

VS 2022 调试工具使用方法报告

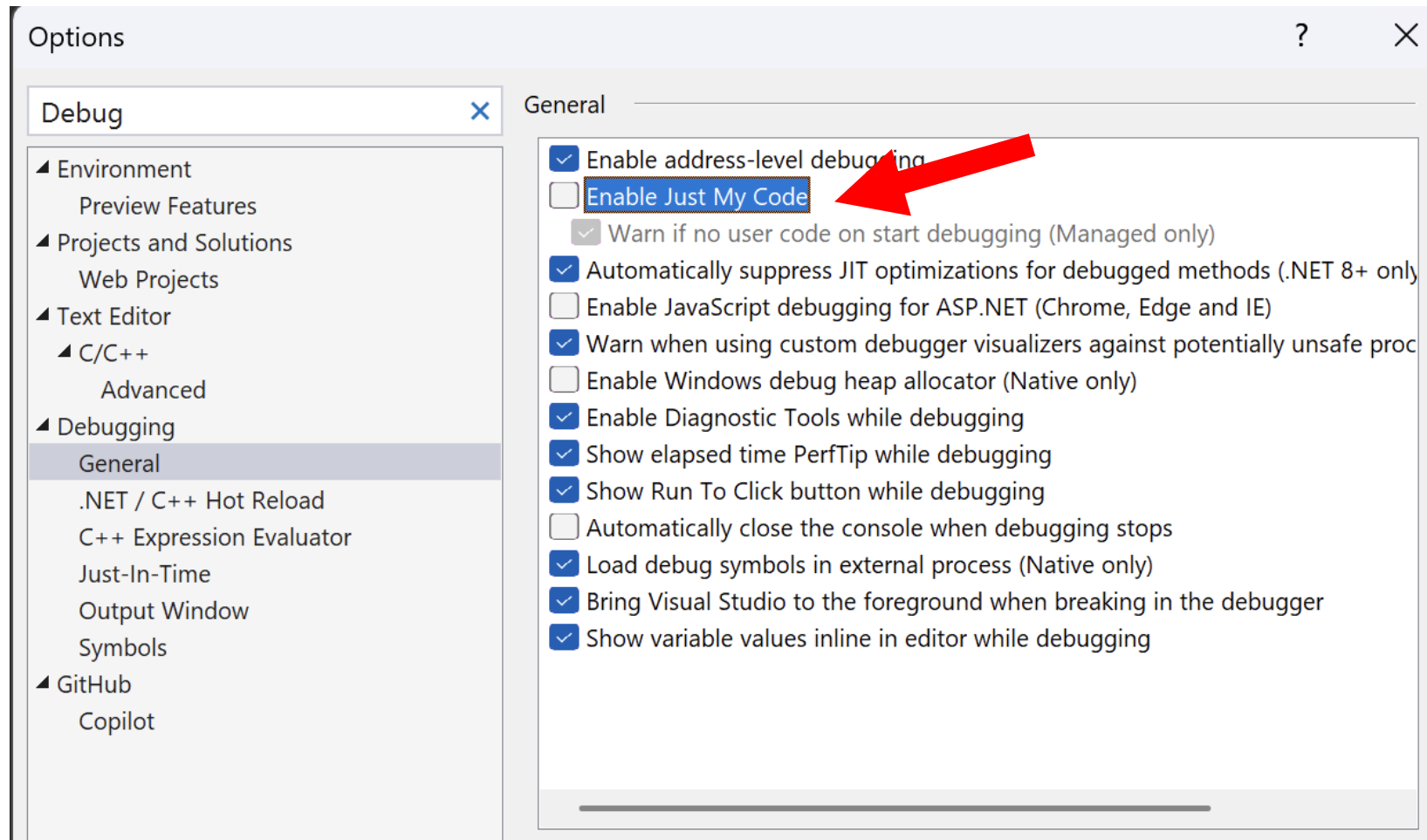
姓名： 汤皓宇

学号： 2454307

班级： 5000244001608

完成日期： 2025年6月4日

在开始之前，首先要取消勾选” Enable Just My Code”（仅我的代码）选项，否则调试无法进入系统函数。



涉及知识点：1.3、1.5

```

1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  extern int g_extern; // 外部全局变量 (文件2定义)
6  static int s_global = 100; // 静态全局变量
7
8  void customFun(int arr[], int& ref) {
9      static int s_local = 0; // 静态局部变量
10     s_local++;
11     cout << s_global << endl;
12     arr[0] = ref + s_local;
13 }
14
15 int main() {
16     // 3.1-3.9 测试变量
17     char c = 'A';
18     int num = 5;
19     float f = 3.14f;
20     int arr1D[3] = { 1,2,3 }; // 一维数组
21     int* ptr = arr1D; // 指针变量
22     int* ptr1 = &num;
23     int arr2D[2][2] = { {1,2},{3,4} };
24     int& ref = num; // 引用
25
26     // 2.1-2.4 作用域测试
27     customFun(arr1D, ref); // 调用自定义函数系统函数
28     cout << sqrt(num) << endl; //
29
30     const char* str = "Hello"; // 3.7
31     // 3.9 指针越界测试
32     ptr[3] = 10; // 故意越界
33
34     return 0;
35 }

```

extern.cpp main.cpp

example (Global Scope)

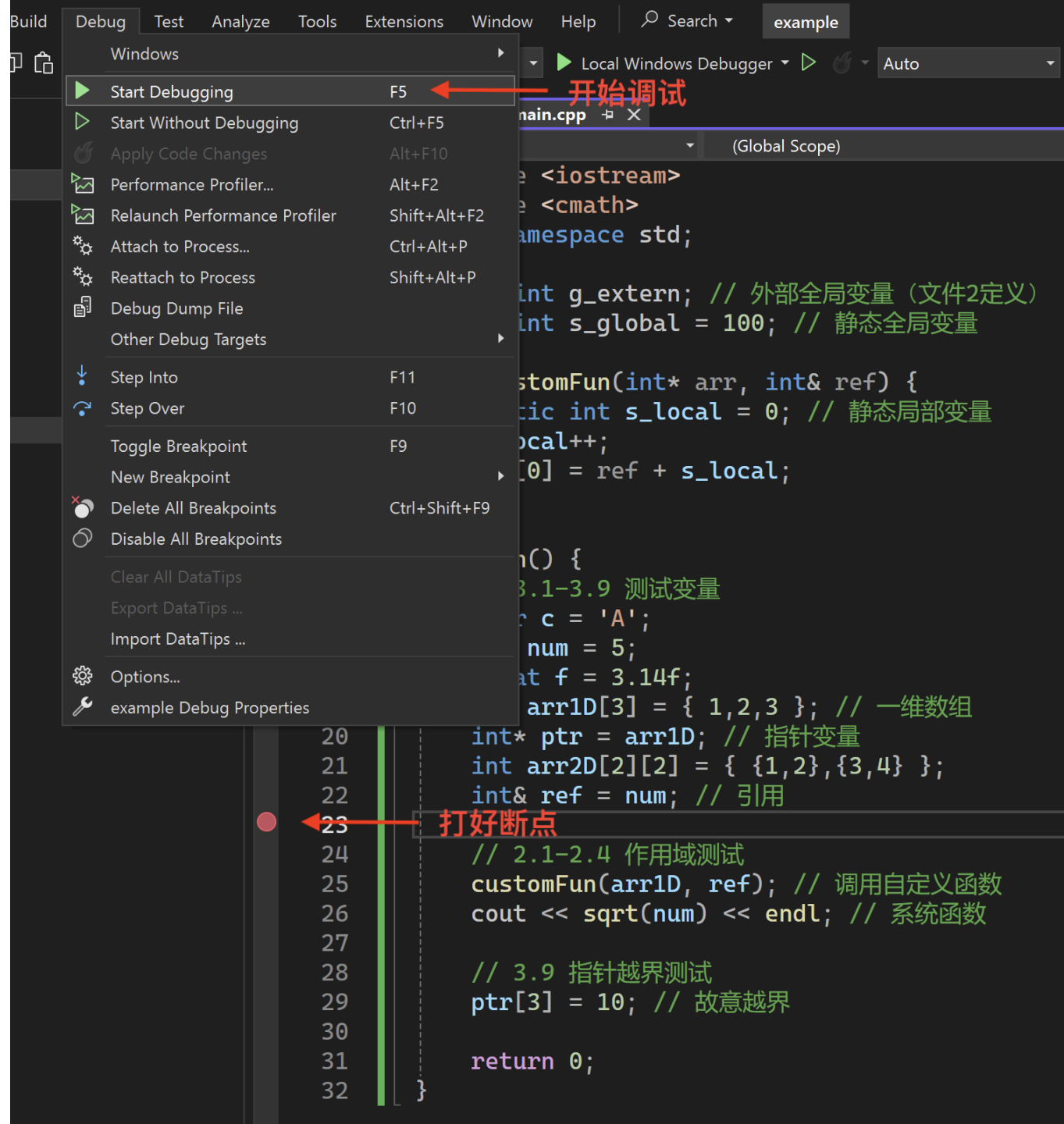
```

1  int g_extern = 50; // 外部全局变量
2  static int s_global = 200; // 同名静态全局变量

```

截图为测试用代码，左侧为测试用主代码，上方有另一个文件内容，用于展示外部全局变量的变化情况。

第一步：点击数字列左侧打好断点
第二部：按下F5开始调试



涉及知识点：1.1

接下来:

如果想退出调试, 则按Shift+F5

如果想跳转到下一个断点, 则按F5

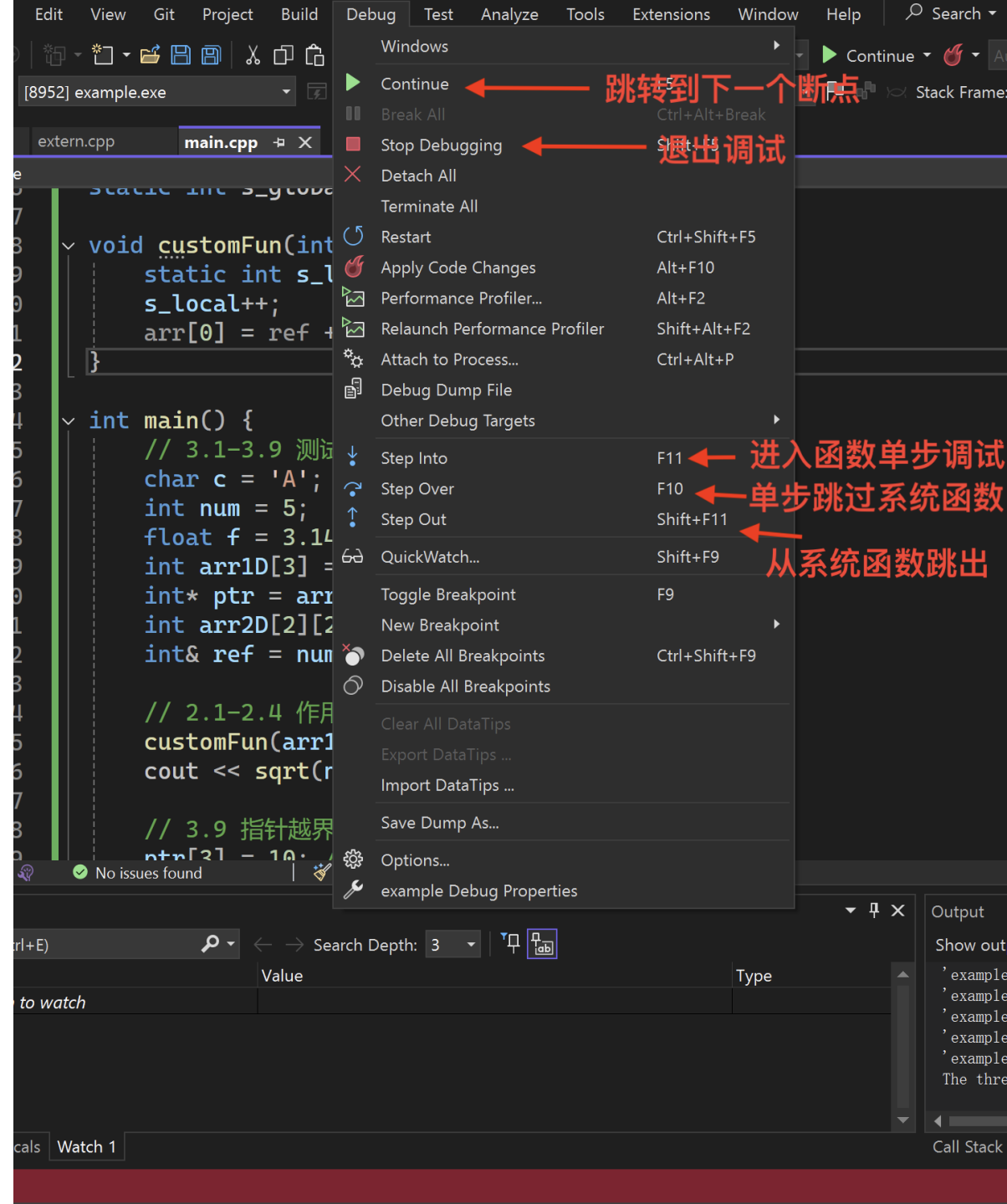
遇到系统函数, 如cout, sqrt时:

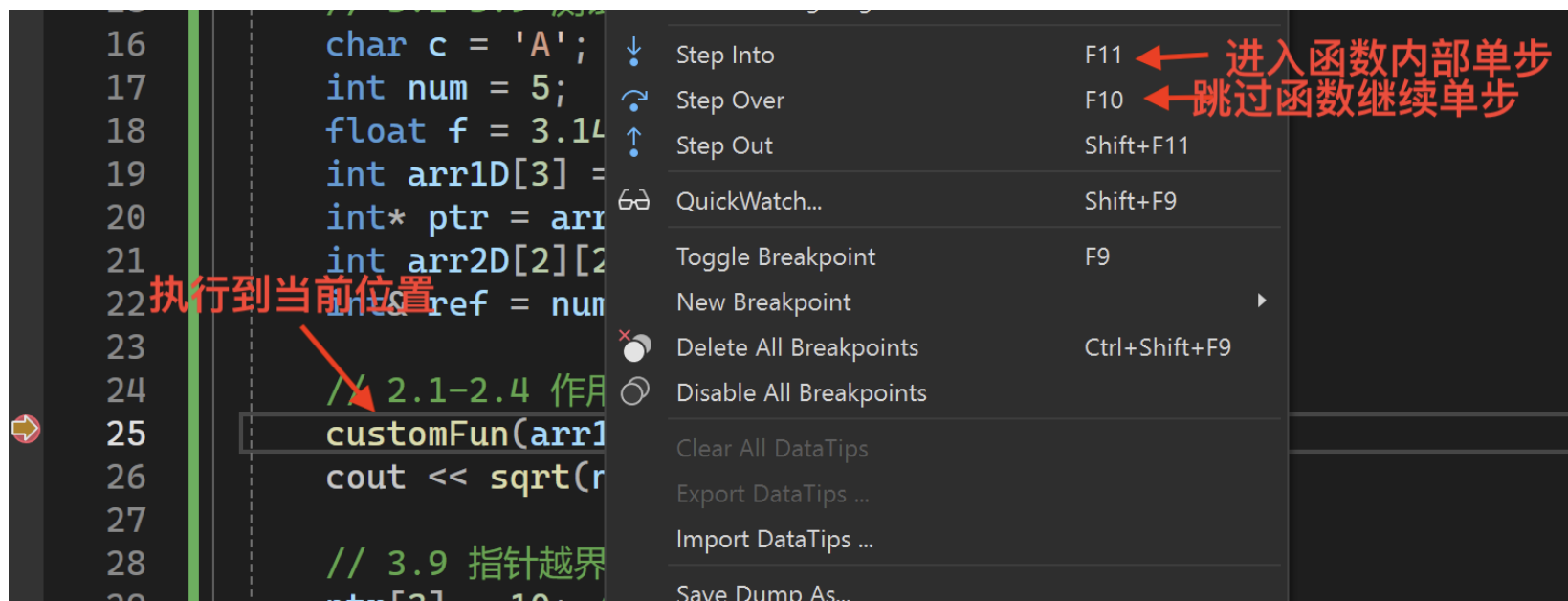
想进入函数单步调试, 则按F11

如果想单步跳过系统函数, 则按F10

如果想从系统函数跳出, 则按Shift+F11

涉及知识点: 1.1、1.2、1.3、1.4





遇到自定义函数时候：
若想跳过函数执行，按F10
若想进入函数单步执行，按F11

涉及知识点：1.5、1.6

```
7
8 void customFun(int* arr, int& ref) {
9     static int s_local = 0; // 静态局部变量
10    s_local++;
11    cout << s_global << endl;
12    arr[0] = ref + s_local;
13 }
14
15 int main() {
16     // 3.1-3.9 测试变量
17     char c = 'A';
18     int num = 5;
19     float f = 3.14f;
20     int arr1D[3] = { 1,2,3 }; // 一维数组
```

100 % No issues found

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
num	5	int
s_local	identifier "s_local" is undefined	
s_global	100	int
g_extern	50	int
arr1D	0x02b3f664 {1, 2, 3}	int[3]
arr2D[0]	0x02b3f640 {1, 2}	int[2]

Add item to watch

自动变量
静态局部变量
静态全局变量
外部全局变量
一维数组
二维数组单下标

未进入函数时

进入函数前：能够在Value栏中观察到自动变量num；

静态全局变量s_global （参考P2中代码，此处=100为main.cpp，并非extern.cpp）

能找到外部全局变量g_extern

能观察到一维数组和单下标二维数组的地址和内容

进入函数后：num、arr1D、arr2D[0]超出作用域，无法查看变化，点击刷新后会显示错误

静态局部变量s_local进入作用域，此时可以查看

涉及内容：2.1、2.2、2.3、2.4、3.1、3.3、3.5

```
8 void customFun(int* arr, int& ref) {
9     static int s_local = 0; // 静态局部变量
10    s_local++;
11    cout << s_global << endl;
12    arr[0] = ref + s_local;
13 }
14
15 int main() {
16     // 3.1-3.9 测试变量
17     char c = 'A';
18     int num = 5;
19     float f = 3.14f;
20     int arr1D[3] = { 1,2,3 }; // 一维数组
```

100 % No issues found

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
num	identifier "num" is undefined	
s_local	1	int
s_global	100	int
g_extern	50	int
arr1D	identifier "arr1D" is undefined	
arr2D[0]	identifier "arr2D" is undefined	

Add item to watch

超出作用范围
与cpp内定义值相同

进入函数后

```
17 char c = 'A';
18 int num = 5;
19 float f = 3.14f;
20 int arr1D[3] = { 1,2,3 }; // 一维数组
21 int* ptr = arr1D; // 指针变量
22 int* ptr1 = &num;
23 int arr2D[2][2] = { {1,2},{3,4} };
24 int& ref = num; // 引用
25
```

100 % No issues found

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
c	65 'A'	char
num	5	int
f	3.14000010	float

显示了char/int/float型简单变量

涉及知识点：3.1


```
26 // 2.1-2.4 作用域测试
27 customFun(arr1D, ref); // 调用自定义函数系统函数
28 cout << sqrt(num) << endl; //
29
30 // 3.9 指针越界测试
31 ptr[3] = 10; // 故意越界
32
```

100 % No issues found

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
ptr	0x008ff9a4 {6}	int *
*ptr	6	int
&ref	0x008ff9c4 {5}	int *
&num	0x008ff9c4 {5}	int *
ptr1	0x008ff9c4 {5}	int *
*ptr1	5	int
ref	5	int &

Add item to watch

指针ptr1指向简单变量num， ptr1显示地址，*ptr1显示内容
指针ptr指向一维数组的开头， ptr显示arr1D[0]的地址， *ptr显示arr1D[0]的地址
ref为引用， ref变量地址和num地址相同， 其本身保存值为num， 而指针储存的是目标变量的地址

涉及知识点： 3.2、3.4、3.8

```
10     s_local++;
11     cout << s_global << endl;
12     arr[0] = ref + s_local;
13 }
14
15 int main() {
16     // 3.1-3.9 测试变量
17     char c = 'A';
18     int num = 5;
19     float f = 3.14f;
20     int arr1D[3] = { 1, 2, 3 }; // 一维数组
}
```

100 % No issues found

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
arr,3	0x02affc64 {6, 2, 3}	int[3]
Add item to watch		

arr, 3表示查看从arr指向内存开始连续3块内存的内容，可以通过这种方式在函数中查看实参数数组的地址、值

涉及知识点：3.6

The screenshot shows a Visual Studio IDE with a C++ program. The code is as follows:

```
29  
30 const char* str = "Hello"; // 3.7  
31 // 3.9 指针越界测试  
32 ptr[3] = 10; // 故意越界  
33  
34 return 0;  
35 }
```

An "Exception Thrown" dialog box is open, displaying the message: "Run-Time Check Failure #2 - Stack around the variable 'arr1D' was corrupted." Below the message, there are links for "Ask Copilot", "Show Call Stack", and "Copy Details".

Below the code editor, the "Watch" window is open, showing a table of variables:

Name	Value	Type
str	0x001c7bd0 "Hello"	const char *
*str	72 'H'	const char
ptr	0x0336f844 {6}	int *
ptr[3]	10	int

Red arrows point to the "str" row and the "ptr[3]" row, with red text annotations: "能显示无名字字符串" (can display unnamed string) and "越界变红表示出错" (overflow turns red to indicate error).

The "Output" window on the right shows a list of file paths.

添加对str的监视，能查看到无名字字符串常量的内容和地址
使用指针越界时，监视栏值变红，同时跳出报错信息

包含内容：3.7、3.9