LinkList L 与 LinkList* 的使用

到目前为止，还有相当一部分同学迷惑于 LinkList L 和 LinkList*L 作为函数参数的使用，分不清二者有什么差异，搞不清何时用何参。下面，将以带头结点的空表的创建为例进行说明。以后若在使用中再遇迷惑，那就可以翻看此处文字，应该可以自我释疑。如果依然在迷惑中犯错，那多次练习之后，也必然会在某一时刻豁然开朗的！

下面是两种创建空表方法的解析。

由于空表只包含一个头结点，因此首先申请一个结点作为头结点，将此结点的指针返回。具体实现时，可以有两种方法。第 1 种方法，定义无参函数，函数类型是 LinkList 类型。在函数体里，申请结点空间，并用 s 指向该结点，然后通过判断 s==NULL，确定是否申请成功，如果成功，则将 Null 赋值给 s->next，并返回指针 s。调用函数时，首先定义 LinkList 的变量 L；然后将函数调用返回值赋值给 L = InitList( )；如此便可以创建空表，其头指针为 L。

方法 1：

```
LinkList     InitList( )
{
      Linklist s=NULL;
      s=(LinkList)malloc(sizeof(LNode));
      if(s == NULL) return NULL;
      s->next=NULL;
      return s;
}
```

调用：LinkList L = InitList( )；

方法 2：

```
status    InitList(LinkList *L)
{
          *L = (LNode*)malloc(sizeof(LNode));
           if(*L == NULL) return 0;
           (*L)->next = NULL;
           return 1;
}
```

调用：LinkList H ; InitList(&H)；

另外一种方法，就是初学者最容易搞晕的方法：通过形参作为输出。形式为：InitList(LinkList *L)。在函数体里，申请结点空间，如果申请失败，则返回 0，否则赋值给 L 所指的目标变量*L，并将(*L)->next = NULL。并返回 1。再来看函数的调用形式。假设主函数调用该函数，则在主函数中先定义 LinkList 变量 H; 然后把&H 作为实参进行函数调用 InitList(&H)；这里的实参是 H 的地址，当进行函数调用时，将实参传递给形参 L（相当于做了一个赋值 L=&H），则形参的 L 便指向 H，那么在函数体里的*L 即是主函数中的变量 H，新申请的结点地址赋值给了 H。这样一来，主函数通过调用函数创建了一个头指针为 H 的空表。

有的同学可能比较疑惑，为什么不这样定义函数呢？InitList(LinkList L)，把参数用一个

LinkList 的变量 L 表示。我们可以看一下。假如现在主函数是调用函数，在主函数中定义了变量 LinkList H，然后将 H 作为实参，进行函数调用 InitList（H）。调用时，实参 H 的值传递给形参 L，在函数体里对 L 进行赋值。函数调用返回时，形参变量 L 释放空间，返回主函数。主函数中 H 变量的值并未发生变化。为什么呢？因为整个过程从来没有触碰过主函数中的 H，我们只是“自以为是”的想要通过“形参 L 的值改变实参 H”，但这在 C 语言中是不会发生的。因此，这样定义参数，并不能够通过调用函数构造一个空表。

需要记住一点：在 C 语言中，想要改变 H，就只能访问 H；如果不能直接访问 H，那就需要通过指针变量间接访问 H。形参的值改变不了实参的值。