顺序表操作中 SqList 和 SqList* 作为形参的理解

顺序表实现一共讲了五个函数，两个查找，还有插入、删除以及初始化。同学们可能会发现：对于查找操作，是直接把 SqList 类型的变量 L 作为函数参数；而对于插入、删除、初始化操作，却是把 SqList 类型的指针变量作为形参。为什么呢？我们这里通过一个具体的例子进行说明。

首先我们得明确变量使用的规则。C 语言中，函数内定义的变量都是局部变量，只能在本函数范围内使用。A 函数定义了变量 y，B 函数绝对不可以直接去使用 A 中的 y。这就类似于，我们不能跑到别人家拿别人家的东西当自己的来使用。那如果两个函数之间需要进行数据交换，该怎么办呢？很简单，只能通过函数调用。

比如 B 函数想要对 A 函数的变量 y 进行访问与修改，那就只能在 B 中定义一个与 y 同类型的变量作为形参，假设为 by；然后把 A 中 y 的值传递给 B 中对应的该形参 by 就可以了。所谓传递，就是简单赋值：by=y。说通俗一点，就像是 B 中的形参变量抄袭了 A 中的 y 的值一样。所以，by 和 y 依然是各自函数体内的变量，只不过具有相同的值而已。这个值的传递有一个受限的发生时间，就是当 A 调用 B 的时候，才可以进行传值。当 B 函数调用结束，返回到 A 函数的时候，B 中变量 by 的生命就终结了。这是一个非常有限的生命周期！那这个时候，A 函数如果还未执行完，变量 y 依然保持调用 B 函数之前的模样，没有发生任何值的变化。因为你从来没有对 A 中的变量 y 进行过任何一丁点的操作，它的值自然不会发生变化啦！

有了这个基础知识。我们再来理解查找函数 GetElem**(L, i，*e)**中的形参 L，假如主函数中定义了一个 SqList 类型的变量 L1，通过调用 GetElem 函数进行查找，设调用形式是 GetElem(L1,3,e),那么，这里的 L 把 L1 的值全部复制过来了。L 和 L1 就像“克隆的双胞胎”一样，虽然完全一样，但决不是同一个东西哈！

本来的任务是对 L1 进行查找，现在呢？我们是不是忽然意识到，通过这样的函数调用形式，根本就没有对 L1 进行查找的，而是对“克隆双胞胎”L 进行了查找，然后输出 L 上的查找结果。不过没关系，因为结果一样的，所以可以“蒙混过关”,“受点骗”也没关系。这多少有点像曲线救国。

那对于插入删除以及初始化操作，为什么使用 SqList 类型变量就不可以呢？我们一起来看。假定主函数定义了变量 SqList L1，我们想要通过调用 InsertList（L,i,e）函数，完成对顺序表 L1 的插入。函数调用时，L1 的值复制到形参变量 L 中。在函数体内，对这个类似于“克

隆兄弟”的形参变量 L 进行插入操作。注意：我们是对形参 L 进行的操作（编程序以及写算法，确实需要时时保持清醒的头脑，要每一时刻明了你操控的对象是谁？）。当函数调用结束时，L 的生命就终结了。返回到主函数，L1 还是 L1，从未发生过变化。真是竹篮打水一场空，劳而无功！不过也只能接纳这个完全背道而驰的结果了。为什么呢？因为当你在执行 InitList 函数的时候，你从来就没有碰过主函数中的变量 L1，又怎么可能修改它的值呢？

那再看用指针作为形参。形参 L 是一个 SqList 类型的指针，要保存的是顺序表的地址。主函数调用时，对应的实参形式是&L1，把&L1 复制给形参变量 L。现在*L 就是主函数中的变量 L1 了。所以，在 InsertList 函数执行过程中，始终是对主函数中的变量 L1 进行操作，而完全没有对形参 L 进行操作。函数返回时，形参 L 生命终结了。不过一点关系都没有，我们并不关心也不需要保留这个 L 的值，我们只关心主函数中的 L1 是否得到了修改。当然，L1 已经被一个隐藏起来的东西*L“悄悄地”修改了。

指针作为形参就是这样，看起来的确有点“行为越界”。你不让我“光明正大地直接”拿你家的东西用（不让我直接使用你函数体内定义的变量），那我加个*“潜伏”起来，间接使用你家的东西（访问你定义的变量）。不仅访问，还可以“随心所欲”地进行修改、删除，可以让你的变量 L1 完全不同于初始的模样。不过函数调用的初始意图，不就是让 L1 得到修改吗？所以，虽然通过“行为越界”的方式达到目的，但就目标而言却“无可厚非”了！