

# 第8讲：VS实用调试技巧

## 目录

1. 什么是bug?
2. 什么是调试?
3. debug和release
4. VS调试快捷键
5. 监视和内存观察
6. 调试举例1
7. 调试举例2
8. 调试举例3：扫雷
9. 编程常见错误归类

---

## 正文开始

### 1. 什么是 bug

bug本意是“昆虫”或“虫子”，现在一般是指在电脑系统或程序中，隐藏着的一些未被发现的缺陷或问题，简称**程序漏洞**。

“Bug”的创始人格蕾丝·赫柏（Grace Murray Hopper），她是一位为美国海军工作的电脑专家，1947年9月9日，格蕾丝·赫柏对Harvard Mark II设置好17000个继电器进行编程后，技术人员正在进行整机运行时，它突然停止了工作。于是他们爬上去找原因，发现这台巨大的计算机内部一组继电器的触点之间有一只飞蛾，这显然是由于飞蛾受光和热的吸引，飞到了触点上，然后被高电压击死。所以在报告中，赫柏用胶条贴上飞蛾，并把“bug”来表示“一个在电脑程序里的错误”，“Bug”这个说法一直沿用到今天。

9/9

0800 Antan started  
 1000 " stopped - antan ✓  
 1300 (032) MP-MC 1.2700 9.037 847 025  
 (033) PRO 2 2.130476415 9.037 846 995 correct  
 correct 2.130676415  
 Relays 6-2 in 033 failed special speed test  
 in relay 10.000 test.  
 Relays changed  
 1100 Started Cosine Tapc (Sine check)  
 1525 Started Multy Adder Test.  
 1545 Relay #70 Panel F (moth) in relay.  
 First actual case of bug being found.  
 1630 Antan started.  
 1700 closed down.

历史上的第一个bug (图片来自网络)

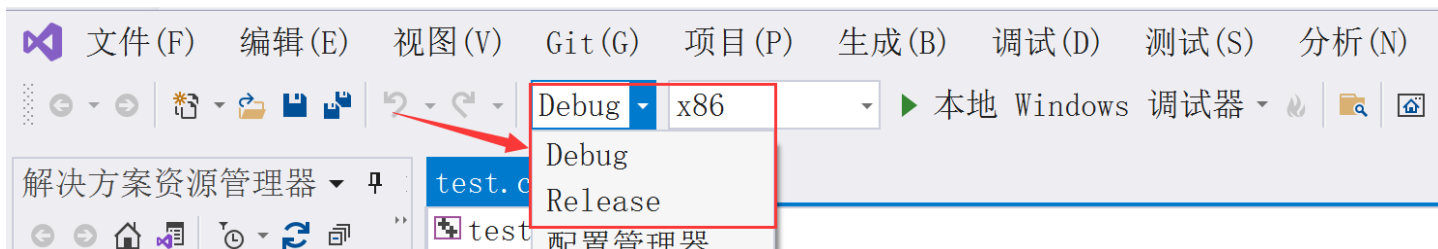
## 2. 什么是调试 (debug) ?

当我们发现程序中存在的问题的时候,那下一步就是找到问题,并修复问题。

这个找问题的过程称为**调试**,英文叫**debug** (消灭bug) 的意思。

调试一个程序,首先是**承认出现了问题**,然后通过各种手段去定位问题的位置,可能是逐过程的调试,也可能是隔离和屏蔽代码的方式,找到问题所的位置,然后确定错误产生的原因,再修复代码,重新测试。

## 3. Debug 和 Release



在VS上编写代码的时候,就能看到有 **debug** 和 **release** 两个选项,分别是什么意思呢?

比特就业课主页: <https://m.cctalk.com/inst/s9yewhfr>  
**Debug 通常称为调试版本**，它包含调试信息，并且不作任何优化，便于程序员调试程序；

程序员在写代码的时候，需要经常性的调试代码，就将这里设置为 `debug`，这样编译产生的是 `debug` 版本的可执行程序，其中包含调试信息，是可以直接调试的。

**Release 称为发布版本**，它往往是进行了各种优化，使得程序在代码大小和运行速度上都是最优的，以便用户很好地使用。当程序员写完代码，测试再对程序进行测试，直到程序的质量符合交付给用户使用的标准，这个时候就会设置为 `release`，编译产生的就是 `release` 版本的可执行程序，这个版本是用户使用的，无需包含调试信息等。

Data (D:) > code > 2023 > test > 课件代码测试 > Release		
名称	大小	
课件代码测试.exe	9 KB	
课件代码测试.pdb	396 KB	

Data (D:) > code > 2023 > test > 课件代码测试 > Debug		
名称	大小	
课件代码测试.exe	39 KB	
课件代码测试.pdb	1,092 KB	

release版本和debug版本的对比

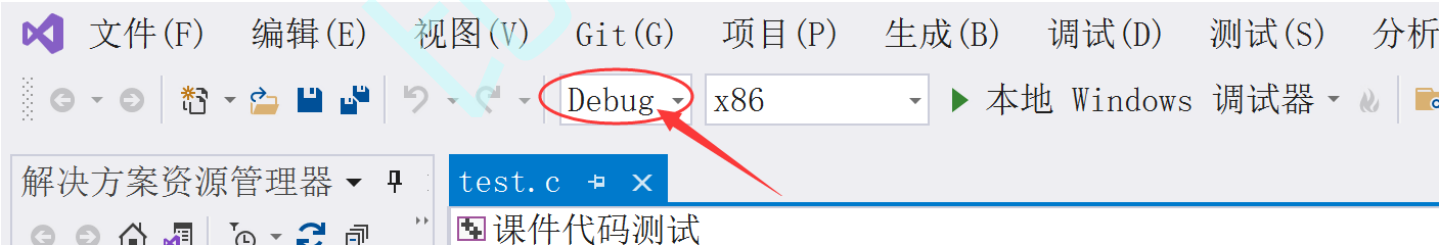
对比可以看到从同一段代码，编译生成的可执行文件的大小，`release`版本明显要小，而`debug`版本明显大。

## 4. VS调试快捷键

那程序员怎么调试代码呢？

### 4.1 环境准备

首先是环境的准备，需要一个支持调试的开发环境，我们上课使用VS，应该把VS上设置为**Debug**，如图：



### 4.2 调试快捷键

调试最常使用的几个快捷键：

**F9：**创建断点和取消断点

**断点**的作用是可以程序的任意位置设置断点，打上断点就可以使得程序执行到想要的位置暂停执行，接下来我们就可以使用F10，F11这些快捷键，观察代码的执行细节。

**条件断点：**满足这个条件，才触发断点

比特就业课主页: <https://m.cctalk.com/inst/s9yewhfr>

**F5:** 启动调试, 经常用来直接跳到下一个断点处, 一般是和F9配合使用。

**F10:** 逐过程, 通常用来处理一个过程, 一个过程可以是一次函数调用, 或者是一条语句。

**F11:** 逐语句, 就是每次都执行一条语句, 但是这个快捷键可以使我们的执行逻辑**进入函数内部**。在函数调用的地方, 想进入函数观察细节, 必须使用F11, 如果使用F10, 直接完成函数调用。

**CTRL + F5:** 开始执行不调试, 如果你想让程序直接运行起来而不调试就可以直接使用。

VS更多快捷键了解: <http://blog.csdn.net/mrlisky/article/details/72622009>

## 5. 监视和内存观察

在调试的过程中我们, 如果要观察代码执行过程中, 上下文环境中的变量的值, 有哪些方法呢?

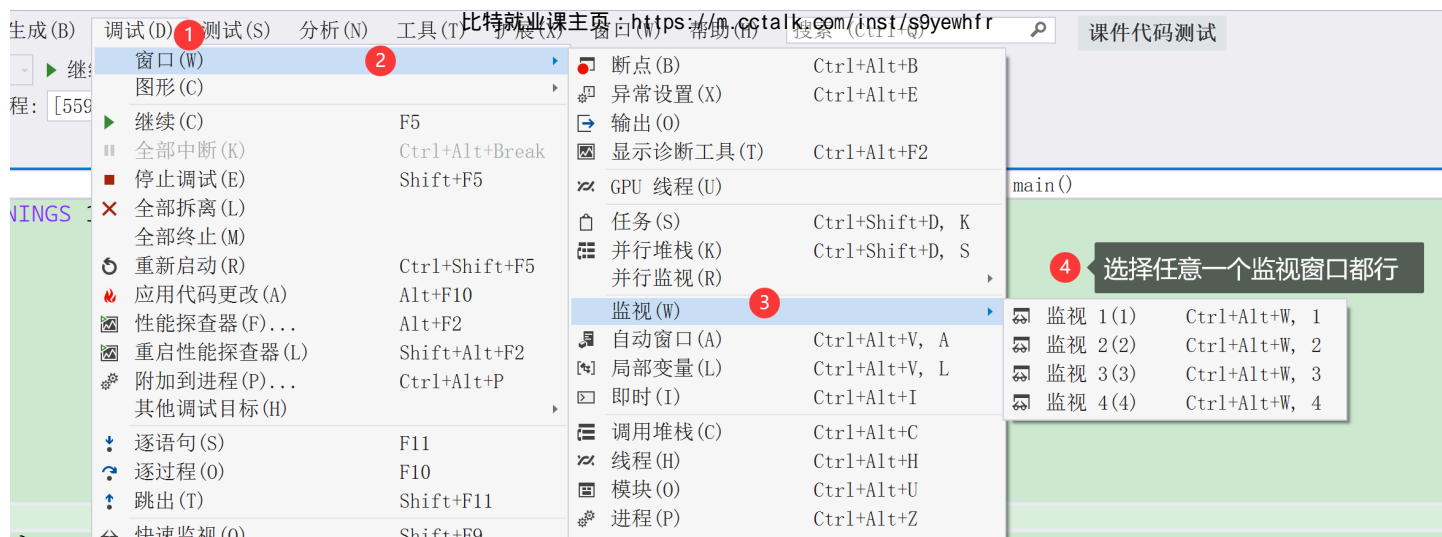
这些观察的前提条件一定是开始调试后观察, 比如:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int arr[10] = { 0 };
6      int num = 100;
7      char c = 'w';
8
9      int i = 0;
10     for (i = 0; i < 10; i++)
11     {
12         arr[i] = i;
13     }
14     return 0;
15 }
```

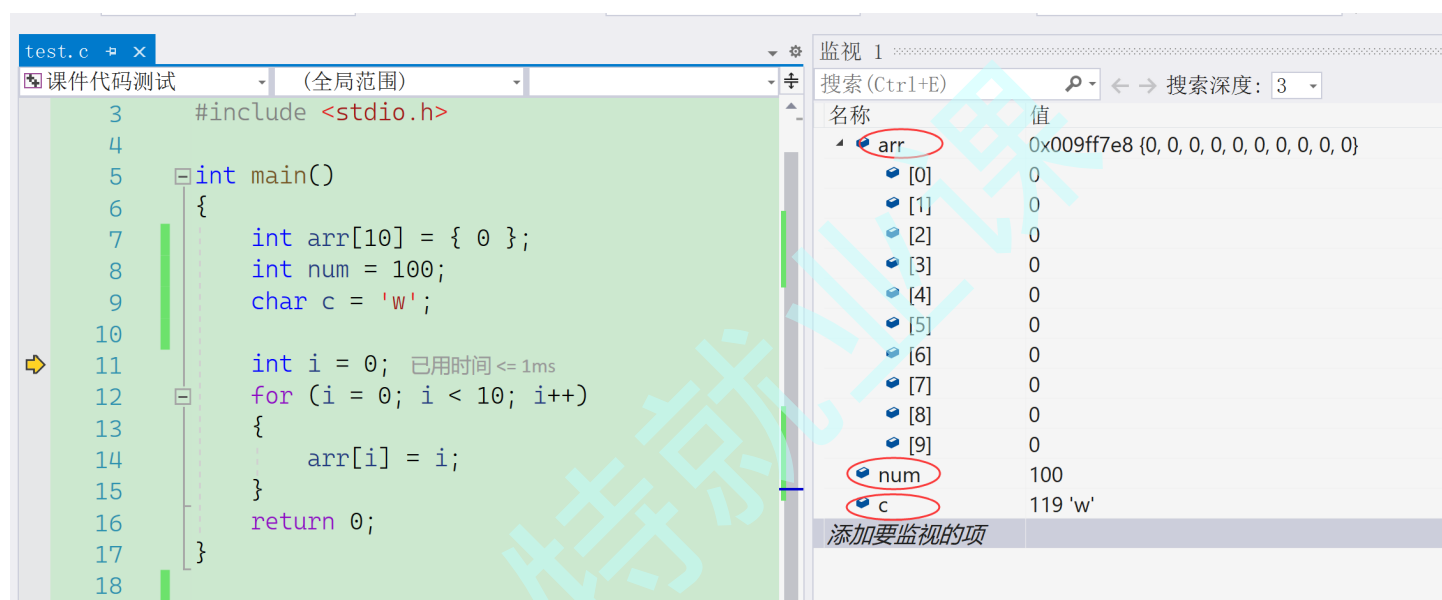
### 5.1 监视

开始调试后, 在菜单栏中【调试】->【窗口】->【监视】, 打开任意一个监视窗口, 输入想要观察的对象就行。

打开监视窗口:



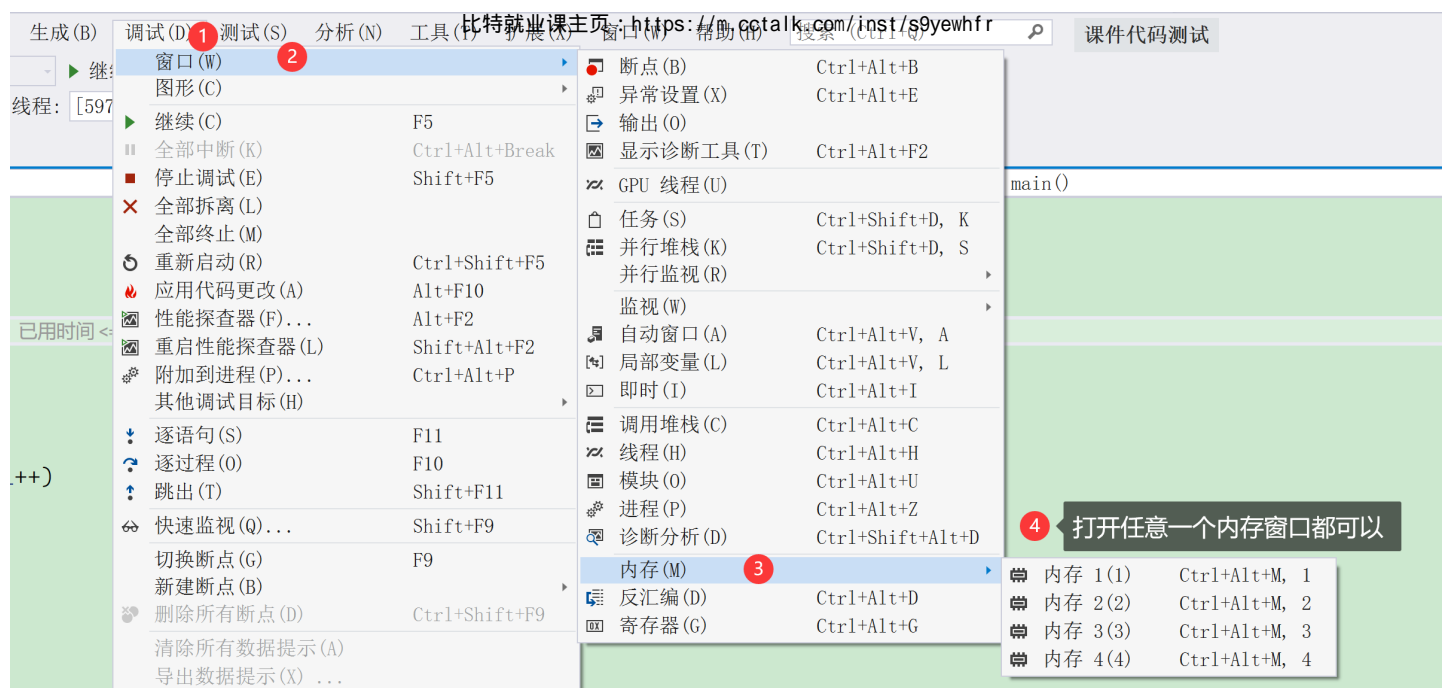
在监视窗口中观察：



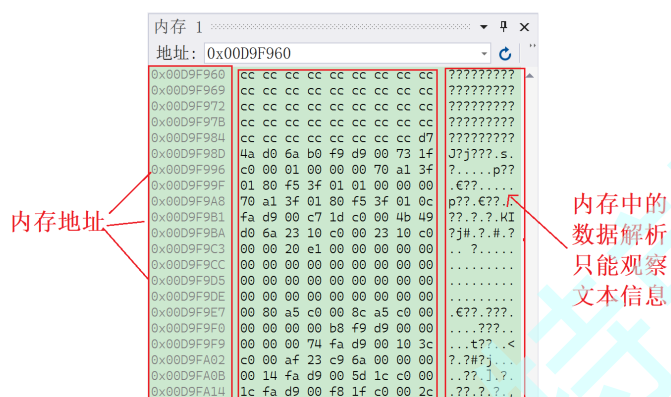
## 5.2 内存

如果监视窗口看的不够仔细，也是可以观察变量在内存中的存储情况，还是在【调试】->【窗口】->【内存】

打开内存窗口：

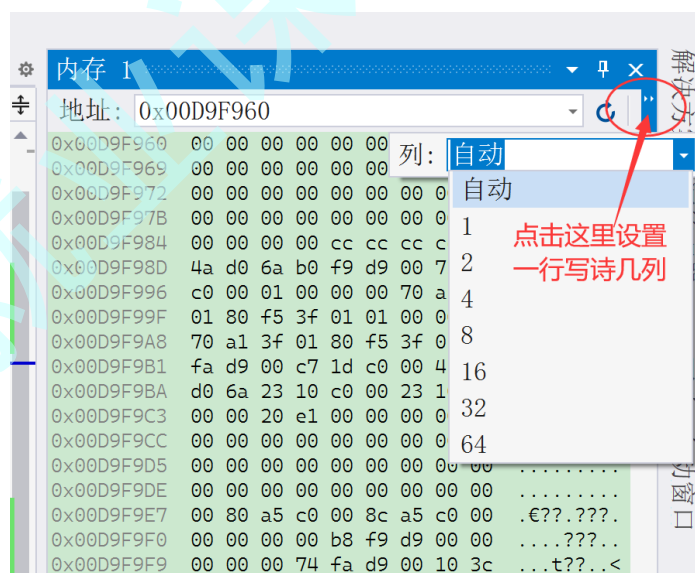


在内存窗口中观察数据:



内存中的数据  
16进制展示

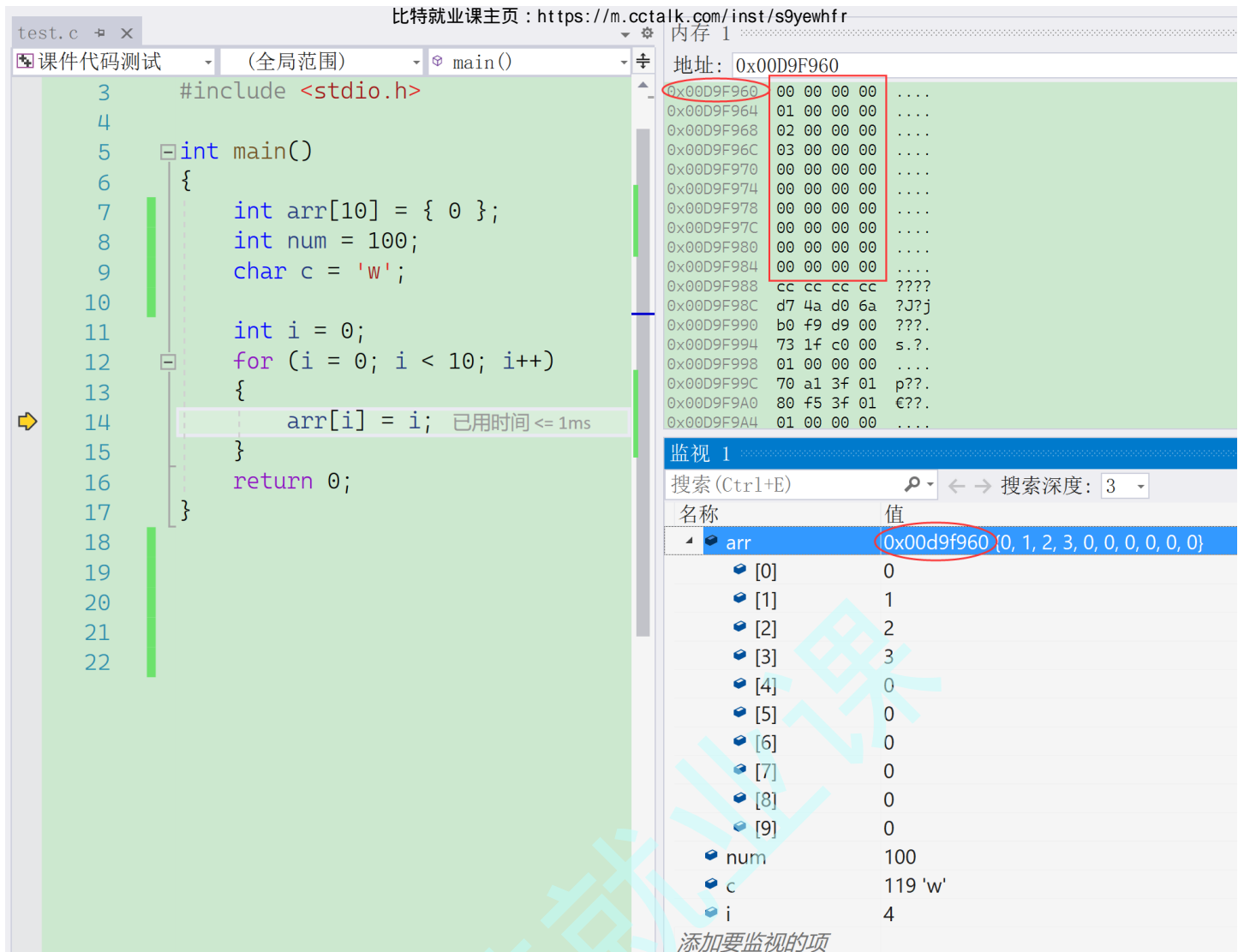
内存窗口怎么看



内存窗口设置列

在打开内存窗口后, 要在地址栏输入: arr, &num, &c, 这类地址, 就能观察到该地址处的数据。





调试内存窗口演示

除此之外，在调试的窗口中还有：自动窗口，局部变量，反汇编、寄存器等窗口，自行验证使用一下。

## 6. 调试举例1

求  $1!+2!+3!+4!+\dots+10!$  的和，请看下面的代码：

```

1  #include <stdio.h>
2  //写一个代码求n的阶乘
3  int main()
4  {
5      int n = 0;
6      scanf("%d", &n);
7      int i = 1;
8      int ret = 1;
9      for(i=1; i<=n; i++)
10     {
11         ret *= i;

```

```

12     }
13     printf("%d\n", ret);
14     return 0;
15 }
16
17
18 //如果n分别是1,2,3,4,5...10,求出每个数的阶乘,再求和就好了
19 //在上面的代码上改造
20 int main()
21 {
22     int n = 0;
23     int i = 1;
24     int sum = 0;
25     for(n=1; n<=10; n++)
26     {
27         for(i=1; i<=n; i++)
28         {
29             ret *= i;
30         }
31         sum += ret;
32     }
33     printf("%d\n", sum);
34     return 0;
35 }
36 //运行结果应该是错的?

```

调试找一下问题。

## 7. 调试举例2

在VS2022、X86、Debug 的环境下，编译器不做任何优化的话，下面代码执行的结果是啥？

```

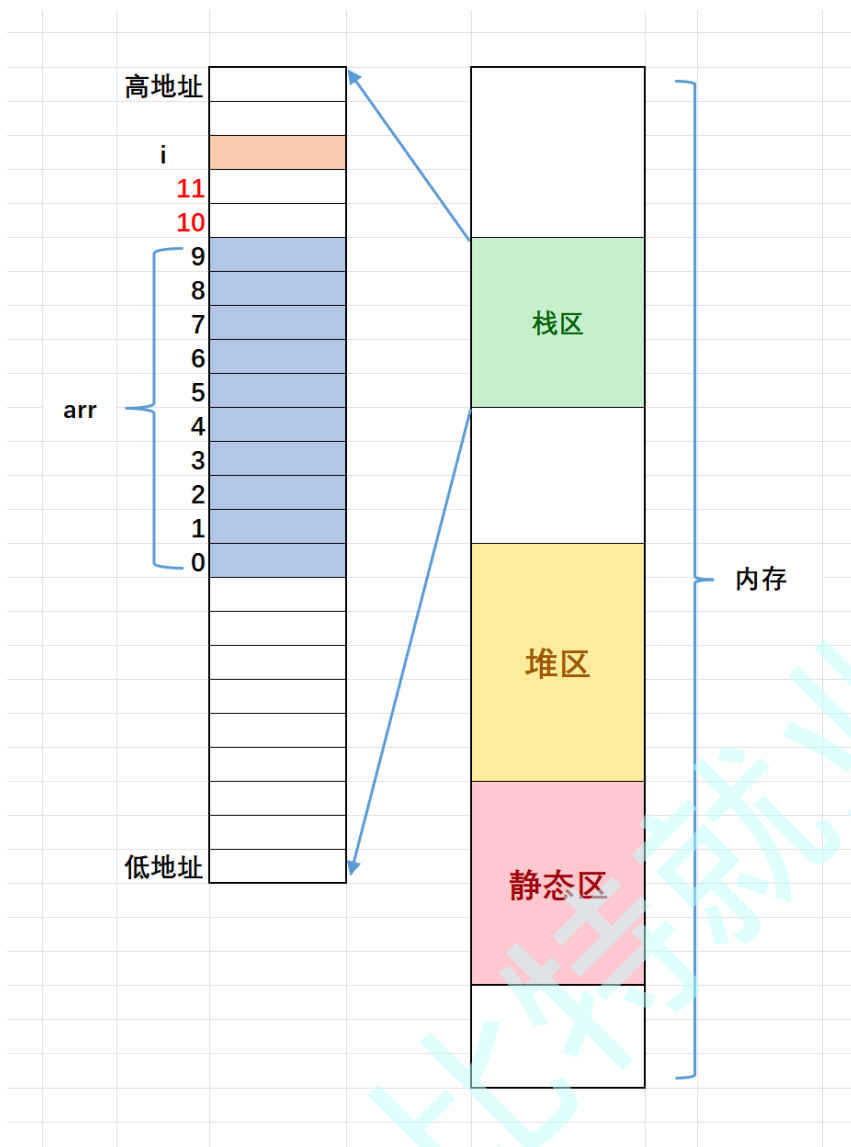
1  #include <stdio.h>
2  int main()
3  {
4      int i = 0;
5      int arr[10] = {1,2,3,4,5,6,7,8,9,10};
6      for(i=0; i<=12; i++)
7      {
8          arr[i] = 0;
9          printf("hehe\n");
10     }
11     return 0;
12 }

```



程序运行，死循环了，调试看看为什么？

调试可以上面程序的内存布局如下：



1. 栈区内存的使用习惯是从高地址向低地址使用的，所以变量*i*的地址是较大的。*arr*数组的地址整体是小于*i*的地址。
2. 数组在内存中的存放是：随着下标的增长，地址是由低到高变化的。

所以根据代码，就能理解为什么是左边的代码布局了。

如果是左边的内存布局，那随着数组下标的增长，往后越界就有可能覆盖到*i*，这样就可能造成死循环的。

这里肯定有同学有疑问：为什么*i*和*arr*数组之间恰好空出来2个整型的空间呢？这里确实是巧合，在不同的编译器下可能中间的空出的空间大小是不一样的，代码中这些变量内存的分配和地址分配是编译器指定的，所以的不同的编译器之间就有差异了。所以这个题目是和环境相关的。

从这个理解我们能够体会到调试的重要性，只有调试才能观察到程序内部执行的细节，就像医生给病人做B超，CT一样。

**注意：**栈区的默认的使用习惯是先使用高地址，再使用低地址的空间，但是这个具体还是要编译器的实现，比如：

在VS上切换到X64，这个使用的顺序就是相反的，在Release版本的程序中，这个使用的顺序也是相反的。

## 8. 调试举例3：扫雷

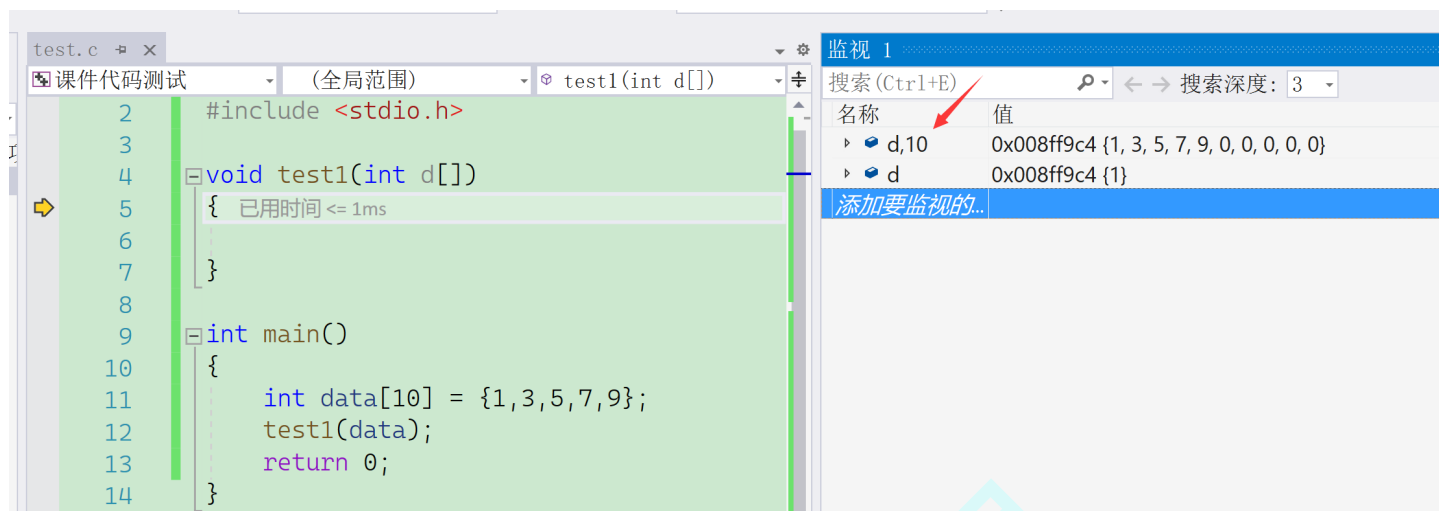
如果一个代码稍微复杂，那怎么调试呢？

这里我们就上手调试一下扫雷的代码。

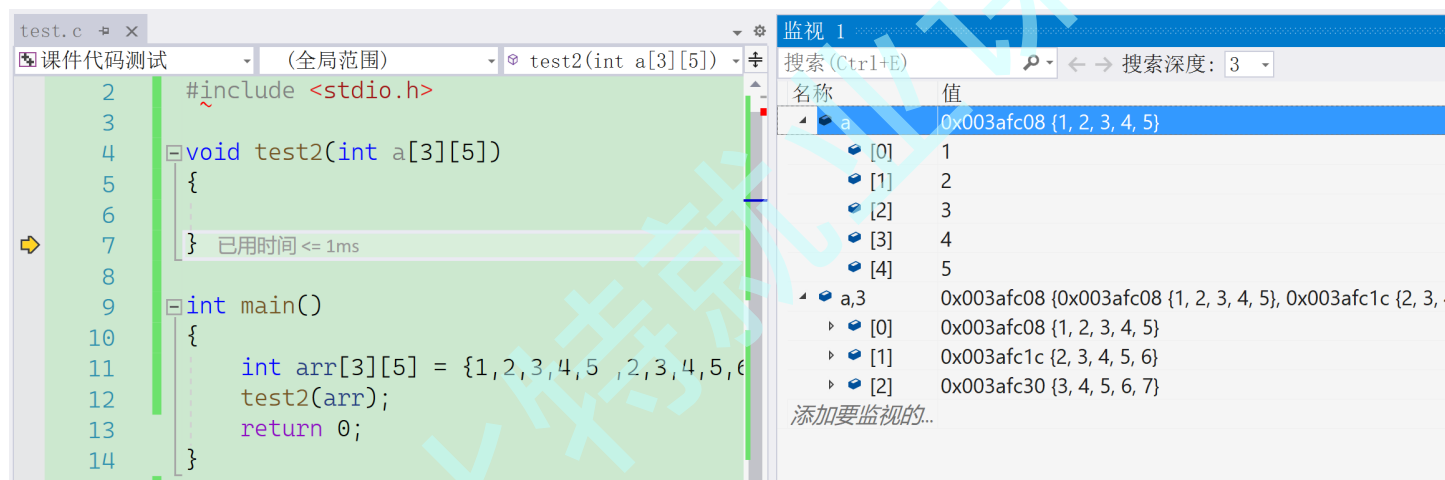
演示：

比特就业课主页：<https://m.cctalk.com/inst/s9yewhfr>

- 在函数内部打断点，快速跳转到函数
- 在数组传参，调试进入函数，如何在监视窗口观察数组的内容：数组名,n 的形式



一维数组通过形参关键数组内容



二维数组通过形参关键数组内容

调试过程中，要做到**心中有数**，也就是程序员自己心里要清晰的知道希望代码怎么执行，然后再去看代码有没有按照我们预定的路线在执行。

**调试**是需要反复去动手练习的，调试是可以增加程序员对代码的理解和掌控的，掌握了调试的能力，就能看到本质，就像能给程序做B超一样，对程序内部一览无余。

## 9. 编程常见错误归类

### 9.1 编译型错误

编译型错误一般都是语法错误，这类错误一般看错误信息就能找到一些蛛丝马迹的，双击错误信息也能初步的跳转到代码错误的地方或者附近。编译错误，随着语言的熟练掌握，会越来越少，也容易解决。

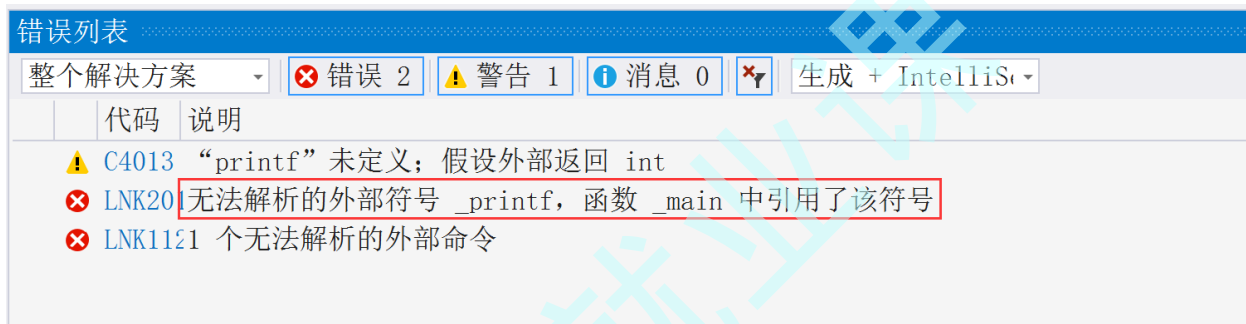
比特就业课主页：<https://m.cctalk.com/inst/s9yewhfr>



## 9.2 链接型错误

看错误提示信息，主要在代码中找到错误信息中的标识符，然后定位问题所在。一般是因为

- 标识符名不存在
- 拼写错误
- 头文件没包含
- 引用的库不存在



## 9.3 运行时错误

运行时错误，是千变万化的，需要借助调试，逐步定位问题，调试解决的是运行时问题。

完