

**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201802**

**学 号： U201814542**

**姓 名：**

**指导教师：**

**报告日期： 201X年 X月 X 日**

**计算机科学与技术学院**

目 录

[1 基于顺序存储结构的线性表实现 2](#_Toc458159879)

[1.1 问题描述 2](#_Toc458159880)

[1.2 系统设计 2](#_Toc458159882)

[1.3 系统实现 2](#_Toc458159883)

[1.4 实验小结 2](#_Toc458159884)

[2 基于链式存储结构的线性表实现 2](#_Toc458159885)

[2.1 问题描述 2](#_Toc458159886)

[2.2 系统设计 2](#_Toc458159887)

[2.3 系统实现 2](#_Toc458159888)

[2.4 实验小结 2](#_Toc458159889)

[3 基于二叉链表的二叉树实现 2](#_Toc458159890)

[3.1 问题描述 2](#_Toc458159891)

[3.2 系统设计 2](#_Toc458159892)

[3.3 系统实现 2](#_Toc458159893)

[3.4 实验小结 2](#_Toc458159894)

[4 基于二叉链表的二叉树实现 2](#_Toc458159895)

[4.1 问题描述 2](#_Toc458159896)

[4.2 系统设计 2](#_Toc458159897)

[4.3 系统实现 2](#_Toc458159898)

[4.4 实验小结 2](#_Toc458159899)

[参考文献 2](#_Toc458159900)

[附录A 基于顺序存储结构线性表实现的源程序 2](#_Toc458159901)

[附录B 基于链式存储结构线性表实现的源程序 2](#_Toc458159902)

[附录C 基于二叉链表二叉树实现的源程序 2](#_Toc458159903)

[附录D 基于邻接表图实现的源程序 2](#_Toc458159904)

1 基于顺序存储结构的线性表实现

1.1 问题描述

创建一套顺序表的管理系统，并要求通过该系统实现初始化表、销毁表、清空表、判定空表、求表长、获得元素、查找元素、获得前驱、获得后继、插入元素、删除元素、遍历表、以文件形式保存线性表、多个线性表管理等功能。

1.1.1 具体要求

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InitList(L)；初始条件是线性表L不存在；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ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。

⒀其他操作：线性表的文件形式保存：①需要设计文件数据记录格式，以高效保存线性表数据逻辑结构(D,{R})的完整信息；②需要设计线性表文件保存和加载操作合理模式。多个线性表的管理。

## 1.2 系统设计

该系统包含了用户交互模块、操作管理模块、功能实现模块、文件I/O模块、多线性表管理模块。

用户交互模块提示用户对程序进行交互并提示用户可以进行的操作。

操作管理模块根据用户的输入和底层的功能实现模块进行交互，并且根据用户已有的操作判断用户当前的操作是否合法，并与用户交互模块交互，

功能实现模块根据操作管理模块传递的信息来实现相应的功能。

文件I/O模块实现了系统的文件I/O功能，与本地文件进行交互，实现文件读取/储存操作。

多线性表管理模块与操作管理模块进行交互，实现了多线性表管理的功能。



图1-1 顺序表管理系统模块结构示意图

## 1.3 系统实现

开发环境为windows系统，使用了Clion IDE进行编程，编译器为MinGW。

在用户交互模块实现了显示菜单的功能。该功能通过使用printf()向屏幕输出字符实现。使用system(“cls”)清理输出使交互界面保持整洁，通过循环维持整体程序的运行并维护交互界面。

操作管理模块维护了变量op保存用户选择的操作序号，来实现用户交互模块和操作管理模块之间的交互。维护了变量L来指向用户当前选择的顺序表，实现了与功能实现模块的交互。维护全局变量is\_created来判断用户是否使用了多顺序表管理功能，若没有，则通过用户交互模块提示用户先进行多顺序表管理功能的初始化。

在功能实现模块实现了初始化表、删除表、清空表、判定空表、求表长、获得元素、查找元素、获得前驱、获得后继、插入元素、遍历表、删除元素十二个功能。这十二个功能由十二个函数单独实现。

InitList(SqList &L)函数返回操作完成的状态，实现初始化表功能，判断L指向的表是否已初始化，若已初始化则返回ERROR状态与操作管理模块进行交互提示用户已初始化表，否则为表分配内存空间。

DestroyList(SqList &L)函数返回操作完成的状态，实现销毁表功能。判断表是否已被初始化，若未被初始化则返回ERROR状态，否则释放表的内存空间并返回OK。

ClearList(SqList &L)函数返回操作完成的状态，实现清理表功能。判断表是否已被初始化，未被初始化返回ERROR状态，否则设置表长为0并返回OK。

ListEmpty(SqList &L)函数返回操作完成的状态，实现判断表是否为空的功能。判断表是否已被初始化，未被初始化返回ERROR状态，否则判断表长是否为0，若为0则返回TRUE，否则返回FALSE。

ListLength(SqList &L)函数返回表长，实现查询表长的功能。判断表是否已被初始化，未初始化返回-1，否则返回表的长度。

GetElem(SqList &L,int i,ElemType &e)返回操作完成的状态，实现获得指定位置元素的功能，i为查询的位置，e为存放查询结果的变量。判断表是否已被初始化，未被初始化返回NONE状态，判断i是否为合法位置，不合法返回ERROR。

LocateElem(SqList &L,ElemType e)返回查询元素的位置，e为查询的元素，判断表是否已被初始化，未被初始化返回NONE状态。否则返回元素位置，若不存在该元素，返回0。

PriorElem(SqList &L,ElemType cur,ElemType &pre\_e)返回操作完成的状态，实现查询元素前驱功能，cur为查询的元素，pre\_e为前驱存放位置，判断表是否已被初始化，未被初始化返回NONE状态。否则返回OK。

NextElem(SqList &L,ElemType cur,ElemType &next\_e)返回操作完成的状态，实现查询元素后继功能，cur为查询的元素，next\_e为后继存放位置，判断表是否已被初始化，未被初始化返回NONE状态。否则返回OK。

ListTrabverse(SqList &L)返回操作完成的状态，实现遍历表的功能。判断表是否已被初始化，未被初始化返回ERROR状态。

ListInsert(SqList &L,int i,ElemType e)返回操作完成的状态，实现插入元素的功能，i为元素插入的位置，e为要插入的元素。ListInsert的算法流程如图2所示。



图1-2 ListInsert算法流程示意图

ListDelete(SqList &L,int i,ElemType & e) 返回操作完成的状态，实现插入元素的功能，i为元素删除的位置，e保存要删除的元素。ListDelete的算法流程如图3所示。



图1-3 ListDelete算法流程示意图

在文件I/O模块实现了与文件进行交互的功能。主要由ReadFromFile(SqList &L)和SaveToFile(SqList &L)两个函数实现。

SaveToFile函数返回函数运行状态，把文件存储为D:\classtask\save.txt，使用fprintf在文件内进行输出，输出格式第一行为表长，第二行为表容量，第三行为表内容。

ReadFromFile函数返回函数运行状态，从D:\classtask\save.txt读取。使用fgets分别读取三行内容并解析存入表中。其中第一二行使用atoi()函数进行字符串到整数的转换，第三行读取表内容时使用了FindNumAndFillArray函数进行读取。FindNumAndFillArray的算法流程如图1-4所示。



图1-4 FindNumAndFillArray的算法流程

多线性表管理系统采取了分配一个线性表指针数组的方式实现，通过用户交互模块获得要分配的内存大小，并获得L当前所指向的内存表序号。

**1.4 系统测试**

系统测试中将根据使用的实际流程对每个功能进行逐一测试。

初始化流程测试：

输入：（1）4 （2）-2 （3）重复初始化当前线性表

预期结果：（1）初始化成功 （2）报告错误 （3）报告错误

实际结果：（1）如图1-5所示 （2）如图1-6所示 （3）如图1-7所示

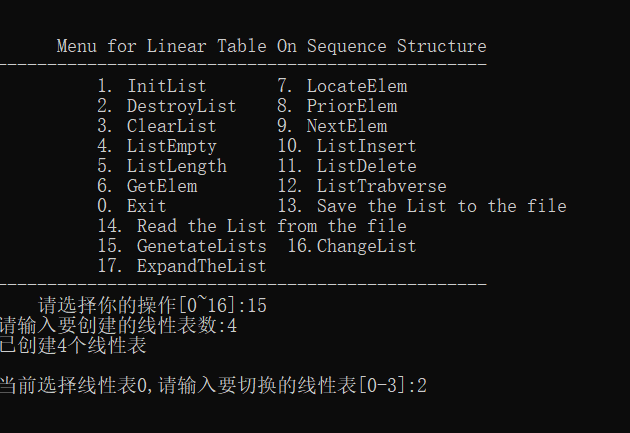


图1-5 初始化测试（1）结果

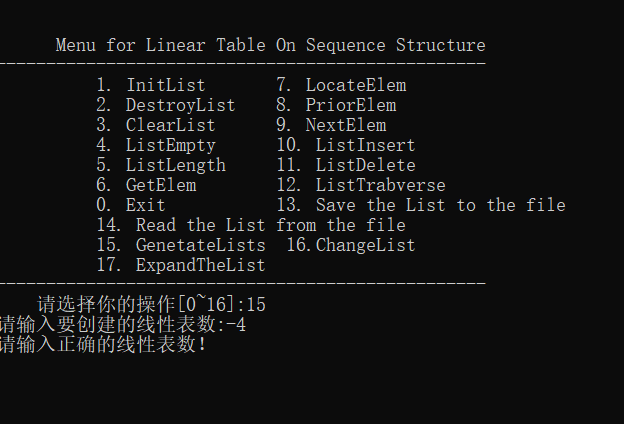


图1-6 初始化测试（2）结果

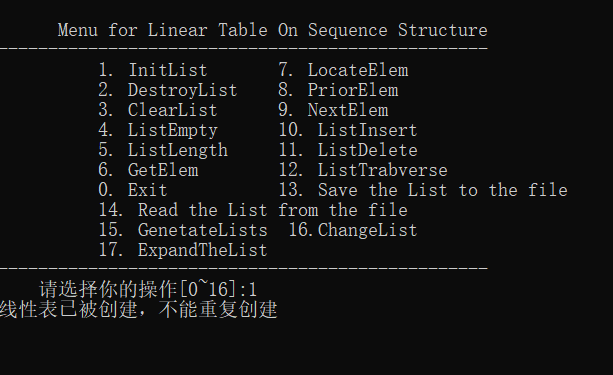


图1-7 初始化测试（3）的结果

插入、删除、遍历功能测试：

输入：（1）按顺序插入5 3 1 4 2 删除3 4并遍历 （2）插入1后向位置3插入4 （3）插入1 2 3 4后删除位置5的元素

预期结果：（1）显示5 1 2 （2）报告错误 （3）报告错误

实际结果：（1）如图1-7 （2）如图1-8 （3）如图1-9

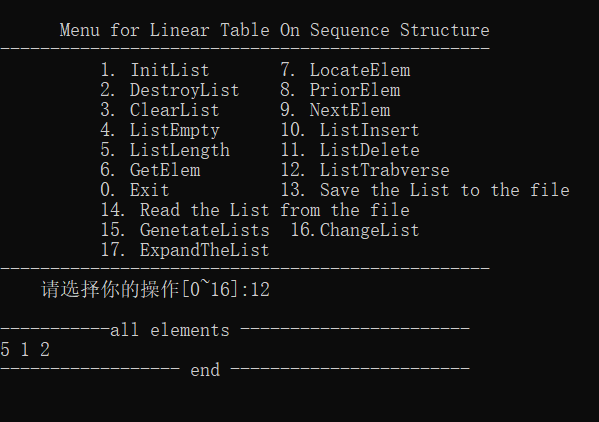


图1-7 插入、删除、遍历功能测试（1）结果

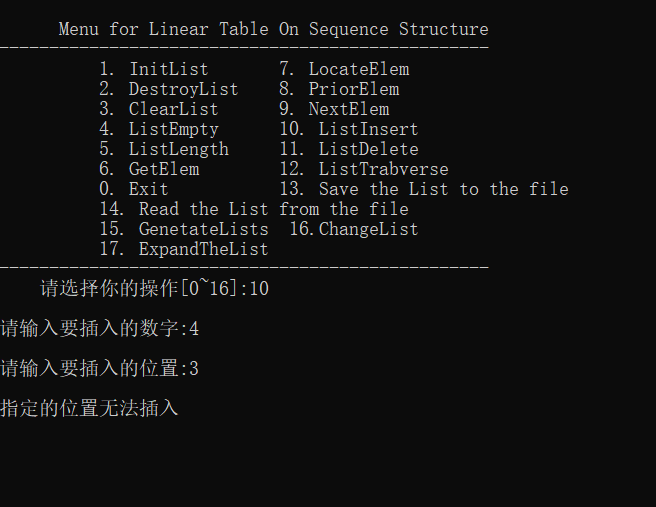


图1-8 插入、删除、遍历功能测试（2）结果

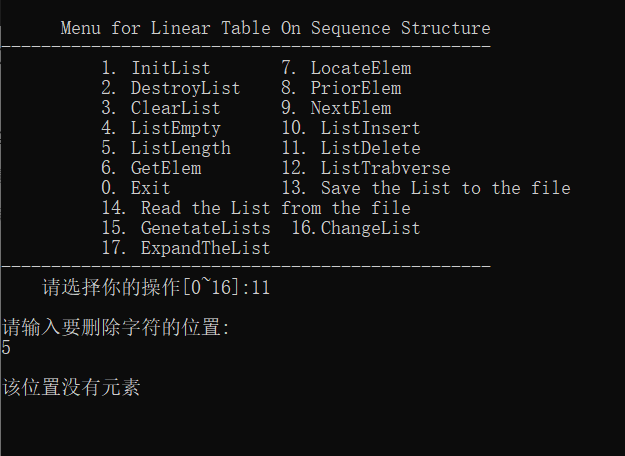


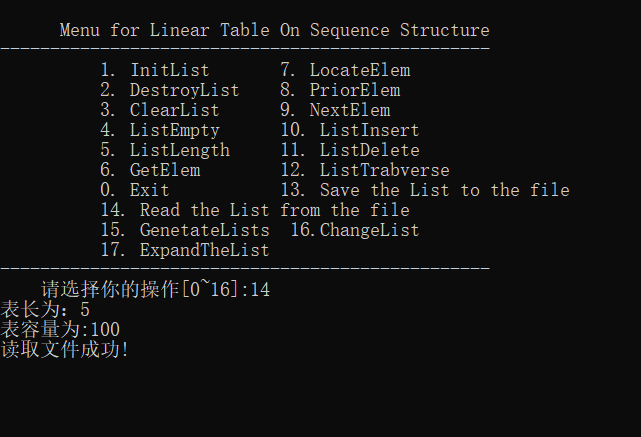
图1-9 插入、删除、遍历功能测试（3）结果

文件I/O功能测试：

输入：（1）将线性表1：[1,2,3,4,5]存入文件，再用线性表2去读取并遍历（2）删除文件后读取文件

预期输出：（1）遍历结果为1 2 3 4 5 （2）报告错误

实际输出：（1）如图1-10 （2）如图1-11



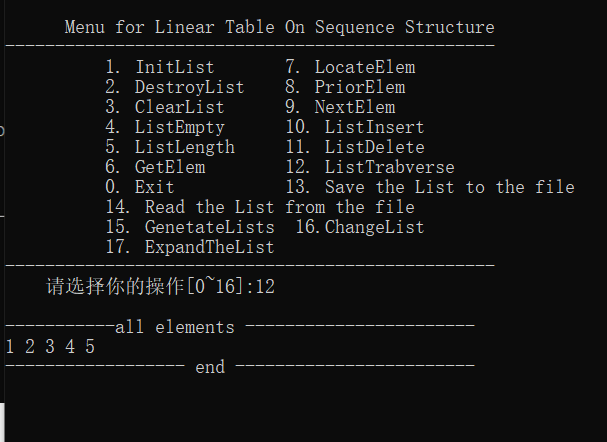


图1-10 文件I/O测试结果（1）

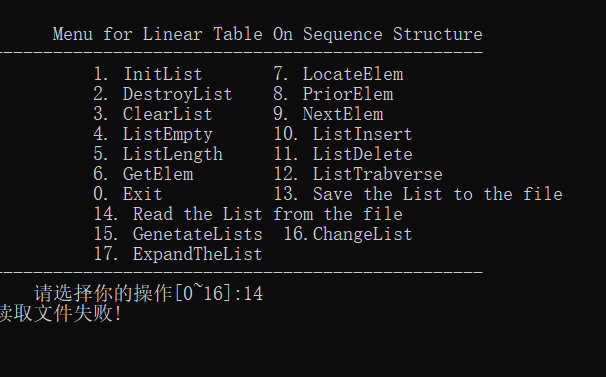


图1-11 文件I/O测试结果（2）

功能4、5、6、7、8、9测试：

输入：当前表为[1,2,3,4,5]（1）测试功能4、5 （2）功能6输入3、7 （3）功能7输入3、7 （4）功能8输入3、7 （5）功能9输入3、7

预期结果：（1）报告情况、表长 （2）得到3、报错 （3）得到3、报错 （4）得到2、报错 （5）得到4、报错

实际结果：（1）如图1-12 （2）如图1-13 （3）如图1-14 （4）如图1-15

（5）如图1-16



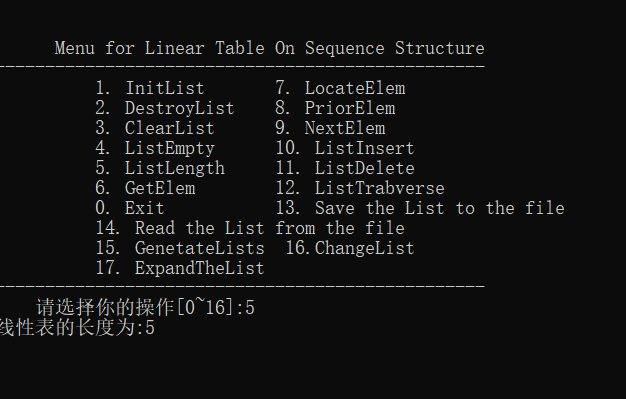
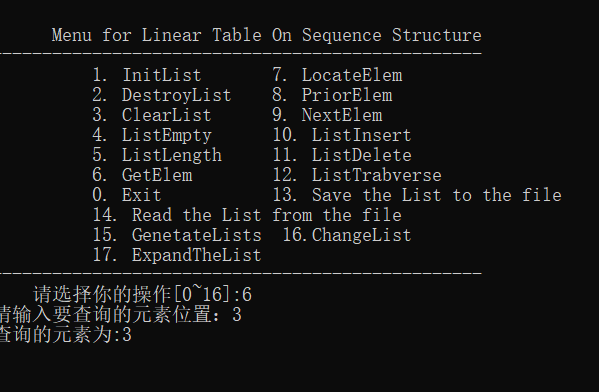
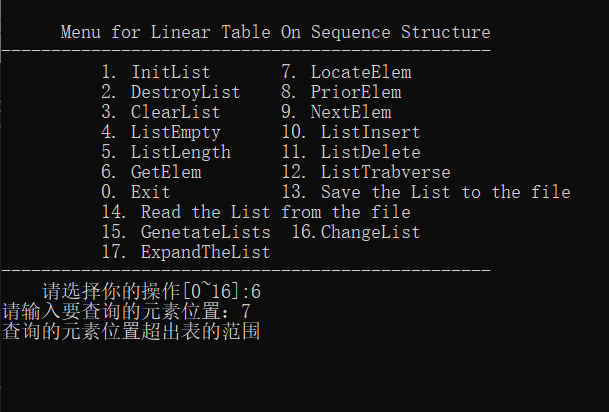
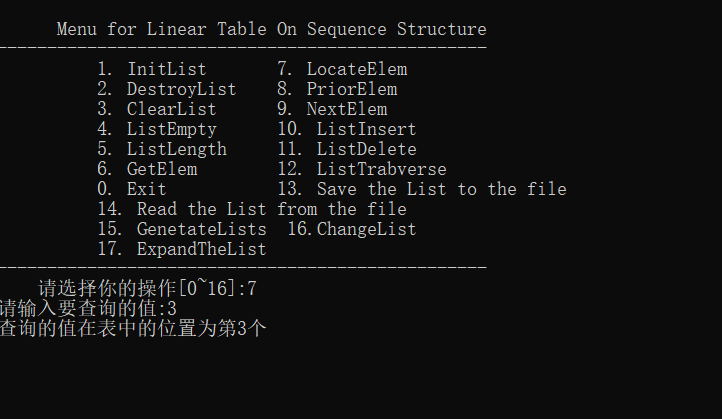


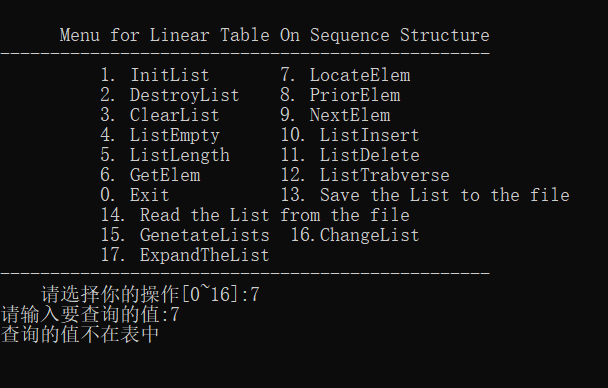
图1-12 功能4、5测试结果



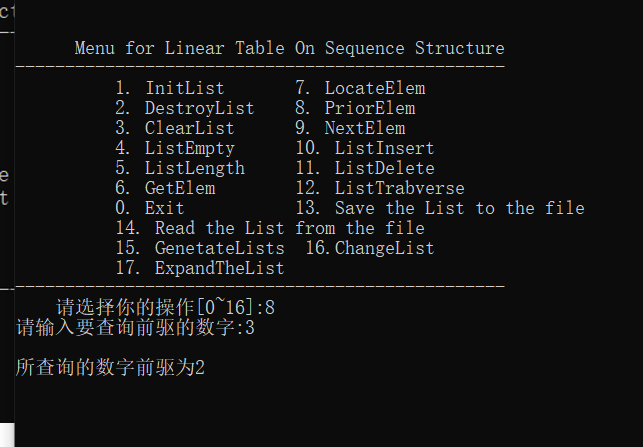


1-13功能6测试结果





1-14 功能7测试结果



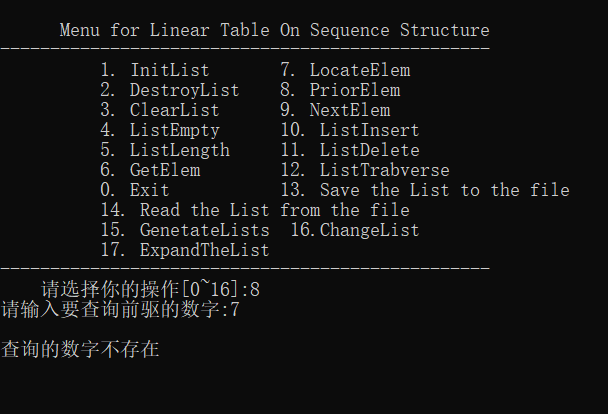
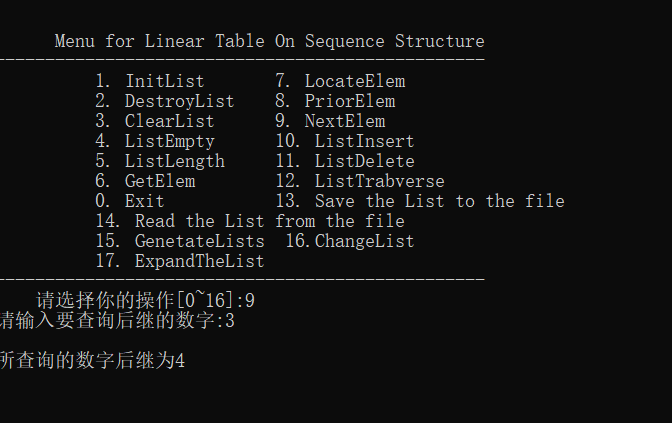


图1-15功能8测试结果



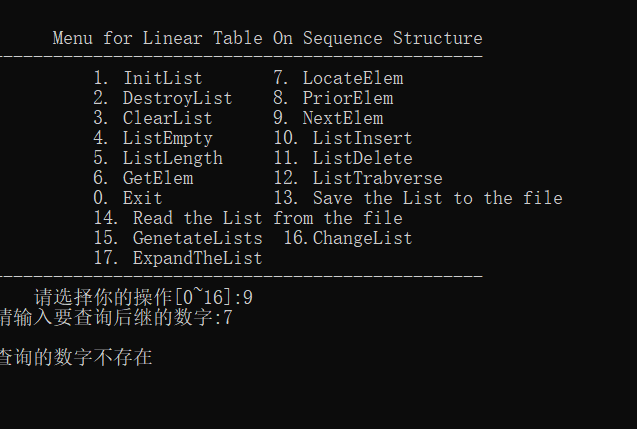


图1-16 功能9测试结果

## 1.5 实验小结

通过本次实验，我加深对线性表的概念、基本运算的理解，熟练掌握线性表的逻辑结构与物理结构的关系，并物理结构采用顺序表,熟练掌握线性表的基本运算的实现。在本次实验中，我总结了以下经验教训：

1.各个功能间的函数都不是独立的，可以适当调用其它功能函数以减小工作量并方便维护。

2.在声明指针后，初始化之前一定要做“预处理”，即把未初始化的指针指向的结构成员先设置为NULL或者0，否则这些结构成员由于所处的内存块中原先的数据并没有被清空，会导致程序在判断处理时发生异常。

3.在文件I/O的处理过程中，考虑格式时不仅要考虑存储文件时的便利性，更要考虑读取文件时的便利性。

4.在设置状态变量时最好不要与可能的返回值冲突，可以使用C++中的enum来规避对status赋整数值的情况。

5.对于边界情况的判断一定要清楚，否则会导致错误。

6.对错误一定要有信息反馈，以增加了使用友好度，也更方便进行调试。

2 基于链式存储结构的线性表实现

## 2.1 问题描述

## 2.2 系统设计

## 2.3 系统实现

## 2.4 系统测试

## 2.5 实验小结

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

## 3.2 系统设计

## 3.3 系统实现

## 3.4 系统测试

## 3.5 实验小结

# 4 基于邻接表的图实现

## 4.1 问题描述

## 4.2 系统设计

## 4.3 系统实现

## 4.4 系统测试

## 4.5 实验小结

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

[2] [Larry Nyhoff](http://www.calvin.edu/~nyhl/index.html). [ADTs, Data Structures, and Problem Solving with C++.](http://vig.prenhall.com/catalog/academic/product/0,1144,0131409093,00.html)Second Edition, [Calvin College](http://cs.calvin.edu/), 2005

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <iostream>

#include <string.h>

#include <ctype.h>

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

#define TRUE 1

#define FALSE 0

#define OK 2

#define ERROR 3

#define INFEASTABLE -1

#define OVERFLOW -2

#define NONE -4

typedef int status;

typedef int ElemType; //数据元素类型定义

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

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

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

status InitList(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);

int LocateElem(SqList &L,ElemType e); //简化过

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); //简化过

status ReadFromFile(SqList &L);

status SaveToFile(SqList &L);

size\_t FindNumAndFillArray(const char \*str, int \*container,size\_t max\_size);

int iscreated = 0;

/\*

\* iscreated全局变量判断用户是否已经生成了线性表 表，以提示用户先执行操作15

\*/

int main(){

/\*

\* L是用来指向当前选择的线性表的指针

\*/

SqList \*L; int op=1; int choice=0; int ListNum; SqList \*manager;int num;

while(op){

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

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

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

printf(" 1. InitList 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 13. Save the List to the file\n");

printf(" 14. Read the List from the file\n");

printf(" 15. GenetateLists 16.ChangeList\n");

printf(" 17. ExpandTheList \n");

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

printf(" 请选择你的操作[0~16]:");

scanf("%d",&op);

/\*

\* 在每个操作被执行之前，都要判断多线性表管理指针是否已被创建，若未被创建，将提示

\* 用户执行操作15，若已创建则执行操作1。

\*/

switch(op){

case 1:

status ILstatus;

if((ILstatus = InitList(\*L))==OK) printf("线性表创建成功！\n");

else if(ILstatus == FALSE)printf("必须先执行操作15,创建线性表集！\n");

else printf("线性表已被创建，不能重复创建\n");

getchar();getchar();

break;

case 2:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

if(DestroyList(\*L) == ERROR) printf("\n尝试删除未创建的表！\n");

else printf("\n线性表删除成功！\n");

getchar();getchar();

break;

case 3:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

if(ClearList(\*L) == ERROR) printf("尝试清空未创建的表！\n");

else printf("线性表清空成功\n");

getchar();getchar();

break;

case 4:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

if(ListEmpty(\*L) == ERROR) printf("线性表不存在\n");

else if(ListEmpty(\*L) == TRUE) printf("所查询的表为空表\n");

else if(ListEmpty(\*L) == FALSE)printf("所查询的表非空！\n");

getchar();getchar();

break;

case 5:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

if(ListLength(\*L) == -1) printf("线性表未初始化\n");

printf("线性表的长度为:%d\n",ListLength(\*L));

getchar();getchar();

break;

case 6:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

printf("请输入要查询的元素位置：");

int i; //i是表中第i个元素的位置

std::cin >> i;

int number\_get;

if(GetElem(\*L,i,number\_get) == ERROR){printf("查询的元素位置超出表的范围\n");}

else if(GetElem(\*L,i,number\_get) == NONE){printf("线性表未初始化\n");}

else{printf("查询的元素为:%d\n",number\_get);}

getchar();getchar();

break;

case 7:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

printf("请输入要查询的值:");

int val\_to\_inquire;

std::cin >> val\_to\_inquire;

int position;

position = LocateElem(\*L,val\_to\_inquire);

if(position == 0) printf("查询的值不在表中\n");

else if(position == NONE) printf(("线性表未初始化\n"));

else printf("查询的值在表中的位置为第%d个\n",position);

getchar();getchar();

break;

case 8:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

status PEstatus;

ElemType prior,num\_to\_inquire\_PE;

printf("请输入要查询前驱的数字:");

std::cin >> num\_to\_inquire\_PE;

PEstatus = PriorElem(\*L,num\_to\_inquire\_PE,prior);

if(PEstatus == FALSE){printf("\n查询的数字不存在\n");}

else if(PEstatus == ERROR){printf("所查询的数字不存在前驱\n");}

else if(PEstatus == NONE){printf(("线性表未初始化\n"));}

else {printf("\n所查询的数字前驱为%d\n",prior);}

getchar();getchar();

break;

case 9:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

status NEstatus;

ElemType next,num\_to\_inquire\_NE;

printf("请输入要查询后继的数字:");

std::cin >> num\_to\_inquire\_NE;

NEstatus = NextElem(\*L,num\_to\_inquire\_NE,next);

if(NEstatus == FALSE){printf("\n查询的数字不存在\n");}

else if(NEstatus == ERROR){printf("\n所查询的数字不存在后继\n");}

else if(NEstatus == NONE){printf(("线性表未初始化\n"));}

else {printf("\n所查询的数字后继为%d\n",next);}

getchar();getchar();

getchar();getchar();

break;

case 10:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

ElemType num\_to\_insert;

int locate\_to\_insert;

printf("\n请输入要插入的数字:");

std::cin >> num\_to\_insert;

printf("\n请输入要插入的位置:");

std::cin >> locate\_to\_insert;

status I\_status;

I\_status = ListInsert(\*L,locate\_to\_insert,num\_to\_insert);

if(I\_status == ERROR){ printf("\n表容量已达上限！\n");}

else if(I\_status == NONE){ printf("线性表未初始化\n");}

else if(I\_status == FALSE){printf("\n指定的位置无法插入\n");}

getchar();getchar();

break;

case 11:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

ElemType recver;

status DLstatus;

int delete\_locate;

printf("\n请输入要删除字符的位置:\n");

std::cin >> delete\_locate;

DLstatus = ListDelete(\*L,delete\_locate,recver);

if(DLstatus == ERROR){

printf("\n该位置没有元素\n");

} else if(DLstatus == NONE) { printf("线性表未初始化\n");}

else

{printf("\n删除的元素为:%d\n",recver);}

getchar();getchar();

break;

case 12:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

status LTstatus;

if((LTstatus = ListTrabverse(\*L))==0) printf("线性表是空表！\n");

else if(LTstatus == ERROR) printf("线性表未初始化\n");

getchar();getchar();

break;

case 13:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

/\*

\* 存储和读取的文件路径都为：D:\classtask\save.txt

\*/

status SAVEstatus;

SAVEstatus = SaveToFile(\*L);

if(SAVEstatus == ERROR){

printf("创建文件失败！\n");

}

else {

printf("保存文件成功！\n");

}

getchar(); getchar();

break;

case 14:

/\*

\* 存储和读取的文件路径都为：D:\classtask\save.txt

\* 读取之前要求先初始化指针

\*/

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

status READstatus;

READstatus = ReadFromFile(\*L);

if(READstatus == ERROR) {

printf("读取文件失败!\n");

}

else printf("读取文件成功!\n");

getchar();getchar();

break;

case 15:

if(iscreated == 1){

printf("线性表已创建");

getchar();getchar();

break;

}

printf("请输入要创建的线性表数:");

std::cin >> num;

manager = (SqList \*)malloc(sizeof(SqList)\*num);

printf("已创建%d个线性表\n",num);

iscreated = 1;

/\*

\* 此处每一个线性表都要先初始化为全空，否则未初始化的内存会影响结果。

\*/

for(int t=0;t<num;t++){

manager[t].elem = NULL;

manager[t].listsize = 0;

manager[t].length = 0;

}

getchar();getchar();

case 16:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

printf("当前选择线性表%d,请输入要切换的线性表[0-%d]:",choice,num-1);

std::cin >> choice;

L = &(manager[choice]);

getchar();

case 17:

if (iscreated == 0){

printf("必须先执行操作15,创建线性表集！\n");

getchar();getchar();

break;

}

L->elem = (ElemType \*)realloc(L->elem,LISTINCREMENT\* sizeof(int));

L->listsize += 10;

printf("扩容成功\n");

getchar();

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

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

/\*

\* Fuction：执行初始化操作

\* Input ：需要进行初始化的对象L

\* Output ：操作执行的状态

\*/

status InitList(SqList& L){

if (iscreated == 0) return FALSE;

if(L.elem != NULL)

return ERROR;

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

if(!L.elem) exit(OVERFLOW);

L.length=0;

L.listsize=LIST\_INIT\_SIZE;

return OK;

}

/\*

\* Fuction：执行遍历操作

\* Input ：需要进行遍历的对象L

\* Output ：操作执行的状态

\*/

status ListTrabverse(SqList &L){

if (L.elem == NULL){ return ERROR;}

if(L.length == 0)

return 0;

int i;

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

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

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

return OK;

}

/\*

\* Fuction：执行释放内存操作

\* Input ：需要释放内存的对象L

\* Output ：操作执行的状态

\*/

status DestroyList(SqList &L){

if(L.elem == NULL){

return ERROR;

}

free(L.elem);

L.elem = NULL;

return OK;

}

/\*

\* Fuction：执行清空操作

\* Input ：需要清空的对象L

\* Output ：操作执行的状态

\*/

status ClearList(SqList &L){

if(L.elem == NULL){

return ERROR;

}

L.length = 0;

return OK;

}

/\*

\* Fuction：执行清空操作

\* Input ：需要清空的对象L

\* Output ：操作执行的状态

\*/

status ListEmpty(SqList &L){

if(L.elem == NULL){

return ERROR;

}

if(L.length == 0){

return TRUE;

} else { return FALSE;}

}

/\*

\* Fuction：查询表长

\* Input ：需要查询的对象L

\* Output ：表长

\*/

int ListLength(SqList &L){

if(L.elem == NULL){

return -1;

}

return L.length;

}

/\*

\* Fuction：查询元素

\* Input ：需要查询的对象L

\* Output ：操作执行的状态

\*/

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

if (L.elem == NULL)

return NONE;

int position = i - 1;

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

return ERROR;

}

return OK;

}

/\*

\* Fuction：查找元素位置

\* Input ：需要查询的对象L

\* Output ：元素的位置

\*/

int LocateElem(SqList &L,ElemType e){

if (L.elem == NULL) return NONE;

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

if(L.elem[i] == e) return i+1;

}

return 0;

}

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

int counter;

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

if(counter == 0 && L.elem[counter] == cur) {

return ERROR;

}

if(L.elem[counter] == cur){

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

return TRUE;

}

}

return FALSE;

}

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

if (L.elem == NULL)

return NONE;

int counter;

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

if(counter == L.length-1 && L.elem[counter] == cur) {

return ERROR;

}

if(L.elem[counter] == cur){

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

return TRUE;

}

}

return FALSE;

}

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

if (L.elem == NULL)

return NONE;

if(L.length+1 > L.listsize){

return ERROR;

}

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

return FALSE;

}

/\*

\* 把从locate起的所有元素往后挪一个，再把要插入的数插入locate

\*/

int locate = i - 1;

for(int l = L.length;l > locate;l--){

L.elem[l] = L.elem[l-1];

}

L.elem[locate] = e;

L.length++;

return OK;

}

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

if (L.elem == NULL)

return NONE;

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

return ERROR;

}

int locate = i - 1;

e = L.elem[locate];

/\*

\* 把从locate起的所有数往前挪一个

\* 再把L.length减1

\*/

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

L.elem[i] = L.elem[i+1];

}

L.length--;

return OK;

}

status SaveToFile(SqList &L){

char file\_location[] = "D:\\classtask\\save.txt";

FILE \*fpWrite;

/\*

\* 存储文件的格式为：

\* 行数

\* 1 顺序表的长度

\* 2 顺序表的容量

\* 3 数组，数组储存格式为:ELEM1 ELEM2......

\*/

if((fpWrite = fopen(file\_location,"w")) == NULL){ return ERROR;}

fprintf(fpWrite,"%d\n",L.length);

fprintf(fpWrite,"%d\n",L.listsize);

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

fclose(fpWrite);

return OK;

}

status ReadFromFile(SqList &L){

char file\_location[] = "D:\\classtask\\save.txt";

//保存的表信息存在D:\classtask\save.txt下

FILE \*fpRead;

char stringLine[2048];

if((fpRead = fopen(file\_location,"r")) == NULL){ return ERROR;}

/\*

\* 存储文件的格式为：

\* 行数

\* 1 顺序表的长度

\* 2 顺序表的容量

\* 3 数组，数组储存格式为:ELEM1 ELEM2......

\*/

fgets(stringLine,2048,fpRead);

L.length = atoi(stringLine);

printf("表长为：%d\n",L.length);

fgets(stringLine,2048,fpRead);

L.listsize = atoi(stringLine);

printf("表容量为:%d\n",L.listsize);

fgets(stringLine,2048,fpRead);

int \* arr = (int \*)malloc(L.listsize \*sizeof(ElemType));

/\*

\* 使用fget把存有数组的一行读入缓存区中，需要使用函数FindNumAndFillArray从字符串中读取并把数字

\* 存入数组中

\*/

FindNumAndFillArray(stringLine,arr,sizeof(arr));

L.elem = arr;

fclose(fpRead);

return OK;

}

size\_t FindNumAndFillArray(const char \*str, int \*container,size\_t max\_size){

size\_t str\_l = strlen(str), i, cur;

int buf = 0;

char flag = 0;

/\*

\* flag用以判断当前是否还在读一个连续的数

\*/

for(i = 0, cur = 0; i < str\_l && cur < max\_size; i++) {

if(isdigit(str[i])) {

/\*

\* 模仿atoi()的写法读取一个完整的十进制数字

\*/

buf = buf \* 10 + str[i] - '0';

flag = 1;

} else if(flag) {

container[cur++] = buf;

buf = 0;

flag = 0;

}

}

if(flag && cur < max\_size)

container[cur++] = buf;

/\*

\* 返回key值上界

\*/

return cur;

}

附录B 基于链式存储结构线性表实现的源程序

# 附录C 基于二叉链表二叉树实现的源程序

# 附录D 基于邻接表图实现的源程序