

VS 2022 调试工具的使用

学号：2351050

姓名：杨瑞晨

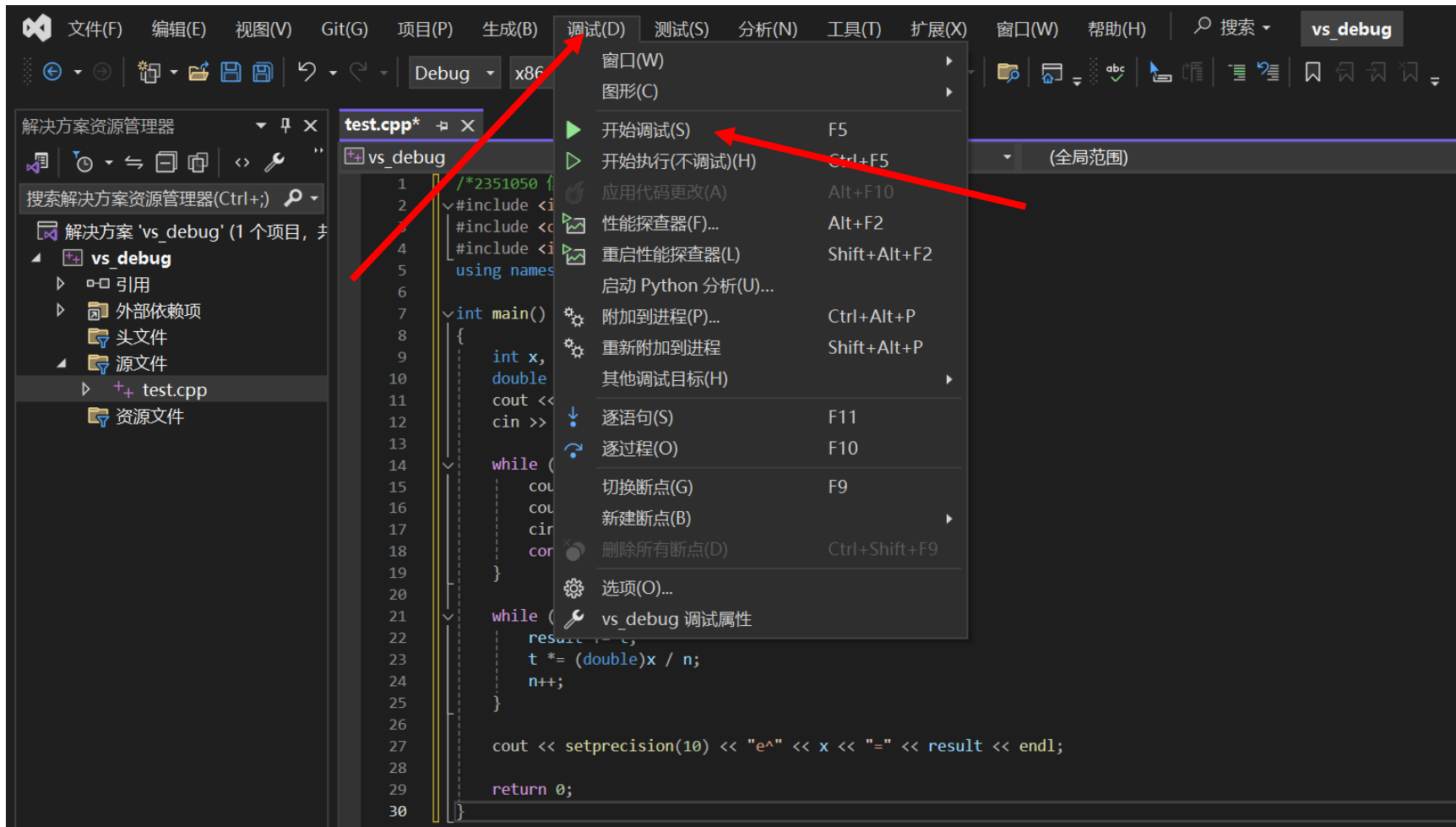
班级：信05/02

完成日期：16/06/24

1. VS2022 下调试工具的基本使用方法

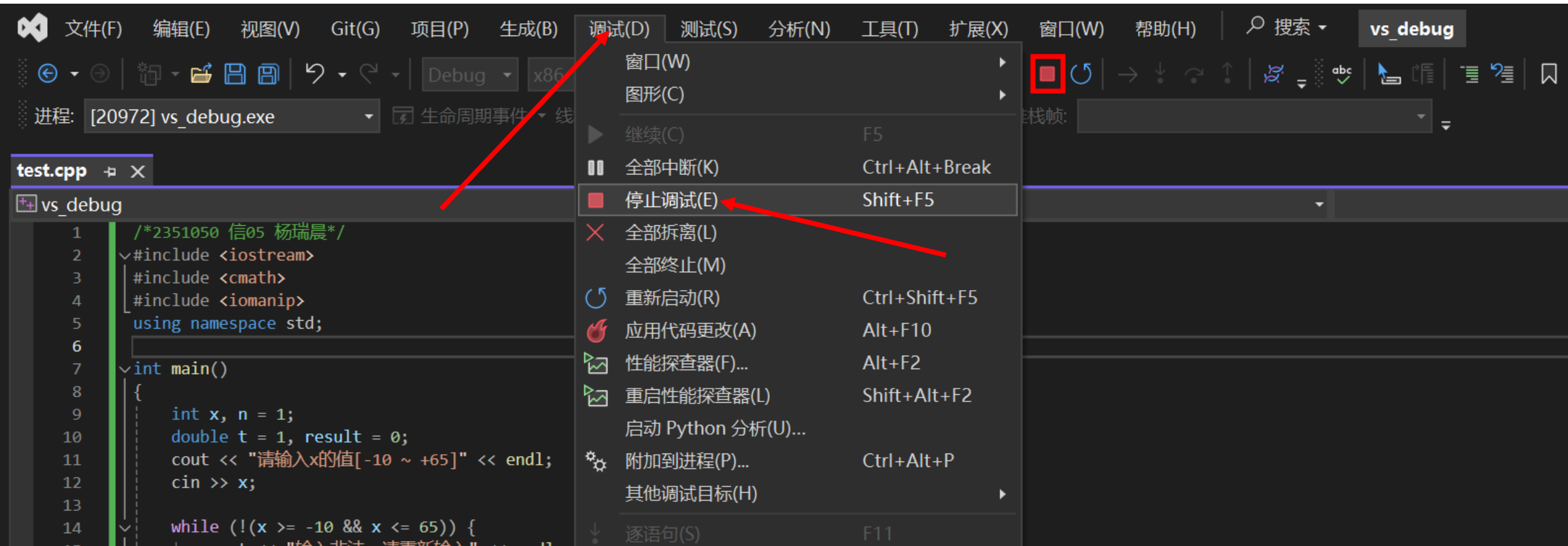
1.1 如何开始调试？ 如何结束调试？

1.1.1 开始调试



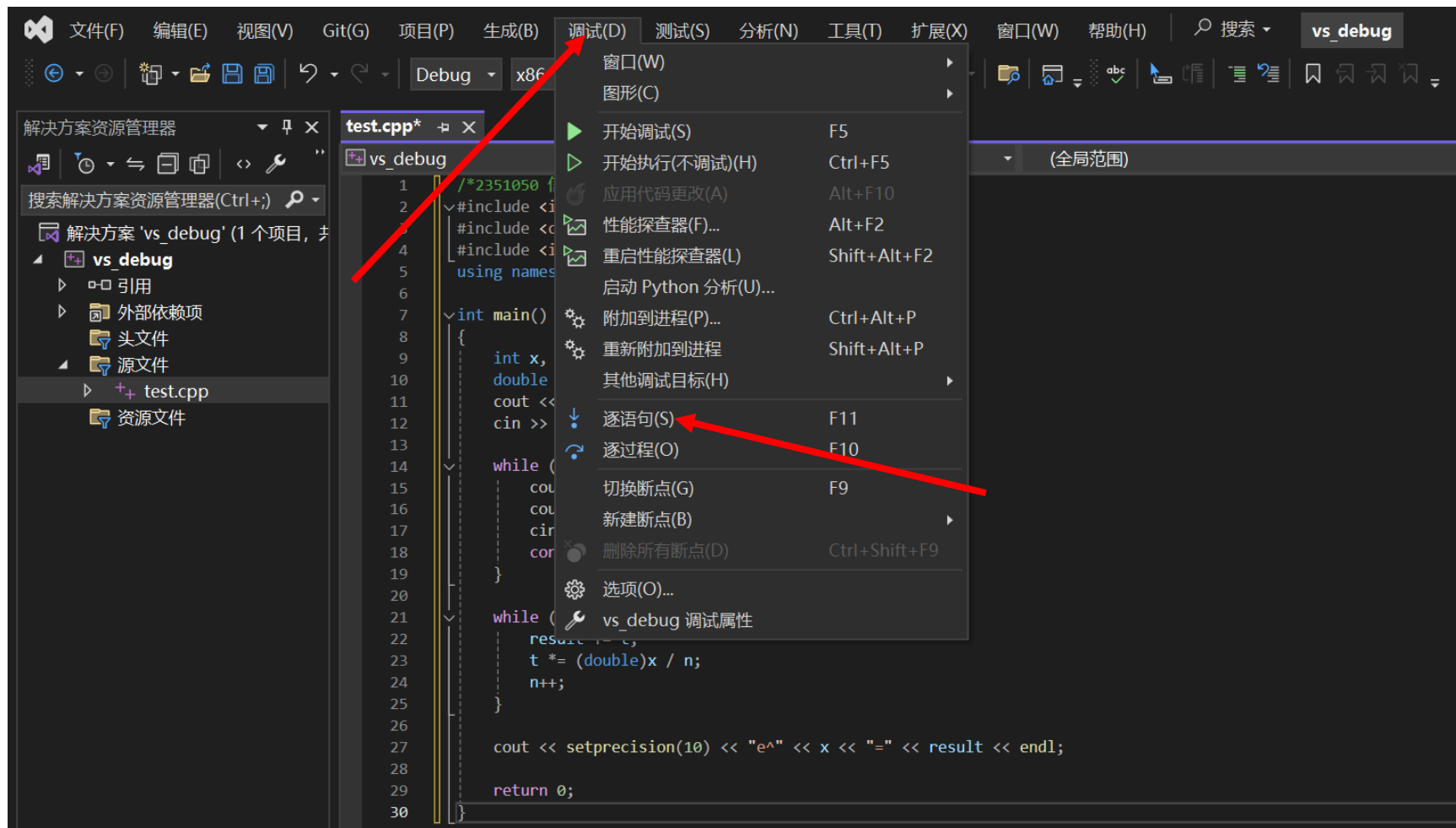
按 F5 键或点击 调试-开始调试

1.1.2 结束调试



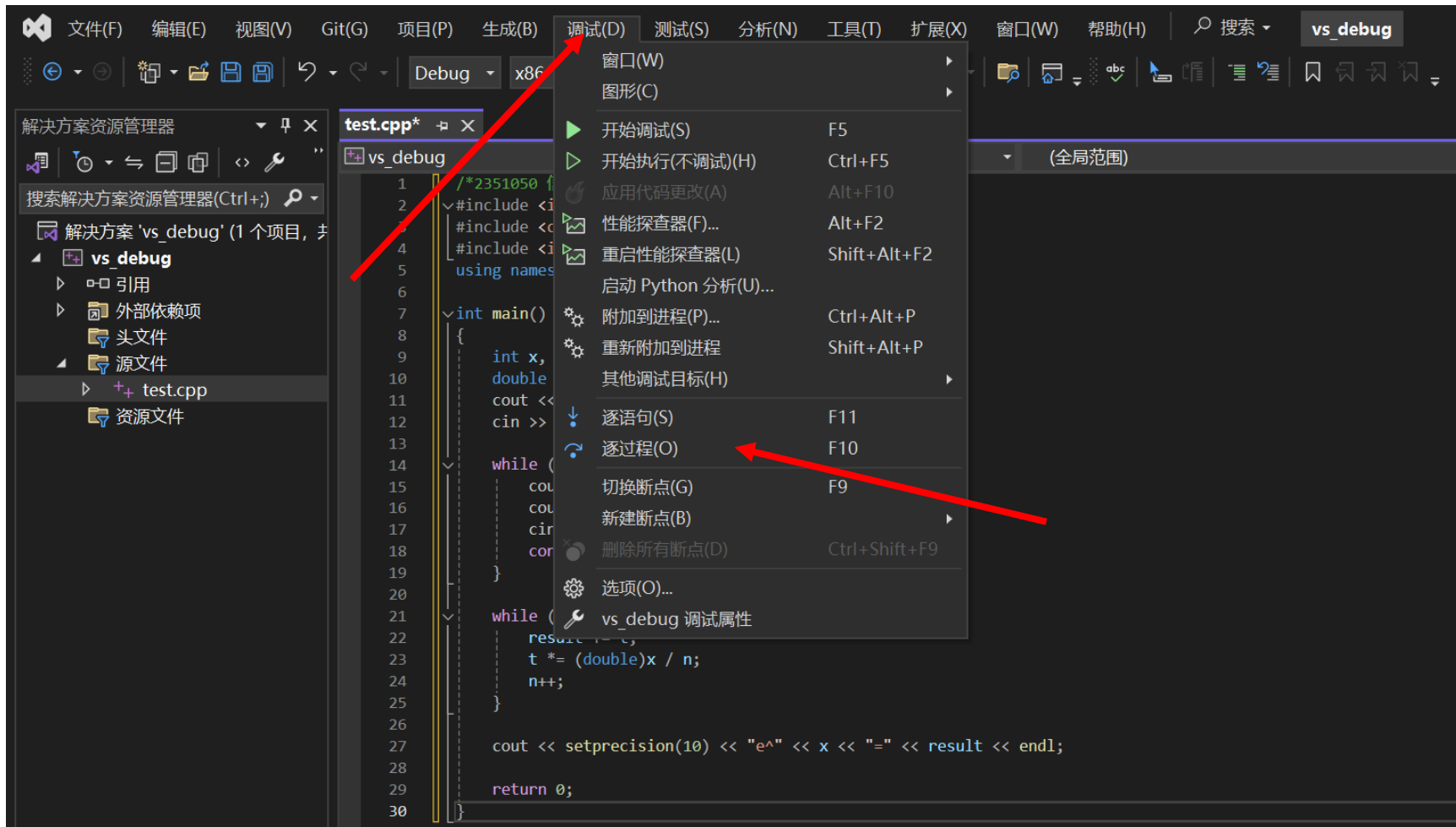
按 Shift+F5 键或点击 调试-停止调试 或直接点击红色方块

1.2 如何在一个函数中每个语句单步执行？



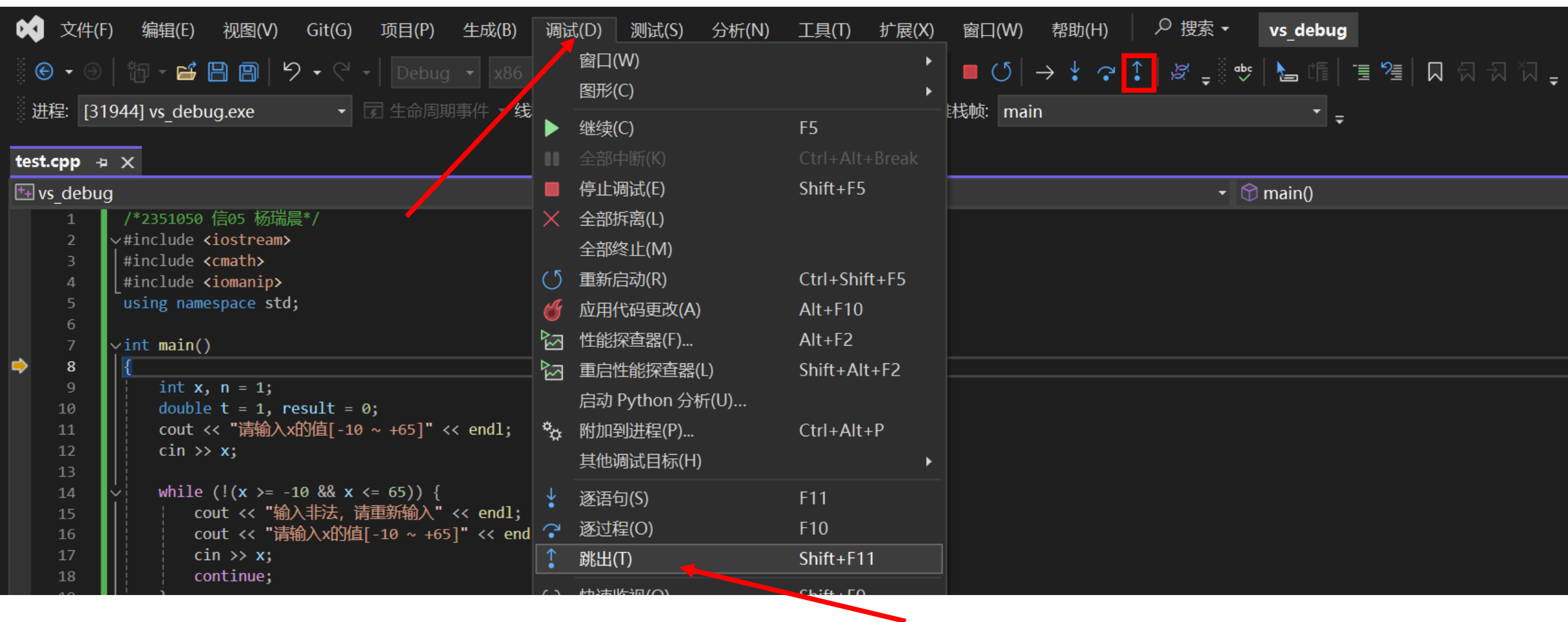
按 F11 键或点击 调试-逐语句

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



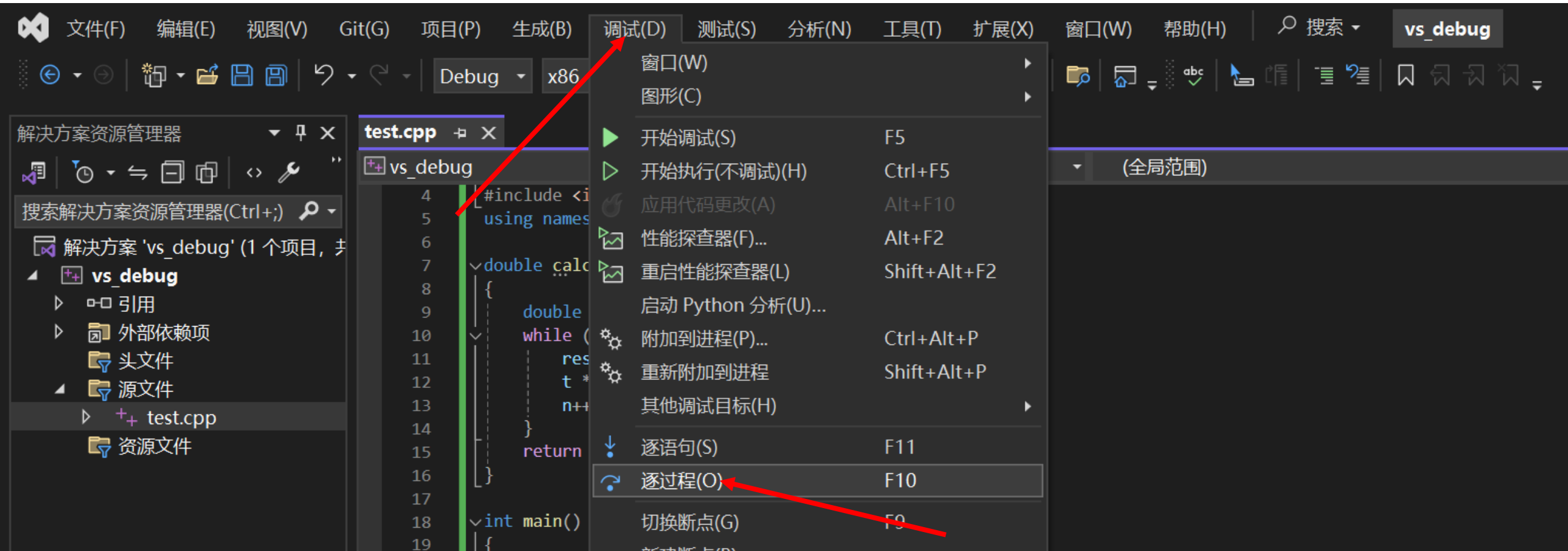
按 F10 键或点击 调试-逐过程

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



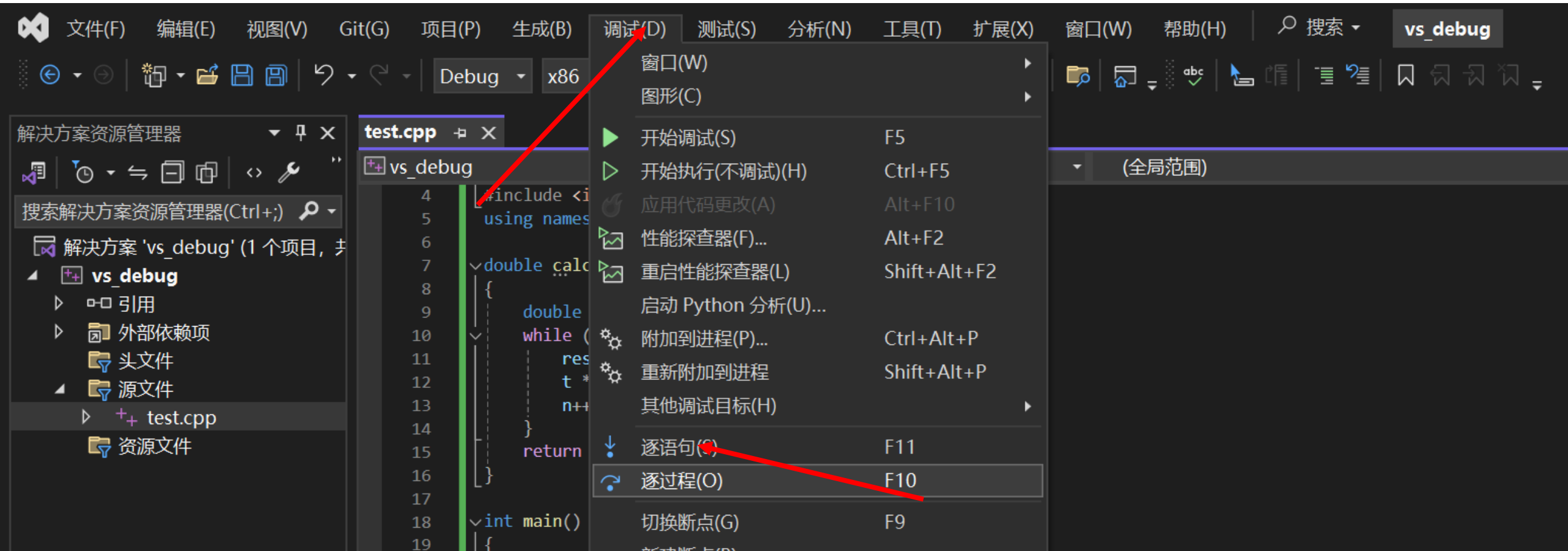
按 Shift+F11 键或点击 调试-跳出 或直接点击红色方块内的图标

1.5 在碰到自定义函数的调用语句（例如在main 中调用自定义的fun 函数）时，如何一步完成自定义函数的执行而不要进入到这些自定义函数的内部单步执行？



按 F10 键或点击 调试-逐过程

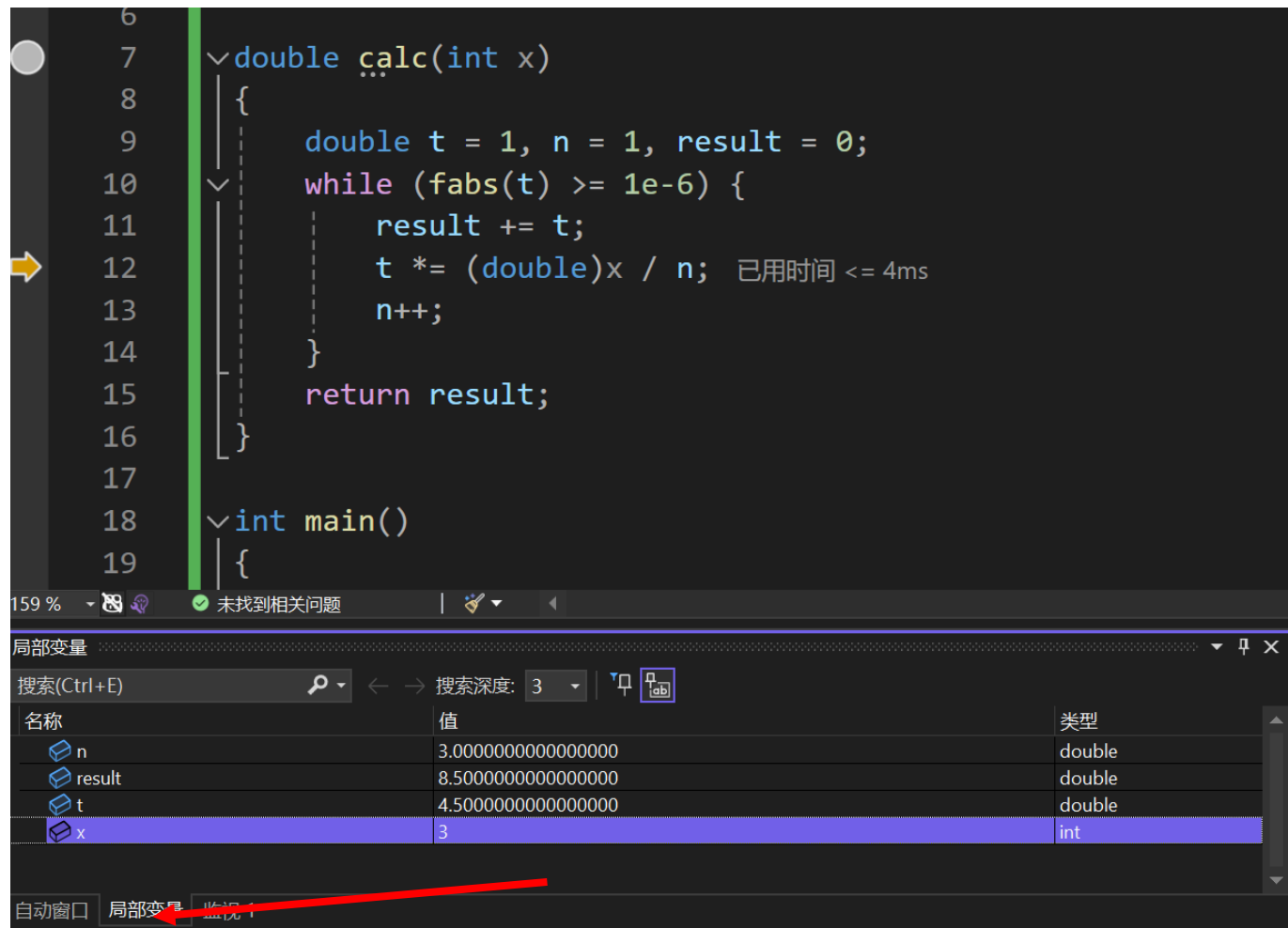
1.6 在碰到自定义函数的调用语句（例如在main 中调用自定义的fun 函数）时，如何转到被调用函数中单步执行？



按 F11 键或点击 调试-逐语句

2. 使用 VS2022 的调试工具 查看各种生存期/作用域变量

2.1 查看形参/自动变量的变化情况



```
6
7  double calc(int x)
8  {
9      double t = 1, n = 1, result = 0;
10     while (fabs(t) >= 1e-6) {
11         result += t;
12         t *= (double)x / n; 已用时间 <= 4ms
13         n++;
14     }
15     return result;
16 }
17
18 int main()
19 {
```

159 % 未找到相关问题

局部变量

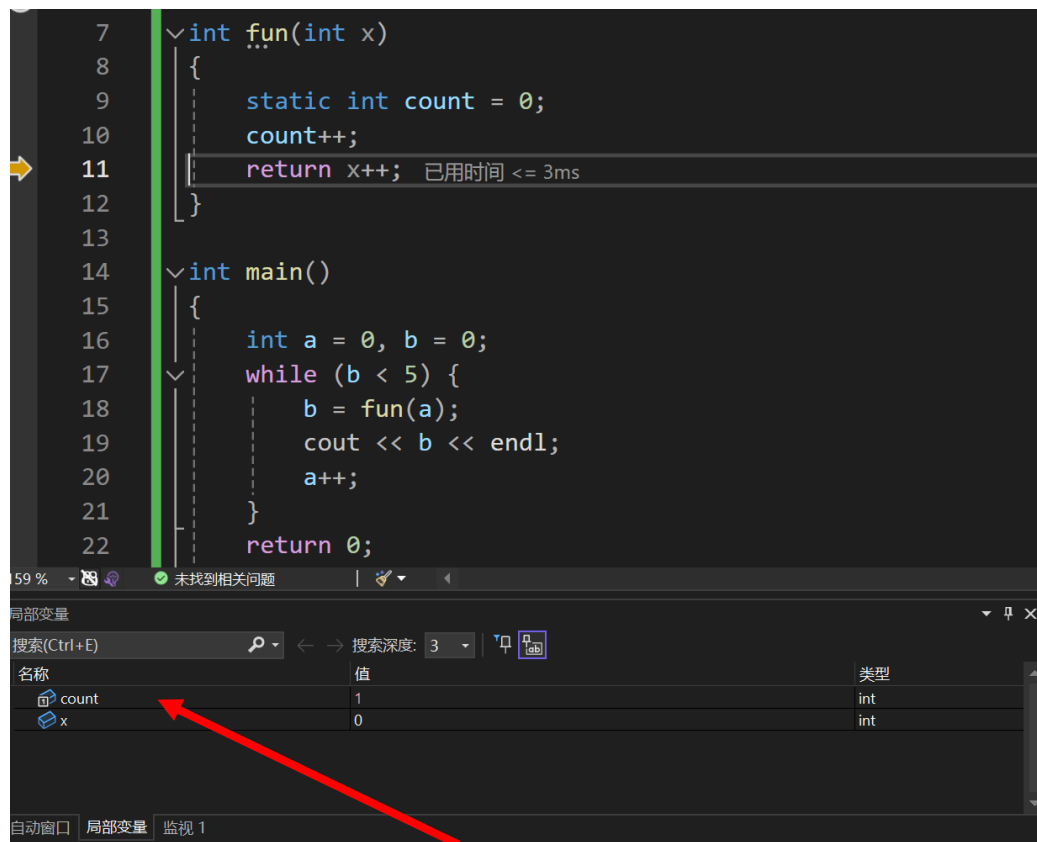
搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
n	3.0000000000000000	double
result	8.5000000000000000	double
t	4.5000000000000000	double
x	3	int

自动窗口 局部变量 监视

在调试过程中，按 窗口-局部变量 在屏幕左下角会有一个窗口，显示自动变量的变化情况。同样，当调试进入函数时，形参的变化情况也会显示。

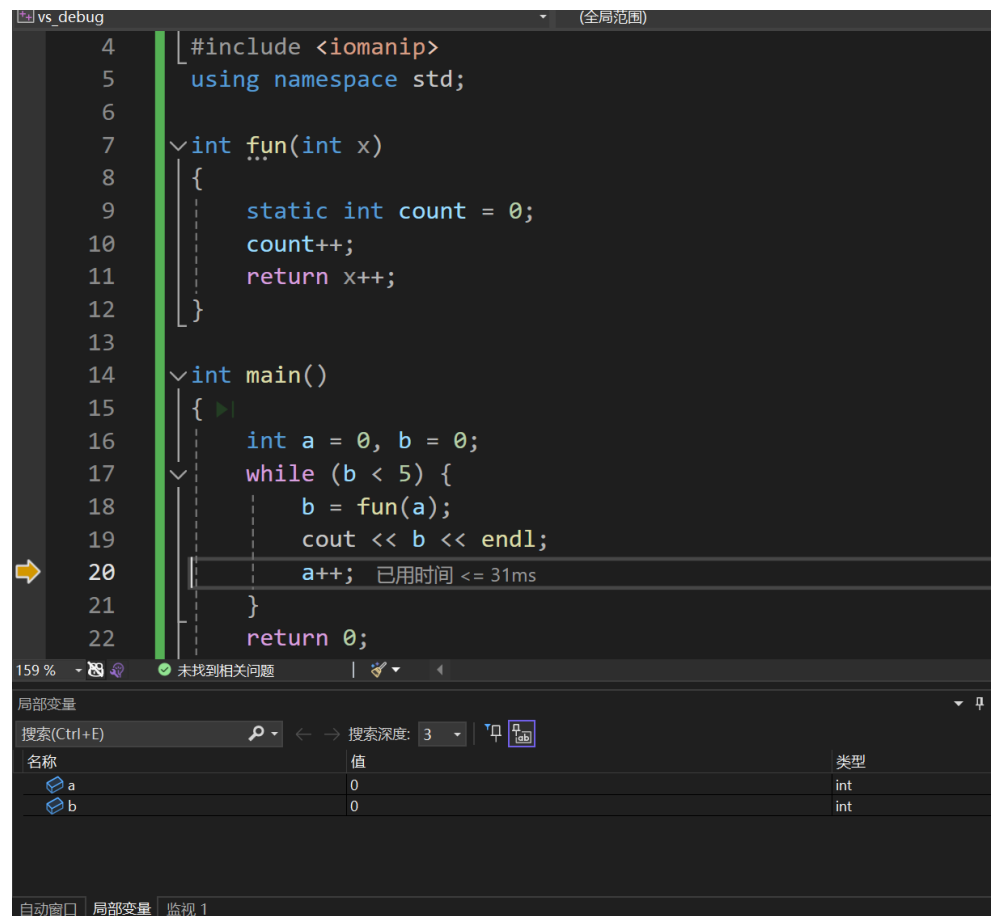
2.2 查看静态局部变量的变化情况（该静态局部变量所在的函数体内/函数体外）



```
7  int fun(int x)
8  {
9      static int count = 0;
10     count++;
11     return x++; 已用时间 <= 3ms
12 }
13
14 int main()
15 {
16     int a = 0, b = 0;
17     while (b < 5) {
18         b = fun(a);
19         cout << b << endl;
20         a++;
21     }
22     return 0;

```

名称	值	类型
count	1	int
x	0	int



```
4  #include <iomanip>
5  using namespace std;
6
7  int fun(int x)
8  {
9      static int count = 0;
10     count++;
11     return x++;
12 }
13
14 int main()
15 {
16     int a = 0, b = 0;
17     while (b < 5) {
18         b = fun(a);
19         cout << b << endl;
20         a++; 已用时间 <= 31ms
21     }
22     return 0;

```

名称	值	类型
a	0	int
b	0	int

静态局部变量同样在窗口中显示;
但只有当调试在该静态局部变量所在的函数内进行时才能被查看到, 在函数体外无法查看。

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）

The screenshot shows the Visual Studio IDE with a C++ project. The code editor displays two source files: test1.cpp and test2.cpp. test1.cpp contains a static global variable 'count' and a function 'fun1'. test2.cpp contains a function 'fun2'. The 'watch' window at the bottom shows the variable 'count' with a value of 4. A red arrow points to the 'watch' window.

```
test1.cpp
5 using namespace std;
6 int fun2(int x);
7 int fun1(int x)
8 {
9     static int count = 0;
10    count++; 已用时间 <= 12ms
11    return x++;
12 }
13
14 int main()
15 {
16     int a = 0, b = 0, c = 0;
17     while (b < 5) {
18         b = fun1(a);
19         c = fun2(a);
20         cout << "b " << b << endl;
21         cout << "c " << c << endl;
22         a++;
23     }
24 }
```

名称	值	类型
count	4	int

The screenshot shows the Visual Studio IDE with the same C++ project. The code editor displays the same two source files. The 'watch' window at the bottom shows the variable 'count' with a value of 5. A red arrow points to the 'watch' window.

```
test2.cpp
1 int fun2(int x)
2 {
3     static int count = 0;
4     count++;
5     return x--; 已用时间 <= 4ms
6 }
```

名称	值	类型
count	5	int

通过左下角的监视1 栏可以自己添加想要观察的变量；
但是当两个源程序文件静态全局变量同名时，在哪个文件中，显示的就是哪个文件中的变量。

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）

```
1  /*2351050 信05 杨瑞晨*/
2  #include <iostream>
3  using namespace std;
4  void func();
5  int n = 1;
6  int main()
7  {
8      func();
9      n++; 已用时间 <= 17ms
10     return 0;
11 }
12
```

```
1  extern int n;
2  void func()
3  {
4      n++; 已用时间 <= 14ms
5  }
```

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
n	2	int
添加要监视的项		

自动窗口 局部变量 监视 1

通过左下角的监视栏可以自己添加想要观察的变量；
而当两个源程序文件，一个定义，另一个有 extern 说明时，n 始终唯一，正常查看即可。

3.使用 VS2022 的调试工具 查看各种不同类型变量

3.1 char/int/float 等简单变量

3.2 指向简单变量的指针变量（如何查看地址、值？）

3.3 一维数组

3.4 指向一维数组的指针变量（如何查看地址、值？）

```
2      #include <iostream>
3      using namespace std;
4
5      int main()
6      {
7          char a = 'A';
8          int b = 2;
9          float c = 3.3;
10         char* pa = &a;
11         int* pb = &b;
12         float* pc = &c;
13         cout << *pa << " " << *pb << " " << *pc << endl;
14
15         int arr[5] = { 6,5,4,7,2 }; 已用时间 <= 61ms
16         int* p = arr;
17         for (int i = 0; i < 5; i++)
18             cout << *p++ << " ";
19         return 0;
20     }
21
```


局部变量		
搜索(Ctrl+E) < > 搜索深度: 3		
名称	值	类型
a	65 'A'	char
▸ arr	0x006ffba0 {6, 5, 4, 7, 2}	int[5]
b	2	int
c	3.29999995	float
i	-858993460	int
▸ p	0x006ffba0 {6}	int *
▸ pa	0x006ffbfb <字符串中的字符无效。 > 查看 ▾	char *
▸ pb	0x006ffbec {2}	int *
▸ pc	0x006ffbe0 {3.29999995}	float *
自动窗口 局部变量 监视 1		

简单变量：同2。（局部变量通过局部变量栏查看，全局变量通过在监视1中输入变量名来查看。）









指针：地址在值一栏显示，而后面的大括号内则是具体的值，指向一维数组的指针变量的值为数组首元素

但 char* 貌似无法显示字符值，只能显示目标地址，但输出值正确。

（若 pa 是指向字符的指针，cout << pa; 显示的是 pa 指向的字符串，而不是 pa 变量的值，这是指向字符的指针比较特殊的地方）

3.5 二维数组（包括数组名仅带一个下标的情况）

```
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int a[2][3] = { 1,2,3,4 };
8      int b[][3] = { {7,8},{9,10} };
9      return 0; 已用时间 <= 6ms
10 }
11
```

名称	值	类型
▸  a	0x012ffcec {0x012ffcec {1, 2, 3}, 0x012ffcf8 {4, 0, 0}}	int[2][3]
▸  b	0x012ffccc {0x012ffccc {7, 8, 0}, 0x012ffcd8 {9, 10, 0}}	int[2][3]
▾  a	0x012ffcec {0x012ffcec {1, 2, 3}, 0x012ffcf8 {4, 0, 0}}	int[2][3]
▸  [0]	0x012ffcec {1, 2, 3}	int[3]
▸  [1]	0x012ffcf8 {4, 0, 0}	int[3]
▾  b	0x012ffccc {0x012ffccc {7, 8, 0}, 0x012ffcd8 {9, 10, 0}}	int[2][3]
▸  [0]	0x012ffccc {7, 8, 0}	int[3]
▸  [1]	0x012ffcd8 {9, 10, 0}	int[3]

直接显示的是全部二维数组的地址值，点击二维数组名，会出现各个一维数组的地址和值；在数组名仅带一个下标时，编译器会自动补全，同样显示的是全部二维数组的地址值，点击二维数组名，会出现各个一维数组的地址和值。

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？

```
5 void swap(int* a)
6 {
7     int temp;
8     temp = a[0];
9     a[0] = a[1];
10    a[1] = temp;
11 }
12 int main()
13 {
14     int a[2] = { 1,2 };
15     swap(a); 已用时间 <= 1ms
16     return 0;
17 }
18
```

120 % 未找到相关问题

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
a	0x009af9cc {1, 2}	int[2]

添加要监视的项

实参一维数组名<=>指向一维数组首地址的指针
函数中无法直接查看实参数组的地址和值，
只可看到数组的首地址和首地址值，
但是可以通过监视指针来间接达到该效果。

```
1 /*2351050 信05 杨瑞晨*/
2 #include <iostream>
3 using namespace std;
4
5 void swap(int* a)
6 {
7     int temp;
8     temp = a[0]; 已用时间 <= 9ms
9     a[0] = a[1];
10    a[1] = temp;
11 }
12 int main()
13 {
14     int a[2] = { 1,2 };
15     swap(a);
16     return 0;
17 }
18
```

120 % 未找到相关问题

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
a	0x00f3fd68 {1}	int *
a+1	0x00f3fd6c {2}	int *

添加要监视的项

```
1 /*2351050 信05 杨瑞晨*/
2 #include <iostream>
3 using namespace std;
4
5 void swap(int* a)
6 {
7     int temp;
8     temp = a[0];
9     a[0] = a[1];
10    a[1] = temp;
11 } 已用时间 <= 5ms
12 int main()
13 {
14     int a[2] = { 1,2 };
15     swap(a);
16     return 0;
17 }
18
```

20 % 未找到相关问题

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
a	0x00f3fd68 {2}	int *
a+1	0x00f3fd6c {1}	int *

添加要监视的项

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？）

```
1  /*2351050 信05 杨瑞晨*/
2  #include <iostream>
3  using namespace std;
4
5
6  int main()
7  {
8      const char* p = "六五四七二";
9      return 0; 已用时间 <= 1ms
10 }
11
```

20 % 未找到相关问题

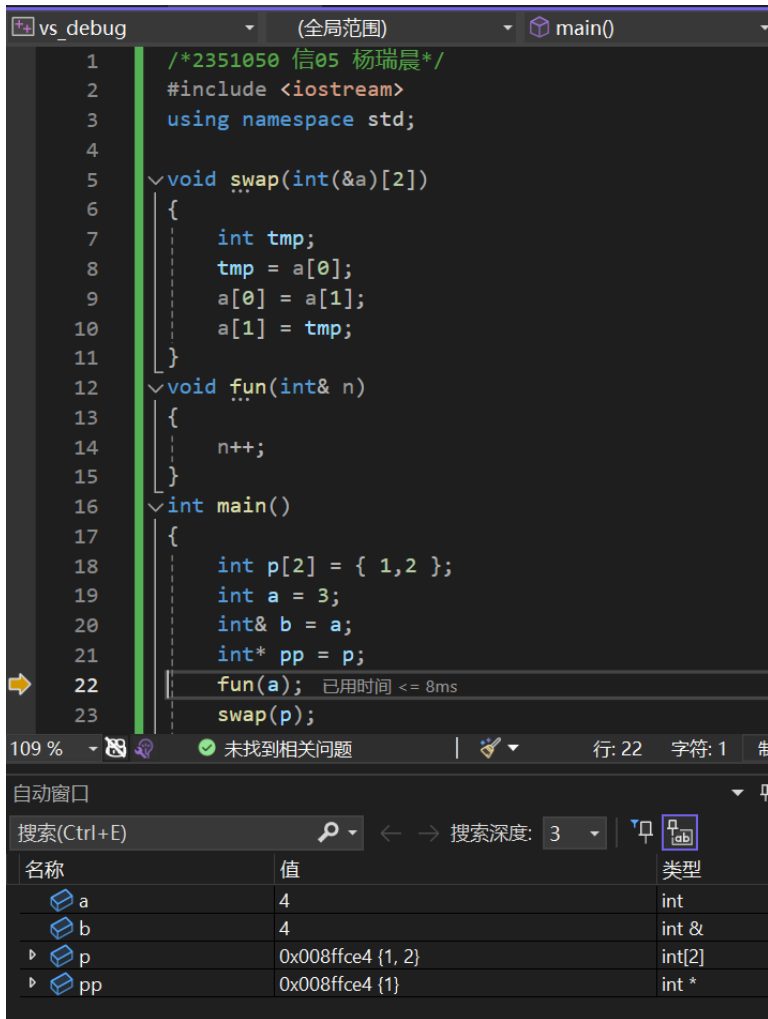
局部变量

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
p	0x00069bd4 "六五四七二"	const char *

可以（红色横线处）

3.8 引用（引用与指针是否有区别？ 有什么区别？）



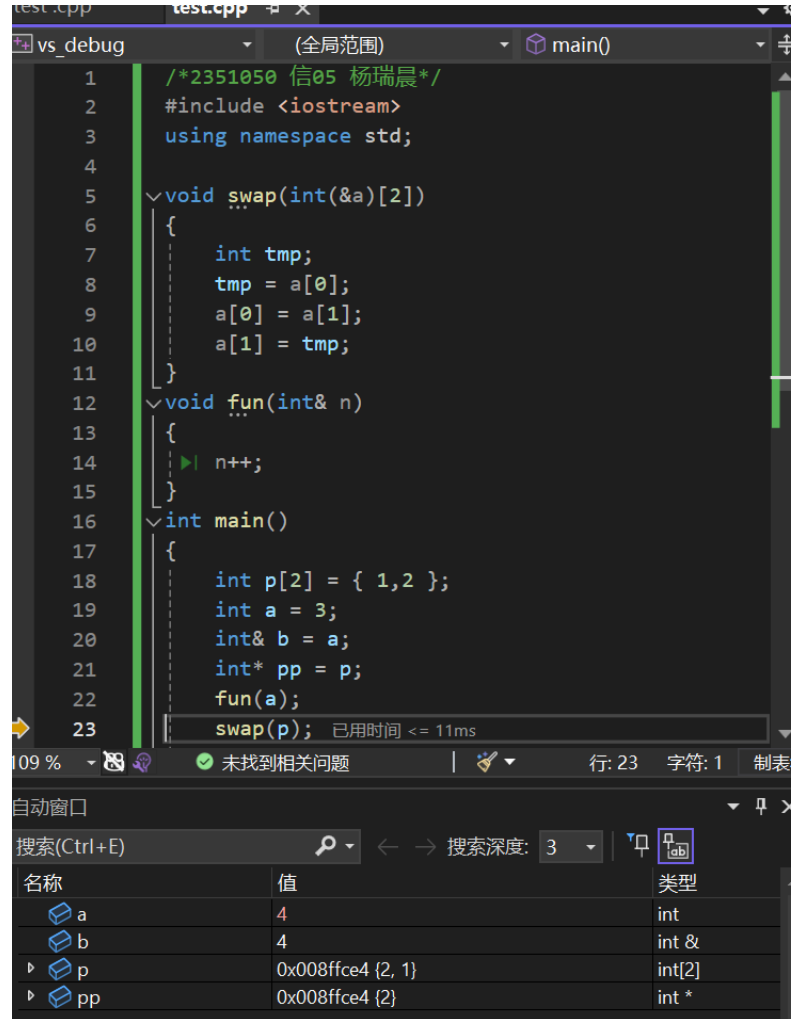
```
1  /*2351050 信05 杨瑞晨*/
2  #include <iostream>
3  using namespace std;
4
5  void swap(int(&a)[2])
6  {
7      int tmp;
8      tmp = a[0];
9      a[0] = a[1];
10     a[1] = tmp;
11 }
12 void fun(int& n)
13 {
14     n++;
15 }
16 int main()
17 {
18     int p[2] = { 1,2 };
19     int a = 3;
20     int& b = a;
21     int* pp = p;
22     fun(a); 已用时间 <= 8ms
23     swap(p);
```

109 % 未找到相关问题 | 行: 22 字符: 1

自动窗口

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
a	4	int
b	4	int &
p	0x008ffce4 {1, 2}	int[2]
pp	0x008ffce4 {1}	int *



```
1  /*2351050 信05 杨瑞晨*/
2  #include <iostream>
3  using namespace std;
4
5  void swap(int(&a)[2])
6  {
7      int tmp;
8      tmp = a[0];
9      a[0] = a[1];
10     a[1] = tmp;
11 }
12 void fun(int& n)
13 {
14     n++;
15 }
16 int main()
17 {
18     int p[2] = { 1,2 };
19     int a = 3;
20     int& b = a;
21     int* pp = p;
22     fun(a);
23     swap(p); 已用时间 <= 11ms
```

109 % 未找到相关问题 | 行: 23 字符: 1

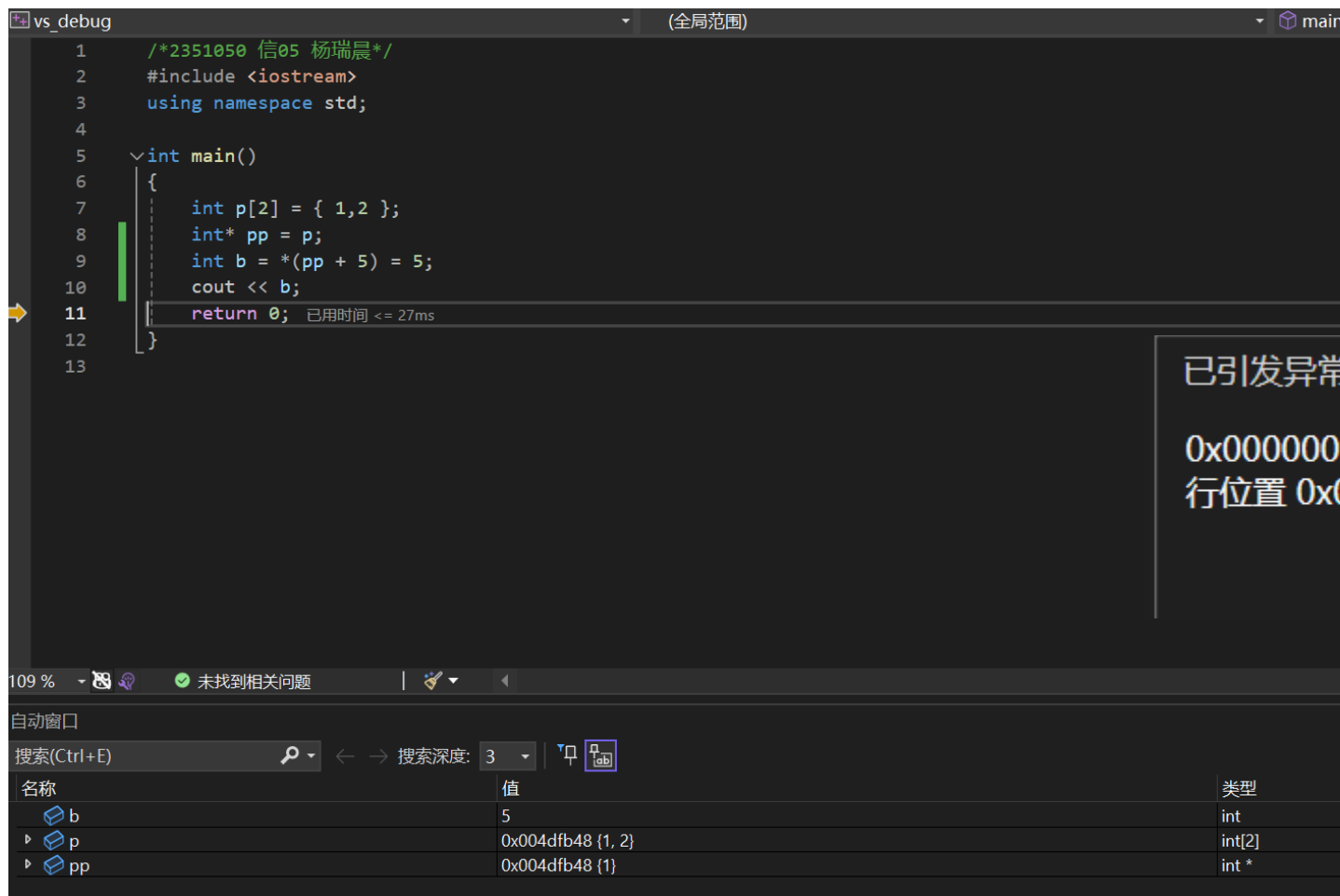
自动窗口

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
a	4	int
b	4	int &
p	0x008ffce4 {2, 1}	int[2]
pp	0x008ffce4 {2}	int *

由调试可知引用是变量的别名，与原变量共享地址；而指针是存放地址的变量。在函数中使用引用或指针都可以改变实参。

3.9 使用指针时出现了越界访问



```
1  /*2351050 信05 杨瑞晨*/
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int p[2] = { 1, 2 };
8      int* pp = p;
9      int b = *(pp + 5) = 5;
10     cout << b;
11     return 0; 已用时间 <= 27ms
12 }
13
```

已引发异常

0x00000005 处(位于 vs_debug.exe 中)引发的异常: 0xC0000005: 执行位置 0x00000005 时发生访问冲突。

自动窗口

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
b	5	int
p	0x004dfb48 {1, 2}	int[2]
pp	0x004dfb48 {1}	int *

在调试时可以看到越界赋值成功。

但是会引发异常，因为访问了非法地址，在程序结束时报错。