实验报告

# 一、设计人员相关信息

## 1.设计者

姓名：陈怡冰

学号：24012022

班号：24计算机科学与技术1班

## 2.设计日期：

2025年9月8日-2025年9月12日

# 二、程序设计相关信息

## 1.实验题及问题描述。

实验  顺序表操作

顺序表(SeqList)的定义

#define MaxSize 10      //最大允许长度

typedef struct

{    Elemtype  data[MaxSize+1];   //存储数组

        int length;       //终端结点在数组的位置

} SqList;

注意：本次作业，main（）中可以调用sqlist.cpp中的初始化、创建、求长度和释放函数; 自己定义的子函数中必须写基本语句，不能调用sqlist.cpp中的函数，可以调用自己写的子函数。

1、已知顺序表L为按值递增有序的，编写算法将数据元素e插入到顺序表L中，使之仍有序。

2、已知顺序表L为按值递增有序的，设计算法清除顺序表中多余重复元素，注意其删除位置之后的元素如何移动。

3、顺序表LA和顺序表LB中的元素值无序，求出LA和LB的并集，结果存储在顺序表LC中，LA、LB单个表中无重复，相互间有重复数据。

4、顺序表LA和顺序表LB中的元素值无序，求出LA和LB的交集，结果存储在顺序表LC中，LA、LB单个表中无重复，相互间有重复数据。

5、顺序表LA和顺序表LB中的元素值无序，求出LA和LB的差集，结果存储在顺序表LC中，LA、LB单个表中无重复，相互间有重复数据。

## 2.实验目的。

（1）复习程序设计调试的工作环境。

（2）熟练掌握顺序表的操作的设计方法和技巧以及调试过程，重点是顺序表的插入和删除操作的设计。

## 3.数据结构设计。

元素（Elemtype）定义

1. **typedef** **int** ElemType;  // 假设元素为整型

顺序表(SeqList)的定义

1. **typedef** **struct** {
2. ElemType data[MaxSize];  // 静态数组存储元素
3. **int** length;               // 记录当前有效元素个数
4. } SqList;

## 4.程序结构(程序中的函数调用关系图)。

## 5.主要的算法描述。

（1）有序插入e

从头开始查找第一个元素大于等于 e 的位置 j；

将顺序表中data[j] 到末尾的所有元素后移一位；

在顺序表位置 j 插入 e；

表长加一。

（2）删除重复元素

遍历顺序表，比较 data[i] 与 data[i+1]；

若相等，则调用 DeleteList(L, i) 删除当前位置；

否则继续向后扫描；

（3）并集（LA ∪ LB）

初始化空表 LC；

先将 LA 所有元素复制到 LC；

再遍历 LB 每个元素，若该元素不在 LC 中，则插入 LC 末尾；

（4）交集（LA ∩ LB）

初始化空表 LC；

遍历 LB 的每个元素；

若该元素在 LA 中存在，则将其加入 LC；

（5）差集（LA - LB）

遍历 LA 每个元素；

若该元素在 LB 中不存在，则插入 LC。

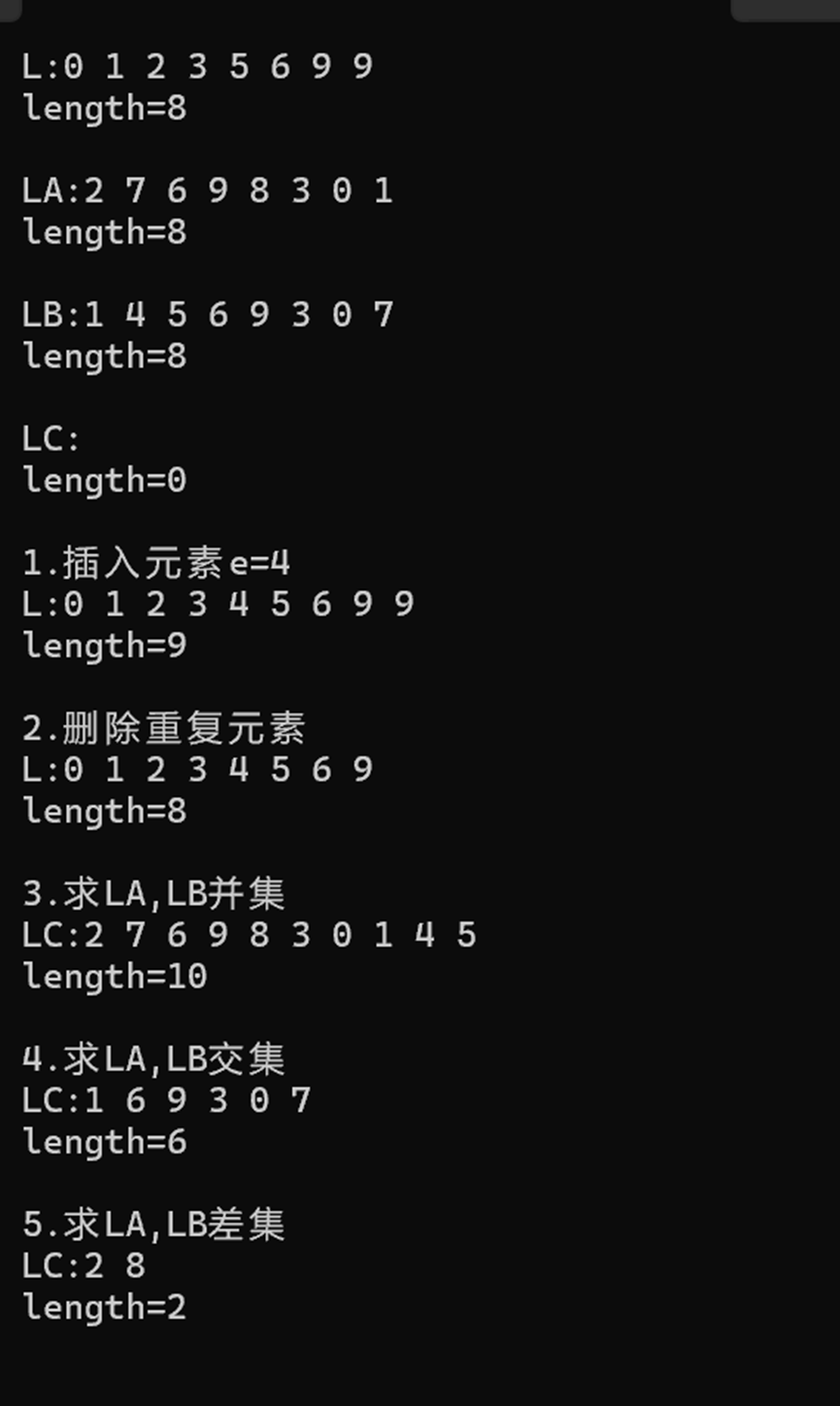
## 6.实验源程序。

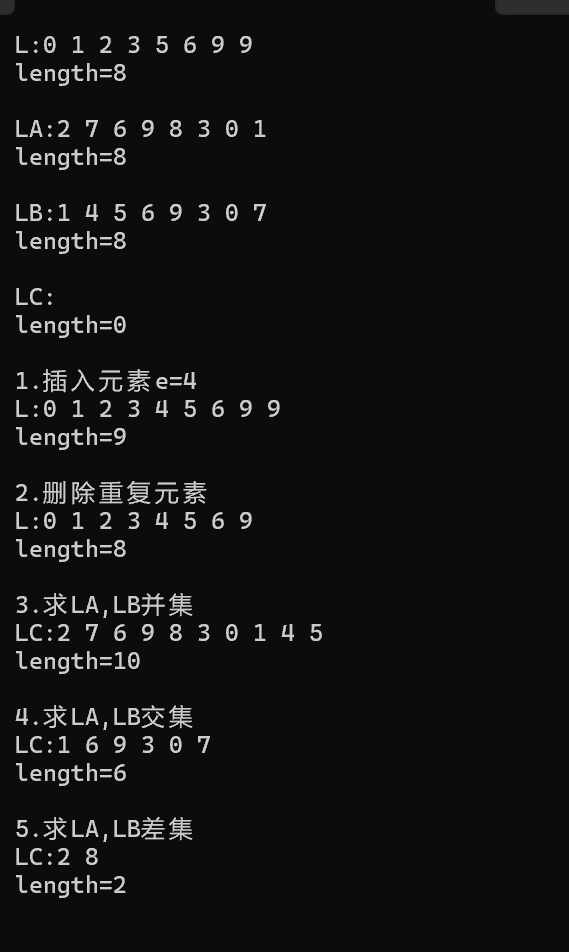
1. #include<stdio.h>
2. #include<malloc.h>
3. #define MaxSize 10
5. **typedef** **int** ElemType;
6. **typedef** **struct** {
7. ElemType data[MaxSize];
8. **int** length;
9. } SqList;
11. **void** InitList(SqList \*&L) {
12. L = (SqList\*)malloc(**sizeof**(SqList));
13. L->length = 0;
14. }
16. **void** CreateList(SqList \*&L, ElemType a[], **int** n) {
17. L = (SqList\*)malloc(**sizeof**(SqList));
18. **for** (**int** i = 0; i < n; i++) {
19. L->data[i] = a[i];
21. }
22. L->length = n;
24. }
26. **void** PrintList(SqList\*&L) {
27. **for** (**int** i = 0; i < L->length; i++) {
28. printf("%d ", L->data[i]);
29. }
30. printf("\nlength=%d\n\n", L->length);
31. }
33. **bool** InsertList(SqList \*&L, **int** j, ElemType &e) {
34. **bool** flag = **true**;
35. **if** (L->length >= MaxSize) {
36. flag = **false**;
37. } **else** {
38. **for** (**int** i = L->length - 1; i >= j; i--) {
39. L->data[i + 1] = L->data[i];
40. }
41. L->data[j] = e;
42. L->length += 1;
43. flag = **true**;
44. }
46. **return** flag;
47. }
49. **bool** DeleteList(SqList \*&L, **int** j) {
50. **bool** flag = **true**;
51. **if** (L->length <= j) {
52. flag = **false**;
53. } **else** {
54. **for** (**int** i = j; i < L->length; i++) {
55. L->data[i] = L->data[i + 1];
56. }
58. L->length -= 1;
59. flag = **true**;
60. }
62. **return** flag;
63. }
65. **bool** FindList(SqList \*&L, ElemType &e) {
66. **bool** flag = **false**;
67. **for** (**int** i = 0; i < L->length; i++) {
68. **if** (L->data[i] == e) {
69. flag = **true**;
70. **break**;
71. }
72. }
73. **return** flag;
74. }
76. **bool** AddList(SqList \*&L, SqList\*&L\_add) {
77. **bool** flag = **true**;
78. **int** j;
79. **if** (L->length + L\_add->length > MaxSize) {
80. printf("Add Error\n");
81. flag = **false**;
82. } **else** {
83. **for** (**int** i = 0; i < L\_add->length; i++) {
84. j = L->length + i;
85. L->data[j] = L\_add->data[i];
86. }
87. L->length = L->length + L\_add->length;
88. }
89. **return** flag;
90. }
92. **void** ClearList(SqList \*&L) {
93. L->length = 0;
94. }

97. **int** main() {
98. SqList\* L, \*LA, \*LB, \*LC;
99. ElemType a[8] = {0, 1, 2, 3, 5, 6, 9, 9};
100. ElemType la[8] = {2, 7, 6, 9, 8, 3, 0, 1};
101. ElemType lb[8] = {1, 4, 5, 6, 9, 3, 0, 7};
102. **int** n = 8;
103. ElemType e = 4;
104. CreateList(L, a, n);
105. CreateList(LA, la, n);
106. CreateList(LB, lb, n);
107. InitList(LC);
108. printf("L:");
109. PrintList(L);
110. printf("LA:");
111. PrintList(LA);
112. printf("LB:");
113. PrintList(LB);
114. printf("LC:");
115. PrintList(LC);

118. **int** j = 0;
119. **for** (**int** i = 0; i < L->length; i++) {
120. **if** (L->data[i] < e) {
121. j = i + 1;
122. } **else** {
123. j = i;
124. **break**;
125. }
126. }
127. InsertList(L, j, e);
128. printf("1.插入元素e=4\n");
129. printf("L:");
130. PrintList(L);
132. j = 0;
133. **while** (j < L->length - 1) {
134. **if** (L->data[j] == L->data[j + 1]) {
135. DeleteList(L, j);
136. } **else** {
137. j++;
138. }
139. }
140. printf("2.删除重复元素\n");
141. printf("L:");
142. PrintList(L);
144. AddList(LC, LA);
145. **for** (**int** i = 0; i < LB->length; i++) {
146. **if** (!FindList(LC, LB->data[i])) {
147. InsertList(LC, LC->length, LB->data[i]);
148. }
149. }
150. printf("3.求LA,LB并集\nLC:");
151. PrintList(LC);
153. ClearList(LC);
154. **for** (**int** i = 0; i < LB->length; i++) {
155. **if** (FindList(LA, LB->data[i])) {
156. InsertList(LC, LC->length, LB->data[i]);
157. }
158. }
159. printf("4.求LA,LB交集\nLC:");
160. PrintList(LC);
162. ClearList(LC);
163. **for** (**int** i = 0; i < LA->length; i++) {
164. **if** (!FindList(LB, LA->data[i])) {
165. InsertList(LC, LC->length, LA->data[i]);
166. }
167. }
168. printf("5.求LA,LB差集\nLC:");
169. PrintList(LC);
171. **return** 0;
173. }

## 7.实验数据和实验结果分析。

实验数据：

实验结果：

实验结果分析：

（1）插入操作正确找到插入点，维持了有序性；

（2）去重逻辑合理，利用相邻比较避免多次查找；

（3）集合运算依赖 FindList() 实现成员判定，效率为 O(n)，整体复杂度为 O(n²)，适合小规模数据；

（4）所有操作均未越界，符合 MaxSize=10 的限制；

（5）并集达到最大长度，体现了空间约束的影响。

## 8.实验体会。

通过本次实验，我深入理解了顺序表的物理存储特性及其基本操作的实现机制。特别是在实现“有序插入”和“删除重复元素”时，需要仔细考虑元素的移动方向和边界条件，稍有不慎就会导致覆盖或越界错误。

在完成集合运算的过程中，我体会到抽象思维的重要性——将并、交、差转化为对“存在性”的判断，再结合插入操作即可实现。虽然当前采用的是简单遍历法（时间复杂度较高），但对于小型数据集是可行且直观的。

此外，在调试过程中，我发现当删除元素后索引未及时调整会导致漏删，因此在去重部分采用了 while 循环配合手动控制索引的方式，确保每个位置都被检查。

总体而言，本次实验不仅巩固了我对线性表的理解，也提升了我的编程实践能力和调试技巧。未来可以尝试动态扩容或优化查找方式（如哈希）来提升性能。

# 三、实验提交内容

实验报告、实验源程序清单和可执行文件。

实验源程序清单：

实验1-顺序表.cpp