華中科技大学

课 程 实 验 报 告

课程名称： 数据结构实验

专业班级：计算机科学与技术 1703

学　　号：U201714607

姓　　名：钟子琛

指导教师：袁凌

报告日期：2019 年 1 月 2 日

计算机科学与技术学院

目 录

1 基于顺序存储结构的线性表实现

1.1 问题描述

通过实验达到

（1）加深对线性表的概念、基本运算的理解；

（2）熟练掌握线性表的逻辑结构与物理结构的关系；

（3）物理结构采用顺序表,熟练掌握线性表的基本运算的实现。

1.1.1 具体问题

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

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

⑵销毁表：函数名称是 DestroyList(L)；初始条件是线性表 L 已存在；操作结果是销毁线性表 L。

⑶清空表：函数名称是 ClearList(L)；初始条件是线性表 L 已存在；操作结果是将 L 重置为空表。

⑷判定空表：函数名称是 ListEmpty(L)；初始条件是线性表 L 已存在；操作结果是若 L 为空表则返回 TRUE,否则返回 FALSE。

⑸求表长：函数名称是 ListLength(L)；初始条件是线性表已存在；操作结果是返回 L 中数据元素的个数。

⑹获得元素：函数名称是 GetElem(L,i,e)；初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用 e 返回 L 中第 i 个数据元素的值。

⑺查找元素：函数名称是 LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回 L 中第 1 个与 e 满足关系 compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为 0。

⑻获得前驱：函数名称是 PriorElem(L,cur_e,pre_e)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是第一个，则用 pre_e 返回它

的前驱，否则操作失败，pre_e 无定义。

⑼获得后继：函数名称是 NextElem(L,cur_e,next_e)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是最后一个，则用 next_e 返回它的后继，否则操作失败，next_e 无定义。

⑽插入元素：函数名称是 ListInsert(L,i,e)；初始条件是线性表 L 已存在，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 ListTrabverse(L)，初始条件是线性表 L 已存在；操作结果是利用循环依次输出表中的每一个元素。

1.2 系统设计

1.2.1 系统总体设计

（1）通过 switch 语句选择需要使用的功能，然后调用对应的方法实现相关功能。

（2）再通过 while 循环语句实现反复使用菜单中的功能。

（3）设置一个选项让用户能够退出该程序。

1.2.2 算法设计

（1）InitaList(&L)

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

复杂度：时间复杂度 T(n) = O(1)，空间复杂度 S(n) = O(1)。

（2）DestroyList(*L)

设计：让 ele 指向 NULL，表长及存储容量置为 0，全局变量 isNull 设置为 TRUE。

复杂度：时间复杂度 T(n) = O(1)，空间复杂度 S(n) = O(1)。

（3）ClearList (&L)

设计：将表长设为 0。

复杂度：时间复杂度 T(n) = O(1)，空间复杂度 S(n) = O(1)。

（4）ListEmpty(L)

设计：读取表长 length 的值，为 0 则返回 TRUE，否则 FALSE。

复杂度：时间复杂度 T(n) = O(1)，空间复杂度 S(n) = O(1)。

（5）ListLength(L)

设计：返回 L.length 的值。

复杂度：时间复杂度 T(n) = O(1)，空间复杂度 S(n) = O(1)。

（6）GetElem(L, i, &e)

设计：将 L.elem[i - 1]的值赋给 e，并返回 OK

复杂度：时间复杂度 T(n) = O(1)，空间复杂度 S(n) = O(1)。

（7）LocateElem(L, e，compare())

设计：遍历顺序表，将与给定元素 e 满足关系 compare 的元素的位序返回。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（8）PriorElem（L，cur_e，*pre_e）

设计：从头遍历，若当前节点后继值为 cur_e，则将当前结点的元素赋给 pre_e，return OK。若未找到，则返回 FALSE。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（9）NextElem（L，cur_e，*next_e）

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

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（10）ListInsert(&L, i, e)

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

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（11）ListDelete(&L, i, &e)

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

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（12）ListTrabverse(L)

设计：遍历顺序表，用循环输出表中的每一个元素。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

1.3 系统实现

编程环境：

操作系统：WIN 10

IDE：Dev—C++，C 语言。

主模块：包含各变量申明、运行各个方法需要的输入与输出以及使用 switch 函数实现各函数的调用。程序运行主模块图如图 1.1 所示：

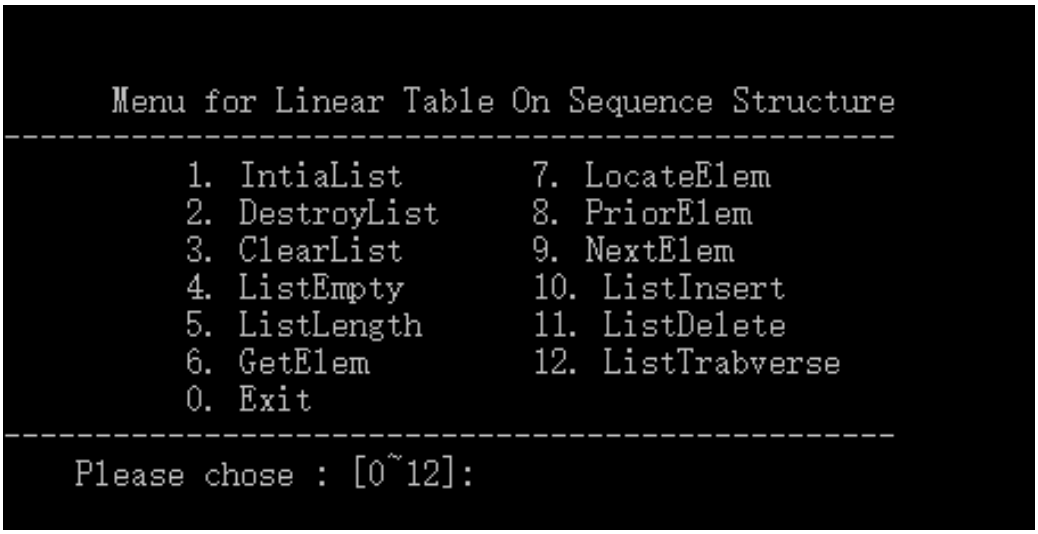

图 1.1 主模块图

功能模块：

1、初始化表：为表分配存储空间，若存储分配成功，头地址 L 为存储空间基址,表的长度为 0,表的存储容量为 10,返回 OK,输出 Success;否则返回 ERROR;如图 1.2 所示：

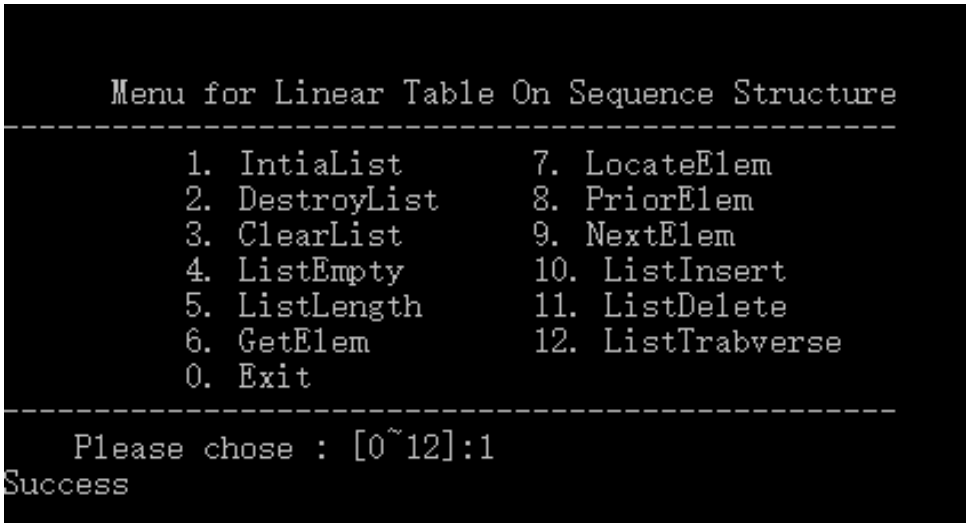

图 1.2 初始化表

2、销毁表：释放存储空间，使头地址为空，表长与存储容量置为 0，若成功，输出 success；如图 1.3 所示：

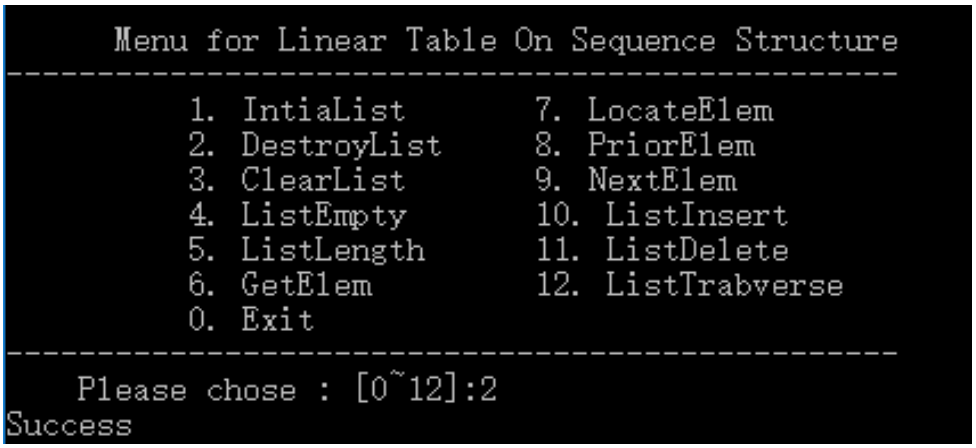

图 1.3 销毁表

当表并不存在时，销毁表会失败，如图 1.4 所示：

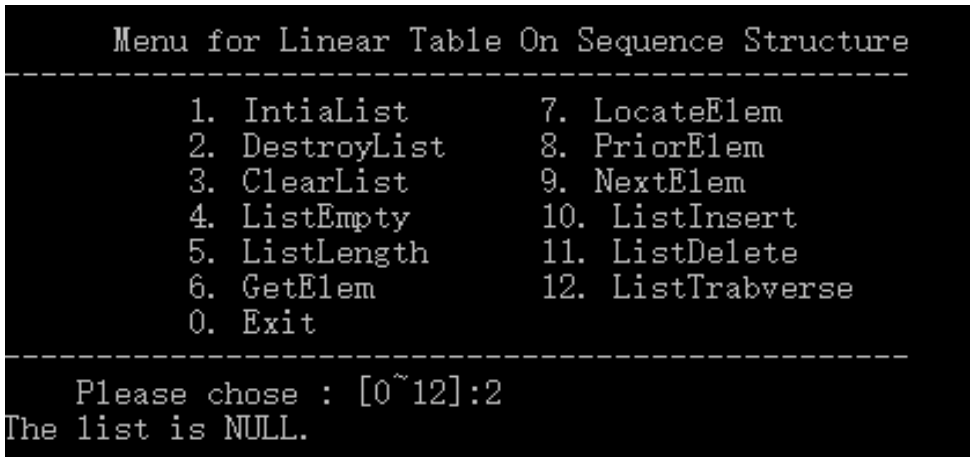

图 1.4 销毁表失败

3、清空表，将表长置为 0，即将表置为空表。调用插入元素方法、求表长和遍历表方法函数验证功能。如图 1.5，1.6，1.7，1.8 所示：

（1）新建一个表，插入元素，遍历；

（2）清空表；

（3）再次遍历表；

（4）求表长；

```
     Menu for Linear Table On Sequence Structure
------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
------------------------------------------------
    Please chose : [0~12]:12
23 49 15 51 34
```

图 1.5 新建表

```
     Menu for Linear Table On Sequence Structure
------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
------------------------------------------------
    Please chose : [0~12]:3
Success
```

图 1.6 清空表

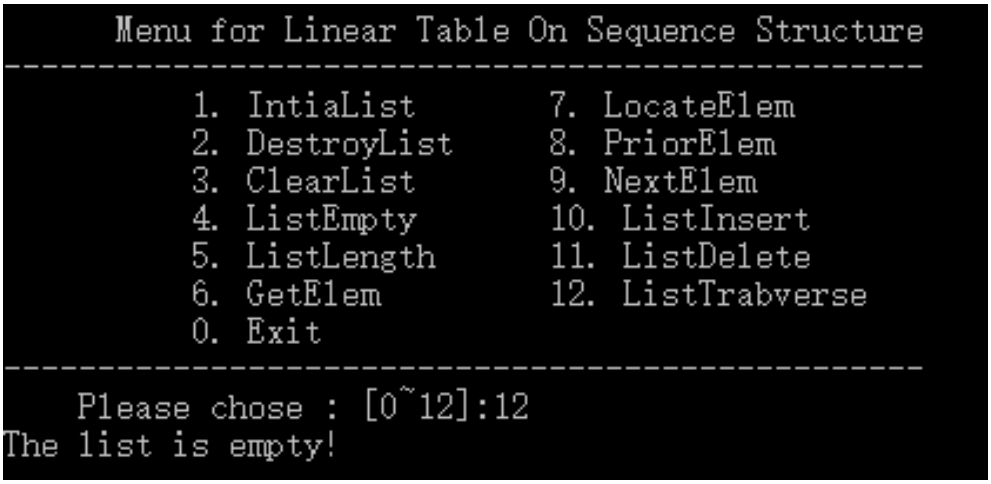

图 1.7 遍历表

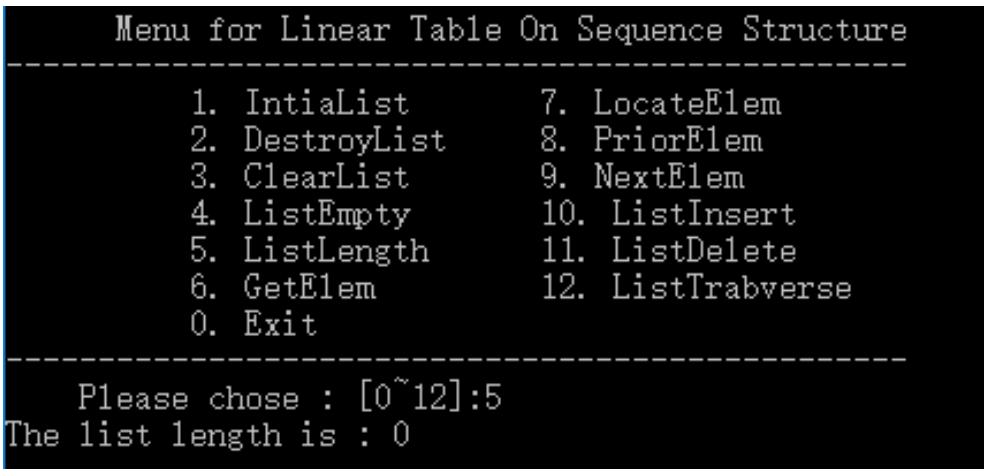

图 1.8 求表长

4、判断表空，通过判断表长是否为 0 来判断表是否为空。使用上面已经初始化的表 1，如图 1.9，1.10，1.11，1.12 所示：

（1）使用之前创建的表；

（2）判断是否为空；

（3）清空表，再次判断是否为空

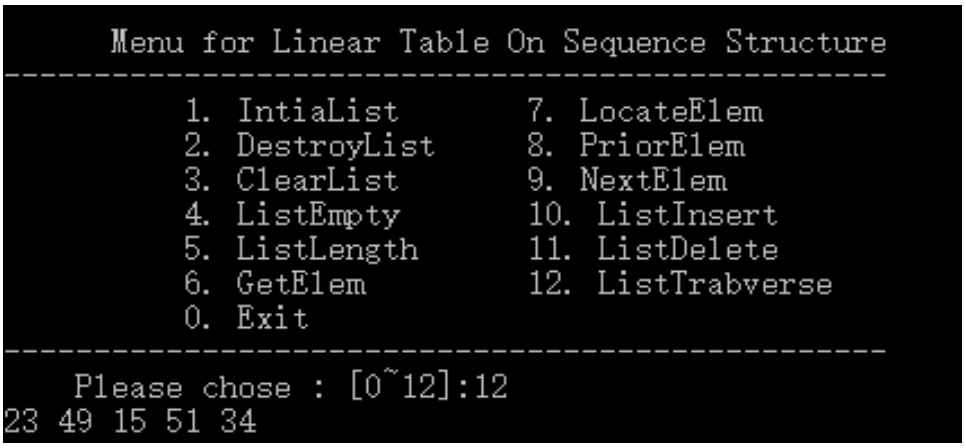

图 1.9 新建表

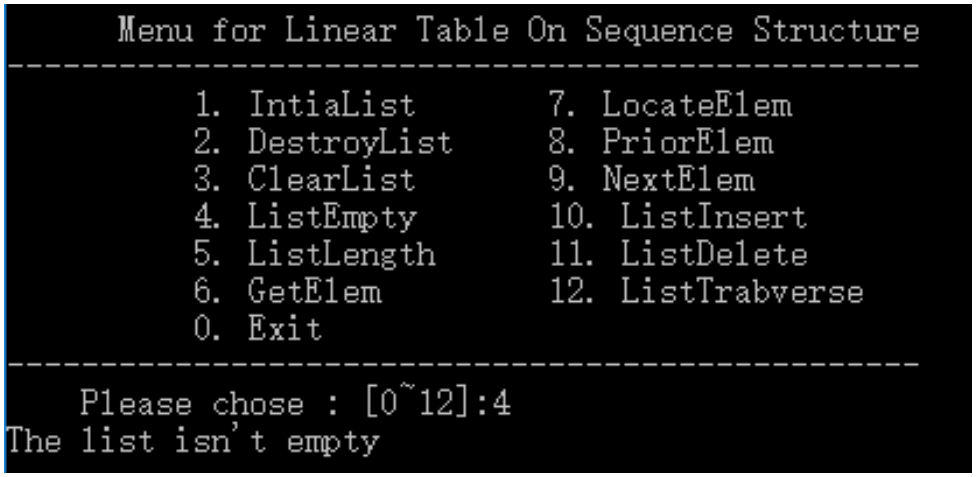

图 1.10 判断是否为空

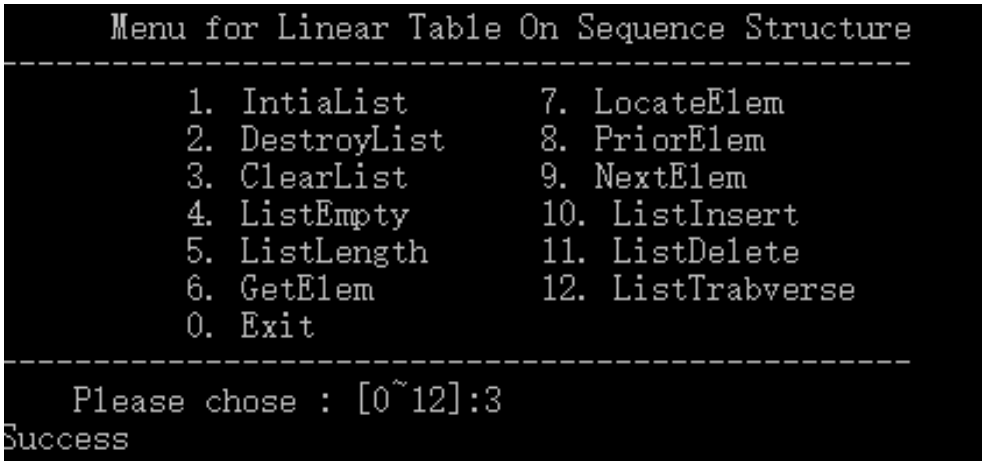

图 1.11 清空表

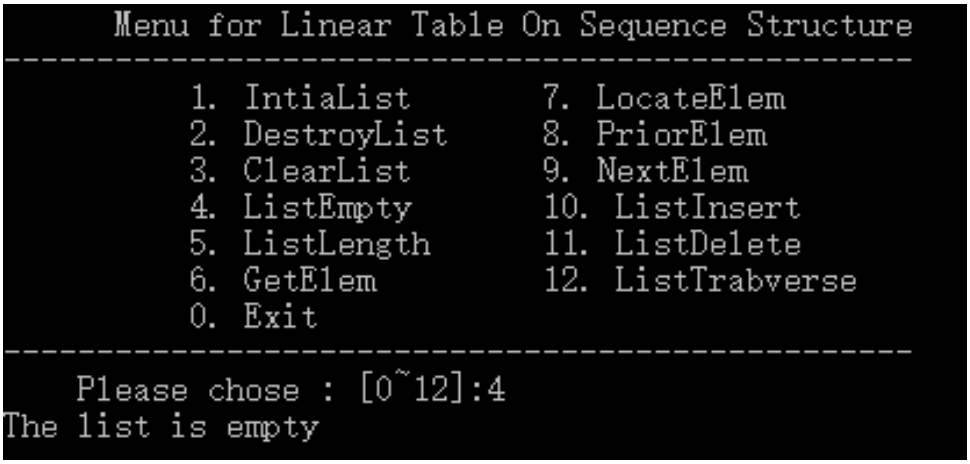

图 1.12 判断是否为空

5、求表长，通过循环求当前表中元素的个数，即求表长，如图 1.13，1.14 所示：

（1）使用之前初始化的表

（2）求表长应该为：5

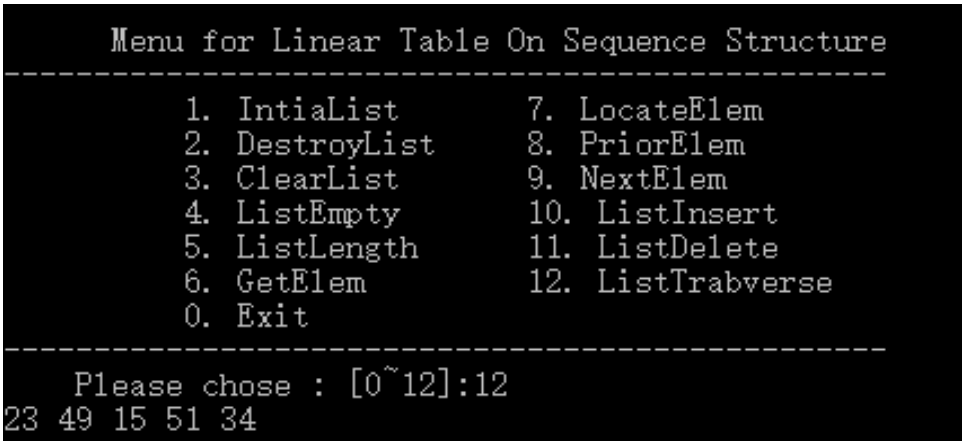

图 1.13 新建表

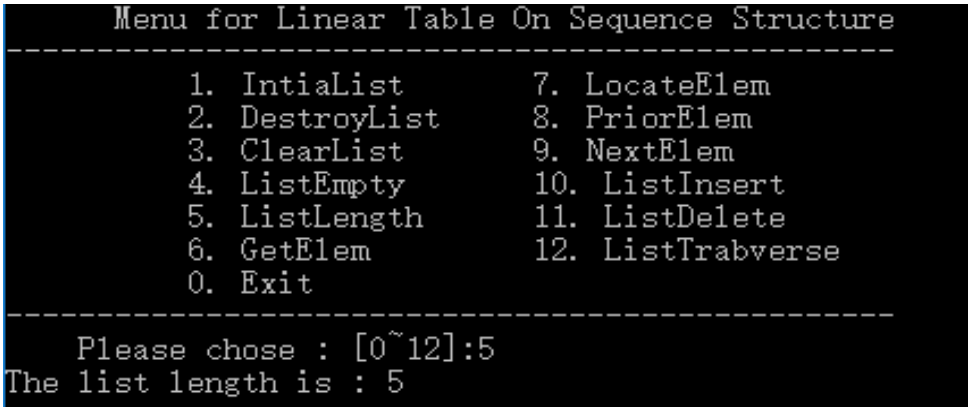

图 1.14 求表长

6、获取元素，使用自定义数据类型，将该数据类型变量 e 作为实参与头地址、所求元素位置一起传入函数，将表内对应位置元素赋值给 e，再返回 e 值，获得所求元素。如图 1.15，1.16 所示：

（1）使用之前初始化的表；

（2）使用该方法获取对应的元素；

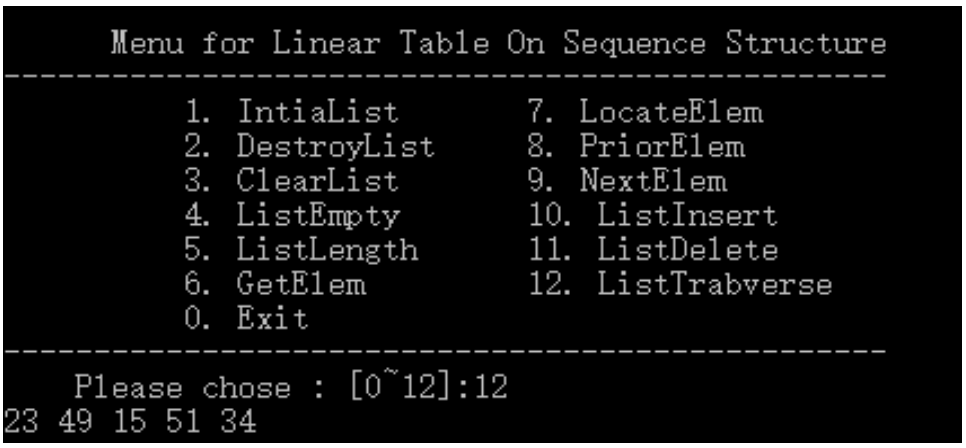

图 1.15 新建表

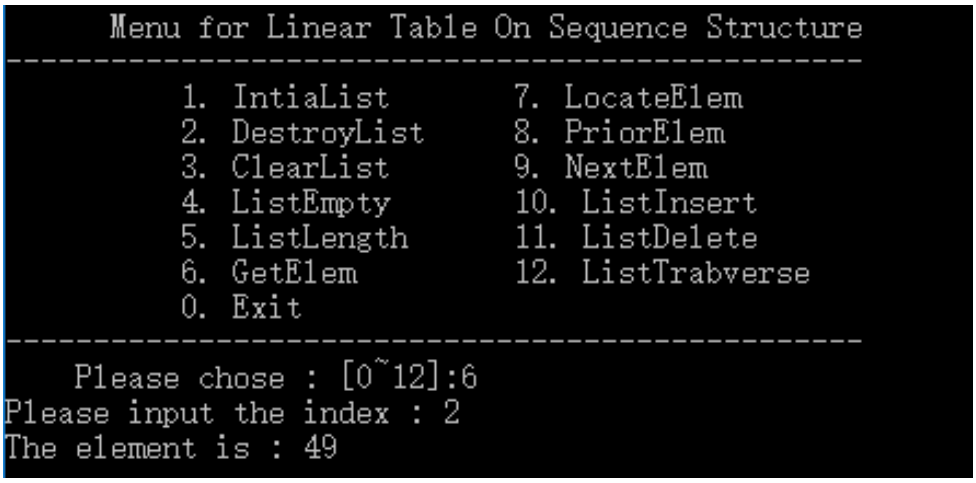

图 1.16 获取元素

7、查找元素，将用户需要查找的元素赋值给变量 e，传入函数，通过循环遍历查找表中与 e 相同的第一个元素，返回元素的位序，若找不到则输出”There is not the element.”，如图 1.17，1.18，1.19 所示：

（1）使用之前初始化的表，添加一个相同的元素

（2）使用该方法获取对应元素的位置；

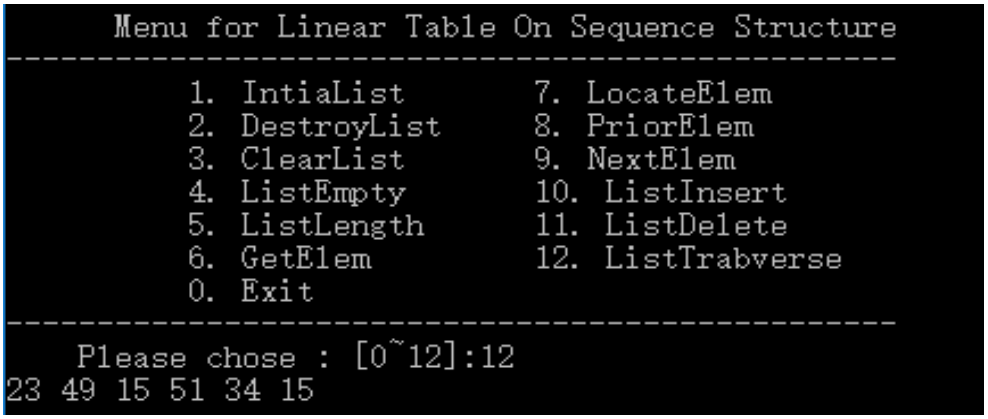

图 1.17 新建表

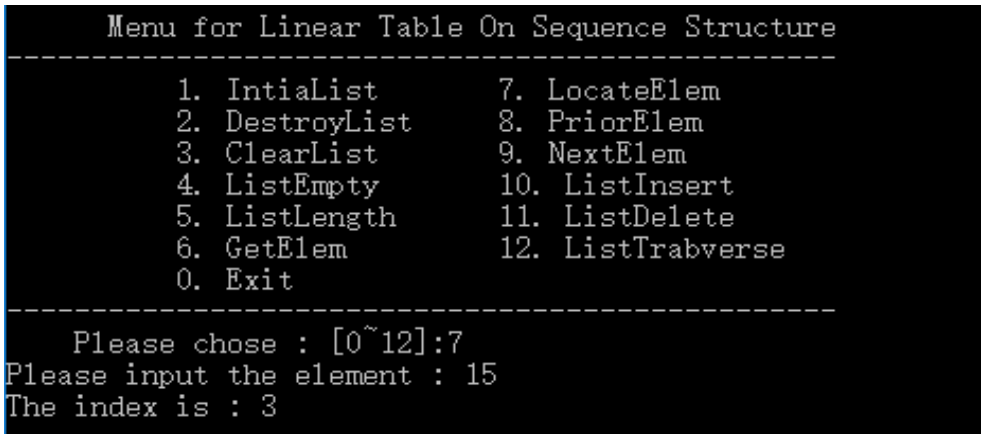

图 1.18 搜索元素

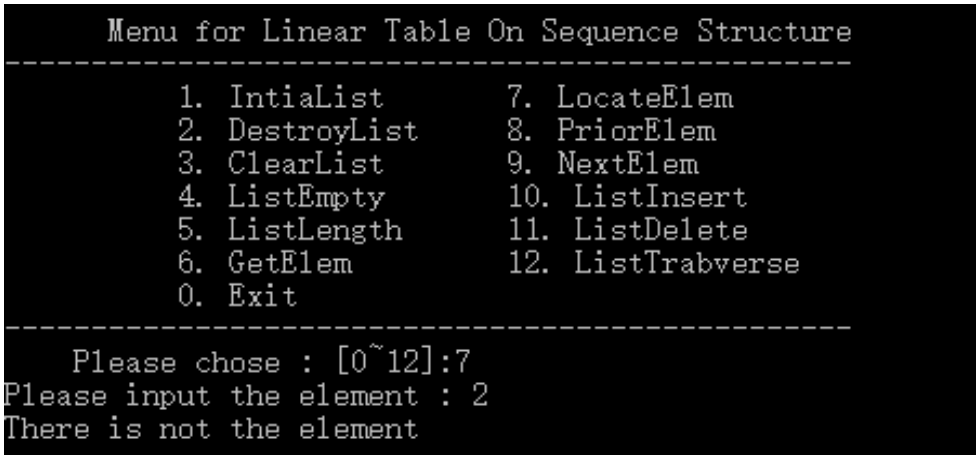

图 1.19 搜索不存在元素

8、获得前驱，输入元素，遍历表，若找到该元素且序位不为 1，返回该元素的前驱元素；若找不到或者元素位序为 1 则报错。如图 1.20，1.21，1.22 所示：

（1）使用之前初始化的表；

（2）寻找非第一个元素的前驱；

（3）寻找第一个元素的前驱；

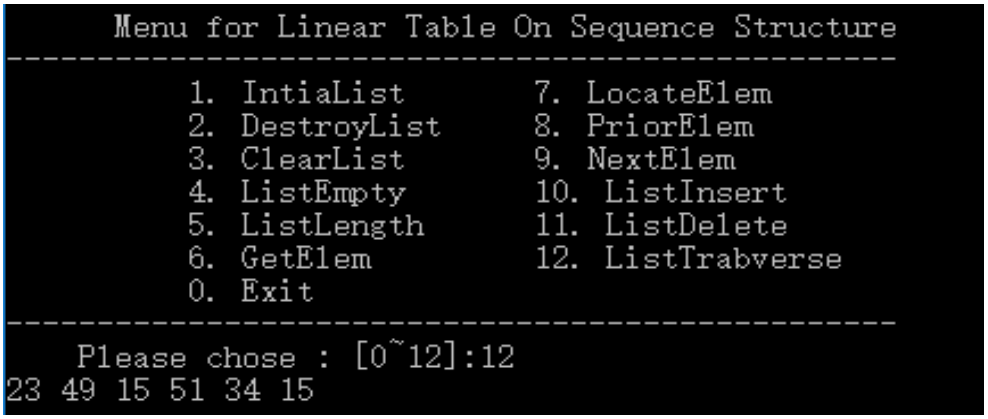

图 1.20 新建表

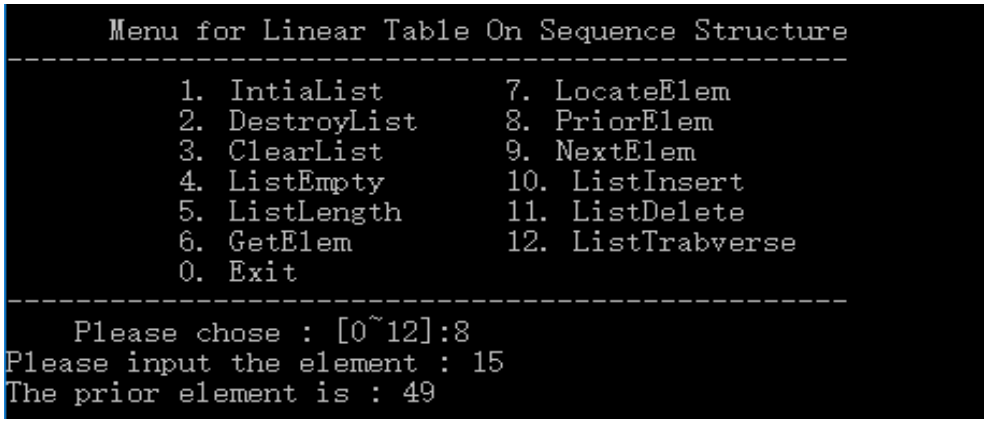

图 1.21 找前驱

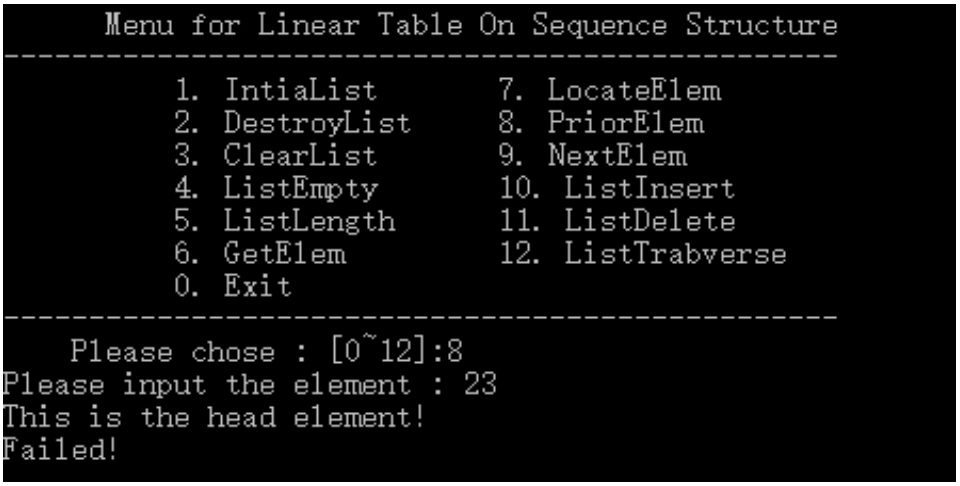

图 1.22 找首元素前驱

9、获得后继，输入元素，遍历表，若找到该元素且不为尾元素，返回该元素的后继元素；若找不到或者元素为尾元素则报错。如图 1.23，1.24，1.25 所示：

（1）使用之前初始化的表；

（2）寻找非第一个元素的后继；

（3）寻找第一个元素的后继；

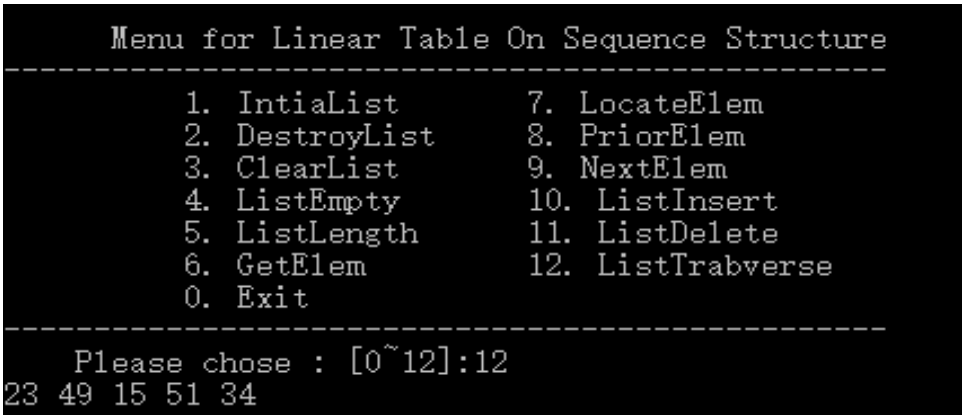

图 1.23 新建表

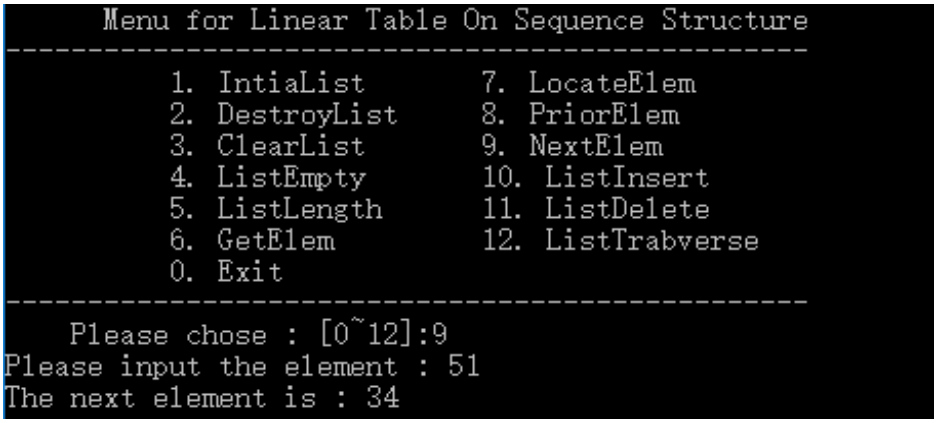

图 1.24 获取后继

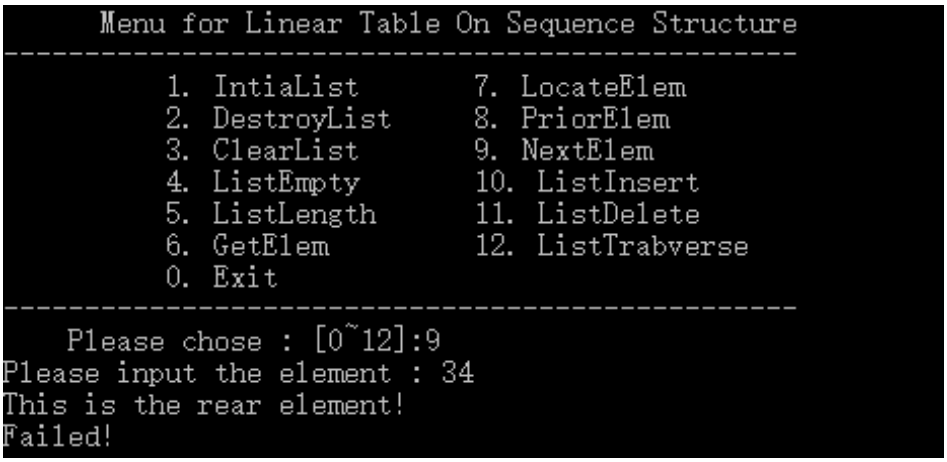

图 1.25 获取尾元素后继

10、插入元素，输入插入位置、插入元素与表地址一起传递给函数，若插入位置超出范围则报错，通过循环，将插入点之后的元素全部后移一位，插入元素后，将表长增加 1。如图 1.26，1.27，1.28，1.29 所示：

（1）使用之前初始化的表；

（2）插入一个元素；

（3）遍历表，看插入是否成功；

（4）在超出范围处插入元素；

```
     Menu for Linear Table On Sequence Structure
-------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
-------------------------------------------------
    Please chose : [0~12]:12
23 49 15 51 34
```

图 1.26 新建表

```
     Menu for Linear Table On Sequence Structure
-------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
-------------------------------------------------
    Please chose : [0~12]:10
Please input the index : 3
Please input the element : 123
Success
```

图 1.27 插入元素

```
     Menu for Linear Table On Sequence Structure
-------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
-------------------------------------------------
    Please chose : [0~12]:12
23 49 123 15 51 34
```

图 1.28 遍历表

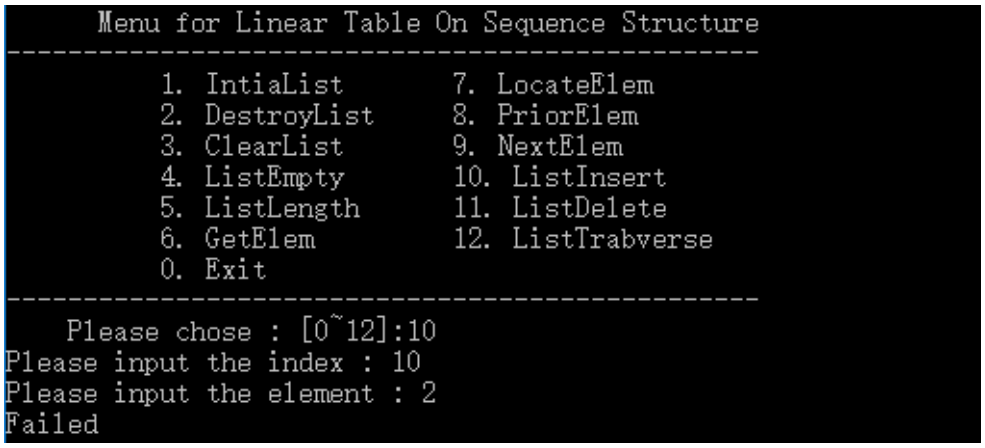

图 1.29 插入失败

10、删除元素，输入删除元素的位序与表头地址一起传入函数，若删除位置超出范围则会报错，删除了该位序元素后，将之后的元素全向前移动一位，表长减 1，返回被删除的元素。如图 1.30，1.31，1.32，1.33 所示：

（1）使用之前初始化的表；

（2）删除一个元素；

（3）遍历表，看删除是否成功；

（4）删除超出范围的元素；

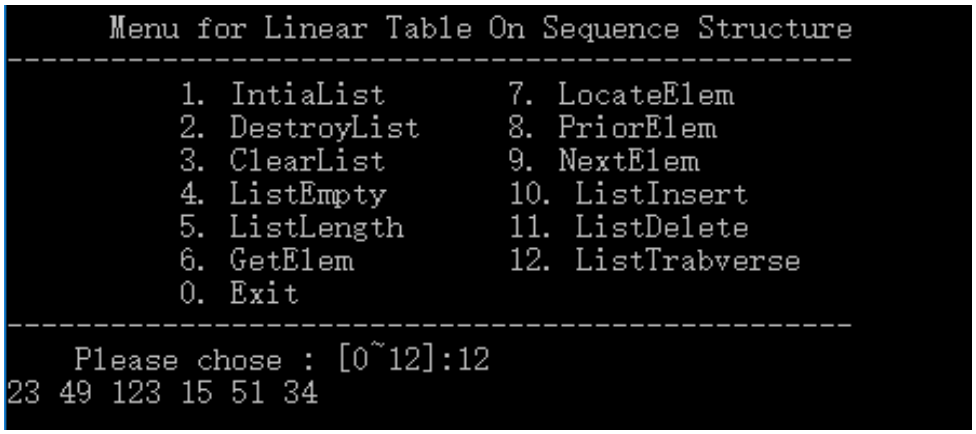

图 1.30 初始表

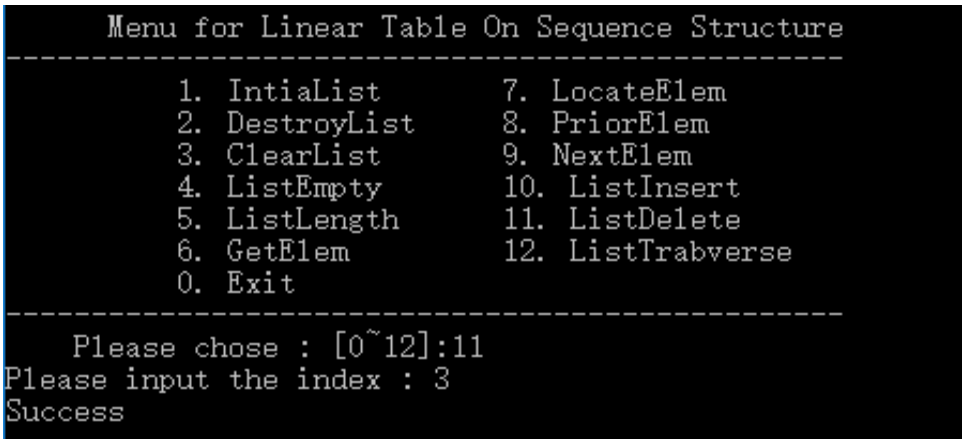

图 1.31 删除元素

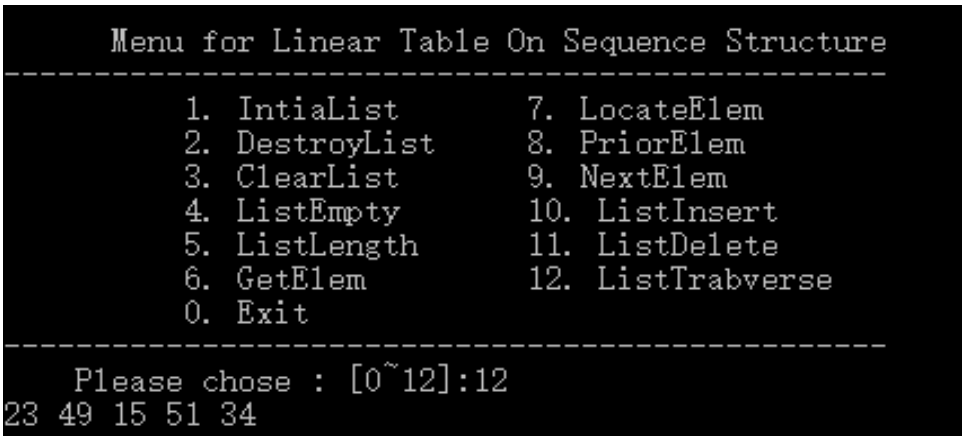

图 1.32 遍历表

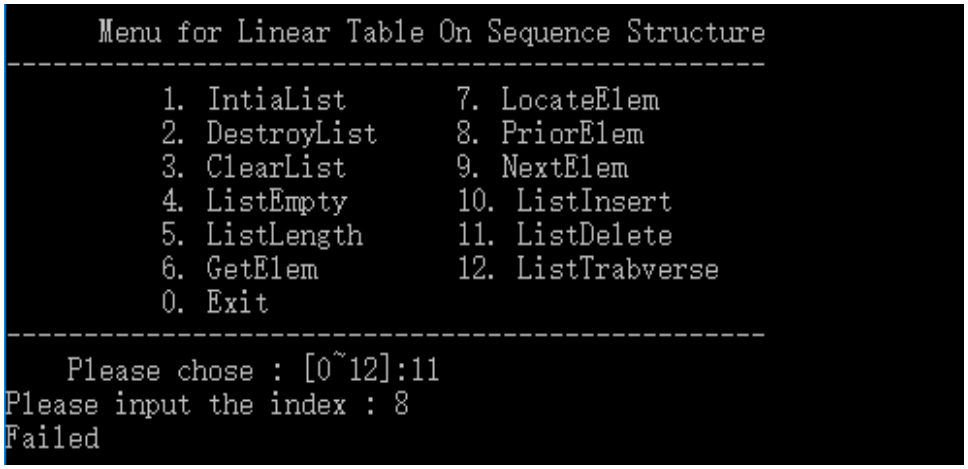

图 1.33 删除失败

12、遍历表，通过循环输出表中的所有元素。如图 1.34 所示：

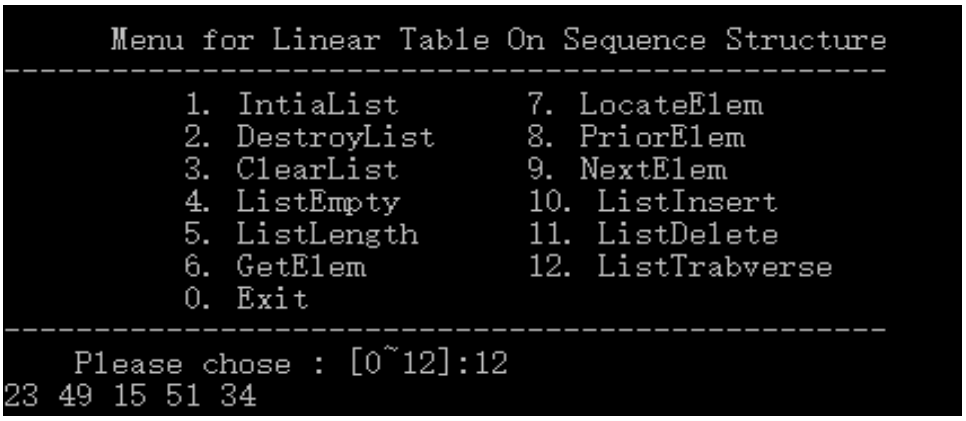

图 1.34 遍历表

1.4 实验小结

因为有较好的 C 语言基础，本次实验总体难度并不是很大，只不过有许多细节需要处理得更好，下面总结一下遇到的问题：

（1） 进行异常处理的时候，刚开始并没有将错误反映给用户，比如说表空、删除超出范围、首元素得前驱等等，后面进行了修改，使得用户明白了自己得错误。

（2） 在最初进行销毁表的操作时，并没有真正做到销毁表，对 free 函数不太理解，后面通过查询资料，理解了 free 函数的用法，解决了销毁表遇到的问题。

这次实验相当于自己做了一个小的项目，对自己的将来有了更加清楚的认识；回忆了上学期学习的 C 语言基础知识，巩固了这学期学习的数据结构知识，总的来说，有很大收获。

2 基于链式存储结构的线性表实现

2.1 问题描述

通过实验达到：

（1）加深对线性表的概念、基本运算的理解；

（2）熟练掌握线性表的逻辑结构与物理结构的关系；

（3）物理结构采用单链表,熟练掌握线性表的基本运算的实现。

2.1.1 具体问题

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

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

⑵销毁表：函数名称是 DestroyList(L)；初始条件是线性表 L 已存在；操作结果是销毁线性表 L。

⑶清空表：函数名称是 ClearList(L)；初始条件是线性表 L 已存在；操作结果是将 L 重置为空表。

⑷判定空表：函数名称是 ListEmpty(L)；初始条件是线性表 L 已存在；操作结果是若 L 为空表则返回 TRUE,否则返回 FALSE。

⑸求表长：函数名称是 ListLength(L)；初始条件是线性表已存在；操作结果是返回 L 中数据元素的个数。

⑹获得元素：函数名称是 GetElem(L,i,e)；初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用 e 返回 L 中第 i 个数据元素的值。

⑺查找元素：函数名称是 LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回 L 中第 1 个与 e 满足关系 compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为 0。

⑻获得前驱：函数名称是 PriorElem(L,cur_e,pre_e)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是第一个，则用 pre_e 返回它的前驱，否则操作失败，pre_e 无定义。

⑼获得后继：函数名称是 NextElem(L,cur_e,next_e)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是最后一个，则用 next_e 返回它的后继，否则操作失败，next_e 无定义。

⑽插入元素：函数名称是 ListInsert(L,i,e)；初始条件是线性表 L 已存在，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 ListTrabverse(L,visit())，初始条件是线性表 L 已存在；操作结果是依次对 L 的每个数据元素调用函数 visit()。

2.2 系统设计

2.2.1 系统总体设计

（1）通过 switch 语句选择需要使用的功能，然后调用对应的方法实现相关功能。

（2）再通过 while 循环语句实现反复使用菜单中的功能。

（3）设置一个选项让用户能够退出该程序。

2.2.2 算法设计

（1）InitaList(&L)

设计：分配存储空间，设置链表头节点的数据域和指针域皆为空。

复杂度：时间复杂度 T(n) = O(1)，空间复杂度 S(n) = O(1)。

（2）DestroyList(*L)

设计：将链表的所有节点依次 free()，让 L 指向 NULL，全局变量 isNull 设置为 TRUE。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（3）ClearList (&L)

设计：将除了头节点以外的节点空间全部释放，头节点指针域指向 NULL。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（4）ListEmpty(L)

设计：利用循环计算链表中的元素个数，为 0 则返回 TRUE，否则 FALSE。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（5）ListLength(L)

设计：利用循环计算链表中的元素个数，返回元素个数。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（6）GetElem(L, i, &e)

设计：检索链表，循环 i - 1 次，将对应的节点的数据域的值赋给 e，并返回 OK。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（7）LocateElem(L, e，compare())

设计：遍历链表，将与给定元素 e 满足关系 compare 的元素的位序返回。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（8）PriorElem（L，cur_e，*pre_e）

设计：从头遍历，若当前节点后继值为 cur_e，则将当前结点的元素赋给 pre_e，return OK。若未找到，则返回 FALSE。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（9）NextElem（L，cur_e，*next_e）

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

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（10）ListInsert(&L, i, e)

设计：先判断 i 值是否符合要求，不符返回 ERROR。i 值合法，则检查空间大小，若空间不够，增加存储容量。然后将插入位置之前的节点的指针域指向新节点，新节点的指针域指向原位置的节点。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（11）ListDelete(&L, i, &e)

设计：先判断 i 值是否符合要求，不符返回 ERROR。i 值合法，则将被删除元素的值赋给 e，将该位置的前驱节点的指针域指向该节点的后继节点，释放该节点的空间。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

（12）ListTrabverse(L)

设计：遍历顺序表，用循环输出链表中的每一个元素。

复杂度：时间复杂度 T(n) = O(n)，空间复杂度 S(n) = O(1)。

2.3 系统实现

编程环境：

操作系统：WIN 10

IDE：Dev—C++，C 语言。

主模块：包含各变量申明、运行各个方法需要的输入与输出以及使用 switch 函数实现各函数的调用。程序运行主模块图如图 2.1 所示：

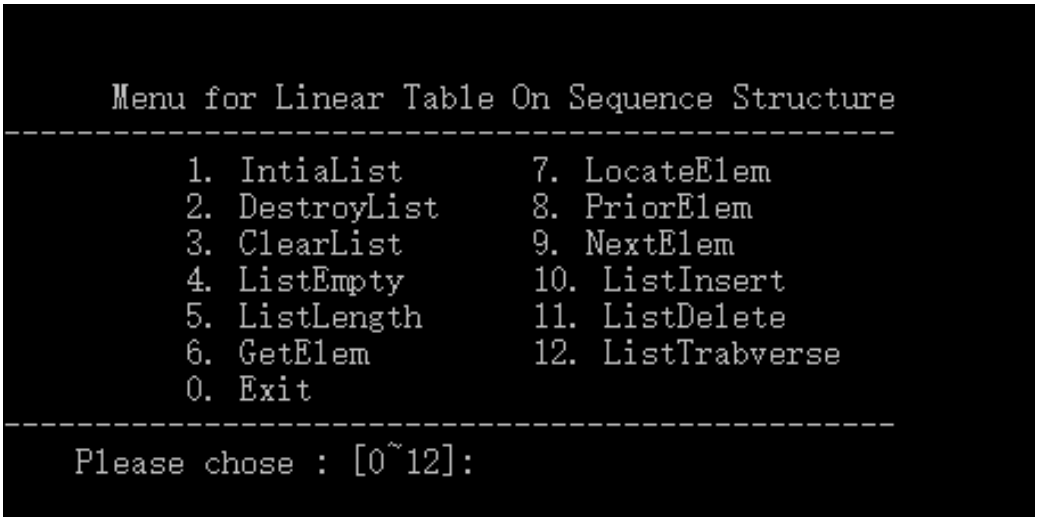

图 2.1 主模块图

功能模块：

1、初始化表：为表分配存储空间，若存储分配成功，头地址 L 为存储空间基址，表的存储容量为 10，返回 OK，输出 Success；否则返回 ERROR；如图 2.2 所示：

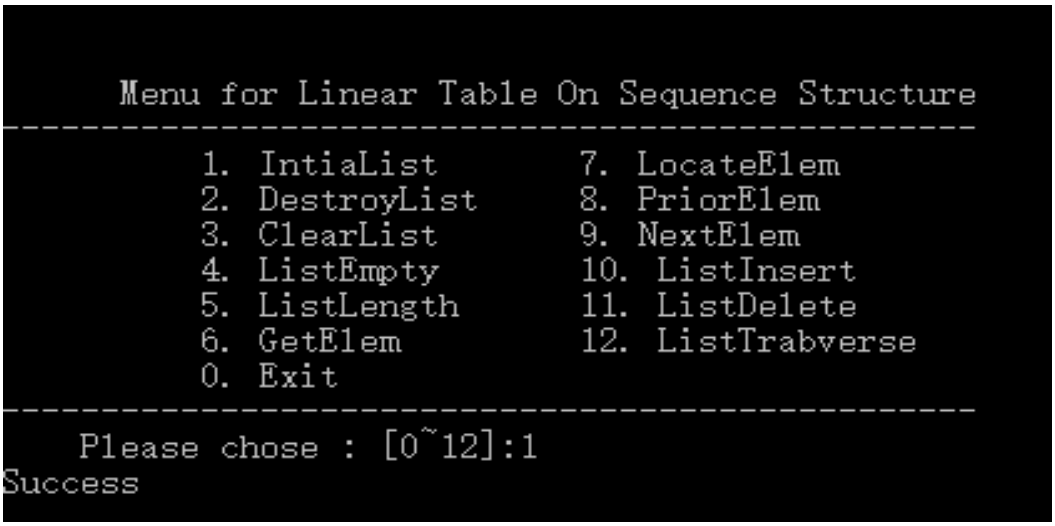

图 2.2 初始化表

2、销毁表：释放存储空间，使头地址为空，若成功，输出 Success；如图 2.3 所示：

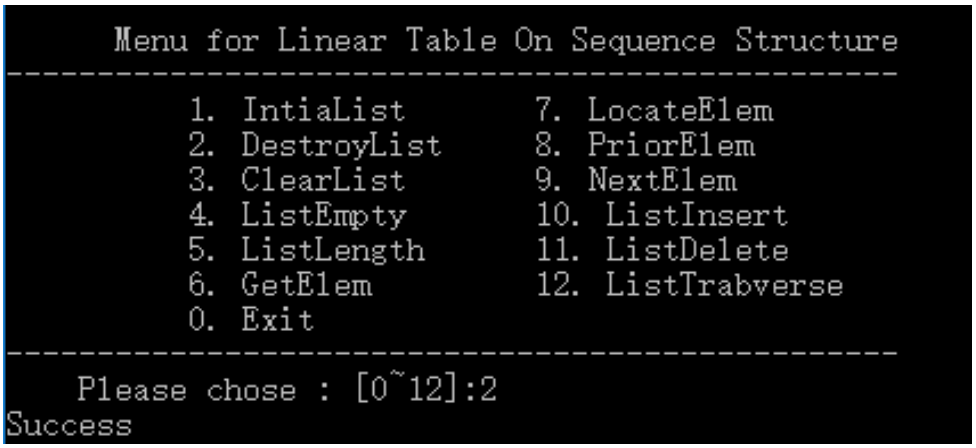

图 2.3 销毁表

当表并不存在时，销毁表会失败，如图 2.4 所示：

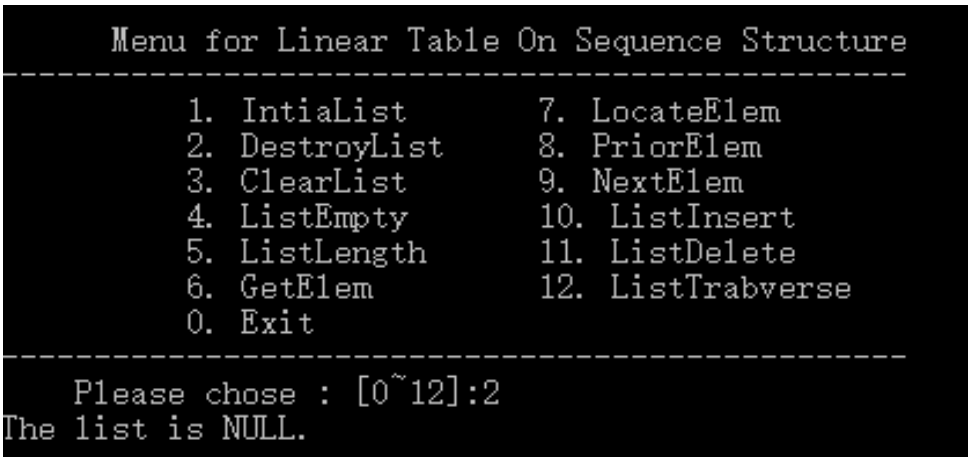

图 2.4 销毁表失败

3、清空表，一次释放节点空间。调用插入元素方法、求表长和遍历表方法函数验证功能。如图 2.5，2.6，2.7，2.8 所示：

（1）新建一个表，插入元素，遍历；

（2）清空表；

（3）再次遍历表；

（4）求表长；

```
     Menu for Linear Table On Sequence Structure
-------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
-------------------------------------------------
    Please chose : [0~12]:12
34 92 57 43 927
```

图 2.5 新建表

```
     Menu for Linear Table On Sequence Structure
-------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
-------------------------------------------------
    Please chose : [0~12]:3
Success
```

图 2.6 清空表

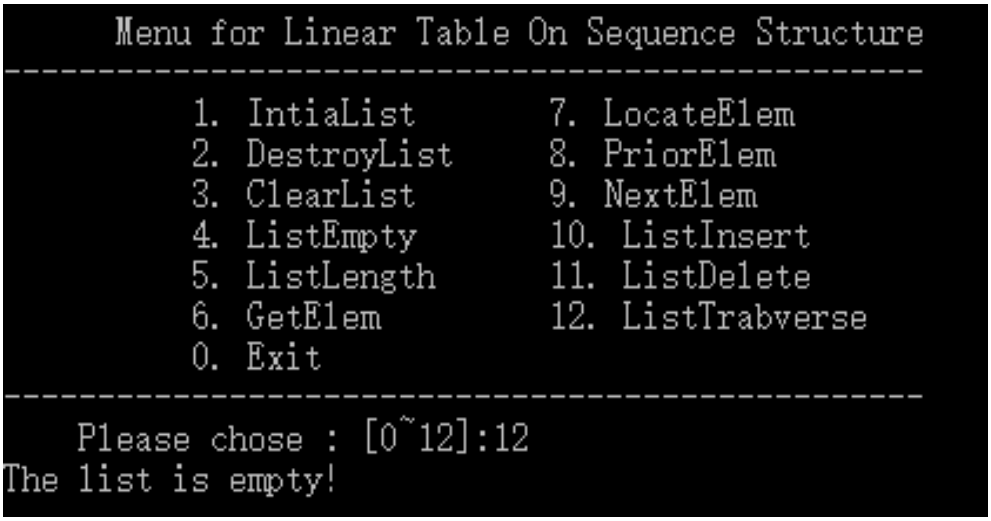

图 2.7 遍历表

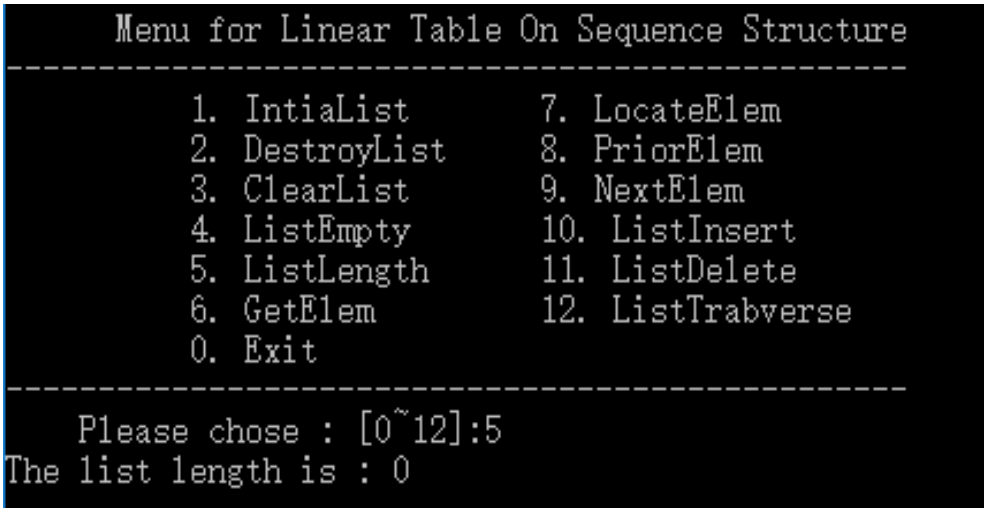

图 2.8 求表长

4、判断表空，通过循环获取链表的长度来判断是否为空。使用上面已经初始化的链表，如图 2.9，2.10，2.11，2.12 所示：

（1）使用之前创建的表；

（2）判断是否为空；

（3）清空表，再次判断是否为空

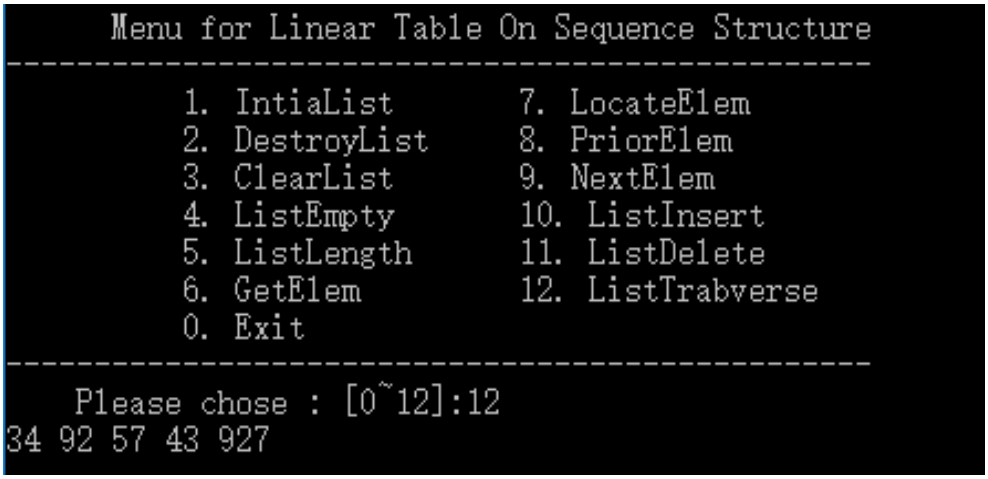

图 2.9 新建表

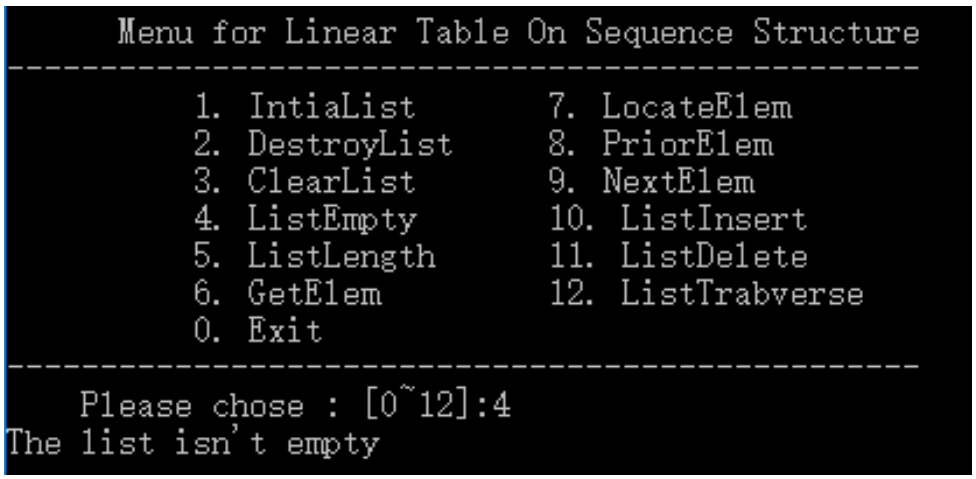

图 2.10 判断是否为空

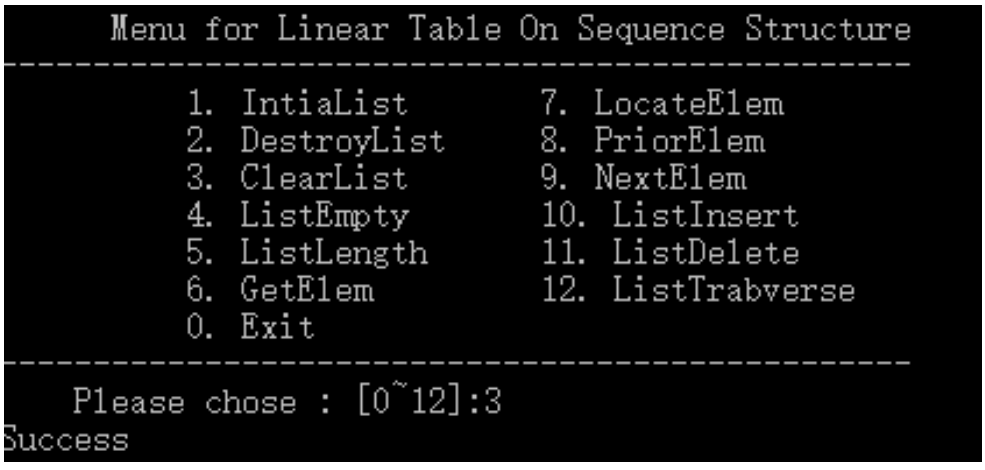

图 2.11 清空表

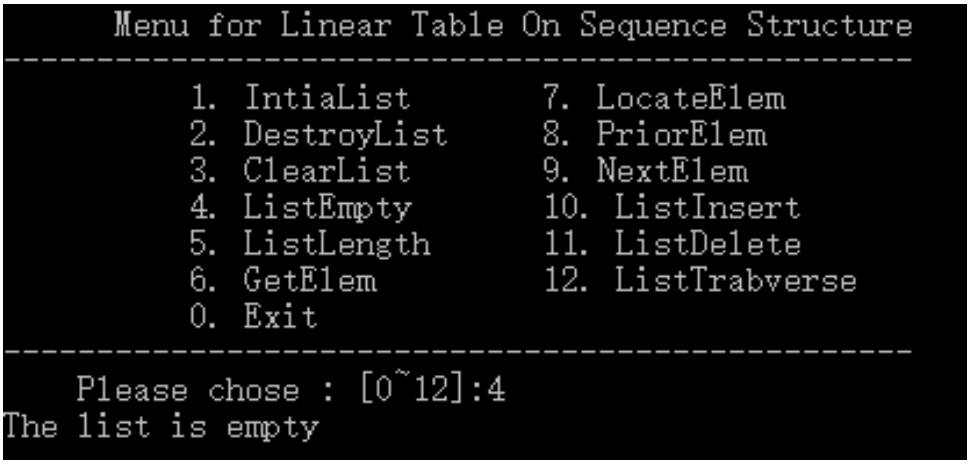

图 2.12 判断是否为空

5、求表长，通过循环求当前表中元素的个数，即求表长，如图 2.13，2.14 所示：

（1）使用之前初始化的表

（2）求表长应该为：5

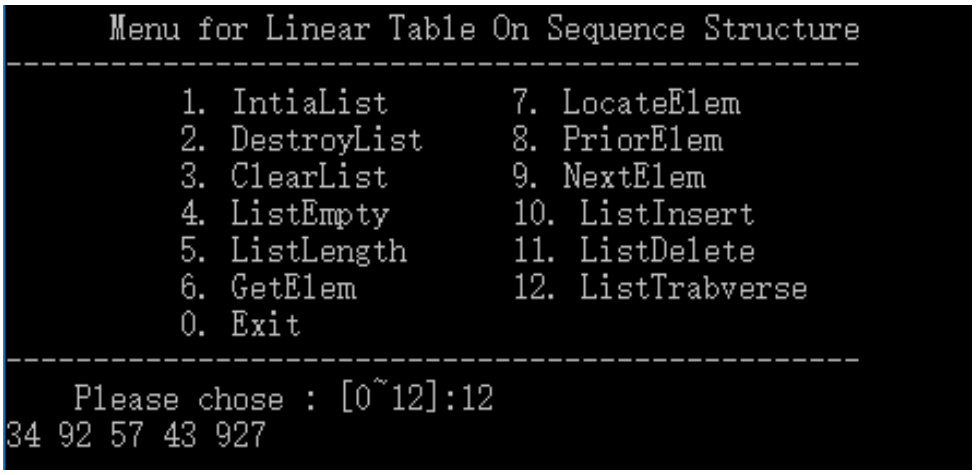

图 2.13 新建表

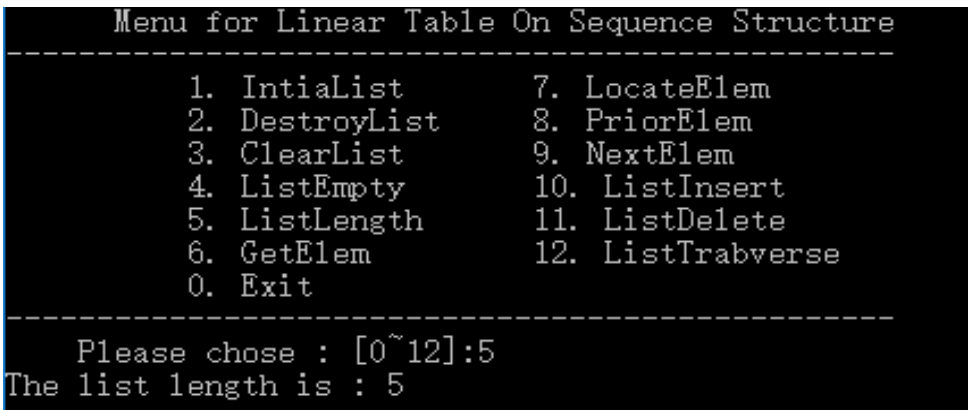

图 2.14 求表长

6、获取元素，使用自定义数据类型，将该数据类型变量 e 作为实参与头地址、所求元素位置一起传入函数，将表内对应位置元素赋值给 e，再返回 e 值，获得所求元素。如图 2.15，2.16 所示：

（1）使用之前初始化的表；

（2）使用该方法获取对应的元素；

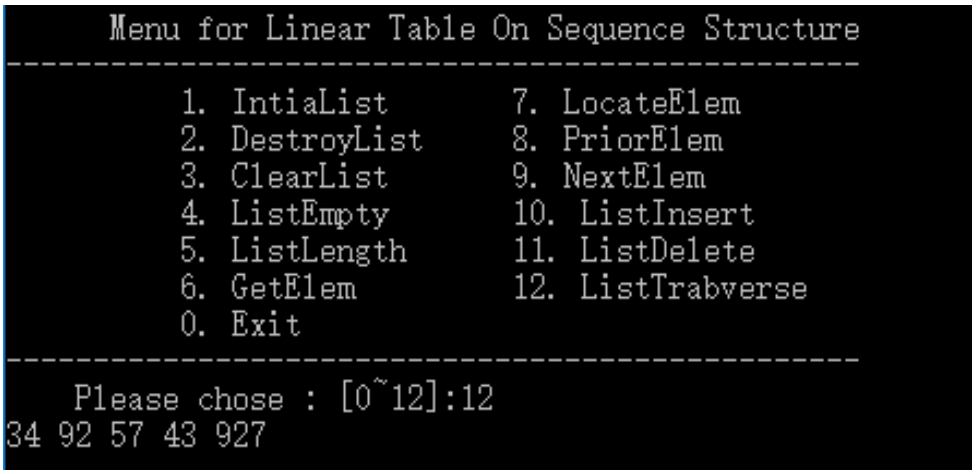

图 2.15 新建表

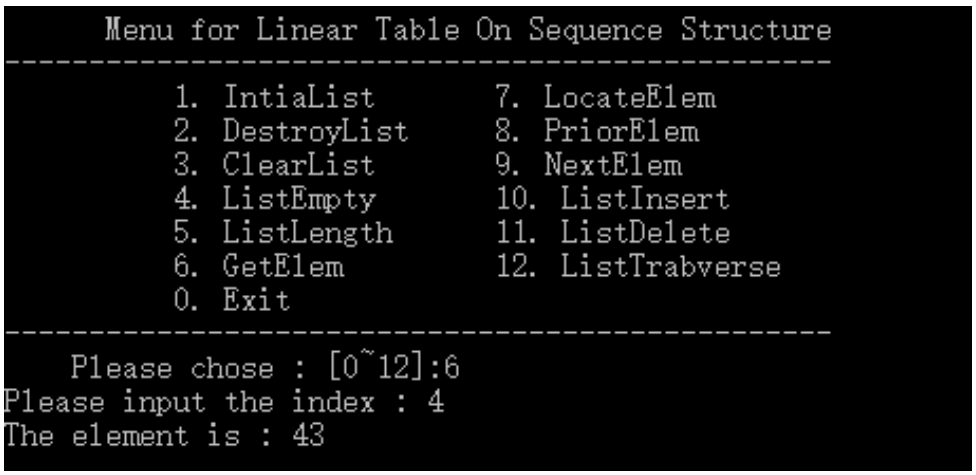

图 2.16 获取元素

7、查找元素，将用户需要查找的元素赋值给变量 e，传入函数，通过循环遍历查找表中与 e 相同的第一个元素，返回元素的位序，若找不到则输出”There is not the element.”，如图 2.17，2.18，2.19 所示：

（1）使用之前初始化的表，添加一个相同的元素

（2）使用该方法获取对应元素的位置；

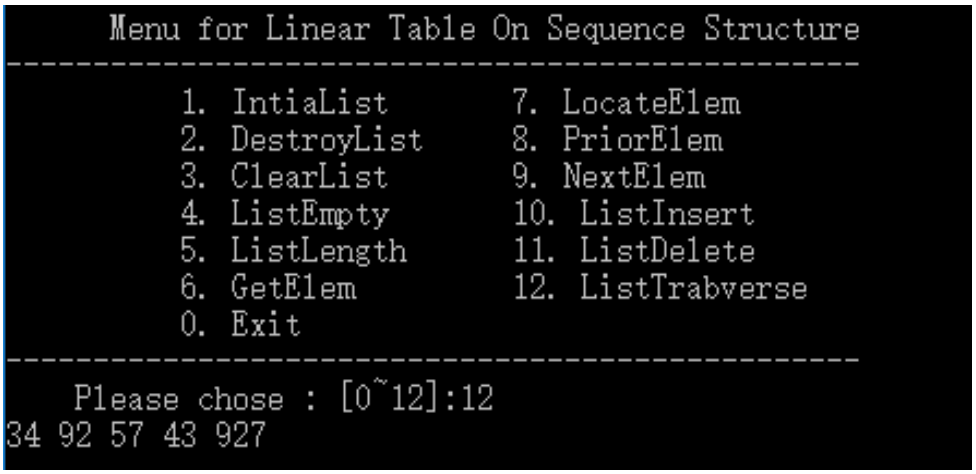

图 2.17 新建表

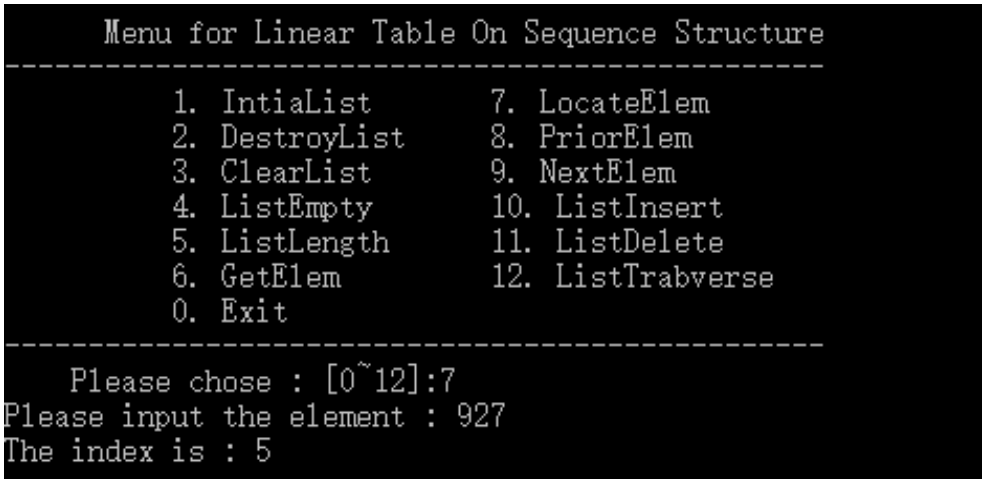

图 2.18 搜索元素

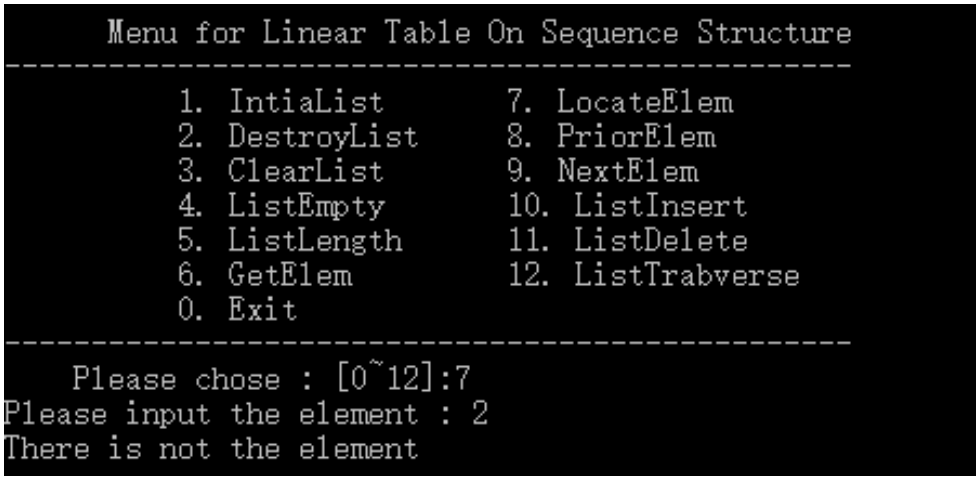

图 2.19 搜索不存在元素

8、获得前驱，输入元素，遍历表，若找到该元素且序位不为 1，返回该元素的前驱元素；若找不到或者元素位序为 1 则报错。如图 2.20，2.21，2.22 所示：

（1）使用之前初始化的表；

（2）寻找非第一个元素的前驱；

（3）寻找第一个元素的前驱；

```
     Menu for Linear Table On Sequence Structure
------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
------------------------------------------------
    Please chose : [0~12]:12
34 92 57 43 927
```

图 2.20 新建表

```
     Menu for Linear Table On Sequence Structure
------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
------------------------------------------------
    Please chose : [0~12]:8
Please input the element : 92
The prior element is : 34
```

图 2.21 找前驱

```
     Menu for Linear Table On Sequence Structure
------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
------------------------------------------------
    Please chose : [0~12]:8
Please input the element : 34
The pre element doesn't exist.
```

图 2.22 找首元素前驱

9、获得后继，输入元素，遍历表，若找到该元素且不为尾元素，返回该元素的后继元素；若找不到或者元素为尾元素则报错。如图 2.23，2.24，2.25 所示：

（1）使用之前初始化的表；

（2）寻找非第一个元素的后继；

（3）寻找第一个元素的后继；

```
     Menu for Linear Table On Sequence Structure
------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
------------------------------------------------
    Please chose : [0~12]:12
34 92 57 43 927
```

图 2.23 新建表

```
     Menu for Linear Table On Sequence Structure
------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
------------------------------------------------
    Please chose : [0~12]:9
Please input the element : 57
The next element is : 43
```

图 2.24 获取后继

```
     Menu for Linear Table On Sequence Structure
------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
------------------------------------------------
    Please chose : [0~12]:9
Please input the element : 927
The next element doesn't exist.
```

图 2.25 获取尾元素后继

10、插入元素，输入插入位置、插入元素与表地址一起传递给函数，若插入位置超出范围则报错。如图 2.26，2.27，2.28，2.29 所示：

（1）使用之前初始化的表；

（2）插入一个元素；

（3）遍历表，看插入是否成功；

（4）在超出范围处插入元素；

```
     Menu for Linear Table On Sequence Structure
------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
------------------------------------------------
    Please chose : [0~12]:12
34 92 57 43 927
```

图 2.26 新建表

```
     Menu for Linear Table On Sequence Structure
------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
------------------------------------------------
    Please chose : [0~12]:10
Please input the index : 4
Please input the element : 12321
Success
```

图 2.27 插入元素

```
     Menu for Linear Table On Sequence Structure
------------------------------------------------
         1. IntiaList       7. LocateElem
         2. DestroyList     8. PriorElem
         3. ClearList       9. NextElem
         4. ListEmpty       10. ListInsert
         5. ListLength      11. ListDelete
         6. GetElem         12. ListTrabverse
         0. Exit
------------------------------------------------
    Please chose : [0~12]:12
34 92 57 12321 43 927
```

图 2.28 遍历表

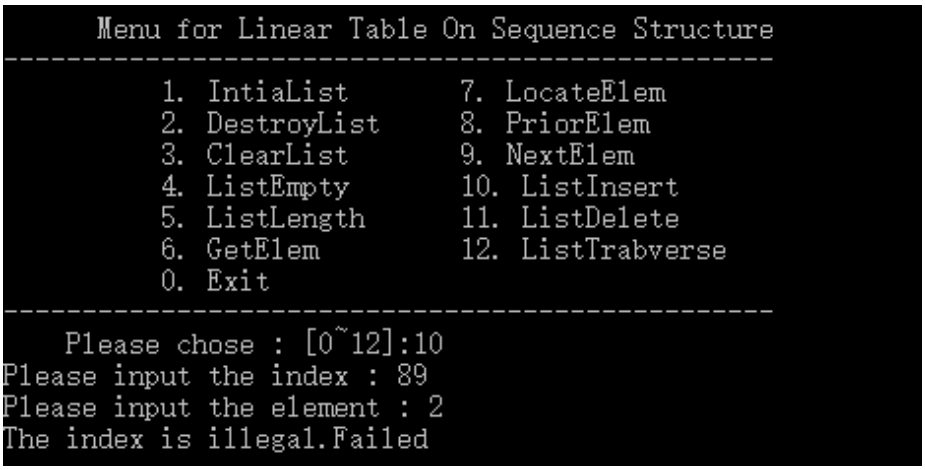

图 2.29 插入失败

11、删除元素，输入删除元素的位序与表头地址一起传入函数，若删除位置超出范围则会报错。如图 2.30，2.31，2.32，2.33 所示：

（1）使用之前的表；

（2）删除一个元素；

（3）遍历表，看删除是否成功；

（4）删除超出范围的元素；

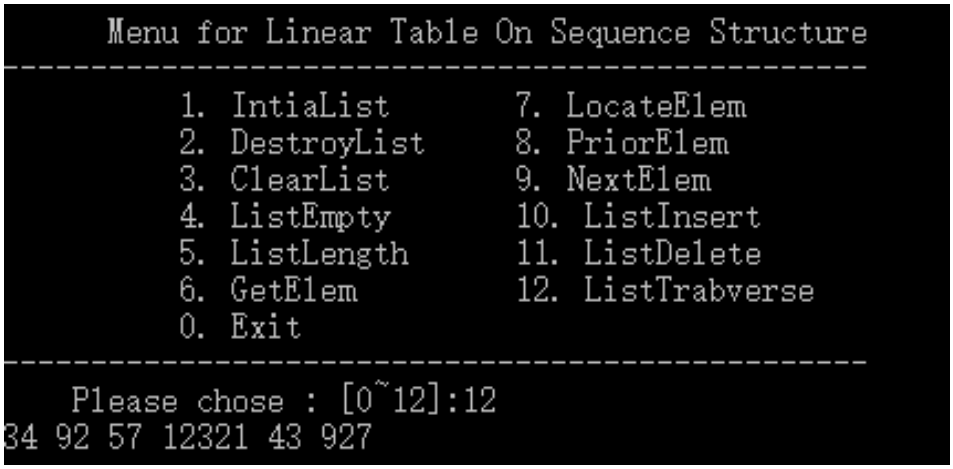

图 2.30 初始表

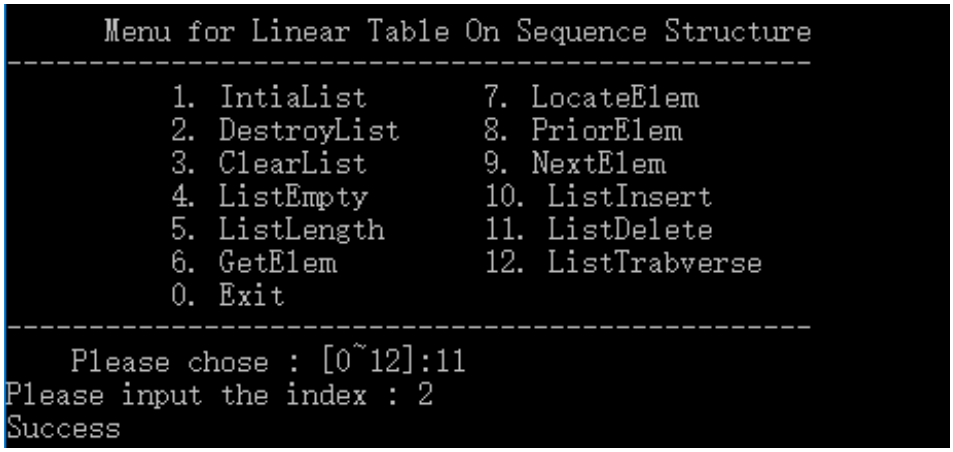

图 2.31 删除元素

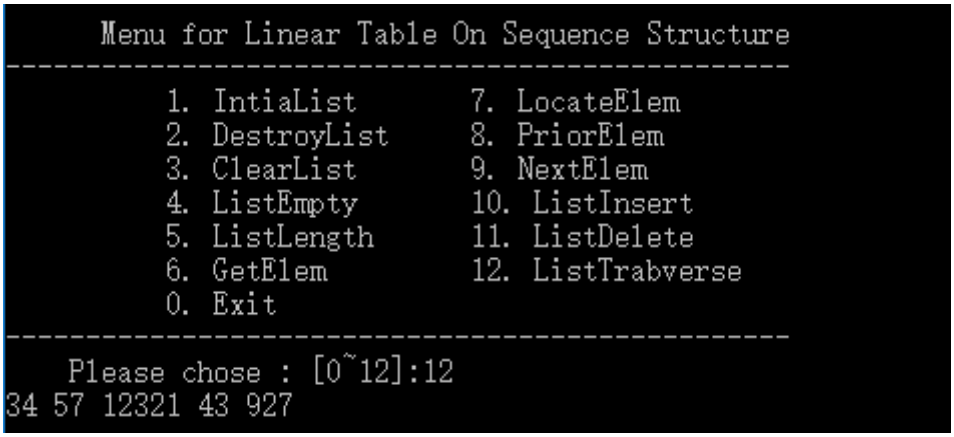

图 2.32 遍历表

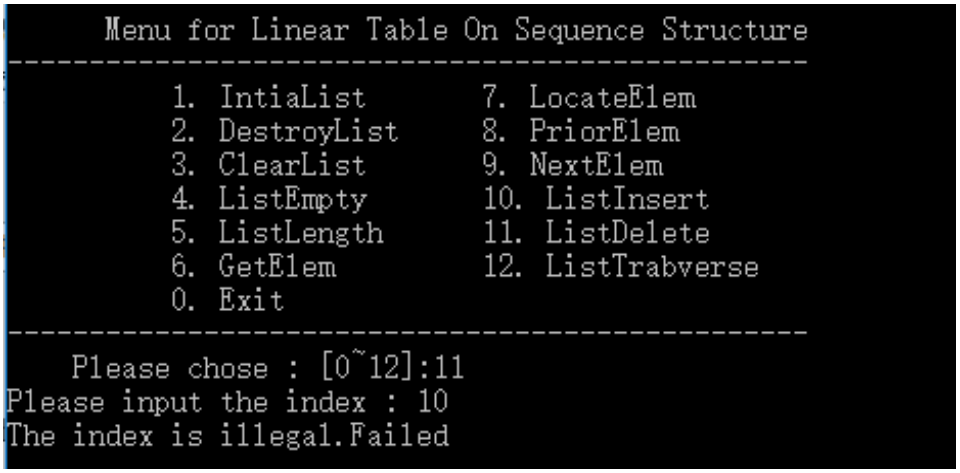

图 2.33 删除失败

12、遍历表，通过循环输出表中的所有元素。如图 2.34 所示：

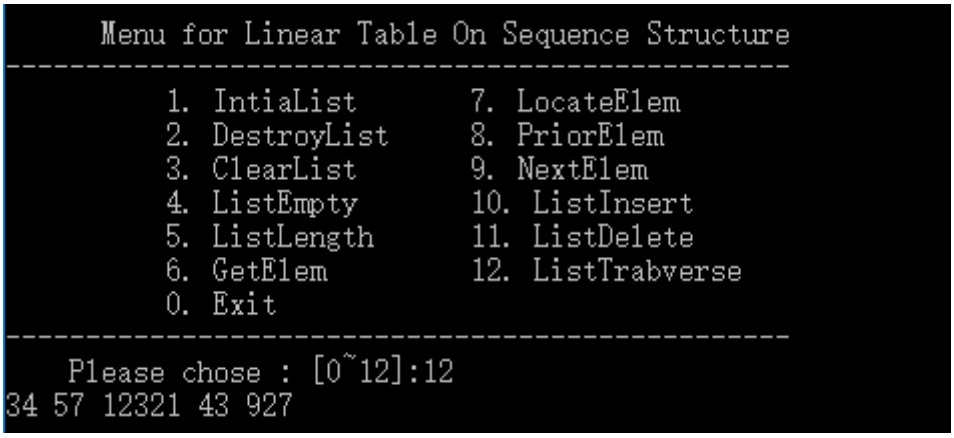

图 2.34 遍历表

2.4 实验小结

在大一第一学期已经有接触过链表了，而且此次实验内容与第一次的实验答题相同，不同点主要就是数据存储方式的差别，但是只要区分了与顺序表的差别以及熟练掌握的链式表结构，本次实验就不会有太大的问题。

虽然在实验过程中遇到了很多小问题，但是都不是关于数据结构的问题，因此这些问题经过自己的反复调试，都得到了解决。

3 基于二叉链表的二叉树实现

3.1 问题描述

通过实验达到：

（4）加深对二叉树的概念、基本运算的理解；

（5）熟练掌握二叉树的逻辑结构与物理结构的关系；

（6）以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。

3.1.1 具体问题

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

⒄前序遍历：函数名称是 PreOrderTraverse(T)；初始条件是二叉树 T 存在；操作结果：先序遍历 t。

⒅中序遍历：函数名称是 InOrderTraverse(T)；初始条件是二叉树 T 存在；操作结果是中序遍历 t。

⒆后序遍历：函数名称是 PostOrderTraverse(T)；初始条件是二叉树 T 存在，操作结果是后序遍历 t。

3.2 系统设计

3.2.1 系统总体设计

（1）通过 switch 语句选择需要使用的功能，然后调用对应的方法实现相关功能。

（2）再通过 while 循环语句实现反复使用菜单中的功能。

（3）设置一个选项让用户能够退出该程序。

3.2.2 方法设计

（1）InitBiTree(&T)

操作结果：构造空二叉树 T。

（2）DestroyBiTree(&T)

初始条件：二叉树 T 已存在。

操作结果：销毁二叉树 T。

（3）CreateBiTree(&T,definition)

初始条件：definition 给出二叉树 T 的定义。

操作结果：按 definition 构造二叉树 T。

（4）ClearBiTree (&T)

初始条件：二叉树 T 存在。

操作结果：将二叉树 T 清空。

（5）BiTreeEmpty(T)

初始条件：二叉树 T 存在。

操作结果：若 T 为空二叉树，则返回 TRUE,否则返回 FALSE.

（6）BiTreeDepth(T)

初始条件：二叉树 T 存在。

操作结果：返回 T 的深度。

（7）Root(T)

初始条件：二叉树 T 已存在。

操作结果：返回 T 的根。

（8）Value(T，e)

初始条件：二叉树 T 已存在，e 是 T 中的某个结点。

操作结果：返回 e 的值。

（9）Assign(T，&e，value)

初始条件：二叉树 T 已存在，e 是 T 中的某个结点。

操作结果：结点 e 赋值为 value。

（10）Parent(T，e)

初始条件：二叉树 T 已存在，e 是 T 中的某个结点。

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

（11）LeftChild(T，e)

初始条件：二叉树 T 存在，e 是 T 中某个节点。

操作结果：返回 e 的左孩子结点指针。若 e 无左孩子，则返回 NULL。

（12）RightChild(T，e)

初始条件：二叉树 T 已存在，e 是 T 中某个结点。

操作结果：返回 e 的右孩子结点指针。若 e 无右孩子，则返回 NULL。

（13）LeftSibling(T，e)

初始条件：二叉树 T 存在，e 是 T 中某个结点。

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

（14）RightSibling(T，e)

初始条件：二叉树 T 已存在，e 是 T 中某个结点。

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

（15）InsertChild(T,p,LR,c)

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

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

（16）DeleteChild(T，p，LR)

初始条件：二叉树 T 存在，p 指向 T 中的某个结点，LR 为 0 或 1。

操作结果：根据 LR 为 0 或者 1，删除 c 为 T 中 p 所指结点的左或右子树。

（17）PreOrderTraverse（T）

初始条件：二叉树 T 存在，Visit 是对结点操作的应用函数。

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

（18）InOrderTraverse（T）

初始条件：二叉树 T 存在，Visit 是对结点操作的应用函数。

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

（19）PostOrderTraverse（T）

初始条件：二叉树 T 存在，Visit 是对结点操作的应用函数。

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

3.3 系统实现

编程环境：

操作系统：WIN 10

IDE：Dev—C++，C 语言。

主模块：包含各变量申明、运行各个方法需要的输入与输出以及使用 switch 函数实现各函数的调用。程序运行主模块图如图 3.1 所示：

图 3.1 主模块图

功能模块：

1、初始化二叉树：为一棵二叉树分配存储空间，若存储分配成功，二叉树头节点 T 为存储空间基址，返回 OK；否则返回 ERROR。如图 3.2 所示：

图 3.2 初始化二叉树

2、销毁二叉树：释放存储空间，使头地址为空。如图 3.3 所示：

图 3.3 销毁二叉树

当表并不存在时，销毁表会失败，如图 3.4 所示：

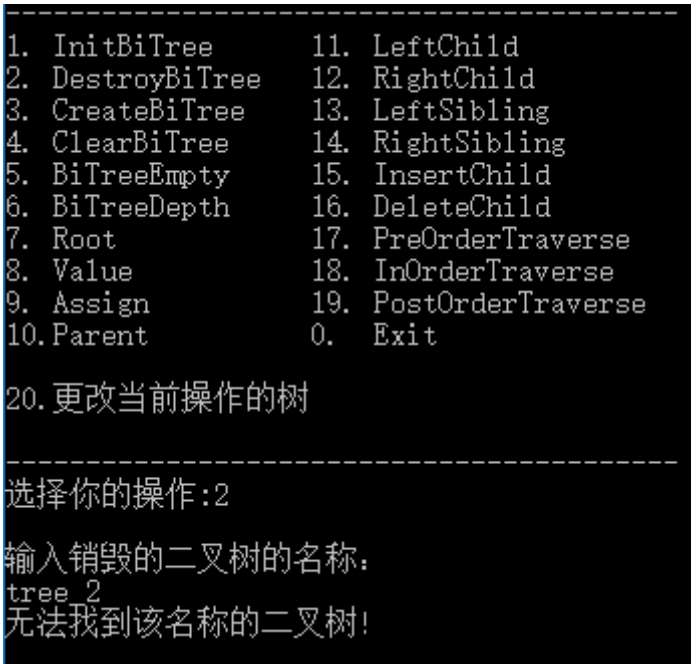

图 3.4 销毁二叉树失败

3、创建二叉树，通过用户输入前序遍历来构造二叉树。如图 3.5，3.6 所示：

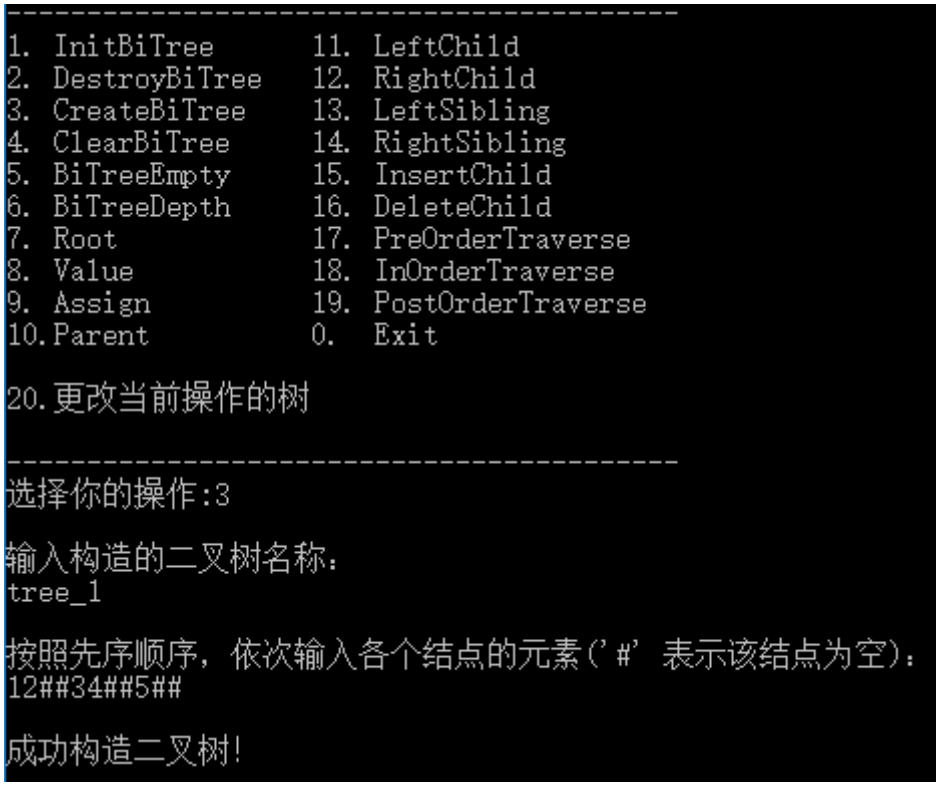

图 3.5 新建二叉树

图 3.6 遍历二叉树

4、清空二叉树，初始条件是二叉树 T 存在，操作结果是利用循环将二叉树 T 的每个节点清空，T 头节点再指向空。如图 3.7，3.8，3.9，3.10 所示：

（1）使用之前创建的表；

（2）判断是否为空；

（3）清空表，再次判断是否为空

图 3.7 二叉树

图 3.8 判断是否为空

图 3.9 清空二叉树

图 3.10 判断是否为空

5、判断二叉树是否为空，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，如图 3.11，3.12，3.13 所示：

（1）使用之前初始化的表

（2）清空二叉树，再次遍历

图 3.11 二叉树

图 3.12 清空二叉树

图 3.13 遍历二叉树

6、计算二叉树深度，遍历二叉树，找到二叉树最深的字数节点，返回深度。如图 3.14 所示：

（1）使用之前的二叉树；

（2）计算深度；

图 3.14 计算二叉树深度

7、求二叉树的根，二叉树的根储存在 T 头结点的左子树上，返回头结点的左子树上的元素。如图 3.15 所示：

（1）使用之前二叉树；

（2）求二叉树的根；

图 3.15 求二叉树的根

8、获得节点的值，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。如图 3.16，3.17 所示：

（1）使用之前的二叉树，为其中的一个节点赋值；

（2）求这个节点的值；

```
------------------------------------------
1. InitBiTree       11. LeftChild
2. DestroyBiTree    12. RightChild
3. CreateBiTree     13. LeftSibling
4. ClearBiTree      14. RightSibling
5. BiTreeEmpty      15. InsertChild
6. BiTreeDepth      16. DeleteChild
7. Root             17. PreOrderTraverse
8. Value            18. InOrderTraverse
9. Assign           19. PostOrderTraverse
10.Parent           0.  Exit

20.更改当前操作的树

------------------------------------------
选择你的操作:9

输入你希望赋值结点(char)
2

输入你希望附的值(char):
x

给结点赋值成功!
```

图 3.16 为节点赋值

```
1. InitBiTree       11. LeftChild
2. DestroyBiTree    12. RightChild
3. CreateBiTree     13. LeftSibling
4. ClearBiTree      14. RightSibling
5. BiTreeEmpty      15. InsertChild
6. BiTreeDepth      16. DeleteChild
7. Root             17. PreOrderTraverse
8. Value            18. InOrderTraverse
9. Assign           19. PostOrderTraverse
10.Parent           0.  Exit

20.更改当前操作的树

------------------------------------------
选择你的操作:8

输入你想查找的结点标记:
2

在当前二叉树中，此二叉树结点2的值为:x
```

图 3.17 找到对应节点的值

9、为节点赋值，e 是 T 中的某个结点；操作结果是结点 e 赋值为 value。如图 3.18 所示：

（1）使用之前的二叉树；

（2）为其中一个节点赋值；

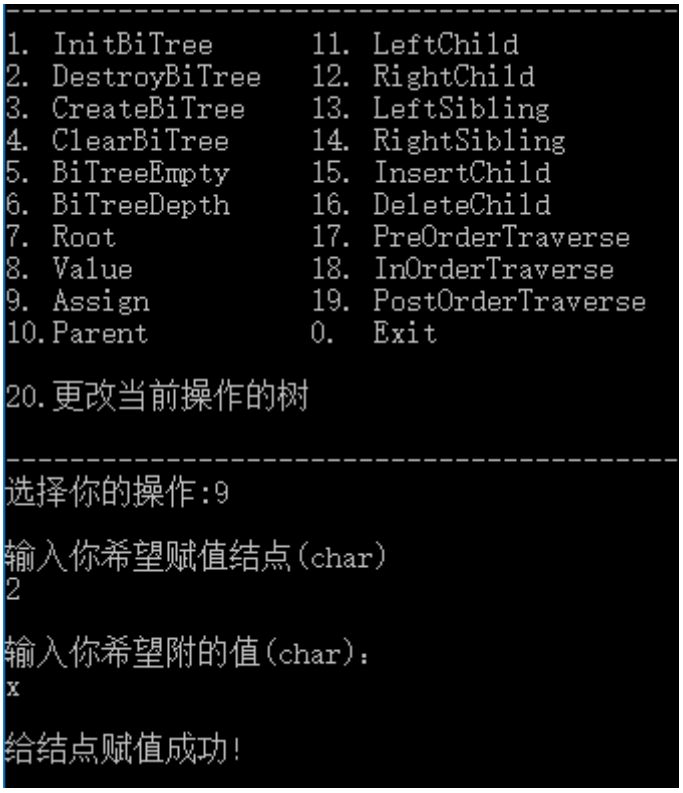

图 3.18 为节点赋值

10、求二叉树节点的父节点，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。如图 3.19，3.20 所示：

（1）使用之前的二叉树；

（2）获取一个节点的父节点；

（3）获取失败；

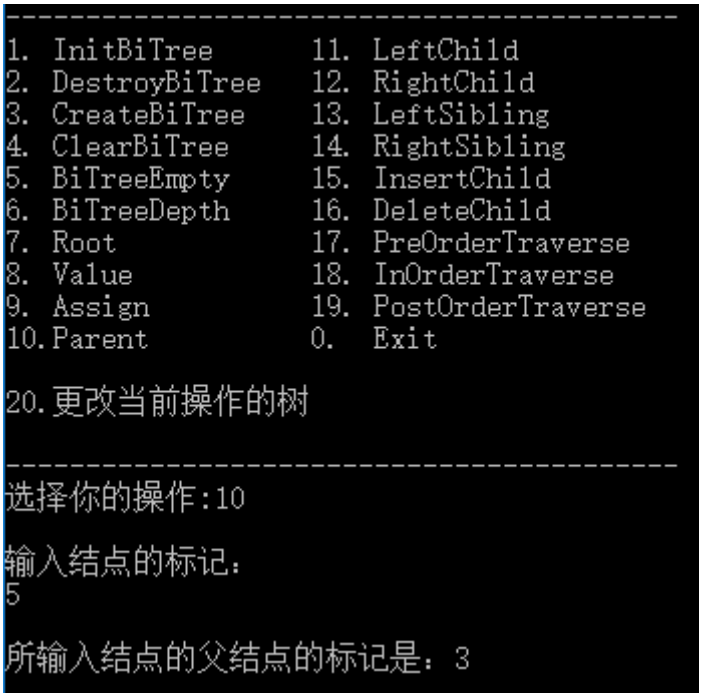

图 3.19 新建表

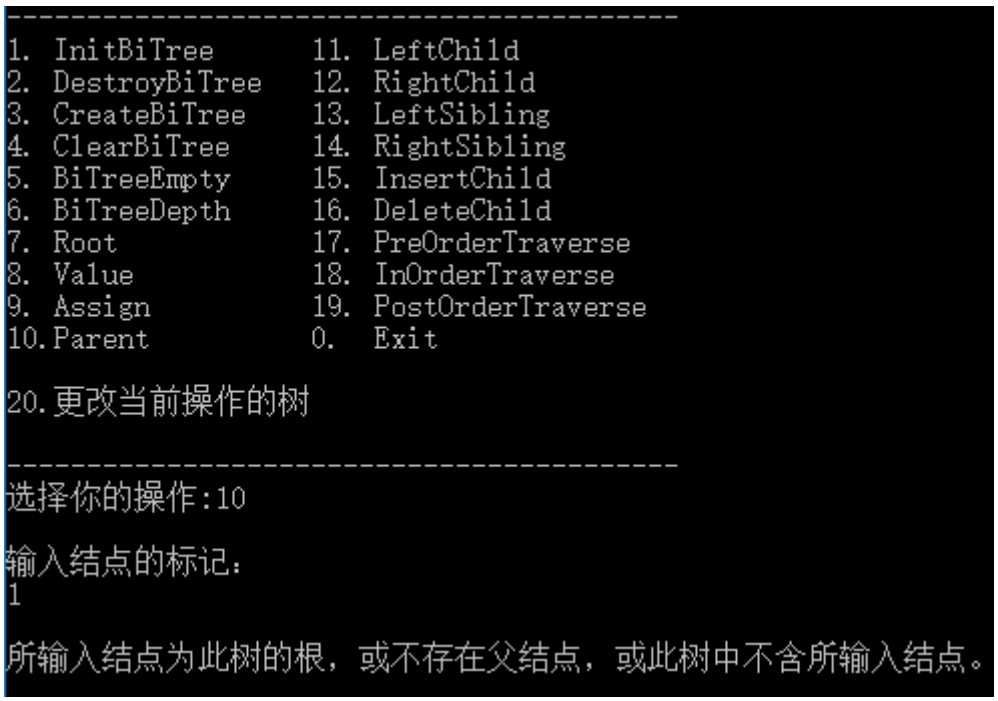

图 3.20 获取失败

11、获得左孩子结点，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。如图 3.21，3.22，3.23 所示：

（1）新建一棵树（前序、中序遍历）；

（2）获取某个节点的左孩子；

图 3.21 前序遍历

图 3.22 中序遍历

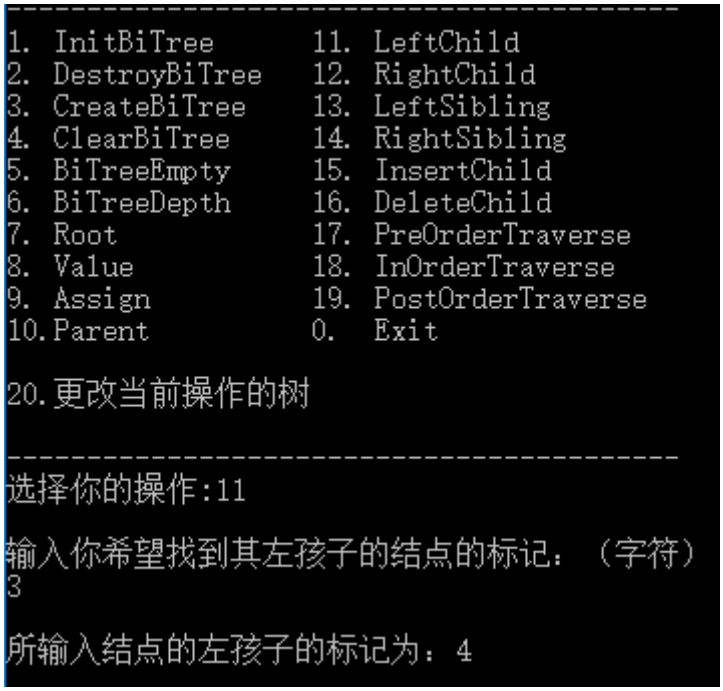

图 3.23 左孩子

12、获得右孩子结点，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 的右孩子结点指针。若 e 无右孩子，则返回 NULL。如图 3.24 所示：

（1）使用之前的树；

（2）获取某个节点的右孩子；

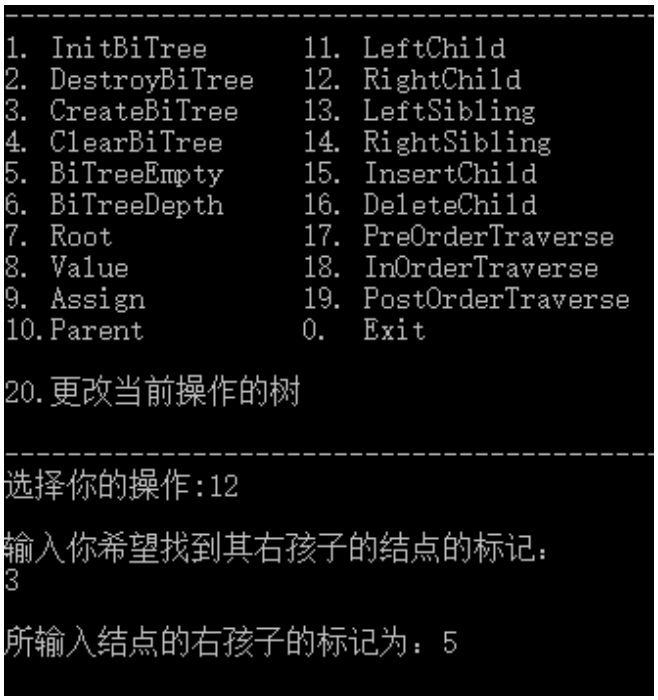

图 3.24 右孩子

13、获得左兄弟结点，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。如图 3.25 所示：

（1）使用之前的树；

（2）获取某个节点的左兄弟；

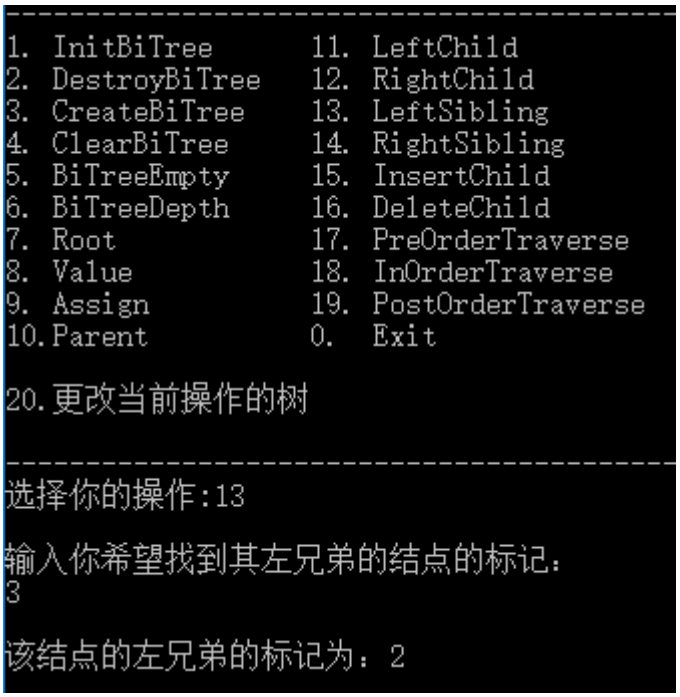

图 3.25 左兄弟

14、获得右兄弟结点，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 的右兄弟结点指针。若 e 是 T 的右孩子或者无右兄弟，则返回 NULL。如图 3.26 所示：

（1）使用之前的树；

（2）获取某个节点的右兄弟；

图 3.26 右兄弟

15、插入子树，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。如图 3.27，3.28 所示：

（1）使用之前的树；

（2）插入一个子树；

（3）遍历；

```
-----------------------------------------
1. InitBiTree      11. LeftChild
2. DestroyBiTree   12. RightChild
3. CreateBiTree    13. LeftSibling
4. ClearBiTree     14. RightSibling
5. BiTreeEmpty     15. InsertChild
6. BiTreeDepth     16. DeleteChild
7. Root            17. PreOrderTraverse
8. Value           18. InOrderTraverse
9. Assign          19. PostOrderTraverse
10.Parent          0.  Exit

20.更改当前操作的树

-----------------------------------------
选择你的操作:15

请输入你希望插入子树的结点（字符）
2
插入到该结点的左子树，请输入0；右子树，请输入1。
0

请按照先序输入插入树的各个结点的数据（char），'#'表示此节点不存在!
a

成功插入子树!
```

图 3.27 插入子树

```
-----------------------------------------
1. InitBiTree      11. LeftChild
2. DestroyBiTree   12. RightChild
3. CreateBiTree    13. LeftSibling
4. ClearBiTree     14. RightSibling
5. BiTreeEmpty     15. InsertChild
6. BiTreeDepth     16. DeleteChild
7. Root            17. PreOrderTraverse
8. Value           18. InOrderTraverse
9. Assign          19. PostOrderTraverse
10.Parent          0.  Exit

20.更改当前操作的树

-----------------------------------------
选择你的操作:17

----------------所有元素 ----------------------
1 2 a 3 4 5
----------------- 结束 ----------------------
```

图 3.28 遍历

16、删除子树，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。如图 3.29，3.30 所示：

（1）使用之前的树；

（2）删除一个子树；

（3）遍历；

```
------------------------------------------
1. InitBiTree      11. LeftChild
2. DestroyBiTree   12. RightChild
3. CreateBiTree    13. LeftSibling
4. ClearBiTree     14. RightSibling
5. BiTreeEmpty     15. InsertChild
6. BiTreeDepth     16. DeleteChild
7. Root            17. PreOrderTraverse
8. Value           18. InOrderTraverse
9. Assign          19. PostOrderTraverse
10.Parent          0.  Exit

20.更改当前操作的树

------------------------------------------
选择你的操作:16

请输入你希望删除子节点的字符标记（char）
1

删除该结点的左结点，请输入0；删除该结点的右结点，请输入1
1

成功删除结点!
```

图 3.29 删除子树

```
------------------------------------------
1. InitBiTree      11. LeftChild
2. DestroyBiTree   12. RightChild
3. CreateBiTree    13. LeftSibling
4. ClearBiTree     14. RightSibling
5. BiTreeEmpty     15. InsertChild
6. BiTreeDepth     16. DeleteChild
7. Root            17. PreOrderTraverse
8. Value           18. InOrderTraverse
9. Assign          19. PostOrderTraverse
10.Parent          0.  Exit

20.更改当前操作的树

------------------------------------------
选择你的操作:17

-----------------所有元素 -----------------------
1 2 a
------------------ 结束 -----------------------
```

图 3.30 遍历

17、前序遍历，输出二叉树的先序遍历。如图 3.31 所示：

图 3.31 先序遍历

18、中序遍历，输出二叉树的中序遍历。如图 3.32 所示：

图 3.31 中序遍历

19、后序遍历，输出二叉树的后序遍历。如图 3.33 所示：

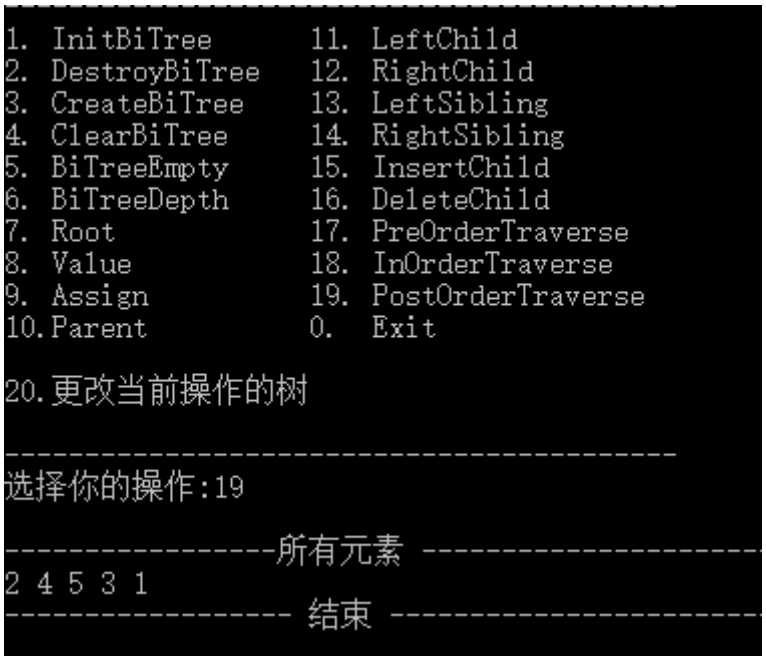

图 3.31 后序遍历

3.4 实验小结

这是我目前在课程学习中接触到的规模最大的实验，需要实现的内容非常多，但是我通过自己自主学习、查找资料，慢慢的将整个程序写出来了。虽然花的时间很长，但是体会到了以后自己写代码不仅仅需要技术，还需要坚持和耐心。只要这样，再大的工程都能够完成。

4 基于邻接表的图实现

4.1 问题描述

通过实验达到：

（1） 加深对图的概念、基本运算的理解

（2） 熟练掌握图的逻辑结构与物理结构的关系

（3） 以邻接表作为物理结构，熟练掌握图基本运算的实现

4.1.1 具体问题

依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等 13 种基本运算，具体运算功能定义如下。

⑴创建图：函数名称是 CreateGraph(&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 DestroyGraph (G)；初始条件图 G 已存在；操作结果是销毁图 G。

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

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

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

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

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

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

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

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

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

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

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

4.2 系统设计

4.2.1 系统总体设计

（1）通过 switch 语句选择需要使用的功能，然后调用对应的方法实现相关功能。

（2）再通过 while 循环语句实现反复使用菜单中的功能。

（3）设置一个选项让用户能够退出该程序。

3.2.2 方法设计

（1）CreateCraph(&G,V,VR)

操作结果：创建图。

（2）DestroyGraph (G)

初始条件：图 G 已存在。

操作结果：销毁图 G。

（3）LocateVex(G,u)

初始条件：图 G 存在，u 和 G 中的顶点具有相同特征。

操作结果：若 u 在图 G 中存在，返回顶点 u 的位置信息，否则返回其它信息

（4）GetVex (G,v)

初始条件：图 G 存在，v 是 G 中的某个顶点。

操作结果：返回 v 的值。

（5）PutVex (G,v,value)

初始条件：图 G 存在，v 是 G 中的某个顶点。

操作结果：操作结果是对 v 赋值 value.

（6）FirstAdjVex(&G, v)

初始条件：图 G 存在，v 是 G 的一个顶点。

操作结果：返回 v 的第一个邻接顶点，如果 v 没有邻接顶点，返回“空”。

（7）NextAdjVex(&G, v, w)

初始条件：图 G 存在，v 是 G 的一个顶点,w 是 v 的邻接顶点。

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

（8）InsertVex(&G,v)

初始条件：图 G 存在，v 和 G 中的顶点具有相同特征。

操作结果：在图 G 中增加新顶点 v。

（9）DeleteVex(&G,v)

初始条件：图 G 存在，v 是 G 的一个顶点。

操作结果：在图 G 中删除顶点 v 和与 v 相关的弧。

（10）InsertArc(&G,v,w)

初始条件：图 G 存在，v、w 是 G 的顶点。

操作结果：在图 G 中增加弧<v,w>，如果图 G 是无向图，还需要增加<w,v>。

（11）DeleteArc(&G,v,w)

初始条件：图 G 存在，v、w 是 G 的顶点。

操作结果：在图 G 中删除弧<v,w>，如果图 G 是无向图，还需要删除<w,v>。

（12）DFSTraverse(G,visit())；

初始条件：图 G 存在。

操作结果：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。

（13）BFSTraverse(G,visit())；

初始条件：图 G 存在。

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

4.3 系统实现

编程环境：

操作系统：WIN 10

IDE：Dev—C++，C 语言。

主模块：包含各变量申明、运行各个方法需要的输入与输出以及使用 switch 函数实现各函数的调用。程序运行主模块图如图 4.1 所示：

图 4.1 主模块图

功能模块：

1、初始化图：为一图分配存储空间，然后按照引导构建一个图，若成功，返回成功。如图 4.2 所示：

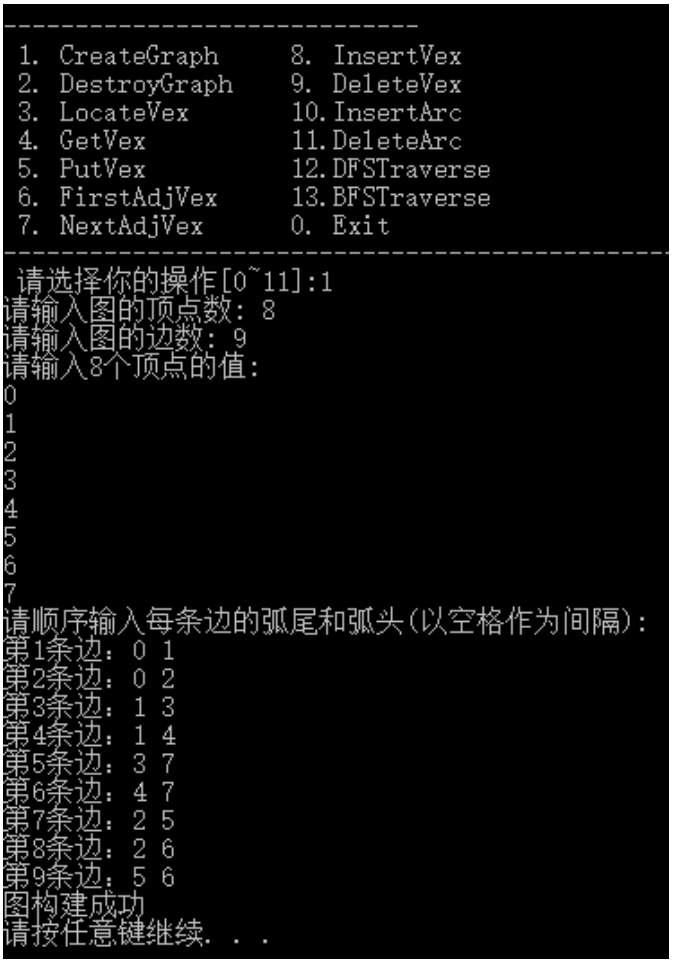

图 4.2 初始化二叉树

2、销毁图：释放存储空间，使头地址为空。如图 4.3 所示：

图 4.3 销毁二叉树

当表并不存在时，销毁表会失败，如图 4.4 所示：

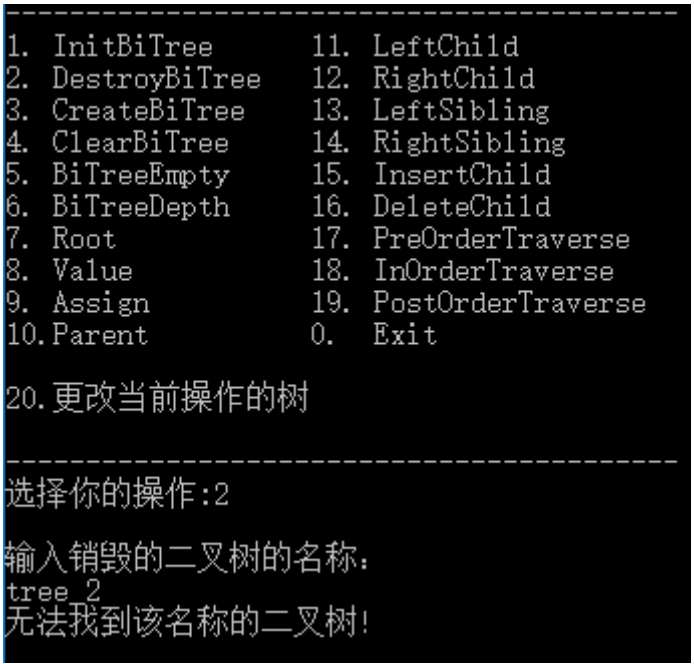

图 4.4 销毁二叉树失败

3、定位顶点：读取用户输入前序遍历来构造二叉树。如图 4.5 所示：

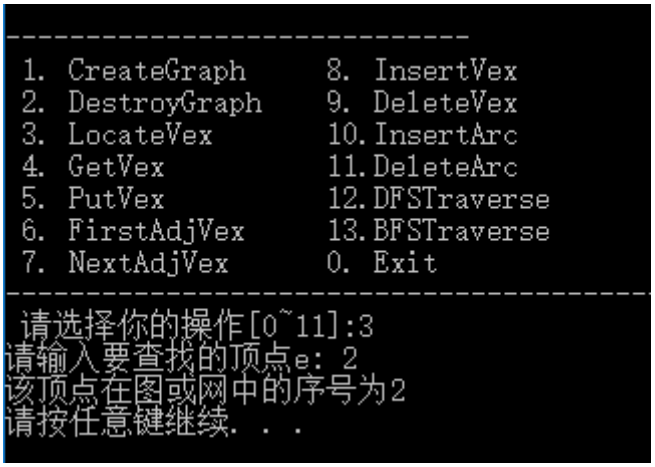

图 4.5 新建二叉树

4、得到顶点的值：读取用户输入的顶点序号，返回为这个顶赋予的值。如图 4.6，4.7 所示：

（1）为顶点赋值；

（2）读取顶点的值

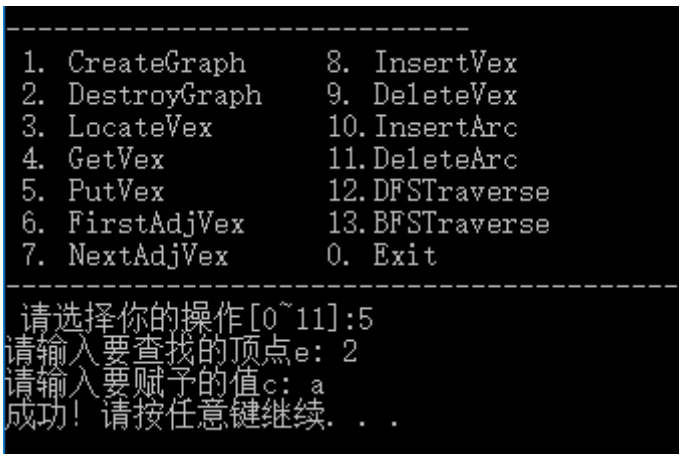

图 4.6 为顶点赋值

图 4.7 获取顶点的值

5、获取顶点的值：读取用户输入的顶点序号，为该序号对应的顶点赋值。如图 4.8 所示：

图 4.8 二叉树

6、获取第一个邻接顶点：读取用户输入的节点序号，返回该节点的第一个邻接点。如图 4.9 所示：

（1）使用之前的图；

（2）获取对应顶点的第一个邻接点；

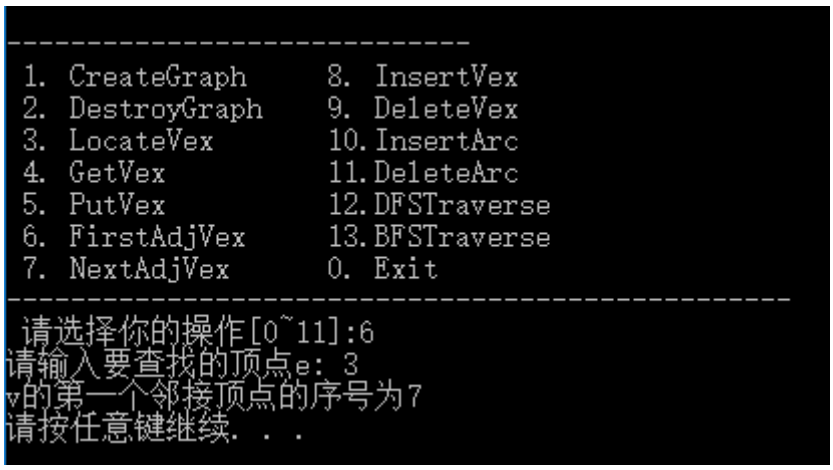

图 4.9 获取第一个邻接点

7、返回 v 的（相对于 w）下一个邻接顶点：读取用户输入的两个顶点 v 和 w，返回 v 相对于 w 的下一个邻接点，如果是最后一个邻接点则说明是最后一个邻接点。如图 4.10，4.11 所示：

（1）使用之前图；

（2）返回下一个邻接点；

图 4.10 下一个邻接点

图 4.11 最后一个邻接点

8、插入顶点，在用户所创建的图中插入一个新的顶点。如图 4.12，4.13 所示：

（1）在之前的图中插入一个顶点；

（2）遍历这个图；

图 4.12 插入顶点

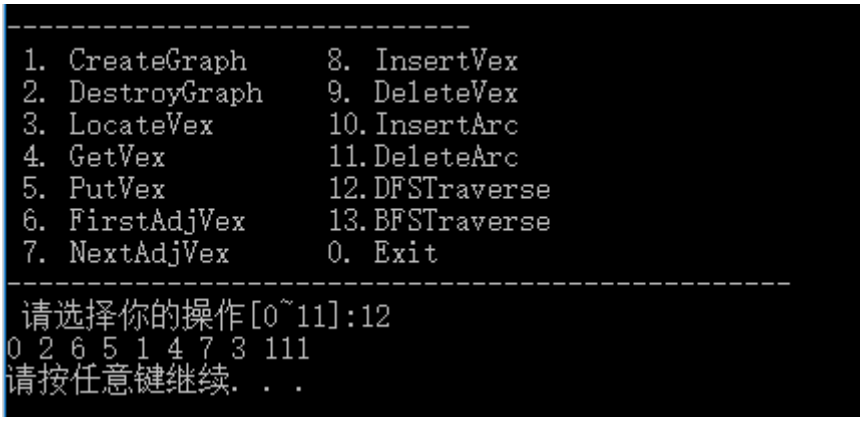

图 4.13 遍历图

9、删除顶点：读取用户输入的顶点序号，然后删除对应的顶点以及预定点相连的边。如图 4.14，4.15 所示：

（1）使用之前的图，删除其中的一个顶点；

（2）遍历；

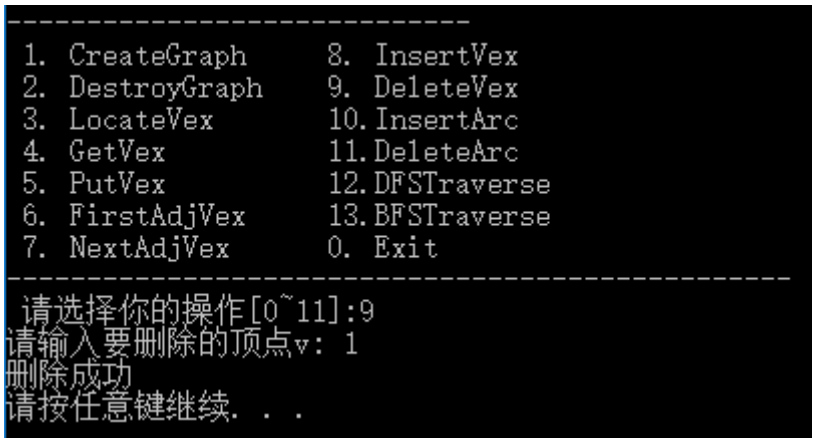

图 4.14 删除顶点

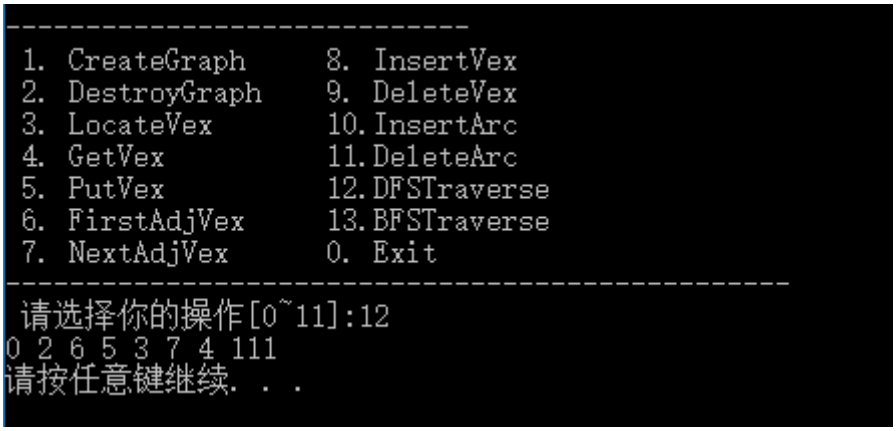

图 4.15 遍历

10、插入弧（边）：读取用户输入的两个顶点序号，在这两个顶点之间添加一条边。如图 4.16，4.17 所示：

（1）使用之前的图，在 2 和 111 之间添加一条边；

（2）遍历；

图 4.16 插入边

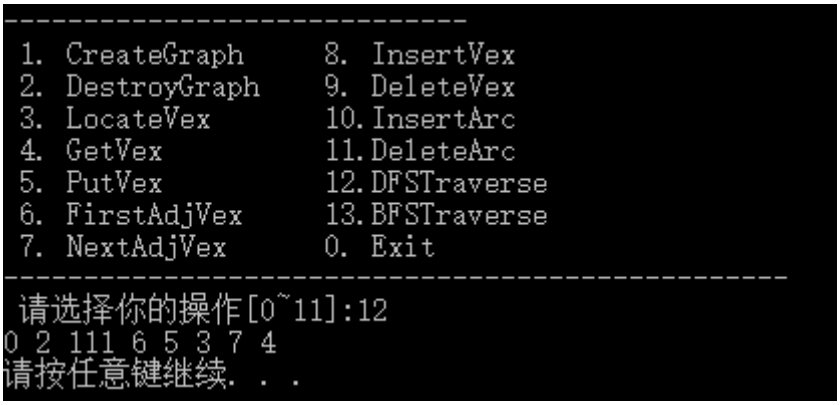

图 4.17 遍历

11、DFS：深度优先遍历图。如图 4.18，4.19 所示：

（1）使用最开始创建的图；

（2）删除一条边以显示两种遍历区别：

（3）DFS；

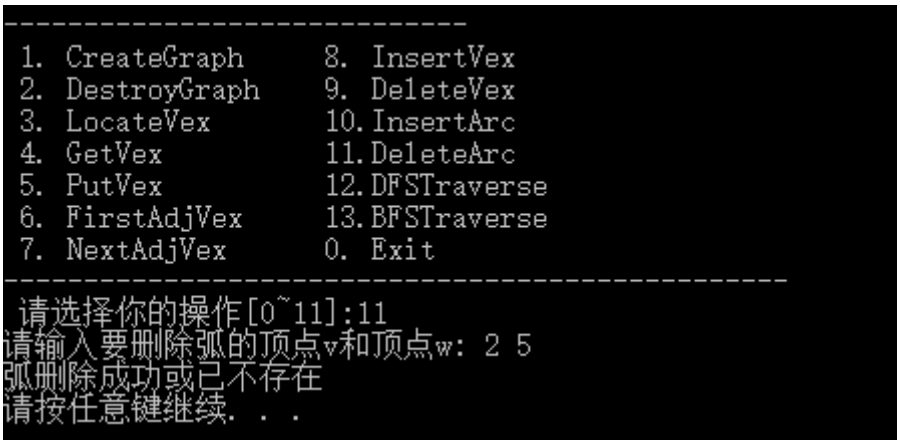

图 4.18 删除一条边

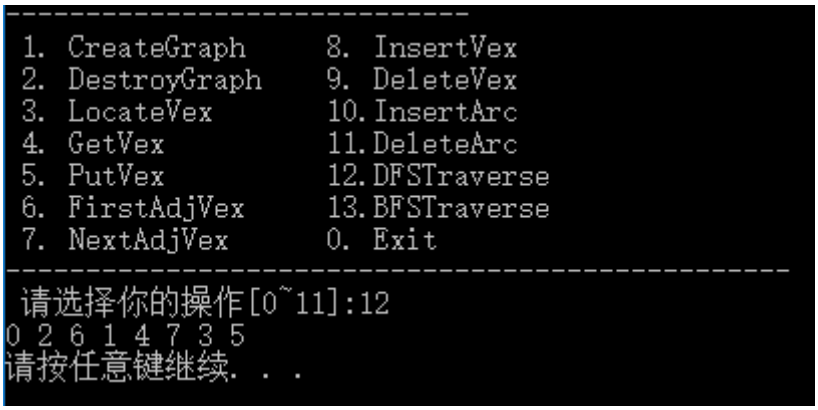

图 4.19 DFS

12、BFS：广度优先遍历图。如图 4.20 所示：

（1）使用之前的树；

（2）BFS；

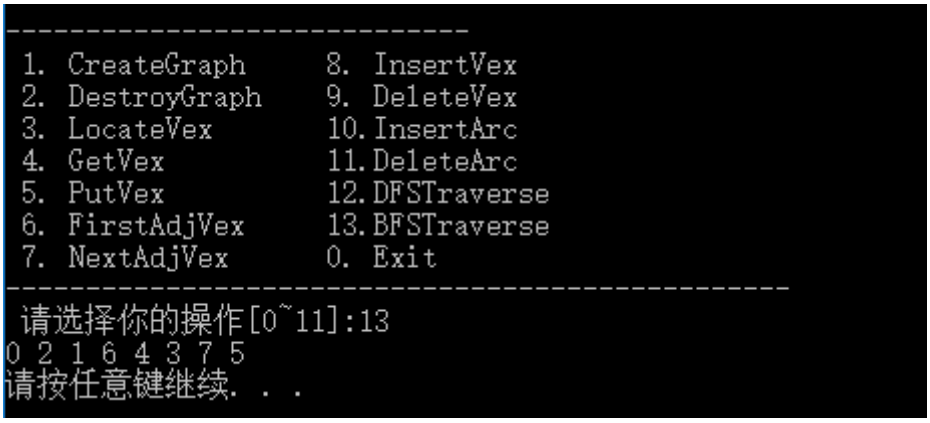

图 4.20 BFS

4.4 实验小结

本课程的最后一个实验，在有着前几个实验锻炼的基础上，这个实验还是进行的非常顺利，没有遇到太多的 bug 或者不懂得地方，希望在以后能够多多运用这一段时间所学到的东西。