

**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： 2018年 11 月 12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](#_Toc531184541)

[1.1 问题描述 1](#_Toc531184542)

[1.2 系统设计 3](#_Toc531184543)

[1.3 系统实现 4](#_Toc531184544)

[1.4 实验小结 17](#_Toc531184545)

[2基于链式存储结构的线性表实现 18](#_Toc531184546)

[2.1 问题描述 18](#_Toc531184547)

[2.2 系统设计 19](#_Toc531184548)

[2.3 系统实现 21](#_Toc531184549)

[2.4 实验小结 34](#_Toc531184550)

[附录A 基于顺序存储结构线性表实现的源程序 36](#_Toc531184551)

[附录B 基于链式存储结构线性表实现的源程序 45](#_Toc531184552)

# 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：

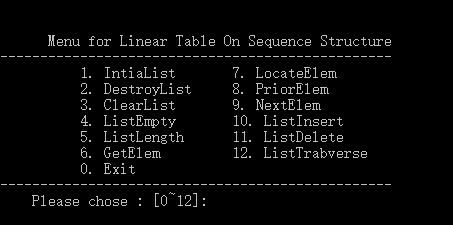


图1.1主模块图

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：

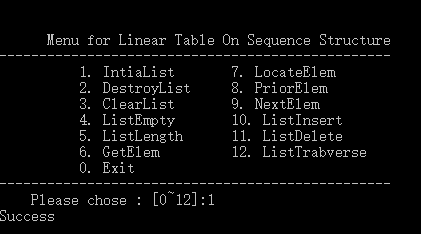


图1.2 初始化表

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：

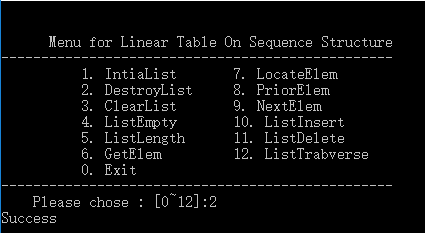


图1.3 销毁表

当表并不存在时，销毁表会失败，如图1.4所示：

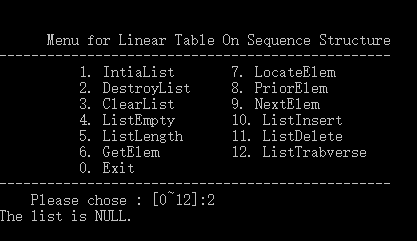


图1.4 销毁表失败

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；



图1.5 新建表

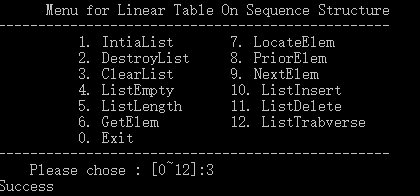


图1.6 清空表

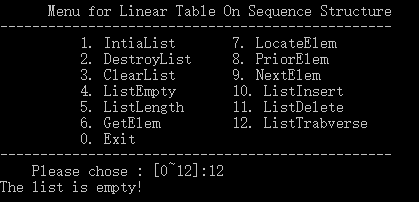


图1.7 遍历表

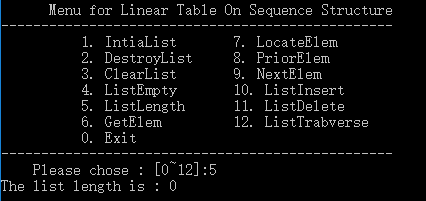


图1.8 求表长

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

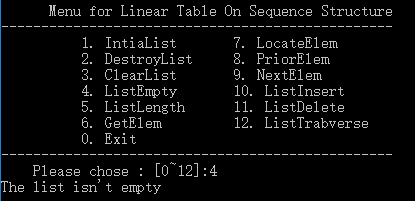


图1.10 判断是否为空

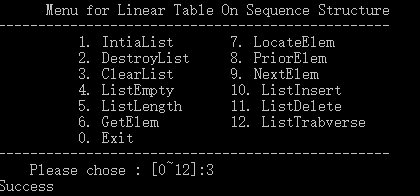


图1.11 清空表

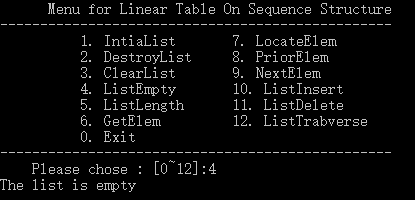


图1.12 判断是否为空

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

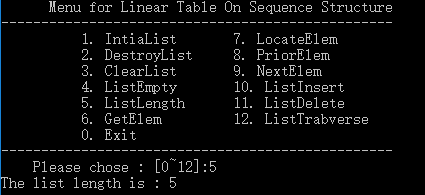


图1.14 求表长

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

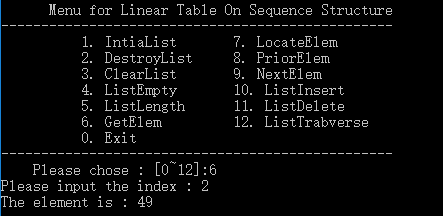


图1.16 获取元素

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；

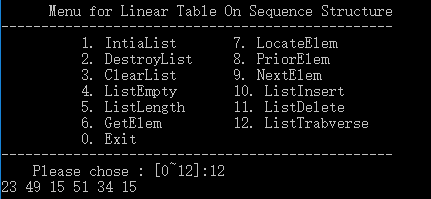


图1.17 新建表

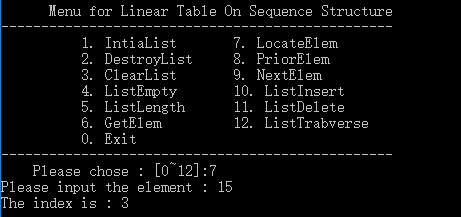


图1.18 搜索元素

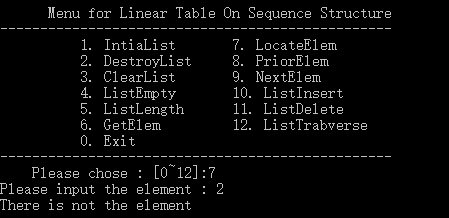


图1.19 搜索不存在元素

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；

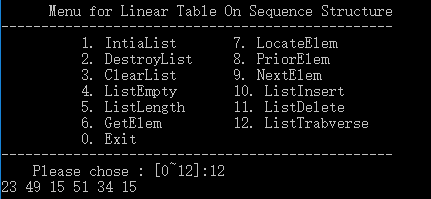


图1.20 新建表

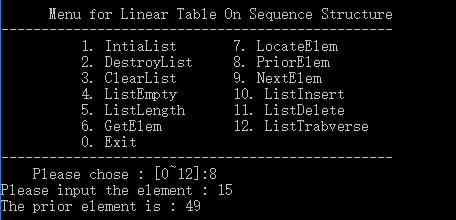


图1.21 找前驱

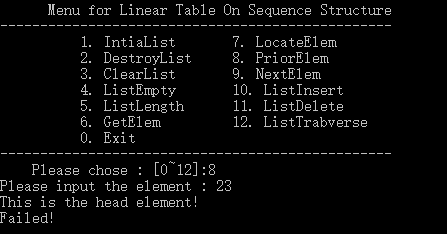


图1.22 找首元素前驱

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

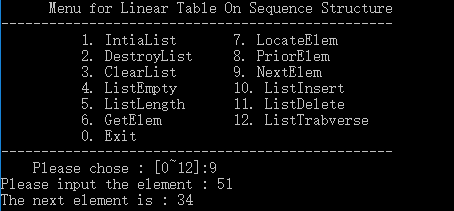


图1.24 获取后继

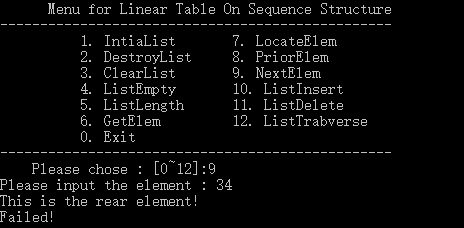


图1.25 获取尾元素后继

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；



图1.26 新建表

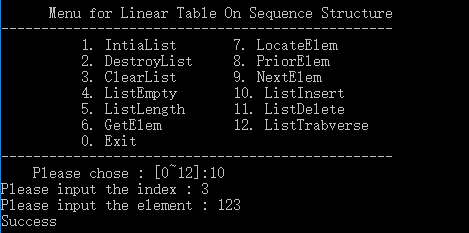


图1.27 插入元素

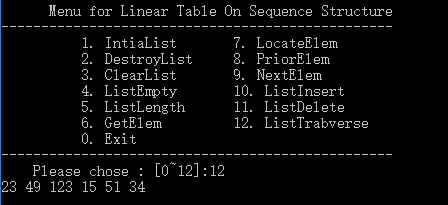


图1.28 遍历表

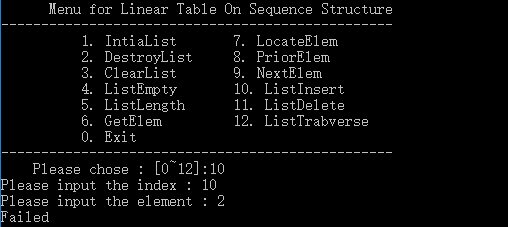


图1.29 插入失败

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；

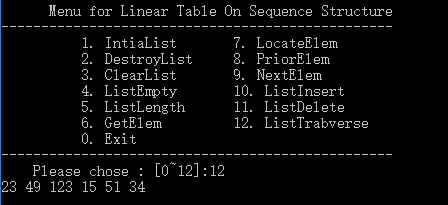


图1.30 初始表

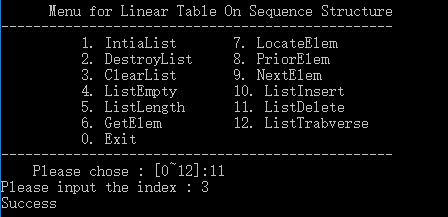


图1.31 删除元素



图1.32 遍历表

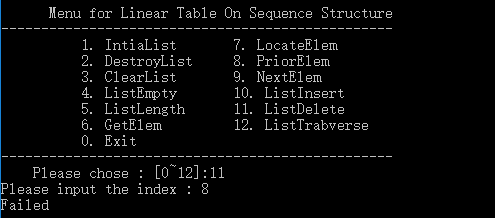


图1.33 删除失败

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：

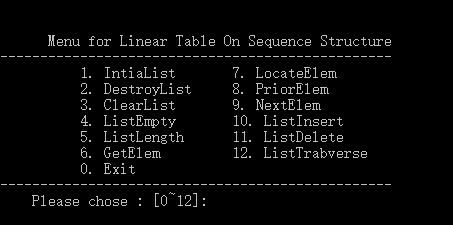


图2.1主模块图

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：

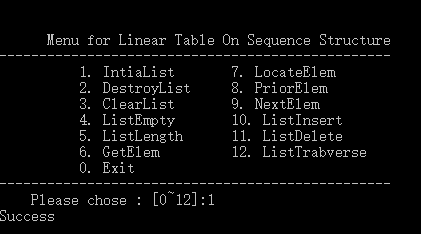


图2.2 初始化表

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：

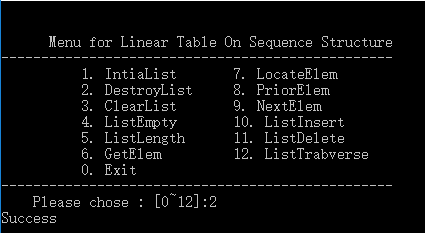


图2.3 销毁表

当表并不存在时，销毁表会失败，如图2.4所示：

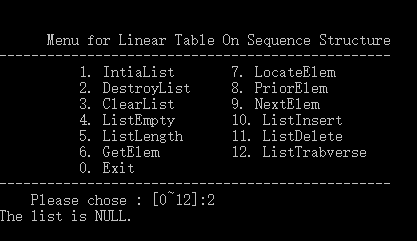


图2.4 销毁表失败

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；



图2.5 新建表

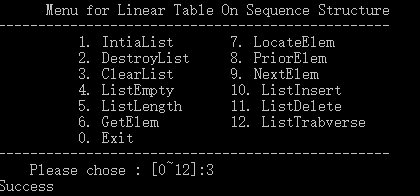


图2.6 清空表

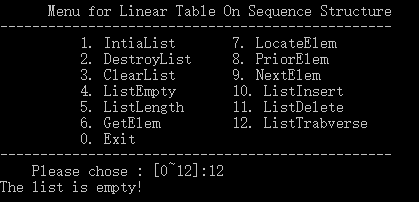


图2.7 遍历表

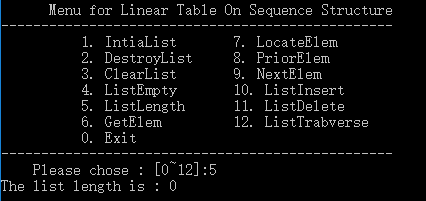


图2.8 求表长

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

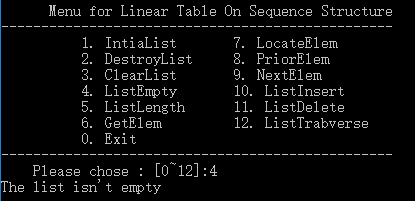


图2.10 判断是否为空

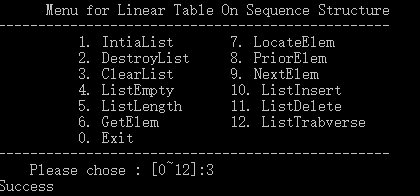


图2.11 清空表

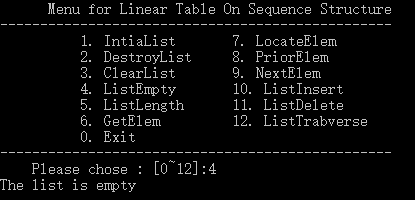


图2.12 判断是否为空

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

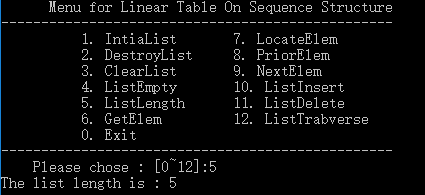


图2.14 求表长

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

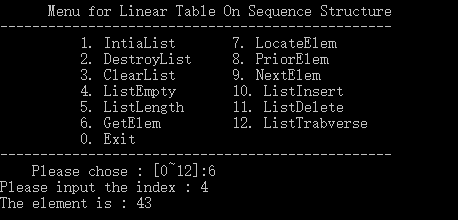


图2.16 获取元素

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

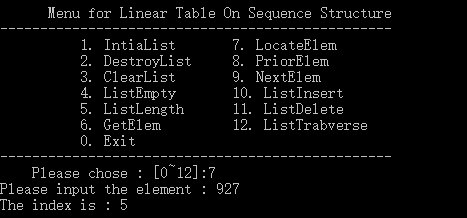


图2.18 搜索元素

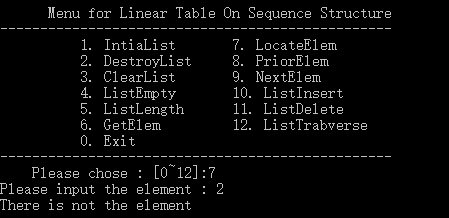


图2.19 搜索不存在元素

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；



图2.20 新建表

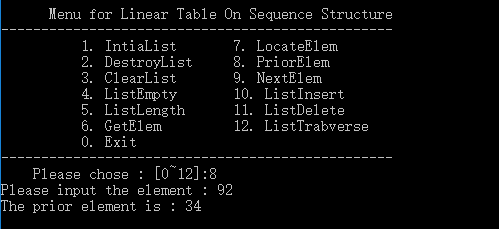


图2.21 找前驱

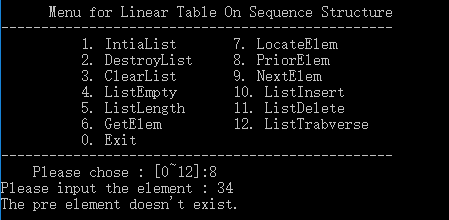


图2.22 找首元素前驱

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；



图2.23 新建表

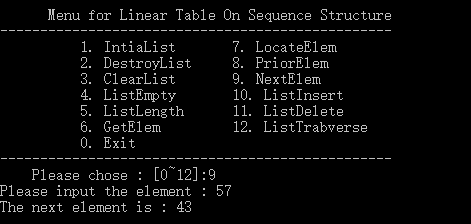


图2.24 获取后继

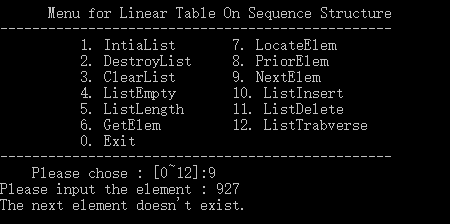


图2.25 获取尾元素后继

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；



图2.26 新建表

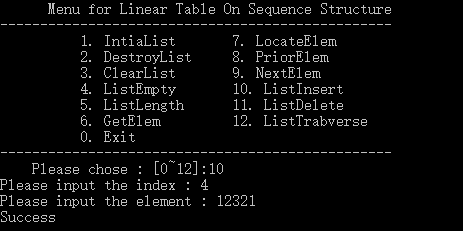


图2.27 插入元素

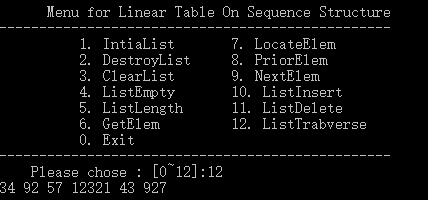


图2.28 遍历表

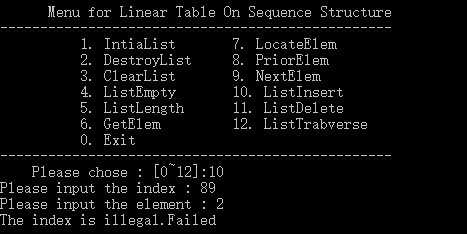


图2.29 插入失败

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。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；

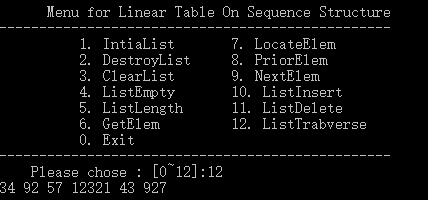


图2.30 初始表

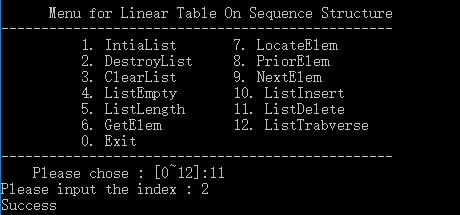


图2.31 删除元素

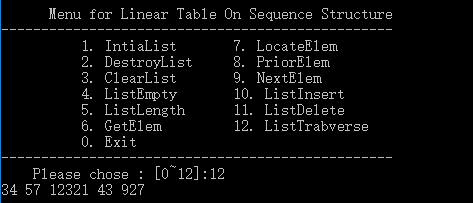


图2.32 遍历表

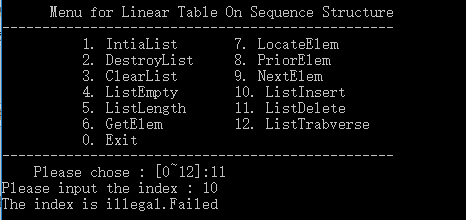


图2.33 删除失败

12、遍历表，通过循环输出表中的所有元素。如图2.34所示：

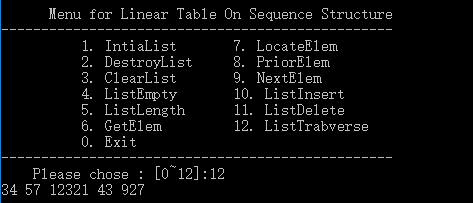


图2.34 遍历表

## 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。

附录A 基于顺序存储结构线性表实现的源程序

#include "stdio.h"

#include "malloc.h"

#include "stdlib.h"

/\* DATA DEFINE \*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

/\* TYPE DEFINE \*/

typedef int status;

typedef int ElemType;

#define LIST\_INIT\_SIZE 1

#define LISTINCREMENT 10

typedef struct {

ElemType \* elem;

int length;

int listSize;

} SqList;

status IntiaList(SqList & L);

status DestroyList(SqList \* L);

status ClearList(SqList & L);

status ListEmpty(SqList L);

int ListLength(SqList L);

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

status LocateElem(SqList L,ElemType e, status (\*Compare)(ElemType a, ElemType b));

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

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

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

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

status ListTrabverse(SqList L);

status Compare(ElemType a, ElemType b);

extern bool isNull = TRUE;

int main() {

SqList L;

int op=1;

while(op){

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

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

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

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

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

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

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

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

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

printf(" 0. Exit\n");

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

printf(" Please chose : [0~12]:");

scanf("%d",&op);

getchar();

switch(op) {

case 1:

if (IntiaList(L) == OK) {

printf("Success\n");

}

else

printf("failed\n");

getchar();

break;

case 2:

if (isNull) {

printf("The list is NULL.");

getchar();

break;

}

if (DestroyList(&L) == OK) {

printf("Success\n");

}

else

printf("Failed\n");

getchar();

break;

case 3:

if (ClearList(L) == OK)

printf("Success\n");

else

printf("Failed\n");

getchar();

break;

case 4:

if (ListEmpty(L) == TRUE)

printf("The list is empty\n");

else if (!isNull)

printf("The list isn't empty\n");

getchar();

break;

case 5:

if (!isNull)

printf("The list length is : %d\n", ListLength(L));

else

ListLength(L);

getchar();

break;

case 6:

int index;

ElemType e;

if (isNull) {

printf("The list is NULL.");

getchar();

break;

}

printf("Please input the index : ");

scanf("%d", &index);

getchar();

if (GetElem(L, index, e) != ERROR)

printf("The element is : %d\n", e);

else

printf("Failed\n");

getchar();

break;

case 7:

if (isNull) {

printf("The list is NULL.");

getchar();

break;

}

printf("Please input the element : ");

scanf("%d", &e);

getchar();

if (LocateElem(L, e, Compare) == ERROR)

printf("There is not the element\n");

else

printf("The index is : %d", LocateElem(L, e, Compare));

getchar();

break;

case 8:

if (isNull) {

printf("The list is NULL.");

getchar();

break;

}

ElemType cue, pre;

printf("Please input the element : ");

scanf("%d", &cue);

getchar();

if (PriorElem(L, cue, &pre) == OK)

printf("The prior element is : %d\n", pre);

else

printf("Failed!\n");

getchar();

break;

case 9:

if (isNull) {

printf("The list is NULL.");

getchar();

break;

}

ElemType next;

printf("Please input the element : ");

scanf("%d", &cue);

getchar();

if (NextElem(L, cue, &next) == OK)

printf("The next element is : %d\n", next);

else

printf("Failed!\n");

getchar();

break;

case 10:

if (isNull) {

printf("The list is NULL.");

getchar();

break;

}

printf("Please input the index : ");

scanf("%d", &index);

getchar();

printf("Please input the element : ");

scanf("%d", &e);

getchar();

if (ListInsert(L, index, e) == OK)

printf("Success\n");

else

printf("Failed\n");

getchar();

break;

case 11:

if (isNull) {

printf("The list is NULL.");

getchar();

break;

}

printf("Please input the index : ");

scanf("%d", &index);

getchar();

if (ListDelete(L, index, &e) == OK)

printf("Success\n");

else

printf("Failed\n");

getchar();

break;

case 12:

if (isNull) {

printf("The list is NULL.");

getchar();

break;

}

ListTrabverse(L);

getchar();

break;

case 0:

break;

}

}

printf("Thank you for using!\n");

getchar();

return 0;

}

status IntiaList(SqList & L) {

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

if (!L.elem)

exit(OVERFLOW);

isNull = FALSE;

L.length = 0;

L.listSize = LIST\_INIT\_SIZE;

return OK;

}

status DestroyList(SqList \* L) {

L->length = 0;

isNull = TRUE;

free(L->elem);

L = NULL;

return OK;

}

status ClearList(SqList & L) {

if (isNull) {

printf("The list is NULL.");

return ERROR;

}

L.length = 0;

return OK;

}

status ListEmpty(SqList L) {

if (isNull) {

printf("The list is NULL.");

return ERROR;

}

if (L.length == 0) {

return TRUE;

}

else {

return ERROR;

}

}

int ListLength(SqList L) {

if (isNull) {

printf("The list is NULL.");

return ERROR;

}

return L.length;

}

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

if (i < 1 || i > ListLength(L))

return ERROR;

e = L.elem[i - 1];

return e;

}

int LocateElem(SqList L, ElemType e, status(\*Compare)(ElemType, ElemType)) {

int index = 1;

for (index; index <= L.length; index++) {

if ((\*Compare)(L.elem[index - 1], e)) {

return index;

}

}

return ERROR;

}

status PriorElem(SqList L, ElemType cue, ElemType \* pre) {

int i;

for (i = 0; i < L.listSize; i++) {

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

if (i == 0) {

printf("This is the head element!\n");

return ERROR;

}

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

return OK;

}

}

return FALSE;

}

status NextElem(SqList L, ElemType cue, ElemType \* next) {

int i;

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

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

if (i == L.length - 1) {

printf("This is the rear element!\n");

return ERROR;

}

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

return OK;

}

}

printf("There is not the element!\n");

return FALSE;

}

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

ElemType \* newbase, \*p, \*q;

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

return ERROR;

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

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

if (!newbase)

exit(OVERFLOW);

L.elem = newbase;

L.listSize += LISTINCREMENT;

}

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

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

\*(p + 1) = \*p;

}

\*q = e;

++L.length;

return OK;

}

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

ElemType \*p, \*q;

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

return ERROR;

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

e = p;

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

for (p++; p <= q; p++) {

\*(p - 1) = \*p;

}

--L.length;

return OK;

}

status ListTrabverse(SqList L) {

int i;

if (ListEmpty(L)) {

printf("The list is empty!\n");

return ERROR;

}

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

printf("%d ", L.elem[i]);

}

return OK;

}

status Compare(ElemType a, ElemType b) {

if (a == b) {

return TRUE;

}

else {

return FALSE;

}

}

附录B 基于链式存储结构线性表实现的源程序

#include "stdio.h"

#include "malloc.h"

#include "stdlib.h"

#include "string.h"

/\* DATA DEFINE \*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR -1

#define OVERFLOW -2

/\* TYPE DEFINE \*/

typedef int status;

typedef int ElemType;

#define LIST\_INIT\_SIZE 10

#define LISTINCREMENT 10

typedef struct Lnode {

ElemType data;

struct Lnode \* next;

}LNode, \*LinkList;

status IntiaList(LinkList \* L);

status DestroyList(LinkList \* L);

status ClearList(LinkList \* L);

status ListEmpty(LinkList L);

int ListLength(LinkList L);

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

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

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

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

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

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

status ListTrabverse(LinkList L, status (\*Visit)(ElemType e));

status Compare(ElemType a, ElemType b);

status Visit(ElemType e);

extern bool isNULL = true;

int main() {

LinkList L = NULL;

int op=1;

while(op){

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

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

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

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

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

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

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

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

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

printf(" 0. Exit\n");

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

printf(" Please chose : [0~12]:");

scanf("%d",&op);

getchar();

switch(op) {

case 1:

if (IntiaList(&L) == OK) {

printf("Success\n");

}

else

printf("failed\n");

getchar();

break;

case 2:

if (DestroyList(&L) == OK) {

printf("Success\n");

}

else

printf("Failed\n");

getchar();

break;

case 3:

if (ClearList(&L) == OK)

printf("Success\n");

else

printf("Failed\n");

getchar();

break;

case 4:

ListEmpty(L);

getchar();

break;

case 5:

if (ListLength(L) != ERROR) {

printf ("The length is : %d", ListLength(L));

}

getchar();

break;

case 6:

int index;

ElemType e;

printf("Please input the index : ");

scanf("%d", &index);

getchar();

if (GetElem(L, index, &e) != ERROR)

printf("The element is : %d\n", e);

else

printf("Failed\n");

getchar();

break;

case 7:

printf("Please input the element : ");

scanf("%d", &e);

getchar();

if (LocateElem(L, e, Compare) != ERROR)

printf("The index is : %d", LocateElem(L, e, Compare));

getchar();

break;

case 8:

ElemType cue, pre;

printf("Please input the element : ");

scanf("%d", &cue);

getchar();

if (PriorElem(L, cue, &pre) == OK)

printf("The prior element is : %d\n", pre);

getchar();

break;

case 9:

ElemType next;

printf("Please input the element : ");

scanf("%d", &cue);

getchar();

if (NextElem(L, cue, &next) == OK)

printf("The next element is : %d\n", next);

getchar();

break;

case 10:

printf("Please input the index : ");

scanf("%d", &index);

getchar();

printf("Please input the element : ");

scanf("%d", &e);

getchar();

if (ListInsert(&L, index, e) == OK)

printf("Success\n");

else

printf("Failed\n");

getchar();

break;

case 11:

printf("Please input the index : ");

scanf("%d", &index);

getchar();

if (ListDelete(&L, index, &e) == OK)

printf("Success\n");

else

printf("Failed\n");

getchar();

break;

case 12:

ListTrabverse(L, Visit);

getchar();

break;

case 0:

break;

}

}

printf("Thank you for using!\n");

getchar();

return 0;

}

status IntiaList(LinkList \* L) {

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

if (\*L == NULL)

exit(OVERFLOW);

(\*L)->data = 0;

(\*L)->next = NULL;

isNULL = false;

return OK;

}

status DestroyList(LinkList \* L) {

if (isNULL) {

printf ("The list is NULL.");

return ERROR;

}

LinkList p, q;

p = \*L;

while (p) {

q = p->next;

free(p);

p = q;

}

\*L = NULL;

isNULL = true;

return OK;

}

status ClearList(LinkList \* L) {

if (isNULL) {

printf ("The list is NULL.");

return ERROR;

}

LinkList p, q;

p = (\*L)->next;

while (p) {

q = p->next;

free(p);

p = q;

}

(\*L)->next = NULL;

return OK;

}

status ListEmpty(LinkList L) {

if (isNULL) {

printf ("The list is NULL.");

return ERROR;

}

if (L->next == NULL) {

printf ("The list is empty.");

return TRUE;

}

else {

printf ("The list isn't empty.");

return FALSE;

}

}

int ListLength(LinkList L) {

if (isNULL) {

printf ("The list is NULL.");

return ERROR;

}

int i = 0;

LinkList p = L->next;

while (p) {

i++;

p = p->next;

}

return i;

}

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

if (isNULL) {

printf ("The list is NULL.");

return ERROR;

}

int times = 1;

LinkList p = L->next;

if (i < 1 || i > ListLength(L)) {

printf ("The index is illegal.");

return ERROR;

}

while (p && times < i) {

p = p->next;

times++;

}

\*e = p->data;

return OK;

}

int LocateElem(LinkList L, ElemType e, status(\*Compare)(ElemType, ElemType)) {

if (isNULL) {

printf ("The list is NULL.");

return ERROR;

}

int index = 0;

LinkList p = L->next;

while (p) {

index++;

if (Compare(p->data, e)) {

return index;

}

p = p->next;

}

printf ("The element is not in the list.");

return ERROR;

}

status PriorElem(LinkList L, ElemType cue, ElemType \* pre) {

if (isNULL) {

printf ("The list is NULL.");

return ERROR;

}

LinkList p = L->next;

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

p = p->next;

}

if (p->next == NULL) {

printf ("The pre element doesn't exist.");

return ERROR;

}

\*pre = p->data;

return OK;

}

status NextElem(LinkList L, ElemType cue, ElemType \* next) {

if (isNULL) {

printf ("The list is NULL.");

return ERROR;

}

LinkList p = L->next;

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

p = p->next;

}

if (p->next == NULL) {

printf ("The next element doesn't exist.");

return ERROR;

}

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

return OK;

}

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

if (isNULL) {

printf ("The list is NULL.");

return ERROR;

}

int j = 1;

LinkList p, newLode;

p = \*L;

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

p = p->next;

j++;

}

if (p == NULL || j > i) {

printf ("The index is illegal.");

return ERROR;

}

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

if (newLode == NULL) {

exit(OVERFLOW);

}

newLode->data = e;

if (p->next == NULL) {

newLode->next = NULL;

}

else {

newLode->next = p->next;

}

p->next = newLode;

return OK;

}

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

if (isNULL) {

printf ("The list is NULL.");

return ERROR;

}

int j = 1;

LinkList p, q;

p = \*L;

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

p = p->next;

j++;

}

if (i > ListLength(\*L)) {

printf ("The index is illegal.");

return ERROR;

}

q = p->next;

p->next = q->next;

\*e = q->data;

free(q);

return OK;

}

status ListTrabverse(LinkList L, status(\*visit)(ElemType e)) {

if (isNULL) {

printf ("The List is NULL.");

return ERROR;

}

if (ListLength(L) == 0) {

printf ("The List is empty.");

return ERROR;

}

LinkList p = L->next;

while (p != NULL) {

if (!(\*visit)(p->data)) {

printf ("Failed");

}

p = p->next;

}

return OK;

}

status Compare(ElemType a, ElemType b) {

if (a == b) {

return TRUE;

}

else {

return FALSE;

}

}

status Visit(ElemType e) {

printf ("%d ", e);

return OK;

}