)

空间复杂度：O(1)

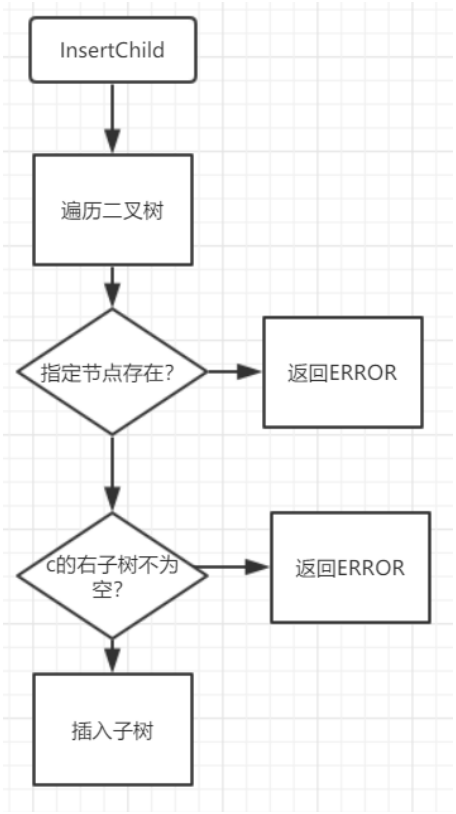

图 3-10 插入子树的流程图

（15）DeleteChild(BiTree &T, int index, int LR)

设计：查找 index 对应的节点。如果存在，删除其左子树或者右子树。

操作结果：如果操作可行，删除对应节点的左或右子树。

时间复杂度：O(n)

空间复杂度：O(1)

（17）PreOrderTraverse(const BiTree &T)

操作结果：前序遍历树。

时间复杂度：O(n)

空间复杂度：O(1)

（18）InOrderTraverse(const BiTree &T)

操作结果：中序遍历树。

时间复杂度：O(n)

空间复杂度：O(1)

（19）PostOrderTraverse(const BiTree &T)

操作结果：后序遍历树。

时间复杂度：O(n)

空间复杂度：O(1)

（20）LevelOrderTraverse(const BiTree &T)

操作结果：层序遍历树。

时间复杂度：O(n)

空间复杂度：O(1)

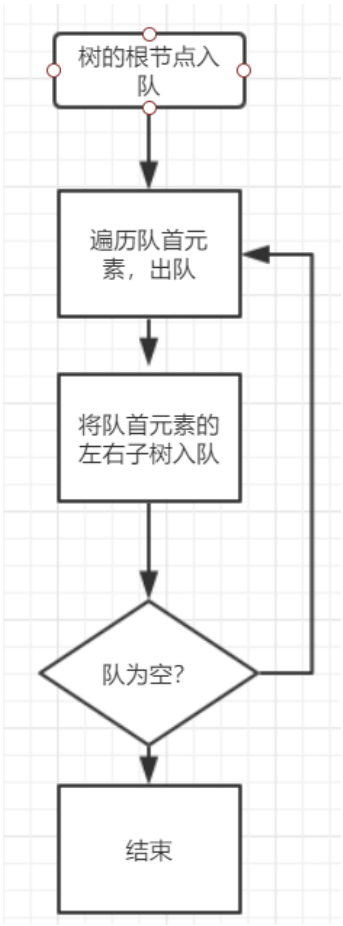

图 3-11 层序遍历树的流程图

（21）SaveData()

设计：本函数将所有的二叉树保存到文件中。每个二叉树保存 index 的前序和中序遍历以及 data 的中序遍历。

操作结果：所有的二叉树保存到文件中。

时间复杂度：O(n)

空间复杂度：O(n)

(22)LoadData()

设计：本函数将所有的二叉树从文件中读取并重建在内存中

操作结果：所有的二叉树被读取并重建。

时间复杂度：O(n)

空间复杂度：O(n)

3.3 二叉树系统测试

3.3.1 二叉树演示系统实现说明

本演示系统包括一个循环，每次循环开始打印出演示菜单，菜单如图所示

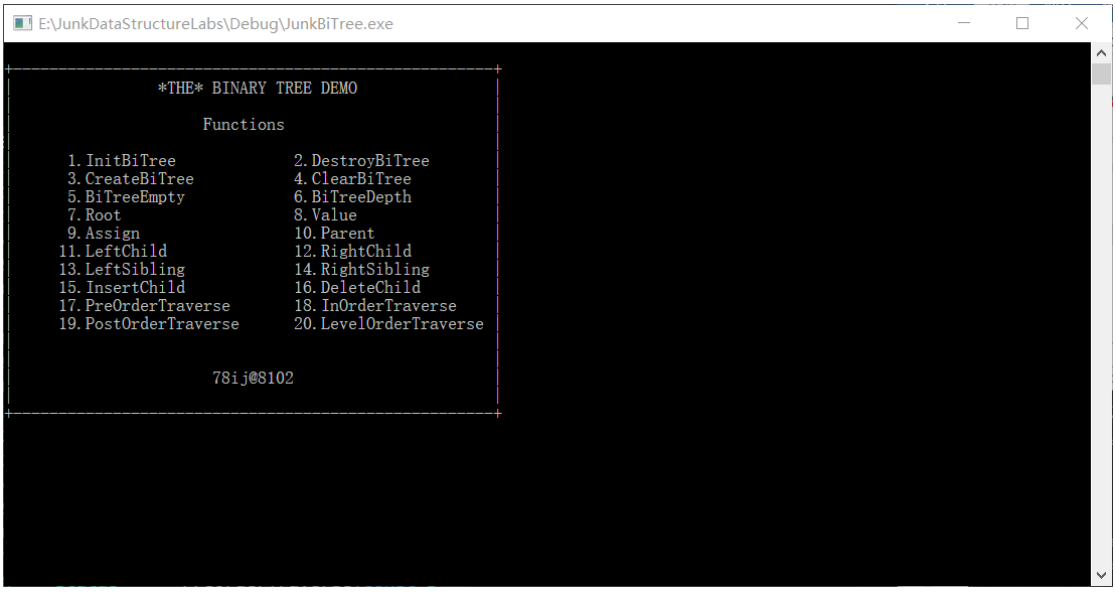

图 3-12 演示系统菜单

每次操作时要求输入本次操作二叉树的编号：

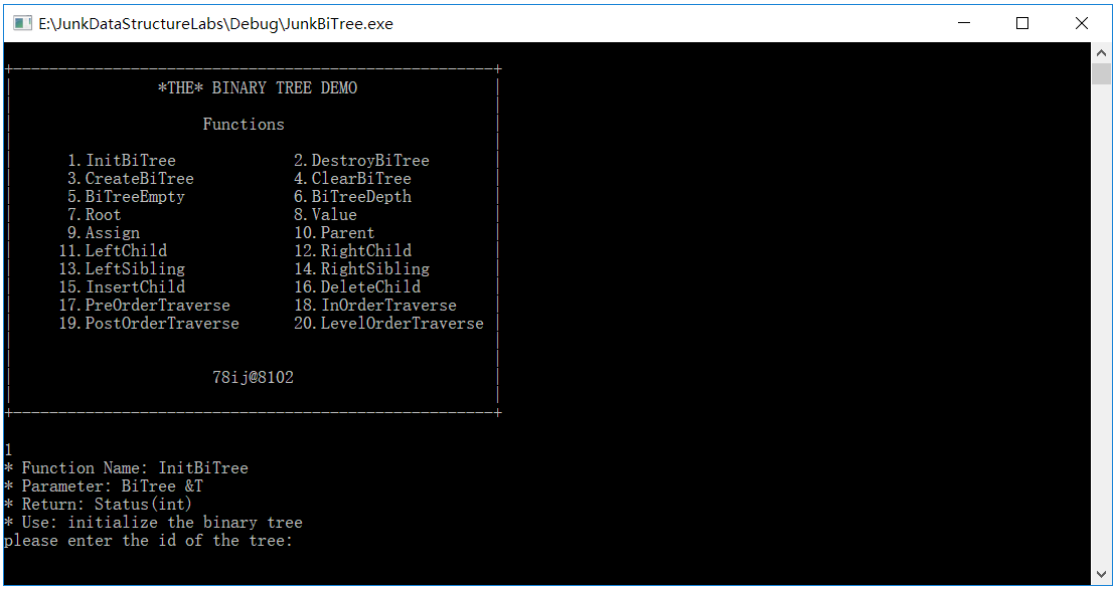

图 3-13 演示系统要求输入编号

每次打印出菜单之前，从文件中将所有二叉树读取到内存中，每次进行完毕一个操作之后，将内存中的二叉树都存到文件之中。

3.3.2 系统测试

下面，选取几个具有代表性的函数进行测试。

（14）初始化二叉树

表 3-1 初始化树的数据及结果

操作	输入	输出	是否正确
初始化树	无	创建成功	正确

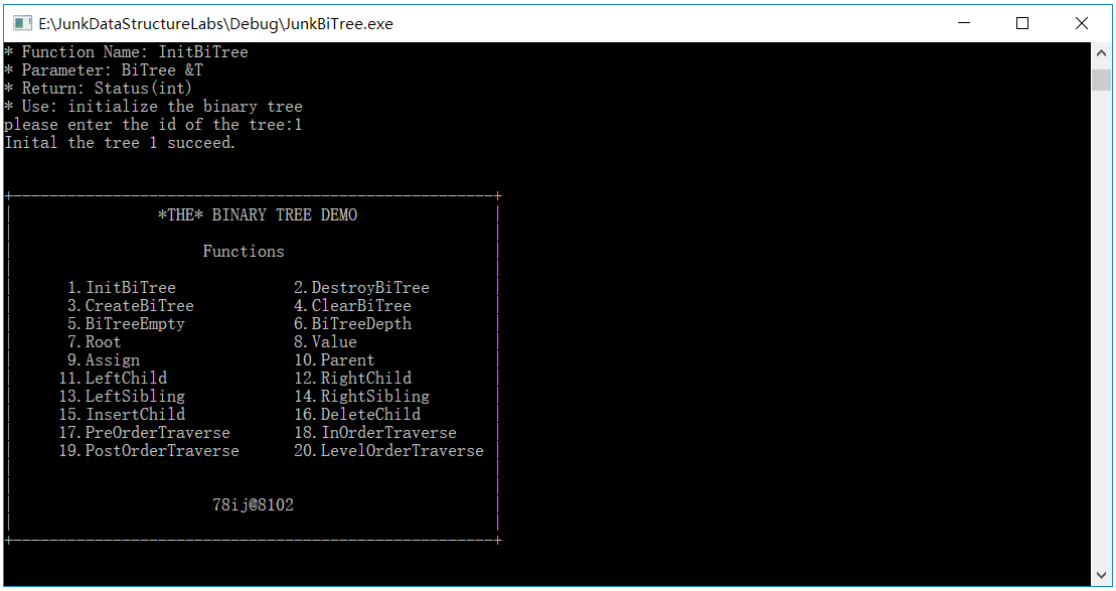

图 3-14 初始化树

（15）销毁

表 3-2 销毁树的数据及结果

操作	输入	输出	是否正确
销毁树	树存在	销毁成功	正确
销毁树	树不存在	销毁失败	正确

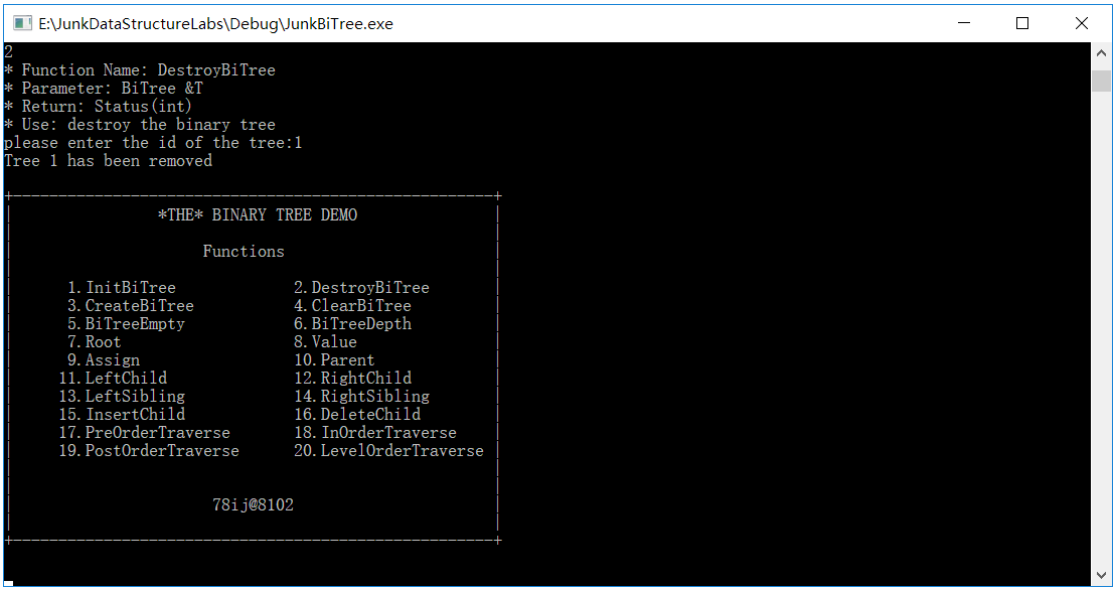

图 3-15 销毁树

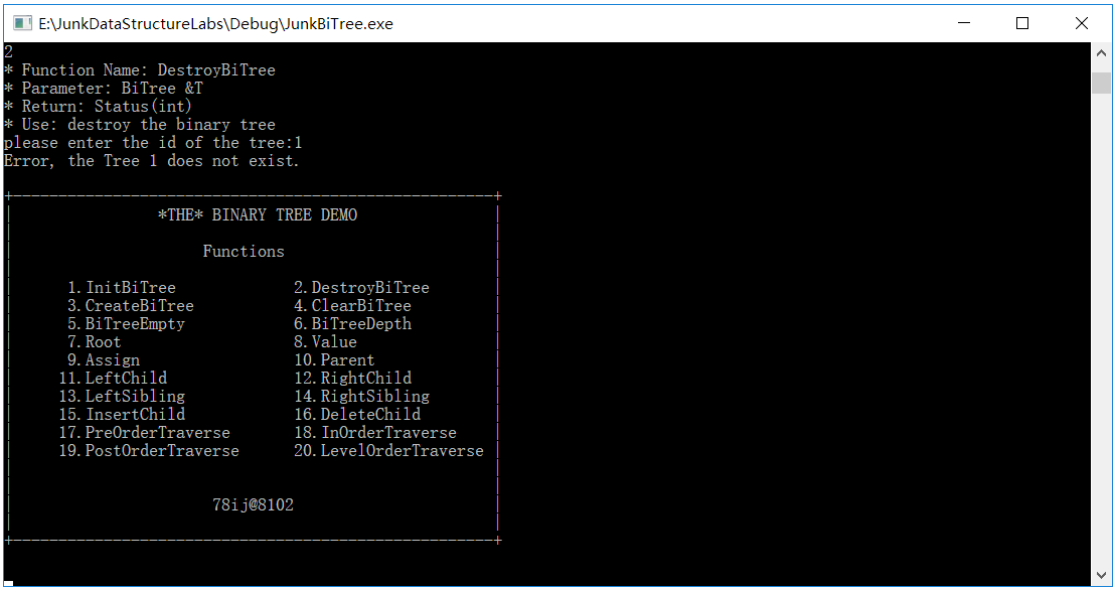

图 3-16 销毁树（2）

（16）建树

表 1-3 建树的数据及结果

操作	输入	输出	是否正确
建树	长为 5 的树	建立成功	正确
建树	输入不合法	建立失败	正确

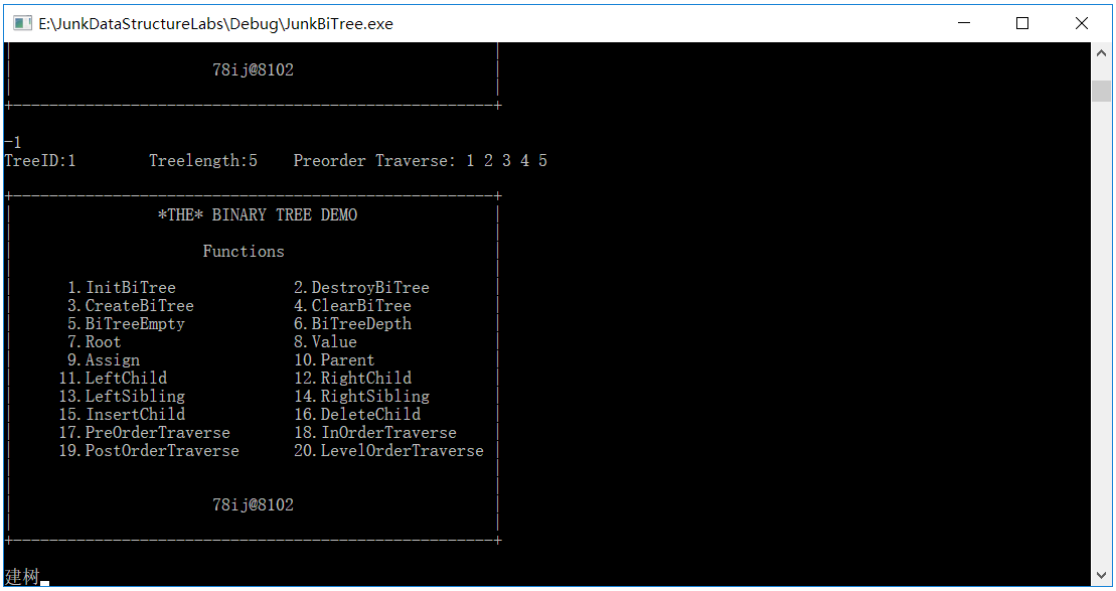

图 3-18 建树

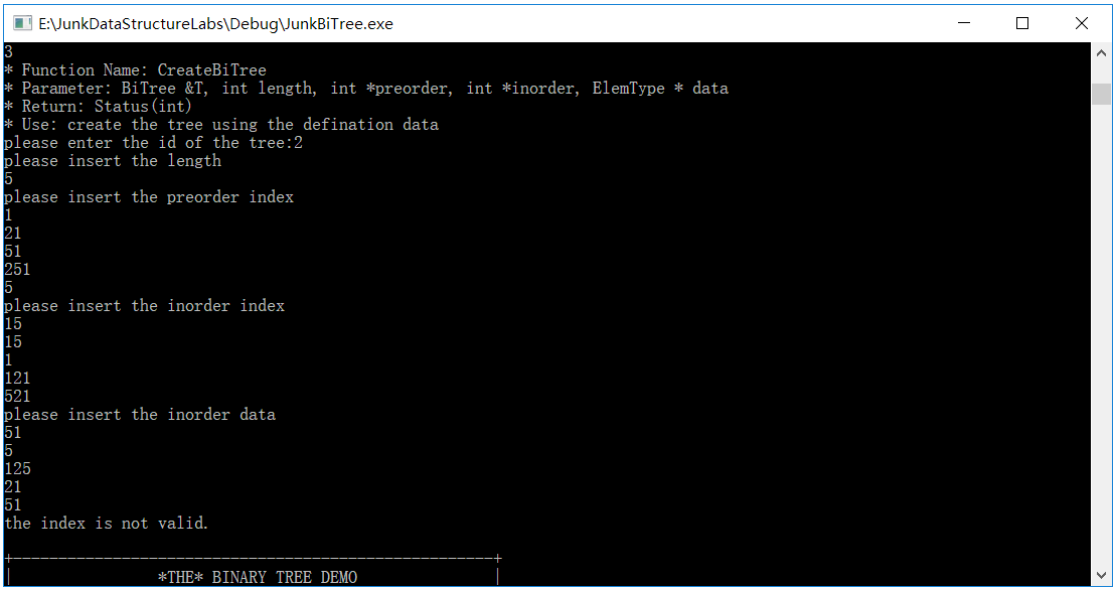

图 3-19 建树（2）

（17）求二叉树深度

表 3-4 插入元素的数据及结果

操作	输入	输出	是否正确
求深度	空树	0	正确
求深度	深度 5	5	正确

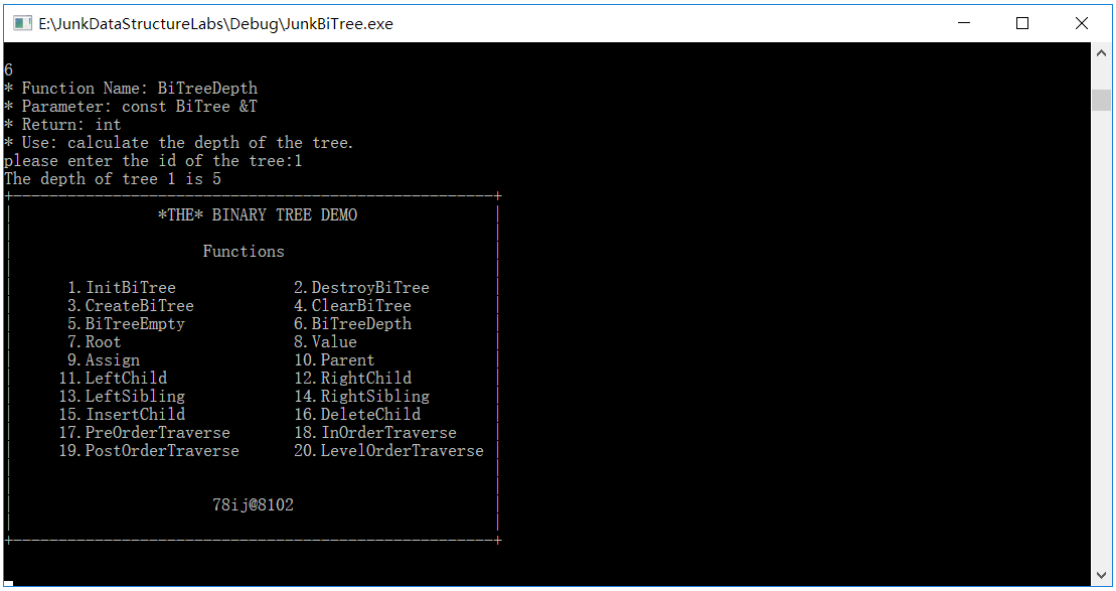

图 3-20 求深度

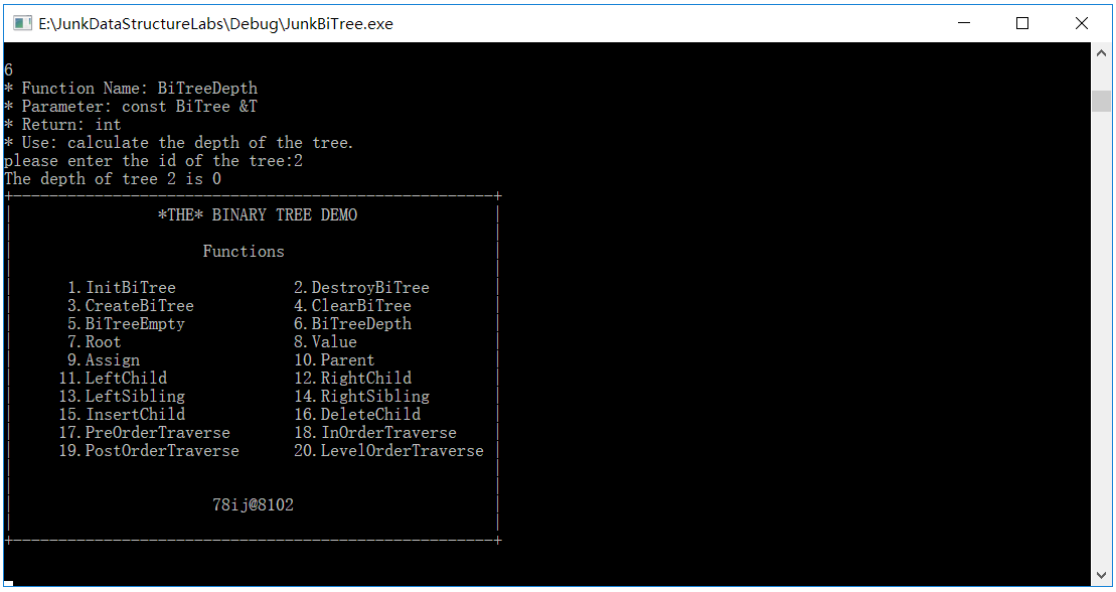

图 3-21 求深度（2）

（18）求双亲节点

表 3-5 求双亲节点的数据及结果

操作	输入	输出	是否正确
求双亲节点	双亲节点为 4	4	正确
求双亲节点	双亲节点不存在	Error	正确

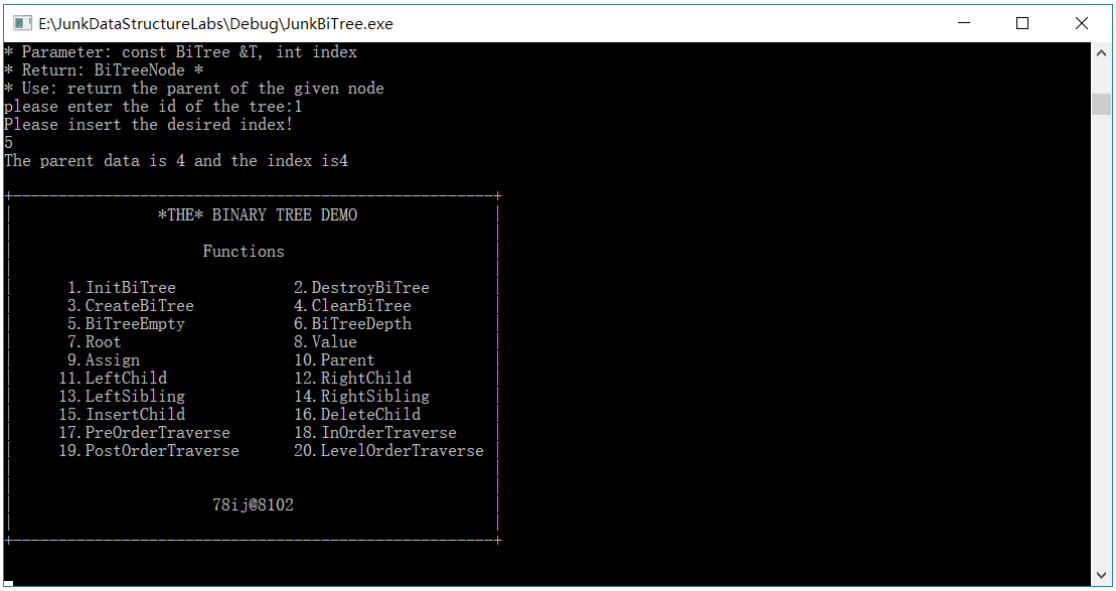

图 3-22 双亲节点

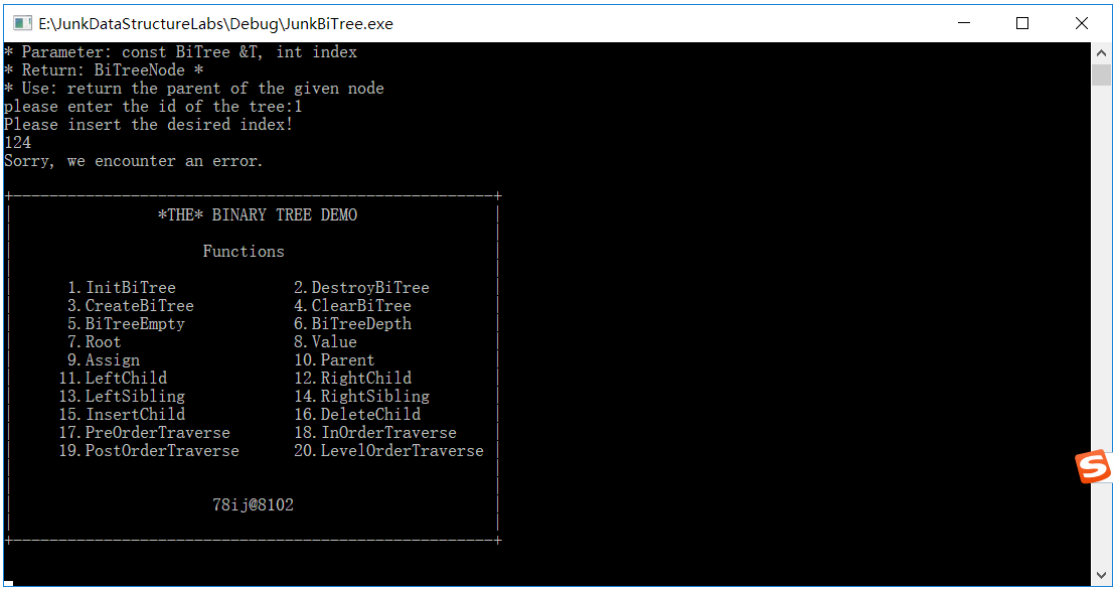

图 3-23 双亲节点（2）

（19）删除子树

表 3-7 删除子树的数据及结果

操作	输入	输出	是否正确
删除子树	节点不存在	错误	正确
删除子树	5 元素，删除 2 节点	3 元素树	正确

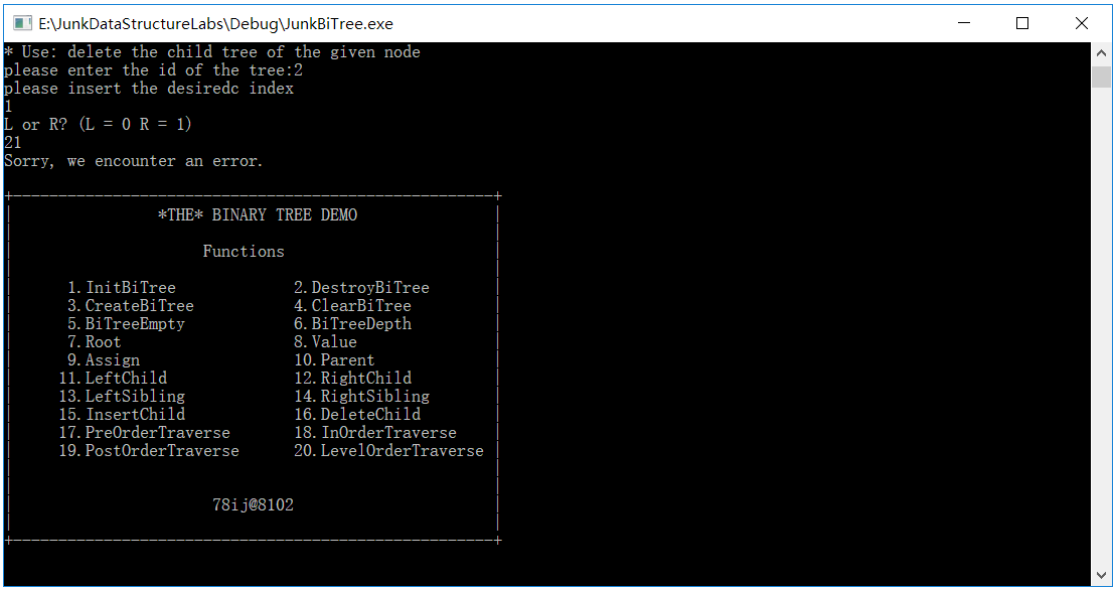

图 3-24 删除子树

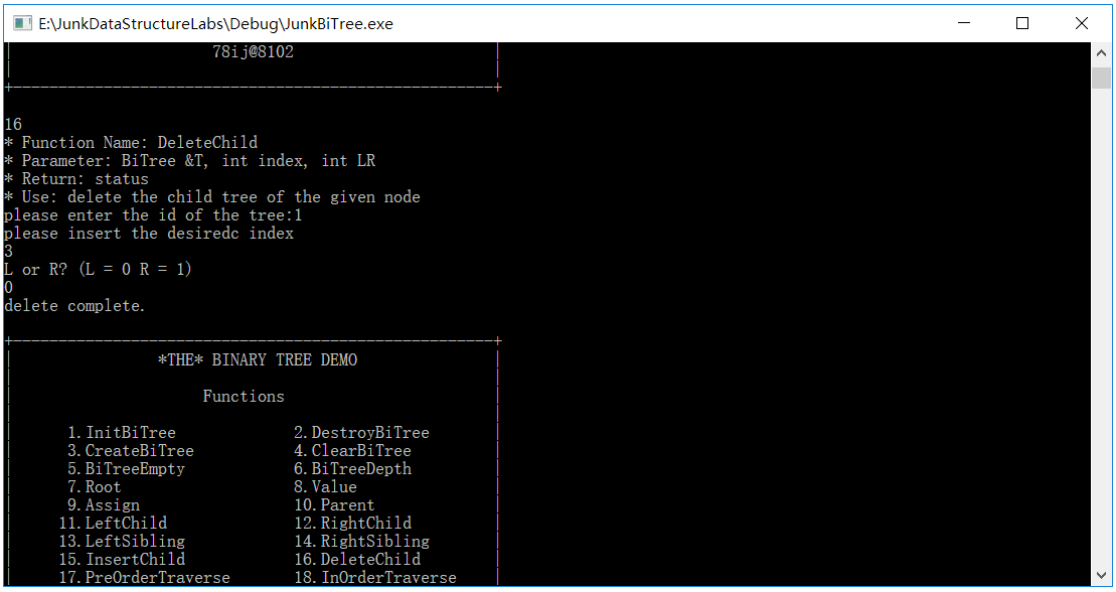

图 3-25 删除子树(2)

3.5 实验小结

本次实验可以说是四次实验中，代码量最多，实现起来最麻烦的一次实验了。一时因为树的数据结构本身是比较难实现的，尤其是二叉链表，涉及很多递归操作，有时候不好理解，第二是因为 API 比较多，做的重复无用功比较多。但是在这个过程中我学会了代码比较多时的开发方法，积累了很多经验。

3.6 附录 A BiTree.h 代码清单

```
/*
* AUTHOR: Jiamu Sun
* EMAIL: x739566858@outlook.com
* GITHUB: 78ij
*/

#include "common.h"


typedef struct BiTreeNode {
        ElemType data;
        int index; // used to mark the node
        BiTreeNode *parent = NULL;
        BiTreeNode *left = NULL;
        BiTreeNode *right = NULL;
}BiTreeNode;

typedef struct BiTree {
        int TreeID = -1;
        int length = 0;
        BiTreeNode *root;
        BiTree *next;
}BiTree;

enum TraverseMethod {
        PRE, IN, POST, LEVEL
};

//APIs
bool            isvalid(int *pre, int *in, int length);
void        write(BiTreeNode *root, TraverseMethod method, bool isindex,
FILE *fp);
status          InitBiTree(BiTree &T);
status          DestroyBiTree(BiTree &T);
status      CreateBiTree(BiTree &T, int length, int *preorder, int
*inorder, ElemType * data);
status          ClearBiTree(BiTree &T);
bool            BiTreeEmpty(const BiTree &T);
int                     BiTreeDepth(const BiTree &T);
BiTreeNode *Root(const BiTree &T);
status          Value(const BiTree &T, int index, ElemType &value);
```

```
status          Assign(BiTree &T, int index,ElemType &value);
BiTreeNode *Parent(const BiTree &T, int index);
BiTreeNode *LeftChild(const BiTree &T, int index);
BiTreeNode *RightChild(const BiTree &T, int index);
BiTreeNode *LeftSibling(const BiTree &T, int index);
BiTreeNode *RightSibling(const BiTree &T, int index);
status      InsertChild(BiTree &T, int index, int LR, BiTree &c);
status      DeleteChild(BiTree &T, int index, int LR);
status      PreOrderTraverse(const BiTree &T);
status      InOrderTraverse(const BiTree &T);
status      PostOrderTraverse(const BiTree &T);
status          LevelOrderTraverse(const BiTree &T);
```

4 基于邻接表的图实现

4.1 问题描述

有向无权图在内存中可以以邻接表方式实现。即定义一个节点数组，每个数组数据元素指向一个链表头结点，这个链表保存此节点连接的所有节点的索引。

本次实验主要完成有向无权图在物理内存中用邻接表的实现，和定义在其上的一系列算法。

实验要完成的图算法：

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。

实验目标：

⑴ 加深对图的概念、基本运算的理解；

⑵ 熟练掌握有向无权图的逻辑结构与物理结构的关系；

⑶ 以邻接表作为物理结构，熟练掌握图基本运算的实现。

4.2 系统设计

4.2.1 系统总体设计

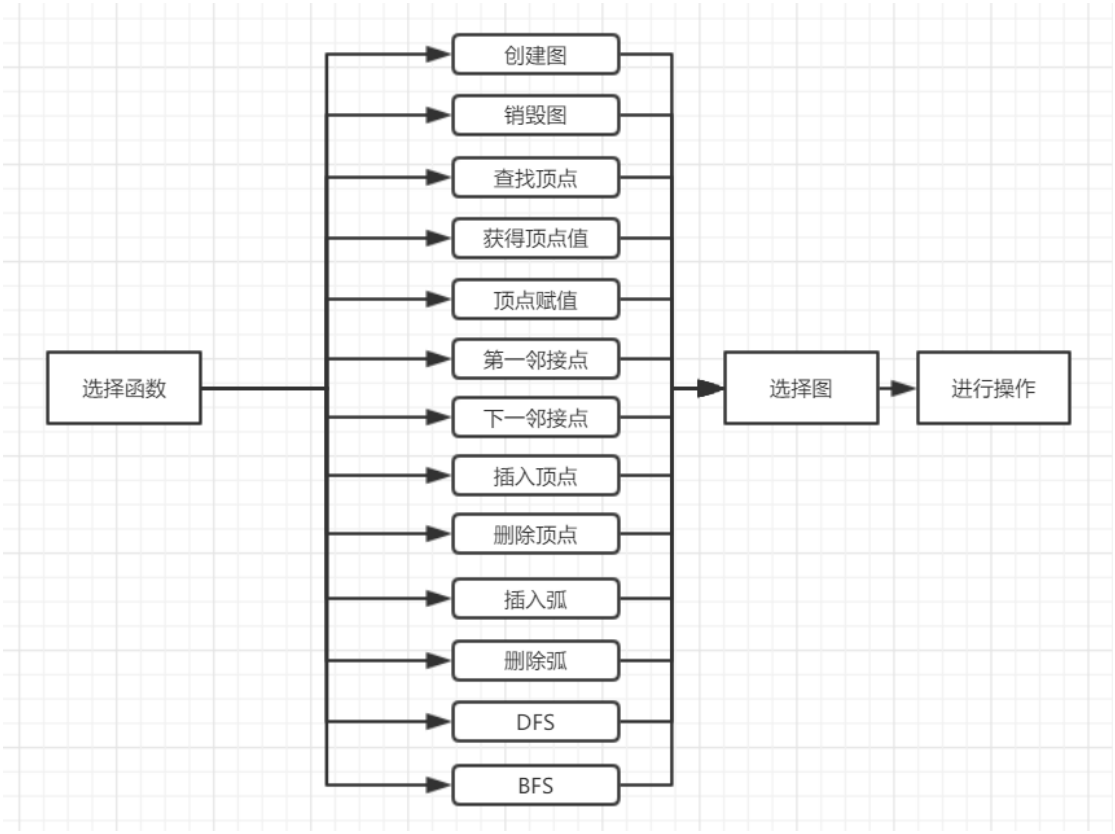

图 4-1 系统总体结构示意图

系统具有一个功能菜单。在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

系统中，图结构体中含有一个 next 指针域，指向下一个图，相当于用链表的方式存储了多个图。在系统运行时，通过输入一个给定的 ID 来标识不同的树，并对它们进行操作。

系统开始运行时调用函数读取文件中的数据，并提供数据保存功能以实现有向图形式保存。

该演示系统提供的操作有：图的创建，销毁，查找顶点，获得值，赋值，第一邻接点，下一邻接点，插入，删除顶点，插入，删除弧，DFS，BFS。

在程序中实现消息处理和操作提示，包括数据的输入和输出，错误操作提示、程序的退出。

本系统采用 c++写成，但没有使用 class（因为 api 设计是面向过程样式

的），但使用了常量引用一类的 c++特性。

4.2.3 二叉树算法的思想和设计

在算法设计中，函数参数的选择我参照了 c++的一些不言自明的标准：在不改变图内部结构，数据的情况下，我选择常量引用（const &）类型作为形参，其他时候采用引用类型。

另外，在图数据结构实现过程中，数据元素定义为结构类型，由数据，索引和指针域组成。索引用来唯一标识节点，必须定义为从 0 开始，递增步长为 1 的 int 类型变量。而指针域分别指向第一个邻接点或者下一个邻接点。这些数据项在操作过程中会被维护。具体的定义参看附录。

（1）CreateGraph(Graph &G, ElemType *Nodedata, int *matrix, int length)

设计：接受 ElemType 类型指针和 int 类型指针，nodedata 参数是每个节点中存储的数据值，matrix 是图的邻接矩阵，其中 1 代表有边，-1 代表无边。

结果：建立一个按照参数确定的图。

时间复杂度：O(n^2)

空间复杂度：O(n+e)

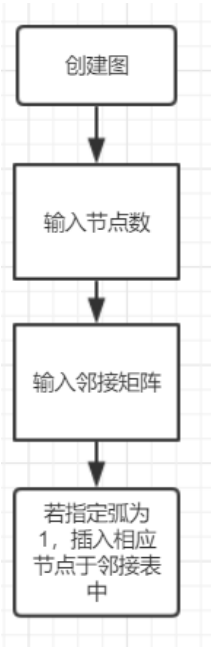

图 4-2 创建图的示意图

（2）DestroyGraph(Graph &G)

设计：销毁图 G

结果：销毁图 G

时间复杂度：O(n+e)

空间复杂度：O(1)

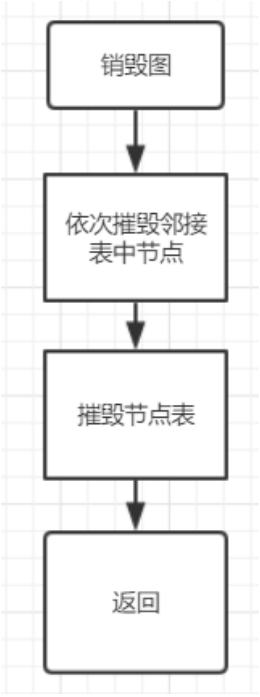

图 4-3 销毁图的示意图

（3）LocateVex(Graph &G, int index)

设计：在节点表中顺序查找索引等于 index 的节点，并且打印其 data 值，和其邻接的所有节点。

结果：打印出指定节点的 data 值和邻接节点，否则返回 error。

时间复杂度：O(n)

空间复杂度：O(1)

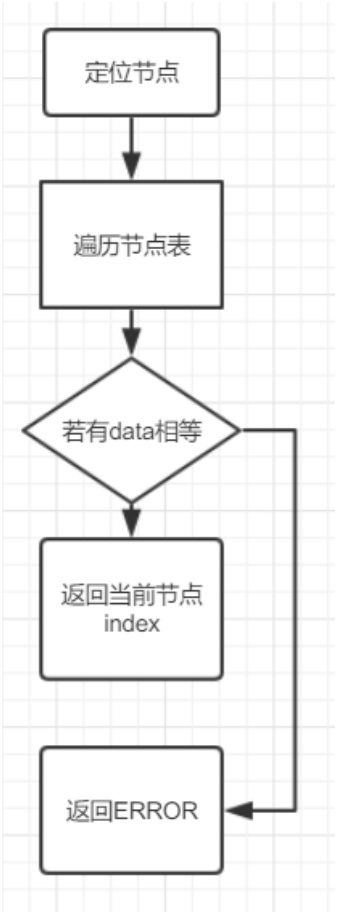

图 4-4 定位节点的示意图

（4）GetVex(Graph &G, int index)

设计：在节点表中顺序查找索引等于 index 的节点，并且打印其 data 值。

结果：打印出指定节点的 data 值。

时间复杂度：O(n)

空间复杂度：O(1)

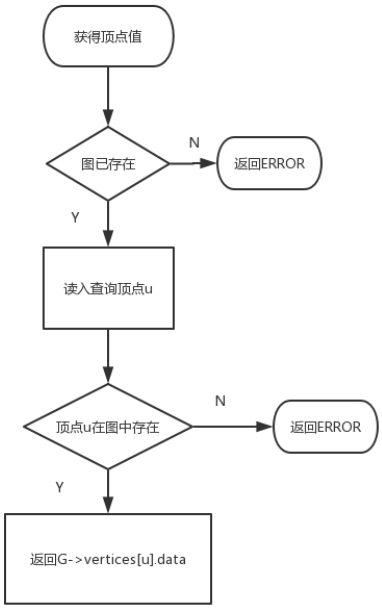

图 4-5 获得顶点的示意图

（5）PutVex(Graph &G, int index，ElemType &data)

设计：在节点表中顺序查找索引等于 index 的节点，将其节点值赋值为 data。

结果：赋值指定节点 data 值为 data。

时间复杂度：O(n)

空间复杂度：O(1)

（6）FirstAdjVex(Graph &G, int index)

设计：在节点表中顺序查找索引等于 index 的节点，并返回其第一个邻接节点的 index。

结果：返回指定节点的第一个邻接节点的 index

时间复杂度：O(n)

空间复杂度：O(1)

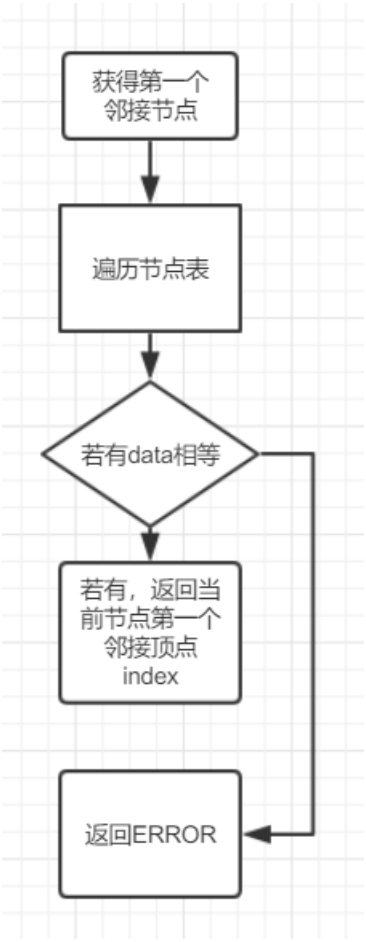

图 4-6 获得第一邻接顶点的示意图

（7）NextAdjVex(Graph &G, int index，int &adj)

设计：在节点表中顺序查找索引等于 index 的节点，查找其有没有邻接节点索引为 adj 的节点。若有，则返回其下一个邻接节点。

结果：返回指定节点的下一个邻接节点的 index。

时间复杂度：O(n)

空间复杂度：O(1)

（8）InsertVex(Graph &G, ElemType data)

设计：节点表中插入一个值为 data 的节点。

结果：插入节点

时间复杂度：O(1)

空间复杂度：O(1)

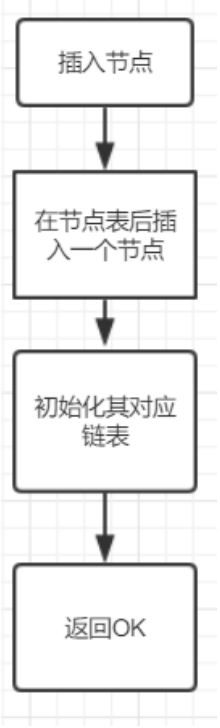

图 4-7 插入顶点的示意图

（9）DeleteVex(Graph &G, int index)

设计：查找是否有索引为 index 的节点。若有，则删除此节点和所有其关联的边，并将 index 在其后的所有节点的 index 减一。

结果：指定节点和所有关连边被删除。

时间复杂度：O(n)

空间复杂度：O(1)

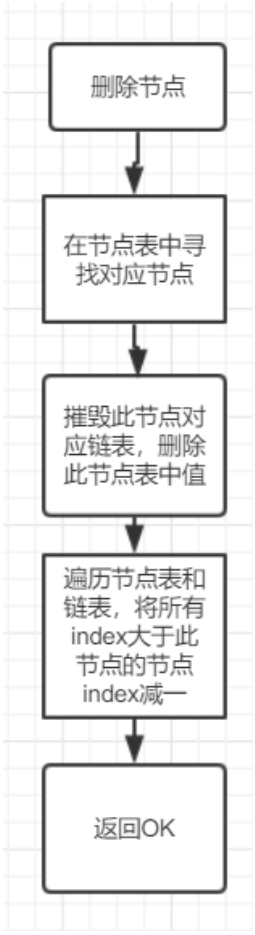

图 4-8 删除顶点的示意图

（10）InsertArc(Graph &G, int v, int w)

设计：在 u 的邻接表中插入 w。

结果：插入一个弧尾为 v，弧头为 w 的弧。

时间复杂度：O(1)

空间复杂度：O(1)

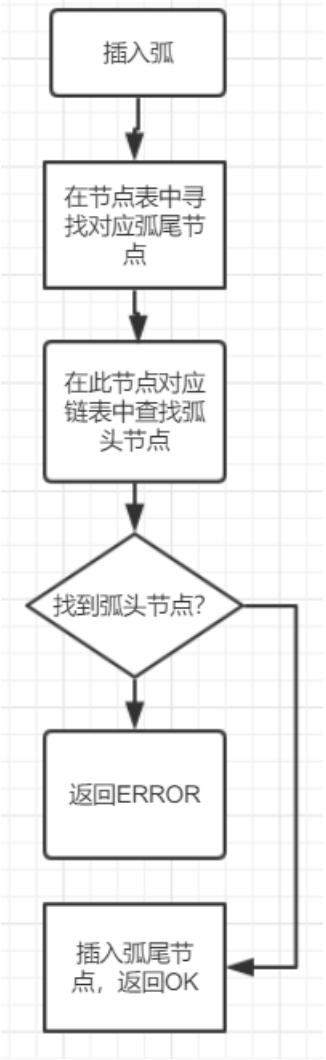

图 4-9 插入弧的示意图

（11）DeleteArc(Graph &G, int v, int w)

设计：查找是否有弧(v,w)如果有，在 v 的邻接表中删除 w。

结果：指定弧被删除。

时间复杂度：O(e)

空间复杂度：O(1)

（12）DFSTraverse(Graph &G)

设计：使用一个栈来遍历。外层循环将未遍历节点入栈，此后深度优先访问时先将这个栈顶结点的第一个邻接点入栈，并继续循环，直到栈空为止。

结果：深度优先遍历图。

时间复杂度：O(n+e)

空间复杂度：O(1)

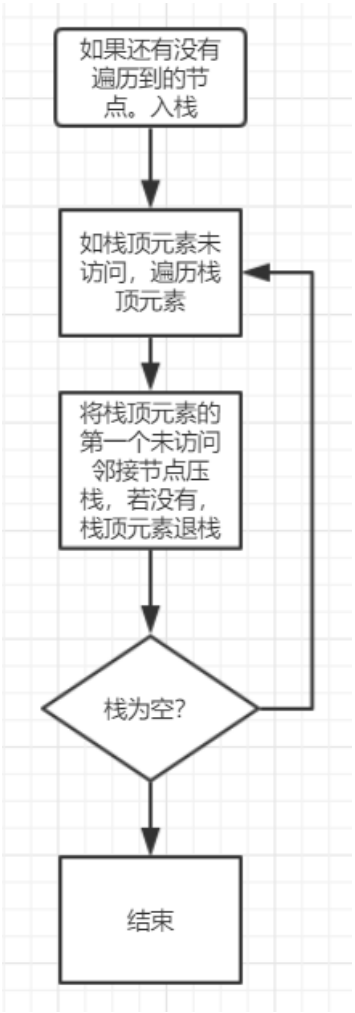

图 4-10 DFS 遍历的示意图

（13）BFSTraverse(Graph &G)

设计：使用一个队列来遍历。外层循环将未遍历节点入队，此后广度优先访问时将队首节点所有的邻接点入队，并将此节点出队，直到队空为止。

结果：广度优先遍历图。

时间复杂度：O(n+e)

空间复杂度：O(1)

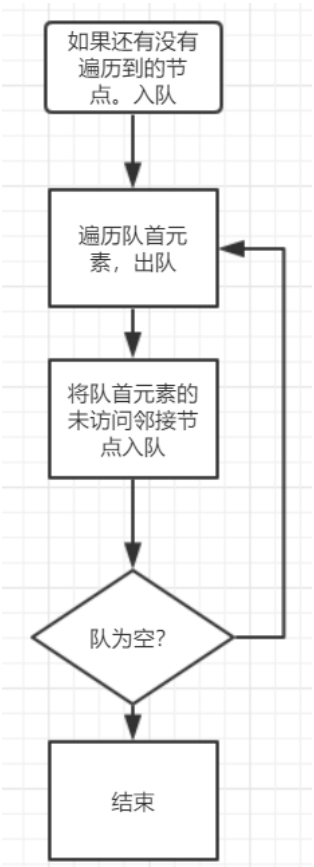

图 4-11 BFS 遍历的示意图

4.3 有向图系统测试

4.3.1 演示系统实现说明

本演示系统包括一个循环，每次循环开始打印出演示菜单，菜单如图所示

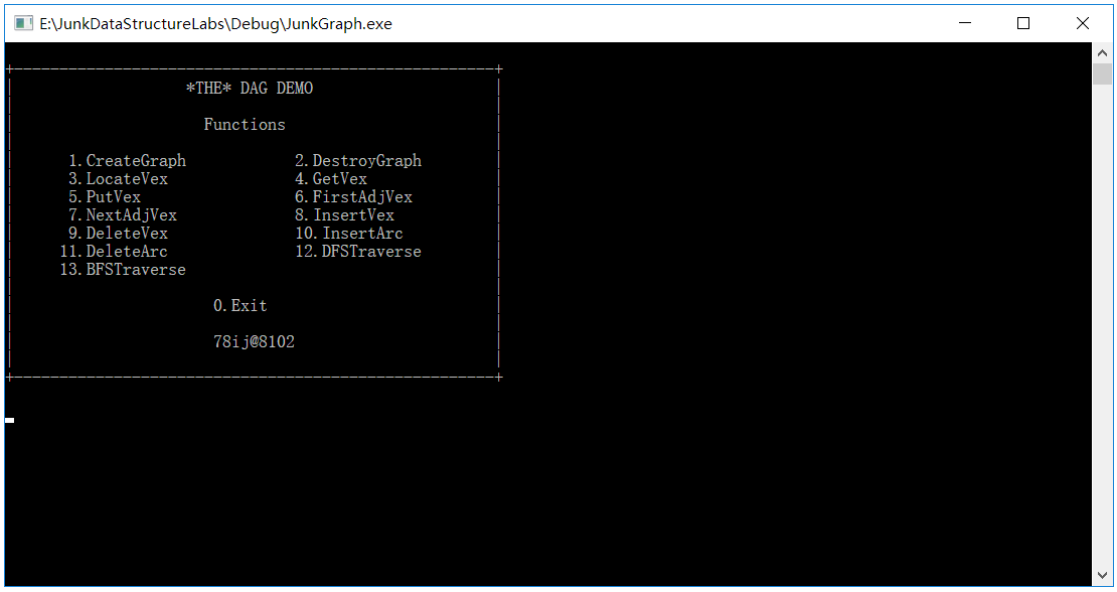

图 4-12 演示系统的示意图

每次操作时要求输入本次操作二叉树的编号：

图 4-13 演示系统的示意图

每次打印出菜单之前，从文件中将所有有向图读取到内存中，每次进行完毕一个操作之后，将内存中的有向图都存到文件之中。

3.3.2 系统测试

下面，选取几个具有代表性的函数进行测试。
（1）创建图

表 3-1 创建图的数据及结果

操作	输入	输出	是否正确
初始化图	输入邻接矩阵等	创建成功	正确

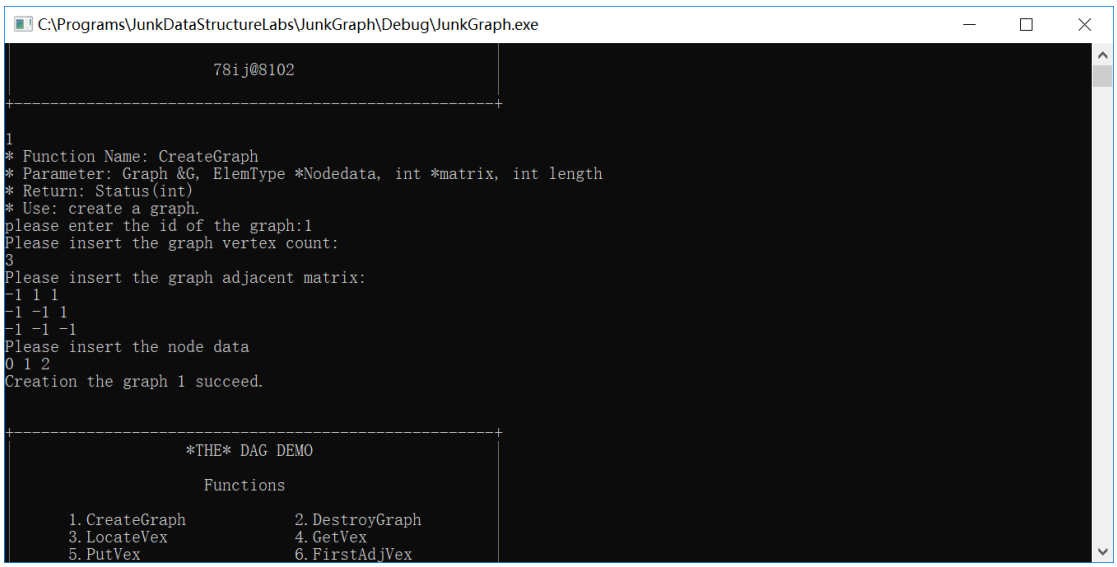

图 4-14 创建图

（2）销毁

表 3-2 销毁图的数据及结果

操作	输入	输出	是否正确
销毁图	图存在	销毁成功	正确
销毁图	图不存在	销毁失败	正确

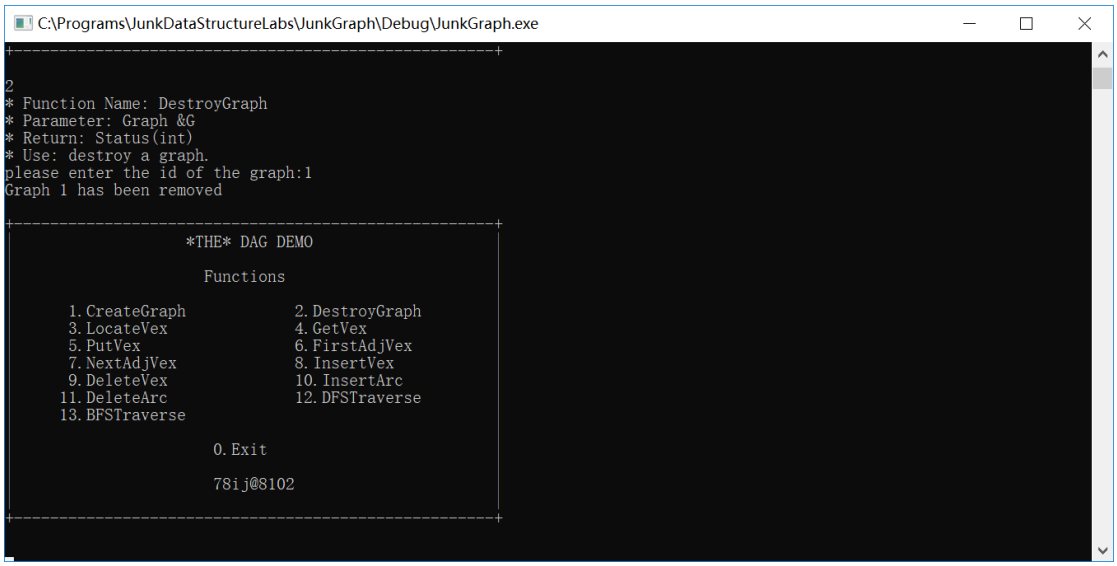

图 4-15 销毁图

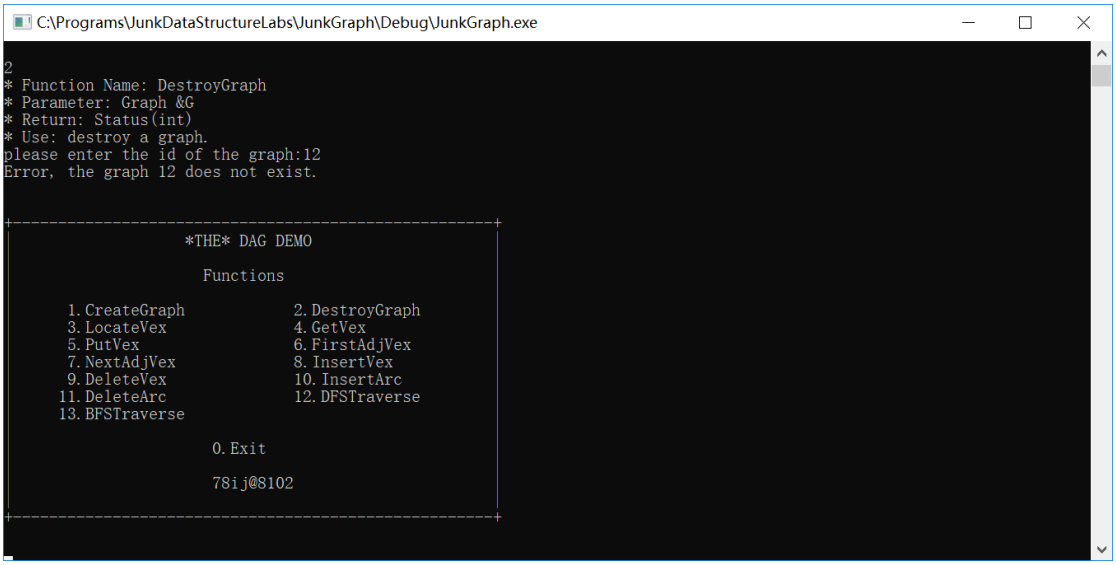

图 4-16 销毁图（2）

（3）定位顶点

表 4-4 定位顶点的数据及结果

操作	输入	输出	是否正确
定位顶点	顶点数据 1	1	正确
定位顶点	顶点不存在	错误	正确

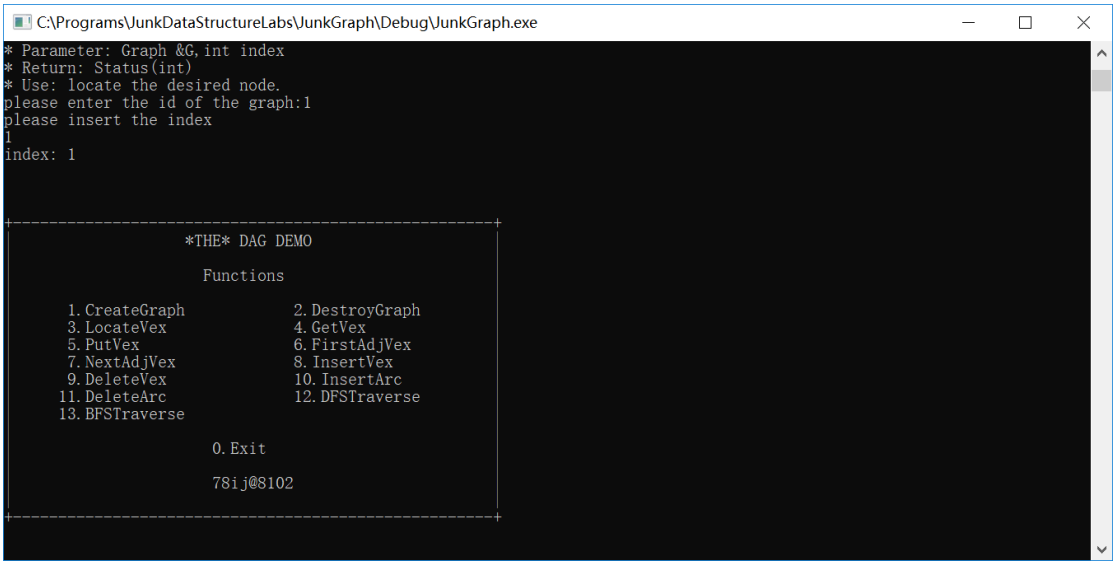

图 4-20 定位顶点

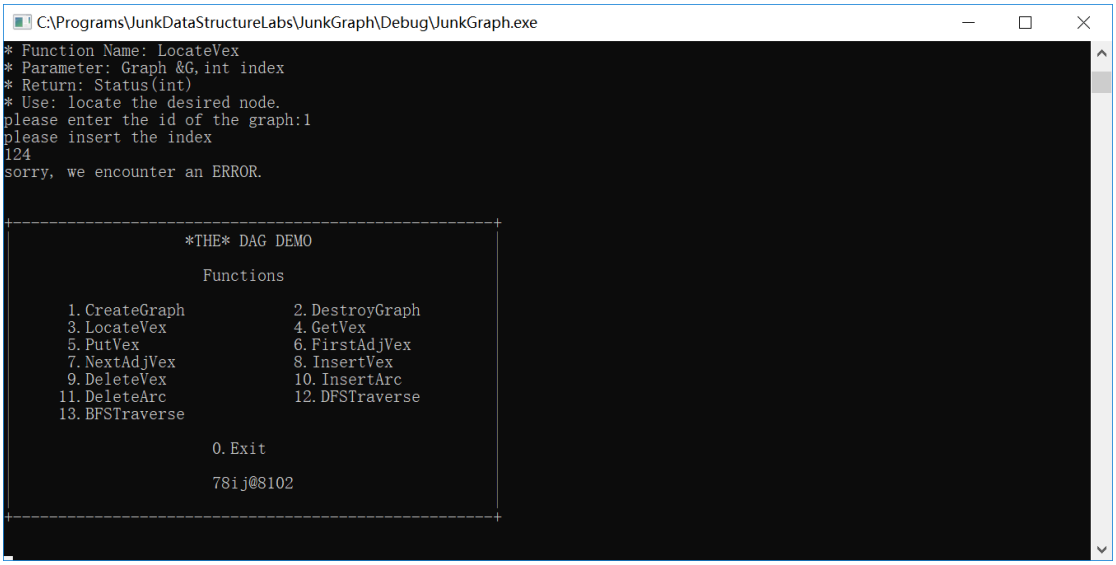

图 4-21 定位顶点（2）

（20）求第一邻接顶点

表 4-5 求第一邻接顶点的数据及结果

操作	输入	输出	是否正确
求第一邻接顶点	第一顶点 1	1	正确
求第一邻接顶点	第一顶点不存在	Error	正确

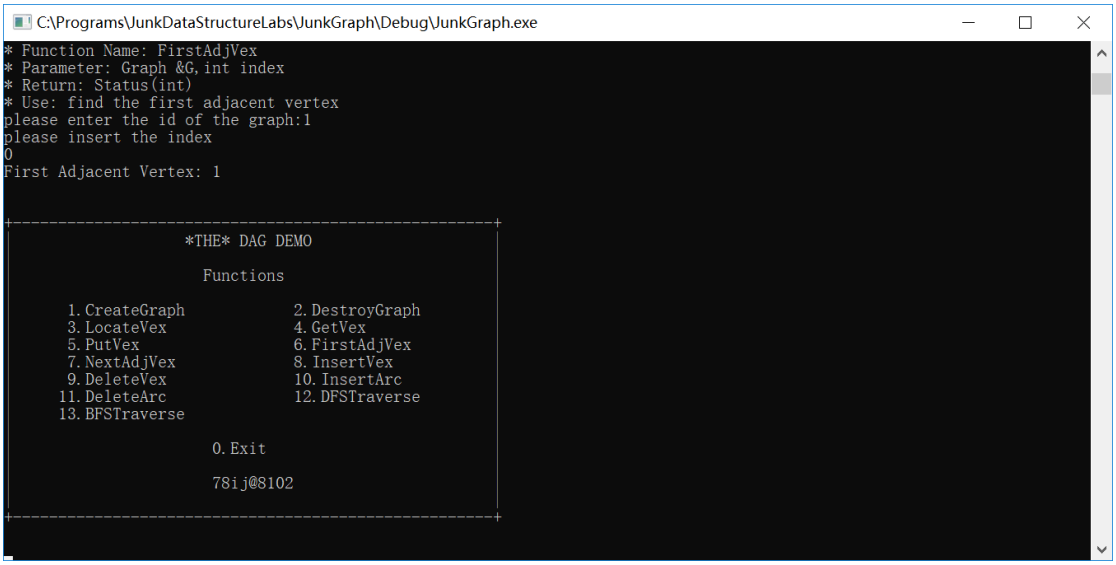

图 4-22 第一邻接节点

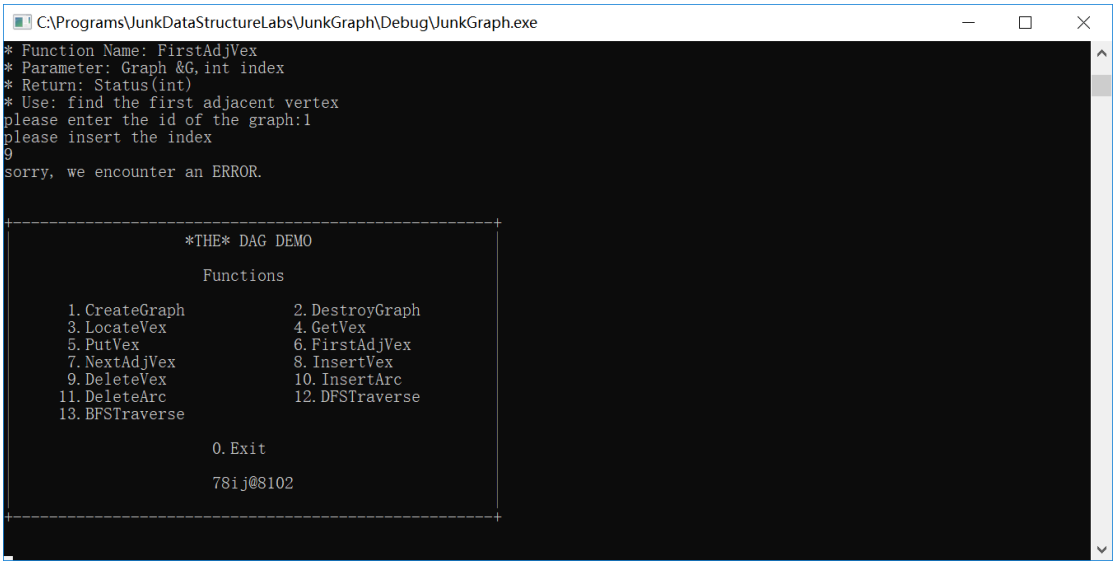

图 4-23 第一邻接节点（2）

（21）BFS 遍历

表 4-6 BFS 遍历的数据及结果

操作	输入	输出	是否正确
BFS 遍历	图 1	0 1 2	正确
BFS 遍历	图 2	0 2 1	正确

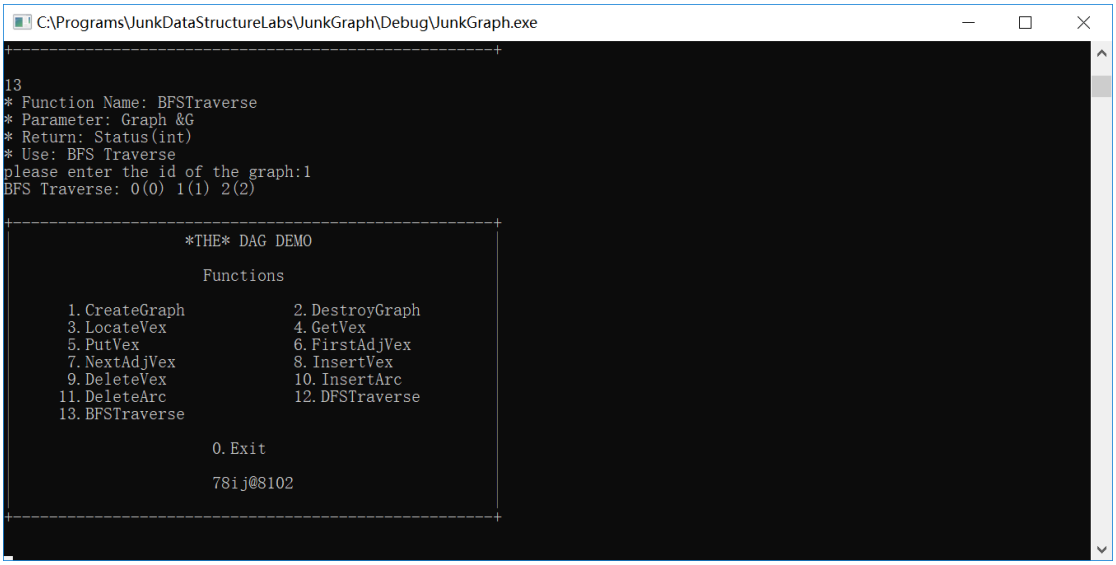

图 4-24 BFS

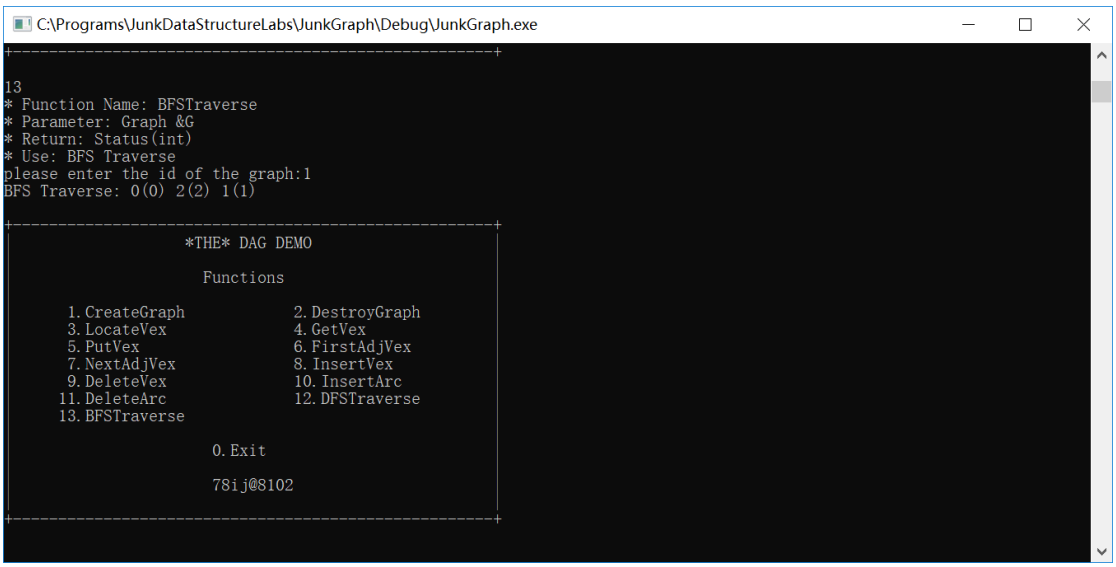

图 4-25 BFS(2)

4.5 实验小结

在进行了树的实验之后，图的实验显得没有那么难，但是还是有一些关键的要点需要注意，比如在销毁和创建的时候图内部链表的处理，很容易产生野指针，还有删除顶点的时候容易产生一些其他的 bug。总之，本次实验完成的比较轻松，也算是为数据结构实验画上了一个完满的句号。

4.6 附录 A Graph.h 代码清单

```
#ifndef GRAPH_H
#define GRAPH_H

#include "LinkedList.h"
#include "common.h"

//有向无权图
typedef struct GNode {
        int nodeindex = 0;
        ElemType nodedata = 0;
        bool visited = false;// used for dfs/bfs
};

typedef struct Graph {
        vector<std::pair<GNode,LinkedList>> data;
        int vexs = 0;
        int GraphID = -1;
        Graph *next;
} Graph;


void   write(Graph &G, FILE *fp);
status CreateGraph(Graph &G, ElemType *Nodedata, int *matrix, int
length);
status DestroyGraph(Graph &G);
status LocateVex(Graph &G, ElemType data);
status GetVex(Graph &G, int index);
status PutVex(Graph &G, int index, ElemType &data);
status FirstAdjVex(Graph &G, int index);
status NextAdjVex(Graph &G, int index, int &adj);
status InsertVex(Graph &G, ElemType data);
status DeleteVex(Graph &G, int index);
status InsertArc(Graph &G, int v, int w);
status DeleteArc(Graph &G, int v, int w);
status DFSTraverse(Graph &G);
status BFSTraverse(Graph &G);

#endif
```

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. ADTs, Data Structures, and Problem Solving with C++. Second Edition, Calvin College, 2005

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

[4] 严蔚敏等.数据结构题集(C 语言版). 清华大学出版社

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 四次实验的代码清单

Common.h:

```
#ifndef COMMON_H
#define COMMON_H

#include <iostream>
#include <malloc.h>
#include <cstdlib>
#include <cstring>
#include <queue>
#include <vector>
#include <stack>
#include <algorithm>

using std::cin;
using std::cout;
using std::endl;
using std::memset;
using std::queue;
using std::vector;
using std::stack;
using std::sort;

//Page 10 on Textbook
#define OK                  1
#define ERROR               0
#define INFEASTABLE -1
#define OVERFLOW        -2

typedef int status;
typedef int ElemType;


#endif //#ifndef COMMON_H
```

SqList.h:

```
#ifndef SqList_H
#define SqList_H

#include "common.h"

//page 22 on textbook
#define LIST_INIT_SIZE 100
#define LISTINCREMENT    10
#define LIST_MAX 500


typedef struct SqList {
```

```
        ElemType *head;
        int length;
        int listsize;
        int ListID = -1;
        SqList *next; //Initially I didn't want to add this.....
}SqList;


//page 19 on textbook
status IntiaList(SqList &L);
status DestroyList(SqList &L);
status ClearList(SqList &L);
bool      ListEmpty(const SqList &L);
int       ListLength(const SqList &L);
status GetElem(const SqList &L,int i, ElemType &e);
int       LocateElem(const SqList &L,const ElemType &e); //¼ò»¯ý
status PriorElem(const SqList &L,const ElemType &cur_e,ElemType &pre_e);
status NextElem(const SqList &L,const ElemType &cur_e,ElemType &next_e);
status ListInsert(SqList &L,int i,ElemType &e);
status ListDelete(SqList &L, int i, ElemType &e);
status ListTraverse(const SqList &L);    //¼ò»¯ý



#endif //SqList_H
```

SqList.cpp:

```
/*
* AUTHOR: Jiamu Sun
* EMAIL: x739566858@outlook.com
* GITHUB: 78ij
*/


#include "SqList.h"

/*
* Function Name: InitaList
* Parameter: SqList &L
* Return: Status(int)
* Use: initialize the linear list
*/
status IntiaList(SqList &L) {
        L.listsize = LIST_INIT_SIZE;
        L.length = 0;
        L.head = (ElemType *)malloc(sizeof(ElemType) * LIST_INIT_SIZE);
        if (L.head == NULL) //分配失败
                return ERROR;
        else return OK;
}

/*
* Function Name: DestroyList
* Parameter: SqList &L
* Return: Status(int)
* Use: destroy the linear list
*/
status DestroyList(SqList &L) {
        L.length = 0;
```

```
    L.listsize = 0;
    free(L.head);
    L.head = NULL; //防止野指针
    return OK;
}

/*
* Function Name: ClearList
* Parameter: SqList &L
* Return: Status(int)
* Use: make the list empty
*/
status ClearList(SqList &L) {
    ElemType *head = (ElemType *)memset(L.head, 0, sizeof(ElemType) * L.length);
    L.length = 0;
    return OK;
}

/*
* Function Name: ListEmpty
* Parameter: const SqList &L
* Return: bool
* Use: check if the list is empty.
*/
bool ListEmpty(const SqList &L) {
    if (L.length != 0) return false;
    else return true;
}

/*
* Function Name: ListLength
* Parameter: SqList &L
* Return: int
* Use: returns the length of the list.
*/
int ListLength(const SqList &L) {
    return L.length;
}

/*
* Function Name: GetElem
* Parameter: const SqList &L, int i ElemType &e
* Return: Status(int)
* Use: get the i-th element of the list(i starts from 1)
*/
status GetElem(const SqList &L, int i, ElemType &e) {
    if (i < 1 || i > L.length) {
        return ERROR; //访问越界
    }
    int j = --i;
    ElemType *elementloc = (ElemType *)L.head + j;
    e = *elementloc;
    return OK;
}

/*
* Function Name: LocateElem
* Parameter: const SqList &L, const ElemType &e
```

```
* Return: int
* Use: return the number of the element that equals the parameter(number starts from 1)
*/
int LocateElem(const SqList &L, const ElemType &e) {
	for (int i = 1; i <= L.length; i++) {
		ElemType ele;
		GetElem(L, i, ele);
		if (ele == e) return i;
	}
	return 0;
}

/*
* Function Name: PriorElem
* Parameter: const SqList &L, ElemType &cur_e, ElemType &pre_e
* Return: Status(int)
* Use: get the the prior element of the specified element, pass it using parameter.
*/
status PriorElem(const SqList &L, const ElemType &cur_e, ElemType &pre_e) {
	int loc = LocateElem(L, cur_e);
	if (loc == 0 || loc == 1) return ERROR;
	else {
		loc--;
		GetElem(L, loc, pre_e);
		return OK;
	}
}

/*
* Function Name: NextElem
* Parameter: const SqList &L, ElemType &cur_e, ElemType &next_e
* Return: Status(int)
* Use: get the the next element of the specified element, pass it using parameter.
*/
status NextElem(const SqList &L, const ElemType &cur_e, ElemType &next_e) {
	int loc = LocateElem(L, cur_e);
	if (loc == L.length || loc == 0) return ERROR;
	else {
		loc++;
		GetElem(L, loc, next_e);
		return OK;
	}
}

/*
* Function Name: ListInsert
* Parameter: SqList &L, int i, ElemType &e
* Return: Status(int)
* Use: insert an element after the specifyed number
*/
status ListInsert(SqList &L, int i, ElemType &e) {

	if (i < 1 || i > L.length + 1) return ERROR;
	if (L.length + 1 > L.listsize) { //扩大存储空间并复制元素
		if (L.listsize + LISTINCREMENT > LIST_MAX) return OVERFLOW;
		L.listsize += LISTINCREMENT;
		ElemType *temp = L.head;
		L.head = (ElemType *)realloc(L.head,sizeof(ElemType) * L.listsize);
```

```
        }
        for (int j = L.length - 1; j >= i - 1; j--) {
                *(L.head + j + 1) = *(L.head + j);
        }
        *(L.head + i - 1) = e;
        L.length++;
        return OK;
}

/*
* Function Name: ListDelete
* Parameter: SqList &L, int i, ElemType &e
* Return: Status(int)
* Use: Delete the specified element.
*/
status ListDelete(SqList &L, int i, ElemType &e) {
        if (i < 1 || i > L.length) return ERROR;
        GetElem(L, i, e);
        for (int j = i; j < L.length; j++) {
                *(L.head + j - 1) = *(L.head + j);
        }
        L.length--;
        return OK;
}

/*
* Function Name: ListTraverse
* Parameter: const SqList &L
* Return: Status(int)
* Use: Traverse the list and output its elements.
*/
status ListTraverse(const SqList &L) {
        ElemType e;
        for (int i = 1; i <= L.length; i++) {
                GetElem(L, i, e);
                cout << e << " ";
        }
        cout << endl;
        return OK;
}
```

Main.cpp:

```
#include "SqList.h"

void PrintMenu(void) {
        /*
        * Function Name: PrintMenu
        * Parameter: None
        * Return: None
        * Use: Print the main menu
        */

        printf("\n+--------------------------------------------------------+\n");
        printf("|                        *THE* LINEAR LIST DEMO                        |\n");
        printf("|                                                                      |\n");
        printf("|                              Functions                               |\n");
        printf("|                                                                      |\n");
        printf("|        1.InitalList              2.DestroyList             |\n");
```

```
        printf("|         3.ClearList                 4.IsListEmpty           |\n");
        printf("|         5.ListLength                6.GetElem               |\n");
        printf("|         7.LocateElem                8.PriorElem             |\n");
        printf("|         9.NextElem                  10.ListInsert           |\n");
        printf("|        11.ListDelete                12.ListTraverse         |\n");
        printf("|                                                             |\n");
        printf("|                          0.Exit                             |\n");
        printf("|                                                             |\n");
        printf("|                          78ij@8102                          |\n");
        printf("|                                                             |\n");
        printf("+-------------------------------------------------------------+\n");
        printf("\n");

}

status LoadData(SqList **head) {
        /*
        * Function Name: LoadData
        * Parameter: std::vector<SqList> lists
        * Return Status(int)
        * Use: load data from file
        */
        FILE *fp = fopen("SLDB", "rb");
        if (fp == NULL)
                return ERROR;

        int size = 0;
        int count = 0;
        SqList *tmp = (SqList *)malloc(sizeof(SqList));
        size = fread(tmp, sizeof(SqList), 1, fp);
        if (size == 0)
                return OK;
        count++;
        tmp->head = (int *)malloc(sizeof(int) * tmp->listsize);
        size = fread(tmp->head, sizeof(int) * tmp->listsize, 1, fp);
        *head = tmp;


        while (1) {
                SqList *tmp = (SqList *)malloc(sizeof(SqList));
                size = fread(tmp, sizeof(SqList), 1, fp);
                if (size == 0)
                        break;
                count++;
                tmp->head = (int *)malloc(sizeof(int) * tmp->listsize);
                size = fread(tmp->head, sizeof(int) * tmp->listsize, 1, fp);
                (*head)->next = tmp;
                *head = (*head)->next;
        }
        (*head)->next = NULL;
        *head = tmp;
        fclose(fp);
        return OK;
}

status SaveData(SqList *head) {
        /*
        * Function Name: SaveData
```

```
	* Parameter: vector<SqList> lists
	* Return: Status(int)
	* Use: save data to file
	*/

	FILE *fp = fopen("SLDB", "wb");
	if (fp == NULL)
		return ERROR;

	SqList *L = head,*p = head;
	while (L != NULL) {
		fwrite(L, sizeof(SqList), 1, fp);
		fwrite(L->head, sizeof(int) * L->listsize, 1, fp);
		p = L->next;
		DestroyList(*L);
		L = p;
	}

	fclose(fp);
	return OK;
}


int main() {
	int selection = -1;
	SqList *head = NULL;
	while (selection != 0){
		PrintMenu();
		scanf("%d", &selection);
		LoadData(&head);
		SqList *L = head;
		SqList *tmp = head;
		int list_index;
		switch (selection) {
		case -1: //for debug purposes
			while (head != NULL) {
				printf("ListID:%d\tListlength:%d\tListsize:%d\n",
head->ListID,head->length,head->listsize);
				head = head->next;
			}
			head = L;
			break;
		case 1:
			printf("* Function Name: InitaList\n");
			printf("* Parameter: SqList &L\n");
			printf("* Return: Status(int)\n");
			printf("* Use: initialize the linear list\n");
			printf("please enter the id of the list:");
			scanf("%d", &list_index);
			while (head != NULL) {
				if (head->ListID == list_index)
					break;
				head = head->next;
			}
			if (head != NULL) {
				printf("Error, the list %d already exist.\n", list_index);
			}
			else {
```

```
            SqList *new_list = (SqList *)malloc(sizeof(SqList));
            if (IntiaList(*new_list) == OK) {
                printf("Inital the list %d succeed.\n", list_index);
                new_list->ListID = list_index;
                new_list->next = L;
                head = new_list;
            }
            else {
                printf("ERROR, something wrong with the RAM\n");
            }
        }
        printf("\n");
        break;
    case 2:
        printf("/*\n");
        printf("* Function Name: DestroyList\n");
        printf("* Parameter: SqList &L\n");
        printf("* Return: Status(int)\n");
        printf("* Use: destroy the linear list\n");
        printf("please enter the id of the list:");
        scanf("%d", &list_index);
        if (head == NULL) {
            printf("Error, the list %d does not exist.\n", list_index);
            break;
        }
        if (head->ListID == list_index) {
            head = head->next;
            DestroyList(*L);
            printf("List %d has been removed\n", list_index);
            break;
        }
        while (head->next != NULL) {
            if (head->next->ListID == list_index)
                break;
            head = head->next;
        }
        if (head->next == NULL) {
            printf("Error, the list %d does not exist.\n", list_index);
            head = L;
        }
        else {
            L = head->next;
            head->next = head->next->next;
            DestroyList(*L);
            printf("List %d has been removed\n", list_index);
            head = tmp;
        }
        printf("\n");
        break;
    case 3:
        printf("* Function Name: ClearList\n");
        printf("* Parameter: SqList &L\n");
        printf("* Return: Status(int)\n");
        printf("* Use: make the list empty\n");
        printf("please enter the id of the list:");
        scanf("%d", &list_index);
        while (head != NULL) {
            if (head->ListID == list_index)
```

```
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the list %d does not exist.\n", list_index);
            head = L;
        }
        else {
            ClearList(*head);
            head = L;
            printf("the list %d has been cleared.\n", list_index);
        }

        printf("\n");
        break;
    case 4:
        printf("* Function Name: ListEmpty\n");
        printf("* Parameter: const SqList &L\n");
        printf("* Return: bool\n");
        printf("* Use: check if the list is empty.\n");
        printf("please enter the id of the list:");
        scanf("%d", &list_index);
        while (head != NULL) {
            if (head->ListID == list_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the list %d does not exist.\n", list_index);
            head = L;
        }
        else {
            bool isempty = ListEmpty(*head);
            head = L;
            if(isempty)
                printf("the list %d is empty.\n", list_index);
            else
                printf("the list %d is not empty.\n", list_index);

        }

        printf("\n");
        break;
    case 5:
        printf("* Function Name: ListLength\n");
        printf("* Parameter: SqList &L\n");
        printf("* Return: int\n");
        printf("* Use: returns the length of the list.\n");
        printf("please enter the id of the list:");
        scanf("%d", &list_index);
        while (head != NULL) {
            if (head->ListID == list_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the list %d does not exist.\n", list_index);
            head = L;
```

```
        }
        else {
            int length = ListLength(*head);
            head = L;
            printf("the list %d's length is %d.\n", list_index, length);
        }
        printf("\n");
        break;
    case 6:
        printf("* Function Name: GetElem\n");
        printf("* Parameter: const SqList &L, int i ElemType &e\n");
        printf("* Return: Status(int)\n");
        printf("* Use: get the i-th element of the list(i starts from 1)\n");
        printf("please enter the id of the list:");
        scanf("%d", &list_index);
        while (head != NULL) {
            if (head->ListID == list_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the list %d does not exist.\n", list_index);
            head = L;
        }
        else {
            printf("please enter the element number:\n");
            int num;
            ElemType value;
            scanf("%d", &num);
            status res = GetElem(*head, num, value);
            head = L;

            if (res == ERROR) {
                printf("Sorry, your number is out of bound.\n");
                break;
            }
            else
                printf("the element value is %d.\n", value);

        }
        printf("\n");
        break;
    case 7:
        printf("* Function Name: LocateElem\n");
        printf("* Parameter: const SqList &L, const ElemType &e\n");
        printf("* Return: int\n");
        printf("* Use: return the number of the element that equals the parameter(number starts from 1)\n");
        printf("please enter the id of the list:");
        scanf("%d", &list_index);
        while (head != NULL) {
            if (head->ListID == list_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the list %d does not exist.\n", list_index);
            head = L;
        }
```

```
        else {
            printf("please enter the element value:\n");
            ElemType value;
            scanf("%d", &value);
            int res = LocateElem(*head, value);
            head = L;

            if (res == 0) {
                printf("Sorry, no such element.\n");
                break;
            }
            else
                printf("the element number is %d.\n", res);

        }
        printf("\n");
        break;
    case 8:
        printf("* Function Name: PriorElem\n");
        printf("* Parameter: const SqList &L, ElemType &cur_e, ElemType &pre_e\n");
        printf("* Return: Status(int)\n");
        printf("* Use: get the the prior element of the specified element, pass it using parameter.\n");
        printf("please enter the id of the list:");
        scanf("%d", &list_index);
        while (head != NULL) {
            if (head->ListID == list_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the list %d does not exist.\n", list_index);
            head = L;
        }
        else {
            printf("please enter the element value:\n");
            ElemType cur;
            ElemType value;
            scanf("%d", &cur);
            status res =PriorElem(*head, cur,value);
            head = L;

            if (res == ERROR) {
                printf("Sorry, we encounter an error.\n");
                break;
            }
            else
                printf("the prior element number is %d.\n", value);

        }
        printf("\n");
        break;
    case 9:
        printf("* Function Name: NextElem\n");
        printf("* Parameter: const SqList &L, ElemType &cur_e, ElemType &next_e\n");
        printf("* Return: Status(int)\n");
        printf("* Use: get the the next element of the specified element, pass it using parameter.\n");
        printf("please enter the id of the list:");
        scanf("%d", &list_index);
```

```
        while (head != NULL) {
            if (head->ListID == list_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the list %d does not exist.\n", list_index);
            head = L;
        }
        else {
            printf("please enter the element value:\n");
            ElemType cur;
            ElemType value;
            scanf("%d", &cur);
            status res = NextElem(*head, cur, value);
            head = L;

            if (res == ERROR) {
                printf("Sorry, we encounter an error.\n");
                break;
            }
            else
                printf("the next element number is %d.\n", value);

        }
        printf("\n");
        break;
    case 10:
        printf("* Function Name: ListInsert\n");
        printf("* Parameter: SqList &L, int i, ElemType &e\n");
        printf("* Return: Status(int)\n");
        printf("* Use: insert an element after the specifyed number(the list must be non-empty)\n");
        printf("please enter the id of the list:");
        scanf("%d", &list_index);
        while (head != NULL) {
            if (head->ListID == list_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the list %d does not exist.\n", list_index);
            head = L;
        }
        else {
            int num;
            ElemType e;
            printf("please input the number of the element\n");
            scanf("%d", &num);
            printf("please input the inserted value:\n");
            scanf("%d", &e);
            status res = ListInsert(*head, num, e);
            head = L;
            if (res == ERROR) {
                printf("Sorry, we encounter an error.\n");
                break;
            }
            else
                printf("value %d has been successfully insert to the %d position of %d list.", e, num,
```

```
list_index);
                }
                printf("\n");
                break;
        case 11:
                printf("* Function Name: ListDelete\n");
                printf("* Parameter: SqList &L, int i, ElemType &e\n");
                printf("* Return: Status(int)\n");
                printf("* Use: Delete the specified element.\n");
                printf("please enter the id of the list:");
                scanf("%d", &list_index);
                while (head != NULL) {
                        if (head->ListID == list_index)
                                break;
                        head = head->next;
                }
                if (head == NULL) {
                        printf("Error, the list %d does not exist.\n", list_index);
                        head = L;
                }
                else {
                        int num;
                        printf("please input the number of the element\n");
                        scanf("%d", &num);
                        ElemType e;
                        status res = ListDelete(*head, num,e);
                        head = L;
                        if (res == ERROR) {
                                printf("Sorry, we encounter an error.\n");
                                break;
                        }
                        else
                                printf("value %d has benn successfully delete, it's in %d position of %d list.", e, num,
list_index);
                }
                printf("\n");
                break;
        case 12:
                printf("* Function Name: ListTraverse\n");
                printf("* Parameter: const SqList &L\n");
                printf("* Return: Status(int)\n");
                printf("* Use: Traverse the list and output its elements.\n");
                printf("please enter the id of the list:");
                scanf("%d", &list_index);
                while (head != NULL) {
                        if (head->ListID == list_index)
                                break;
                        head = head->next;
                }
                if (head == NULL) {
                        printf("Error, the list %d does not exist.\n", list_index);
                        head = L;
                }
                else {
                        printf("Traverse the %d-th list:\n", list_index);
                        ListTraverse(*head);
                        head = L;
                }
```

```
                printf("\n");
                break;
            case 0:
                printf("Thank you for using~\n");
                break;
            default:
                printf("no such selection.\n");
                break;
            }
            SaveData(head);
        }
        return 0;
}
```

LinkedList.h:

```
/*
* AUTHOR: Jiamu Sun
* EMAIL: x739566858@outlook.com
* GITHUB: 78ij
*/

#include "common.h"

//带头结点的链表.
typedef struct LinkedListNode {
        ElemType data;
        LinkedListNode *next;
} LinkedListNode;

typedef struct LinkedList{
        int length;
        LinkedListNode *head;
        LinkedList *next;
        int ListID = -1;
} LinkedList;

status IntiaList(LinkedList &L);
status DestroyList(LinkedList &L);
status ClearList(LinkedList &L);
bool      ListEmpty(const LinkedList &L);
int       ListLength(const LinkedList &L);
status GetElem(const LinkedList &L, int i, ElemType &e);
int       LocateElem(const LinkedList &L, const ElemType &e); //简化过
status PriorElem(const LinkedList &L, const ElemType &cur_e, ElemType &pre_e);
status NextElem(const LinkedList &L, const ElemType &cur_e, ElemType &next_e);
status ListInsert(LinkedList &L, int i, ElemType &e);
status ListDelete(LinkedList &L, int i, ElemType &e);
status ListTraverse(const LinkedList &L);    //简化过
```

LinkedList.cpp:

```
/*
* AUTHOR: Jiamu Sun
* EMAIL: x739566858@outlook.com
* GITHUB: 78ij
*/

#include "LinkedList.h"

/*
* Function Name: InitaList
```

```
* Parameter: LinkedList &L
* Return: Status(int)
* Use: initialize the linear list
*/
status IntiaList(LinkedList &L) {
	L.head = (LinkedListNode *)malloc(sizeof(LinkedListNode));
	if (L.head == NULL) return ERROR;
	L.head->next = NULL;
	L.length = 0;
	/*L.length = 1;
	LinkedListNode *tmp = (LinkedListNode *)malloc(sizeof(LinkedListNode));
	tmp->data = 1;
	L.head->next = tmp;
	tmp->next = NULL;*/
	return OK;
}

/*
* Function Name: DestroyList
* Parameter: LinkedList &L
* Return: Status(int)
* Use: destroy the linear list
*/
status DestroyList(LinkedList &L) {
	LinkedListNode *tmp = L.head->next;
	LinkedListNode *tmp2 = tmp;
	while (tmp != NULL) {
		tmp2 = tmp->next;
		free(tmp);
		tmp = tmp2;
	}
	free(L.head);
	L.head = NULL;
	L.length = 0;
	return OK;
}

/*
* Function Name: ClearList
* Parameter: LinkedList &L
* Return: Status(int)
* Use: make the list empty
*/

status ClearList(LinkedList &L) {
	LinkedListNode *tmp = L.head->next;
	LinkedListNode *tmp2 = tmp;
	while (tmp != NULL) {
		tmp2 = tmp->next;
		free(tmp);
		tmp = tmp2;
	}
	L.head->next = NULL;
	L.length = 0;
	return OK;
}

/*
```

```
* Function Name: ListEmpty
* Parameter: const LinkedList &L
* Return: bool
* Use: check if the list is empty.
*/

bool ListEmpty(const LinkedList &L) {
	if (L.length == 0) return true;
	else return false;
}

/*
* Function Name: ListLength
* Parameter: LinkedList &L
* Return: int
* Use: returns the length of the list.
*/

int ListLength(const LinkedList &L) {
	return L.length;
}

/*
* Function Name: GetElem
* Parameter: const LinkedList &L, int i ElemType &e
* Return: Status(int)
* Use: get the i-th element of the list(i starts from 1)
*/
status GetElem(const LinkedList &L, int i, ElemType &e) {
	if (i < 1 || i > L.length) {
		return ERROR; //·أختش½½ç
	}
	LinkedListNode *ele = L.head;
	for (int j = 0; j < i; j++) {
		ele = ele->next;
	}
	e = ele->data;
	return OK;
}

/*
* Function Name: LocateElem
* Parameter: const LinkedList &L, const ElemType &e
* Return: int
* Use: return the number of the element that equals the parameter(number starts from 1)
*/
int LocateElem(const LinkedList &L, const ElemType &e) {
	LinkedListNode *ele = L.head;
	for (int i = 0; i < L.length; i++) {
		ele = ele->next;
		if (ele->data == e) return i + 1;
	}
	return 0;
}

/*
* Function Name: PriorElem
* Parameter: const LinkedList &L, ElemType &cur_e, ElemType &pre_e
```

```
* Return: Status(int)
* Use: get the the prior element of the specified element, pass it using parameter.
*/
status PriorElem(const LinkedList &L, const ElemType &cur_e, ElemType &pre_e) {
	int loc = LocateElem(L, cur_e);
	if (loc == 0 || loc == 1) return ERROR;
	else {
		loc--;
		GetElem(L, loc, pre_e);
		return OK;
	}
}

/*
* Function Name: NextElem
* Parameter: const LinkedList &L, ElemType &cur_e, ElemType &next_e
* Return: Status(int)
* Use: get the the next element of the specified element, pass it using parameter.
*/
status NextElem(const LinkedList &L, const ElemType &cur_e, ElemType &next_e) {
	int loc = LocateElem(L, cur_e);
	if (loc == L.length || loc == 0) return ERROR;
	else {
		loc++;
		GetElem(L, loc, next_e);
		return OK;
	}
}

/*
* Function Name: ListInsert
* Parameter: LinkedList &L, int i, ElemType &e
* Return: Status(int)
* Use: insert an element after the specifyed number(the list must be non-empty)
*/
status ListInsert(LinkedList &L, int i, ElemType &e) {
	if (i < 1 || i > L.length + 1) return ERROR;
	LinkedListNode *ele = L.head;
	for (int j = 0; j < i - 1; j++) {
		ele = ele->next;
	}
	LinkedListNode *tmp = (LinkedListNode *)malloc(sizeof(LinkedListNode));
	tmp->data = e;
	tmp->next = ele->next;
	ele->next = tmp;
	L.length++;
	return OK;
}

/*
* Function Name: ListDelete
* Parameter: LinkedList &L, int i, ElemType &e
* Return: Status(int)
* Use: Delete the specified element.
*/
status ListDelete(LinkedList &L, int i, ElemType &e) {
	if (i < 1 || i > L.length) return ERROR;
	L.length--;
```

```
        GetElem(L, i, e);
        LinkedListNode *ele = L.head;
        for (int j = 0; j < i - 1; j++) {
                ele = ele->next;
        }
        LinkedListNode *tmp = ele->next;
        ele->next = tmp->next;
        free(tmp);
        return OK;
}

/*
* Function Name: ListTraverse
* Parameter: const LinkedList &L
* Return: Status(int)
* Use: Traverse the list and output its elements.
*/
status ListTraverse(const LinkedList &L) {
        LinkedListNode *ele = L.head->next;
        while (ele != NULL) {
                cout << ele->data << " ";
                ele = ele->next;
        }
        cout << endl;
        return OK;
}
```

Main.cpp:

```
/*
* AUTHOR: Jiamu Sun
* EMAIL: x739566858@outlook.com
* GITHUB: 78ij
*/

#include "LinkedList.h"

/*
* Function Name: InitaList
* Parameter: LinkedList &L
* Return: Status(int)
* Use: initialize the linear list
*/
status IntiaList(LinkedList &L) {
        L.head = (LinkedListNode *)malloc(sizeof(LinkedListNode));
        if (L.head == NULL) return ERROR;
        L.head->next = NULL;
        L.length = 0;
        /*L.length = 1;
        LinkedListNode *tmp = (LinkedListNode *)malloc(sizeof(LinkedListNode));
        tmp->data = 1;
        L.head->next = tmp;
        tmp->next = NULL;*/
        return OK;
}

/*
* Function Name: DestroyList
* Parameter: LinkedList &L
* Return: Status(int)
```

```
* Use: destroy the linear list
*/
status DestroyList(LinkedList &L) {
	LinkedListNode *tmp = L.head->next;
	LinkedListNode *tmp2 = tmp;
	while (tmp != NULL) {
		tmp2 = tmp->next;
		free(tmp);
		tmp = tmp2;
	}
	free(L.head);
	L.head = NULL;
	L.length = 0;
	return OK;
}

/*
* Function Name: ClearList
* Parameter: LinkedList &L
* Return: Status(int)
* Use: make the list empty
*/

status ClearList(LinkedList &L) {
	LinkedListNode *tmp = L.head->next;
	LinkedListNode *tmp2 = tmp;
	while (tmp != NULL) {
		tmp2 = tmp->next;
		free(tmp);
		tmp = tmp2;
	}
	L.head->next = NULL;
	L.length = 0;
	return OK;
}

/*
* Function Name: ListEmpty
* Parameter: const LinkedList &L
* Return: bool
* Use: check if the list is empty.
*/

bool ListEmpty(const LinkedList &L) {
	if (L.length == 0) return true;
	else return false;
}

/*
* Function Name: ListLength
* Parameter: LinkedList &L
* Return: int
* Use: returns the length of the list.
*/

int ListLength(const LinkedList &L) {
	return L.length;
}
```

```
/*
* Function Name: GetElem
* Parameter: const LinkedList &L, int i ElemType &e
* Return: Status(int)
* Use: get the i-th element of the list(i starts from 1)
*/
status GetElem(const LinkedList &L, int i, ElemType &e) {
	if (i < 1 || i > L.length) {
		return ERROR; //·أختش½½ç
	}
	LinkedListNode *ele = L.head;
	for (int j = 0; j < i; j++) {
		ele = ele->next;
	}
	e = ele->data;
	return OK;
}

/*
* Function Name: LocateElem
* Parameter: const LinkedList &L, const ElemType &e
* Return: int
* Use: return the number of the element that equals the parameter(number starts from 1)
*/
int LocateElem(const LinkedList &L, const ElemType &e) {
	LinkedListNode *ele = L.head;
	for (int i = 0; i < L.length; i++) {
		ele = ele->next;
		if (ele->data == e) return i + 1;
	}
	return 0;
}

/*
* Function Name: PriorElem
* Parameter: const LinkedList &L, ElemType &cur_e, ElemType &pre_e
* Return: Status(int)
* Use: get the the prior element of the specified element, pass it using parameter.
*/
status PriorElem(const LinkedList &L, const ElemType &cur_e, ElemType &pre_e) {
	int loc = LocateElem(L, cur_e);
	if (loc == 0 || loc == 1) return ERROR;
	else {
		loc--;
		GetElem(L, loc, pre_e);
		return OK;
	}
}

/*
* Function Name: NextElem
* Parameter: const LinkedList &L, ElemType &cur_e, ElemType &next_e
* Return: Status(int)
* Use: get the the next element of the specified element, pass it using parameter.
*/
status NextElem(const LinkedList &L, const ElemType &cur_e, ElemType &next_e) {
	int loc = LocateElem(L, cur_e);
```

```
    if (loc == L.length || loc == 0) return ERROR;
    else {
        loc++;
        GetElem(L, loc, next_e);
        return OK;
    }
}

/*
* Function Name: ListInsert
* Parameter: LinkedList &L, int i, ElemType &e
* Return: Status(int)
* Use: insert an element after the specifyed number(the list must be non-empty)
*/
status ListInsert(LinkedList &L, int i, ElemType &e) {
    if (i < 1 || i > L.length + 1) return ERROR;
    LinkedListNode *ele = L.head;
    for (int j = 0; j < i - 1; j++) {
        ele = ele->next;
    }
    LinkedListNode *tmp = (LinkedListNode *)malloc(sizeof(LinkedListNode));
    tmp->data = e;
    tmp->next = ele->next;
    ele->next = tmp;
    L.length++;
    return OK;
}

/*
* Function Name: ListDelete
* Parameter: LinkedList &L, int i, ElemType &e
* Return: Status(int)
* Use: Delete the specified element.
*/
status ListDelete(LinkedList &L, int i, ElemType &e) {
    if (i < 1 || i > L.length) return ERROR;
    L.length--;
    GetElem(L, i, e);
    LinkedListNode *ele = L.head;
    for (int j = 0; j < i - 1; j++) {
        ele = ele->next;
    }
    LinkedListNode *tmp = ele->next;
    ele->next = tmp->next;
    free(tmp);
    return OK;
}

/*
* Function Name: ListTraverse
* Parameter: const LinkedList &L
* Return: Status(int)
* Use: Traverse the list and output its elements.
*/
status ListTraverse(const LinkedList &L) {
    LinkedListNode *ele = L.head->next;
    while (ele != NULL) {
        cout << ele->data << " ";
```

```
            ele = ele->next;
        }
        cout << endl;
        return OK;
}
```

BiTree.h:

```
/*
* AUTHOR: Jiamu Sun
* EMAIL: x739566858@outlook.com
* GITHUB: 78ij
*/

#include "common.h"


typedef struct BiTreeNode {
        ElemType data;
        int index; // used to mark the node
        BiTreeNode *parent = NULL;
        BiTreeNode *left = NULL;
        BiTreeNode *right = NULL;
}BiTreeNode;

typedef struct BiTree {
        int TreeID = -1;
        int length = 0;
        BiTreeNode *root;
        BiTree *next;
}BiTree;

enum TraverseMethod {
        PRE, IN, POST, LEVEL
};

//APIs
bool        isvalid(int *pre, int *in, int length);
void            write(BiTreeNode *root, TraverseMethod method, bool isindex, FILE *fp);
status      InitBiTree(BiTree &T);
status      DestroyBiTree(BiTree &T);
status        CreateBiTree(BiTree &T, int length, int *preorder, int *inorder, ElemType * data);
status      ClearBiTree(BiTree &T);
bool        BiTreeEmpty(const BiTree &T);
int                 BiTreeDepth(const BiTree &T);
BiTreeNode *Root(const BiTree &T);
status      Value(const BiTree &T, int index, ElemType &value);
status      Assign(BiTree &T, int index,ElemType &value);
BiTreeNode *Parent(const BiTree &T, int index);
BiTreeNode *LeftChild(const BiTree &T, int index);
BiTreeNode *RightChild(const BiTree &T, int index);
BiTreeNode *LeftSibling(const BiTree &T, int index);
BiTreeNode *RightSibling(const BiTree &T, int index);
status        InsertChild(BiTree &T, int index, int LR, BiTree &c);
status        DeleteChild(BiTree &T, int index, int LR);
status        PreOrderTraverse(const BiTree &T);
status        InOrderTraverse(const BiTree &T);
status        PostOrderTraverse(const BiTree &T);
status      LevelOrderTraverse(const BiTree &T);
```

BiTree.cpp:

```
#include "BiTree.h"

//-------------------------------------
//       Auxiliary functions
//-------------------------------------



void write(BiTreeNode *root, TraverseMethod method, bool isindex,FILE *fp) {
	if (root == NULL) return;
	if (isindex) {
		switch (method) {
		case PRE:
			fwrite(&root->index, sizeof(int), 1, fp);
			write(root->left, method, isindex, fp);
			write(root->right, method, isindex, fp);
			break;
		case IN:
			write(root->left, method, isindex, fp);
			fwrite(&root->index, sizeof(int), 1, fp);
			write(root->right, method, isindex, fp);
			break;
		}
	}
	else {
		write(root->left, method, isindex, fp);
		fwrite(&root->data, sizeof(ElemType), 1, fp);
		write(root->right, method, isindex, fp);
	}
}


bool isvalid(int *pre, int *in, int length) {
	vector<int> prev;
	vector<int> inv;
	for (int i = 0; i < length; i++) {
		prev.push_back(*(pre + i));
		inv.push_back(*(in + i));
	}
	sort(prev.begin(), prev.end());
	sort(inv.begin(), inv.end());
	for (int i = 0; i < length; i++) {
		if (prev[i] != inv[i]) return false;
	}
	return true;
}
void FreeNodes(BiTreeNode *root) {
	if (root == NULL) return;
	if (root->left != NULL)
		FreeNodes(root->left);
	if (root->right != NULL)
		FreeNodes(root->right);
	free(root);

}

int search(int value, int *string, int length) {
	if (length <= 0) return -1;
```

```
    for (int i = 0; i < length; i++) {
        if (string[i] == value) return i;
    }
    return -1;
}

//data:in order
BiTreeNode *Create(int *pre, int *in, int length,ElemType *data) {
    int rootindex = search(pre[0], in, length);
    if (rootindex == -1) return NULL;
    BiTreeNode *root = (BiTreeNode *)malloc(sizeof(BiTreeNode));
    root->data = data[rootindex];
    root->index = in[rootindex];
    root->left = Create(pre + 1, in, rootindex + 1,data);
    if (root->left != NULL) root->left->parent = root;
    root->right = Create(pre + rootindex + 1, in + rootindex + 1, length - rootindex - 1, data + rootindex + 1);
    if (root->right != NULL) root->right->parent = root;
    return root;
}

void Traverse(BiTreeNode *root, TraverseMethod method) {
    if (root == NULL) return;
    switch (method) {
    case PRE:
        cout << root->data << " ";
        Traverse(root->left, PRE);
        Traverse(root->right, PRE);
        break;
    case IN:
        Traverse(root->left, IN);
        cout << root->data << " ";
        Traverse(root->right, IN);
        break;
    case POST:
        Traverse(root->left, POST);
        Traverse(root->right, POST);
        cout << root->data << " ";
        break;
    case LEVEL:
        queue<BiTreeNode *> q;
        q.push(root);
        while (q.size() != 0) {
            BiTreeNode * n = q.front();
            cout << n->data << " ";
            q.pop();
            if (n->left != NULL) q.push(n->left);
            if (n->right != NULL) q.push(n->right);
        }
        break;
    }
}

void increaseindex(BiTreeNode *root, int length) {
    if (root == NULL) return;
    root->index += length;
    increaseindex(root->left,length);
    increaseindex(root->right,length);
}
```

```
int size(BiTreeNode *root) {
	if (root == NULL) return 0;
	return 1 + size(root->left) + size(root->right);
}

int Depth(BiTreeNode * root) {
	if (root == NULL) return 0;
	int depthleft = Depth(root->left);
	int depthright = Depth(root->right);
	int depth = depthleft > depthright ? depthleft : depthright;
	return depth + 1;
}

BiTreeNode *FindNode(BiTreeNode *root, int index) {
	if (root == NULL) return NULL;
	if (root->index == index) return root;
	else {
		BiTreeNode *left = FindNode(root->left,index);
		BiTreeNode *right = FindNode(root->right,index);
		if (left != NULL) return left;
		if (right != NULL) return right;
		return NULL;
	}
}

//----------------------------
//                  APIs
//----------------------------


/*
* Function Name: InitBiTree
* Parameter: BiTree &T
* Return: Status(int)
* Use: initialize the binary tree
*/
status InitBiTree(BiTree &T) {
	T.next = NULL;
	T.root = NULL;
	T.length = 0;
	return OK;
}

/*
* Function Name: DestroyBiTree
* Parameter: BiTree &T
* Return: Status(int)
* Use: destroy the binary tree
*/
status DestroyBiTree(BiTree &T) {
	FreeNodes(T.root);
	T.root = NULL;
	return OK;
}

/*
* Function Name: CreateBiTree
* Parameter: BiTree &T, int length, int *preorder, int *inorder, ElemType * data
```

```
* Return: Status(int)
* Use: create the tree using the defination data
*/
status CreateBiTree(BiTree &T, int length, int *preorder, int *inorder, ElemType * data) {
	T.root = Create(preorder, inorder, length, data);
	T.length = length;
	return OK;
}

/*
* Function Name: ClearBiTree
* Parameter: BiTree &T
* Return: Status(int)
* Use: clear the BiTree
*/
status ClearBiTree(BiTree &T) {
	if (T.root == NULL) return OK;
	FreeNodes(T.root->left);
	FreeNodes(T.root->right);
	free(T.root);
	T.root = NULL;
	T.length = 0;
	return OK;
}

/*
* Function Name: BiTreeEmpty
* Parameter: const BiTree &T
* Return: bool
* Use: check whether the tree is empty
*/
bool BiTreeEmpty(const BiTree &T) {
	if (T.root == NULL) return true;
	else return false;
}

/*
* Function Name: BiTreeDepth
* Parameter: const BiTree &T
* Return: int
* Use: calculate the depth of the tree.
*/
int	BiTreeDepth(const BiTree &T) {
	return Depth(T.root);
}

/*
* Function Name: Root
* Parameter: const BiTree &T
* Return: BiTreeNode *
* Use: return the root node of the tree.
*/
BiTreeNode *Root(const BiTree &T) {
	return T.root;
}

/*
* Function Name: Value
```

```
* Parameter: const BiTree &T, int index,ElemType &value
* Return: status
* Use: return the value of the node
*/
status Value(const BiTree &T, int index,ElemType &value) {
	BiTreeNode *node = FindNode(T.root, index);
	if (node == NULL) return ERROR;
	else
		value = node->data;
	return OK;
}

/*
* Function Name: Assign
* Parameter: BiTree &T, int index, ElemType &value
* Return: status
* Use: assign given value to given node
*/
status Assign(BiTree &T, int index, ElemType &value) {
	BiTreeNode *node = FindNode(T.root, index);
	if (node == NULL) return ERROR;
	else
		node->data = value;
	return OK;
}

/*
* Function Name: Parent
* Parameter: const BiTree &T, int index
* Return: BiTreeNode *
* Use: return the parent of the given node
*/
BiTreeNode *Parent(const BiTree &T, int index) {
	BiTreeNode *node = FindNode(T.root, index);
	if (node == NULL) return NULL;
	else
		return node->parent;

}

/*
* Function Name: LeftChild
* Parameter: const BiTree &T, int index
* Return: BiTreeNode *
* Use: return the LeftChild of the given node
*/
BiTreeNode *LeftChild(const BiTree &T, int index) {
	BiTreeNode *node = FindNode(T.root, index);
	if (node == NULL) return NULL;
	else
		return node->left;
}

/*
* Function Name: RightChild
* Parameter: const BiTree &T, int index
* Return: BiTreeNode *
* Use: return the RightChild of the given node
```

```
*/
BiTreeNode *RightChild(const BiTree &T, int index) {
	BiTreeNode *node = FindNode(T.root, index);
	if (node == NULL) return NULL;
	else
		return node->right;
}

/*
* Function Name: LeftSibling
* Parameter: const BiTree &T, int index
* Return: BiTreeNode *
* Use: return the LeftSibling of the given node
*/
BiTreeNode *LeftSibling(const BiTree &T, int index) {
	BiTreeNode *node = FindNode(T.root, index);
	if (node == NULL) return NULL;
	if (node->parent == NULL) return NULL;
	else {
		if (node->parent->left == node)
			return NULL;
		if (node->parent->right == node)
			return node->parent->left;
	}
}

/*
* Function Name: RightSibling
* Parameter: const BiTree &T, int index
* Return: BiTreeNode *
* Use: return the RightSibling of the given node
*/
BiTreeNode *RightSibling(const BiTree &T, int index) {
	BiTreeNode *node = FindNode(T.root, index);
	if (node == NULL) return NULL;
	if (node->parent == NULL) return NULL;
	else {
		if (node->parent->right == node)
			return NULL;
		if (node->parent->left == node)
			return node->parent->right;
	}
}

/*
* Function Name: InsertChild
* Parameter: BiTree &T, int index, int LR, BiTree &c
* Return: status
* Use: Insert the BiTree to the given node
*/
status	InsertChild(BiTree &T, int index, int LR, BiTree &c) {
	BiTreeNode *node = FindNode(T.root, index);
	if (node == NULL || c.root == NULL || c.root->right != NULL) return ERROR;
	if (LR == 0) { // left
		BiTreeNode *tmp = node->left;
		increaseindex(c.root, T.length);
		c.root->parent = node;
		node->left = c.root;
```

```
            if (tmp != NULL) {
                c.root->right = tmp;
                tmp->parent = c.root;
            }
            T.length += c.length;
            return OK;
        }
        if (LR == 1) {
            BiTreeNode *tmp = node->right;
            increaseindex(c.root, T.length);
            c.root->parent = node;
            node->right = c.root;
            if (tmp != NULL) {
                c.root->right = tmp;
                tmp->parent = c.root;
            }
            T.length += c.length;
            return OK;
        }
        else return ERROR;
}

/*
* Function Name: DeleteChild
* Parameter: BiTree &T, int index, int LR
* Return: status
* Use: delete the child tree of the given node
*/
status    DeleteChild(BiTree &T, int index, int LR) {
        BiTreeNode *node = FindNode(T.root, index);
        if (node == NULL) return ERROR;
        if (LR == 0) {
            if (node->left == NULL) return ERROR;
            T.length -= size(node->left);
            FreeNodes(node->left);
            node->left = NULL;
            return OK;
        }
        if (LR == 1) {
            if (node->right == NULL) return ERROR;
            T.length -= size(node->right);
            FreeNodes(node->right);
            node->right = NULL;
            return OK;
        }
        else return ERROR;
}

/*
* Function Name: PreOrderTraverse
* Parameter:const BiTree &T
* Return: Status(int)
* Use: pre order traverse the tree.
*/
status PreOrderTraverse(const BiTree &T) {
        Traverse(T.root, PRE);
        return OK;
}
```

```
/*
* Function Name: InOrderTraverse
* Parameter:const BiTree &T
* Return: Status(int)
* Use: in order traverse the tree.
*/
status InOrderTraverse(const BiTree &T) {
	Traverse(T.root, IN);
	return OK;
}

/*
* Function Name: PostOrderTraverse
* Parameter:const BiTree &T
* Return: Status(int)
* Use: post order traverse the tree.
*/
status PostOrderTraverse(const BiTree &T) {
	Traverse(T.root, POST);
	return OK;
}

/*
* Function Name: LevelOrderTraverse
* Parameter:const BiTree &T
* Return: Status(int)
* Use: level order traverse the tree.
*/
status LevelOrderTraverse(const BiTree &T) {
	Traverse(T.root, LEVEL);
	return OK;
}
```

Main.cpp:

```
#include "BiTree.h"

void PrintMenu(void) {
	/*
	* Function Name: PrintMenu
	* Parameter: None
	* Return: None
	* Use: Print the main menu
	*/

	printf("\n+-------------------------------------------------------+\n");
	printf("|                    *THE* BINARY TREE DEMO                    |\n");
	printf("|                                                              |\n");
	printf("|                          Functions                           |\n");
	printf("|                                                              |\n");
	printf("|      1.InitBiTree          2.DestroyBiTree      |\n");
	printf("|      3.CreateBiTree        4.ClearBiTree        |\n");
	printf("|      5.BiTreeEmpty         6.BiTreeDepth        |\n");
	printf("|      7.Root                8.Value              |\n");
	printf("|      9.Assign              10.Parent            |\n");
	printf("|     11.LeftChild           12.RightChild        |\n");
	printf("|     13.LeftSibling         14.RightSibling      |\n");
	printf("|     15.InsertChild         16.DeleteChild       |\n");
	printf("|     17.PreOrderTraverse    18.InOrderTraverse   |\n");
```

```
        printf("|         19.PostOrderTraverse          20.LevelOrderTraverse |\n");
        printf("|                                                              |\n");
        printf("|                                                              |\n");
        printf("|                                78ij@8102                     |\n");
        printf("|                                                              |\n");
        printf("+--------------------------------------------------------------+\n");
        printf("\n");

}

status LoadData(BiTree **head) {
        /*
        * Function Name: LoadData
        * Parameter: none
        * Return Status(int)
        * Use: load data from file
        */
        int *preorder = NULL, *inorder = NULL;
        ElemType *data = NULL;
        FILE *fp = fopen("SLDB", "r");
        if (fp == NULL)
                return ERROR;

        int size = 0;
        int count = 0;
        BiTree *tmp = (BiTree *)malloc(sizeof(BiTree));
        size = fread(tmp, sizeof(BiTree), 1, fp);
        if (size == 0) {
                free(tmp);
                return OK;
        }
        count++;
        size = tmp->length;
        if (size != 0) {
                preorder = (int *)malloc(size * sizeof(int));
                inorder = (int *)malloc(size * sizeof(int));
                data = (ElemType *)malloc(size * sizeof(ElemType));
                fread(preorder, sizeof(int), size, fp);
                fread(inorder, sizeof(int), size, fp);
                fread(data, sizeof(ElemType), size, fp);
                CreateBiTree(*tmp, size, preorder, inorder, data);
                free(preorder);
                free(inorder);
                free(data);
        }

       *head = tmp;

        while (1) {
                BiTree *tmp = (BiTree *)malloc(sizeof(BiTree));
                size = fread(tmp, sizeof(BiTree), 1, fp);
                if (size == 0) {
                        free(tmp);
                        break;
                }
                count++;
                size = tmp->length;
                if (size != 0) {
```

```
                preorder = (int *)malloc(size * sizeof(int));
                inorder = (int *)malloc(size * sizeof(int));
                data = (ElemType *)malloc(size * sizeof(ElemType));
                fread(preorder, sizeof(int), size, fp);
                fread(inorder, sizeof(int), size, fp);
                fread(data, sizeof(ElemType), size, fp);
                CreateBiTree(*tmp, size, preorder, inorder, data);
                free(preorder);
                free(inorder);
                free(data);
            }
            (*head)->next = tmp;
            *head = (*head)->next;
        }
        (*head)->next = NULL;
        *head = tmp;
        fclose(fp);
        return OK;
}

status SaveData(BiTree *head) {
        /*
        * Function Name: SaveData
        * Parameter: BiTree *heAD
        * Return: Status(int)
        * Use: save data to file
        */

        FILE *fp = fopen("SLDB", "w");
        if (fp == NULL)
            return ERROR;
        BiTree *L = head, *p = head;
        while (L != NULL) {
            fwrite(L, sizeof(BiTree), 1, fp);
            write(L->root, PRE, true, fp);
            write(L->root, IN, true, fp);
            write(L->root, IN, false, fp);
            p = L->next;
            DestroyBiTree(*L);
            //free(L);
            L = p;
        }

        fclose(fp);
        return OK;
}

int main() {
        int selection = -1;
        BiTree *head = NULL;
        while (selection != 0) {
            PrintMenu();
            scanf("%d", &selection);
            LoadData(&head);
            BiTree *L = head;
            BiTree *tmp = head;
            int tree_index;
            switch (selection) {
```

```
case -1:
        while (head != NULL) {
                printf("TreeID:%d\tTreelength:%d\t", head->TreeID, head->length);
                cout << "Preorder Traverse: ";
                PreOrderTraverse(*head);
                cout << endl;
                head = head->next;
        }
        head = L;
        break;
case 1:
        printf("* Function Name: InitBiTree\n");
        printf("* Parameter: BiTree &T\n");
        printf("* Return: Status(int)\n");
        printf("* Use: initialize the binary tree\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        while (head != NULL) {
                if (head->TreeID == tree_index)
                        break;
                head = head->next;
        }
        if (head != NULL) {
                printf("Error, the tree %d already exist.\n", tree_index);
        }
        else {
                BiTree *new_tree = (BiTree *)malloc(sizeof(BiTree));
                if (InitBiTree(*new_tree) == OK) {
                        printf("Inital the tree %d succeed.\n", tree_index);
                        new_tree->TreeID = tree_index;
                        new_tree->next = L;
                        head = new_tree;
                }
                else {
                        printf("ERROR, something wrong with the RAM\n");
                }
        }
        printf("\n");
        break;
case 2:
        printf("* Function Name: DestroyBiTree\n");
        printf("* Parameter: BiTree &T\n");
        printf("* Return: Status(int)\n");
        printf("* Use: destroy the binary tree\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        if (head == NULL) {
                printf("Error, the Tree %d does not exist.\n", tree_index);
                break;
        }
        if (head->TreeID == tree_index) {
                head = head->next;
                DestroyBiTree(*L);
                printf("Tree %d has been removed\n", tree_index);
                break;
        }
        while (head->next != NULL) {
                if (head->next->TreeID == tree_index)
```

```
                break;
            head = head->next;
        }
        if (head->next == NULL) {
            printf("Error, the tree %d does not exist.\n", tree_index);
            head = L;
        }
        else {
            L = head->next;
            head->next = head->next->next;
            DestroyBiTree(*L);
            printf("Tree %d has been removed\n", tree_index);
            head = tmp;
        }
        printf("\n");
        break;
    case 3:
        printf("* Function Name: CreateBiTree\n");
        printf("* Parameter: BiTree &T, int length, int *preorder, int *inorder, ElemType * data\n");
        printf("* Return: Status(int)\n");
        printf("* Use: create the tree using the defination data\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        while (head != NULL) {
            if (head->TreeID == tree_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the tree %d does not exist.\n", tree_index);
            head = L;
        }
        else {
            int length = 0;
            cout << "please insert the length" << endl;
            cin >> length;
            int *pre = (int *)malloc(sizeof(int) * length);
            int *in = (int *)malloc(sizeof(int) * length);
            ElemType *data = (ElemType *)malloc(sizeof(ElemType) * length);
            cout << "please insert the preorder index" << endl;
            for (int i = 0; i < length; i++) cin >> pre[i];
            cout << "please insert the inorder index" << endl;
            for (int i = 0; i < length; i++) cin >> in[i];
            cout << "please insert the inorder data" << endl;
            for (int i = 0; i < length; i++) cin >> data[i];
            ClearBiTree(*head);
            if (!isvalid(pre, in, length)) {
                cout << "the index is not valid." << endl;
                free(pre);
                free(in);
                free(data);
                head = L;
                break;
            }
            CreateBiTree(*head, length, pre, in, data);
            free(pre);
            free(in);
            free(data);
```

```
            head = L;
        }

        printf("\n");
        break;
    case 4:
        printf("* Function Name: ClearBiTree\n");
        printf("* Parameter: BiTree &T\n");
        printf("* Return: Status(int)\n");
        printf("* Use: clear the BiTree\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        while (head != NULL) {
            if (head->TreeID == tree_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the tree %d does not exist.\n", tree_index);
            head = L;
        }
        else {
            ClearBiTree(*head);
            head = L;
            printf("The Tree %d has been cleared.\n", tree_index);
        }
        break;
    case 5:
        printf("* Function Name: BiTreeEmpty\n");
        printf("* Parameter: const BiTree &T\n");
        printf("* Return: bool\n");
        printf("* Use: check whether the tree is empty\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        while (head != NULL) {
            if (head->TreeID == tree_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the tree %d does not exist.\n", tree_index);
            head = L;
        }
        else {
            bool empty = BiTreeEmpty(*head);
            head = L;
            if (empty) {
                printf("The tree %d is empty!", tree_index);
            }
            else {
                printf("The tree %d is not empty", tree_index);
            }
        }
        break;
    case 6:
        printf("* Function Name: BiTreeDepth\n");
        printf("* Parameter: const BiTree &T\n");
        printf("* Return: int\n");
```

```
        printf("* Use: calculate the depth of the tree.\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        while (head != NULL) {
            if (head->TreeID == tree_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the tree %d does not exist.\n", tree_index);
            head = L;
        }
        else {
            printf("The depth of tree %d is %d", tree_index, BiTreeDepth(*head));
            head = L;
        }
        break;
    case 7:
        printf("* Function Name: Root\n");
        printf("* Parameter: const BiTree &T\n");
        printf("* Return: BiTreeNode *\n");
        printf("* Use: return the root node of the tree.\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        while (head != NULL) {
            if (head->TreeID == tree_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the tree %d does not exist.\n", tree_index);
            head = L;
        }
        else {
            BiTreeNode * root = Root(*head);
            head = L;
            if (root != NULL)
                printf("The index of the root is %d,the data is %d\n", root->index, root->data);
            else
                printf("The root is empty!");
        }
        break;
    case 8:
        printf("* Function Name: Value\n");
        printf("* Parameter: const BiTree &T, int index,ElemType &value\n");
        printf("* Return: status\n");
        printf("* Use: return the value of the node\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        while (head != NULL) {
            if (head->TreeID == tree_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the tree %d does not exist.\n", tree_index);
            head = L;
        }
```

```
            else {
                ElemType value;
                int index = 0;
                cout << "Please insert the desired index!" << endl;
                cin >> index;
                if (Value(*head, index, value) == OK) {
                    cout << "The value is " << value << endl;
                }
                else
                    cout << "Sorry, we encounter an error." << endl;
            }
            head = L;
            break;
        case 9:
            printf("* Function Name: Assign\n");
            printf("* Parameter: BiTree &T, int index, ElemType &value\n");
            printf("* Return: status\n");
            printf("* Use: assign given value to given node\n");
            printf("please enter the id of the tree:");
            scanf("%d", &tree_index);
            while (head != NULL) {
                if (head->TreeID == tree_index)
                    break;
                head = head->next;
            }
            if (head == NULL) {
                printf("Error, the tree %d does not exist.\n", tree_index);
                head = L;
            }
            else {
                ElemType value;
                int index = 0;
                cout << "Please insert the desired index!" << endl;
                cin >> index;
                cout << "Please insert the desired value!" << endl;
                cin >> value;
                if (Assign(*head, index, value) == OK) {
                    cout << "The value " << value << "is successfully inserted into the node "<< index <<
endl;
                }
                else
                    cout << "Sorry, we encounter an error." << endl;
            }
            head = L;
            break;
        case 10:
            printf("* Function Name: Parent\n");
            printf("* Parameter: const BiTree &T, int index\n");
            printf("* Return: BiTreeNode *\n");
            printf("* Use: return the parent of the given node \n");
            printf("please enter the id of the tree:");
            scanf("%d", &tree_index);
            while (head != NULL) {
                if (head->TreeID == tree_index)
                    break;
                head = head->next;
            }
            if (head == NULL) {
```

```
                printf("Error, the tree %d does not exist.\n", tree_index);
                head = L;
            }
            else {
                BiTreeNode *value;
                int index = 0;
                cout << "Please insert the desired index!" << endl;
                cin >> index;
                value = Parent(*head, index);
                if (value != NULL){
                    cout << "The parent data is " << value->data << " and the index is" << value->index
<< endl;
                }
                else
                    cout << "Sorry, we encounter an error." << endl;
            }
            head = L;
            break;
        case 11:
            printf("* Function Name: LeftChild\n");
            printf("* Parameter: const BiTree &T, int index\n");
            printf("* Return: BiTreeNode *\n");
            printf("* Use: return the LeftChild of the given node\n");
            printf("please enter the id of the tree:");
            scanf("%d", &tree_index);
            while (head != NULL) {
                if (head->TreeID == tree_index)
                    break;
                head = head->next;
            }
            if (head == NULL) {
                printf("Error, the tree %d does not exist.\n", tree_index);
                head = L;
            }
            else {
                BiTreeNode *value;
                int index = 0;
                cout << "Please insert the desired index!" << endl;
                cin >> index;
                value = LeftChild(*head, index);
                if (value != NULL) {
                    cout << "The left child data is " << value->data << " and the index is" << value->index
<< endl;
                }
                else
                    cout << "Sorry, we encounter an error." << endl;
            }
            head = L;
            break;
        case 12:
            printf("* Function Name: RightChild\n");
            printf("* Parameter: const BiTree &T, int index\n");
            printf("* Return: BiTreeNode *\n");
            printf("* Use: return the RightChild of the given node\n");
            printf("please enter the id of the tree:");
            scanf("%d", &tree_index);
            while (head != NULL) {
                if (head->TreeID == tree_index)
```

```
                    break;
                head = head->next;
            }
            if (head == NULL) {
                printf("Error, the tree %d does not exist.\n", tree_index);
                head = L;
            }
            else {
                BiTreeNode *value;
                int index = 0;
                cout << "Please insert the desired index!" << endl;
                cin >> index;
                value = RightChild(*head, index);
                if (value != NULL) {
                    cout << "The right child data is " << value->data << " and the index is" << value->index
<< endl;
                }
                else
                    cout << "Sorry, we encounter an error." << endl;
            }
            head = L;
            break;
        case 13:
            printf("* Function Name: LeftSibling\n");
            printf("* Parameter: const BiTree &T, int index\n");
            printf("* Return: BiTreeNode *\n");
            printf("* Use: return the LeftSibling of the given node\n");
            printf("please enter the id of the tree:");
            scanf("%d", &tree_index);
            while (head != NULL) {
                if (head->TreeID == tree_index)
                    break;
                head = head->next;
            }
            if (head == NULL) {
                printf("Error, the tree %d does not exist.\n", tree_index);
                head = L;
            }
            else {
                BiTreeNode *value;
                int index = 0;
                cout << "Please insert the desired index!" << endl;
                cin >> index;
                value = LeftSibling(*head, index);
                if (value != NULL) {
                    cout << "The left sibling data is " << value->data << " and the index is" <<
value->index << endl;
                }
                else
                    cout << "Sorry, we encounter an error." << endl;
            }
            head = L;
            break;
        case 14:
            printf("* Function Name: RightSibling\n");
            printf("* Parameter: const BiTree &T, int index\n");
            printf("* Return: BiTreeNode *\n");
            printf("* Use: return the RightSibling of the given node\n");
```

```
                printf("please enter the id of the tree:");
                scanf("%d", &tree_index);
                while (head != NULL) {
                        if (head->TreeID == tree_index)
                                break;
                        head = head->next;
                }
                if (head == NULL) {
                        printf("Error, the tree %d does not exist.\n", tree_index);
                        head = L;
                }
                else {
                        BiTreeNode *value;
                        int index = 0;
                        cout << "Please insert the desired index!" << endl;
                        cin >> index;
                        value = RightSibling(*head, index);
                        if (value != NULL) {
                                cout << "The right sibling data is " << value->data << " and the index is" <<
value->index << endl;
                        }
                        else
                                cout << "Sorry, we encounter an error." << endl;
                }
                head = L;
                break;
        case 15:
                printf("* Function Name: InsertChild\n");
                printf("* Parameter: BiTree &T, int index, int LR, BiTree &c\n");
                printf("* Return: status\n");
                printf("* Use: Insert the BiTree to the given node \n");
                printf("please enter the id of the tree:");
                scanf("%d", &tree_index);
                while (head != NULL) {
                        if (head->TreeID == tree_index)
                                break;
                        head = head->next;
                }
                if (head == NULL) {
                        printf("Error, the tree %d does not exist.\n", tree_index);
                        head = L;
                }
                else {
                        int instree_index = 0;
                        cout << "please enter the id of the inserted tree" << endl;
                        cin >> instree_index;
                        BiTree *head2 = L;
                        BiTree *pre = L;
                        while (head2 != NULL) {
                                if (pre->next->TreeID != instree_index) pre = pre->next;
                                if (head2->TreeID == instree_index)
                                        break;
                                head2 = head2->next;
                        }
                        if (head2 == NULL) {
                                printf("Error, the tree %d does not exist.\n", tree_index);
                                break;
                        }
```

```
            else {
                int index = 0, LR = 0;
                cout << "please insert the node index" << endl;
                cin >> index;
                cout << "L or R? (L = 0 R = 1)" << endl;
                cin >> LR;
                if (LR != 0 && LR != 1)
                {
                    cout << "invalid input." << endl;
                    head = L;
                    break;
                }
                if (InsertChild(*head, index, LR, *head2) != OK)
                    cout << "Sorry, we encounter an error." << endl ;
                else {
                    cout << "insert complete." << endl;
                    if(L != head2)
                        pre->next = pre->next->next;
                    else {
                        L = head2->next;
                        head = L;
                    }
                }
            }
        }
        head = L;
        break;
    case 16:
        printf("* Function Name: DeleteChild\n");
        printf("* Parameter: BiTree &T, int index, int LR\n");
        printf("* Return: status\n");
        printf("* Use: delete the child tree of the given node\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        while (head != NULL) {
            if (head->TreeID == tree_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the tree %d does not exist.\n", tree_index);
            head = L;
        }
        else {
            int index = 0;
            int LR = 0;
            cout << "please insert the desiredc index" << endl;
            cin >> index;
            cout << "L or R? (L = 0 R = 1)" << endl;
            cin >> LR;
            if (DeleteChild(*head, index, LR) == OK) {
                cout << "delete complete." << endl;
            }
            else {
                cout << "Sorry, we encounter an error." << endl;
            }
        }
        head = L;
```

```
        break;
    case 17:
        printf("* Function Name: PreOrderTraverse\n");
        printf("* Parameter:const BiTree &T\n");
        printf("* Return: Status(int)\n");
        printf("* Use: pre order traverse the tree.\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        while (head != NULL) {
            if (head->TreeID == tree_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the tree %d does not exist.\n", tree_index);
            head = L;
        }
        else {
            cout << "The pre-order traverse of the tree" << tree_index << " is:" << endl;
            PreOrderTraverse(*head);
        }
        head = L;
        break;
    case 18:
        printf("* Function Name: InOrderTraverse\n");
        printf("* Parameter:const BiTree &T\n");
        printf("* Return: Status(int)\n");
        printf("* Use: in order traverse the tree.\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        while (head != NULL) {
            if (head->TreeID == tree_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the tree %d does not exist.\n", tree_index);
            head = L;
        }
        else {
            cout << "The in-order traverse of the tree" << tree_index << " is:" << endl;
            InOrderTraverse(*head);
        }
        head = L;
        break;
    case 19:
        printf("* Parameter:const BiTree &T\n");
        printf("* Return: Status(int)\n");
        printf("* Use: in order traverse the tree.\n");
        printf("please enter the id of the tree:");
        scanf("%d", &tree_index);
        while (head != NULL) {
            if (head->TreeID == tree_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the tree %d does not exist.\n", tree_index);
```

```
                    head = L;
                }
                else {
                    cout << "The post-order traverse of the tree" << tree_index << " is:" << endl;
                    PostOrderTraverse(*head);
                }
                head = L;
                break;
            case 20:
                printf("* Function Name: LevelOrderTraverse\n");
                printf("* Parameter:const BiTree &T\n");
                printf("* Return: Status(int)\n");
                printf("* Use: level order traverse the tree.\n");
                printf("please enter the id of the tree:");
                scanf("%d", &tree_index);
                while (head != NULL) {
                    if (head->TreeID == tree_index)
                        break;
                    head = head->next;
                }
                if (head == NULL) {
                    printf("Error, the tree %d does not exist.\n", tree_index);
                    head = L;
                }
                else {
                    cout << "The level-order traverse of the tree" << tree_index << " is:" << endl;
                    LevelOrderTraverse(*head);
                }
                head = L;
                break;
            case 0:
                cout << "Thanks for using.";
                break;
            default:
                cout << "No such selection.";
                break;
            }
            SaveData(head);
        }
}
```

Graph.h:

```
#ifndef GRAPH_H
#define GRAPH_H

#include "LinkedList.h"
#include "common.h"

//ÓÐÏòÎÞÈ¨Í¼
typedef struct GNode {
    int nodeindex = 0;
    ElemType nodedata = 0;
    bool visited = false;// used for dfs/bfs
};

typedef struct Graph {
    vector<std::pair<GNode,LinkedList>> data;
    int vexs = 0;
    int GraphID = -1;
```

```
    Graph *next;
} Graph;


void     write(Graph &G, FILE *fp);
status CreateGraph(Graph &G, ElemType *Nodedata, int *matrix, int length);
status DestroyGraph(Graph &G);
status LocateVex(Graph &G, ElemType data);
status GetVex(Graph &G, int index);
status PutVex(Graph &G, int index, ElemType &data);
status FirstAdjVex(Graph &G, int index);
status NextAdjVex(Graph &G, int index, int &adj);
status InsertVex(Graph &G, ElemType data);
status DeleteVex(Graph &G, int index);
status InsertArc(Graph &G, int v, int w);
status DeleteArc(Graph &G, int v, int w);
status DFSTraverse(Graph &G);
status BFSTraverse(Graph &G);

#endif
```

Graph.cpp:

```
#include "Graph.h"

void write(Graph &G, FILE *fp) {
    for (int i = 0; i < G.vexs; i++) {
        for (int j = 0; j < G.vexs; j++) {
            LinkedList tmp = G.data[i].second;
            if (LocateElem(tmp, j) != 0) {
                int t = 1;
                fwrite(&t, sizeof(int), 1, fp);
            }
            else {
                int t = -1;
                fwrite(&t, sizeof(int), 1, fp);
            }
        }
    }
    for (int i = 0; i < G.vexs;    i++) {
        fwrite(&(G.data[i].first.nodedata), sizeof(ElemType), 1, fp);
    }

}

/*
* Function Name: CreateGraph
* Parameter: Graph &G, ElemType *Nodedata, int *matrix, int length
* Return: Status(int)
* Use: create a graph.
*/
status CreateGraph(Graph &G, ElemType *Nodedata, int *matrix, int length) {
    G.vexs = length;
    for (int i = 0; i < length; i++) {
        GNode node{ i,*(Nodedata + i) };
        LinkedList list;
        IntiaList(list);
        G.data.push_back(std::make_pair(node, list));
    }
    for (int i = 0; i < length; i++) {
```

```
            for (int j = 0; j < length; j++) {
                if (*(matrix + length * i + j) != -1)
                    Pushback(G.data[i].second, j);
            }
        }
        return OK;
}

/*
* Function Name: DestroyGraph
* Parameter: Graph &G
* Return: Status(int)
* Use: destroy a graph.
*/
status DestroyGraph(Graph &G) {
        for (int i = 0; i < G.data.size(); i++) {
            ClearList(G.data[i].second);
        }
        G.data.clear();
        return OK;
}

/*
* Function Name: LocateVex
* Parameter: Graph &G,int index
* Return: Status(int)
* Use: locate the desired node.
*/
status LocateVex(Graph &G, ElemType data) {
        int i;
        for (i = 0; i < G.vexs; i++) {
            if (G.data[i].first.nodedata == data) {
                cout << "index: " << G.data[i].first.nodeindex << endl;
                break;
            }
        }
        if (i == G.vexs) return ERROR;
        cout << endl;
        return OK;
}

/*
* Function Name: GetVex
* Parameter: Graph &G,int index
* Return: Status(int)
* Use: get the data of the desired vertex.
*/
status GetVex(Graph &G, int index) {
        if (index < 0 || index >= G.data.size()) return ERROR;
        cout << "Data: " << G.data[index].first.nodedata << endl;
        return OK;
}

/*
* Function Name: PutVex
* Parameter: Graph &G,int index,ElemType &data
* Return: Status(int)
* Use: assign the desired node a value
```

```
*/
status PutVex(Graph &G, int index, ElemType &data) {
	if (index < 0 || index >= G.data.size()) return ERROR;
	G.data[index].first.nodedata = data;
	return OK;
}

/*
* Function Name: FirstAdjVex
* Parameter: Graph &G,int index
* Return: Status(int)
* Use: find the first adjacent vertex
*/
status FirstAdjVex(Graph &G, int index) {
	if (index < 0 || index >= G.data.size()) return ERROR;
	if (ListLength(G.data[index].second) == 0) return ERROR;
	cout << "First Adjacent Vertex: ";
	ElemType e;
	GetElem(G.data[index].second, 1, e);
	cout << e << endl;
	return OK;
}

/*
* Function Name: NextAdjVex
* Parameter: Graph &G, int index, int &adj
* Return: Status(int)
* Use: find the next adjacent vertex
*/
status NextAdjVex(Graph &G, int index, int &adj) {
	if (index < 0 || index >= G.data.size()) return ERROR;
	if (adj < 0 || adj <= G.data.size()) return ERROR;
	int i = LocateElem(G.data[index].second, adj);
	int next;
	if (NextElem(G.data[index].second, i, next) != ERROR) {
		cout << "Next is : " << next;
		return OK;
	}
	return ERROR;
}

/*
* Function Name: InsertVex
* Parameter: Graph &G, ElemType data
* Return: Status(int)
* Use: insert a vertex
*/
status InsertVex(Graph &G, ElemType data) {
	G.vexs++;
	int i = G.data.size();
	GNode node{ i,data };
	LinkedList list;
	IntiaList(list);
	G.data.push_back(std::make_pair(node, list));
	return OK;
}

/*
```

```
* Function Name: DeleteVex
* Parameter: Graph &G, int index
* Return: Status(int)
* Use: delete the desired vertex
*/
status DeleteVex(Graph &G, int index) {
	if (index > G.data.size() - 1 || index < 0) return ERROR;
	G.vexs--;
	DestroyList(G.data[index].second);
	G.data.erase(G.data.begin() + index);
	for (int i = 0; i < G.data.size(); i++) {
		if (G.data[i].first.nodeindex > index) G.data[i].first.nodeindex--;
		for (int j = 1;; j++) {
			ElemType p;
			GetElem(G.data[i].second, j, p);
			if (p > index) p--;
			if (p == index) break;
			if (j > ListLength(G.data[i].second)) break;
		}
	}
	return OK;
}

/*
* Function Name: InsertArc
* Parameter: Graph &G, int v, int w
* Return: Status(int)
* Use: insert a specified arc
*/
status InsertArc(Graph &G, int v, int w){
	if (v < 0 || w < 0) return ERROR;
	if (v == w) return ERROR;
	if (v > G.data.size() - 1 || w > G.data.size() - 1) return ERROR;
	ElemType p;
	for (int j = 1; j <= ListLength(G.data[v].second); j++) {
		GetElem(G.data[v].second, j, p);
		if (p == w) return ERROR;
	}
	ListInsert(G.data[v].second, ListLength(G.data[v].second) + 1, w);
	return OK;
}

/*
* Function Name: DeleteArc
* Parameter: Graph &G, int v, int w
* Return: Status(int)
* Use: delete a specified arc
*/
status DeleteArc(Graph &G, int v, int w) {
	if (v < 0 || w < 0) return ERROR;
	if (v == w) return ERROR;
	if (v > G.data.size() - 1 || w > G.data.size() - 1) return ERROR;
	ElemType p;
	int j;
	for (j = 1; j <= ListLength(G.data[v].second); j++) {
		GetElem(G.data[v].second, j, p);
		if (p == w) break;
	}
```

```
    if (j == ListLength(G.data[v].second) + 1) return ERROR;
    else {
        ListDelete(G.data[v].second, j,p);
        return OK;
    }
}

/*
* Function Name: DFSTraverse
* Parameter: Graph &G
* Return: Status(int)
* Use: DFS Traverse
*/
status DFSTraverse(Graph &G) {
    stack<GNode> s;
    for (int i = 0; i < G.data.size(); i++)
        G.data[i].first.visited = false;
    for (int i = 0; i < G.data.size(); i++) {
        if (G.data[i].first.visited == false) {
            cout << G.data[i].first.nodedata << "(" << G.data[i].first.nodeindex << ")" << " ";
            s.push(G.data[i].first);
            G.data[s.top().nodeindex].first.visited = true;
        }
        while (!s.empty()) {
            GNode top = s.top();
            int j;
            for (j = 1; j <= ListLength(G.data[top.nodeindex].second); j++) {
                ElemType p;
                GetElem(G.data[top.nodeindex].second, j, p);
                if (G.data[p].first.visited == false) {
                    s.push(G.data[p].first);
                    cout << G.data[p].first.nodedata << "(" << G.data[p].first.nodeindex << ")" << " ";
                    G.data[G.data[p].first.nodeindex].first.visited = true;
                    break;
                }
            }
            if (j > ListLength(G.data[top.nodeindex].second)) s.pop();
        }
    }
    return OK;
}

/*
* Function Name: BFSTraverse
* Parameter: Graph &G
* Return: Status(int)
* Use: BFS Traverse
*/
status BFSTraverse(Graph &G) {
    queue<GNode> q;
    for (int i = 0; i < G.data.size(); i++)
        G.data[i].first.visited = false;
    for (int i = 0; i < G.data.size(); i++) {
        if (G.data[i].first.visited == false) {
            cout << G.data[i].first.nodedata << "(" << G.data[i].first.nodeindex << ")" << " "; //output
format:data(index)
            q.push(G.data[i].first);
            G.data[i].first.visited = true;
```

```
            }
            while (!q.empty()) {
                GNode top = q.front();
                q.pop();
                for (int j = 1; j <= ListLength(G.data[top.nodeindex].second); j++) {
                    ElemType p;
                    GetElem(G.data[top.nodeindex].second, j, p);
                    if (G.data[p].first.visited == false) {
                        q.push(G.data[p].first);
                        cout << G.data[p].first.nodedata << "(" << G.data[p].first.nodeindex << ")" << " ";
                        G.data[G.data[p].first.nodeindex].first.visited = true;
                    }
                }
            }
        }
        return OK;
}
```

Main.cpp:

```
#include "Graph.h"

void PrintMenu(void) {
        /*
        * Function Name: PrintMenu
        * Parameter: None
        * Return: None
        * Use: Print the main menu
        */

        printf("\n+-----------------------------------------------------+\n");
        printf("|                        *THE* DAG DEMO                       |\n");
        printf("|                                                             |\n");
        printf("|                          Functions                          |\n");
        printf("|                                                             |\n");
        printf("|        1.CreateGraph              2.DestroyGraph          |\n");
        printf("|        3.LocateVex                4.GetVex                |\n");
        printf("|        5.PutVex                   6.FirstAdjVex           |\n");
        printf("|        7.NextAdjVex               8.InsertVex             |\n");
        printf("|        9.DeleteVex                10.InsertArc            |\n");
        printf("|       11.DeleteArc                12.DFSTraverse          |\n");
        printf("|       13.BFSTraverse                                      |\n");
        printf("|                                                             |\n");
        printf("|                           0.Exit                            |\n");
        printf("|                                                             |\n");
        printf("|                           78ij@8102                         |\n");
        printf("|                                                             |\n");
        printf("+-----------------------------------------------------+\n");
        printf("\n");

}

status LoadData(Graph **head) {
        /*
        * Function Name: LoadData
        * Parameter: none
        * Return Status(int)
        * Use: load data from file
        */
        int *matrix = NULL;
```

```
ElemType *nodedata = NULL;
FILE *fp = fopen("SLDB", "r");
if (fp == NULL)
	return ERROR;

int size = 0;
int count = 0;
Graph *tmp = new Graph();
int ID = 0;
size = fread(&ID, sizeof(int), 1, fp);
if (size == 0) {
	free(tmp);
	return OK;
}
tmp->GraphID = ID;
size = fread(&ID, sizeof(int), 1, fp);
tmp->vexs = ID;
count++;
size = tmp->vexs;
if (size != 0) {
	matrix = (int *)malloc(size * size    * sizeof(int));
	nodedata = (ElemType *)malloc(size * sizeof(ElemType));
	fread(matrix, sizeof(int), size * size, fp);
	fread(nodedata, sizeof(int), size, fp);
	CreateGraph(*tmp, nodedata, matrix,size);
	free(matrix);
	free(nodedata);
}

*head = tmp;

while (1) {
	Graph *tmp = new Graph();
	size = fread(&ID, sizeof(int), 1, fp);
	if (size == 0) {
		free(tmp);
		break;
	}
	tmp->GraphID = ID;
	size = fread(&ID, sizeof(int), 1, fp);
	tmp->vexs = ID;
	count++;
	size = tmp->vexs;
	if (size != 0) {
		matrix = (int *)malloc(size * size * sizeof(int));
		nodedata = (ElemType *)malloc(size * sizeof(ElemType));
		fread(matrix, sizeof(int), size * size, fp);
		fread(nodedata, sizeof(int), size, fp);
		CreateGraph(*tmp, nodedata, matrix, size);
		free(matrix);
		free(nodedata);
	}
	(*head)->next = tmp;
	*head = (*head)->next;
}
(*head)->next = NULL;
*head = tmp;
fclose(fp);
```

```
    return OK;
}

status SaveData(Graph *head) {
    /*
    * Function Name: SaveData
    * Parameter: BiTree *heAD
    * Return: Status(int)
    * Use: save data to file
    */

    FILE *fp = fopen("SLDB", "w");
    if (fp == NULL)
        return ERROR;
    Graph *L = head, *p = head;
    while (L != NULL) {
        fwrite(&(L->GraphID), sizeof(int), 1, fp);
        fwrite(&(L->vexs), sizeof(int), 1, fp);
        write(*L,fp);
        p = L->next;
        DestroyGraph(*L);
        delete(L);
        L = p;
    }

    fclose(fp);
    return OK;
}


int main() {
    int selection = -1;
    Graph *head = NULL;
    while (selection != 0) {
        PrintMenu();
        scanf("%d", &selection);
        LoadData(&head);
        Graph *L = head;
        Graph *tmp = head;
        int graph_index;
        switch (selection) {
        case -1:
            while (head != NULL) {
                printf("GraphID:%d\tVexs:%d\t", head->GraphID, head->vexs);
                cout << "DFS Traverse: ";
                DFSTraverse(*head);
                cout << endl;
                head = head->next;
            }
            head = L;
            break;
        case 1:
            printf("* Function Name: CreateGraph\n");
            printf("* Parameter: Graph &G, ElemType *Nodedata, int *matrix, int length\n");
            printf("* Return: Status(int)\n");
            printf("* Use: create a graph.\n");
            printf("please enter the id of the graph:");
            scanf("%d", &graph_index);
```

```
        while (head != NULL) {
            if (head->GraphID == graph_index)
                break;
            head = head->next;
        }
        if (head != NULL) {
            printf("Error, the tree %d already exist.\n", graph_index);
        }
        else {
            Graph *new_graph = new Graph();
            new_graph->data = vector<std::pair<GNode, LinkedList>>();
            int length;
            cout << "Please insert the graph vertex count:" << endl;
            cin >> length;
            int *matrix = (int *)malloc(sizeof(int) * length * length);
            ElemType *nodedata = (ElemType *)malloc(sizeof(ElemType) * length);
            cout << "Please insert the graph adjacent matrix:" << endl;
            for (int i = 0; i < length * length; i++) {
                cin >> matrix[i];
            }
            cout << "Please insert the node data" << endl;
            for (int i = 0; i < length; i++) {
                cin >> nodedata[i];
            }
            if (CreateGraph(*new_graph, nodedata, matrix, length) == OK) {
                printf("Creation the graph %d succeed.\n", graph_index);
                new_graph->GraphID = graph_index;
                new_graph->next = L;
                head = new_graph;
            }
            else {
                printf("ERROR, something wrong with the RAM\n");
            }
            free(matrix);
            free(nodedata);
        }

        printf("\n");
        break;
    case 2:
        printf("* Function Name: DestroyGraph\n");
        printf("* Parameter: Graph &G\n");
        printf("* Return: Status(int)\n");
        printf("* Use: destroy a graph.\n");
        printf("please enter the id of the graph:");
        scanf("%d", &graph_index);
        if (head == NULL) {
            printf("Error, the Graph %d does not exist.\n", graph_index);
            break;
        }
        if (head->GraphID == graph_index) {
            head = head->next;
            DestroyGraph(*L);
            printf("Graph %d has been removed\n", graph_index);
            break;
        }
        while (head->next != NULL) {
            if (head->next->GraphID == graph_index)
```

```
                break;
            head = head->next;
        }
        if (head->next == NULL) {
            printf("Error, the graph %d does not exist.\n", graph_index);
            head = L;
        }
        else {
            L = head->next;
            head->next = head->next->next;
            DestroyGraph(*L);
            printf("Graph %d has been removed\n", graph_index);
            head = tmp;
        }
        printf("\n");
        break;
    case 3:
        printf("* Function Name: LocateVex\n");
        printf("* Parameter: Graph &G,int index\n");
        printf("* Return: Status(int)\n");
        printf("* Use: locate the desired node.\n");
        printf("please enter the id of the graph:");
        scanf("%d", &graph_index);
        while (head != NULL) {
            if (head->GraphID == graph_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the graph %d does not exist.\n", graph_index);
            head = L;
        }
        else {
            int index;
            cout << "please insert the index" << endl;
            cin >> index;
            if (LocateVex(*head, index) != OK) {
                cout << "sorry, we encounter an ERROR." << endl;
            }
            head = L;
        }
        printf("\n");
        break;
    case 4:
        printf("* Function Name: GetVex\n");
        printf("* Parameter: Graph &G,int index\n");
        printf("* Return: Status(int)\n");
        printf("* Use: get the data of the desired vertex.\n");
        printf("please enter the id of the graph:");
        scanf("%d", &graph_index);
        while (head != NULL) {
            if (head->GraphID == graph_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the graph %d does not exist.\n", graph_index);
            head = L;
```

```
        }
        else {
            ElemType data;
            cout << "please insert the data" << endl;
            cin >> data;
            if (GetVex(*head, data) != OK) {
                cout << "sorry, we encounter an ERROR." << endl;
            }
            head = L;
        }
        printf("\n");
        break;
    case 5:
        printf("* Function Name: PutVex\n");
        printf("* Parameter: Graph &G,int index,ElemType &data\n");
        printf("* Return: Status(int)\n");
        printf("* Use: assign the desired node a value\n");
        printf("please enter the id of the graph:");
        scanf("%d", &graph_index);
        while (head != NULL) {
            if (head->GraphID == graph_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the graph %d does not exist.\n", graph_index);
            head = L;
        }
        else {
            int index;
            ElemType data;
            cout << "please insert the index" << endl;
            cin >> index;
            cout << "please insert the data." << endl;
            cin >> data;
            if (PutVex(*head, index, data) == OK) {
                cout << "successfully insert data " << data << "to vertex index " << index << endl;
            }
            else {
                cout << "sorry, we encounter an ERROR." << endl;
            }
            head = L;
        }
        printf("\n");
        break;
    case 6:
        printf("* Function Name: FirstAdjVex\n");
        printf("* Parameter: Graph &G,int index\n");
        printf("* Return: Status(int)\n");
        printf("* Use: find the first adjacent vertex\n");
        printf("please enter the id of the graph:");
        scanf("%d", &graph_index);
        while (head != NULL) {
            if (head->GraphID == graph_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
```

```
            printf("Error, the graph %d does not exist.\n", graph_index);
            head = L;
        }
        else {
            int index;
            cout << "please insert the index" << endl;
            cin >> index;
            if (FirstAdjVex(*head,index) != OK) {
                cout << "sorry, we encounter an ERROR." << endl;
            }
            head = L;
        }
        printf("\n");
        break;
    case 7:
        printf("* Function Name: NextAdjVex\n");
        printf("* Parameter: Graph &G, int index, int &adj\n");
        printf("* Return: Status(int)\n");
        printf("* Use: find the next adjacent vertex\n");
        printf("please enter the id of the graph:");
        scanf("%d", &graph_index);
        while (head != NULL) {
            if (head->GraphID == graph_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the graph %d does not exist.\n", graph_index);
            head = L;
        }
        else {
            int index,w;
            cout << "please insert the index" << endl;
            cin >> index;
            cout << "please insert the adjacent index" << endl;
            cin >> w;
            if (NextAdjVex(*head, index,w) != OK) {
                cout << "sorry, we encounter an ERROR." << endl;
            }
            head = L;
        }
        printf("\n");
        break;
    case 8:
        printf("* Function Name: InsertVex\n");
        printf("* Parameter: Graph &G, ElemType data\n");
        printf("* Return: Status(int)\n");
        printf("* Use: insert a vertex\n");
        printf("please enter the id of the graph:");
        scanf("%d", &graph_index);
        while (head != NULL) {
            if (head->GraphID == graph_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the graph %d does not exist.\n", graph_index);
            head = L;
```

```
        }
        else {
            ElemType data;
            cout << "please insert the data" << endl;
            cin >> data;
            if (InsertVex(*head, data) == OK) {
                cout << "node with data " << data << "has been successfully inserted." << endl;
            }
            head = L;
        }
        printf("\n");
        break;
    case 9:
        printf("* Function Name: DeleteVex\n");
        printf("* Parameter: Graph &G, int index\n");
        printf("* Return: Status(int)\n");
        printf("* Use: delete the desired vertex\n");
        printf("please enter the id of the graph:");
        scanf("%d", &graph_index);
        while (head != NULL) {
            if (head->GraphID == graph_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the graph %d does not exist.\n", graph_index);
            head = L;
        }
        else {
            int index;
            cout << "please insert the index" << endl;
            cin >> index;
            if (DeleteVex(*head, index) == OK) {
                cout << "node " << index << "has been successfully deleted." << endl;
            }
            else {
                cout << "sorry, we encounter an ERROR." << endl;
            }
            head = L;
        }
        printf("\n");
        break;
    case 10:
        printf("* Function Name: InsertArc\n");
        printf("* Parameter: Graph &G, int v, int w\n");
        printf("* Return: Status(int)\n");
        printf("* Use: insert a specified arc\n");
        printf("please enter the id of the graph:");
        scanf("%d", &graph_index);
        while (head != NULL) {
            if (head->GraphID == graph_index)
                break;
            head = head->next;
        }
        if (head == NULL) {
            printf("Error, the graph %d does not exist.\n", graph_index);
            head = L;
        }
```

```
                else {
                        int v,w;
                        cout << "please insert the index1" << endl;
                        cin >> v;
                        cout << "please insert the index2" << endl;
                        cin >> w;
                        if (InsertArc(*head, v,w) == OK) {
                                cout << "arc with nodes " << v << "and " << w << " has been successfully inserted."
<< endl;
                        }
                        else {
                                cout << "sorry, we encounter an ERROR." << endl;
                        }
                        head = L;
                }
                printf("\n");
                break;
        case 11:
                printf("* Function Name: DeleteArc\n");
                printf("* Parameter: Graph &G, int v, int w\n");
                printf("* Return: Status(int)\n");
                printf("* Use: delete a specified arc\n");
                printf("please enter the id of the graph:");
                scanf("%d", &graph_index);
                while (head != NULL) {
                        if (head->GraphID == graph_index)
                                break;
                        head = head->next;
                }
                if (head == NULL) {
                        printf("Error, the graph %d does not exist.\n", graph_index);
                        head = L;
                }
                else {
                        int v, w;
                        cout << "please insert the index1" << endl;
                        cin >> v;
                        cout << "please insert the index2" << endl;
                        cin >> w;
                        if (DeleteArc(*head, v, w) == OK) {
                                cout << "arc with nodes " << v << "and " << w << " has been successfully deleted." <<
endl;
                        }
                        else {
                                cout << "sorry, we encounter an ERROR." << endl;
                        }
                        head = L;
                }
                printf("\n");
                break;
        case 12:
                printf("* Function Name: DFSTraverse\n");
                printf("* Parameter: Graph &G\n");
                printf("* Return: Status(int)\n");
                printf("* Use: DFS Traverse\n");
                printf("please enter the id of the graph:");
                scanf("%d", &graph_index);
                while (head != NULL) {
```

```
                    if (head->GraphID == graph_index)
                        break;
                    head = head->next;
                }
                if (head == NULL) {
                    printf("Error, the graph %d does not exist.\n", graph_index);
                    head = L;
                }
                else {
                    cout << "DFS Traverse: ";
                    DFSTraverse(*head);
                    head = L;
                }
                printf("\n");
                break;
            case 13:
                printf("* Function Name: BFSTraverse\n");
                printf("* Parameter: Graph &G\n");
                printf("* Return: Status(int)\n");
                printf("* Use: BFS Traverse\n");
                printf("please enter the id of the graph:");
                scanf("%d", &graph_index);
                while (head != NULL) {
                    if (head->GraphID == graph_index)
                        break;
                    head = head->next;
                }
                if (head == NULL) {
                    printf("Error, the graph %d does not exist.\n", graph_index);
                    head = L;
                }
                else {
                    cout << "BFS Traverse: ";
                    DFSTraverse(*head);
                    head = L;
                }
                printf("\n");
                break;
            }
            SaveData(head);
        }
}
```