这里学习到了一个结论：

当要访问一个类对象中的成员变量时，成员变量的定位是通过两个因素来定位的：①this 指针(根据情况编译器知道是否做调整)；②该成员的偏移值。这种 this 指针的调整、成员偏移值的确定都是需要编译器介入进行的。

现在看一点有意思的事情。在 main 主函数中，增加如下代码(注意代码中的注释)：

```
Base2 * pbase2 = &myobj; //注意结果,myobj 地址为 0x009ffb0c,而 pbase2 给值后是 0x009ffb14,
                         //因为 this 指针调整是往后跳 8 字节,所以这里 myobj 地址值 + 8 字节
                         //后给了 pbase2
```

请想一想，编译器内部肯定是对上面这句代码进行了调整。编译器可能像如下的方式调整的：

```
Base2 *pbase2 = (Base2 *)(  ((char *)&myobj)  +  sizeof(Base)  );
```

在 main 主函数中，增加如下代码：

```
Base * pbase = &myobj;  //pbase 的地址就是 myobj 对象的首地址,因为 this 不需要调整,Base 类
                        //的首地址和 MYACLS 类的首地址相同
```

再看一看下面这段比较有意思的代码，增加在 main 主函数中(仔细阅读代码中的注释)：

```
Base2 * pbase3 = new MYACLS(); //父类指针 new 子类对象,返回的地址是 0x01054bc8,实际这里分
                               //配是 24 字节
MYACLS * psubobj = (MYACLS *)pbase3; //得到的地址是 0x01054bc0,比 pbase3 指向的地址少了 8 字节
delete pbase3; //报异常,据此,可以认为 pbase3 里返回的地址不是分配的首地址,是 this 指针调整
               //后的地址,而 new 分配的真正的内存首地址应该是 psubobj 指向的那个地址
delete psubobj;                        //成功,验证了 psubobj 里才是真正的首地址的说法
```

目前，这个继承结构还不复杂，如图 4.30 所示。

如果出现更复杂的继承结构，例如，Base 类下有一个子类 Base11，MYACLS 类直接继承自 Base11 和 Base2，如图 4.31 所示。

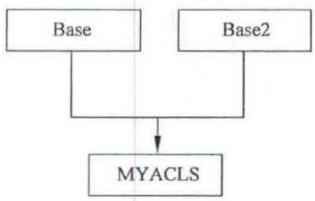

图 4.30　当前的类继承层次关系图

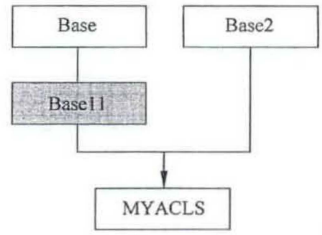

图 4.31　更复杂一点的类继承层次关系图

根据图 4.31，类 MYACLS 对象数据布局很可能是下面这种布局(这里只大概分一下每个子类对象的排列，具体的成员变量等数据就不细分了，留给读者去扩展分析)，如图 4.32 所示。

相信只要掌握笔者所讲述的分析方法，读者也一定能分析出来。希望读者举一反三，根据已经掌握的知识，来学习和推导新知识，这样，对于个人的成长好处是非常巨大的。

另外请注意，虽然目前的继承关系可能比较复杂，但是访问哪个类中的成员变量，会发现成本都差不多，成员位置的计算在编译的时候就已经确定好了，访问成员变量只是做个偏