第6讲：函数

1、什么是函数

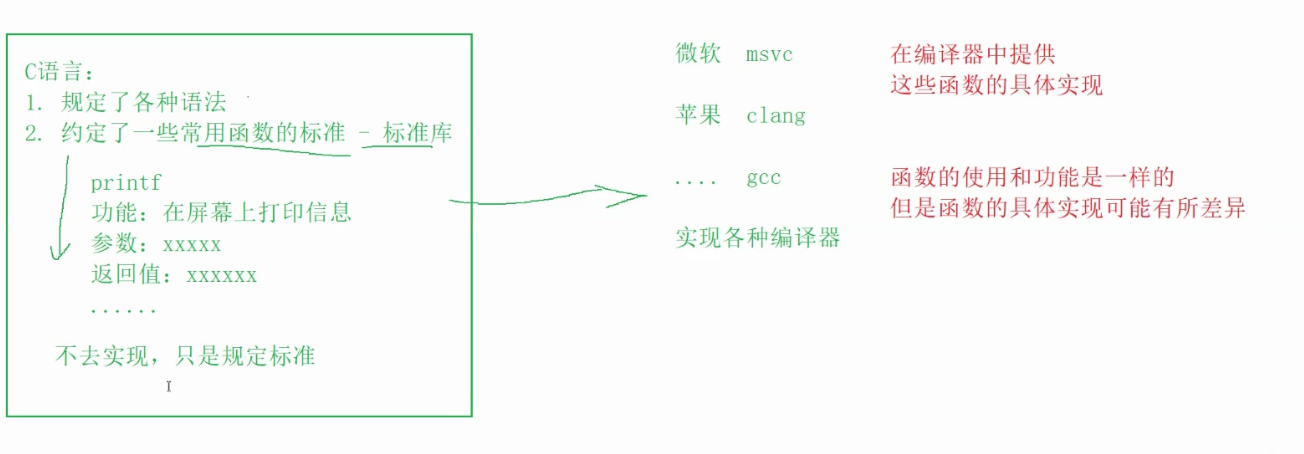
数学中的函数：y=f(x)，x∈D，式中x为自变量，y为因变量，f为对应法则，D为x的定义域，对于任意的x∈D，都有唯一一个y值与之对应。例如一次函数：y=kx+b，其中k、b都为常数。

C语言中的函数：子程序，完成某项特定任务的一小段代码。C语言将其一般分为库函数（现成的函数）、自定义函数（自己创造的函数）。

2、库函数

2.1 标准库和头文件

C语言标准中规定了C语言的各种语法规则，C语言并不提供库函数；C语言的国际标准ANSI C规定了一些常用的函数的标准，被称为标准库，那不同的编译器⼚商根据ANSI提供的C语⾔标准就给出了⼀系列函数的实现。这些函数就被称为库函数。（说人话：1.C语言规定了各种语法，比如：for循环怎么写。2.约定了一些常用函数标准 — 标准库，比如：我们想在屏幕上打印信息，规定一个printf函数（功能、参数、返回值……）。3. 编译器根据标准库设计、实现的函数称为库函数。）



库函数都放在头文件中进行声明，使用时需要包含对应的头文件。

库函数相关头文件：

C/C++官网：<https://en.cppreference.com/w/c/header.html>

Cplusplus：<https://legacy.cplusplus.com/>

2.2 库函数学习和查看

使用Cplusplus查看sqrt函数：



库函数文档的一般格式：

（1）函数原型

（2）函数功能介绍

（3）参数和返回类型说明

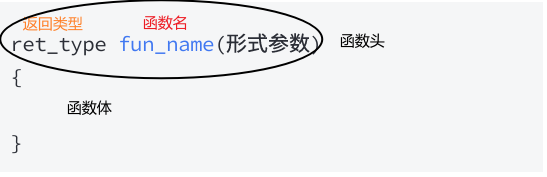
（4）代码举例

（5）代码输出

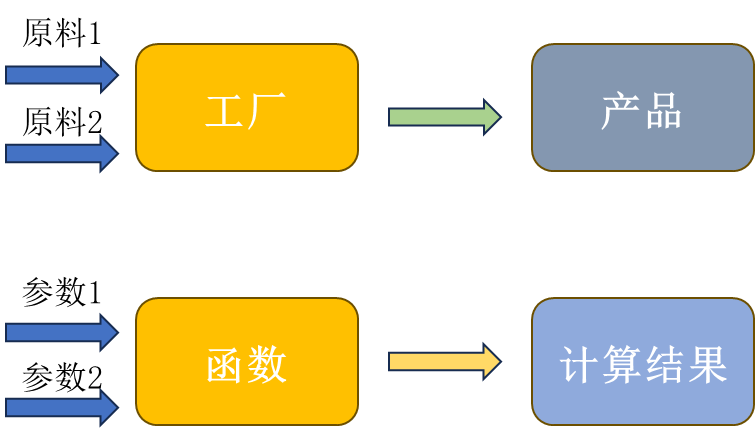
（6）相关知识链接

3、自定义函数

3.1 函数的语法形式

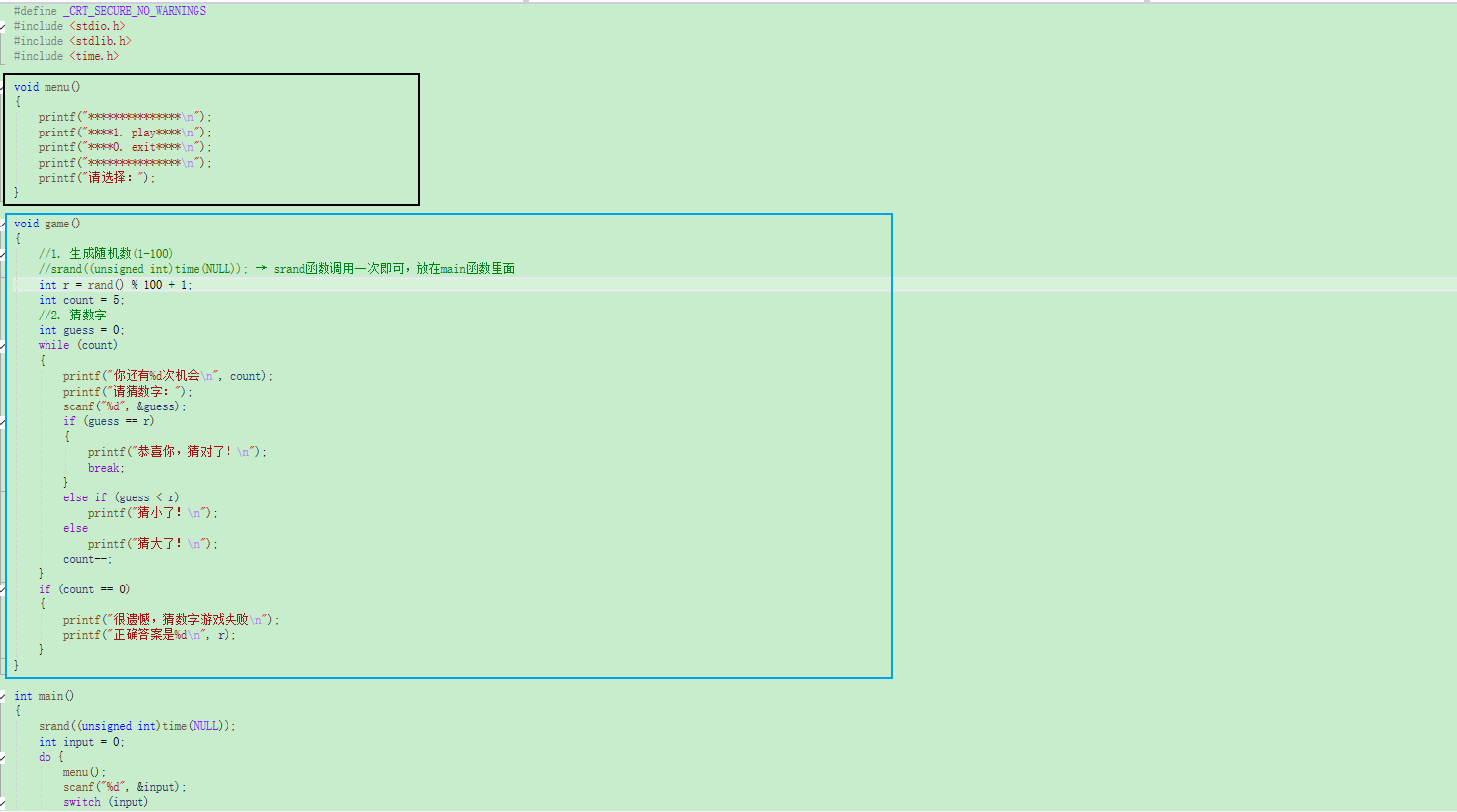


函数可以想象成工厂：



函数的返回类型有两类：（1）void；（2）其他类型 — int short char。

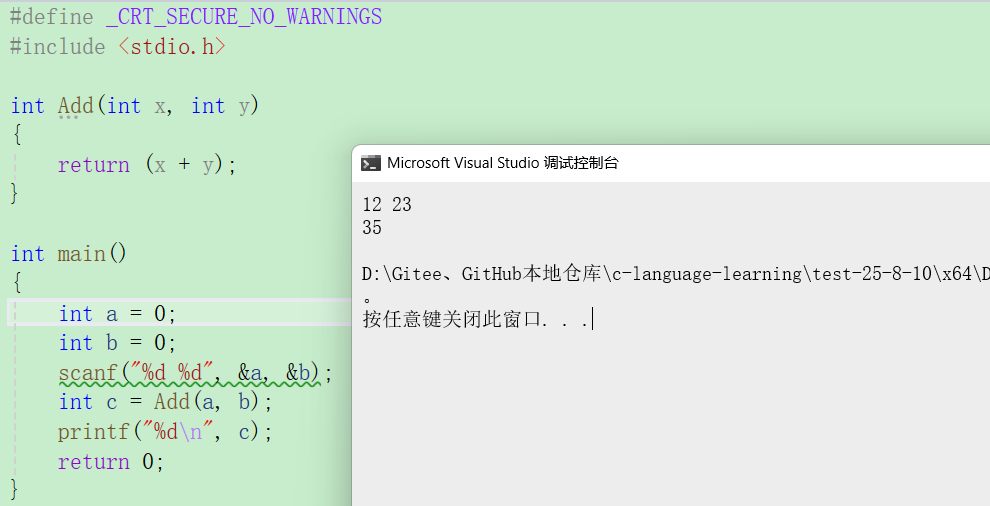
之前写的猜数字游戏：



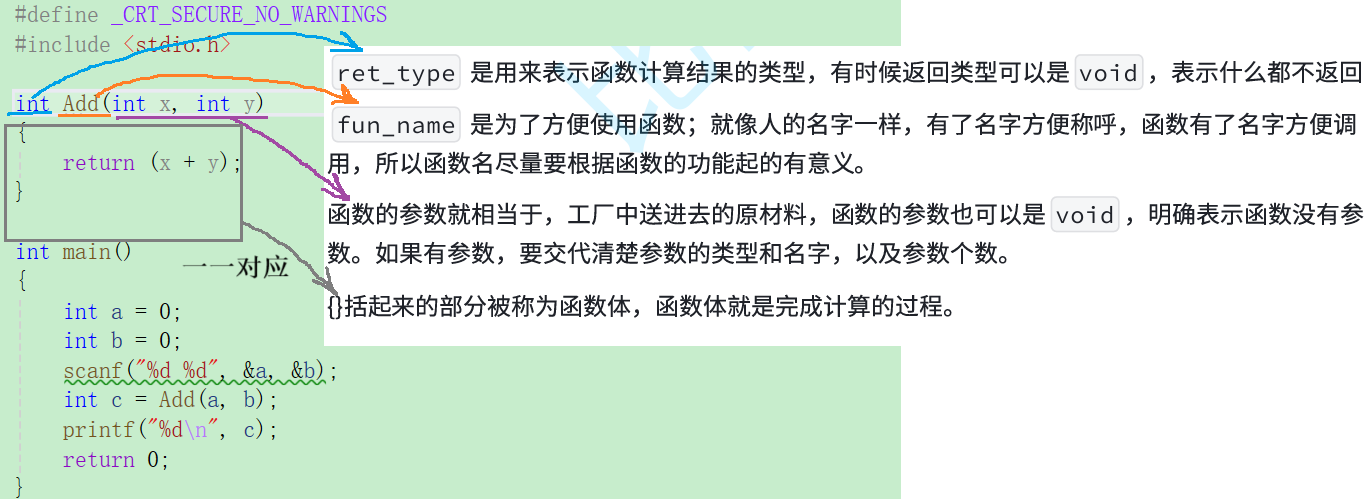
上述代码中，menu、game函数均为我们自己创建的函数，并不需要参数，没有返回值。

3.2 函数应用

使用函数，计算两个整数的和：

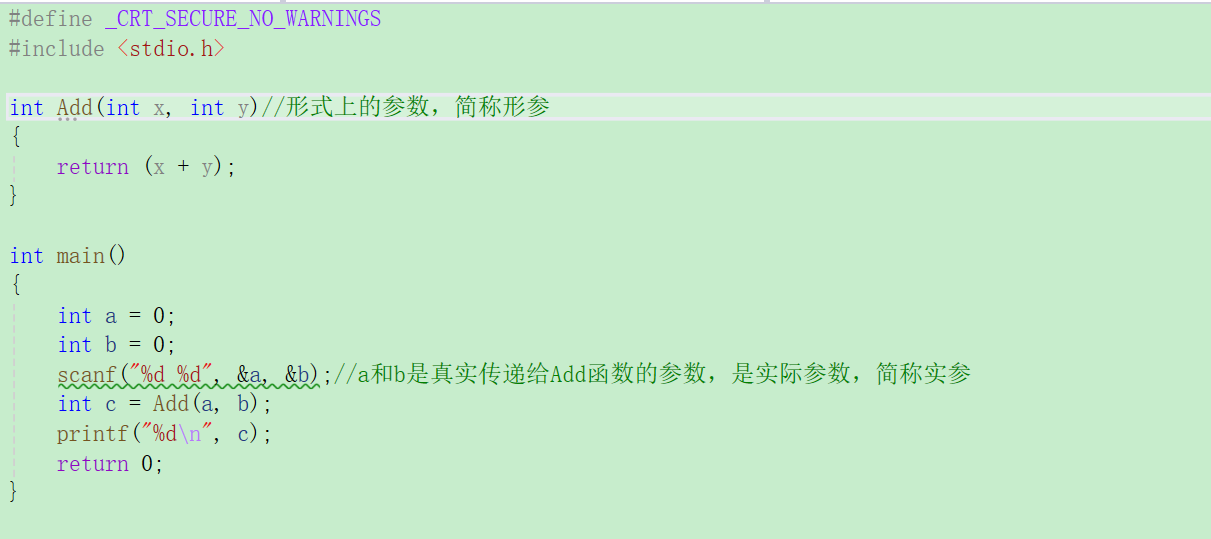


剖析：



4、实参和形参

4.1 形参和实参的概念



代码中：a、b → 实参，x、y → 形参。

形式参数：只有在调用Add函数，存放传递过来的实参时，内存才会为其开辟空间（形参的实例化），其他情况下，x、y只是作为一种形式上的存在。

4.2 形参和实参的关系

分别对a、b，x、y进行取地址操作。



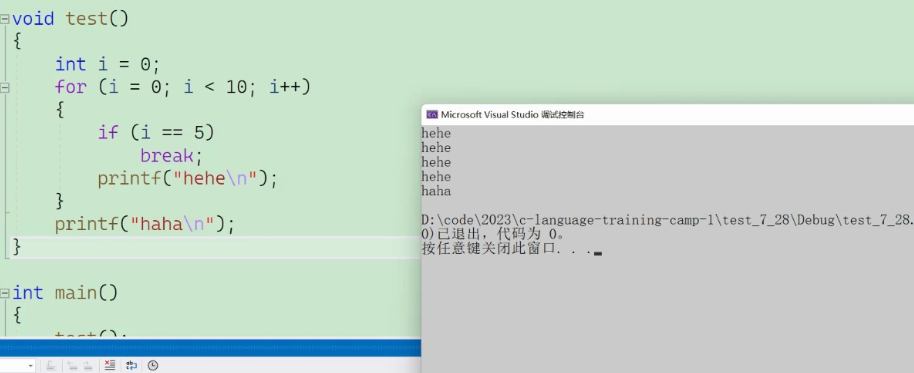
x和y确实得到了a和b的值，但是x和y的地址和a和b的地址是不相同的，所以我们可以理解为**形参是实参的一份临时拷贝**。从Add函数出来后，形参申请的内存还给操作系统。

5、return语句

5.1 return后边可以是⼀个数值，也可以是一个表达式，如果是表达式则先执行表达式，再返回表达式的结果。例如上面的代码。

5.2 return后边也可以什么都没有，直接写return; 这种写法适合函数返回类型是void的情况。

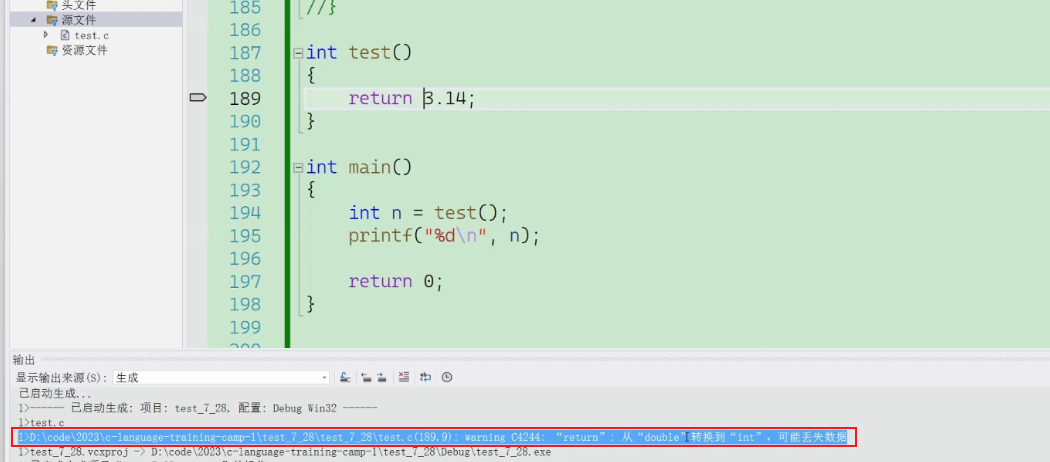
break和return对比：



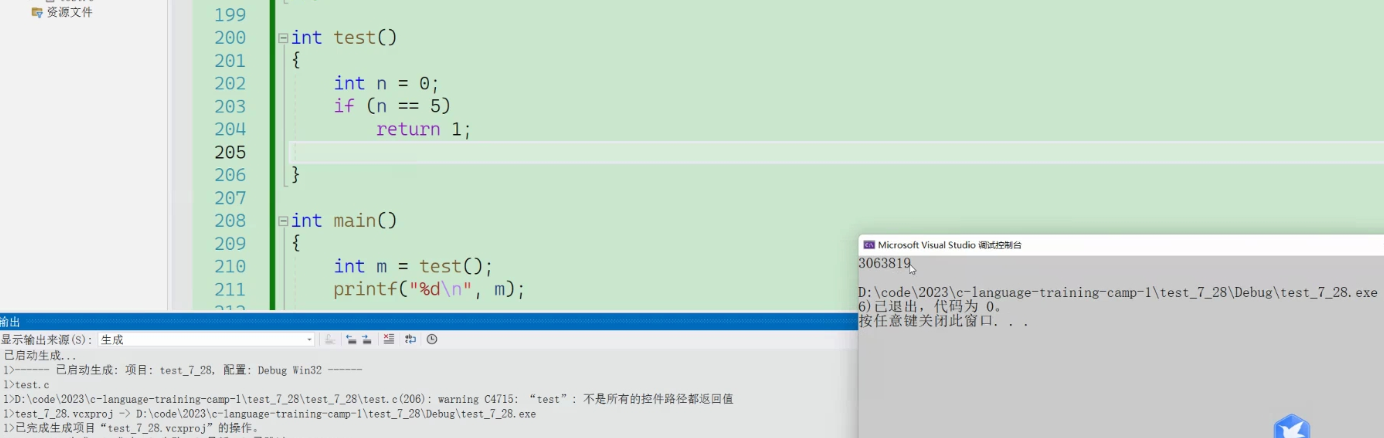


break没那么强势，只是跳出循环；return直接结束函数。

5.3 return返回的值和函数返回类型不⼀致，系统会⾃动将返回的值隐式转换为函数的返回类型。



5.4 如果函数中存在if等分⽀的语句，则要保证每种情况下都有return返回，否则会出现编译错误。



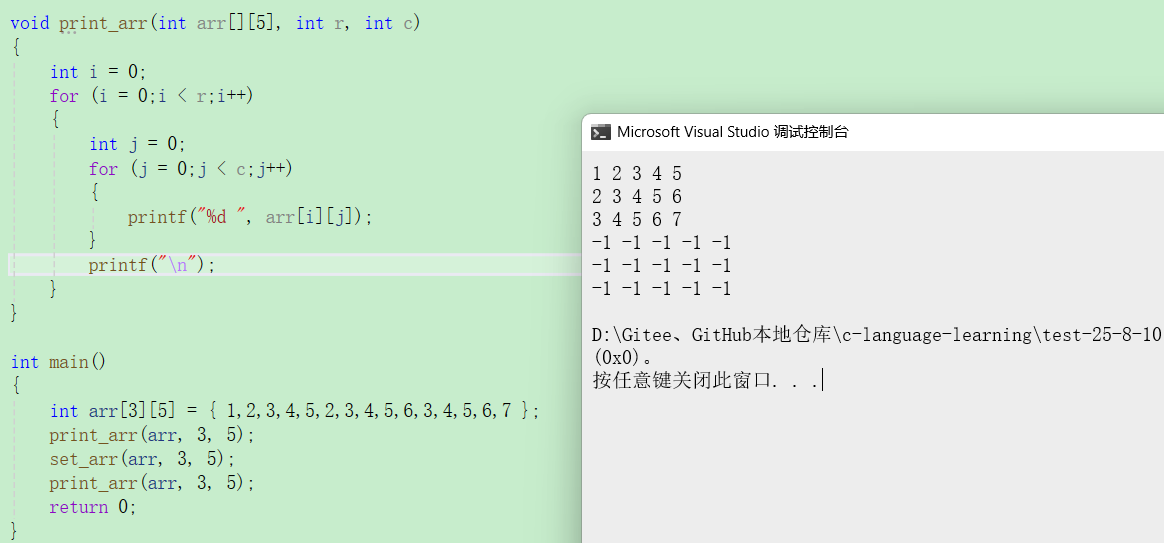
6、数组做函数参数

写一个函数对将一个整型数组的内容，全部置为-1，再写⼀个函数打印数组的内容：

一维数组：



二维数组：



结论：

（1）函数的形式参数要和函数的实参个数匹配。

（2）函数的实参是数组，形参也是可以写成数组形式的。

（3）一维数组传参时，形参部分[]里面的内容可以省略。（可以写错）

（4）形参如果是二维数组，行可以省略，但是列不能省略。

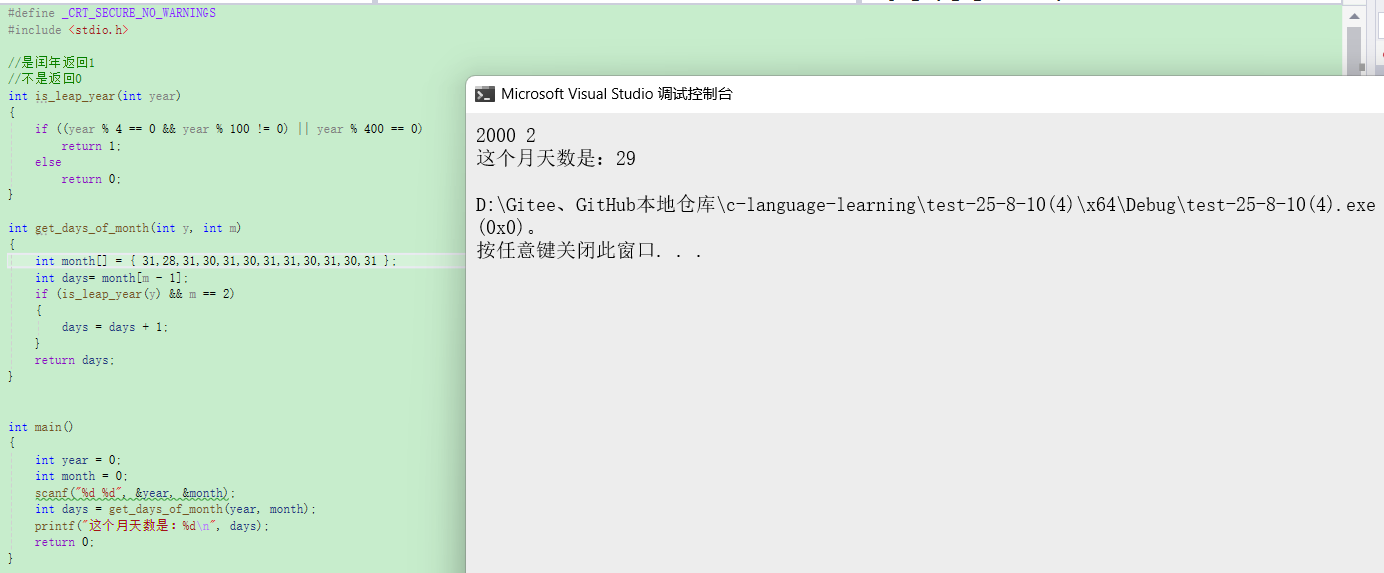
**（5）数组传参，形参是不会创建新的数组的。形参操作的数组和实参的数组是同一个数组。**

7、函数的嵌套调用和链式访问

7.1 嵌套调用

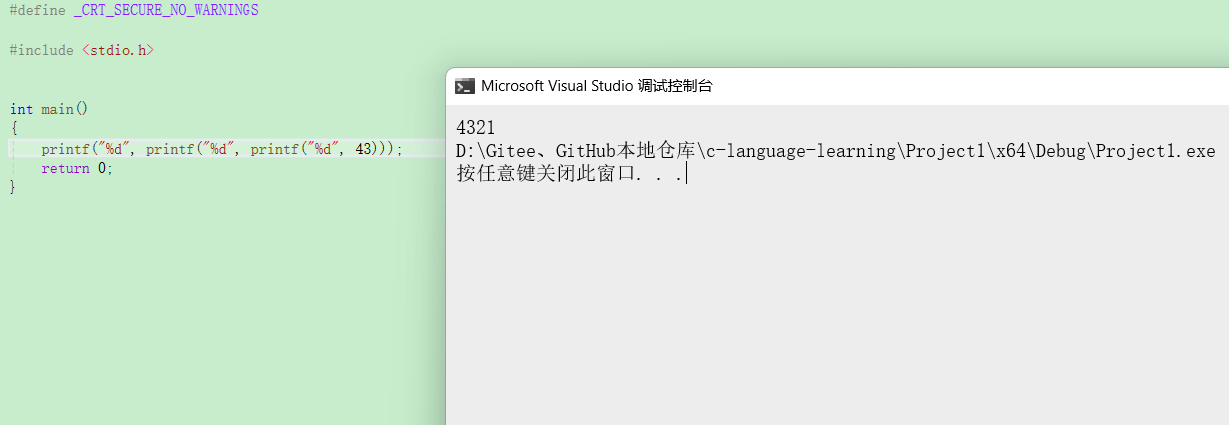
函数之间的互相调用，可以把函数一个乐高零件，正是因为多个乐高零件互相无缝配合才能搭建出精美的乐高玩具，也正是因为函数之间有效的互相调⽤，最后写出来了相对大型的程序。

求某年某月的天数：



7.2 链式访问

把一个函数的返回值当作是另外一个函数的参数。



（1）先执行printf(“%d”,43);打印43。

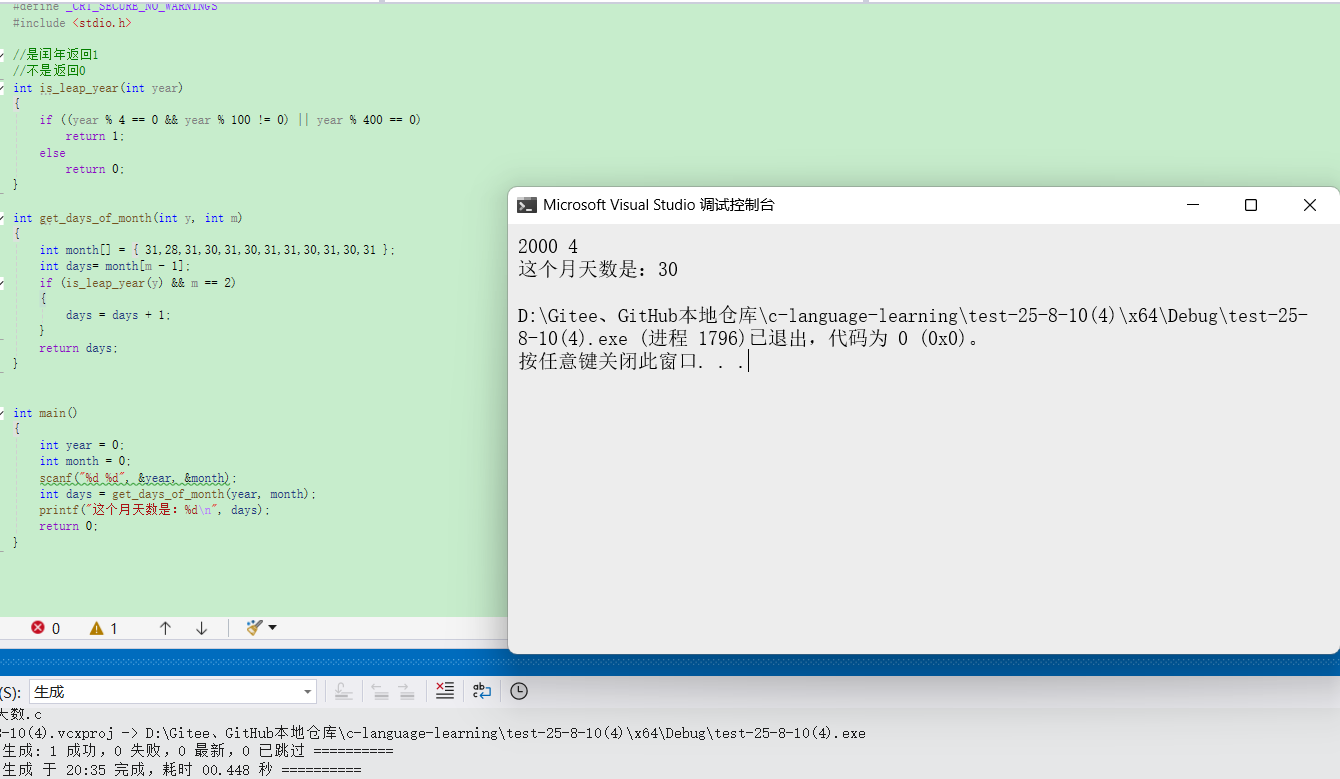
（2）再执行printf(“%d”, printf(“%d”,43)), 打印printf(“%d”,43)返回值2。

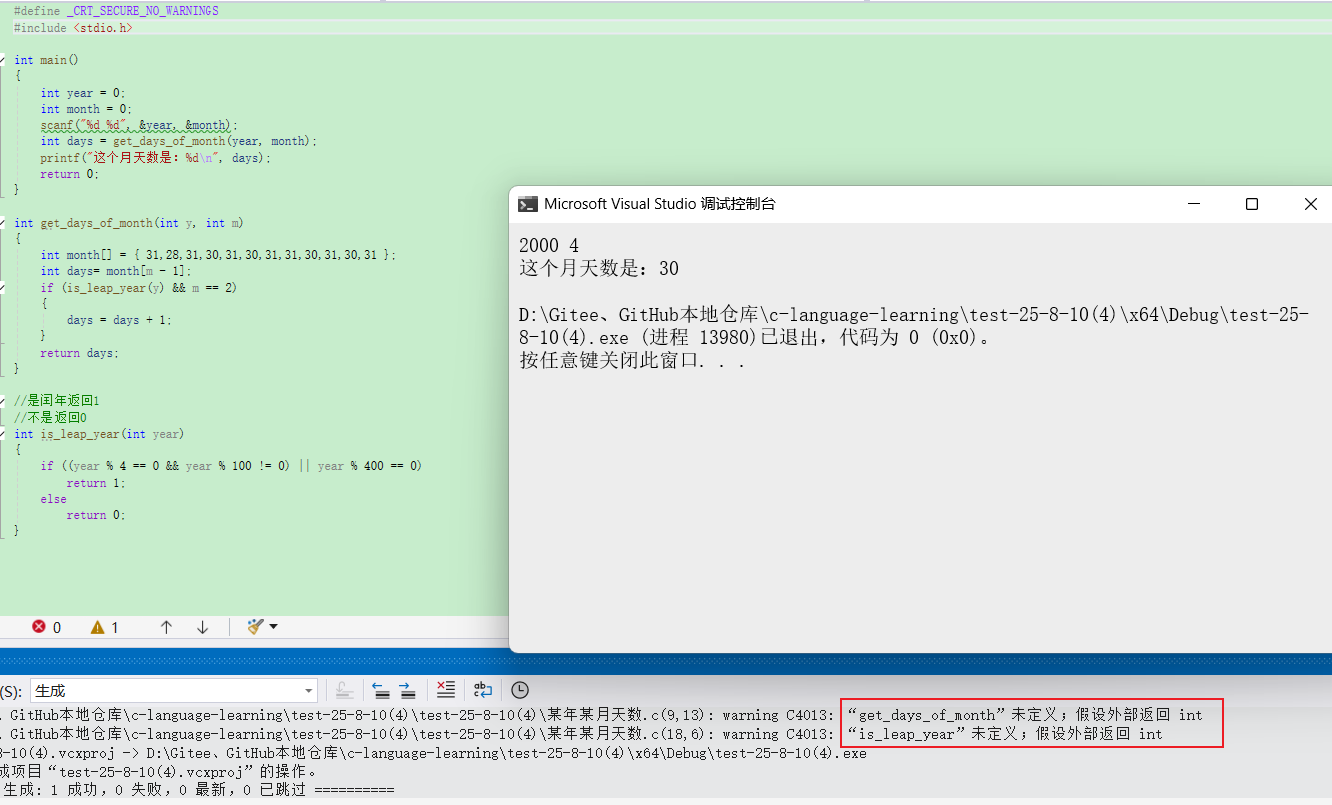
（3）再执行printf(“%d”, printf(“%d”, printf(“%d”,43))),打印printf(“%d”, printf(“%d”,43))返回值1。

8、函数声明和定义

8.1 单个文件

单个文件指我们将函数的定义和声明都写在同一个.c文件里面。

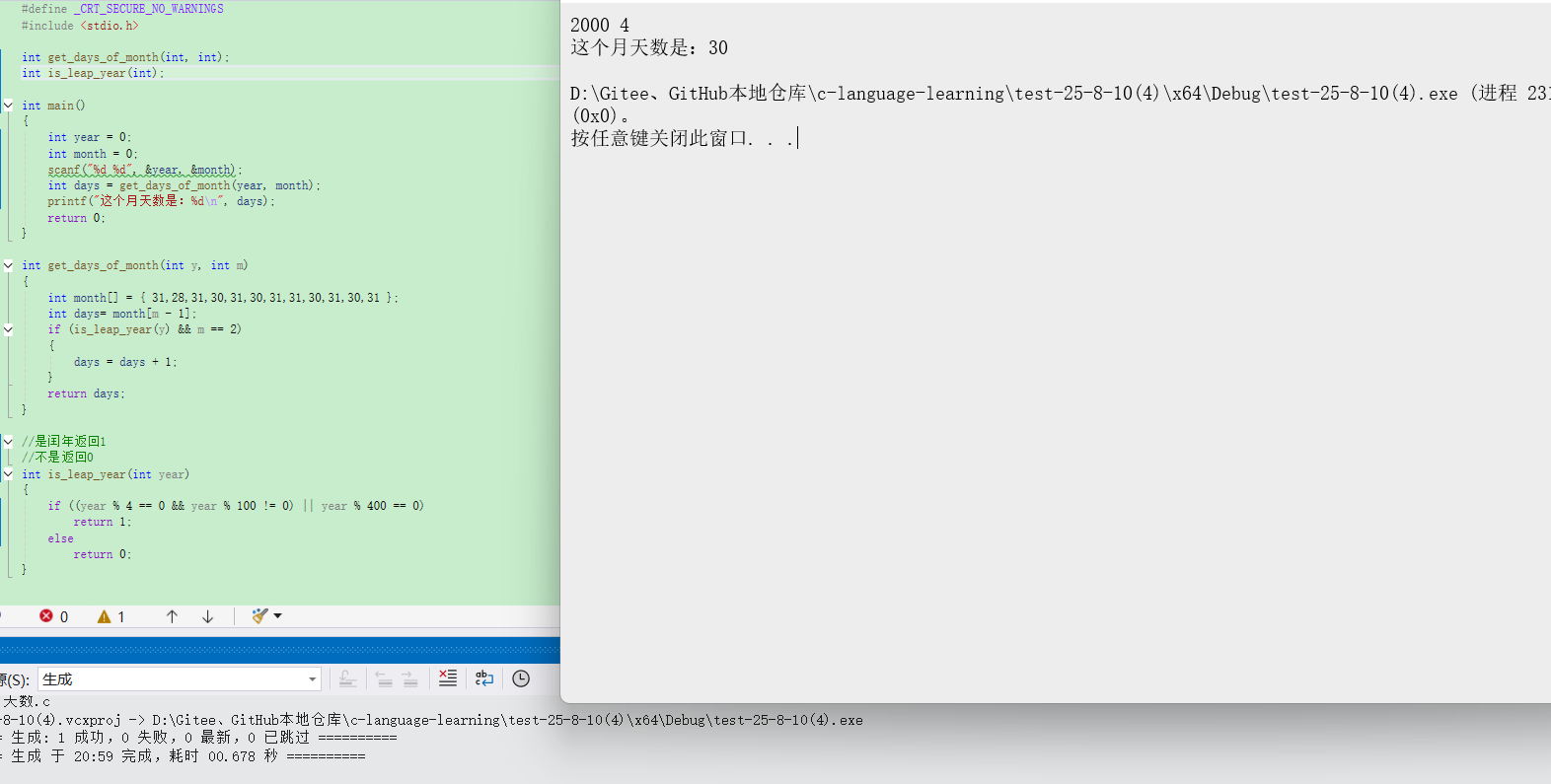




对于之前写过的求某年某月有多少天的代码，函数定义在调用之前，编译器没有报错，但是当我们将函数定义写在定义之后，编译器就会报警告（程序依然能运行）。

原因：**编译器在编译源代码时，是从第一行开始对代码进行逐行扫描，当遇到get\_days\_of\_month函数调用的时候（is\_leap\_year函数类似），并没有发现前面get\_days\_of\_month函数的定义（调用is\_leap\_year函数也没有发现前面有定义），就报出了上述的警告。**

解决方法：**在函数调用之前，先声明一下get\_days\_of\_month函数即可（函数声明包括：函数名、函数的返回类型、函数的参数）。**



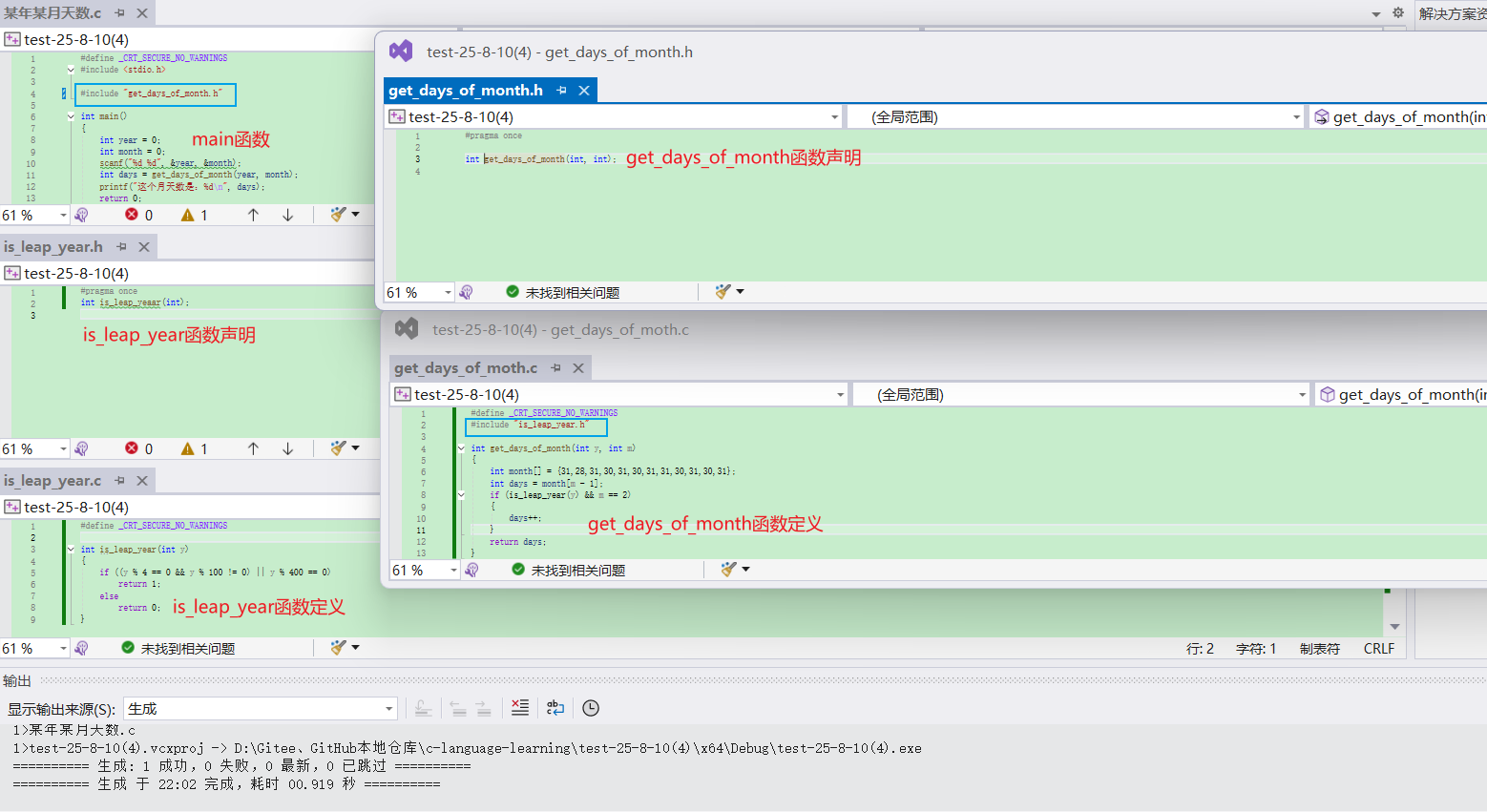
另外，**函数在声明的时候，具体参数名是可以省略的**，即声明可以写成：int get\_days\_of\_month(int, int); int is\_leap\_year(int);

为什么函数写在调用之前，编译器就不会进行报错呢？其实，**函数定义是一种特殊的声明。在使用函数的时候，一定是先声明再使用。**

8.2 多个文件

上面我们把函数定义和声明写在同一个.c文件中，只适用于比较简单的代码，但在实际中，尤其是在公司里面，代码是非常多的，这就需要我们将代码拆分，放在多个文件中。

一般情况下，**函数的声明、类型的声明放在头文件（.h）中，函数的实现是放在源文件（.c）中，**当我们需要使用某个函数的时候，包含对应的头文件即可。



在包含头文件的时候，为了能和库函数包含区别开来，使用" "。另外，使用#include "get\_days\_of\_month.h"、#include "is\_leap\_year.h"相当于将.h文件中的int get\_days\_of\_month(int, int); int is\_leap\_year(int);拷贝到需要使用该函数的.c文件中。

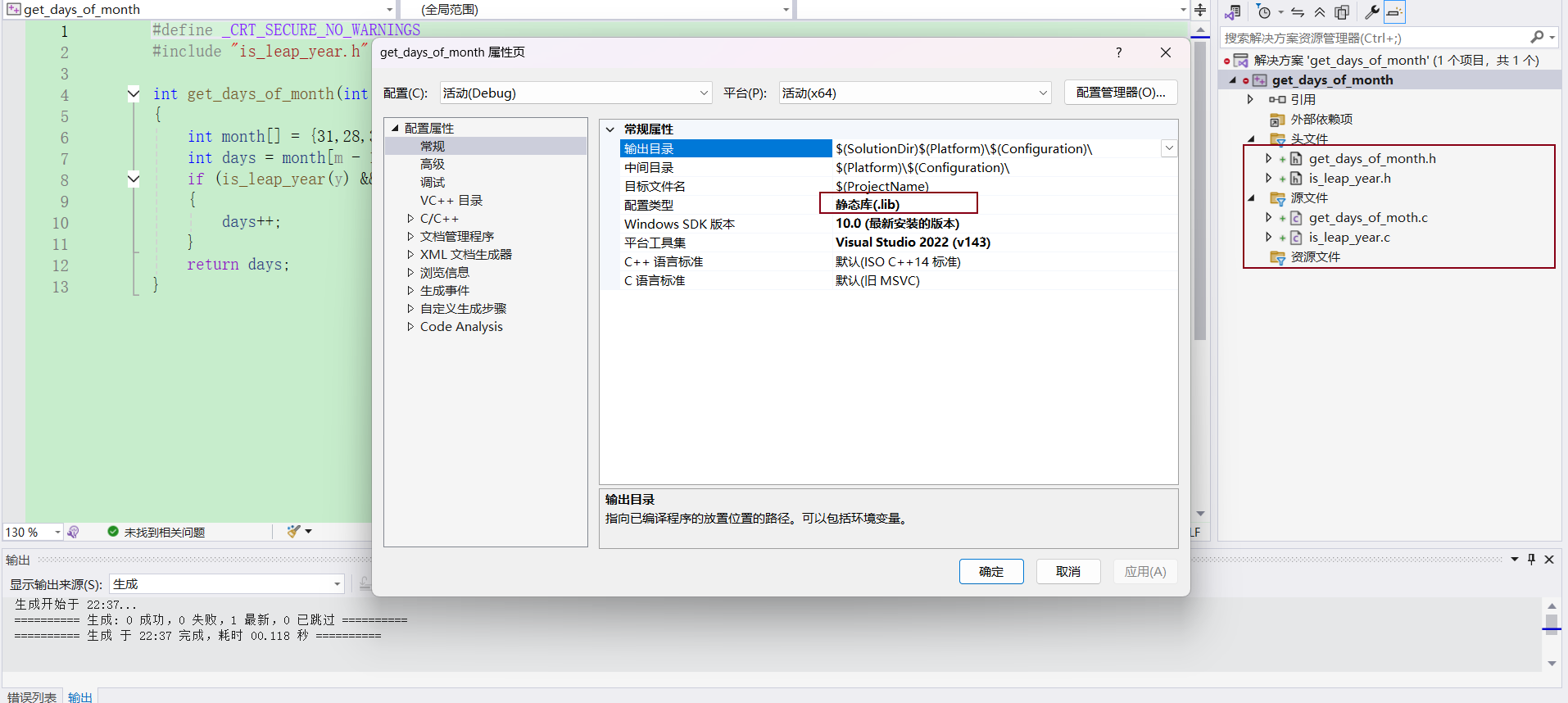
函数头文件和源文件理论上名称可以不同，但是为了阅读代码方便，尽量保持一致。头文件里面也可以声明多个函数，根据自己需要进行选择。

把代码分成几个文件，除了写代码方便协作以外，更大的好处是可以对代码进行一定的隐藏。比如：你写了一个比较厉害的代码，把代码卖给某个公司使用，但是又不想把源码暴露给对方，这时候可以将.h或者是.c文件编译成静态库文件.lib，将lib文件卖给该公司。

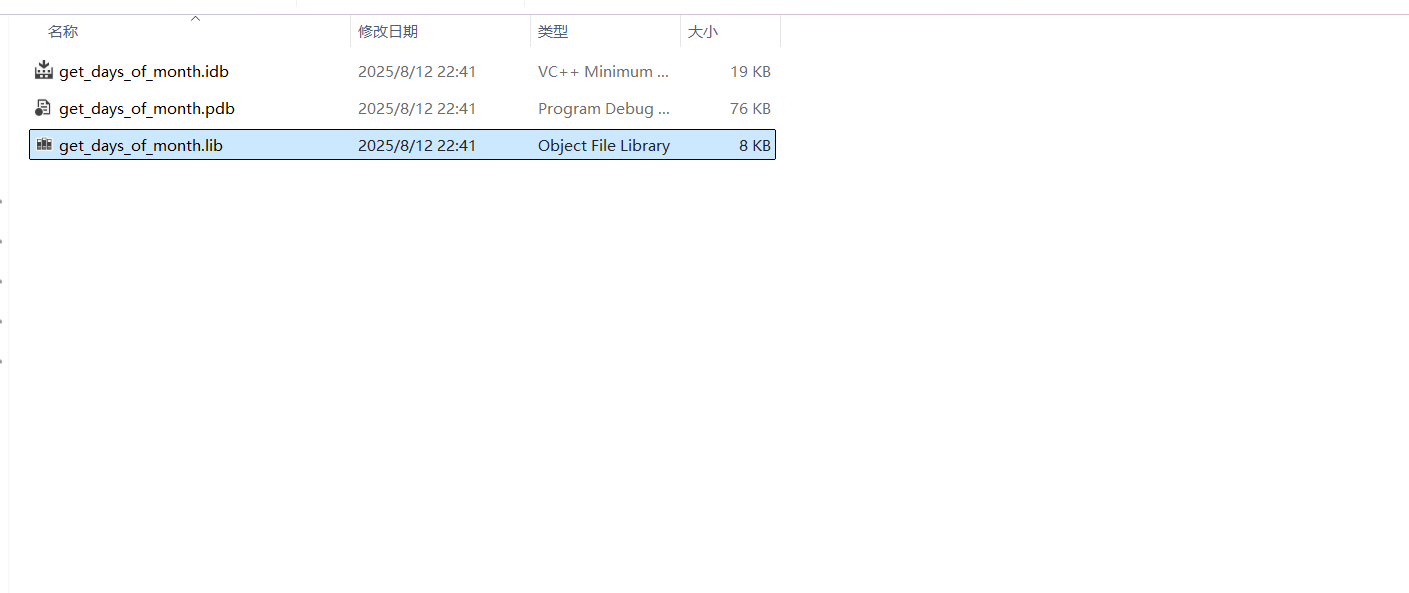
8.3 静态库：

第一步：卖出去

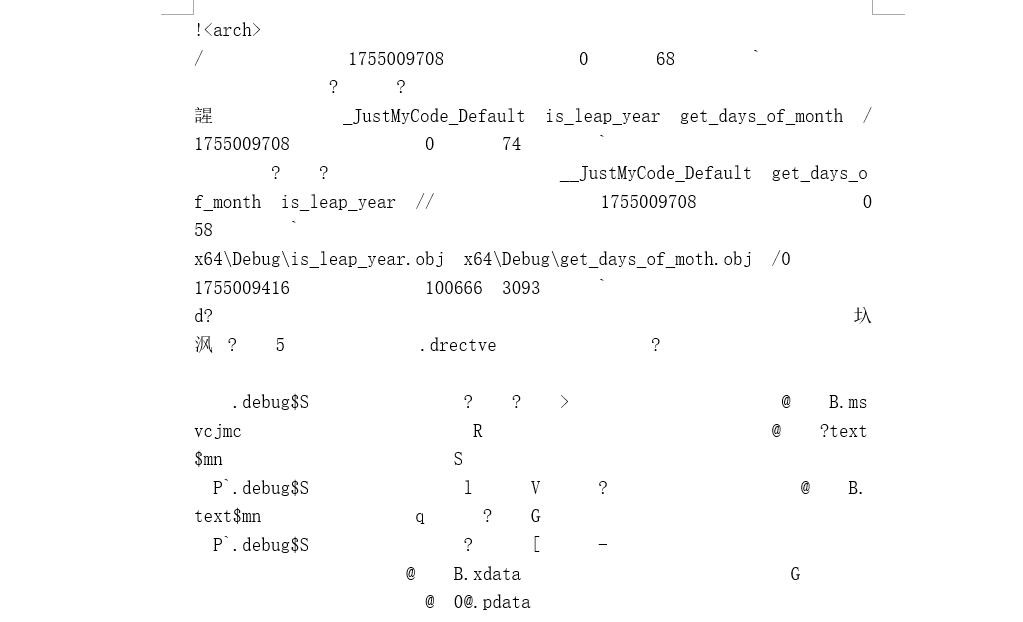
生成lib文件：



Ctrl+F5生成lib文件：



直接打开文件是乱码：

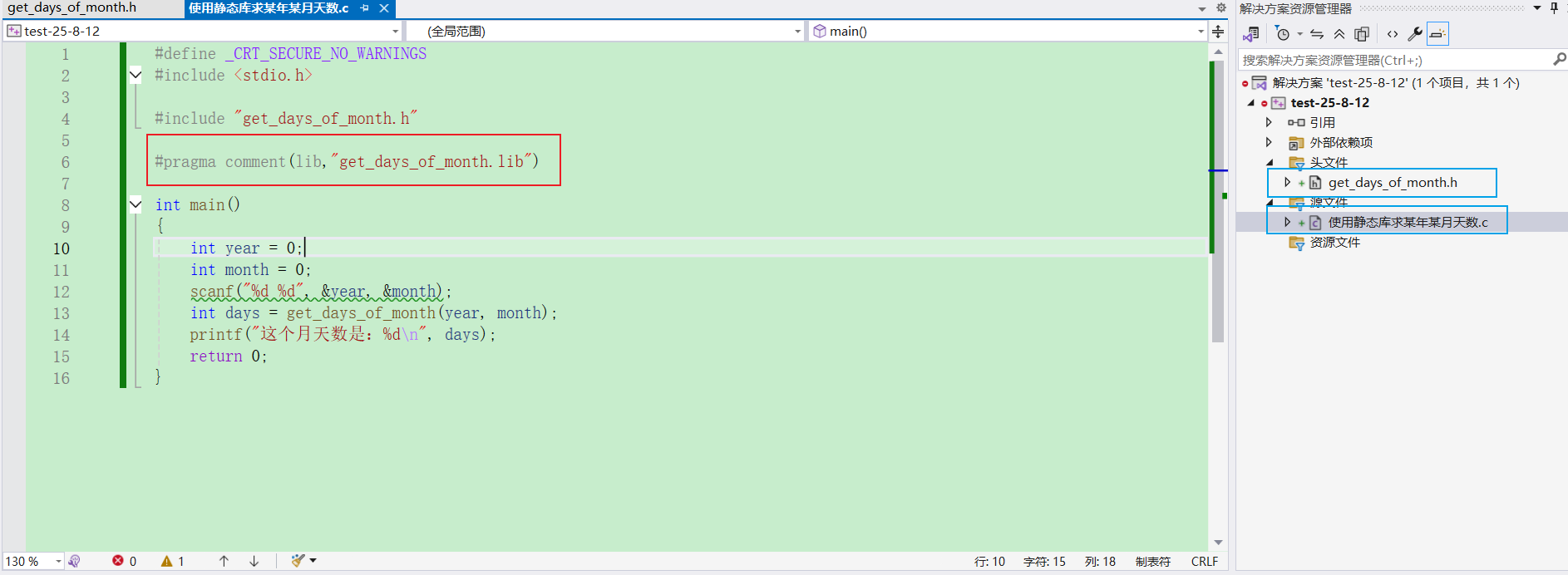


第二步：买家使用

复制粘贴至目标文件夹：



编译器打开目标文件：



导入目标头文件，使用pragma comment解析静态库，运行代码：

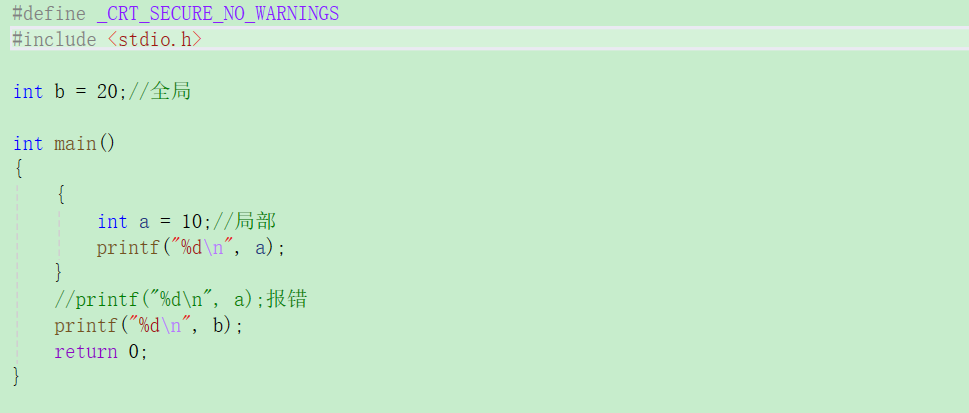


8.3 static和extern关键字

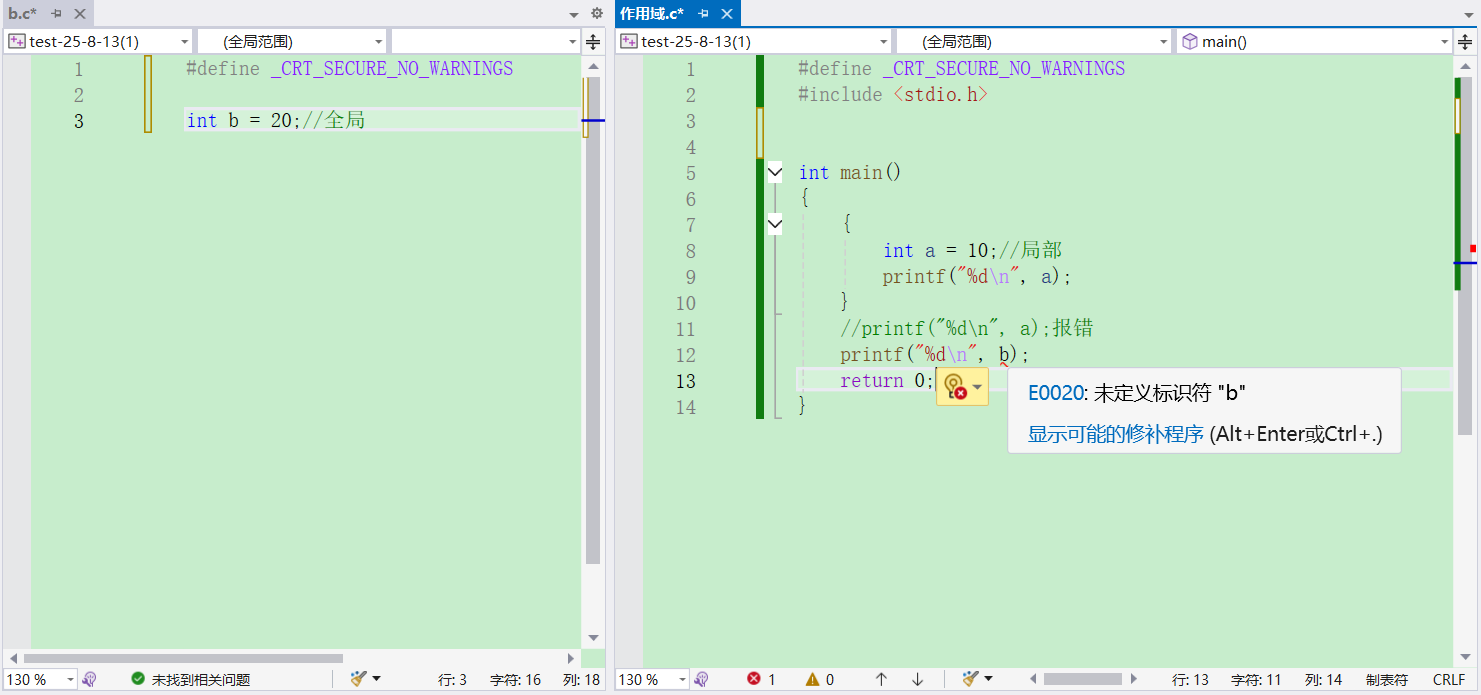
static：静态的意思，用来修饰局部变量、全局变量及函数。

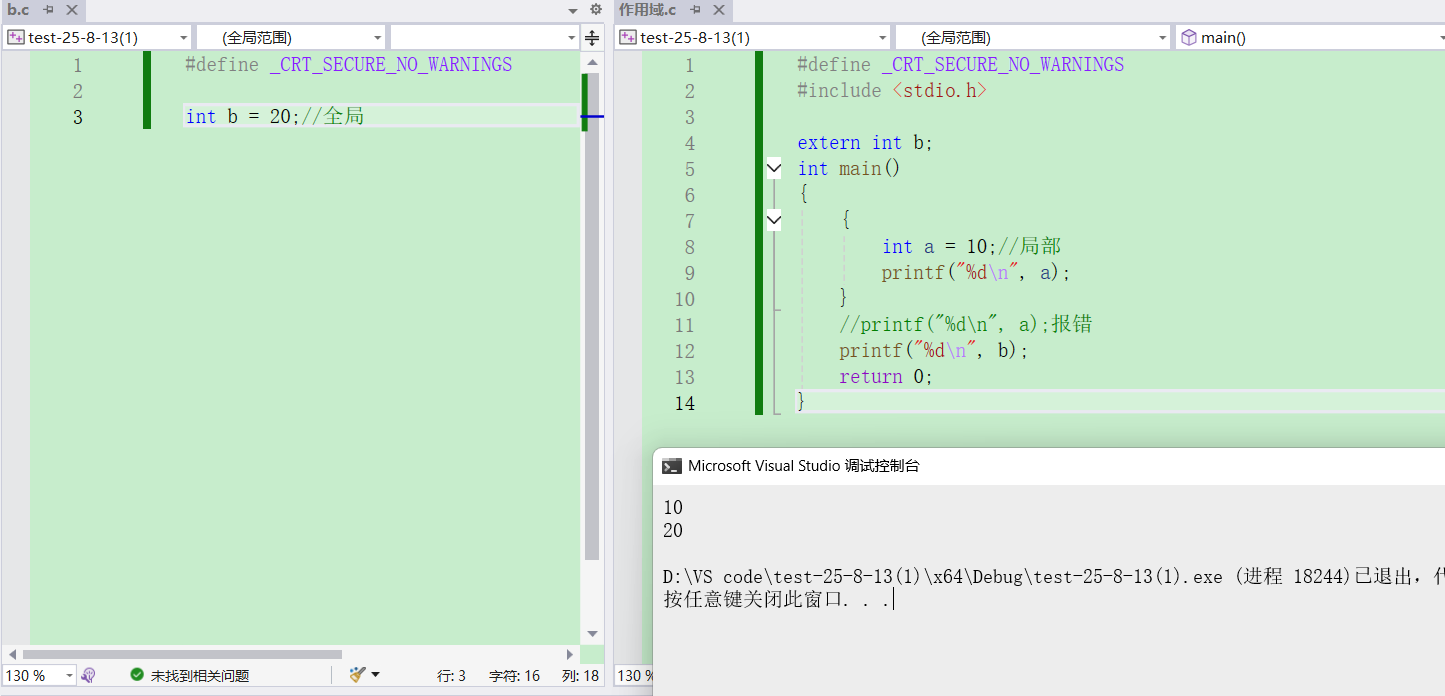
extern：用来声明外部符号。

作用域（scope）：一个变量在哪里可以使用，哪里就是它的作用域。局部变量的作用域是变量所在的局部范围，全局变量的作用域是整个工程。



当我们把int b = 0;这个全局变量放在另外一个.c文件中，该怎么在这个.c文件中使用呢？这就需要用到extern关键字。





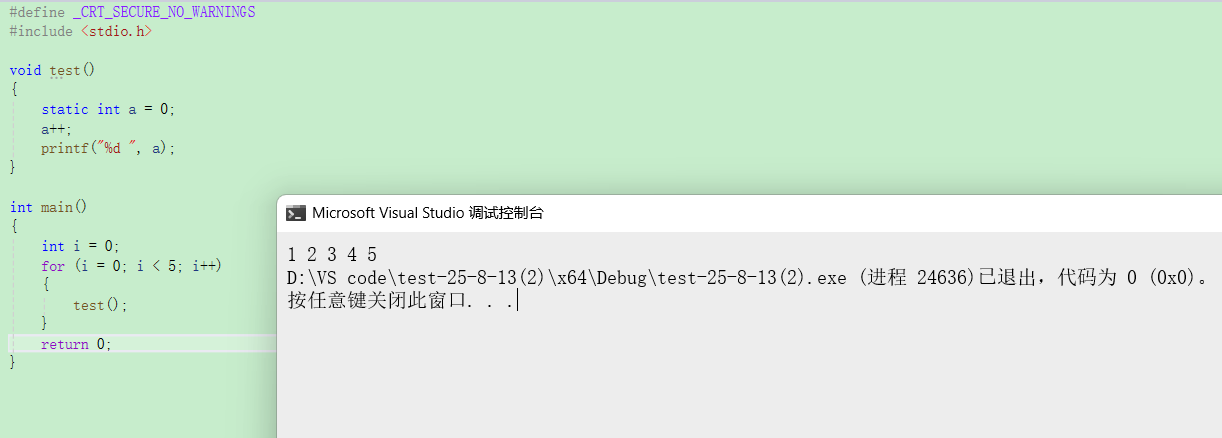
未声明，会报错。声明后，代码正常运行。

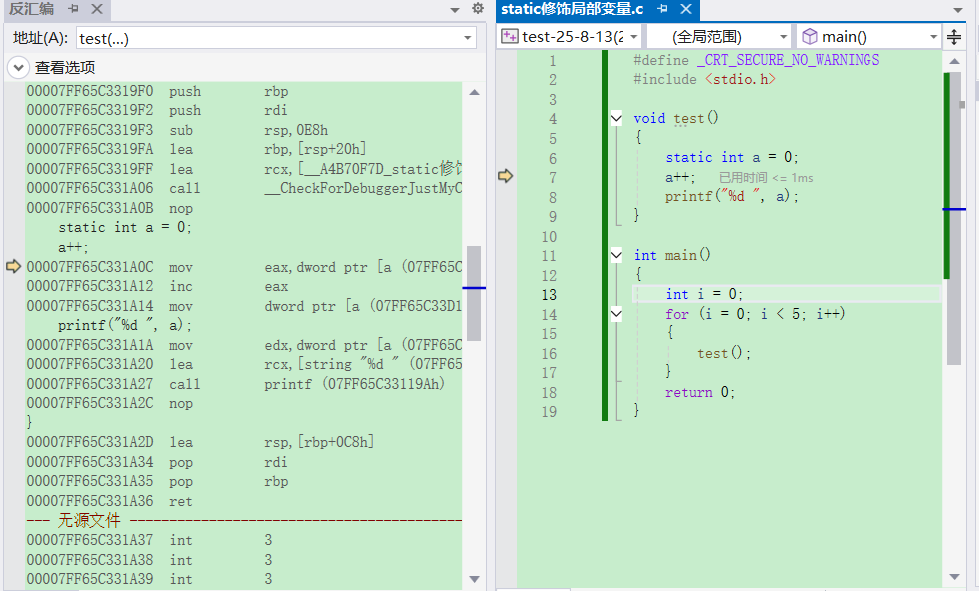
生命周期：变量的创建（申请内存）到变量的销毁（回收内存）之间的时间段。局部变量的生命周期是进入作用域开始，出作用域结束。全局变量的生命周期是整个程序的生命周期。

8.3.1 static修饰局部变量



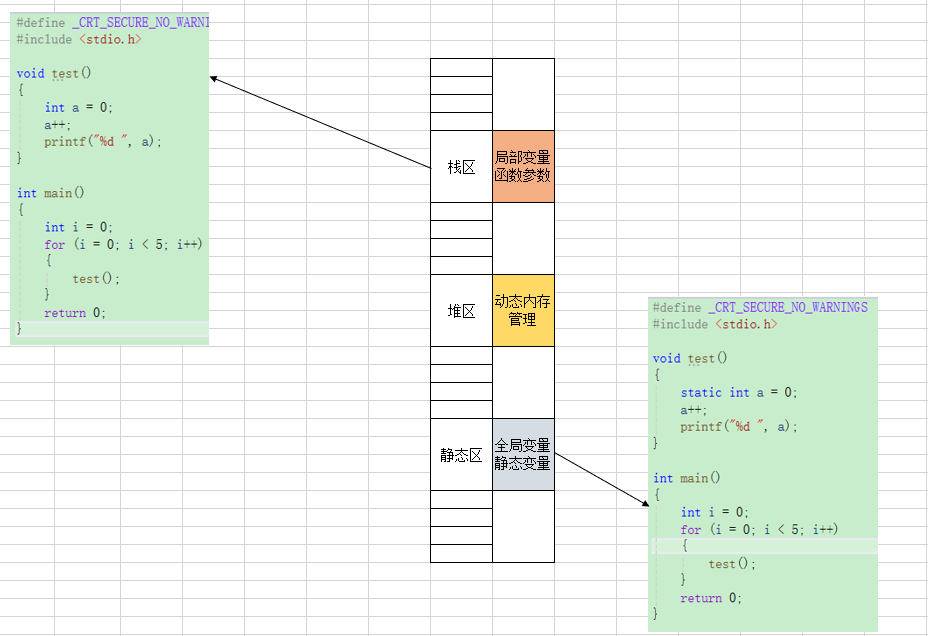
static修饰局部变量之前，局部变量a是每次进⼊test函数先创建（生命周期开始）并赋值为0，然后++，再打印，出函数的时候变量a生命周期结束（释放内存）。





static修饰局部变量后，该语句不会再执行（没有翻译成对应的汇编指令），也就是说变量a在每一次进入test函数时不会重新创建，a在编译的时候就已经创建并赋值0，出函数的时候是不会销毁，重新进入直接上次累积的数值继续计算。

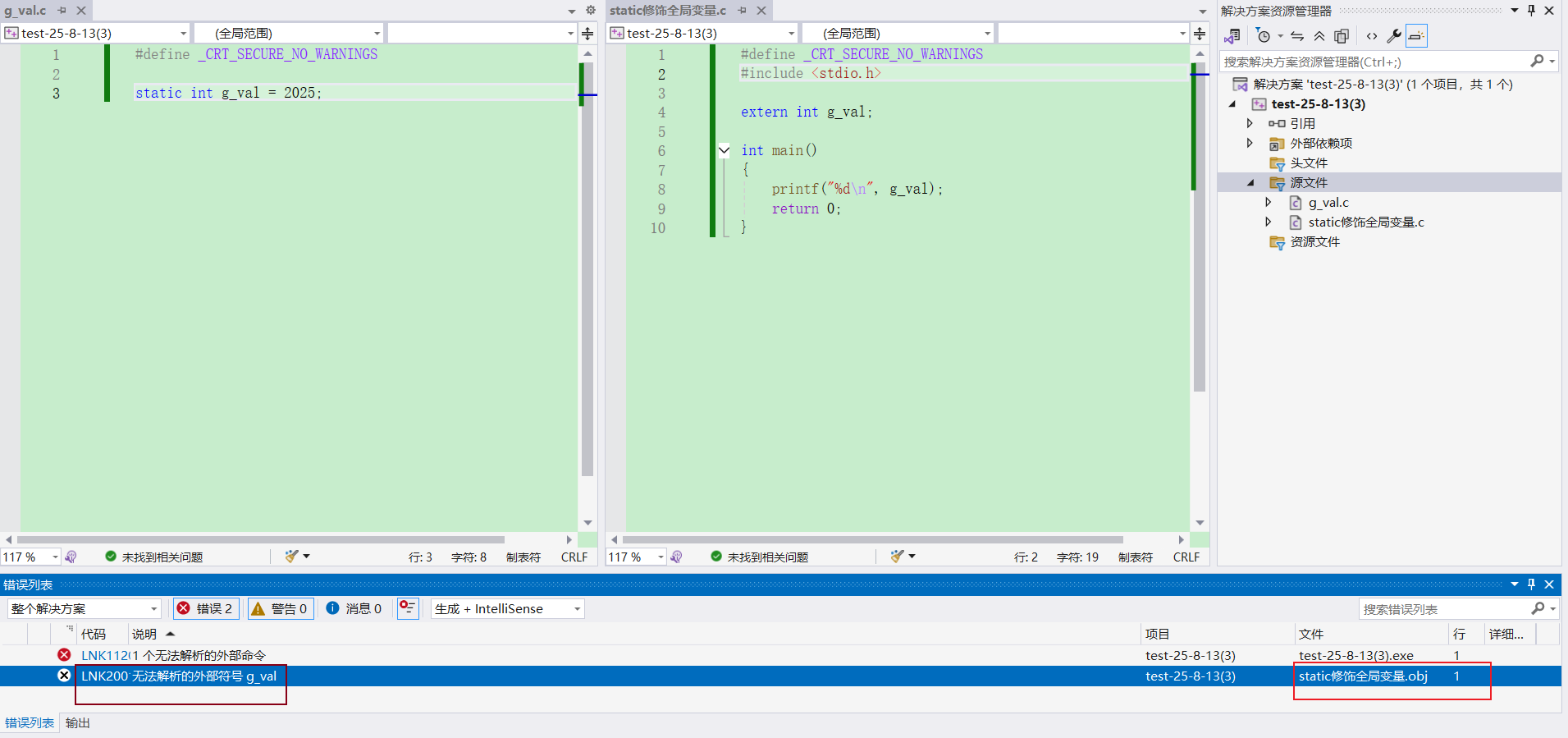
static修饰局部变量改变了变量的生命周期。本质：改变了变量的存储类型，本来局部变量是存储在内存的栈区的，但是被static修饰后存储到了静态区。存储在静态区的变量和全局变量一样，生命周期就和程序的生命周期一样，只有程序结束，变量才销毁，内存才回收。但是作用域不变。



如何使用：一个变量出了函数后，我们还想保留值，等下次进⼊函数继续使用，就可以使用static。

8.3.2 static修饰全局变量

全局变量默认具有外部链接属性：在另外一个源文件创建，需要在该文件中使用时只需要声明外部符号即可，就像上面代码一样。



static修饰全局变量后，全局变量外部链接属性就变成内部链接属性，其他.c文件就无法使用，作用域变小。

如何使用：如果一个全局变量，只想在所在的源文件内部使用，不想被其他文件发现，就可以使用static修饰。

8.3.3 static修饰函数

static修饰函数和static修饰全局变量是一样的，函数也具有外部链接属性，被static修饰的时候外部链接属性同样变为内部链接属性。

