

**课 程 实 验 报 告**

**课程名称： 数据结构试验**

**专业班级： 计科1401**

**学 号： U201414579**

**姓 名： 王德品**

**指导教师： 周时阳**

**报告日期： 2015年11月15日**

**计算机科学与技术学院**

目录

[1 线性表实验 3](#_Toc435384933)

[1.1 实验目的 3](#_Toc435384934)

[1.2 线性表基本操作定义 3](#_Toc435384935)

[1.3 实验任务 4](#_Toc435384936)

[1.4数据结构的描述 5](#_Toc435384937)

[1.5算法设计思想 6](#_Toc435384938)

[1.6算法分析 6](#_Toc435384939)

[1.7算法实现 7](#_Toc435384940)

[1.8算法测试 19](#_Toc435384941)

[1.9实验小结 21](#_Toc435384942)

# 1 线性表实验

## 1.1 实验目的

在这一次的实验中，根据线性表的顺序结构，我们实现关于线性表的12个基本操作，并且用文本菜单做成一个简单的系统。要求在这一个简易系统中，能对线性表进行文件的读取和存入，能进行多表操作。

## 1.2 线性表基本操作定义

对于线性表，我们希望在处理数据能够提供一些函数，使之能够在其他程序中简化程序的操作，所以我们需要对于这一种数据结构的数据实现一些基本的功能。功能如下：

（1）InitaList(&L)

操作结果：构造一个空的线性表。

（2）DestroyList(&L)

初始条件：线性表L已存在。

操作结果：销毁线性表L。

（3）ClearList (&L)

初始条件：线性表L已存在。

操作结果：将L重置为空表。

（4）ListEmpty(L)

初始条件：线性表L已存在。

操作结果：若L为空表，则返回TRUE,否则返回FALSE。

（5）ListLength(L)

初始条件：线性表已存在。

操作结果：返回L中数据元素的个数。

（6）GetElem(L, i,&e)

初始条件：线性表已存在，1≤i≤ListLength(L)。

操作结果：用e返回L中第i个数据元素的值。

（7）LocateElem(L, e，compare（）)

初始条件：线性表已存在。

操作结果：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。

（8）PriorElem（L，cur\_e，&pre\_e）

初始条件：线性表L已存在。

操作结果：若cur\_e是L的数据元素，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。

（9）NextElem（L，cur\_e，&next\_e）

初始条件：线性表L已存在。

操作结果：若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它

的后继，否则操作失败，next\_e无定义。

（10）ListInsert(&L,i,e)

初始条件：线性表L已存在且非空，1≤i≤ListLength(L)+1。

操作结果：在L的第i个位置之前插入新的数据元素e，L的长度加1

（11）ListDelete(&L,i,&e)

初始条件：线性表L已存在且非空，1≤i≤ListLength(L)。

操作结果：删除L的第i个数据元素，用e返回其值，L的长度减1.

（12）ListTraverse(L，visit（）)

初始条件：线性表L已存在。

操作结果：依次对L的每个数据元素调用函数visit()。一旦调用失败，则操

作失败。

## 1.3 实验任务

采用顺序结构的物理结构实现线性表的基本操作。其中ElemType为数据元素的类型名，具体含义可自行定义，在此文件中采用int类型，其它有关类型和常量的定义和引用详见文献[1]的p10。构造成具有功能菜单的系统，线性表采用文件形式保存。

实验内容如下：

1. 以顺序表作为物理结构，实现线性表的基本操作。

2. 以带表头结点的单链表作为物理结构，实现线性表的基本操作。

3. (替换选择题，如选择3，则不需要完成1和2)多个数据元素类型相同的线性表的管理。其中每一个线性表采用以带表头结点的单链表作为物理结构，每一个线性表的属性至少包含有表名称。采用顺序表的方式管理多个线性表。其物理结构如图1-1所示。



图1-1 多线性表的物理结构示意图

对于本实验，则不需要采用多表的结构，只需要在单表的操作基础上，对多文件读取--操作--存储。然后对下一个表进行自己需要的操作。在对每一个线性表进行操作之前，都需要对其初始化才能够操作。

## 1.4数据结构的描述

线性表是最常用且最简单的一种数据结构。简言之，一个线性表是N个数据元素的有限序列。至于每一个数据元素的具体含义，在不同的情况下各不相同，它可以是一个数、或一个符号，也可以是一页书，甚至是其他更复杂的信息。线性表是一个相当灵活的数据结构，他的长度可根据需要增长或缩短即对线性表的数据元素不仅可以进行访问，还可以进行数据插入和删除等。

线性表的表现和实现形式可以有两种不同的方式，即顺序结构和链表结构。本实验采用的是顺序结构。

构造一个结构如下：

typedef struct {

ElemType \* elem;

int length;

int listsize;

}SqList;

length表示线性表的当前长度，listsize表示长度为listsize \* sizeof(ElemType)的内存空间，用于存放数据元素。

## 1.5算法设计思想

在这一次的实验中，我的主菜单的逻辑结构如下:

首先，菜单开始。我们首先要选择是否选用文件操作，如果使用文件的话，在线性表初始化的时候，就会提供给我们5个存储数据的文件选择操作；不用文件的话，会初始化生成一个空的线性表提供操作。由于每一个操作必须建立在初始化的基础上，所以在这一次选择中只能够操作一个文件。在结束操作之后，可以选择是否保存数据:对于文件操作的线性表，直接保存到原来的文件，对于没有文件初始化的，可以选择一个文件用来保存新构造的线性表。在结束一个表的操作后，可以选择是否继续使用此系统，继续的话，开始选择下一个文件操作，不继续的话，将会退出系统。

系统的流程图（图1-2）如下：

使用文件？

Flag = 1

Flag = 0

选择操作

初始化线性表

EXIT

操作其他的表

文件保存

Flag?

InitLIst

InitList\_file

其他操作

图1-2

## 1.6算法分析

（1）InitaList(&L)

对于不用读入文件的Initalize(&), 直接在内存空间中分配大小为listsize \* sizeof(ElemType)的空间给elem, L.length 初始化为0；对于需要读入文件的，先分配空间，在用fead读入数据并且计算出L.length， 在选择文件方面，我将不同的文件名放在了一个叫filename的外部数组之中，便于其他函数使用；其他函数的操作都要建立在初始化的基础上。

（2）DestroyList(&L)

直接free掉L.elem的动态存储空间。

（3）ClearList (&L)

将L.length直接置为0即可。

（4）ListEmpty(L)

直接判断L.length是否为0即可。

（5）ListLength(L)

直接返回L.length。

（6）GetElem(L, i,&e)

找到L.elem[i], 用e返回其值。

（7）LocateElem(L, e，compare（）)

遍历线性表，当e == L.elem[i]时停止，输出i的值。

（8）PriorElem（L，cur\_e，&pre\_e）

遍历线性表，当cur\_e等于L.elem[i]时停止，用pre\_e返回L.elem[i-1]的值。

（9）NextElem（L，cur\_e，&next\_e）

遍历线性表，当cur\_e等于L.elem[i]时停止，用next\_e返回L.elem[i-1]的值。（10）ListInsert(&L,i,e)

先从尾部开始，依次将L.elem[j+1] = L.elem[j], j==i时停止，将e赋值给L.elem[i]，并且将L.length加1.

（11）ListDelete(&L,i,&e)

先将L.elem[i]赋值给e, 在从i开始将L.elem[j] = L.elem[j+1]至表的末端，并将L.length减一。

（12）ListTraverse(L，visit（）)

遍历线性表，依次输入每一个值。

(13）Exit

此处判断是否保存数据并进行相应的变换以后，退出系统。

## 1.7算法实现

包含时间复杂度和空间复杂度的分析：

//这是我在vs2015中写的代码，其中含有c++中引用的使用，如果要移植到其他编译环境下的话，则需要将引用改成指针

// the first experiment in data\_structure\_2.cpp : 定义控制台应用程序的入口点。

//

#include "stdafx.h"

//2015\_11\_11

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

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

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

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

#define MAXINPUT 20 //filename 的最大长度

char filename[5][MAXINPUT] = {

"List1.dat","List2.dat", "List3.dat", "List4.dat", "List.dat"

};

int num = 0;

int key1 = 0;

typedef struct { //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

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

status IntiaList(SqList & L);

status IntiaList\_file(SqList &L);

/\*write by wdp\*/

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); //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

//private functon

status saveFile(SqList L);

/\*------------------------------------------------------\*/

int main(void) {

SqList L;

char getChoose;

int choose = 1;

while (choose) {

char key;

printf("是否需要对文件进行操作? Y or N\n");

key = getchar();

getchar();

if (key == 'Y' || key == 'y')

key1 = 1;

int op = 1, i = 0;

ElemType cur\_e = 0, prev\_e = 0, next\_e = 0, e = 0;

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(" 请选择你的操作[0~12]:");

scanf\_s("%d", &op);

switch (op) {

case 1:

if (key1)

{

if (IntiaList\_file(L) == OK)

{

printf("文件加载创建成功！\n");

if (!L.elem)

printf("线性表创建失败\n");

else

{

printf("线性表创建成功\n");

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

}

}

else

printf("文件加载失败！\n");

getchar(); getchar();

break;

}

else

{

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

else printf("线性表创建失败！\n");

getchar(); getchar();

break;

}

case 2:

if (DestroyList(L) == OK)

printf("线性表销毁成功！\n");

else

printf("线性表销毁失败！\n");

getchar(); getchar();

break;

case 3:

if (ClearList(L) == OK)

printf("线性表清空成功！\n");

else

printf("线性表清空失败！\n");

getchar(); getchar();

break;

case 4:

if (ListEmpty(L) == OK)

printf("线性表为空！\n");

else

printf("线性表不为空！\n");

getchar(); getchar();

break;

case 5:

printf("线性表的长度为%d\n", ListLength(L));

getchar(); getchar();

break;

case 6:

getchar();

printf("请输入线性表需要查询的位置的下标：\n");

scanf\_s("%d", &i);

getchar();

if (GetElem(L, i, e) == OK)

{

printf("查询成功！\n");

printf("查询结果为%d\n", e);

}

else

printf("查询失败！\n");

getchar();

break;

case 7:

getchar();

printf("请输入要查询的数：\n");

scanf\_s("%d", &i);

if (LocateElem(L, i) == OK)

printf("查询成功！\n");

else

printf("查询失败！\n");

getchar(); getchar();

break;

case 8:

getchar();

printf("请输入你想要查询前一个数的当前数：\n");

scanf\_s("%d", &cur\_e);

getchar();

if (PriorElem(L, cur\_e, prev\_e) == OK)

printf("查询结果为：%d\n", prev\_e);

else

printf("查询失败！\n");

getchar();

break;

case 9:

getchar();

printf("请输入你想要查询前一个数的当前数：\n");

scanf\_s("%d", &cur\_e);

if (NextElem(L, cur\_e, next\_e) == OK)

printf("查询结果为：%d\n", next\_e);

else

printf("查询失败！\n");

getchar(); getchar();

break;

case 10:

getchar();

printf("请输入你想要插入的数据：\n");

scanf\_s("%d", &e);

printf("请输入你想要插入数据的位置：\n");

scanf\_s("%d", &i);

int h;

h = ListInsert(L, i, e);

if (h == OK)

printf("插入成功！\n");

else if (h == ERROR)

printf("插入失败！\n");

else

printf("分配空间失败\n");

getchar(); getchar();

break;

case 11:

getchar();

printf("请输入你想要删除的数的位置：\n");

int i11;

scanf\_s("%d", &i11);

if (ListDelete(L, i11, e) == OK)

{

printf("删除成功！\n删除的数字为%d\n", e);

}

else

printf("删除失败！\n");

getchar();

getchar();

break;

case 12:

if (L.length == 0)

printf("线性表为空表\n");

if (ListTrabverse(L) == ERROR) printf("线性表未初始化！\n");

getchar(); getchar();

break;

case 0:

printf("是否需要保持数据？Y or N\n");

getchar();

char save\_or\_not;

save\_or\_not = getchar();

getchar();

if (save\_or\_not == 'Y' || save\_or\_not == 'y')

{

if (saveFile(L) == OK)

printf("保存成功！\n");

else

printf("保存失败！\n");

}

break;

}//end of switch

}//end of while

printf("是否继续使用此系统？Y or N\n");

getChoose = getchar();

getchar();

if (getChoose == 'N' || getChoose == 'n')

break;

}

printf("欢迎下次再使用本系统！\n");

return 0;

}//end of main()

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

/\*时间复杂度:

F(t) = O(1)

空间复杂度：

F(t) = O(1)

\*/

status IntiaList(SqList & L) {

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

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

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

return OK;

}

/\*时间复杂度:

F(t) = O(n)

空间复杂度：

F(t) = O(1)

\*/

status IntiaList\_file(SqList &L)

{

FILE \* fp = NULL;

int i = 0, it = 0;

for (it = 0; it < 5; it++)

printf("%d %s\n", it + 1, filename[it]);

printf("请选择要操作的文件编号:\n");

scanf\_s("%d", &num);

char \*p = filename[num];

if ((fp = fopen(p, "r")) == NULL)

{

fopen(p, "w");

printf("Error!\n");

return ERROR;

}

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

while (!feof(fp))

{

fread(&(L.elem[i]), 1, sizeof(ElemType), fp);

printf("test 1\n");

i++;

}

L.length = i - 1;

L.listsize = LIST\_INIT\_SIZE;

fclose(fp);

return OK;

}

status ListTrabverse(SqList L) {

if (!L.elem)

return ERROR;

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;

}

/\*write by wdp\*/

/\*时间复杂度:

F(t) = O(1)

空间复杂度：

F(t) = O(1)

\*/

status DestroyList(SqList &L)

{

L.length = 0;

free(L.elem);

return OK;

}

/\*时间复杂度:

F(t) = O(1)

空间复杂度：

F(t) = O(1)

\*/

status ClearList(SqList &L)

{

L.length = 0;

return OK;

}

/\*时间复杂度:

F(t) = O(1)

空间复杂度：

F(t) = O(1)

\*/

status ListEmpty(SqList L)

{

if (L.length > 0)

return FALSE;

return OK;

}

/\*时间复杂度:

F(t) = O(1)

空间复杂度：

F(t) = O(1)

\*/

int ListLength(SqList L)

{

if (!L.elem)

return 0;

return L.length;

}

/\*时间复杂度:

F(t) = O(1)

空间复杂度：

F(t) = O(1)

\*/

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

{

if (!L.elem)

return ERROR;

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

return ERROR;

e = L.elem[i - 1];

return OK;

}

/\*时间复杂度:

F(t) = O(n)

空间复杂度：

F(t) = O(1)

\*/

status LocateElem(SqList L, ElemType e)

{

int i = 0;

while (i < L.length)

{

if (L.elem[i] == e)

break;

i++;

}

printf("%d 在线性表中的位置是 %d\n", e, ++i);

return OK;

}

/\*时间复杂度:

F(t) = O(n)

空间复杂度：

F(t) = O(1)

\*/

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

{

int i = 0;

while (i < L.length)

{

if (L.elem[i] == cur\_e)

{

if (i == 0)

return ERROR;

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

break;

}

++i;

}

if (i == L.length)

return FALSE;

return OK;

}

/\*时间复杂度:

F(t) = O(n)

空间复杂度：

F(t) = O(1)

\*/

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

{

int i = 0;

while (i < L.length)

{

if (L.elem[i] == cur\_e)

break;

++i;

}

if (i == L.length)

return FALSE;

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

return FALSE;

next\_e = L.elem[++i];

return OK;

}

/\*时间复杂度:

F(t) = O(n)

空间复杂度：

F(t) = O(1)

\*/

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

{

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

return ERROR;

if (L.length >= L.listsize)

{

ElemType\* newbase;

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

if (!newbase)

return OVERFLOW;

L.elem = newbase;

L.listsize += LISTINCREMENT;

}

if (L.length != 0)

{

int j = 0;

for (j = L.length; j >= i; j--)

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

}

L.elem[i - 1] = e;

L.length += 1;

return OK;

}

/\*时间复杂度:

F(t) = O(n)

空间复杂度：

F(t) = O(1)

\*/

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

{

if (!L.elem)

return ERROR;

int j = i - 1;

e = L.elem[i - 1];

while (j < L.length - 1)

{

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

j++;

}

L.length -= 1;

return OK;

}

/\*时间复杂度:

F(t) = O(n)

空间复杂度：

F(t) = O(1)

\*/

status saveFile(SqList L)

{

FILE \*fp = NULL;

int i = 0;

if (key1)

{

fp = fopen(filename[num], "w");

i = 0;

while (i < L.length)

{

fwrite(&(L.elem[i]), sizeof(ElemType), 1, fp);

i++;

}

fclose(fp);

return OK;

}

else

{

for (i = 0; i < 5; i++)

printf("%d %s\n", i, filename[i]);

printf("请输入你要保存数据的文件代号：\n");

int hereChoose = 0;

scanf\_s("%d", &hereChoose);

getchar();

if ((fp = fopen(filename[i], "w")) == NULL)

return ERROR;

i = 0;

while (i < L.length)

{

fwrite(&(L.elem[i]), sizeof(ElemType), 1, fp);

i++;

}

fclose(fp);

return OK;

}

}

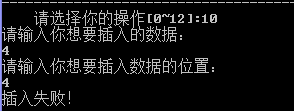
## 1.8算法测试

**对于没有进行文件读入的：**

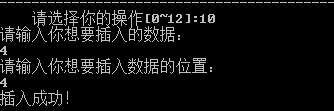
先初始化线性表：



对空表进行第四个位置的插入：



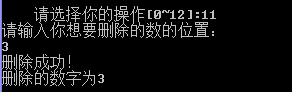
对于有3个数据的表进行第四个位置的插入：



ListTrabverse此时的线性表：



删除第三个数:



删除后的遍历



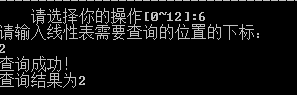
调用ListEmpty:



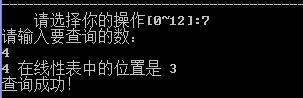
调用ListLength:



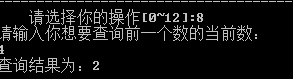
调用GetElem:



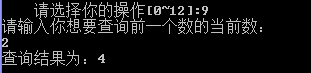
调用LocateList:



调用PriorElem

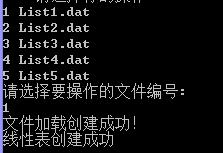


调用NextElem:

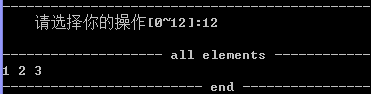


最后退出系统。

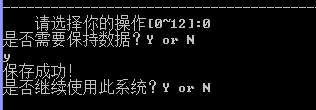
**接下来，是对于文件初始化线性表的部分，下图表示了我选择了文件List1.dat进行线性表初始化：**



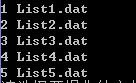
这是初始化之后对其进行遍历：



其他函数的步骤几乎不变，这里是要结束了文件List1.dat的操作，如果要继续对其他线性表进行操作的话，就需要输入y or Y:



这是重新进入系统后的准备对其他文件进行初始化的界面：



综上所述，无论对于无文件线性表初始化并且对其进行操作，还是对于文件的读入并且初始化，或者对于线性表的文件保存，基本功能已经得以实现。

## 1.9实验小结

在这个以线性表为核心的实验中，我自己做的系统，对于线性表的基本操作已经基本实现，我对于这一点很满意。但是，我的系统略显不足的是，它没有实现对于文件名输入—创建—保存的功能，文件名只能够存在一个外部数组之中，这是一个缺陷，不过还是略显幼稚的实现了对于多文件的操作。

在试验的过程中，很长时间没有写这么长代码的我，确实意识到了自己的不足，特别是对于文件的读入、存储工作，我显得有些力不从心，自己翻书查看和问了同学才把这一部分的内容做完，感觉很充实。而且，在调试程序的过程中收获了许多快乐。