

VS 2022 调试工具使用方法报告

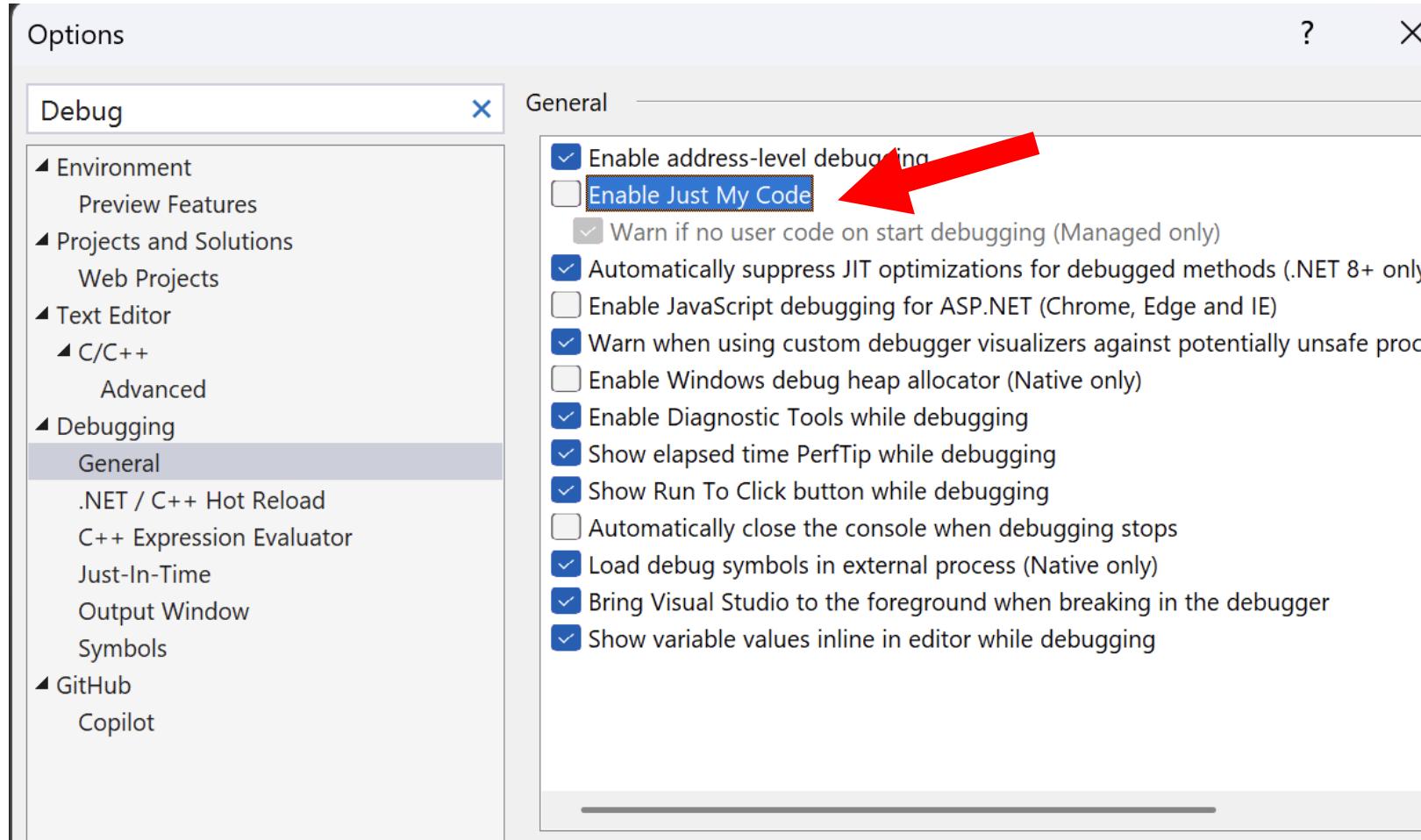
姓名: 汤皓宇

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 X main.cpp

example (Global Sc)

```
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。

Build Debug Test Analyze Tools Extensions Window Help | Search example

Local Windows Debugger Auto

Windows

Start Debugging F5 ← **开始调试**

Start Without Debugging Ctrl+F5

Apply Code Changes Alt+F10

Performance Profiler... Alt+F2

Relaunch Performance Profiler Shift+Alt+F2

Attach to Process... Ctrl+Alt+P

Reattach to Process Shift+Alt+P

Debug Dump File

Other Debug Targets

Step Into F11

Step Over F10

Toggle Breakpoint F9

New Breakpoint

Delete All Breakpoints Ctrl+Shift+F9

Disable All Breakpoints

Clear All DataTips

Export DataTips ...

Import DataTips ...

Options...

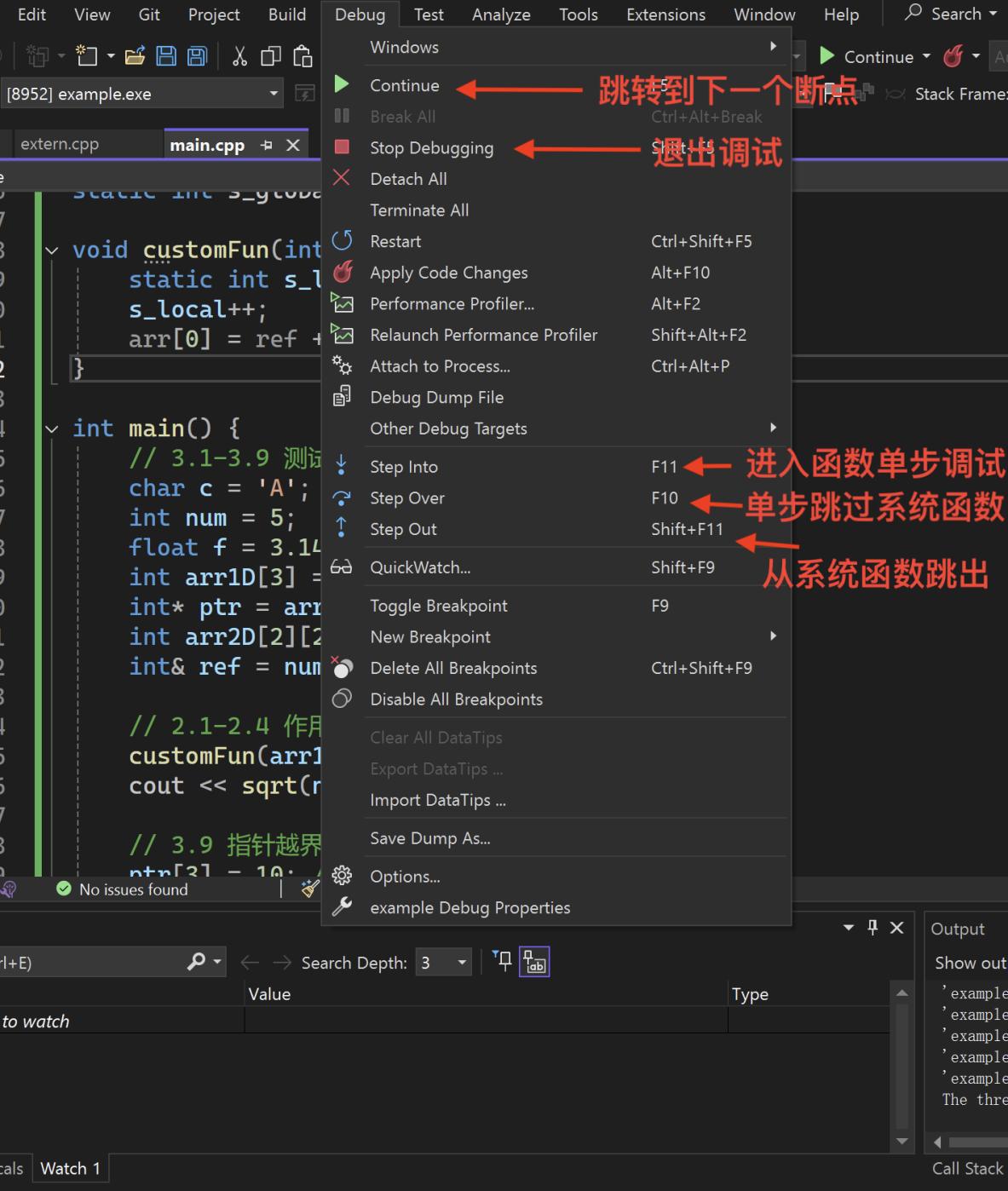
example Debug Properties

```
int g_extern; // 外部全局变量(文件2定义)
int s_global = 100; // 静态全局变量
void customFun(int* arr, int& ref) {
    static int s_local = 0; // 静态局部变量
    s_local++;
    arr[0] = ref + s_local;
}
int main() {
    // 3.1-3.9 测试变量
    char c = 'A';
    int num = 5;
    float pi = 3.14f;
    arr1D[3] = { 1,2,3 }; // 一维数组
    int* ptr = arr1D; // 指针变量
    int arr2D[2][2] = { {1,2},{3,4} };
    int& ref = num; // 引用
    // 2.1-2.4 作用域测试
    customFun(arr1D, ref); // 调用自定义函数
    cout << sqrt(num) << endl; // 系统函数
    // 3.9 指针越界测试
    ptr[3] = 10; // 故意越界
}
return 0;
```

打好断点 ← 23

第一步：点击数字列左侧打好断点
第二步：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

16 char c = 'A'; ↓ Step Into F11 ← 进入函数内部单步
17 int num = 5; ↗ Step Over F10 ← 跳过函数继续单步
18 float f = 3.14;
19 int arr1D[3] =
20 int* ptr = arr;
21 int arr2D[2][2];
22 int& ref = num;
23
24 // 2.1-2.4 作用
25 customFun(arr);
26 cout << sqrt(r);
27
28 // 3.9 指针越界
29 arr[3] = 10;

执行到当前位置

跳过函数继续单步

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

```

Watch 1			
Name	Value	Type	
num	5	int	自动变量
s_local	identifier "s_local" is undefined	int	静态局部
s_global	100	int	静态全局
g_extern	50	int	外部全局
arr1D	0x02b3f664 {1, 2, 3}	int[3]	一维数组
arr2D[0]	0x02b3f640 {1, 2}	int[2]	二维数组单下标
<i>Add item to watch</i>			

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

```

Watch 1			
Name	Value	Type	
num	identifier "num" is undefined	int	超出作用范围
s_local	1	int	与cpp内定义值相同
s_global	100	int	
g_extern	50	int	
arr1D	identifier "arr1D" is undefined	int[3]	
arr2D[0]	identifier "arr2D" is undefined	int[2]	
<i>Add item to watch</i>			

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
```

100 % No issues found | Search Depth: 3

Watch 1

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
```

Watch 1

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 &

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

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

```
10     s_global = s_global + 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
```

The code includes a break point at line 13. Below the editor is a status bar with "No issues found".

Below the editor is a "Watch 1" window with the following content:

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组的地址、值

涉及知识点：3.6

The screenshot shows a debugger interface with the following details:

- Code Editor:** Shows a C/C++ code snippet with a buffer overflow at line 32. The code includes `const char* str = "Hello";` and `ptr[3] = 10;`.
- Exception Thrown:** A modal window displays the error message: "Run-Time Check Failure #2 - Stack around the variable 'arr1D' was corrupted."
- Watch Window:** Titled "Watch 1", it lists variables and their values:

Name	Type
str	const char *
*str	const char
ptr	int *
ptr[3]	int

Annotations highlight:
 - A red arrow points to the value of str: "0x001c7bd0 \"Hello\"", with the text "能显示无名字符串".
 - A red arrow points to the value of ptr[3]: "10", with the text "越界变红表示出错".

添加对str的监视，能看到无名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