

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术校交1901班**

**学 号： U201910681**

**姓 名： 骆瑞霖**

**指导教师： 周全**

**报告日期： 2020年 9月 29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 3](#_Toc458159879)

[1.1 问题描述 3](#_Toc458159880)

[1.2 系统设计 4](#_Toc458159882)

[1.3 系统实现 6](#_Toc458159883)

[1.4 测试用例 9](#_Toc458159883)

[1.5 实验小结 13](#_Toc458159884)

[2 基于链式存储结构的线性表实现 14](#_Toc458159885)

[2.1 问题描述 14](#_Toc458159886)

[2.2 系统设计 15](#_Toc458159887)

[2.3 系统实现 18](#_Toc458159888)

[2.4 测试用例 23](#_Toc458159888)

[2.5 实验小结 2](#_Toc458159889)9

[3 基于二叉链表的二叉树实现 30](#_Toc458159890)

[3.1 问题描述 30](#_Toc458159891)

[3.2 系统设计 31](#_Toc458159892)

[3.3 系统实现 34](#_Toc458159893)

[3.4 测试用例 44](#_Toc458159888)

[3.5 实验小结 57](#_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)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

**1.1.1 总述**

采用顺序表的物理结构，实现一个带有菜单的演示系统，定义了初始化表，销毁表，清空表，判定空表，求表长，多表操作等多个函数；在实现操作过程中会给出相应输入提示，本系统亦可以使用文件的形式进行存储和数据提取。

线性表是最基本、最简单、也是最常用的一种数据结构。线性表中数据元素之间的关系是一对一的关系，即除了第一个和最后一个数据元素之外，其它数据元素都是首尾相接的（注意，这句话只适用大部分线性表，而不是全部。比如，循环链表逻辑层次上也是一种线性表（存储层次上属于链式存储），但是把最后一个数据元素的尾指针指向了首位结点）。我们说“线性”和“非线性”，只在逻辑层次上讨论，而不考虑存储层次，所以双向链表和循环链表依旧是线性表。在数据结构逻辑层次上细分，线性表可分为一般线性表和受限线性表。一般线性表也就是我们通常所说的“线性表”，可以自由的删除或添加结点。受限线性表主要包括栈和队列，受限表示对结点的操作受限制。线性表的逻辑结构简单，便于实现和操作。因此，线性表这种数据结构在实际应用中是广泛采用的一种数据结构。

**1.1.2 多表管理与附加功能**

在该系统中，可以对不同序号的顺序表进行操作，进入系统时可以选择16号操作来选择在哪一个顺序表上进行操作，默认为1号顺序表。

附加：

1）实现线性表的文件保存和线性表文件加载，表中存储线性表的长度length，所能存储的最大元素数量listsize，以及各个元素；选择13号功能以将线性表的长度，容量，元素等存储入以初始化时所输入的文件里；

2）选择14号功能并输入文件的名称，可以将这个字符数组中的长度，容

量，元素等属性赋给线性表。

3）排序：选择15号功能对表中元素进行排序，输入0表示对线性表中元素进行升序排序，输入1表示对顺序表中元素进行降序排序。

## 1.2 系统设计

**1.2.1 数据的物理结构**

typedef struct{

ElemType \*elem;//数据存储的空间基地址

int length;//表长

int listsize;//表能存储的数据容量

}SqList;

**1.2.2 线性表的逻辑结构与函数运算**

线性表的数据逻辑结构定义：

ADT List{

数据对象：D={};

数据关系：}

};

程序以函数形式定义了初始化线性表，销毁线性表，清空线性表，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，数据保存至文件，加载数据文件的功能，这些功能可以在多张表上实现，使用时先选择所操作的表的编号。

1）初始化表（IntiaList）：构造一个空的线性表，初始条件是线性表L不存在或已存在；操作结果是构造一个空的线性表，操作成功返回OK；

2）销毁表（DestroyList）：销毁线性表，初始条件是线性表已存在，操作

结果是销毁线性表，与清空不同，销毁时释放全部空间，操作成功返回OK；

3）清空表（ClearList）：清空线性表，条件是线性表L已存在，操作结果

是将L重置为空表，操作成功返回OK；

4）判定空表（ListEmpty）：判定表是否为空，初始条件是线性表为L存在，若为空表则返回TRUE, 否则返回FALSE；

5）求表长（ListLength）：求线性表的表长，初始条件是线性表L存在，返回值为表中数据元素的个数；

6）获得元素（GetElem）：得到某个位置上元素的值，位置从1开始到表长大小， 操作结果是用int型变量e返回该位置的值；

7）查找元素（LocateElem）：查找与传入参数e满足cmp()关系的元素，程序初始默认为查找与e相同的元素的位置，若元素不存在，返回值为0；

8）获得前驱（PriorElem）：用pre\_e保存传入参数cur\_e的前驱元素的值，若存在返回其值，否则操作失败，pre\_e没有定义；

9）获得后继(NextElem)：用next\_e保存传入参数cur\_e的后继元素的值，若存在返回其值，否则操作失败，next\_e没有定义；

10)插入元素（ListInsert）：L已存在且非空，;结果是向位置i的前方插入元素e，插入成功返回OK，插入元素错误返回ERROR;

11)删除元素(ListDelete)：L已存在且非空，,结果是删除L的第i个元素，用传入参数e返回其值；删除成功返回OK,否则返回ERROR;

12)遍历表（ListTraverse）：按顺序输出线性表中所有元素的值；

13)线性表保存至文件(SaveList)：将线性表保存至文件，文件名为初始化表时所输入的文件名filename，包括写入表长，允许大小，所有元素;

14)加载文件中的线性表(LoadList)：以filename加载线性表，包括表长，允许大小，所有元素。

15）线性表排序（SortList）：选择0或1进行功能选择，输入0是对表中数据元素进行升序排序，选择1是对表中数据元素进行降序排序。

**1.2.3 输入与输出说明**

进入系统后首先出现功能菜单，首先输入所要操作的线性表的序号，这里允

许在1~10号上操作；选择完后就可以选择功能对应的序号进行相应操作，通过switch语句进行对应的功能操作，操作完毕后按回车或其他字符跳出当前操作，

继续执行while循环选择功能并实现，直至输入0退出系统。在每一次操作结束后系统都会显示操作成功或是失败。

若要使用多表操作，只需在当前操作时选择16号功能更换所要操作的表的序号即可，之前的表中的数据依然会保存。

## 1.3 系统实现

**1.3.1 环境说明**

本系统在Microsoft Visual Studio2019上执行，编译器为MSVC++ 14.2 \_MSC\_VER == 1920.并在其上编译运行。

**1.3.2 预定义常量**

#define TRUE 1 #define FALSE 0 #define OK 1

#define ERROR 0 #define INFEASTABLE -1 #define OVERFLOW -2

#define MAX\_NUM 10 #define LIST\_INIT\_SIZE 100 #define LISTINCREMENT 10

typedef int status; typedef int ElemType;

**1.3.3 线性表实现**

顺序存储结构的线性表使用三元表实现，SqList为结构体，\*elem是ElemType类型的指针，length是线性表的长度，listsize是线性表的容量。\*elem可以进行动态分配，当length>listsize时可通过realloc进行容量的重新分配，大小为listsize+容量增量。

使用结构体数组进行多表管理，初始化时，每个线性表的长度设置为0，listsize设置为0，定义MAX\_NUM为最大能操作的表的数量，当线性表数量超过MAX\_NUM时使用realloc重新分配空间,i\_num为所选择的要进行操作的表的序号。

在选择功能后，执行每一个所选择的case后的功能之前，都会首先判断

L[i\_num].elem是否为NULL，若为NULL则会输出“线性表不存在”。

因此，以下功能实现均建立在线性表存在的情况下：

1)status InitList（SqList& L）

传入所选择的序号的顺序表L[i]的地址，分别对三元表的每个元素进行初始

化，使用malloc函数对L.elem进行空间分配，若空间分配失败，返回OVERFLOW；L的长度length置为0，L的listsize置为LIST\_INIT\_SIZE,初始化成功后返回OK;

2）status DestroyList(SqList& L)

首先对L.elem是否为NULL进行判断，若L.elem不为空，则使用free（）函数释放空间，随后将L.elem置为NULL，L的length和listsize属性重置为0，返回OK；

3）status ClearList(SqList& L)

清空线性表，直接将表长length重置为0即可；

4）status ListEmpty(SqList L)

判断表是否为空表，即判断线性表的长度length是否为0，若为0，返回TRUE,否则不为空表，返回FALSE；

5)int ListLength(SqList L)

求线性表L的长度，直接返回L.length即可；

6）status GetElem(SqList L,int i,ElemType\* e)

需要对传入的位置值i进行判断，i的最小值为1，最大值为L.length，当不在这个范围内时，返回ERROR;位置正确时，将L.elem[i-1]赋给e，并返回OK;

7)status LocateElem(SqList L,ElemType e,status(\*cmp)(ElemType a,

ElemType b))

status cmp(ElemType a,ElemType b)

该函数需要返回满足函数cmp（）关系的数据元素的位置，首先需要对cmp（）函数进行定义，在此系统中由于要实现查找元素的功能，所以在cmp中定义为当输入参数相等时返回TRUE,否则返回FALSE；

在LocateElem函数中，对L.elem进行遍历，传入参数1为当前位置的数据元素，参数2位所要查找的数据元素，遍历时逐个传入cmp函数进行比较，当

cmp(L.elem[i],e)返回TRUE时，将该元素在表中的位置(++i)作为返回值返回；

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

cur\_e表示遍历时所要查找的元素，pre\_e用来记录可能存在的所要返回的前驱元素；当cur\_e与L.elem[0]相等时，说明第一个数就与所要查找的数据相等，那么这时它没有前驱元素；否则继续遍历，在遍历完毕前出现与cur\_e相等的元素时返回它的前驱，用pre\_e记录L.elem[i-1]，并返回OK；遍历完毕仍未找到元素，则返回OVERFLOW。

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

cur\_e表示遍历时所要查找的元素，next\_e用来记录可能存在的所要返回的后继元素；当cur\_e与L.elem[L.length-1]相等时，说明直到最后一个数据元素猜得到相等关系，那么此时它没有后继元素，返回OVERFLOW；若在到达最后一个元素之前就得到L.elem[i]==cur\_e，那么用next\_e记录L.elem[i+1]，返回OK；若遍历完毕都未能找到特定元素，则返回ERROR。

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

首先判断插入位置是否正确，若i<1或者i>L.length+1，则插入位置不正确，返回ERROR;

若L.length>=L.listsize，此时需要扩大线性表L的容量，扩充后容量为(L.listsize+LISTINCREMENT)\*sizeof(ElemType);

定义整形指针p指向第i个位置，整形指针q指向最后一个元素，从后向前遍历至p指向的位置，使后一个元素的值等于前一个元素，移动完成后，再将e值赋给\*q;操作完成后L.length自增1，并返回OK。

11)status ListDelete(SqList& L,int I,ElemTpe\* e)

首先判断删除的位置是否正确，若i<1或者i>L.length则输入的位置不正确，返回ERROR;

输入位置正确时，首先用e返回位置i上的数据元素大小，然后从第i个元素开始一直到到倒数第二个元素，使其都等于自身的下一个元素；操作完成后L的长度length自减1并返回OK。

12）status ListTraverse(SqList L)

利用for循环从第一个元素开始遍历到最后一个元素，输出数据元素即可，

返回值为L的长度L.length。

13）status SaveList(SqList &L,char\* filename)

定义指针FILE\*fp=fopen(filename,”w”)；若值为NULL，则输出“文件保存失败”，返回ERROR;

使用fprintf首先写入L的长度和容量两个属性，然后使用while循环遍历表，将每一个数据元素写入文件，写入完毕后打印表中数据元素的信息，并输出存储入文件的表的长度和容量，最后fclose(fp)。

14）status LoadList(SqList &L,char\* filename)

定义文件指针FILE\* fp=fopen(filename,”r”),若值为NULL，打印“文件加载失败”，返回ERROR;首先读到的两个数据元素分别赋值给L.length和L.listsize，因为存入时首先存入的是表长和表的容量；

随后读取的元素依次赋值给L中的elem[i]，每读取一个i++,即位置后移一位。同时为了判断操作成功，加载文件后函数中输出表长，容量大小，以及各个数据元素，最后执行fclose(fp)。

15）status SortList(SqList L,int tag)

如果表长为0，则返回ERROR,输出表为空的信息；表长为1时直接返回OK，因为一个元素无论降序升序都已经排序完毕。否则根据传入的tag来实现功能，当tag为0时使用冒泡排序法进行升序排序，冒泡排序一共需要进行L.length-1次遍历，每次遍历从位置0到L.length-1-i，当前一个元素大于后一个元素时进行交换，这样每一次遍历都将前L.length-i个元素里最大的元素冒泡到最后，这样就实现了升序排序；当tag为1时使用冒泡排序法进行降序排序，其他思路与上述同理，只要将交换条件改为前一个元素小于后一个元素即可。

**1.4测试用例**

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试功能 | 输入的参数 | 预计结果 |
| 用例1 | Choose the specific List | 13 | 选择错误，只能在1~10号顺序表上操作! |
| 用例2 | Choose the specific List | 2 | 选择链表2进行操作 |
| 用例3 | DestroyList | / | 线性表不存在 |
| 用例4 | ListInsert | / | 线性表不存在 |
| 用例5 | ClearList | / | 线性表不存在 |
| 用例6 | ListEmpty | / | 线性表不存在 |
| 用例7 | ListLength | / | 线性表不存在 |
| 用例8 | ListTraverse | / | 线性表不存在 |
| 用例9 | InitList | 表的名称：aa | 创建一个空的线性表，表的名称即filename为“aa” |
| 用例10 | ListTraverse | / | 线性表是空表 |
| 用例11 | ListLength | / | 线性表表长为0 |
| 用例12 | ListInsert | 元素：10 位置：1 | 表：10 |
| 用例13 | ListInsert | 元素：20 位置：2 | 表：10 20 |
| 用例14 | ListInsert | 元素：30 位置：3 | 表：10 20 30 |
| 用例15 | ListInsert | 元素：40 位置：4 | 表：10 20 30 40 |
| 用例16 | ListInsert | 元素：60 位置：6 | 插入位置不正确！  插入数据元素失败！ |
| 用例17 | ListEmpty | / | 线性表不是空表 |
| 用例18 | ListLength | / | 线性表表长为4 |
| 用例19 | GetElem | 2 | 20 |
| 用例20 | GetElem | 5 | 输入位置错误 |
| 用例21 | LocateElem | 40 | 4 |
| 用例22 | LocateElem | 35 | 该元素不存在 |
| 用例23 | PriorElem | 20 | 10 |
| 用例24 | PriorElem | 10 | 它不存在前驱元素 |
| 用例25 | PriorElem | 15 | 表中没有该元素 |
| 用例26 | NextElem | 20 | 30 |
| 用例27 | NextElem | 40 | 它不存在后继元素 |
| 用例28 | NextElem | 60 | 表中没有该元素 |
| 用例29 | ListDelete | 3 | 表：10 20 40 |
| 用例30 | ListDelete | 4 | 删除数据元素失败（此时只有三个元素，位置错误） |
| 用例31 | ListTraverse | / | 10 20 40 |
| 用例32 | SaveList | / | 表中元素为:  10 20 40 存储的表长为3,所能存储数据元素的数量为100.  文件保存成功  文件名为初始化时所输入的“aa” |
| 用例33 | LoadList | aa | 加载得到表长为3,容量为100的线性表.  表中元素为10 20 40 文件加载成功 |
| 用例34 | ClearList | / | 得到长度为0的空表 |
| 用例35 | ListLength | / | 0 |
| 用例36 | ListEmpty | / | 线性表为空表 |
| 用例37 | DestroyList | / | 销毁线性表成功，再操作会显示表不存在（以选择功能3,4,5为例） |
| 用例38 | ListEmpty | / | 线性表不存在 |
| 用例39 | ClearList | / | 线性表不存在 |
| 用例40 | ListLength | / | 线性表不存在 |
| 用例41 | SortList | 1 5 5 14 8 tag:0 | 1 5 5 8 14 |
| 用例42 | SortList | 1 5 5 14 8 tag:1 | 14 8 5 5 1 |
| 用例43 | SortList | 空表 | 表为空表！ |
| 用例44 | ListInsert | 元素：1 2 4 5  位置3插入8 | 1 2 8 4 5 |
| 用例45 | ListInsert | 元素：1 2 4 5  位置2插入-3 | 1 -3 2 4 5 |
| 用例46 | SortList | 1 -3 2 4 5 tag:0 | -3 1 2 4 5 |

输出界面

|  |  |
| --- | --- |
| 测试用例 | 运行结果 |
| 用例1 |  |
| 用例2 |  |
| 用例3 |  |
| 用例4 |  |
| 用例5 |  |
| 用例6 |  |
| 用例7 |  |
| 用例8 |  |
| 用例9 |  |
| 用例10 |  |
| 用例11 |  |
| 用例12 |  |
| 用例13 |  |
| 用例14 |  |
| 用例15 |  |
| 用例16 |  |
| 用例17 |  |
| 用例18 |  |
| 用例19 |  |
| 用例20 |  |
| 用例21 |  |
| 用例22 |  |
| 用例23 |  |
| 用例24 |  |
| 用例25 |  |
| 用例26 |  |
| 用例27 |  |
| 用例28 |  |
| 用例29 |  |
| 用例30 |  |
| 用例31 |  |
| 用例32 |  |
| 用例33 | 加载后表中元素： |
| 用例34 |  |
| 用例35 |  |
| 用例36 |  |
| 用例37 |  |
| 用例38 |  |
| 用例39 |  |
| 用例40 |  |
| 用例41 |  |
| 用例42 |  |
| 用例43 |  |
| 用例44 |  |
| 用例45 |  |
| 用例46 |  |

**1.5 实验小结**

在本次实验中，我实现了12个基本操作与文件读写线性表以及线性表的数据元素的排序，并通过多表管理的方式加深了对线性表管理信息的掌握，在实现函数功能的过程中初次完成了完善一个系统功能的尝试，而不是像C语言那样完成一道道习题，因此，对于函数功能的组装，共同实现一个相对完善的功能系统有了一个大致的掌握。

在实验中也出现了不少问题，例如参数的传递，对各种边界条件的疏忽，在

经过调试与修改后，能够基本对于所有不同的操作情况，系统会给出正确的反馈，操作正确时会正确地进行插入删除定位等操作，当操作错误时，例如插入位置不正确，首结点无前驱，尾结点无后继等特殊情况系统都会打印相应的信息来给出正确的反馈。通过这次实验，锻炼了我设计程序时的全局观，即如何相对完整地制作功能的实现，同时锻炼了我的严谨性，对于不同的情况，尤其是不正确的情况下，需要给出不同的解决办法，收益颇丰。