**实验报告三**

1. **实验内容**

线性表顺序存储结构与链式存储结构基本操作+长整型运算

1. **数据结构设计与思路分析**
2. 线性表顺序存储结构与链式存储结构基本操作

首先介绍一下各种存储结构的适用条件与优缺点：

1.顺序表 (Array List)

适用条件:

当需要频繁访问元素时，顺序表提供了快速的随机访问。

当数据量相对固定或变化不大时。

优点:

随机访问效率高：可以通过索引直接访问任何位置的元素。

内存利用率高：顺序表在内存中是连续存放的，不像链表那样需要额外的空间存放指针。

缺点:

插入和删除效率低：在非末尾位置插入或删除元素时，需要移动大量元素。

大小固定或动态调整成本高：静态数组大小固定，动态数组需要重新分配内存和拷贝数据。

2. 单链表 (Singly Linked List)

适用条件:

当需要频繁进行插入和删除操作时。

当数据量经常变化，且没有固定上限时。

优点:

动态大小：可以根据需要动态增长和缩减。

插入和删除效率高：在任何位置插入或删除只需改变指针，无需移动其他元素。

缺点:

随机访问效率低：需要从头开始遍历，无法直接访问特定位置的元素。

额外的内存消耗：每个元素都需要额外的空间存储指向下一个元素的指针。

3. 双链表 (Doubly Linked List)

适用条件:

当需要频繁进行双向遍历或在链表两端进行操作时。

优点:

双向遍历：每个节点都有指向前一个和后一个节点的指针，支持双向遍历。

在任何位置插入和删除效率较高：相比于单链表，删除操作更高效，因为可以直接访问前驱节点。

缺点:

更多的内存消耗：相比单链表，双链表需要额外空间存放另一个指针。

随机访问效率依旧低：仍然需要通过遍历来访问特定元素。

4. 单循环链表 (Singly Circular Linked List)

适用条件:

当需要周期性地遍历数据时。

在表示有循环结构的场景，例如轮转调度。

优点:

循环访问：可以从任何节点开始，遍历整个链表。

简化某些操作：比如轮转，不需要特殊处理链表的尾部。

缺点:

随机访问效率低：和单链表一样，访问特定元素需要遍历。

操作复杂性：在插入和删除时要小心处理循环。

5. 双循环链表 (Doubly Circular Linked List)

适用条件:

当需要双向周期性遍历数据时。

在复杂的环形结构数据处理中，如高级调度或资源管理。

优点:

双向循环访问：可以向前或向后遍历整个链表。

操作灵活：在任何位置插入和删除相对容易。

缺点：

内存消耗较大：相比单循环链表，需要更多的内存空间存放两个指针。

操作复杂性：在维护前驱和后继指针时需要更多的注意。

在实现上述数据结构的基本操作时，设计了一个菜单，用户可以根据自我需求选择要执行的操作。菜单中包含如下函数：

[0] quit\_system()：退出程序

[1] push\_back()：尾部插入

[2] push\_front()：头部插入

[3] show\_list()：打印

[4] pop\_back()：尾部删除

[5] pop\_front()：头部删除

[6] insert\_pos()：在指定位置插入数据

[7] find()：在线性表中查找数据

[8] lenght()：求线性表的长度

[9] delete\_pos()：删除指定位置的元素

[10] delete\_val()：按值删除元素

[11] sort()：排序

[12] resver()：逆置

[13] clear()：清除

[14] destroy()：摧毁

1. 长整型运算

（1）数据结构选择：完成长整数的运算，我们可以考虑如下的数据结构：

①字符串：每个字符代表数字的一位。例如，长整数 12345 可以表示为字符串 "12345"。

②链表：链表的每个节点存储数字的一位。例如，长整数 12345 可以用一个链表来表示，其中头节点是 1，然后是 2，依此类推，最后一个节点是 5。

字符串实现相对简单，而链表实现可以提供更灵活的操作，尤其是在数位长度非常长的情况下。

本次实验采用的是字符串来解决问题。

（2）算法设计与分析

加法

算法设计：

* 从两个长整数的末尾（最低位）开始，逐位相加。
* 记录进位（如果有），并将其加到下一位的计算中。
* 将结果逐位存储在新字符串中。
* 如果最后有进位，将其添加到结果字符串的最前面。

算法分析：

时间复杂度：O(max(n, m))，其中 n 和 m 是两个长整数的长度。

空间复杂度：O(max(n, m))，用于存储结果。

减法

算法设计：

* 从两个长整数的末尾开始，逐位相减。
* 如果发生借位，从高位借来，并进行相应的减法运算。
* 将结果逐位存储在新字符串中。
* 最后可能需要去除结果字符串前面的零。

算法分析：

时间复杂度：O(max(n, m))。

空间复杂度：O(max(n, m))。

乘法

算法设计：

* 对于两个长整数中的每一位相乘，记录乘积和进位。
* 每次乘法的结果需要根据当前乘积的位置，向左偏移相应的位数（就像手工乘法中的对齐）。
* 将所有乘积结果相加得到最终结果。

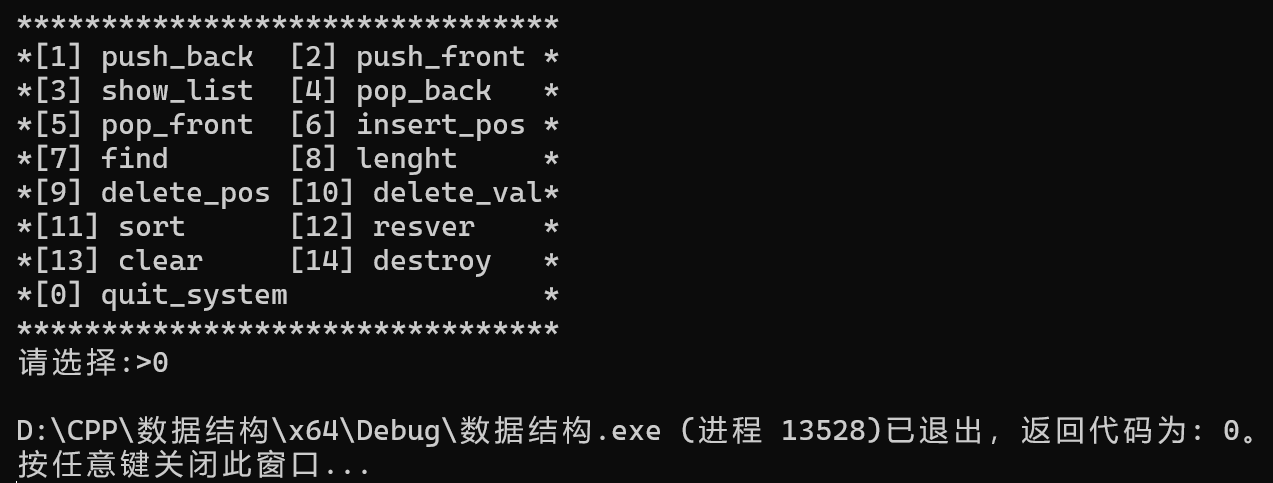
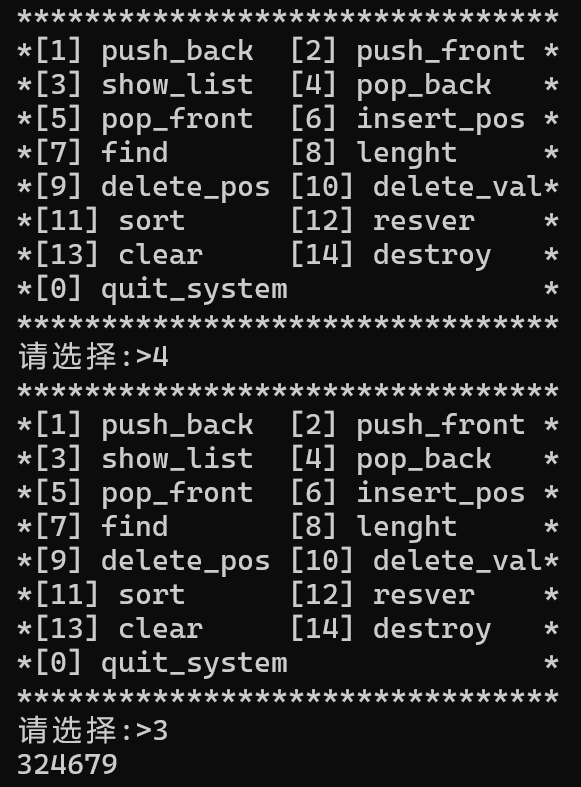
算法分析：

时间复杂度：O(n\*m)，其中 n 和 m 是两个长整数的长度。

空间复杂度：O(n+m)，用于存储结果。

1. **实验结果与分析**
2. 线性表顺序存储结构与链式存储结构基本操作

由于上面提及的几种线性表存储结构均使用的同样的目录，由于篇幅的原因，仅展示其中一份结果截图。其余的经过实验验证均能保证结果的正确性。



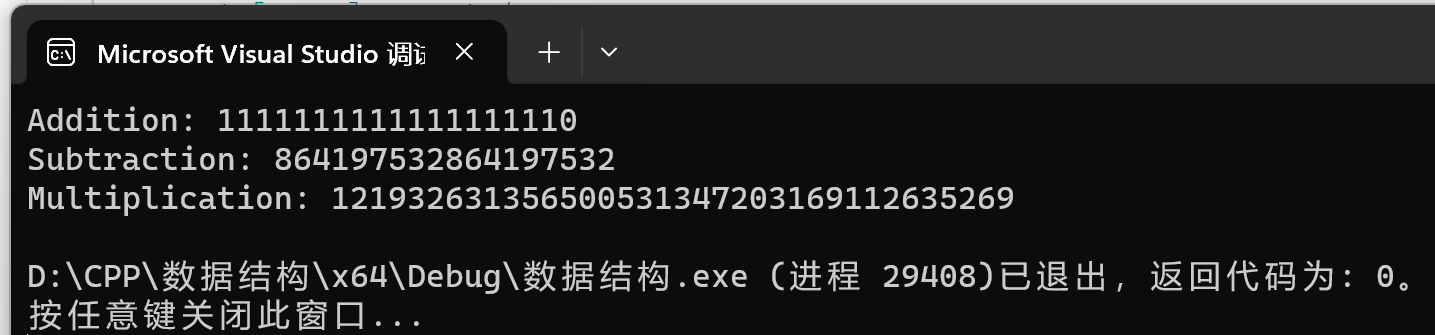
（二）长整型运算

本次测试选取的两个长整数为：

num1 = "123456789123456789"

num2 = "987654321987654321"

所得的结果如下：可以看出，结果具有正确性。

****

1. **实验小结**

通过本次实验，我对线性表各种存储方式的基本操作进行了实践，加深了对知识的理解与掌握。同时运用所学字符串知识解决了长整数的加、减、乘的运算，加深了对字符串操作的理解。