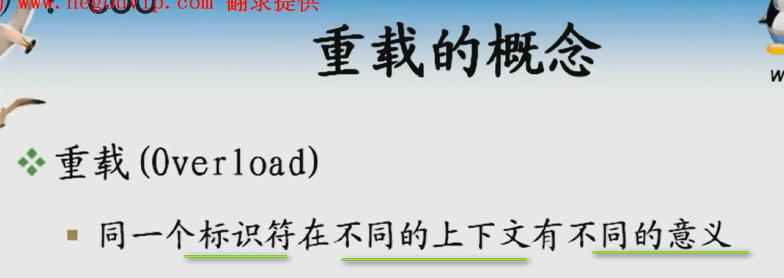
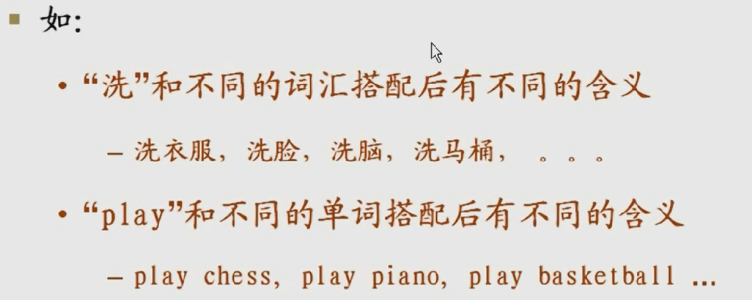


这样的现象叫重载



洗这个表示符 和不同的词搭配 就是不同的含义

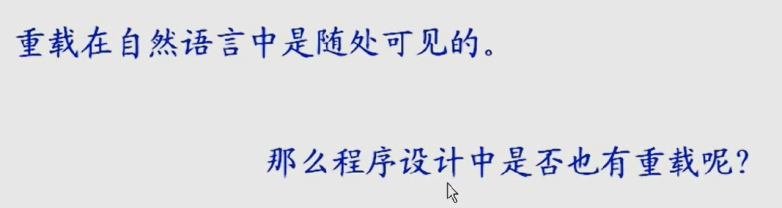
自然语言中 重载很常见



重载的概念 是广泛存在于自然语言中的

**无非就是同一个表示符 在不同的上下文 有不同的含义**

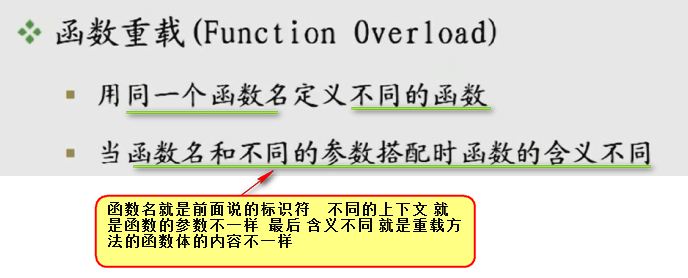
**【重载 就是 重新载入 把含义重新理解】**

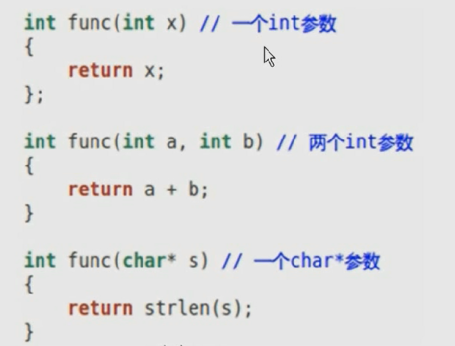


C++是现代程序设计语言 有函数重载的概念

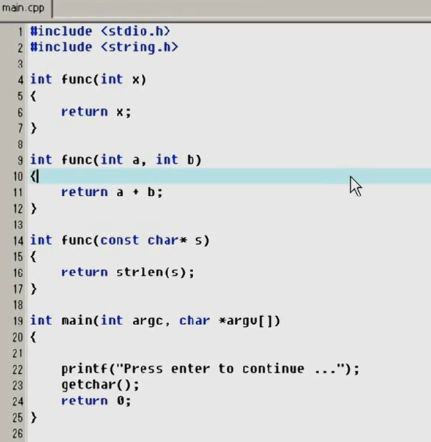
**函数重载：就是用同一个函数名来定义不同的函数**

如何区分？





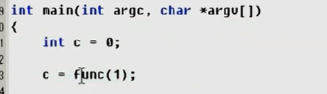
func就是标识符 但是 后面搭配的参数不一样



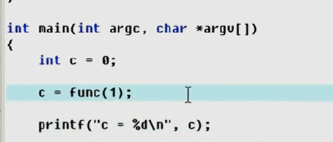
第一个func是将参数返回 第二个func是求和 第三个func是求参数指向字符串的长度

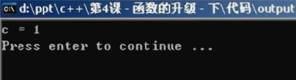
三个同名函数 每一个函数都有自己的意义

编译通过 C++支持重载 重载表现在函数上面

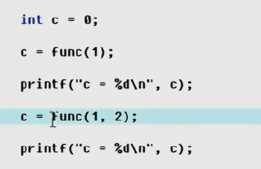


现在调用的func是上面的哪一个？ 这个func跟一个int进行了搭配 --- func在调用的时候 实参是一个int ----这样一找 就知道是第一个func

 编译器一定知道 调用的是哪一个方法

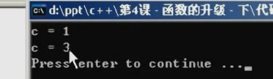


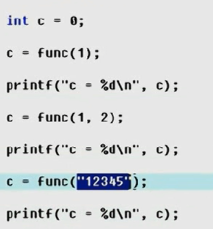
这三个func中 只有第一个func才有可能是一个参数



第二个func就是上面的第二个函数 ---- 因为和两个int进行了搭配 ---- 可以断言 C的值应该是3

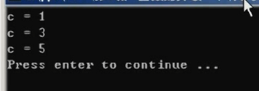
编译通过 编译器已经知道调用了哪一个

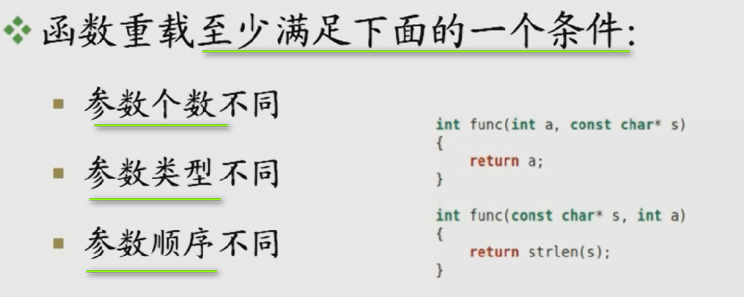




第三个 func和字符串进行搭配 很显然只有第三个进行了匹配

这样c的值应该是5

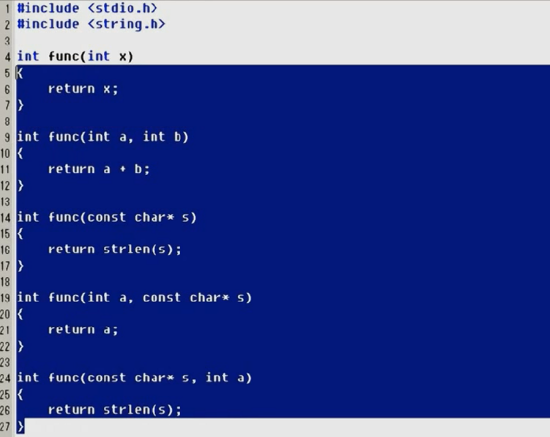




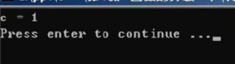
重载的这两个函数 参数个数 参数类型 但是 顺序不同 上面图中的例子 这两个func是两个同名的函数 这两个能否成为重载？

参数个数 参数类型都相同 但是 参数顺序不一样 ---- 所以 也发生了函数的重载

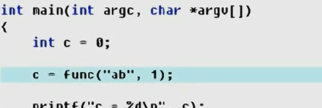
如果编译过了 C++编译器就知道调用了哪一个函数

 一共是5个函数 编译通过 意味着后面func是可以同时存在的 是重载函数

 问我们究竟调用的是哪一个函数



换一个位置



看搭配

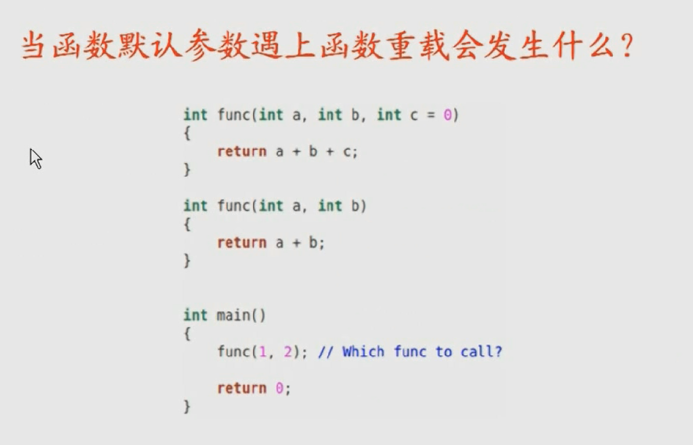
只可能是最后一个func



----- 函数可以重载 是非常有意义的

C语言里面 编写函数的时候 必须要求不同的函数名 即便是里面的行为很相似 也要不同的函数名

C++可以把这些行为类似的函数 重载起来 仅仅通过参数的不同来决定调用哪一个

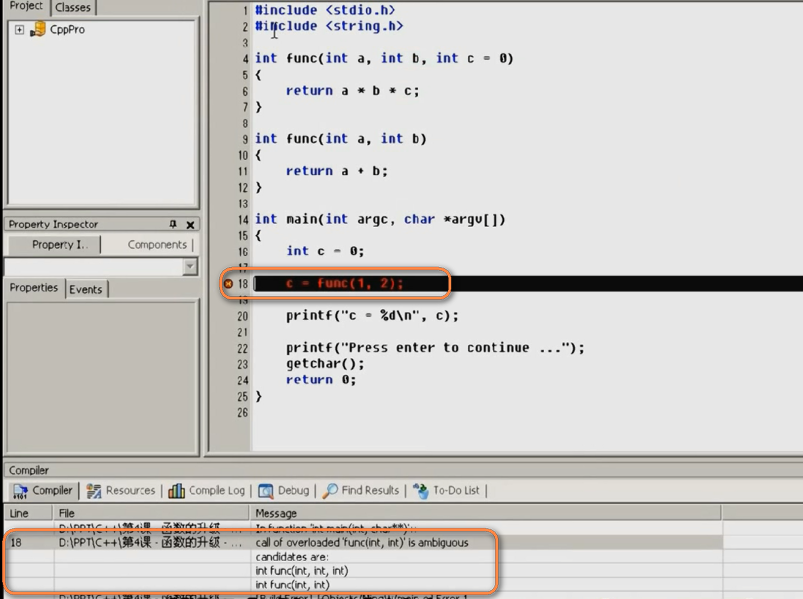


函数重载和函数默认参数相遇 会发生什么？

上面两个函数是满足重载的 因为函数个数不一样

**能够满足重载 能够编译通过**

\*\*下面进行调用 调用的时候 看看能不能通过编译？

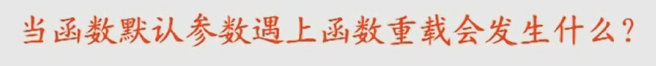


\*\*首先 C++承认这两个函数是重载的

结果 重载是没有问题 编译通过 但是 真的调用的时候 出现了二义性 C++报错了

但是 报错 报什么错？调用重载函数func ----- 说这里面有两个候选者 都可以适合这个地方的调用

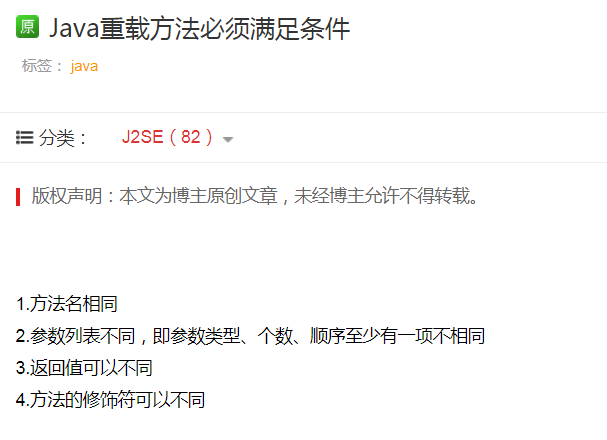
**C++编译器也不知道调用哪一个函数 这个就是程序的二义性**



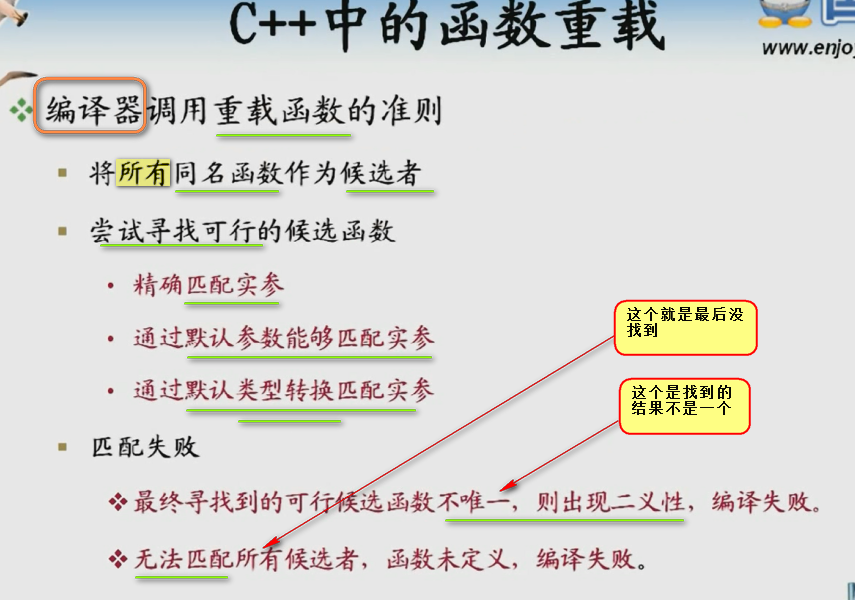
就很有可能出现二义性 --- 所以 工作的时候 用了C++语言 使用了函数默认参数的时候 就不要使用函数重载 ---- 如果决定使用函数重载 就不要使用函数默认值 否则就可能出现二义性 编译无法通过

我们的程序在设计的时候 必须无二义性 ------ 虽然C++支持函数默认参数 和函数重载 但是 最好不要使用 很容易发生二义性

【java中的重载

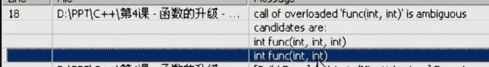


可以看到 Java中的重载的要求 和C++是一样的】



编译器先把func中的候选者挑出来

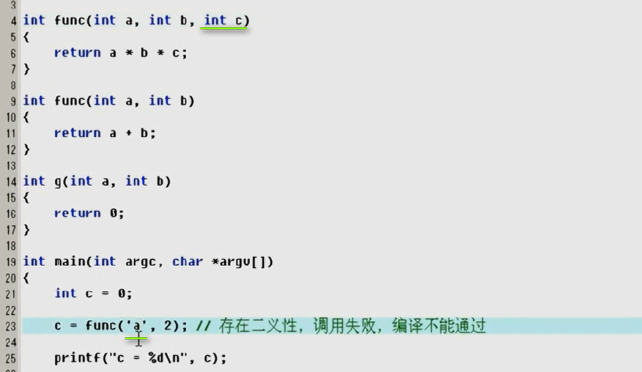
精确匹配/默认参数/类型转换



告诉 不知道使用哪一个



去掉默认的函数参数



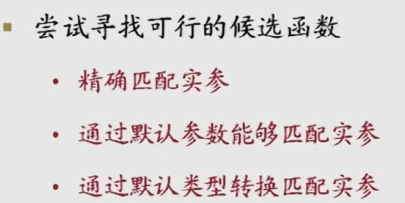
Func都找出来 前面两个 匹配2个参数 只有第二个

但是 实参第一个是字符 但是 func里面没有一个字符 一个int

这时候 编译器中 字符类型能否默认转换成int类型？

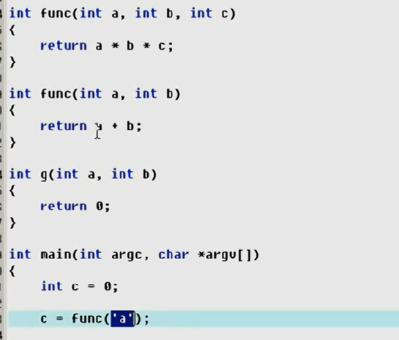
这个字符可以转化为int

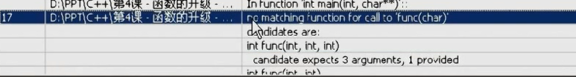
编译通过



精确匹配：就是个数匹配 + 类型匹配

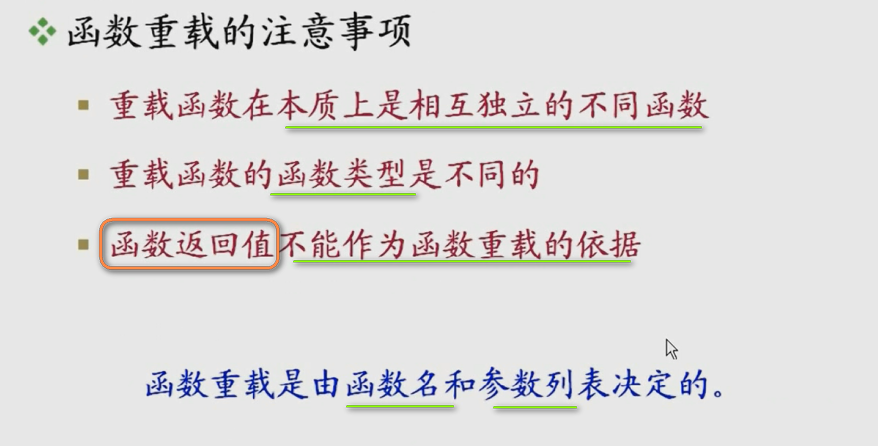
并且这三个条件是同时进行的 没有先后

 找出来 发现没有默认参数值



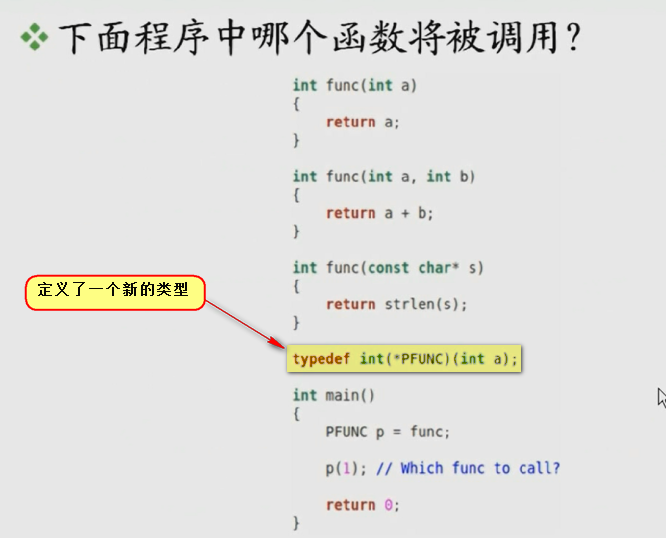
告诉你 找不到这个匹配的

这个时候 直接报错了



【**注意 这个说的是 相互独立的不同的函数 为下面的例子埋下伏笔**】

**函数重载与返回值没有任何关系**



这三个一定是重载函数 下面定义了一个函数指针 PFUNC

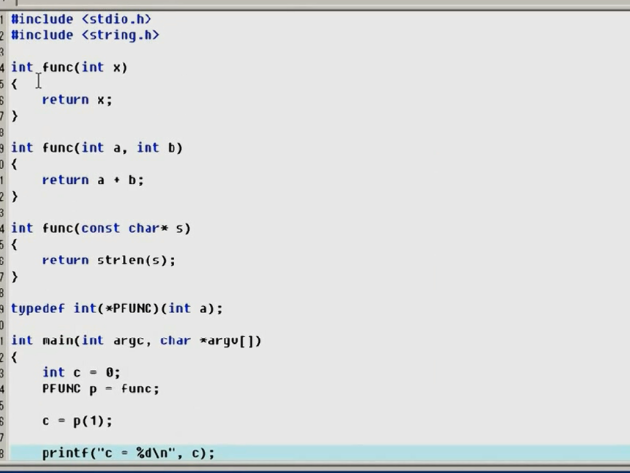
里面声明了一个函数指针  然后进行调用

这里面 哪一个func函数被调用？

是第一个么？ 有人认为因为个数的原因 所以是第一个

但是 运行到PFUNC p = func; 编译器凭什么就会把第一个函数的地址传递给这个PFUNC呢？ 这时候也没有参数过来

实践一下



编译通过了 意味着c++编译器知道c=p(1)知道调用哪一个func 也就是 PFUNC p =func;



C++为什么知道是第一个 而不是第二个？

 这个没有任何参数进行搭配 但是C++怎么知道的？下面的知识点

赋值的时候  --- -根据重载规则挑选与函数指针参数列表一致的候选者 --- 这里面定义的是 typedef int(\*PFUNC)(int a)

根据重载规则挑选 只有第一个 所以 这个时候 就挑选出来了

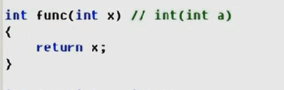


 指向的函数类型 是 int(int a) 正好和第一个匹配上

这样C++编译器就能够肯定这个func一定是第一个

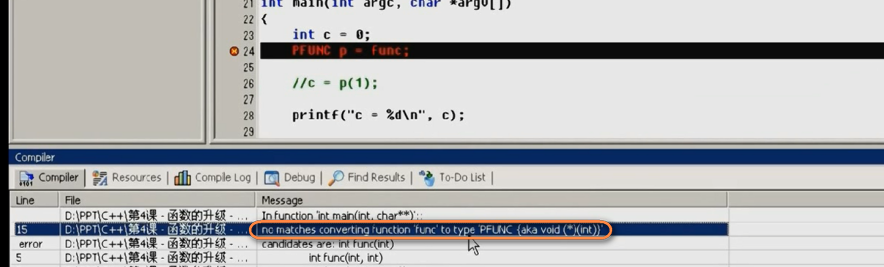
修改一下

 ---- 这个指针的参数列表类型一个int ---- 这样 根据重载规则挑选与函数指针参数列表一致的候选者 只是挑选到第一个func



然后就严格匹配函数类型 ---- 这个时候 PFUNC的函数类型是void(int a) 但是 目标是int(int a )

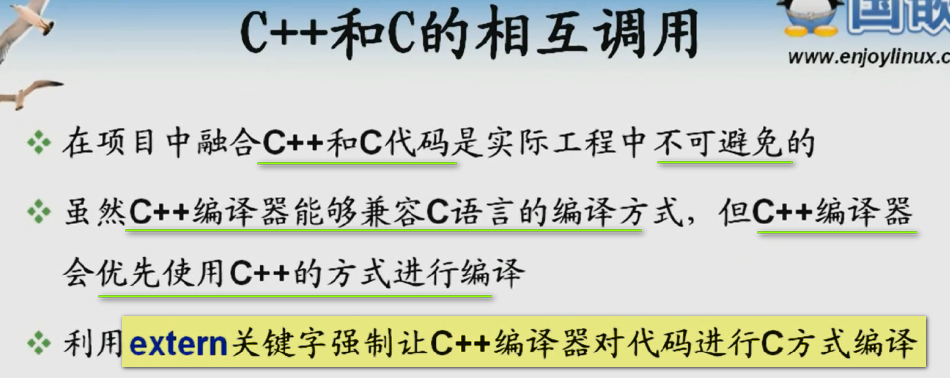
此时 编译器报错



实质上 编译器就是这么进行的

到目前为止 函数重载就有这么多了

只要对函数重载 意义 有了解就可以了



第0课的图片



操作系统用的是C语言 上面的应用程序框架层 和 操作系统抽象层 使用的是C++的话 那么C++和C怎么交互？

现在C++编写的代码 怎么调用C语言编译器编译出来的代码

什么叫C语言编译方式？

什么叫C++编译方式？

编译器不一样 那么如何编译代码肯定不一样

两者有不同的编译方式 只是 C++天生继承C 所以 C++会两种编译方式 ---- 既会C++编译方式 也会C编译方式

但是 C语言不会C++的编译方式

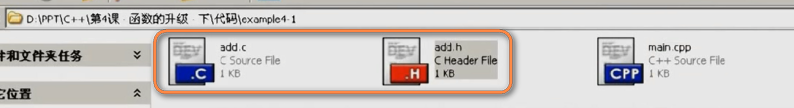
如果让C++和C相互调用 如何进行？

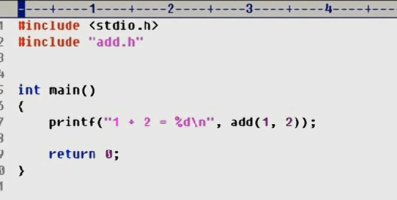
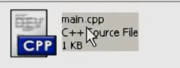
**有一个共同点 就是 都会C编译方式**

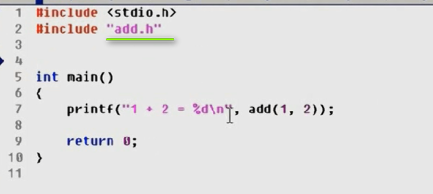
是不是可以使用这个共同点 让C和C++相互调用？ ----

**C++中extern这个关键字被升级了 升级成可以向C++编译器 现在就用C的方式来编译代码**



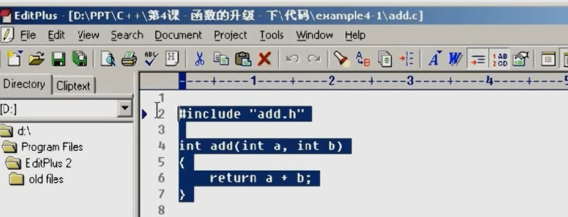


  这个已经是C++工程了

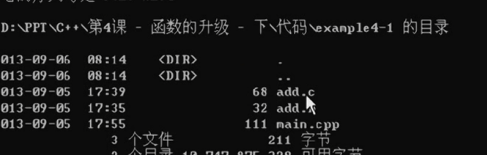


我们这个c++工程就要调用这个add.h中的函数

这个add函数是C语言实现的



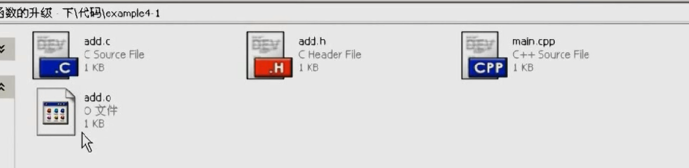
现在假设 做这个产品 有一个产品组 使用C语言开发的



现在 使用gcc对add.c进行编译



编译之后 就生成了一个.o文件

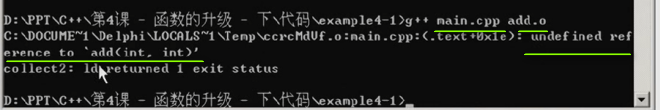


这个组的同事就把.o和.h交给C++组的同事就可以了

C++的同事开始进行测试

因为在main中用到的add是别的组提供的.o 并不用关心如何实现

进行编译



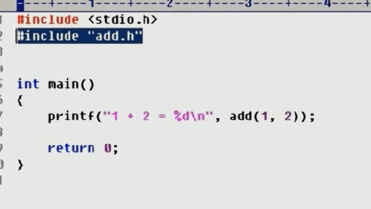
编译不通过 说的是 没有定义这个add函数 ---- 那就奇怪了

然后c++ ---- 为什么这个.o文件无法使用？

产品要上线 但是 不可能都用C++重写

问题就是C++是现代编译器 默认是C++方式进行编译的

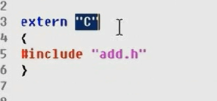
在main中

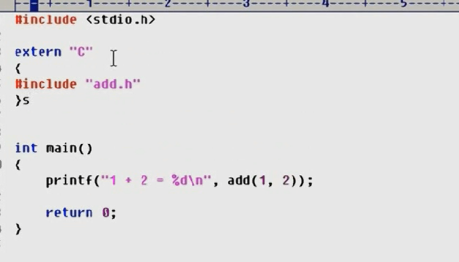
 这样 c++的方式编译头文件 编译这段main方法

但是 提供的add.o是c方式编译出来的

所以 这个g++编译器就找不到 ----- 因为编译方式不同 如何纠正？

头文件和.o 是C语言编译出来的

 这个时候 这样写 表示 用C的方式来编译头文件中的内容





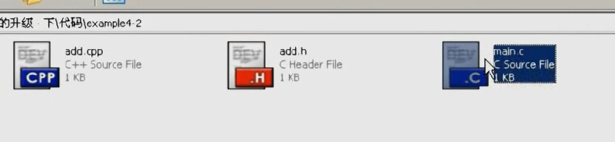
编译通过



这个就是C++调用C代码 --- 关键就在高速C++编译器 头文件中所有的函数 都是C语言编译出来的

知道之后 就可以直接找add.o里面的内容

**反过来可以么？用C调用C++的函数？**



现在主程序是main.c是C语言编写的

但是 add.cpp这个是c++编写的

C++组的同事 编写了add.cpp 给C语言组

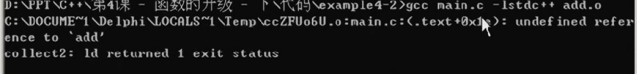
此时 进行编译





这个.o和.h 就拿给C语言的同事使用

C语言拿到之后 直接编译

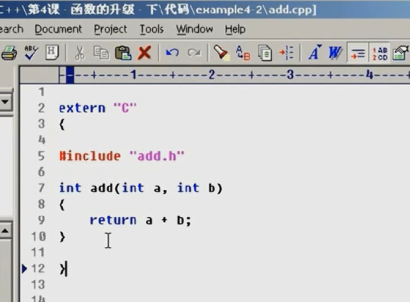


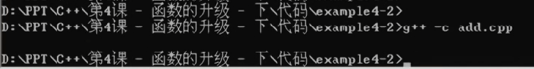
还是说明是没有定义的

原因很简单 add.o是C++编译器编译的 但是 程序使用的C++默认编译方式 ----- C语言编译器却不会C++的编译方式 ---- 所以找不到这个add

这个时候 C++和C功能 就是**有一个共同点 就是 都会C编译方式**

所以 在C++中 就直接用C的方式编译代码





编译通过

C语言组的同事 重新使用add.o

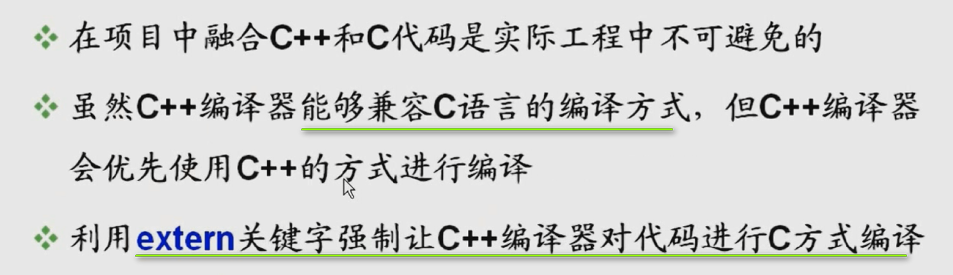


编译通过了 可执行程序出来了

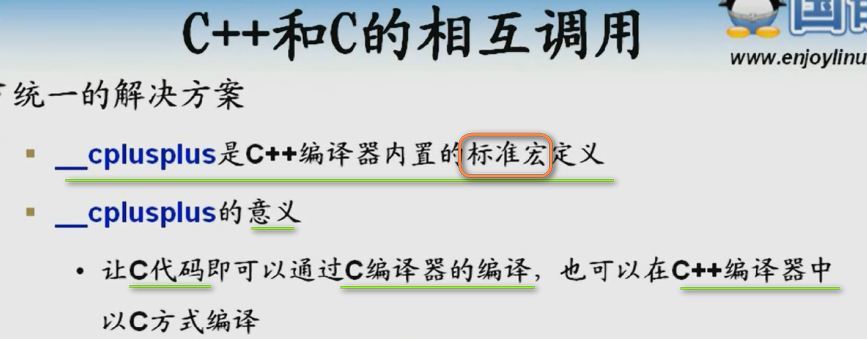
【<http://www.kuqin.com/language/20090322/41866.html>

Typedef的用法

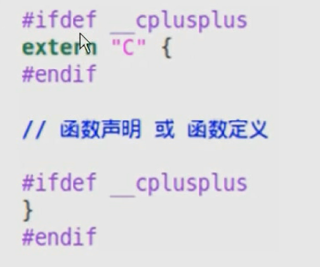
】



=========== C++和C相互调用的小技巧 用的非常多

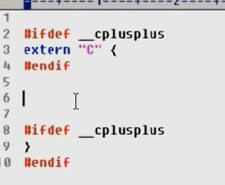


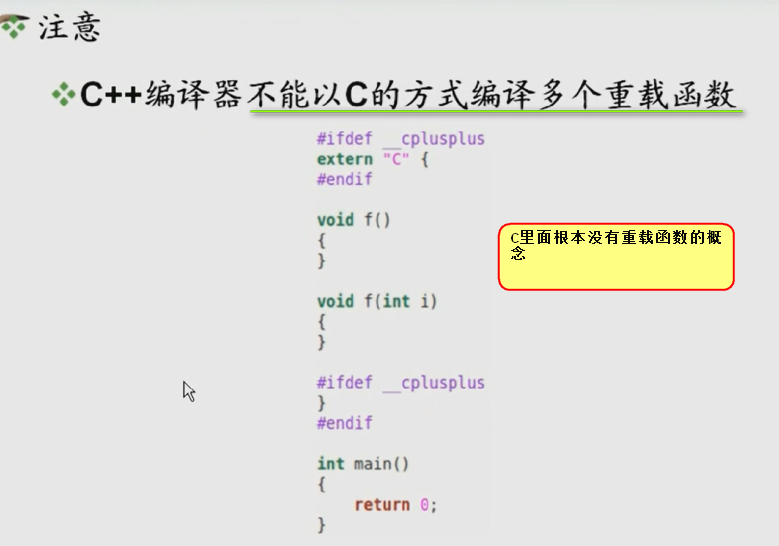
\_\_cplusplus 写一份代码 ---- 编写一份代码 在C语言编译器可以编译通过 在C++编译器下面 是C方式编译通过

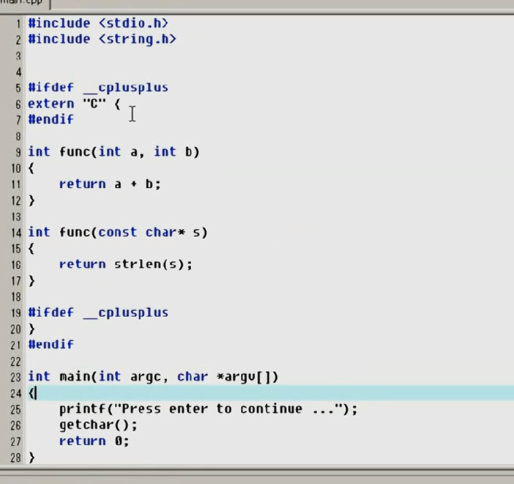


如果\_\_cplusplus 存在 就表示这个是C++编译器 这个时候 加上extern “C”

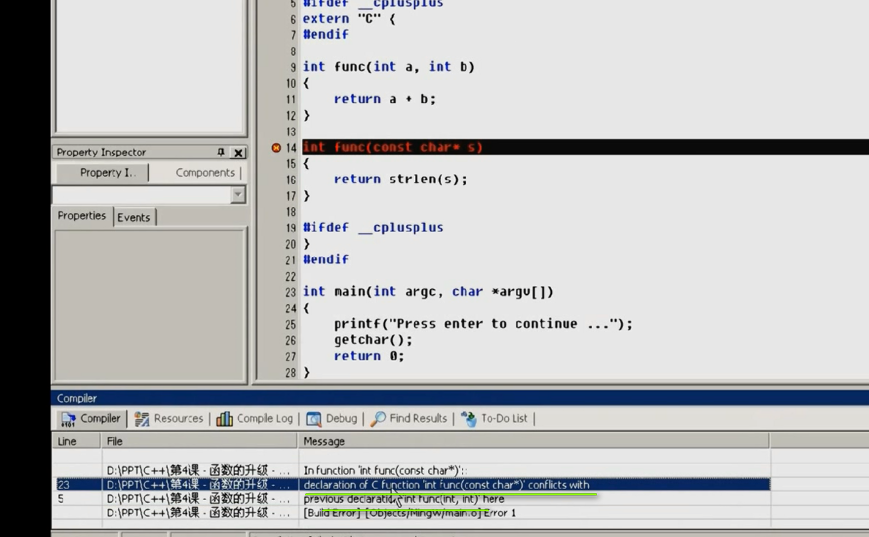
如果是C编译器 这里面肯定没有\_\_cplusplus 没有的话 这条语句 就会被预处理器删除掉

 这样 无论什么版本的C版本都能通过 然后在C++下面 又是以C方式进行编译的 所以 这个技巧是非常重要的





这段代码 就是想以C的方式编译这两个重载函数



C语言的方式编译这两个函数 但是C不允许函数同名 所以 就会报错 所以 事实告诉我们 我们的实践 重载函数只属于C++的

