

# 实验报告

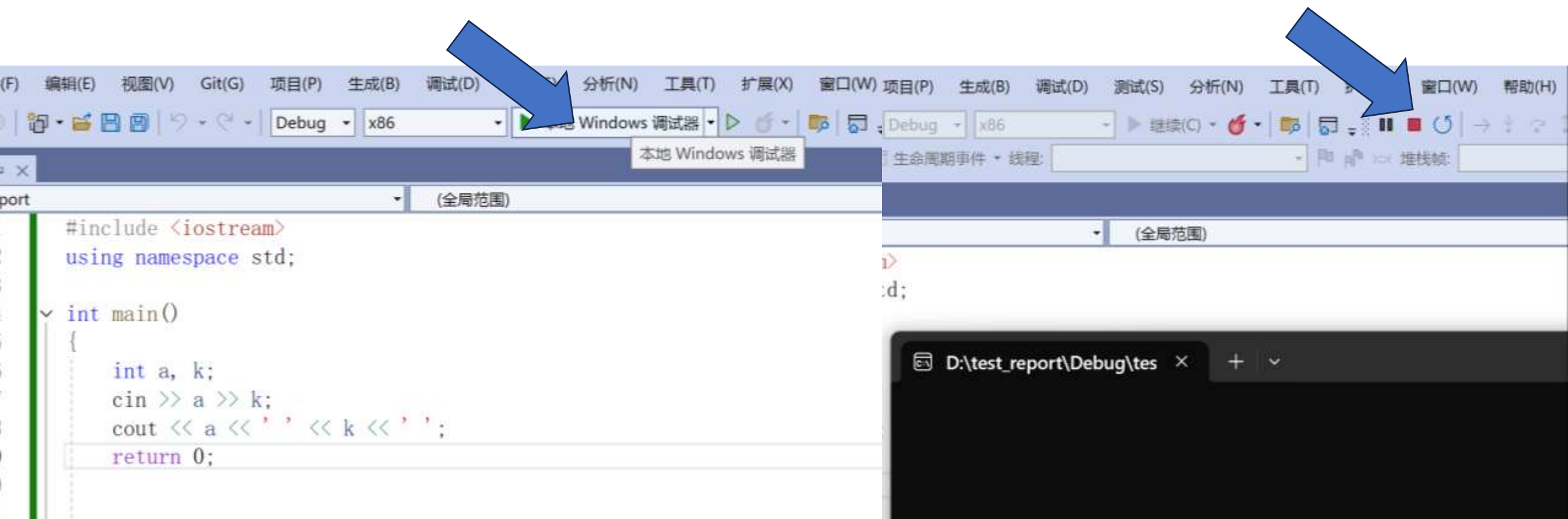
## ——VS2022中调试工具的使用

2452545 刘晴语 计算机 5000244001608

# 1. VS2022下调试工具的基本方法

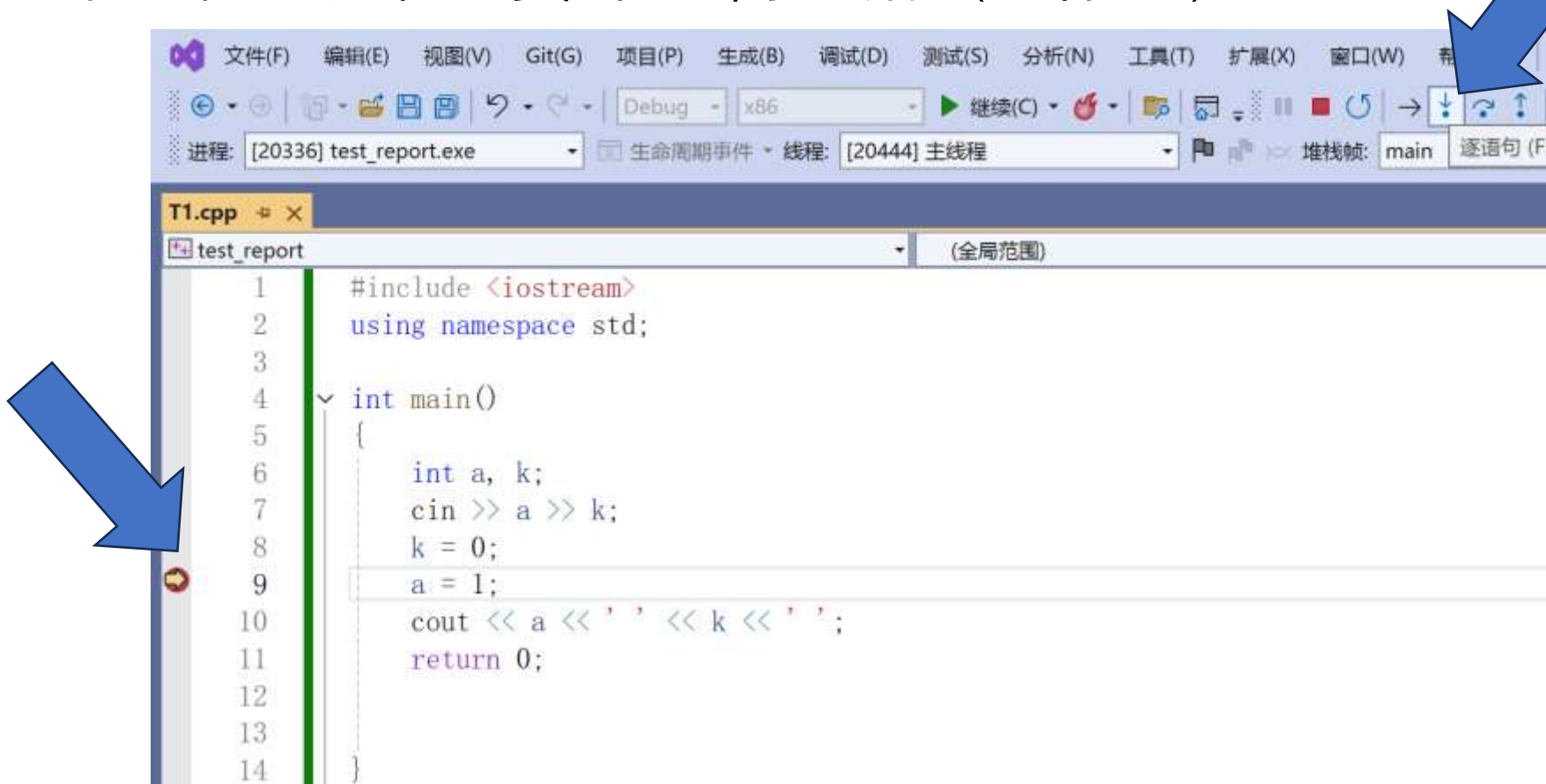
# 1.1 如何开始调试？ 如何结束调试？

- 直接点击“本地Windows调试器”（或者按F5），开始调试
- 进入调试界面后，点击停止键，结束调试



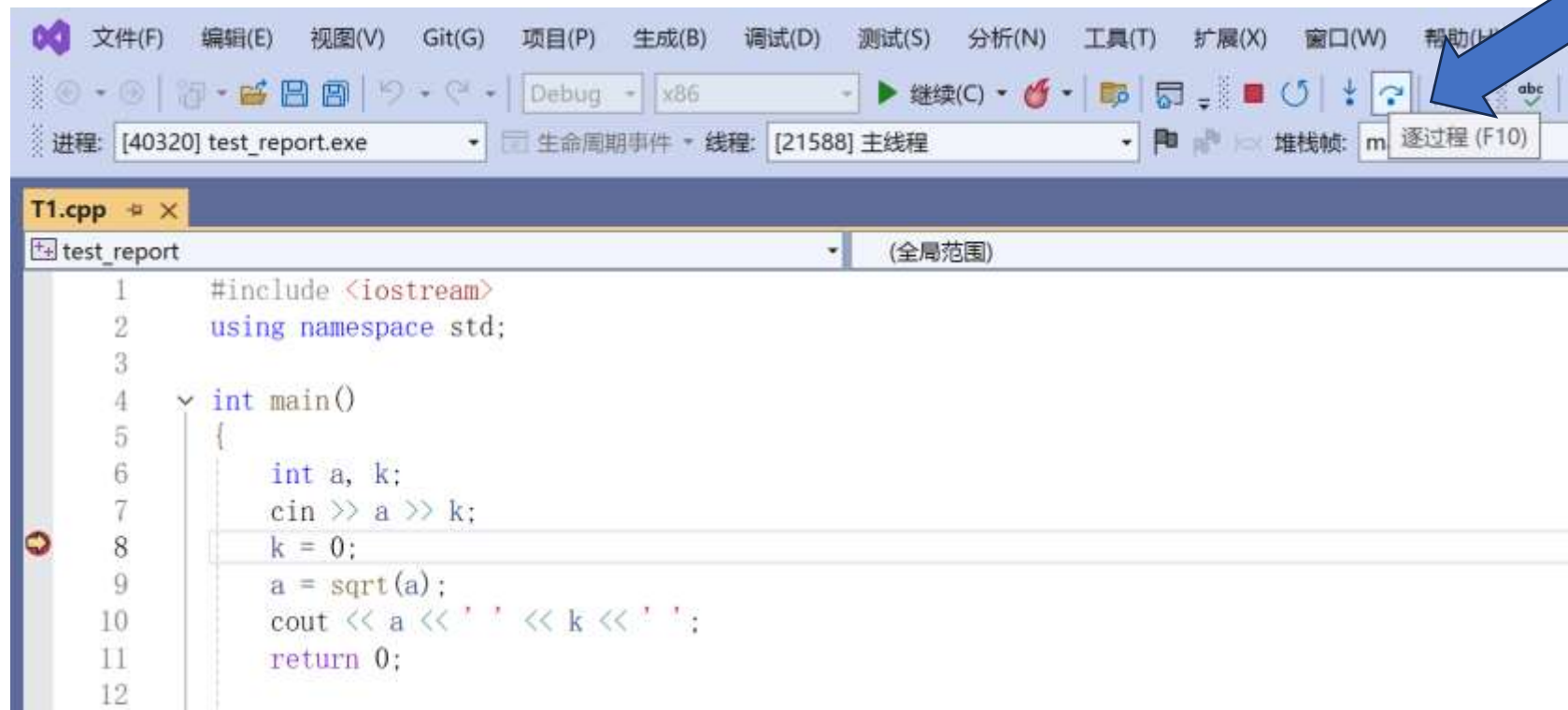
## 1.2 如何在一个函数中每个语句单步执行？

- 先在代码左侧单击设置断点，再按F5进入调试，程序会在第一个断点处暂停
- 点击“逐语句”键即可每个语句单步执行（或者F11）

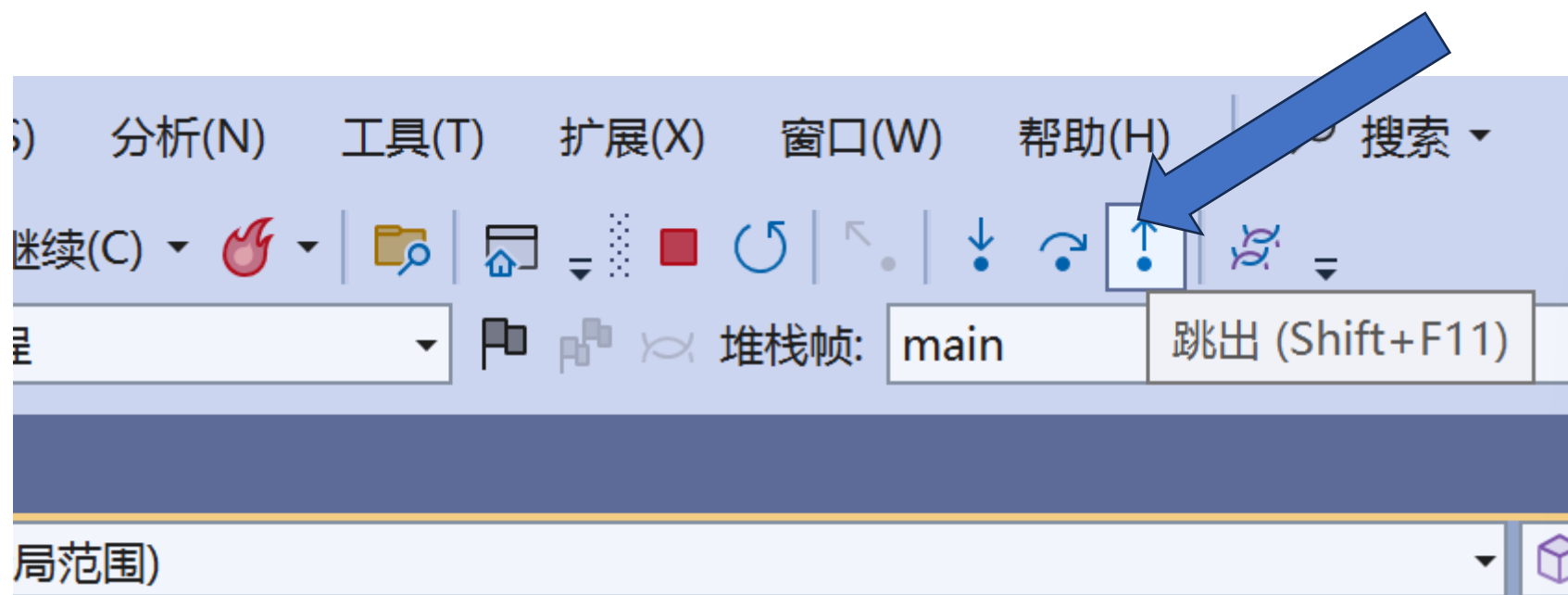


## 1.3 在碰到cout/sqrt等系统函数时，如何一步完成这些系统函数的执行而不进入到这些系统函数的内部单步执行？

- 先在代码左侧单击设置断点，再按F5进入调试，程序会在第一个断点处暂停
- 点击“逐过程”键即可一步完成这些系统函数的执行（或者F10）

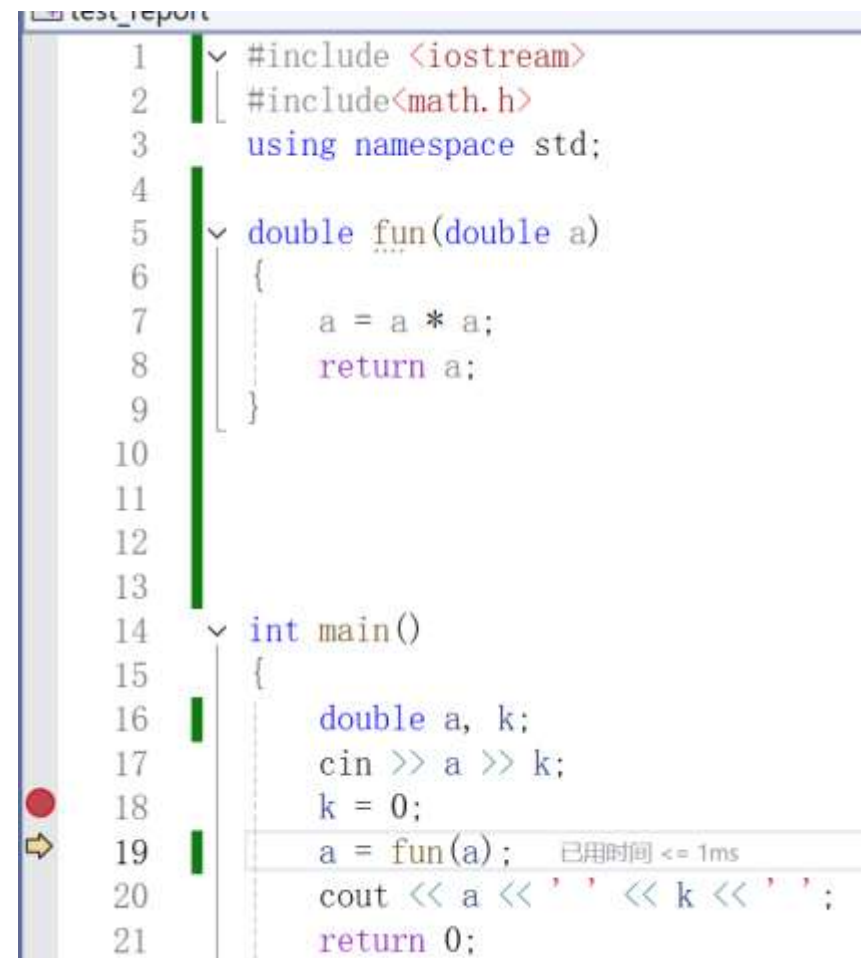
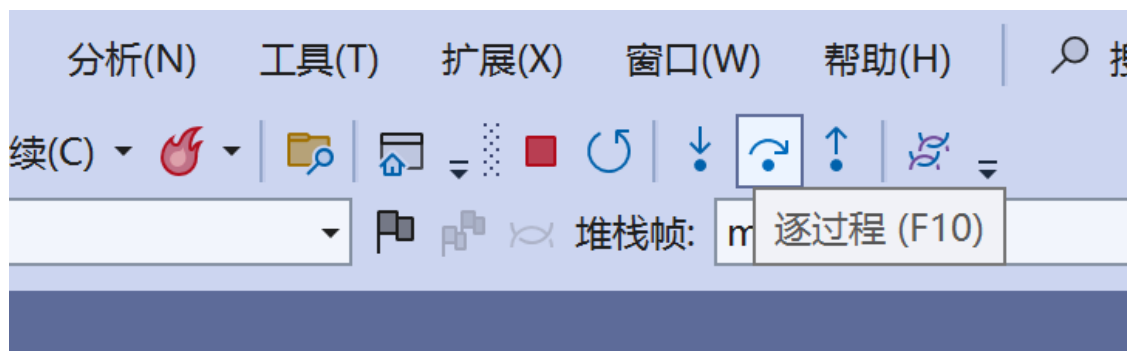


## 1.4 如果已经进入到cout/sqrt等系统函数的内部，如何跳出并返回自己的函数



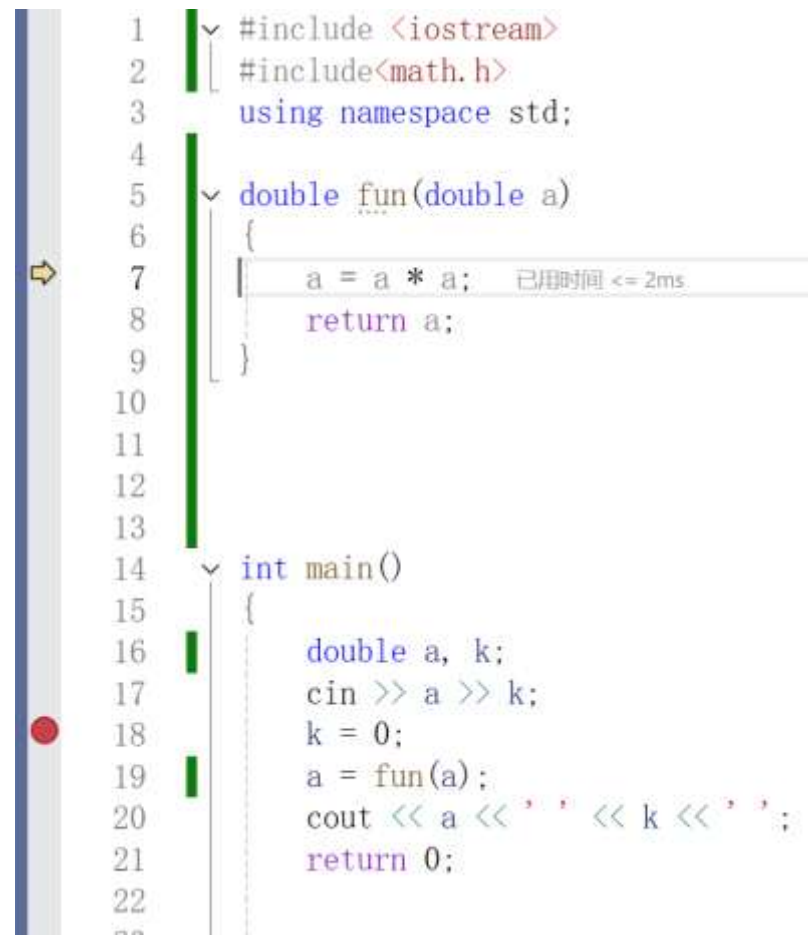
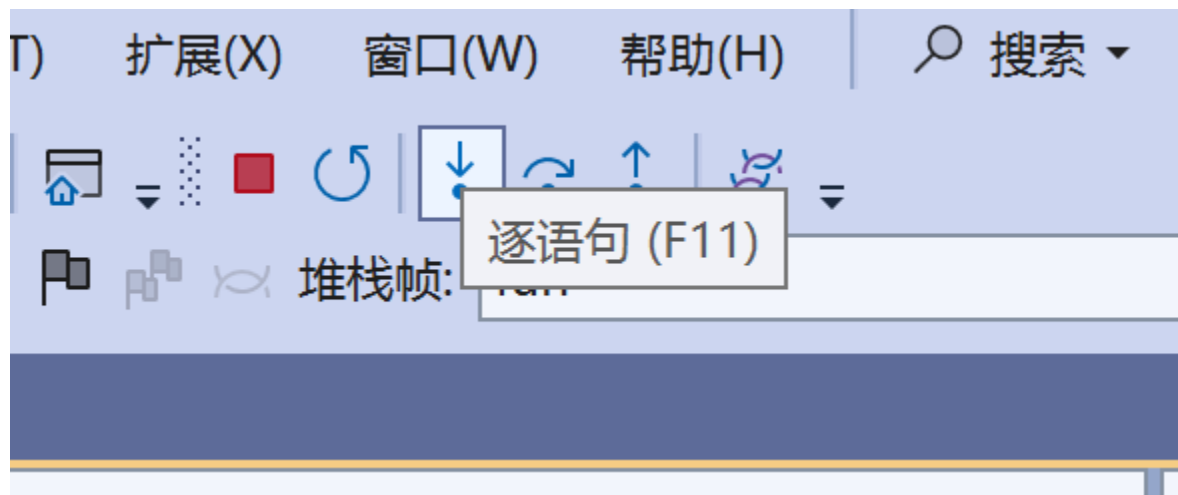
## 1.5 在碰到自定义函数的调用语句时，如何一步步完成自定义函数的执行

- 如图，程序运行到自定义函数句时，点击逐过程则一步完成自定义函数的执行



## 1.6 在碰到自定义函数的调用语句中时，如何转到被调用函数中单步执行

- 如图，程序运行到自定义函数句时，点击逐语句则转到被调用函数中单步执行

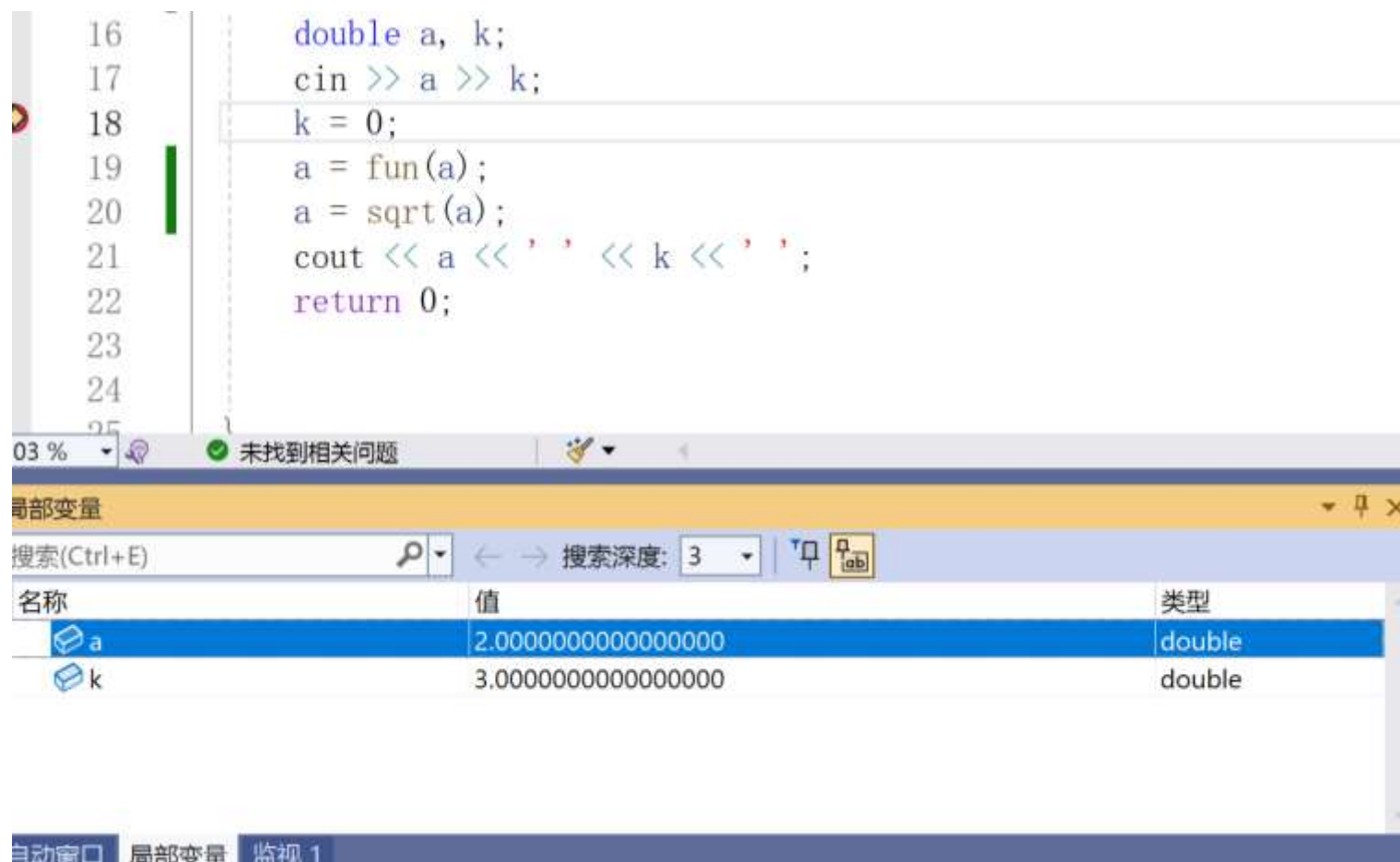




## 2. 用VS2022调试工具查看各种生存期/作用域变量的方法

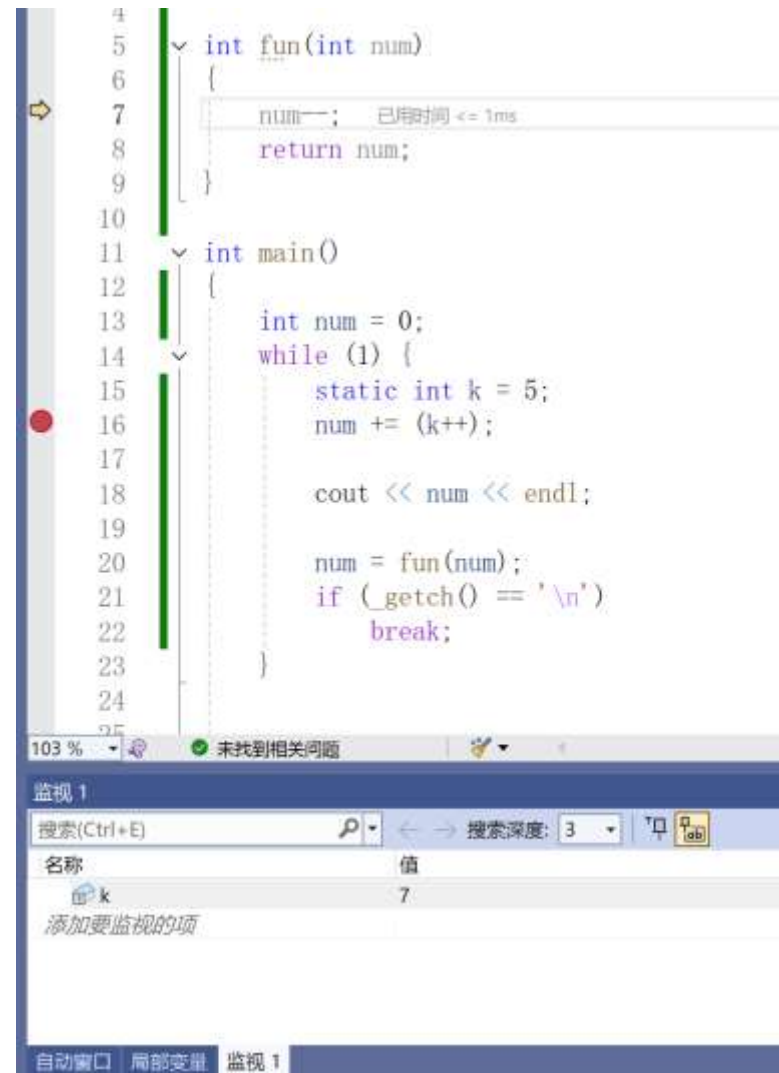
## 2.1 查看形参/自动变量的变化情况

- 形参会在生存周期内自动出现在局部变量监视表格中进行变化



## 2.2 查看静态局部变量的变化情况（该静态局部变量所在的函数体内/外）

- 在函数体内：自动进入“局部变量”、“自动窗口”监视栏
- 在函数体外：消失在“局部变量”、“自动窗口”监视栏，提前将其加入监视项则可查看其变化情况。



## 2.3 查看静态全局变量的变化情况（两个源程序文件，有静态全局变量同名）

- 各文件的静态全局变量互不影响

```
T2.cpp
1 #include <iostream>
2 using namespace std;
3
4 void fun1();
5 void fun2();
6
7 static int k = 10;
8
9 void fun1() {
10     k += 100;
11     cout << "[T1] = " << k << endl;
12 }
13
14
15
16 int main() {
17     fun1(); // 修改这里的
18     fun2(); // 修改T2的
19     return 0;
20 }
```

```
T2.cpp*
1 #include <iostream>
2 using namespace std;
3
4 static int k = 20;
5
6 void fun2() {
7     k += 200;
8     cout << "[T2] = " << k << endl;
9 }
```

名称	值
k	110

名称	值
k	220

## 2.4 查看外部全局变量的变化情况（两个源程序文件，一个定义，另一个有extern说明）

- extern 声明不分配内存，全局变量共享同一内存地址

The image displays two side-by-side windows from a C++ IDE, illustrating the use of `extern` to link variables across source files.

**Left Window (Source File 1):**

```
1 #include <iostream>
2 using namespace std;
3
4 void printk();
5
6
7
8 int k = 100;
9
10
11 void updatek() {
12     k += 50;
13     cout << "k = " << k << endl;
14 }
15
16 int main() {
17     updatek();
18     printk();
19     return 0;
20 }
```

**Right Window (Source File 2):**

```
1 #include <iostream>
2 using namespace std;
3
4 extern int k;
5
6 void printk() {
7     cout << "[T3] k = " << k << endl;
8 }
```

**Memory Monitor (Bottom):**

The memory monitor at the bottom shows the state of memory. The left monitor shows the initial state where `k` is 100. The right monitor shows the state after the `updatek()` function is called, where `k` has been updated to 150.

名称	值	类型
"no"	"no"	char[3]
k	150	int
&k	0x00c6c000 (test_report.exe!int k) (150)	int *

### 3. 用VS2022的调试工具查看各种不同类型变量的方法

## 3.1 char/int/float

- 将光标移到变量上或查看下方监视窗口均可查看各种不同类型变量的现有值

The screenshot shows a C++ program in an IDE. The code defines three variables: `int num = 0;`, `char ch = 'B';`, and `float f = 1.0;`. The program includes a `while (1) { ... }` loop. The cursor is positioned on the variable `ch` in the line `ch -= 1;`. Below the code editor, the '局部变量' (Local Variables) window is open, displaying the current values and types of the variables.

名称	值	类型
ch	66 'B'	char
f	1.00000000	float
num	0	int

At the bottom of the IDE, the '监视' (Watch) window is also visible, showing the same variable information.

## 3.2 指向简单变量的指针变量（如何查看地址、值？）

- 将光标放在p上（或查看监视栏）可以查看p指向的地址（即num的地址）
- 选中“\*p”，将光标放在“\*p”上（或查看监视栏）可以查看指针变量指向的值

```
18 int* p = &num;
19 *p++;
20
21
22
23 while (1) {
24
25     num += (k++);
26
27     cout << num << endl;
28
29     num = fun1(num);
30     num = fun2(num);
31     if (getch() == '\n')
```

03 %

未找到相关问题

局部变量

搜索(Ctrl+E)



← → 搜索深度: 3

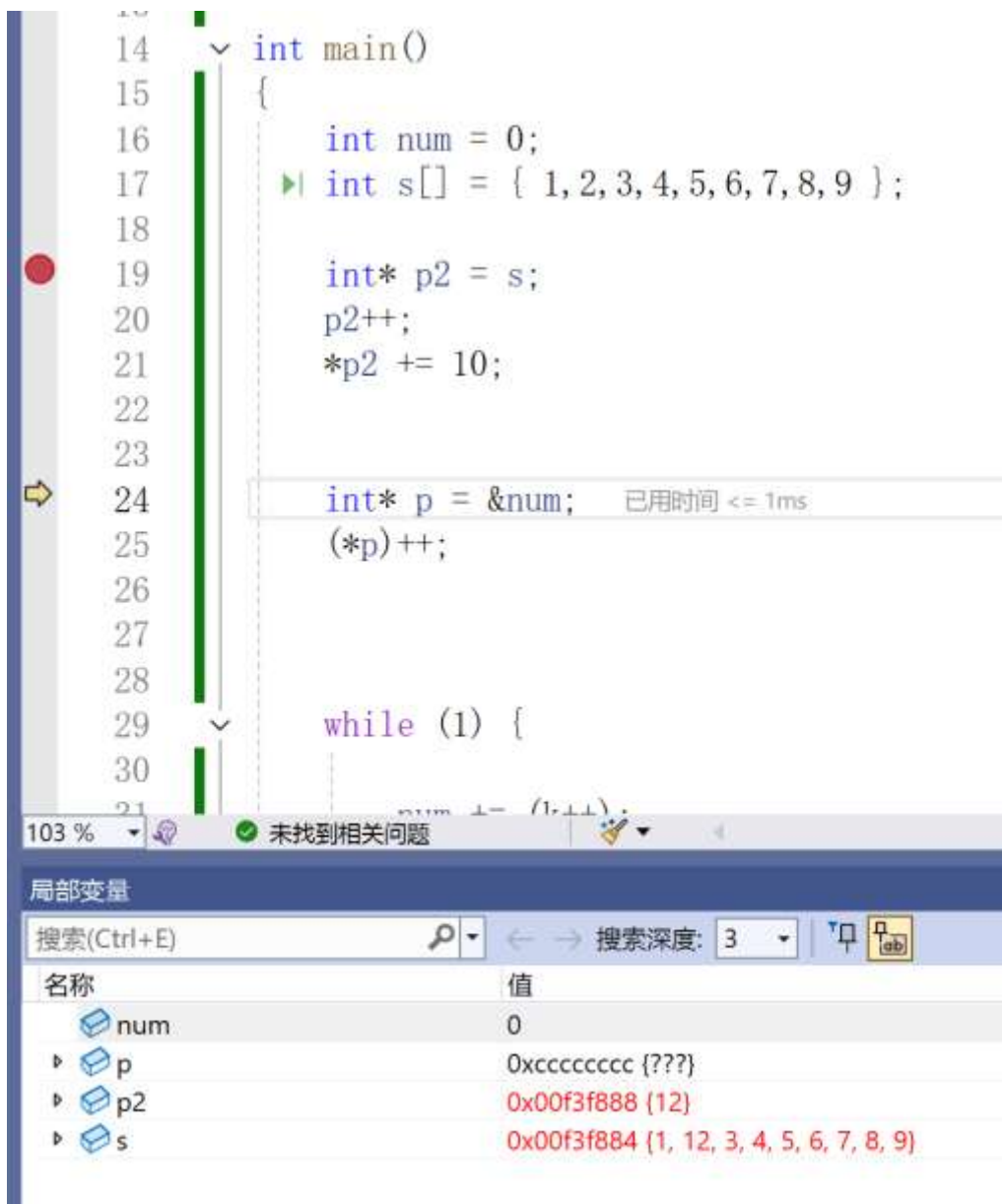


名称	值	类型
num	0	int
p	0x001afc34 {0}	int *



## 3.3&3.4 一维数组&指向一维数组的指针变量（如何查看地址、值？）

- 将光标放在p2上（或查看监视栏）可以查看p2指向的元素地址（此时即s[0]的地址）
- 将光标放在s上（或查看监视栏）可以查看s[]内的各元素值



The screenshot shows a C++ IDE with the following code:

```
14 int main()
15 {
16     int num = 0;
17     int s[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
18
19     int* p2 = s;
20     p2++;
21     *p2 += 10;
22
23
24     int* p = &num; 已用时间 <= 1ms
25     (*p)++;
26
27
28
29     while (1) {
30
31     }
```

The local variable window at the bottom shows the following variables and their values:

名称	值
num	0
p	0xc0000000 {???
p2	0x00f3f888 (12)
s	0x00f3f884 (1, 12, 3, 4, 5, 6, 7, 8, 9)

## 3.5 二维数组（包括数组名仅带一个下标的情况）

- 将光标放在s上（或查看监视栏）可以查看s的值和地址，点进小三角可以查看和监视每行/每个元素

The screenshot shows a C++ IDE with a source code window and a variable watch window.

**Source Code:**

```
12 static int k = 5;  
13  
14 int main()  
15 {  
16     int num = 0;  
17     int s[2][10] = { {0}, 1, 2, 3, 4, 5, 6, 7, 8, 9 };  
18  
19     int* p2 = s[0];  
20     p2++;  
21     *p2 += 10;  
22  
23  
24     int* p = s[1];  
25     (*p)++;  
26  
27 }
```

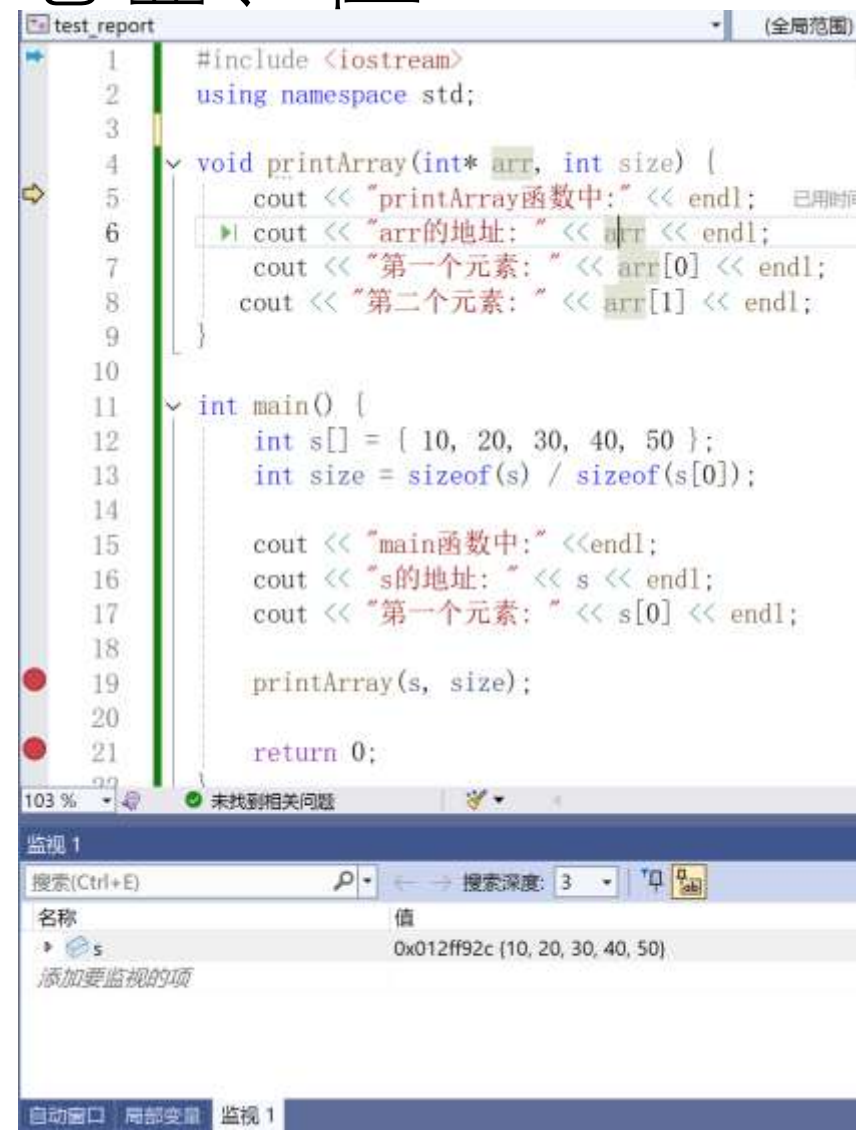
**Variable Watch Window:**

名称	值	类型
num	0	int
p	0xc0000000 {???	int *
p2	0xc0000000 {???	int *
s	0x001ef708 {0x001ef708 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, 0x001ef730 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}}	int [2][10]

The watch window shows the memory address 0x001ef708 for the first row of the array s, which contains the values {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}. The second row, starting at 0x001ef730, contains the values {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}.

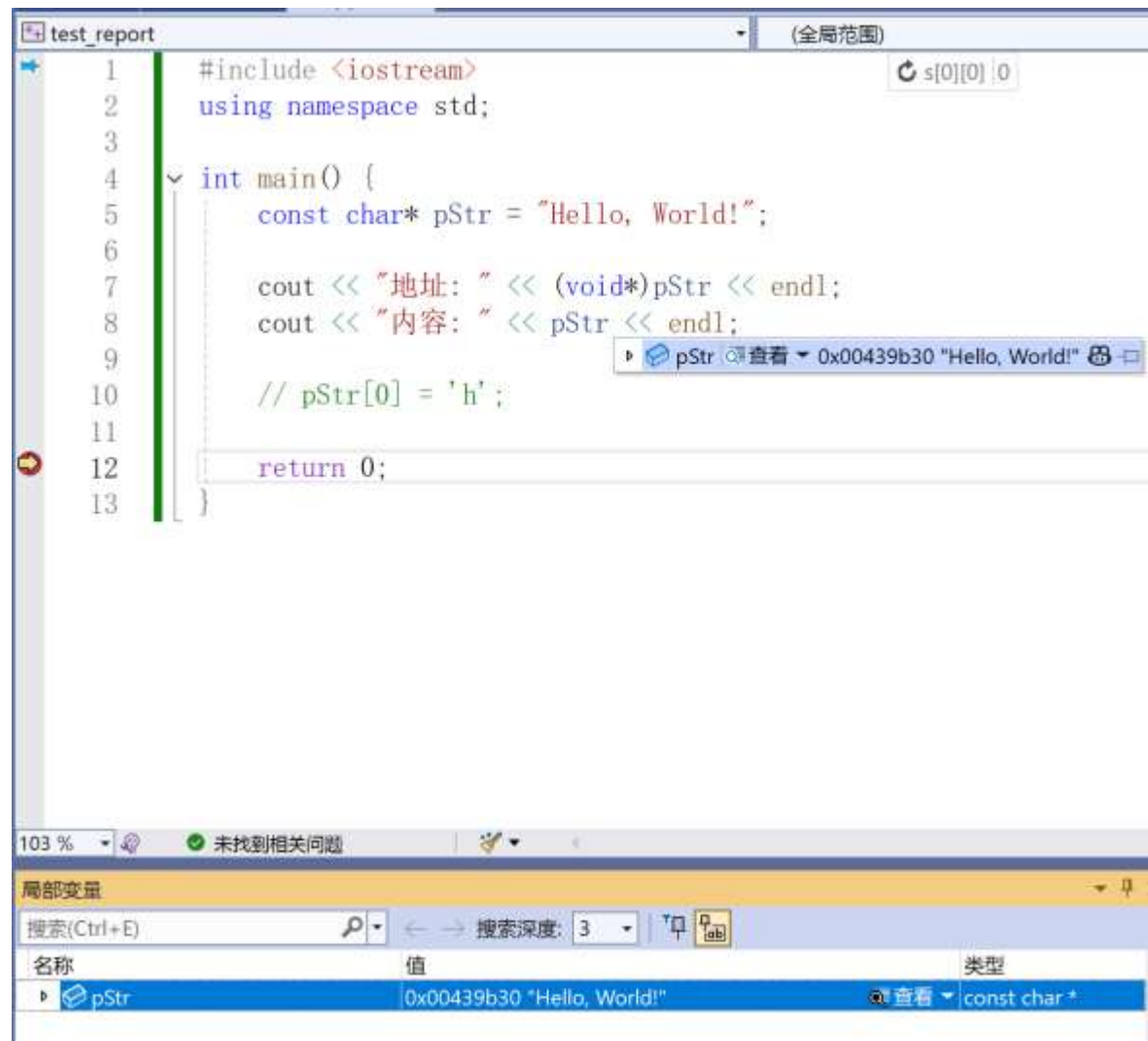
### 3.6 实参是一维数组名，形参是指针的情况，如何在函数中查看实参数组的地址、值？

- 在进入使用形参的函数前将实参数组加入监视。



## 3.7 指向字符串常量的指针变量（能否看到无名字符串常量的地址？）

- 将光标放在pStr上（或查看监视栏）可以查看pStr指向的内容和地址
- 可以直接添加cout << &"xxx";输出无名字符串常量的地址；可以添加指针直接指向这个无名字符串



```
test_report (全局范围)
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      const char* pStr = "Hello, World!";
6
7      cout << "地址: " << (void*)pStr << endl;
8      cout << "内容: " << pStr << endl;
9
10     // pStr[0] = 'h';
11
12     return 0;
13 }
```

103 % 未找到相关问题

局部变量

名称	值	类型
pStr	0x00439b30 "Hello, World!"	const char *

## 3.8 引用（引用与指针是否有区别？ 有什么区别？）

- 引用 ref 的地址与变量 x 相同（`&ref == &x`。
- 指针 ptr 需显式解引用（`*ptr`），且调试器显示其存储的地址。
- 引用无独立内存占用，而指针占用 4/8 字节存储地址。

The screenshot shows a C++ IDE with a file named `test_report`. The code defines a `main` function that declares an integer `x`, a pointer `ptr` pointing to `x`, and a reference `ref` to `x`. It then prints the addresses of `x`, `ptr`, and `ref`, and modifies the values of `ptr` and `ref`. The debug console shows the output of the program, confirming that the addresses of `x`, `ptr`, and `ref` are all the same (010FF958).

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int x = 10;
6     int* ptr = &x; // 指针
7     int& ref = x;  // 引用
8
9     cout << "x 的地址: " << &x << endl;
10    cout << "ptr 的值 (指向的地址): " << ptr << endl;
11    cout << "ref 的地址: " << &ref << endl;
12
13    // 修改值
14    *ptr = 20;
15    ref = 30;
16
17    return 0;
18 }
```

Debug Console Output:

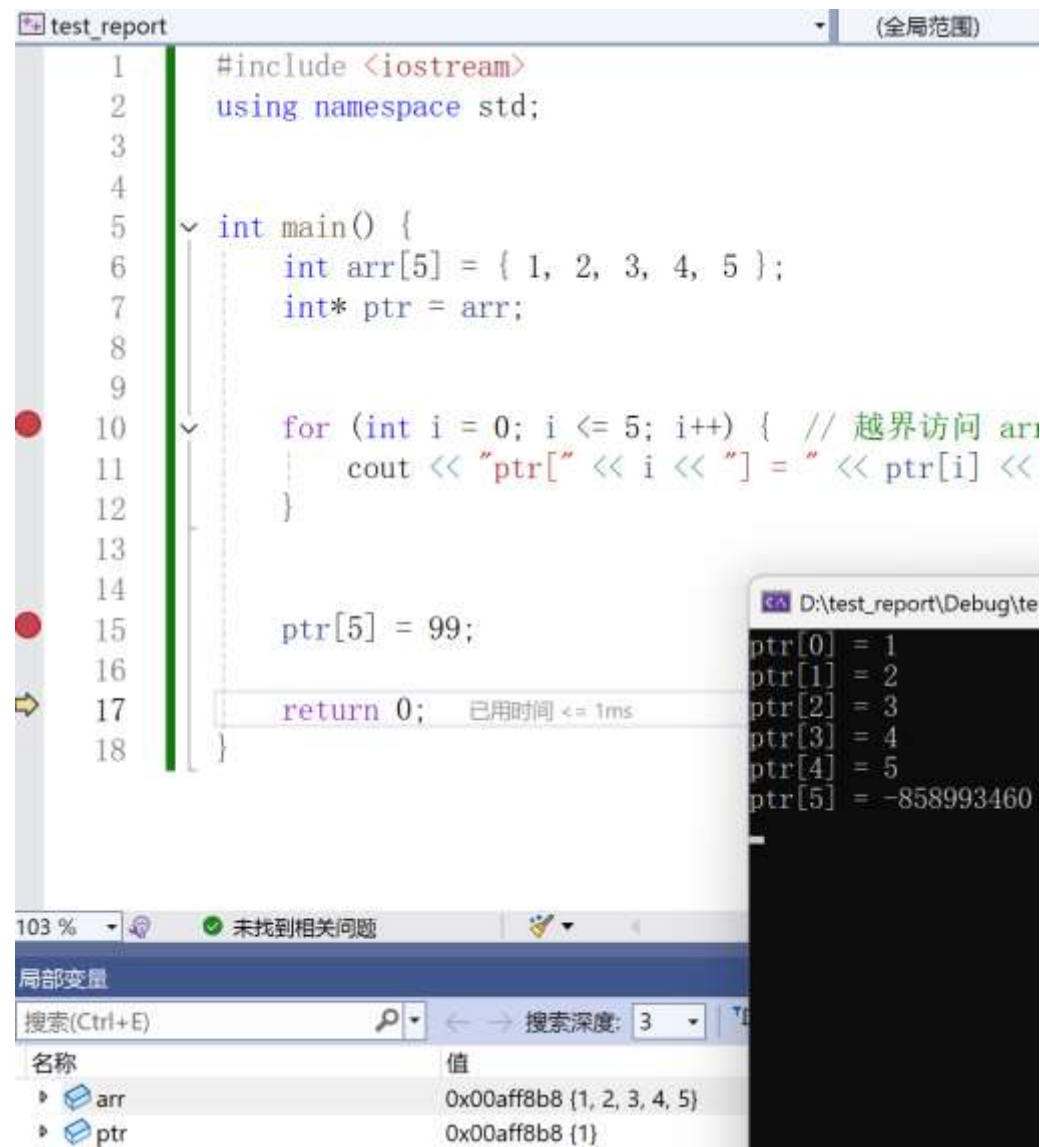
```
x 的地址: 010FF958
ptr 的值 (指向的地址): 010FF958
ref 的地址: 010FF958
```

Local Variables Table:

名称	值
ptr	0x010ff958 (30)
ref	30
x	30

## 3.9 使用指针时出现了越界访问

- 本次调试中，当进行越界访问并输出值时没有报错，且最后成功修改这个值。（其实是未定义行为）



```
1  #include <iostream>
2  using namespace std;
3
4
5  int main() {
6      int arr[5] = { 1, 2, 3, 4, 5 };
7      int* ptr = arr;
8
9
10     for (int i = 0; i <= 5; i++) { // 越界访问 arr
11         cout << "ptr[" << i << "] = " << ptr[i] << endl;
12     }
13
14
15     ptr[5] = 99;
16
17     return 0; 已用时间 <= 1ms
18 }
```

输出窗口 (D:\test\_report\Debug\te):

```
ptr[0] = 1
ptr[1] = 2
ptr[2] = 3
ptr[3] = 4
ptr[4] = 5
ptr[5] = -858993460
```

局部变量:

名称	值
arr	0x00aff8b8 {1, 2, 3, 4, 5}
ptr	0x00aff8b8 {1}