### 第三周作业

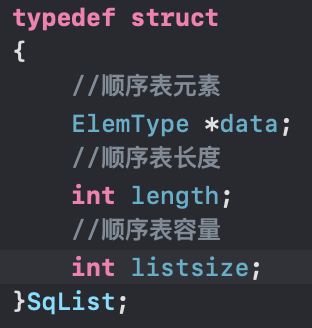
学号：2021529620004 姓名：LOW REN HONG

遇到的问题和解决方法

1. 头文件不匹配
   1. #include<malloc.h> 这里修改成#Include<sys/malloc.h>
   2. #include<io.h> 修改成#include<sys/uio.h>
   3. #include<process.h> 因为没找到所以这里直接注释掉

# 线性表

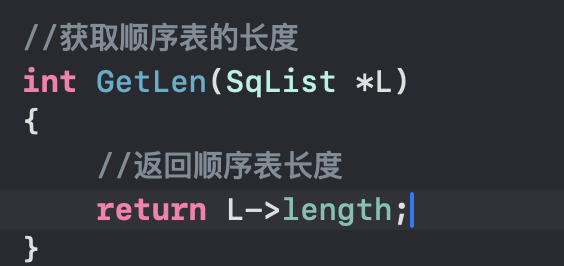
1. 定义结构题



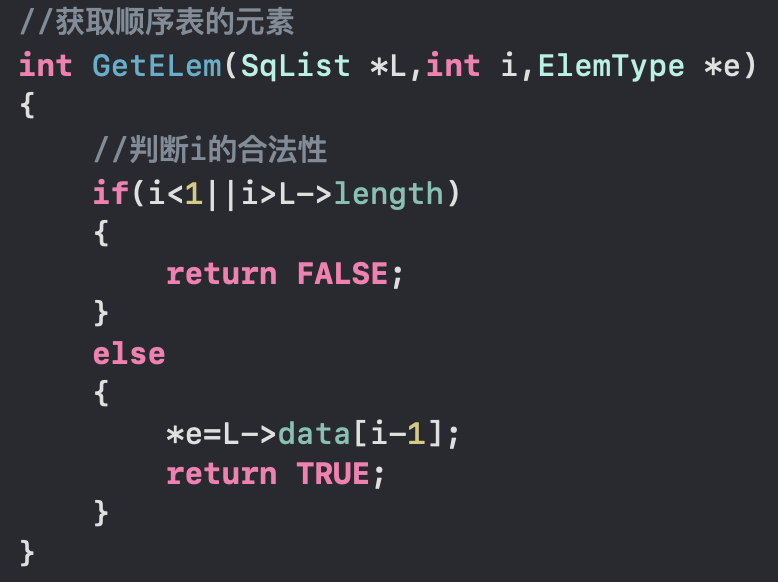
1. 初始化顺序表



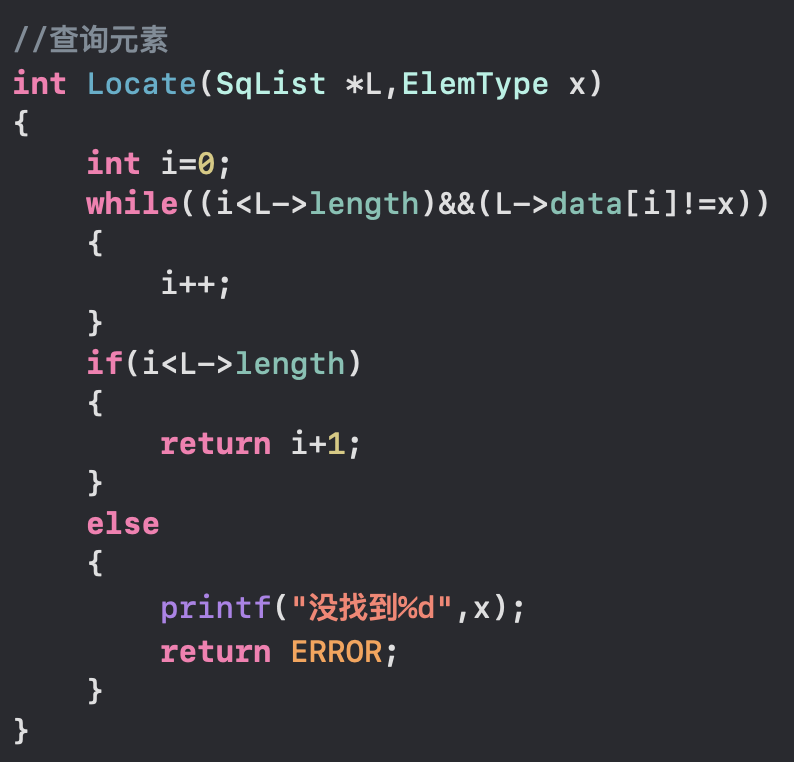
1. 获取顺序表的长度



1. 获取顺序表的元素



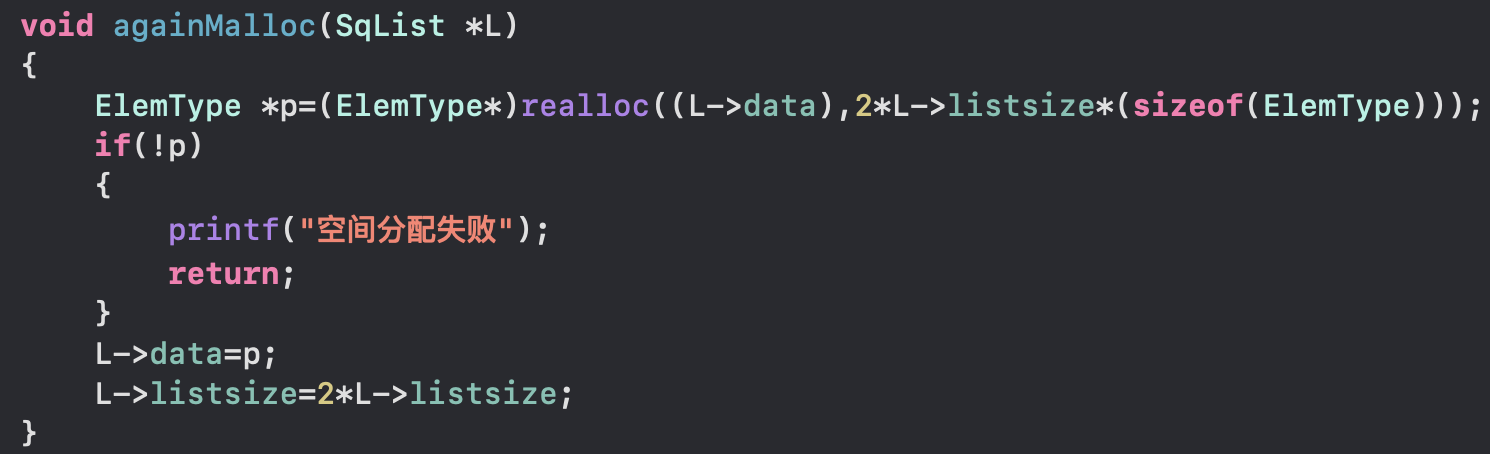
1. 查询元素



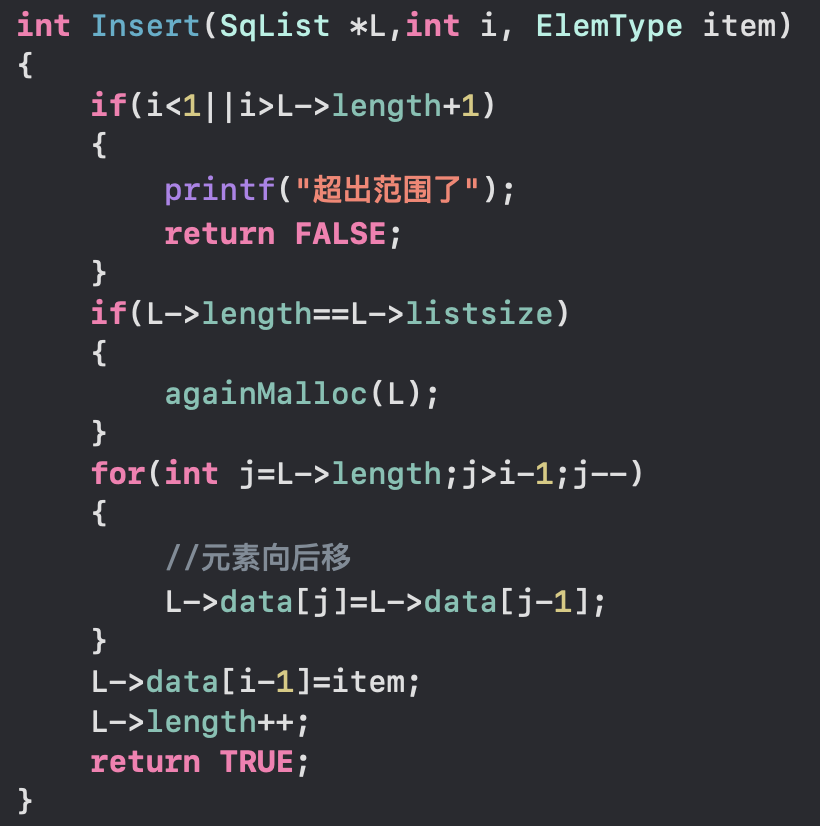
1. 清空顺序表



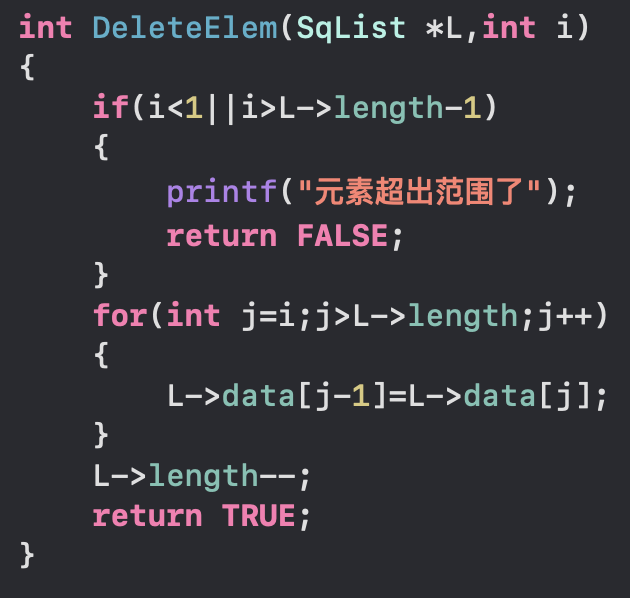
1. 重新分配空间



1. 插入顺序表



1. 删除元素



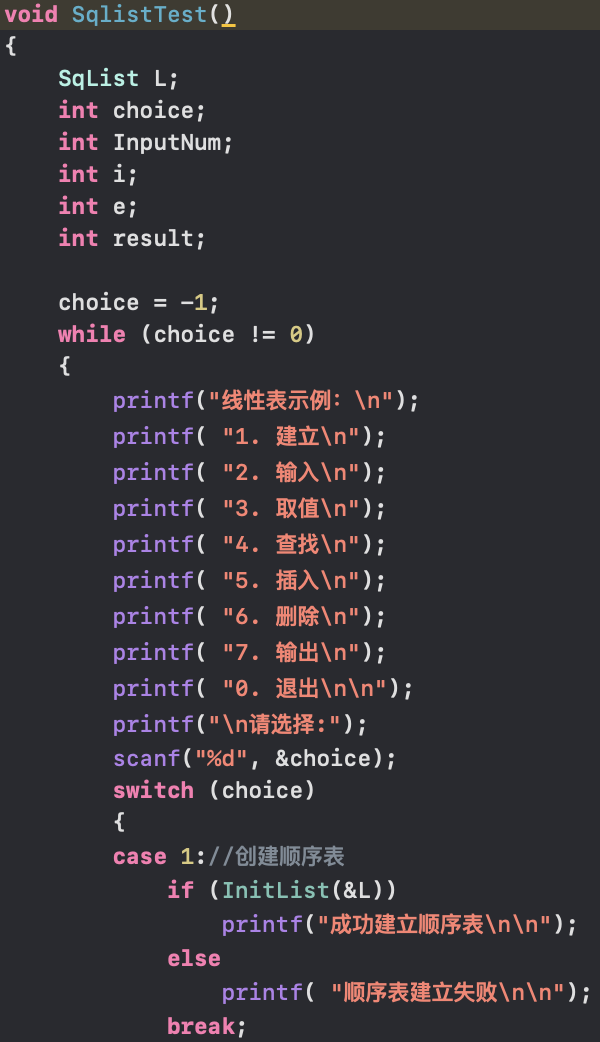
1. 输出元素



1. 创建顺序表



1. 测试链表

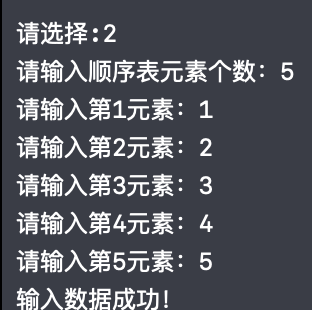


## **测试**

1.创建线性表

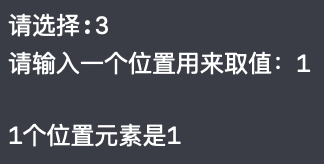


2.输入线性表

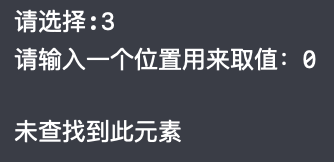
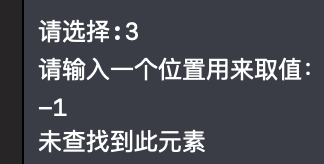


3.取值线性表

正常取值

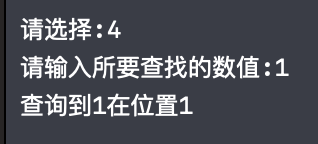


超出范围取值

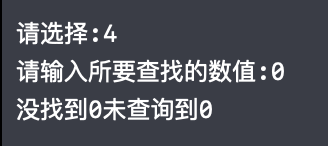
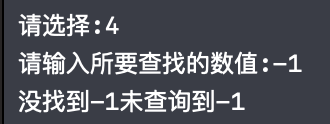
 

4.查找线性表

正常查询

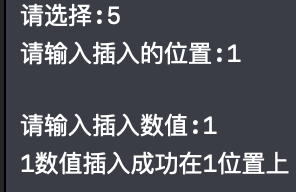


输入元素外的值

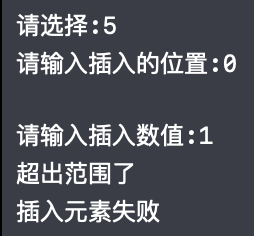
 

5．插入线性表

正常插入

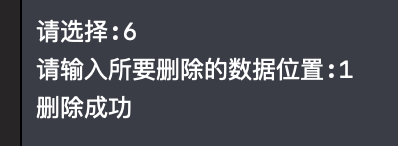


插入异常

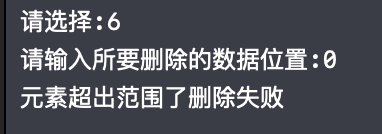


6．删除线性表

正常删除



删除异常



7.输出

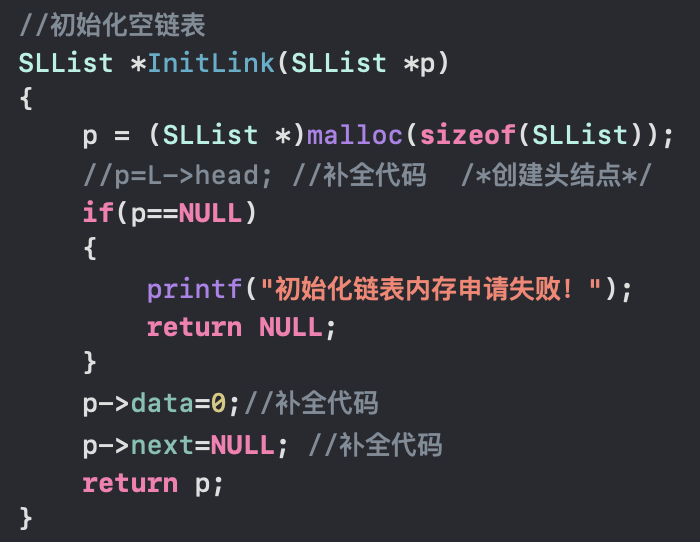


# 单链表

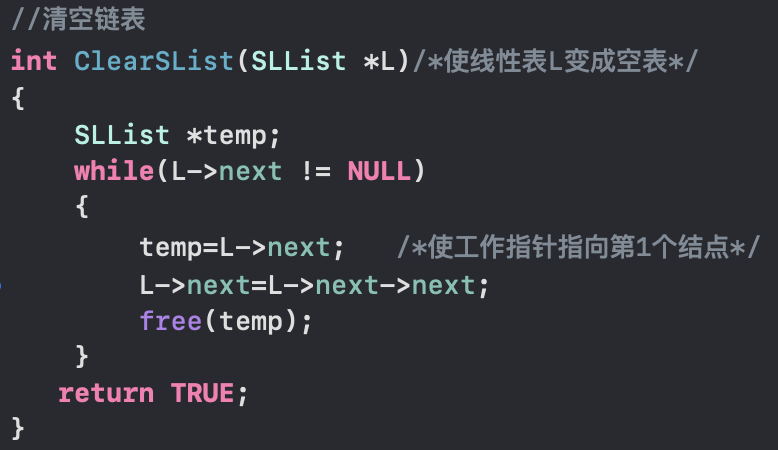
1. 定义结构题



1. 初始化空链表



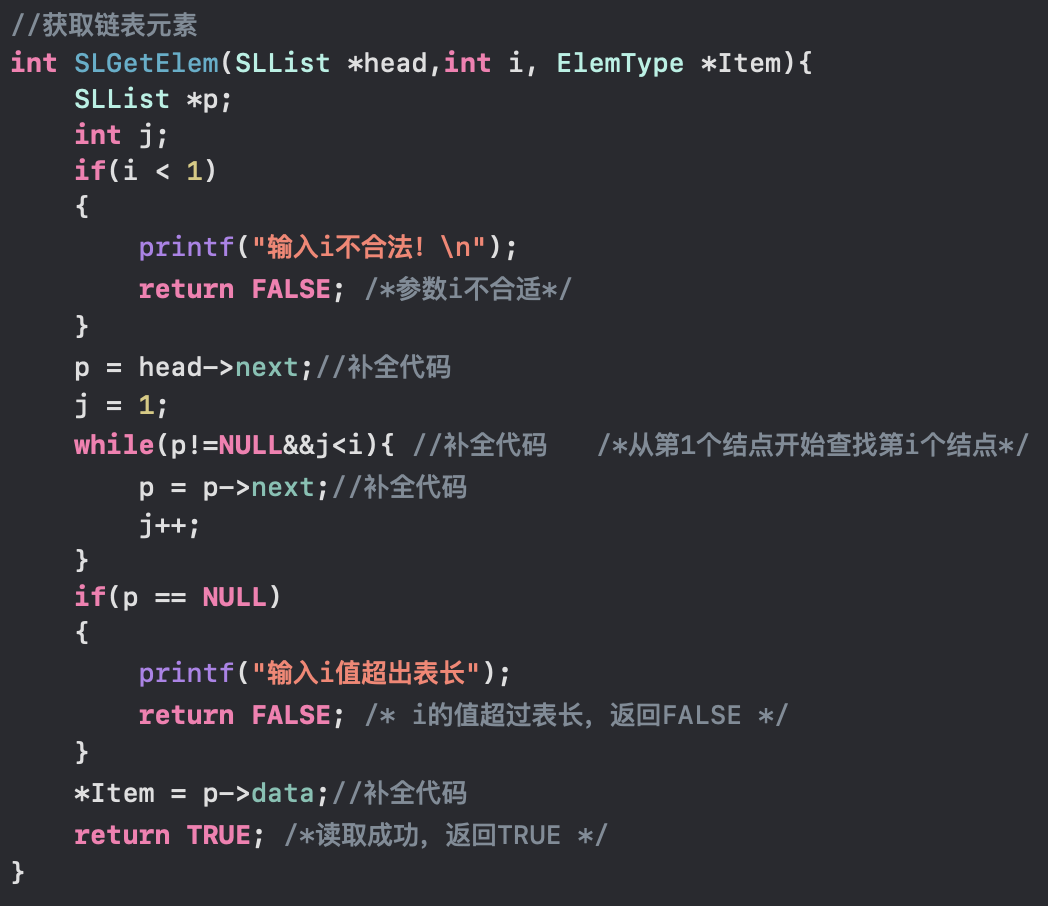
1. 清空链表



1. 获取链表长度



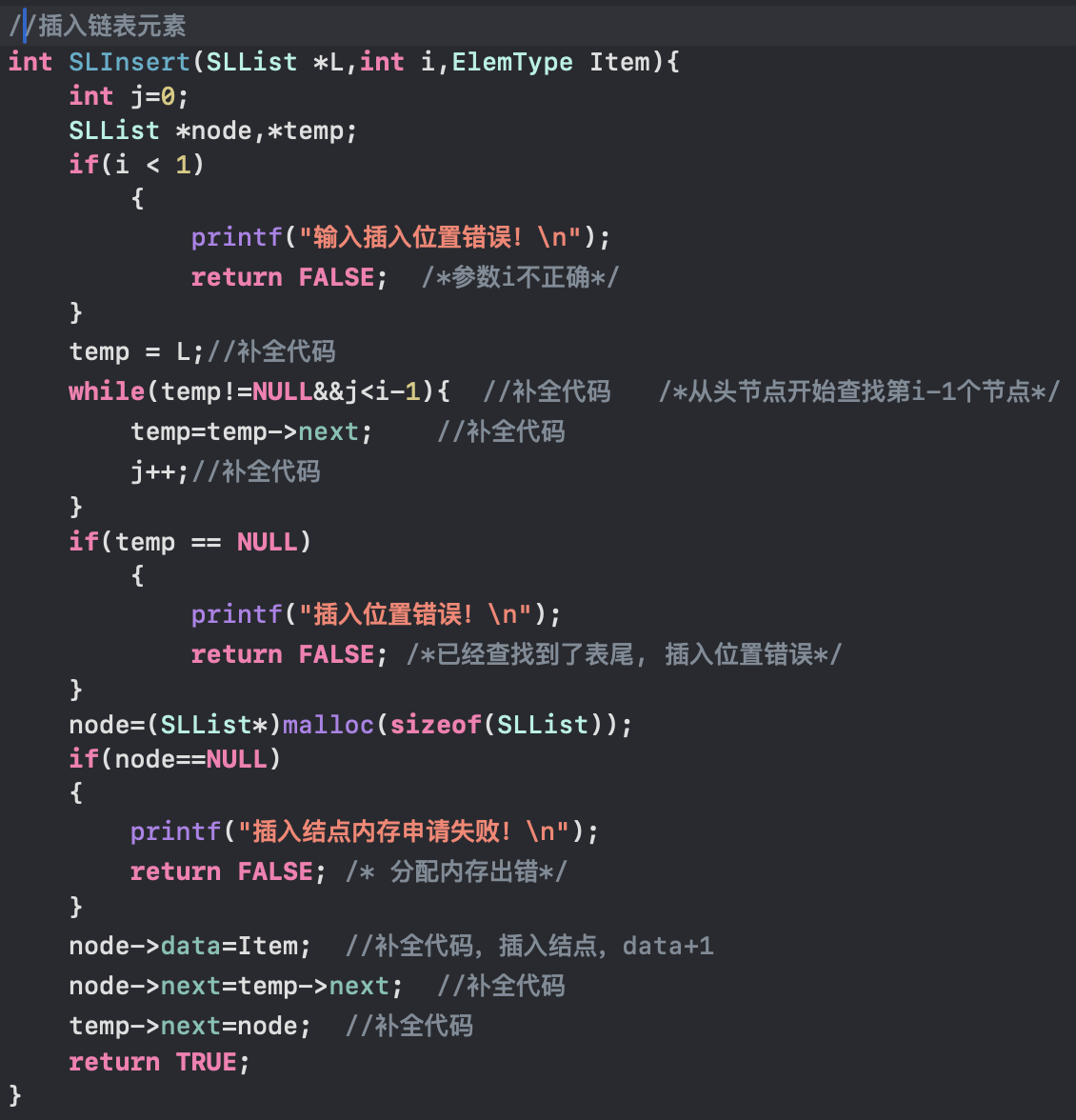
1. 获取链表元素



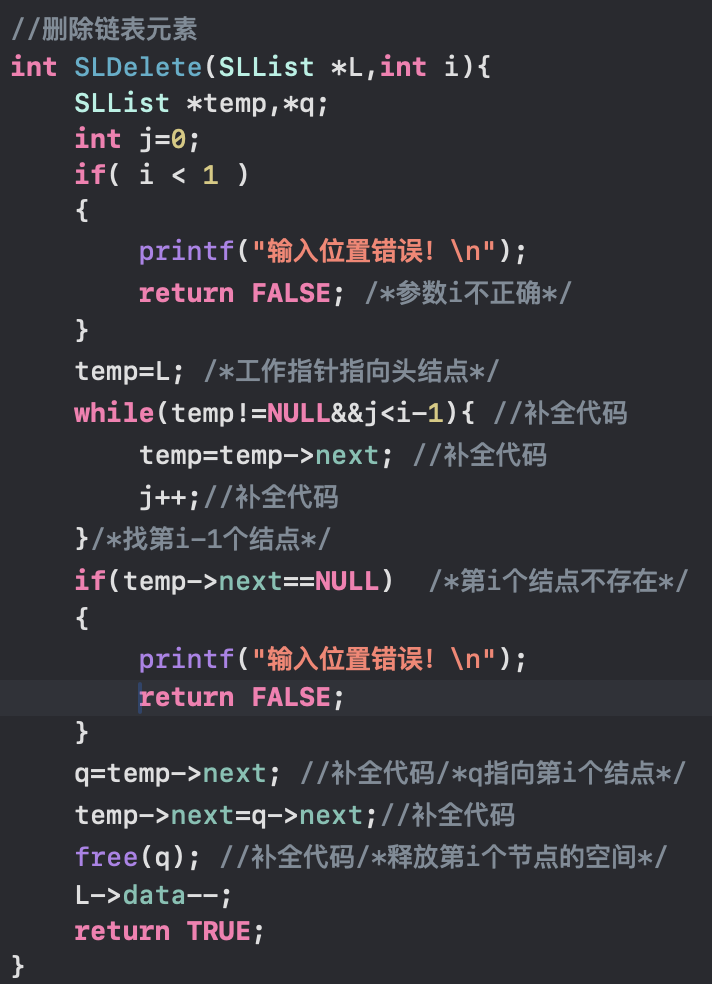
1. 定位链表元素



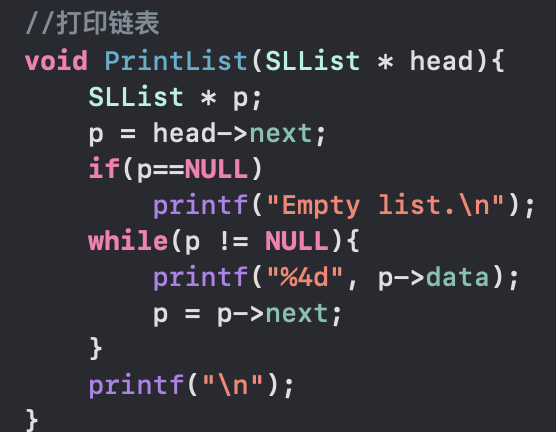
1. 插入链表元素



1. 删除链表元素



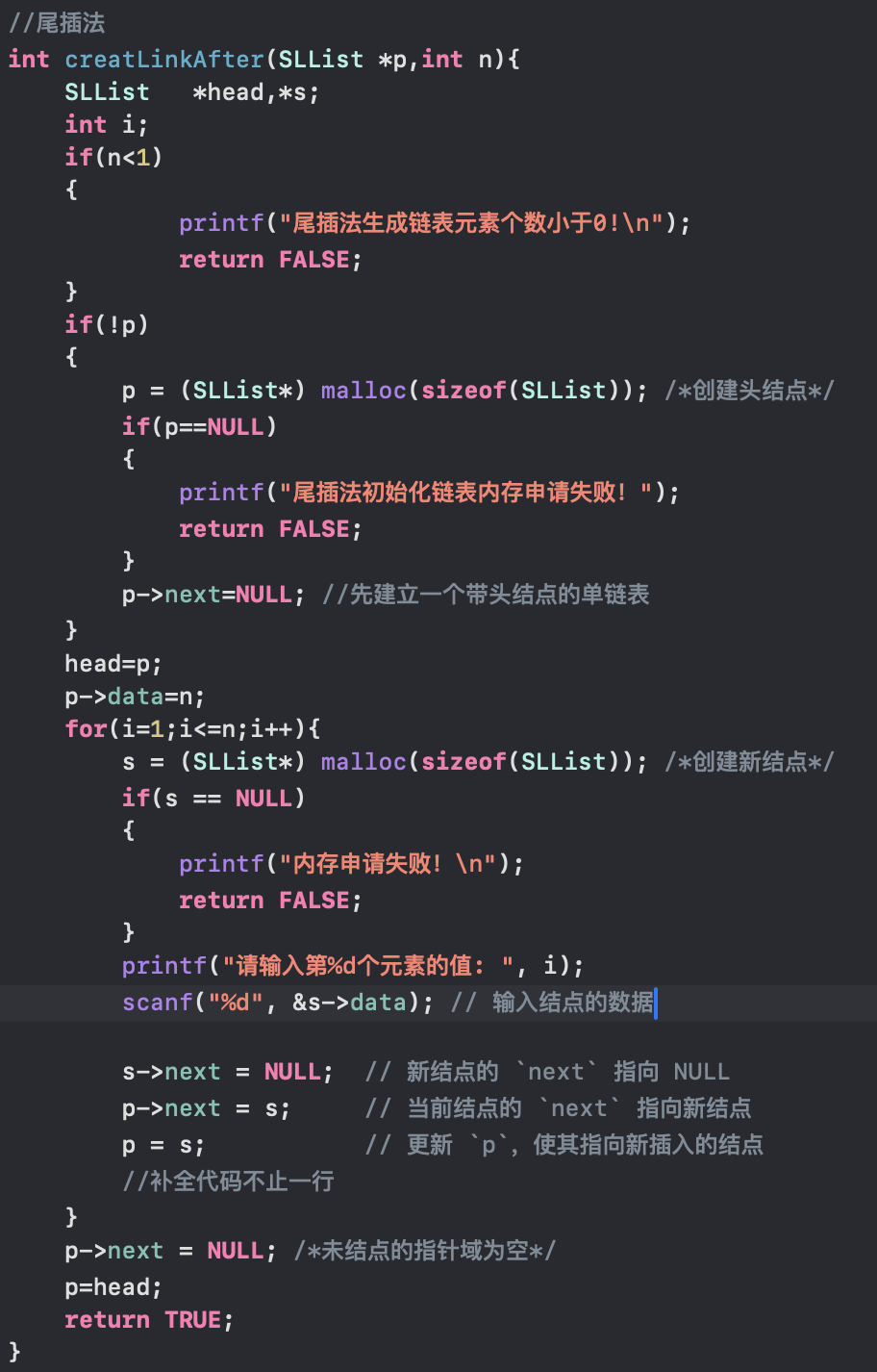
1. 打印链表



1. 头插法

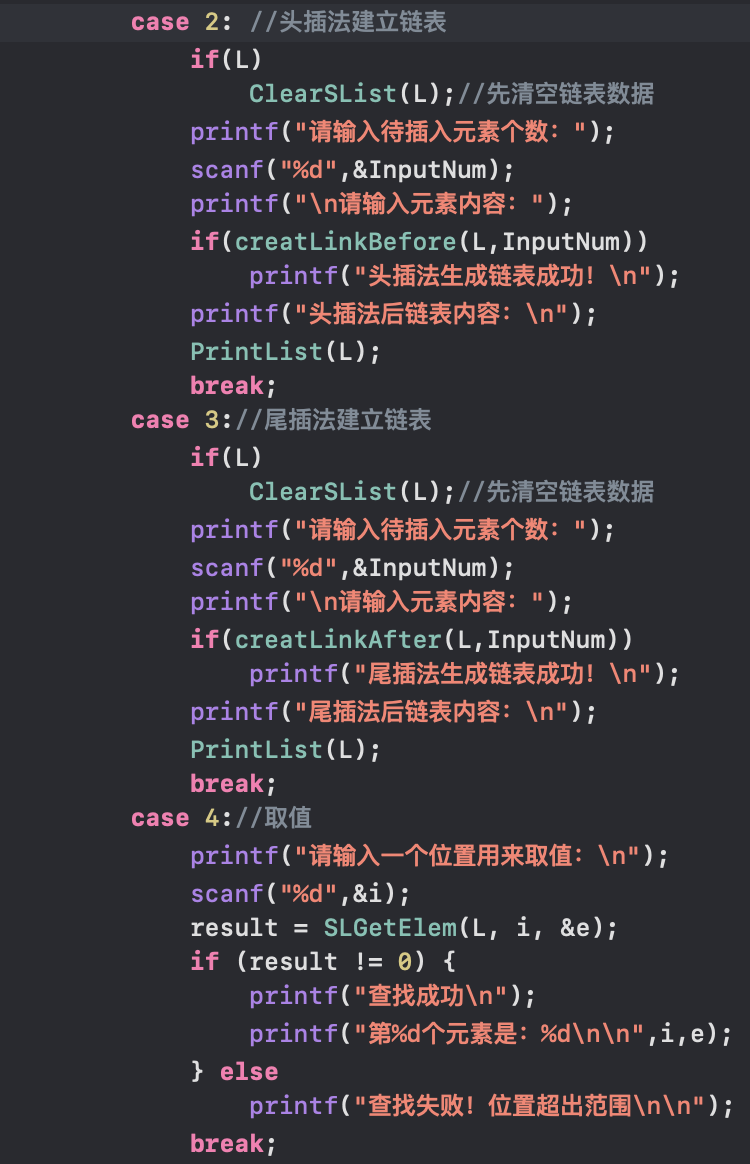


1. 尾插法



1. 测试链表



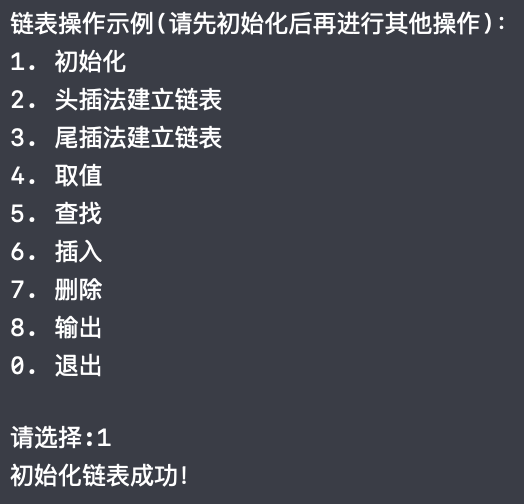




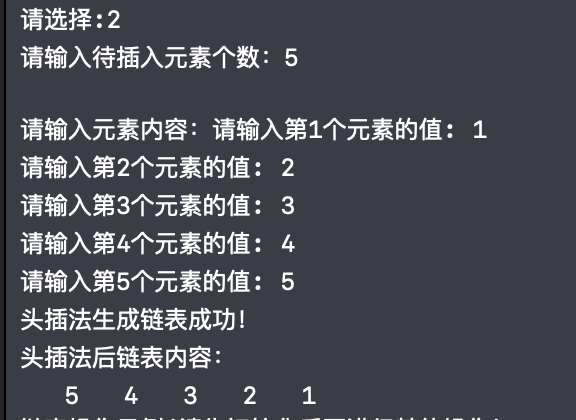


## 测试

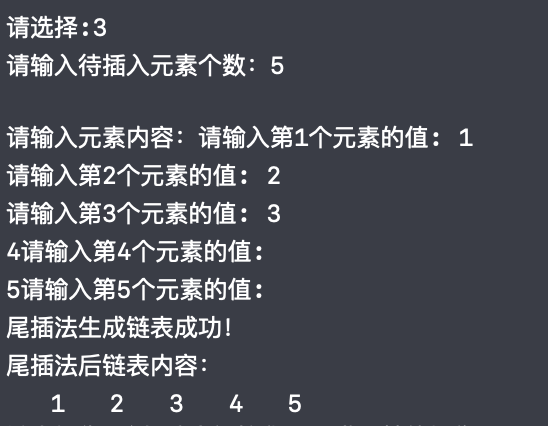
1. 初始化



1. 头插法



1. 尾插法

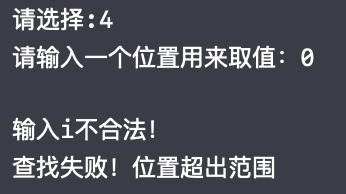


1. 取值

正常取值

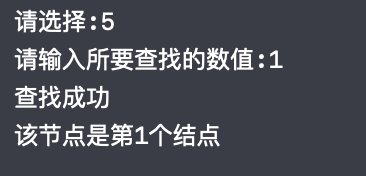


异常取值

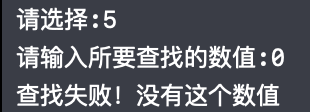


1. 查找

正常查找

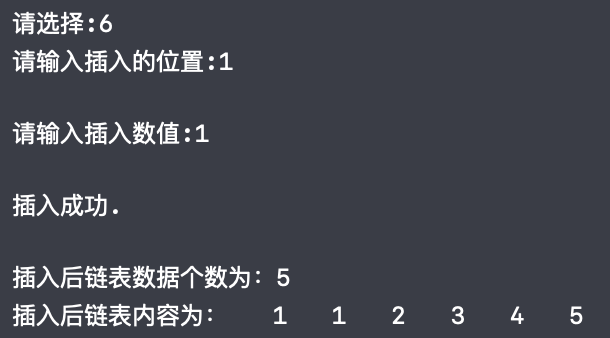


异常查找

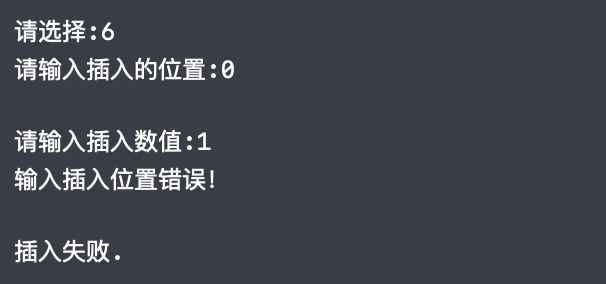


1. 插入

正常插入

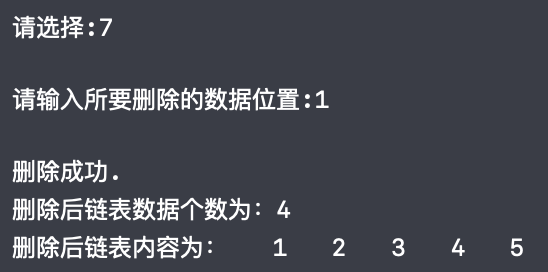


异常插入

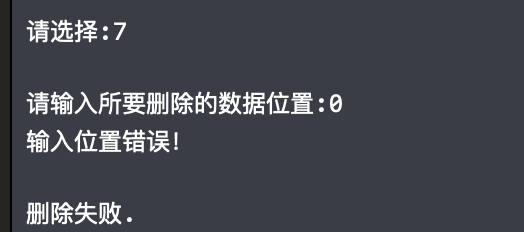


1. 删除

正常删除



异常删除



1. 输出



# 选题：题目1

* 1. 题目描述

编写算法，将两个非递减顺序表L1和L2合并到顺序表L中，使得合并后的顺序表L仍然保持非递减特性，删除重复元素。

* 1. 问题分析

（1）程序的功能要求

* 输入两个已经排序好的顺序表 L1 和 L2。
* 将它们合并为一个顺序表 L，合并后 L 中不包含重复元素，并且保持非递减顺序。
* 输出合并后的顺序表 L。

（2）程序的界面设计

* 用户无需输入，程序通过预定义的顺序表 L1 和 L2 来进行合并操作。
* 合并后的顺序表在终端输出显示。

（3）程序的错误处理

• 检查输入的数据是否符合要求。

• 如果动态内存分配失败，提示用户并终止程序。

• 检查两个顺序表的合法性，确保它们已经排序并且可以合并。

* 1. 方案设计

1. 数据类型设计

使用顺序表为数据结构

每个顺序表由动态数组来存储元素，通过length来记录表的长度

定义INITSIZE用于初始化顺序表的初始大小

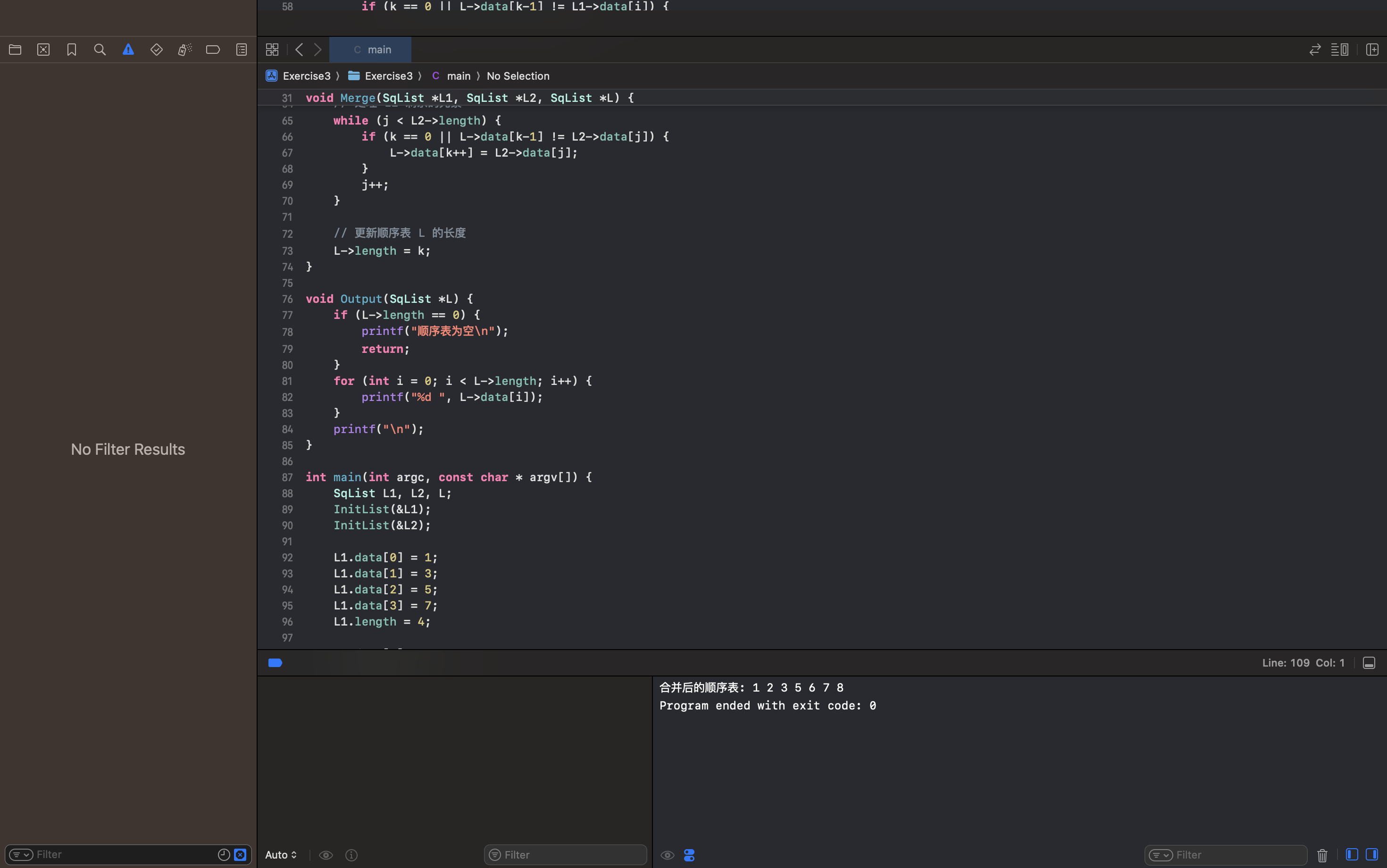
1. 算法设计（算法的基本思想、具体步骤、流程图等）

算法的基本思想:

* 创建两个有序表L1和L2
* 便利L1和L2，根据当前元素大小，插入顺序表L中，跳过重复的元素
* 确保L中没重复的

具体步骤；

* 初始化顺序表 L。
* 使用两个指针分别指向 L1 和 L2 的当前元素。
* 比较 L1 和 L2 的当前元素，将较小的元素插入到 L 中。如果相等则跳过重复元素。
* 当任一表遍历完后，将剩余元素插入到 L 中。
* 输出合并后的顺序表。
  1. 测试分析



输入L1: [1, 3, 5, 7]，L2: [2, 3, 6, 8]，合并结果应为 [1, 2, 3, 5, 6, 7, 8]。

边界条件：

* L1 和 L2 都为空表，合并结果应为空表。
* L1 和 L2 都只有一个元素，且相同，如 L1: [3]，L2: [3]，合并结果应为 [3]。
* L1 和 L2 没有重复元素，且所有元素均为递增，合并后的顺序表应直接包含所有元素。

遇到的问题

* 在处理剩余元素时，由于循环嵌套，导致未能正确处理。解决方案是将处理剩余元素的逻辑拆分出两个独立的循环。
* 动态内存分配出现问题，导致合并后的顺序表无法正确存储数据。通过重新检查内存分配函数，并加入错误处理逻辑解决。



* 1. 心得

在这次实验中，我理解了如何设计和操作顺序表，并成功实现了两个有序顺序表的合并。特别是在去重的过程中，如何确保算法高效，并避免重复元素是一个重要的挑战。

* 1. 附录

1. //
2. // main.c
3. // Exercise3
4. //
5. // Created by RENHONGLOW on 07/10/2024.
6. //
7. #include <stdio.h>
8. #include <stdlib.h>
9. #define ElemType int
10. #define INITSIZE 100
11. **typedef** **struct** {
12. ElemType \*data;
13. **int** length;
14. **int** listsize;
15. } SqList;
16. **int** InitList(SqList \*L) {
17. L->data = (ElemType\*)malloc(**sizeof**(ElemType)\*INITSIZE);
18. **if**(!L->data) {
19. printf("动态内存分配失败\n");
20. **return** 0;
21. }
22. L->length = 0;
23. L->listsize = INITSIZE;
24. **return** 1;
25. }
26. **void** Merge(SqList \*L1, SqList \*L2, SqList \*L) {
27. **int** i = 0, j = 0, k = 0;
28. InitList(L);
30. // 合并两个有序表，并删除重复元素
31. **while** (i < L1->length && j < L2->length) {
32. **if** (L1->data[i] < L2->data[j]) {
33. **if** (k == 0 || L->data[k-1] != L1->data[i]) { // 检查重复
34. L->data[k++] = L1->data[i];
35. }
36. i++;
37. } **else** **if** (L1->data[i] > L2->data[j]) {
38. **if** (k == 0 || L->data[k-1] != L2->data[j]) { // 检查重复
39. L->data[k++] = L2->data[j];
40. }
41. j++;
42. } **else** { // L1->data[i] == L2->data[j]
43. **if** (k == 0 || L->data[k-1] != L1->data[i]) { // 检查重复
44. L->data[k++] = L1->data[i];
45. }
46. i++;
47. j++;
48. }
49. }
50. // 处理 L1 剩余的元素
51. **while** (i < L1->length) {
52. **if** (k == 0 || L->data[k-1] != L1->data[i]) {
53. L->data[k++] = L1->data[i];
54. }
55. i++;
56. }
57. // 处理 L2 剩余的元素
58. **while** (j < L2->length) {
59. **if** (k == 0 || L->data[k-1] != L2->data[j]) {
60. L->data[k++] = L2->data[j];
61. }
62. j++;
63. }
64. // 更新顺序表 L 的长度
65. L->length = k;
66. }
67. **void** Output(SqList \*L) {
68. **if** (L->length == 0) {
69. printf("顺序表为空\n");
70. **return**;
71. }
72. **for** (**int** i = 0; i < L->length; i++) {
73. printf("%d ", L->data[i]);
74. }
75. printf("\n");
76. }
77. **int** main(**int** argc, **const** **char** \* argv[]) {
78. SqList L1, L2, L;
79. InitList(&L1);
80. InitList(&L2);
82. L1.data[0] = 1;
83. L1.data[1] = 3;
84. L1.data[2] = 5;
85. L1.data[3] = 7;
86. L1.length = 4;
88. L2.data[0] = 2;
89. L2.data[1] = 3;
90. L2.data[2] = 6;
91. L2.data[3] = 8;
92. L2.length = 4;
94. Merge(&L1, &L2, &L);
95. printf("合并后的顺序表: ");
96. Output(&L);
97. **return** 0;
98. }