

同济大学计算机系

VS2022 调试实验报告



学 号	1850772
姓 名	张哲源
专 业	计算机科学与工程
授课老师	沈坚老师

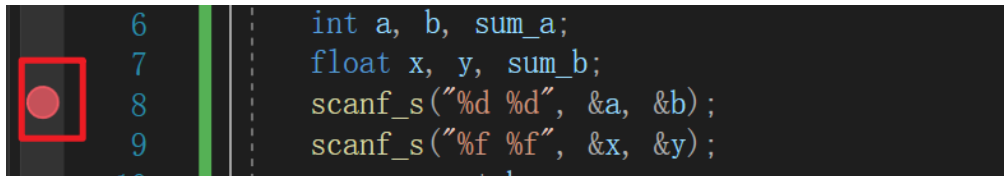
目录（使用手册）

一、	Vs2022 下调试工具的基本使用方法.....	3
	开始/关闭调试.....	3
	单步执行语句（两种方式）.....	4
	碰到系统类函数（cout/sqrt 等）一步完成不进入内部调试使用逐语句即可（f11） .	4
	进入到系统函数内部（cout/sqrt 等）如何跳出并返回自己的函数（shift+f11）	4
	碰到自定义的函数调用语句，如何一步完成而不进入自定义（f10）	5
	在碰到自定义的函数的调用语句（例如在 main 函数中调用自定义的 fun 函数）时，如何转到被调用函数中单步执行（f11）	5
二、	Vs2022 查看变量等相关信息的方法.....	5
	打开显示信息的窗口（调试状态下才能打开）	6
	查看形参/自动变量.....	7
	自动窗口的变量的显示（不会显示全部，可以根据需要添加监视）	8
	数组和二维数组.....	8
	结构体.....	9
	指向数组和结构体的指针	9
	引用.....	10
	指向字符串常量的指针变量（查看无名字符串常量的地址）	10
	使用指针出现越界访问的情况.....	10

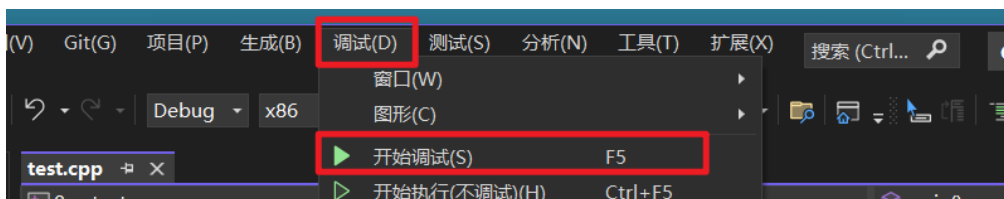
一、Vs2022 下调试工具的基本使用方法

开始/关闭调试

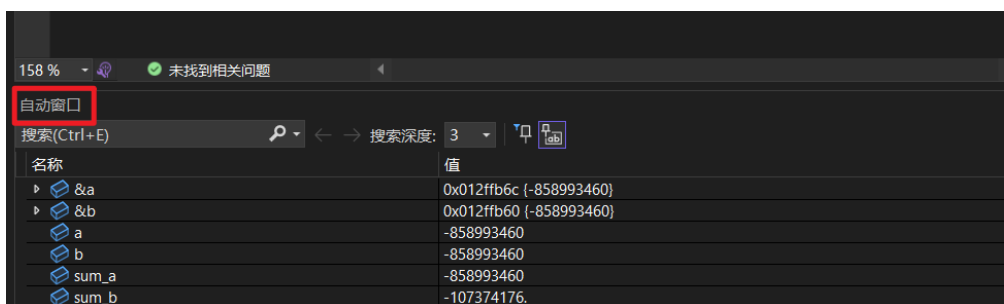
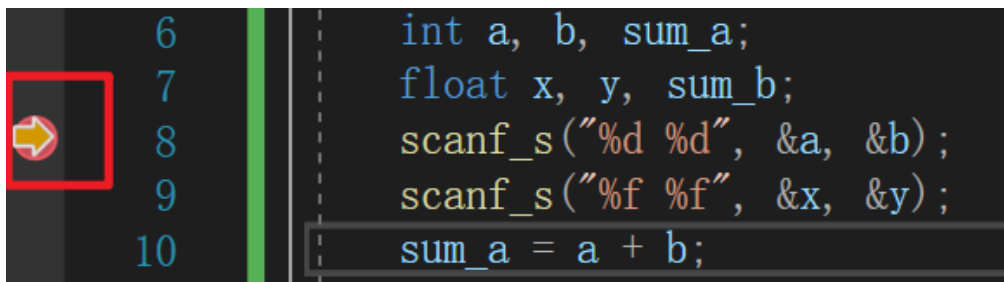
设置断点



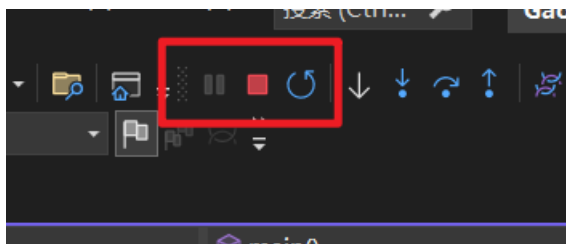
开始调试 (F5)



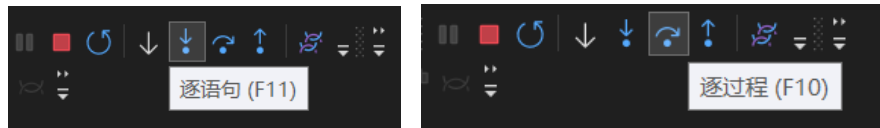
进入调试界面



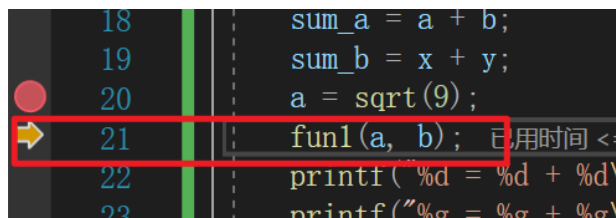
结束调试 (shift+f5)



单步执行语句（两种方式）

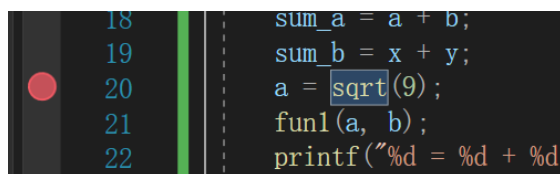
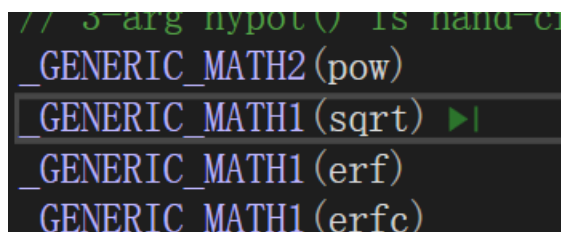


碰到系统类函数（cout/sqrt 等）一步完成不进入内部调试使用逐语句即可（f11）



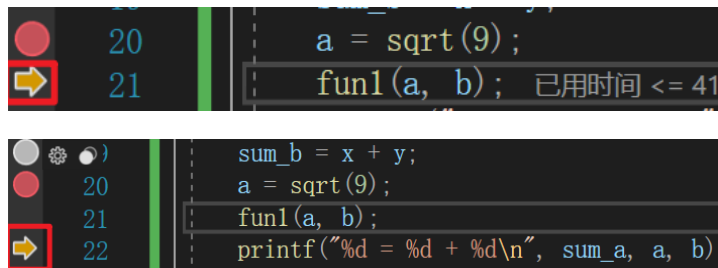
（直接跳过）

进入到系统函数内部（cout/sqrt 等）如何跳出并返回自己的函数（shift+f11）

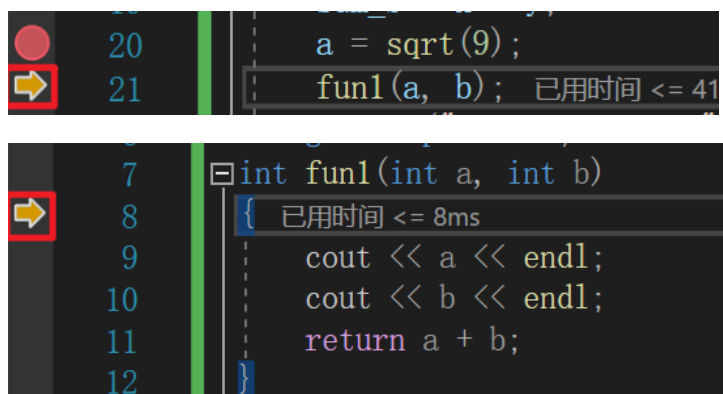


（回到原界面）

碰到自定义的函数调用语句，如何一步完成而不进入自定义
(f10)

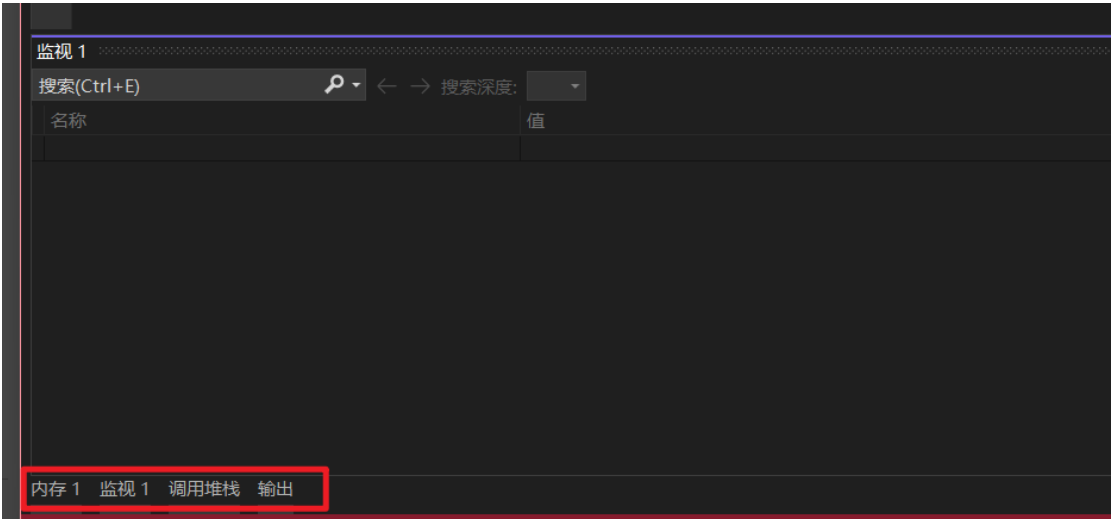
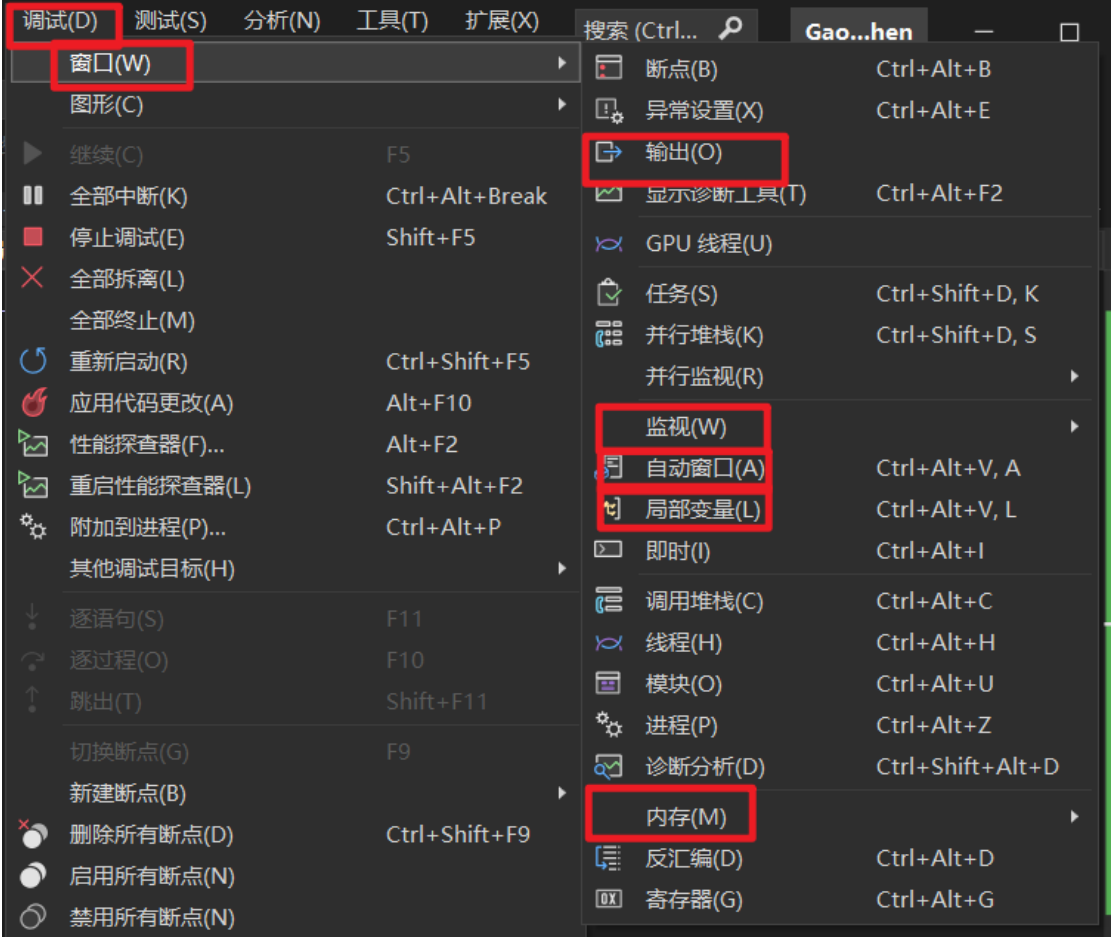


在碰到自定义的函数的调用语句（例如在 main 函数中调用自定义的 fun 函数）时，如何转到被调用函数中单步执行 （f11）



二、Vs2022 查看变量等相关信息的方法

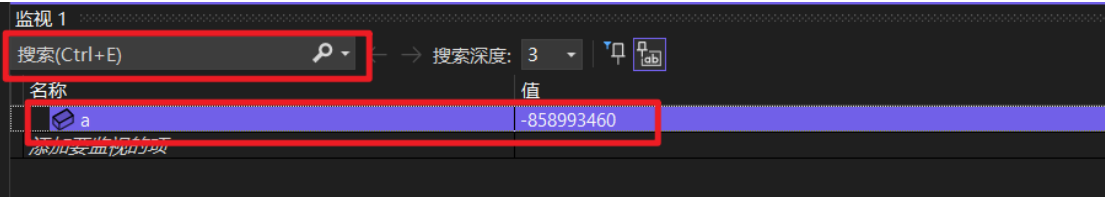
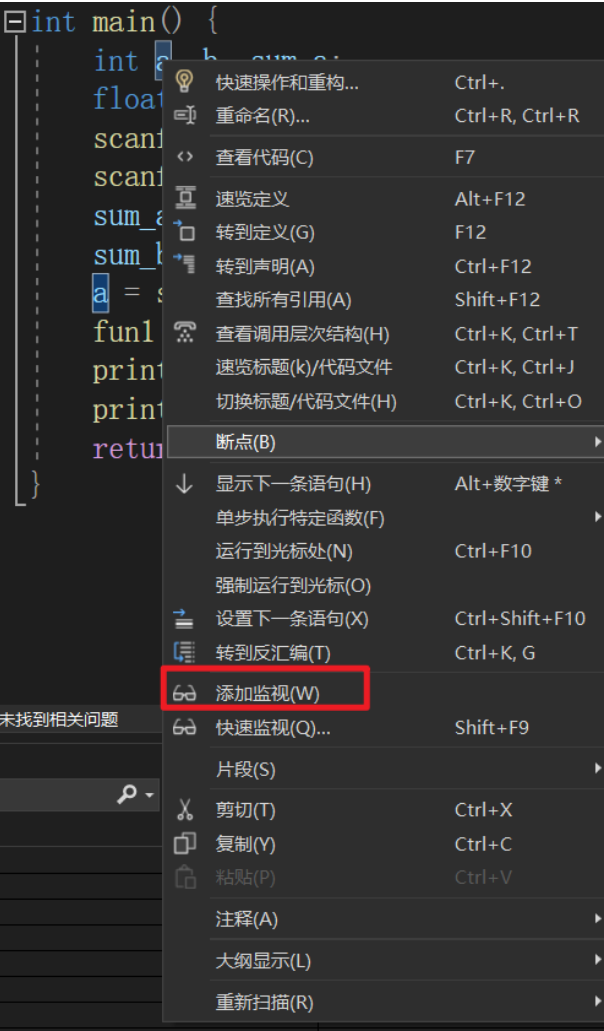
打开显示信息的窗口（调试状态下才能打开）



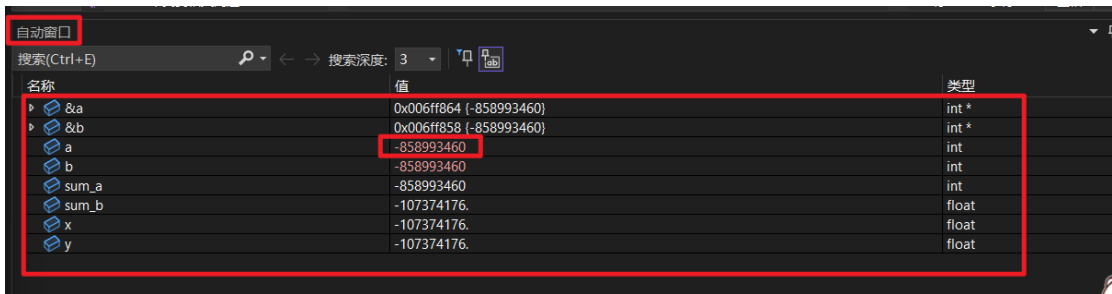
调到舒服的位置，就可以在下方查看了

查看形参/自动变量

右键一个变量，添加监视，变量可以是全局变量，静态变量，指针变量等（也可以在搜索框里面搜索，不同作用域的变量重名会都显示）



自动窗口的变量的显示（不会显示全部，可以根据需要添加监视）



数组和二维数组

同样也可以添加监视
一维数组

```
int buffer[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

名称	值
buffer	0x0020fa40 {1, 2, 3, 4, 5, 6, 7, 8, 9}
[0]	1
[1]	2
[2]	3
[3]	4
[4]	5
[5]	6
[6]	7
[7]	8

二维数组

```
int map[10][10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
```

名称	值
map	0x009cf798 {0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, 0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, 0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, 0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, 0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, 0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, 0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, 0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, 0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, 0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}}
[0]	0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}
[1]	0x009cf7c0 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
[2]	0x009cf7e8 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
[3]	0x009cf810 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
[4]	0x009cf838 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
[5]	0x009cf860 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
[6]	0x009cf888 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
[7]	0x009cf8b0 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
[8]	0x009cf8d8 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
[9]	0x009cf8f0 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

map	0x009cf798 {0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, 0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}}
map[0]	0x009cf798 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}
map[0][0]	1
map[0][1]	2
map[0][2]	3

结构体

结构体也可以增加监视

```
student s1;

enum Sex{ male, female };

struct student
{
    char name[20];
    int no;
    Sex sex;
};
```

female	female (1)
s1	{name=0x009cf774 "蔡徐坤" no=123 sex=female (1) }
s1.name	0x009cf774 "蔡徐坤"
s1.no	123
s1.sex	female (1)
s1.name	0x009cf774 "蔡徐坤"

指向数组和结构体的指针

```
int buffer[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int map[10][10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
student s1;
student* p_s1 = &s1;
int* p_map = map[0];
int* p_buffer = buffer;
```

显示当前指向的地址和改地址存储的内容，以及指针的基类型

p_map	0x0073fca8 (1)	int *
p_s1	1	int
p_s1	0x0073fc84 {name=0x0073fc84 "qwe" no=123 sex=-858993460 }	student *
name	0x0073fc84 "qwe"	char[20]
no	123	int
sex	-858993460	Sex
p_buffer	0x0073fe40 (1)	int *
	1	int

引用

```
int a = 10;
int* p = &a;
int& b = a;
```

&a	0x009ffc7c (10)	int *
a	10	int
p	0x009ffc7c (10)	int *

指向字符串常量的指针变量（查看无名字符串常量的地址）

```
const char* p = "坤坤打篮球";
```

p_buffer	未定义标识符 'p_buffer'	
p	0x00b37bcc "坤坤打篮球"	const char *
	-64 '?'	const char

使用指针出现越界访问的情况

只出现地址，可能不显示指向的内容，或指向的内容不可信
非法写入时会弹窗

```

int main() {
    int buffer[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int* p = buffer;

    cout << (p + 11) << ": " << *(p + 11) << endl;
    for (int i = 0; i < 11; i++) 已用时间 <= 5ms
        p++;
    *(p + 11) = 12;

    return 0;
}

```

值不可信

 p	0x00e8f82c {-858993460}
	-858993460

非法写入会弹窗警告

```

if (IsProcessorFeaturePresent(PF_FASTFAIL_AVAILABLE))
{
    __fastfail(FAST_FAIL_STACK_COOKIE_CHECK_FAILURE);
}

```

未经处理的异常

0x001D384A 处有未经处理的异常(在 0__test_area.exe 中): 堆栈 Cookie 检测代码检测到基于堆栈的缓冲区溢出。