**单继承和多继承**

C++的继承方式是支持单继承和多继承的，首先看一下代码，分清单继承和多继承

单继承

class A

{

public:

int \_a;

};

class B :public A

{

public:

int \_b;

};

class C : public A

{

public:

int \_c;

};

类似于上面的方式就是单继承，或者C类可以不继承A类，而是直接继承B类，这种方式也是单继承

多继承

**class** D :**public** B,**public** C

{

**public**:

**int** \_d;

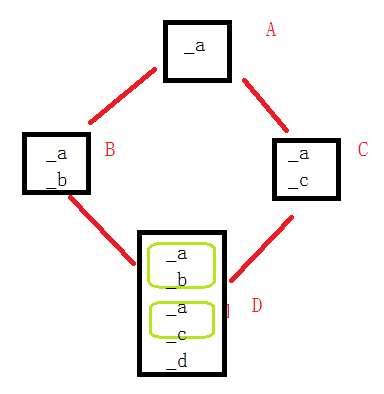
};

上面的这种方式就是多继承，即是D类既继承了B类，又继承了C

通过上一个章节的学习我们知道，继承的时候，在子类的 成员变量里面包含了父类的成员变量，则我们可以分析出，上面的继承 关系中，B类里面包含变量\_a,和变量\_b；C类里面包含了变量\_a和变量\_c；这个时候我们的D类里面就包含了B类的成员变量和C类的成员变量，所以D类里面包含的成员变量是\_a,\_b,\_a,\_c,\_d。

**菱形继承**

基于上面的分析我们引入了菱形继承，下面我们用一个图来表示上面的成员的关系



菱形继承问题 -- 二义性和数据冗余

还是上面的代码，这个时候我们实例化一个D类的对象，这个时候我们想对\_a赋值，就会出现报错，请看下面的代码

**D** **d**;

**d**.\_a = 1;

**return** 0;

这个时候编译器报的错误就是D::\_a不明确，为什么不明确呢，这个从上面的图中可以看出来，因为我们的D类的对象中有有两个\_a的成员变量，这个时候对\_a赋值的时候，编译器不知道该给哪个\_a赋值，这个问题就是\*\*二义性\*\*问题。

首先看下面的解决办法

D d*;*

d.B::\_a = 1*;*

通过这种方式我们就不会出现二义性的问题了。  
还有一个问题就是，我们的D类型的对象中，有两个\_a这个时候的\_a表示的意义都是一样的，如果我们的A类里面的成员变量非常的大，这个时候是不是就是数据冗余呢，我们需要一个A类的成员就够了。

**解决菱形继承 -- 虚拟继承**

为了解决上面的问题，我们引入了虚拟继承的概念，请看下面的代码

**class** A

{

**public**:

**int** \_a;

};

**class** B :**virtual** **public** A

{

**public**:

**int** \_b;

};

**class** C : **virtual** **public** A

{

**public**:

**int** \_c;

};

**class** D :**public** B,**public** C

{

**public**:

**int** \_d;

};

**int** **main**()

{

D d;

d.\_a = 1;

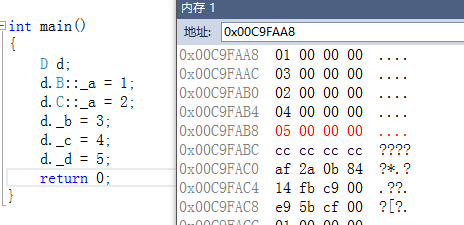
**return** 0;

}

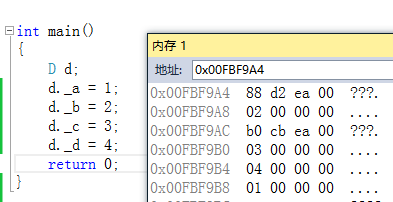
这个时候在主函数中，再次使用上面的方式对\_a赋值的时候，就不会报错了，那么我们不禁想问，编译器到底是如何做的呢，为什么这么做的时候就不会出现问题了呢，这个问题要从C++对象模型说起

**虚继承对象模型**

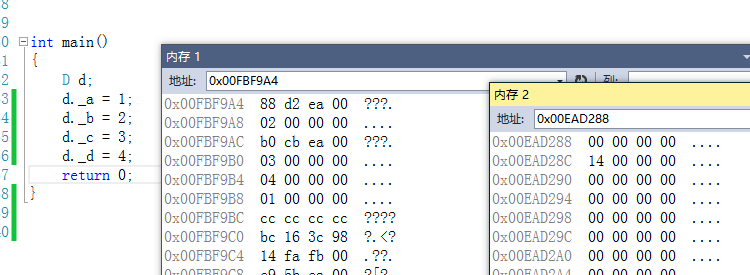
下面是普通的继承，没有虚拟继承，我们结合旁边的代码可以看到，如果没有虚拟继承的时候，D类的对象中会为两个A类型里面的\_a变量，这就是数据冗余和二义性



下面我们再来看一下如果我们使用的是虚拟继承给赋值的给赋值的时候是什么结果呢，看下面的截图



这个时候我们注意到一个问题是，和刚刚的那个截图不一样的是，在变量\_b的上面放置的不是一个变量\_a,而是一个类似于一个地址一样的数据，同样的道理，在变量\_c的上面放置的也不是变量\_a,也是一个地址一样的内容，我们又来发现一个问题就是，我们给变量\_a，赋值的地方显示的是在变量\_d的下面，然后我们再来看看那个类似于地址一样的内容到底是什么呢

我们把这个类似地址的数据放置放在监视窗口中看一下 

这个时候我们看到，监视窗口2里面放置的内存的位置放置的十六进制数是14，转化成十进制就是20，我们再来比较一下监视窗口1中，那个地址和\_a实际存放的位置的距离是20，这个时候我们就能够理解了，原来这里放置的是\_a这个变量的偏移量。

还有一个疑问就是，为什么监视窗口2中对应的地址上面放置的是0，而不是14呢，这是因为，这个0所在位置是为了给接下来的虚函数准备的，这个以后会讨论到。

这里需要说明的是，监视窗口1中放置的是地址是虚基表指针，它指向的是一个虚基表。

我们还需要考虑的一个问题就是，为什么这里不直接放置\_a的地址，而是放置的是相对位置的偏移量呢，这里我们分析这样的一个问题，如果我们拿这个类去实例化多个对象的话，是不是每个对象就有一个存放\_a的地址呢，那么这个时候，每个对象的放置的地址都一样了，而如果放置的是偏移量的话，每个对象的地址空间中放置的内容都是一样的，这就减少了计算的开销。