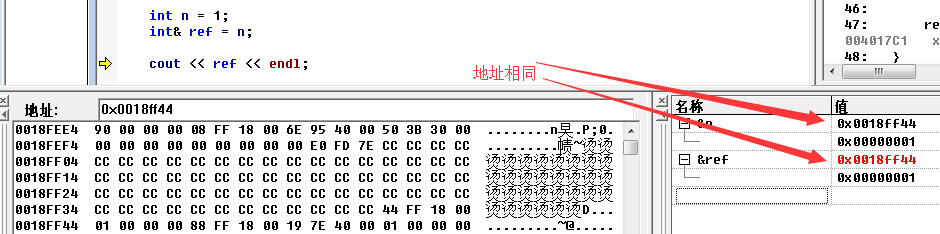
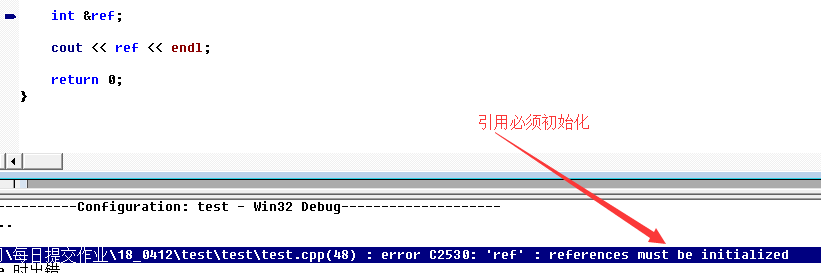
1. 引用

为了解决指针太危险。

1.引用是变量的别名，引用某一个变量，那么该引用相当于给这个变量起别名

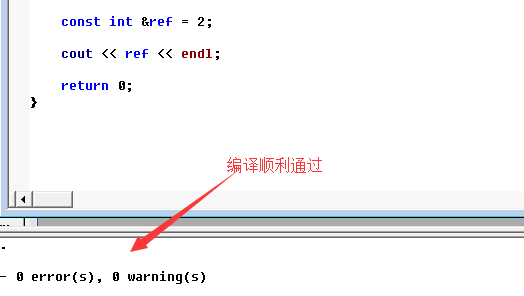


2.引用时必须初始化



3.引用不能用常量初始化，但是const引用可以

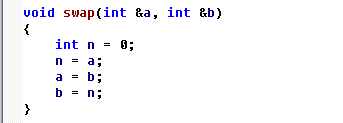


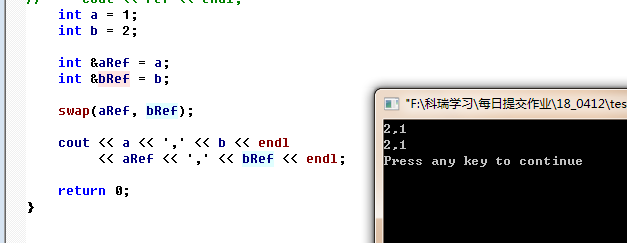


4.没有二级引用

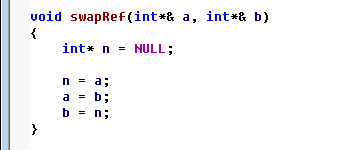
5.引用的常用场景：引用做参数

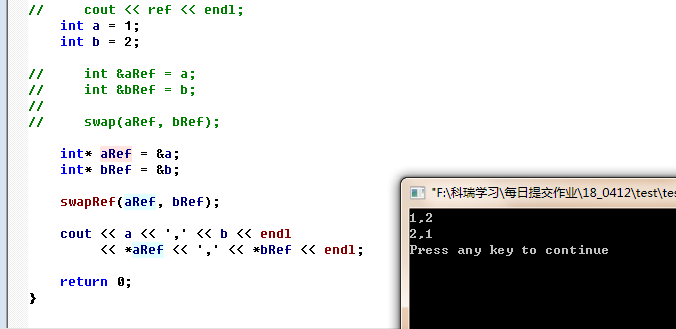
a. 交换值





b.交换地址





6.引用做返回值

int& foo()

{

int n = 3;

int& ref = n;

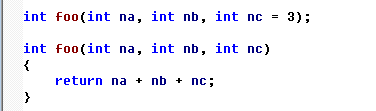
return ref;

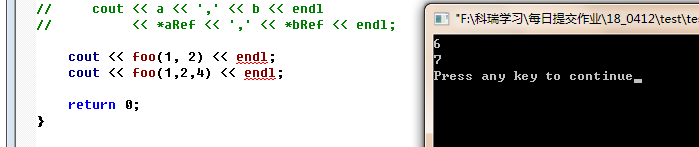
}

foo运行结束后，ref相当于引用一个释放了的值，n的生命期已经过了，所以并不能解决指针的问题，其他高级语言有垃圾回收机制在语法上解决此问题。

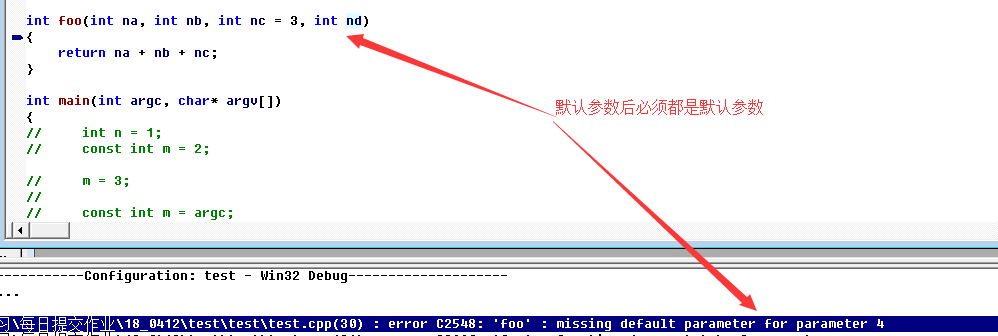
1. 函数默认参数

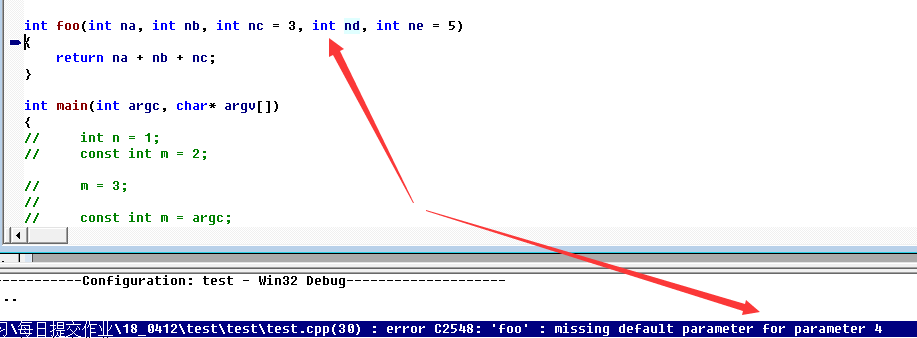
1.当nc没传值时，为默认值；当nc传值是则为具体值





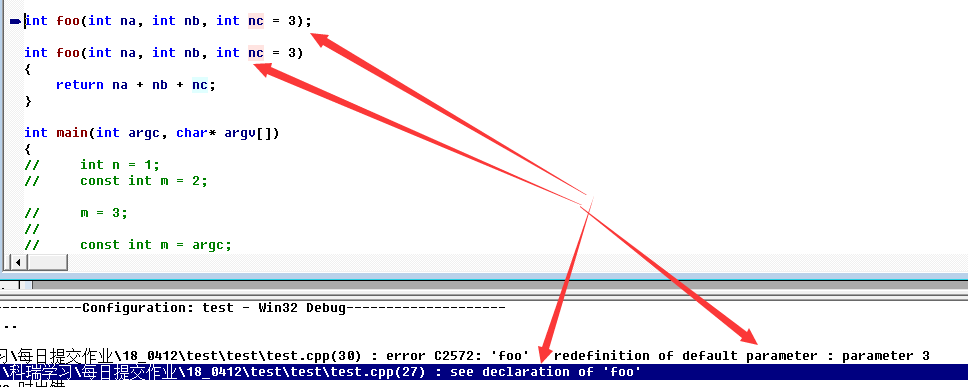
2.默认参数右边必须都是默认参数



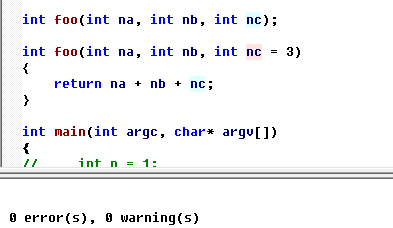


1. 默认参数申明和实现在一起是，默认参数需要写在申明中

a.若申明和实现都有默认参数



b.若默认参数在实现中(疑问)



三.内联

1.关键字：line

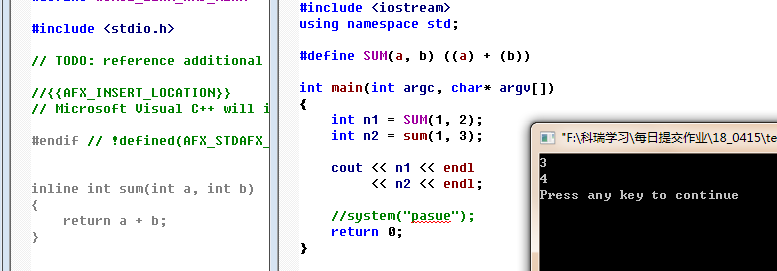
2.关键：

a.Debug版本，release版本不起作用。

b.inline是对编译器的建议。//有时可以，有时不行。

c.通常函数内有switch-case或者递归，那么函数不会被内联。

d.inline函数必须在头文件.h中申明和定义

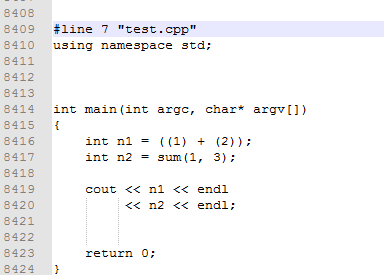


注意：

3.函数、预处理和内联函数区别：

预处理：函数有出栈入栈的过程，同等情况比宏慢。

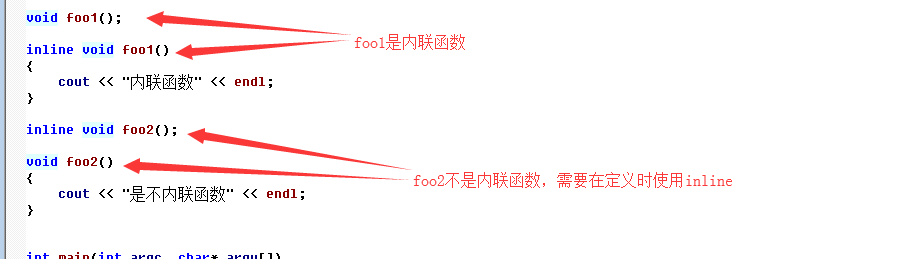
函数：预处理是文本替换，本身是没有参数检查的。



4.内联函数：

a.解决了函数和预处理的缺点即有参数检查同时不压栈，但实际也使用栈针(debug版)，releas版本才做到真正内联。

b.关键字inline必须与函数定义放在一起才能使函数变成内联，仅将inline放在函数声明是没有作用的。



c.内联函数虽然能提高函数的执行效率，但是它是以膨胀为代价(每一处内联函数的调用都要拷贝代码，会使程序的总代码量增大，消耗更多的内存空间)，只是省去了函数调用的开销，从而提高程序的执行效率。

d.由c点知道，当执行函数体内的代码时间比函数调用的开销大得多时，则要慎重使用内联函数。