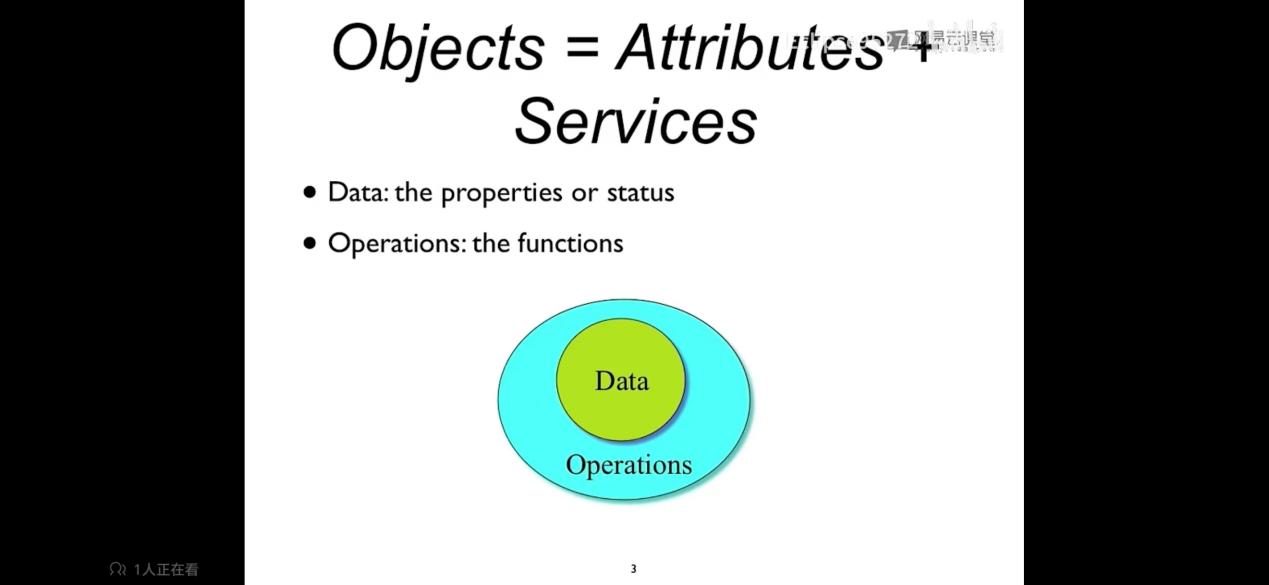
**C++编程学习经验**

**<………………………………………………………………>**

**面向对象的概念**

**Object（对象）：**

对象（Object）= 属性（Attributes）+ 服务（Services）



一个对象，它有内部的数据（Data）和像是包裹这数据的操作（Operations），这个对象可以利用它自身的数据去执行这些操作，完成一些事情。

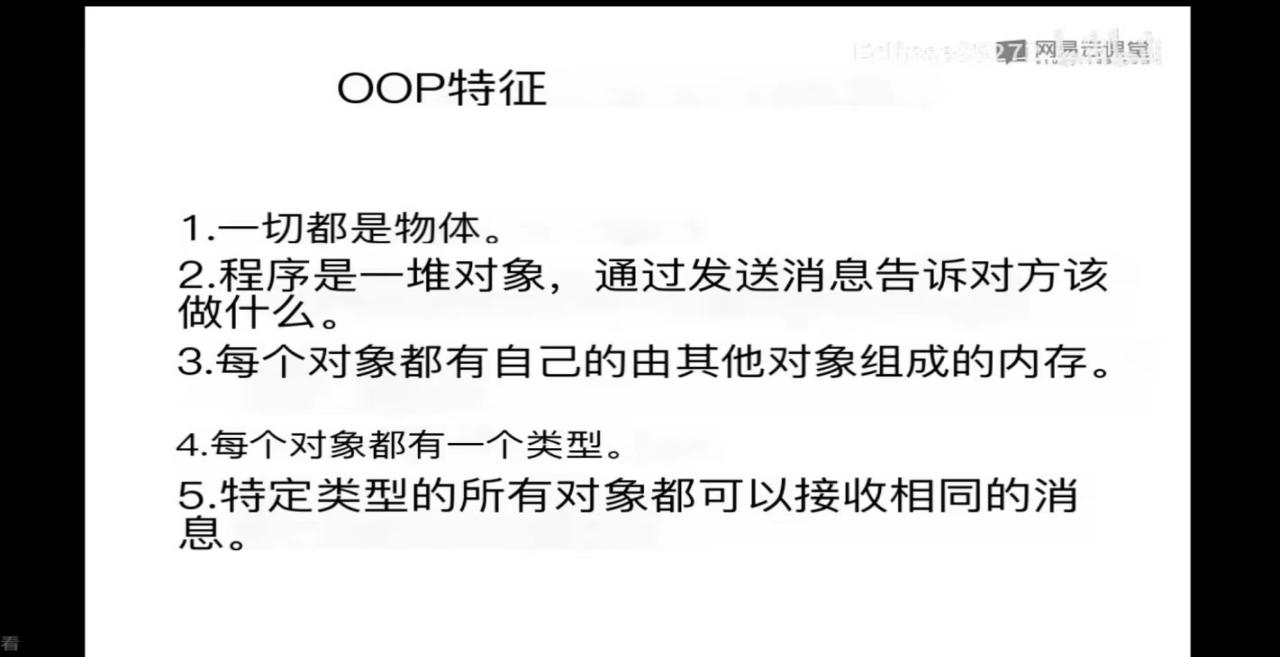
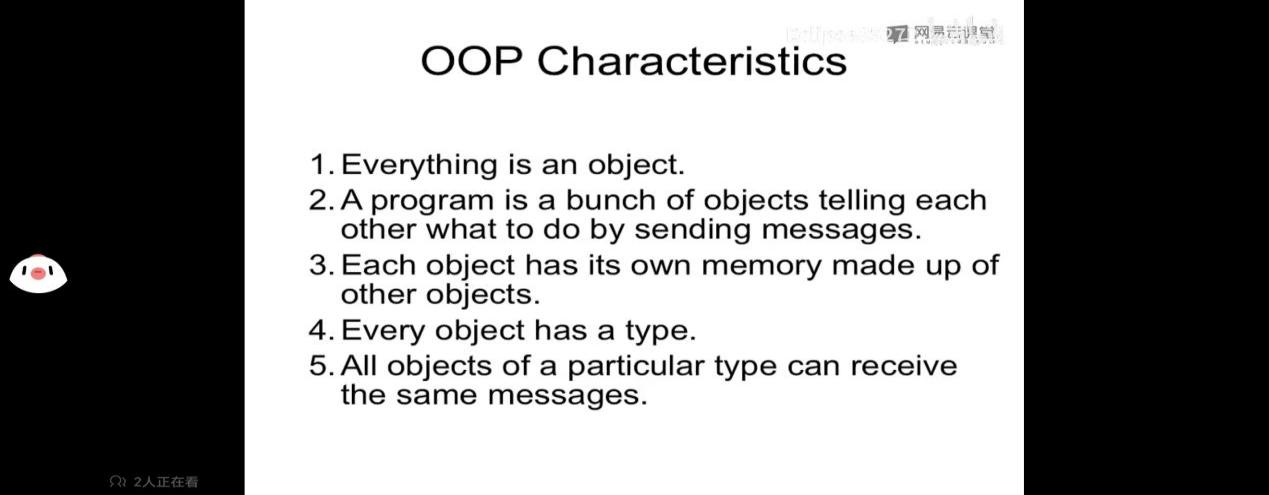
简单理解就是你可以想象是一个机器人一样，你创建了一个对象就是创建了一个小人儿，这个人儿可以给你做事情，帮你完成数据的使用、功能的实现。

**Class（类）和Object（对象）的关系：**

个人理解：**class是object的抽象表达**，通过class我们可以了解到object的大致样子，而**object则是class的具体表达**。二者好比一个杯子（Class）和一个蓝色的保温茶杯（Object），

即，Class告诉我们有这么一类东西它是杯子，它有什么样的形状、什么样的功能、等等，然而Object就是说很确定的一个杯子，它是蓝色的保温茶杯，这个杯子是一个很具体的事物，**所以说Object是Class的具体实现**。

**OOP（面向对象编程）：**



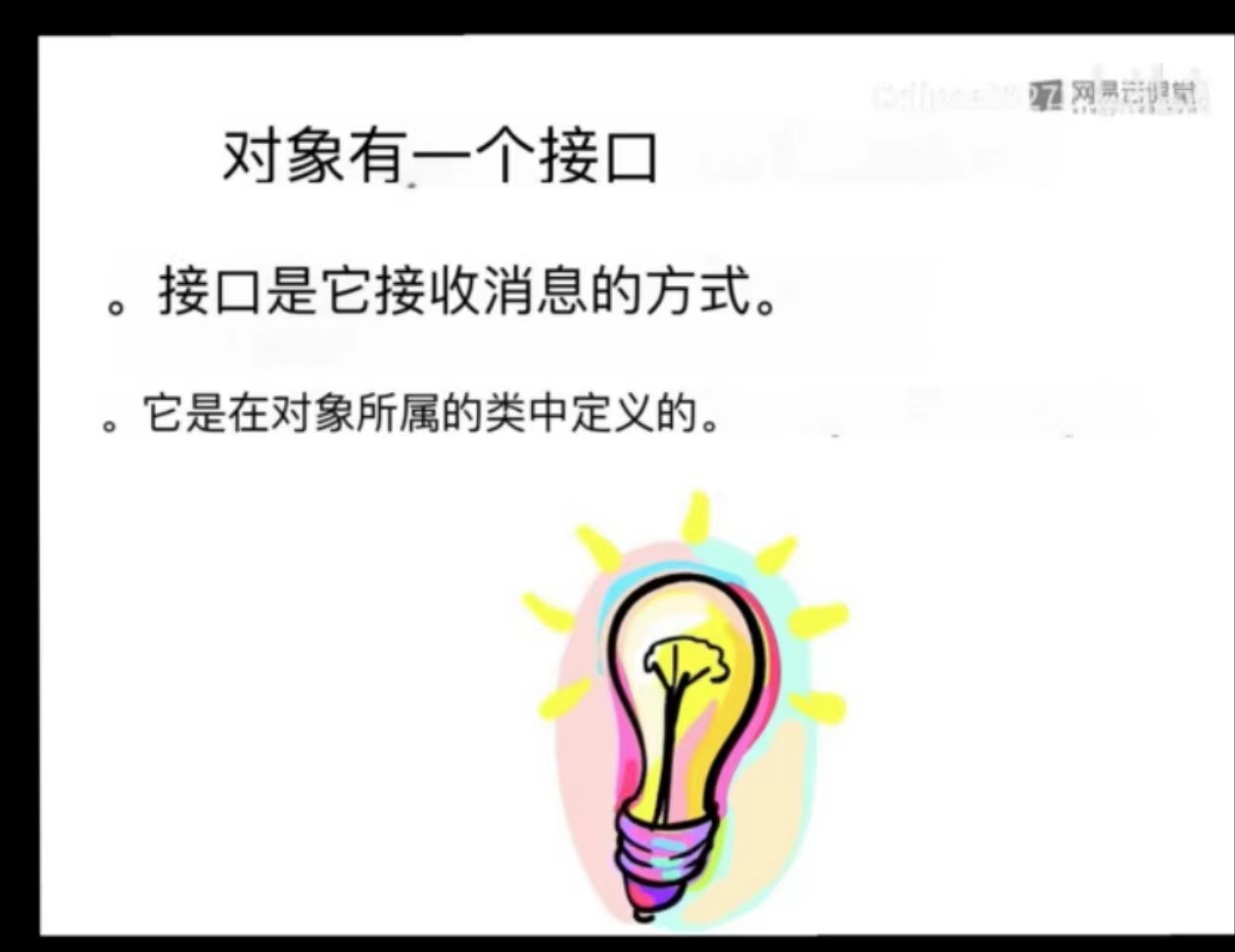
**（第二张图是第一张图的中文翻译）**

上图第二条，“告诉对方该做什么”仅仅只是告诉一个对象它应该做什么，而不是应该“怎么做”，也就是说一个对象只负责告诉另一个对象应该做什么，至于另一个对象是否开始做以及怎么去做，我们不知道或是不关心。

就像是人与人之间的交流一样，我们不可能总要告诉别人办事每一步应该怎么样做，而只是告诉别人你应该去办一个什么事，至于这个人去不去办这个事以及他会怎样解决这个事情，我们个人实际上是不关心也不在意的。

上图第五条，告诉我们两个相对的事情，**“特定类型的所有对象都可以接收相同的消息，所有可以接收相同消息的对象可以被认为是相同的类型”**打比方解释就是：所有的杯子对象，它们都能装水；反之，所有可以装水的对象，我们也可以认为它们是同一个类型，也就是杯子，再比如苹果和梨，它们都咬一口出水，所以就是水果类型。

**对象的接口：（高内聚，低耦合）**



接口的好处就是，我们可以随时切换传递给接口的东西。

**C++中的编译和链接**

我目前不想分析太多这部分的内容，因为我自身也不太了解其内部深层次的工作原理，但为了后面内容的讲解我不得不说一些知识作为理解后面讲解的“前置插件”。

**编译：**

我们知道当我们写完代码之后，如果我们需要运行我们自己的程序，那么我们首先需要用编译器去编译它，有时甚至是多文件编程，那么就会编译多个文件。

在编译器编译我们的代码文件时，我们的每一个文件都被编译为一个名叫**“翻译单元”**的东西，这些东西将被编译器组合起来进而实现我们代码的功能

（现在理解到这里就行，而且我也不保证上面的内容完全正确，但它至少会让你更加理解后面所讲解的内容）

**链接：**

链接就是把上面的翻译单元链接组合到一起，如果翻译单元自身或它们之间有冲突，链接就会出错，编译器也会报错。

**静态（static）相关内容详解**

基于static作用的不同地方我们有两种分类：

1. static在类或结构体范围之外；
2. static在类或结构体范围之内；
3. static在函数范围之内。

**1、static在类或结构体范围之外：**

在类或结构体之外使用static的话，我们一般就是定义**静态变量**或者是**静态函数**。

通常定义变量或函数我们大概就是int、void等关键字后面加我们要定义的东西，比如我们可以写**int a;**表示我们定义了一个整型变量。当我们在int之前加上static之后也就是变成**static int a;**它就表示我们定义的这个变量a将会只作用在它被写在的那个文件中，也就是说，当这个代码文件被编译为翻译单元的时候，**变量a只能作用在自身所处的翻译单元之中**，不能被链接到其他翻译单元上，不能与其他翻译单元互动。（静态函数同理）

**也就是说：**

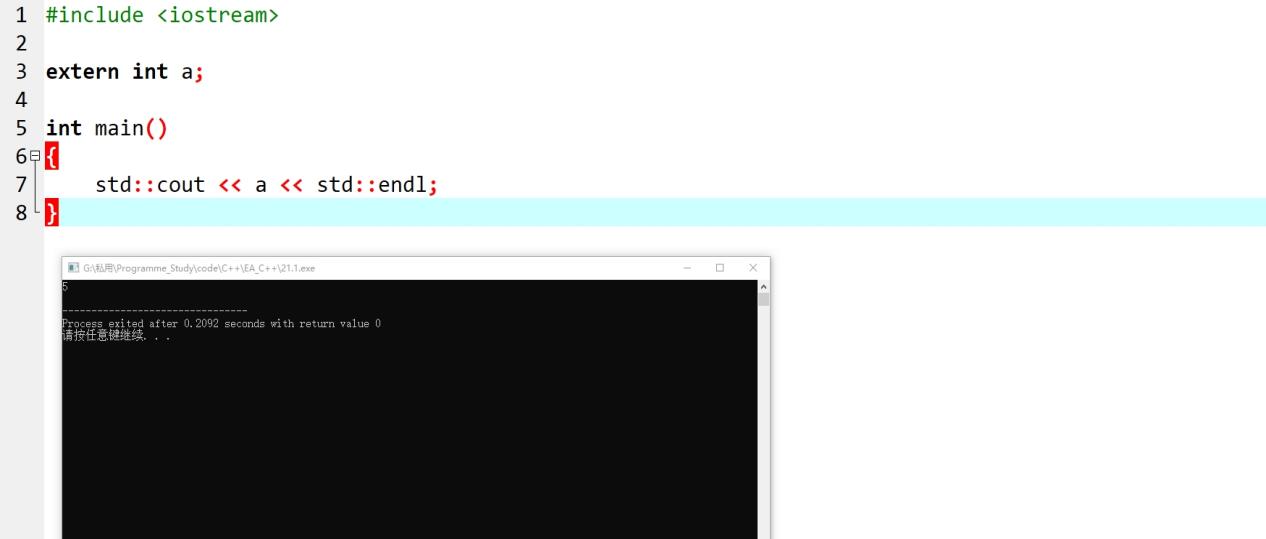
**如果**我们在另一个文件中写了**extern int a;**即在另一个文件中定义变量a，当编译器编译链接时他会在所有翻译单元中去寻找a的值，那么如果我们在本文件中写的是**int a;那么写有extern int a;的文件就可以找到本文件的a的值并将该值用于写有extern int a;的那个文件中去。**

**如果**我们在本文件中写的是**static int a;**那么当写有**extern int a;的文件去其他翻译单元寻找a的值的时候，由于我们本文件的a是静态变量，它只能作用于该翻译单元自身，因此不会被写有extern int a;的文件所找到，那么写有extern int a;的文件也就不能使用本问文件的a的值了。**

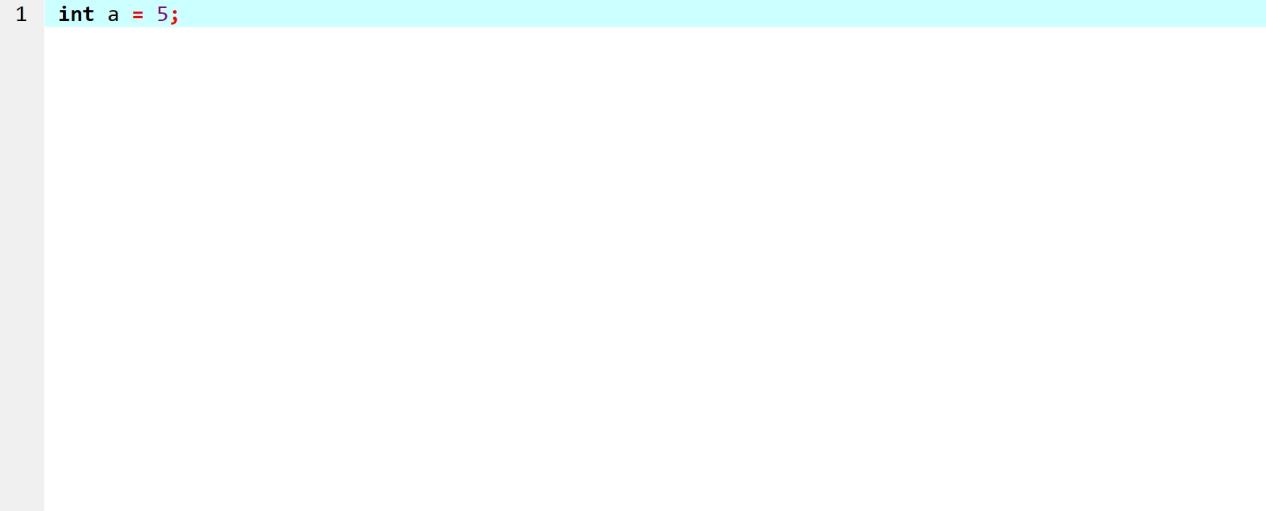
这听起来static的作用与类（class）中的private有些许相似，他们都限制了变量或函数可以发挥作用的空间，**private将变量设置为只能在类（class）内部使用，而static也是类似的将变量或函数设置为只能在本文件的代码中使用**。

具体代码请看下图：

文件1：

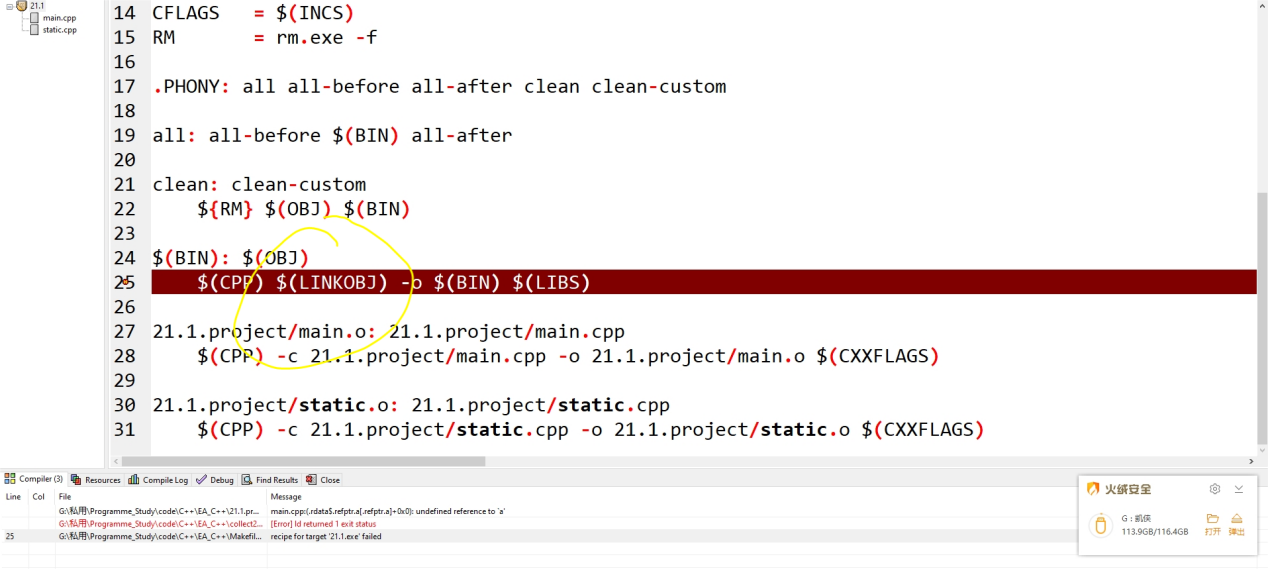


文件2：

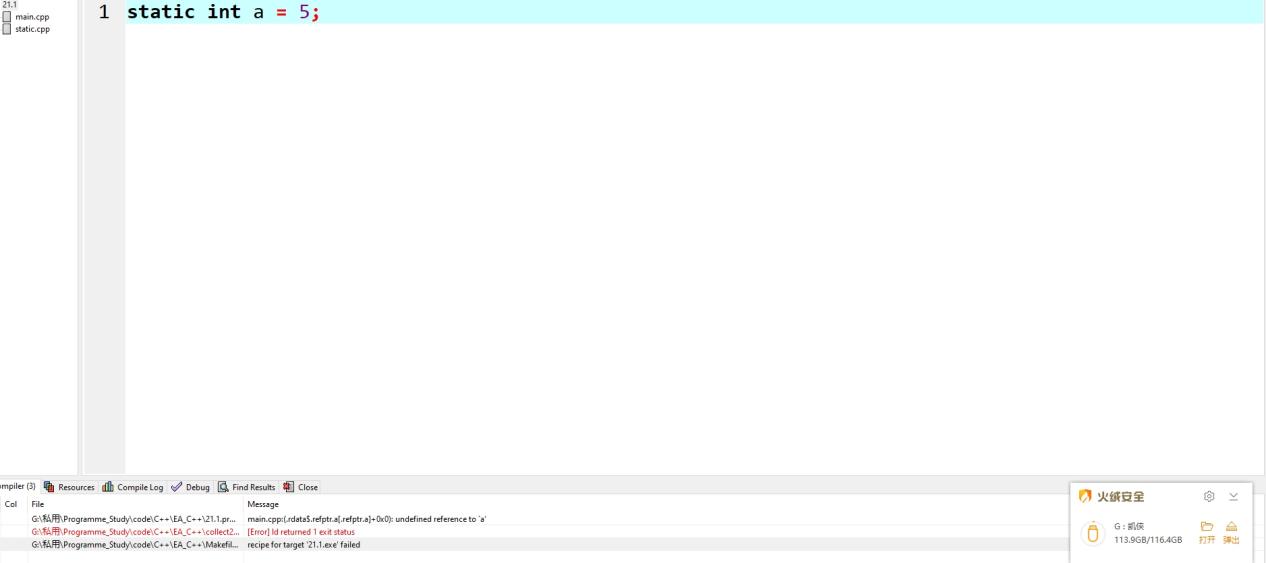


如上图所示，文件1中的**extern int a;**在编译时找到了文件2中的a的值，并在文件1中输出了a的值到屏幕上

加入static之后的代码：



文件2：



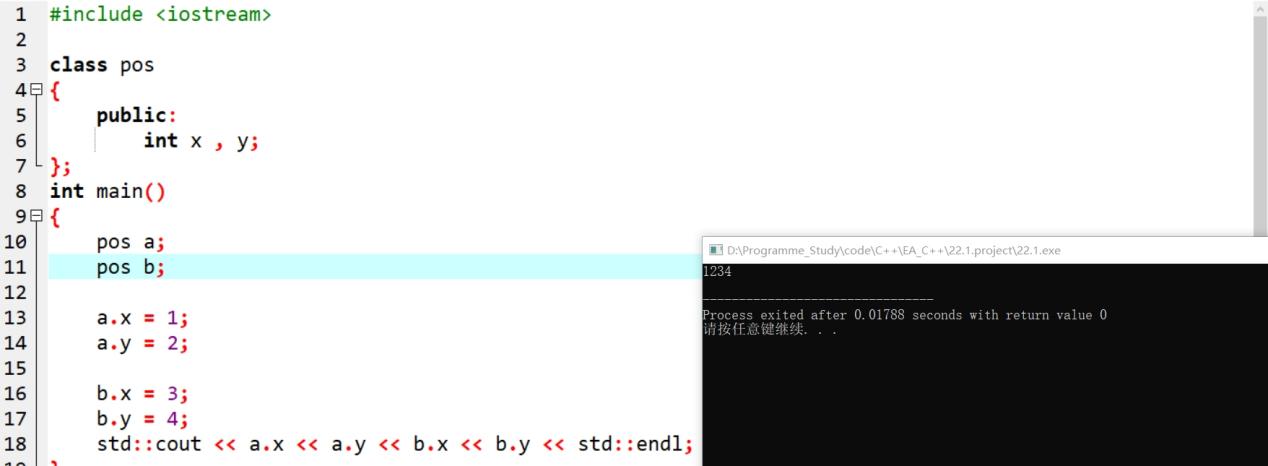
文件1不变，还是同一个文件2，只不过这次我们不再是**int a;**而是**static int a;**这次当我们编译我们的代码时，编译器出现了报错。我们可以清晰的看到第一张图圈出的部分，显示了一个LINKOBJ，它表示链接对象（link object）出错，也就是**static int a;**的a只能作用在文件2中，当我们寻找a的值时，文件2的a的值对于文件1乃至其他文件都是不可见的，所以会报错，因为找不到a这个对象。

**2、static在类或结构体范围之内：**

对变量和函数使用static其所产生的功能其实还是有区别的，下面我将逐一讲解。

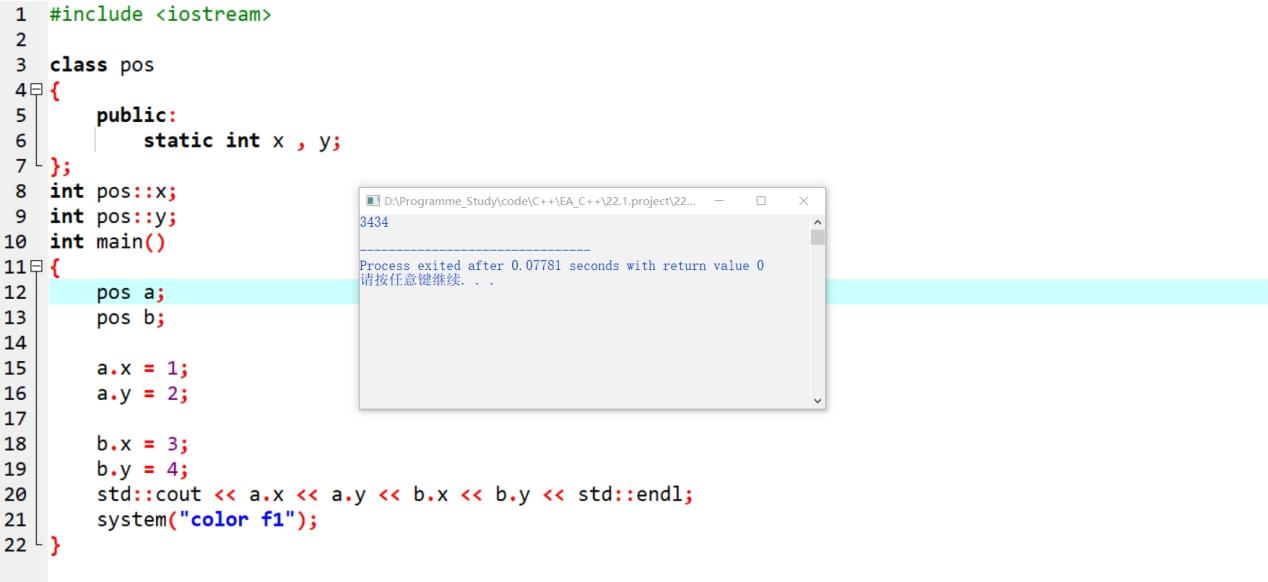
**（1）对变量：**

在一个类或结构体（下面统称为类）中对**public**的变量的定义（诸如**int a;**）使用static，会增加变量a的作用域到整个类对象。通俗的讲就是，假如我们有下图两个类的实例



在我们**没有**对**类对象**（pos）中**public**的**int x , y;**使用**static**时我们在main函数中可以随意使用pos的各个**实例**（a和b），而且值得我么你注意的是，实例a和实例b它们自身的x值和y值互不影响，它们的数据存在不同的地方，因而我们打印出的信息就是1234。

但是一旦我们对**类对象**（pos）中**public**的**int x , y;**使用**static**那么情况会大不相同，上面说到我们会增加变量的作用域。static相当于将变量x和y变为了类对象（pos）这一整个对象下的变量，**对于pos所定义的实例a和b它们共用x和y。**



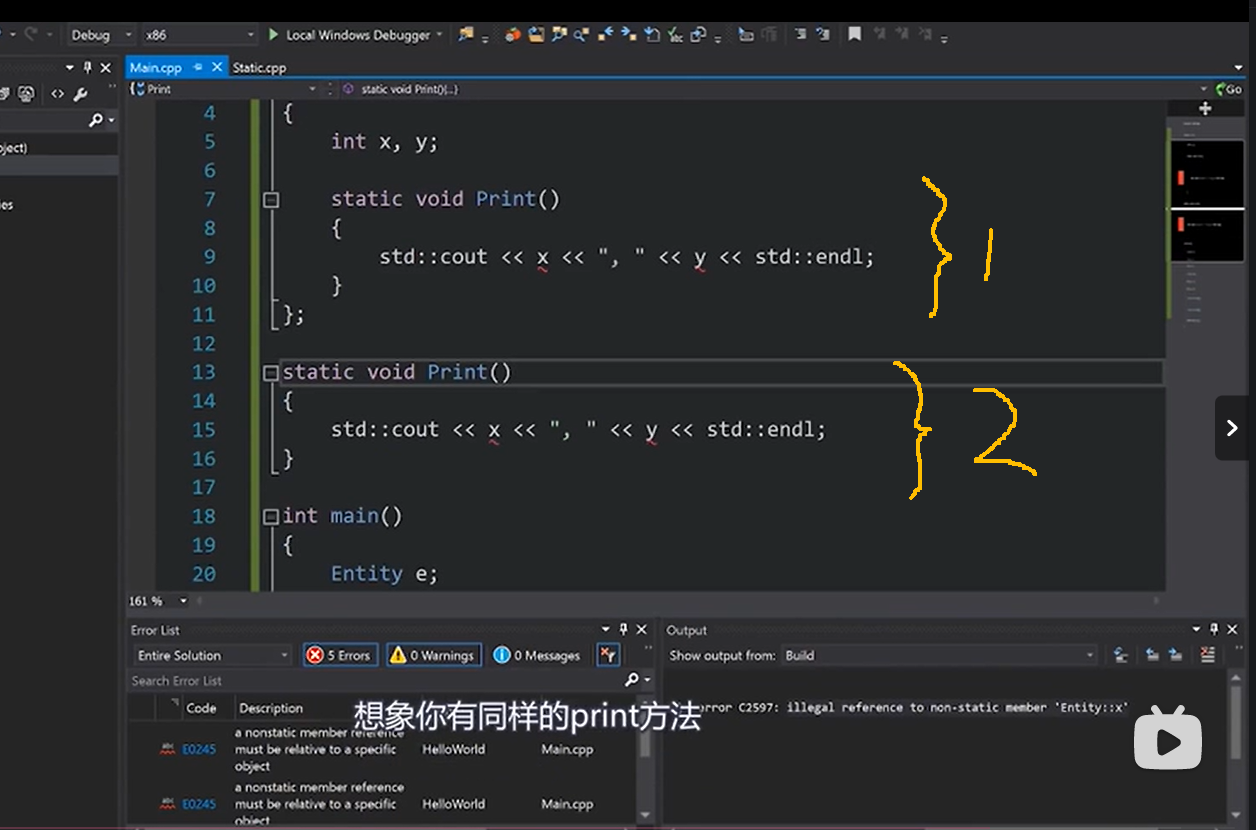
ok这里我们可以清晰的看到在main函数中，我们虽然用a给x和y赋值了1、2，但是实际打印时并没有显示12，而是3434，**这就是因为a和b公用x和y**，所以在后面我们用b为x和y重新赋值为3、4的时候，实际上这个时候**a.x**就是3，**a.y**就是4。  
 **注意第8、9行代码，我们static int x , y;之后便需要像这样声明出来在pos作用域里的x和y，只有这样我们才可以正常使用x和y。**

**当然我们也可以使用pos::x和pos::y对它们进行赋值或其他操作。**

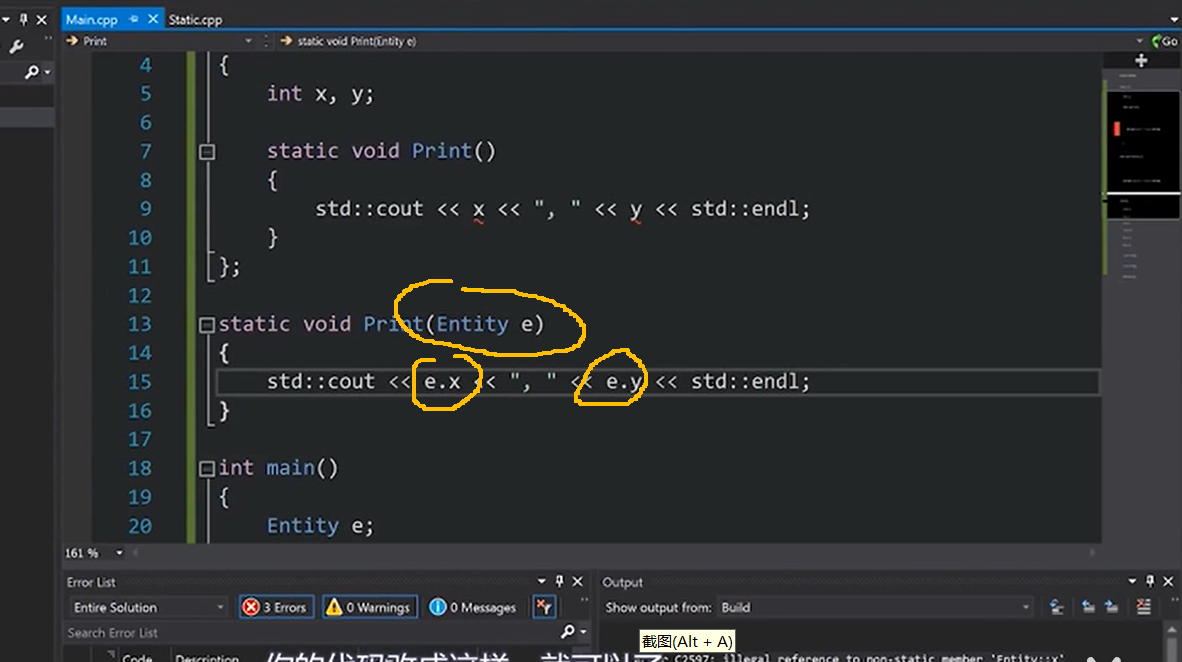
1. **对函数(方法)：**

首先我们需要熟悉一个概念，那就是在类中写的自定义函数实际上就是C++中所说的**方法**，即:**类中的函数-->方法。**

对方法使用static就是静态方法,**静态方法没有类实例,静态方法只能访问静态变量,不能访问非静态变量。**

****

上图第一部分的**静态方法**写在类的里面,它就**相当于**是在外面写了一个同样的**静态函数**,这就是为什么当我们使用类中的静态方法时会出错,因为他就等于你把函数写在了外面,它们并不知道函数里写的这些x和y都表示什么意思,我们只有这样写(见下图)才能真正的将Entity类传入函数中,才能进而使用x和y。



当然我们也可以像使用静态变量那样用作用域名称去使用静态方法,比如Entity::Printf()。

**3、static在函数范围之内:**

在函数之内对变量使用static的作用是延长变量的**生存期**,我们知道通常情况下我们在函数中定义的**变量会在函数开始时生成,函数结束时就会消失**,但是在定义变量前面加上static之后,该变量会在我们**第一次调用这个函数时生成,函数结束时不会再消失**,也就是说,它可以伴随着函数的使用而变化.具体可以看下图:



当我们没有对**int i = 0;**使用**static**时,我们再**main**函数中连续调用了三次**function()**函数,其输出结果都为1.这不需要过多解释。



但是当我们使用了**static**之后,伴随着**function()**函数的三次使用,**i的值却是递增的状态**,这就是上文所说的我们用**static延长了变量i的生存期**,使得它可以在被第一次函数调用生成**i**之后不消失,它的值仍然是你上一次调用函数结束时的值。