



同濟大學

Tongji University

《高级语言程序设计》

实验报告III

报告名称: VS2022调试工具的使用与总结

班 级: 软件工程

学 号: 2253744

姓 名: 林觉凯

完成日期: 2023年12月9日

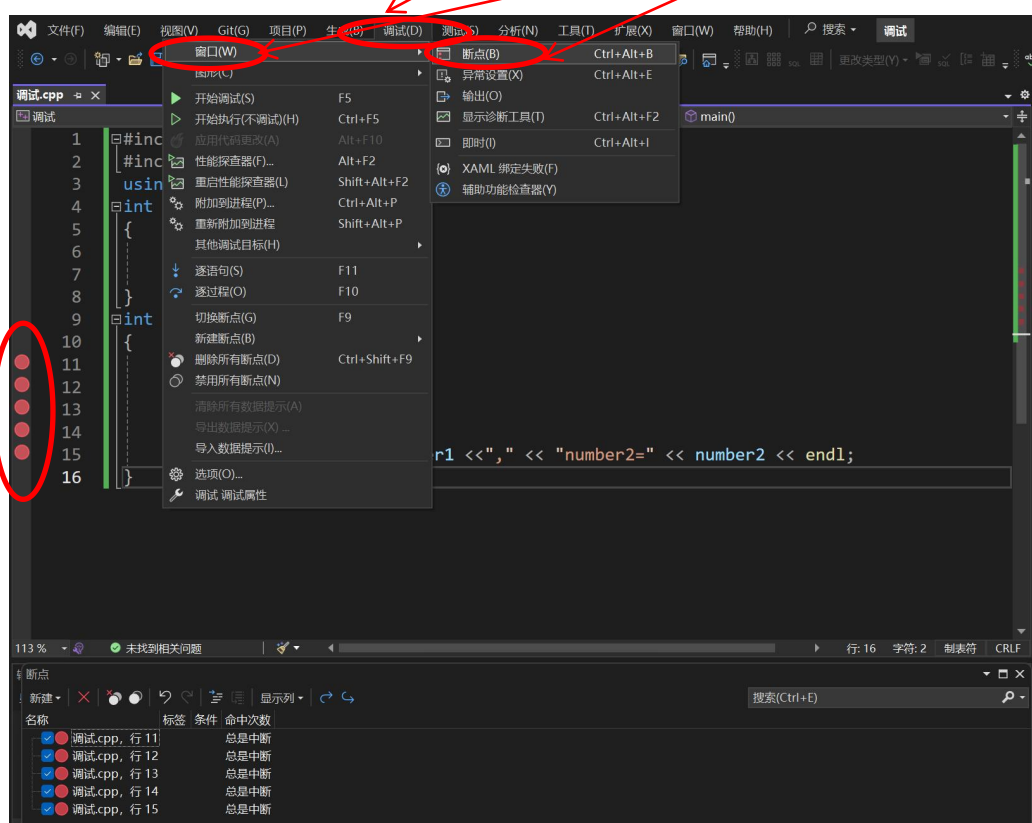
1. 调试工具的基本使用方法

1.1 开始/结束调试与配置

可在指定程序行的左侧点击添加断点，再次点击红点则取消该断点

点击“调试”

选择“窗口”中的“断点”



点击此按钮/按 F5 进入调试



点击此按钮/按 shift+F5 结束调试



在默认的情况下，开始调试后程序会连续执行到结束或者是等待输出；通过添加断点可以在程序将要执行到的指定位置时停止执行。

本页涉及知识点 1.1

1.2 单步执行与调用函数执行

单步执行时，程序执行下一行/下一个表达式；若为自定义的函数调用，则跳入被调用的函数，执行被调用的函数的第一行。

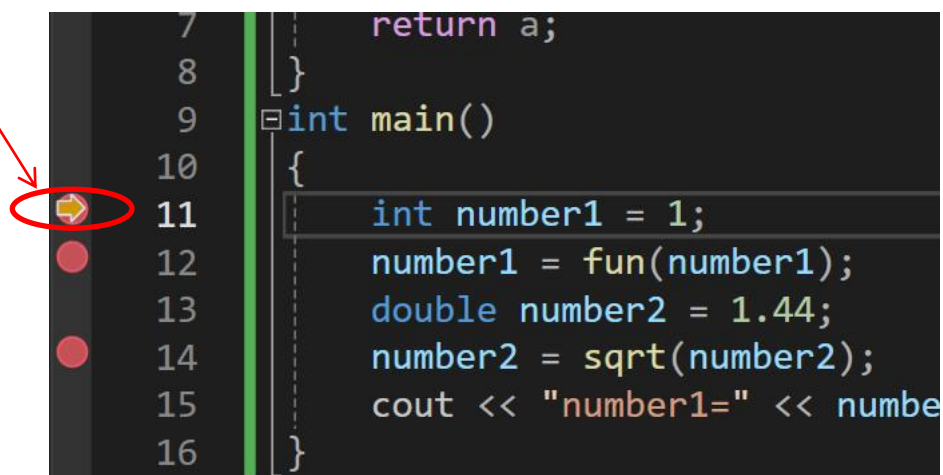
以下为调试程序：

```
#include <iostream>
#include <cmath>
using namespace std;
int fun(int a)
{
    a++;
    return a;
}
int main()
{
    int number1 = 1;
    number1 = fun(number1);
    double number2 = 1.44;
    number2 = sqrt(number2);
    cout << "number1=" << number1 << ", " << "number2=" << number2 << endl;
}
```

点击此按钮/按F11进行单步执行




左侧箭头指示
下一条指令，
即下一次单步
执行将要执行
的指令



本页涉及知识点 1.2, 1.3

非自定义的标准库函数/系统函数/...类的使用视为一个完整的语句，单步执行不会进入其内部,该语句执行完后便跳出并返回自己的函数。

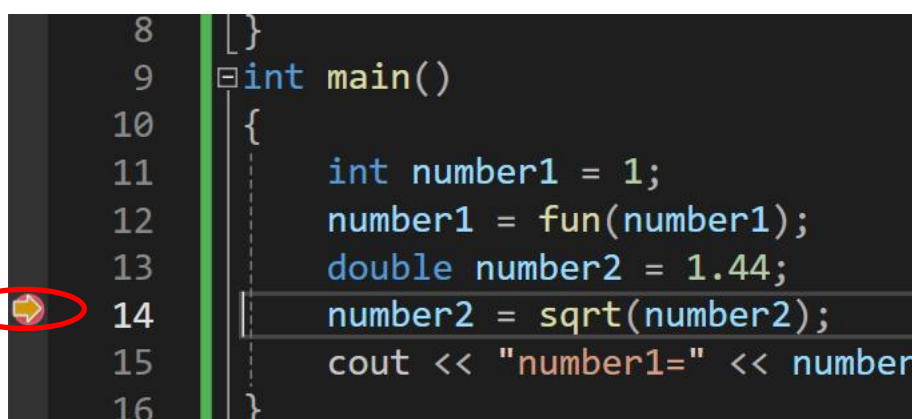
下一条执行语句为自定义函数 fun, 则跳入被调用的函数, 执行被调用的函数的第一行



```

4  int fun(int a)
5  {
6      a++;
7      return a;
8  }
9  int main()
10 {
11     int number1 = 1;
12     number1 = fun(number1);
13     double number2 = 1.44;
14     number2 = sqrt(number2);
15     cout << "number1=" << number1;
16 }
    
```

下一条执行语句为标准库函数 sqrt, 视为一个完整的语句，单步执行不会进入其内部。



```

8  }
9  int main()
10 {
11     int number1 = 1;
12     number1 = fun(number1);
13     double number2 = 1.44;
14     number2 = sqrt(number2);
15     cout << "number1=" << number1;
16 }
    
```

1.3 逐过程执行与跳出函数

单步执行时，若发生自定义的函数调用，执行流会跳转入被调用函数中执行单步执行；如果想要将函数调用视为单个语句一次性执行，不会停留在被调用的函数内，则需要使用逐过程执行。



点击此按钮/按下
F11 进行逐过程
执行

当执行流停在函数内时，可以选择一次性完成该函数的执行，跳出函数。



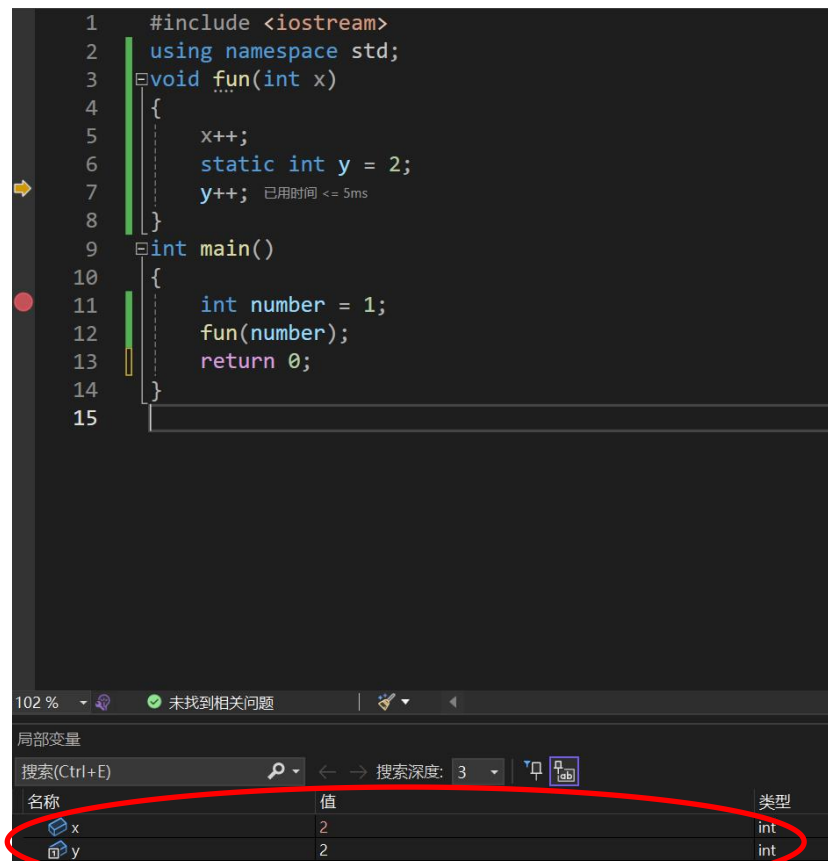
点击此按钮/按下
Shift+F11 完成当
前函数执行，跳
出函数

本页涉及知识点 1.5, 1.6

2. 查看各种生存期/作用域变量

2.1,2.2 形参/自动变量与静态局部变量

```
#include <iostream>
using namespace std;
void fun(int x)
{
    x++;
    static int y = 2;
    y++;
}
int main()
{
    int number = 1;
    fun(number);
    return 0;
}
```



局部变量窗口内显示当前执行的位置所在作用域内所有可访问的局部变量(形参、静态全局变量)和当前其他的值。在本程序中可以观察到形参 x 和静态局部变量 y 的数值的变化。注意，只有当调试到该静态局部变量所在函数体内才能观察到该静态局部变量的值，在函数(本题 `fun` 函数)体外无法查看

2.3 静态全局变量

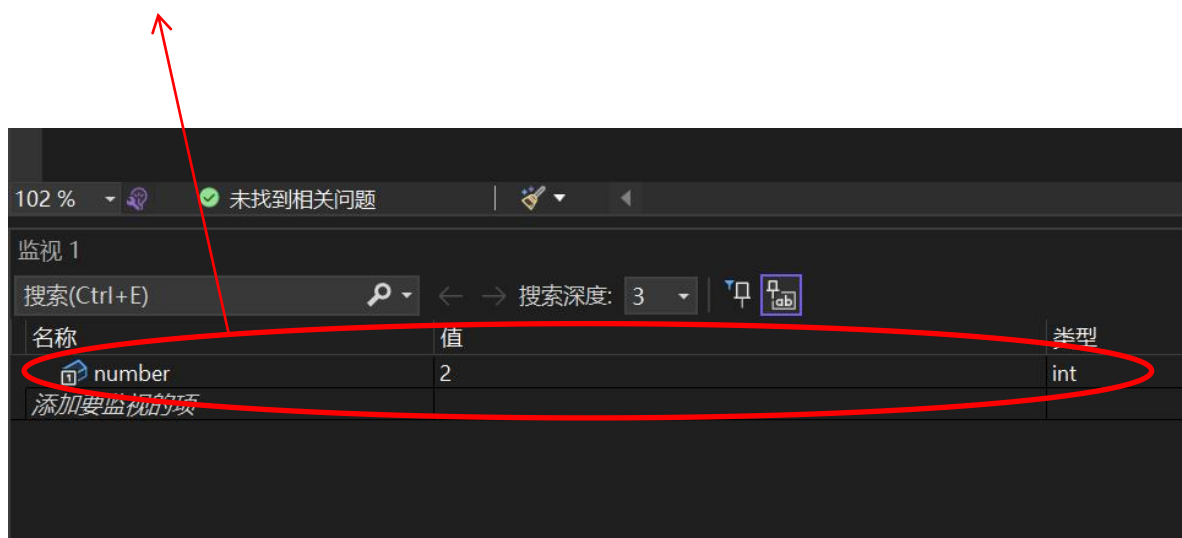
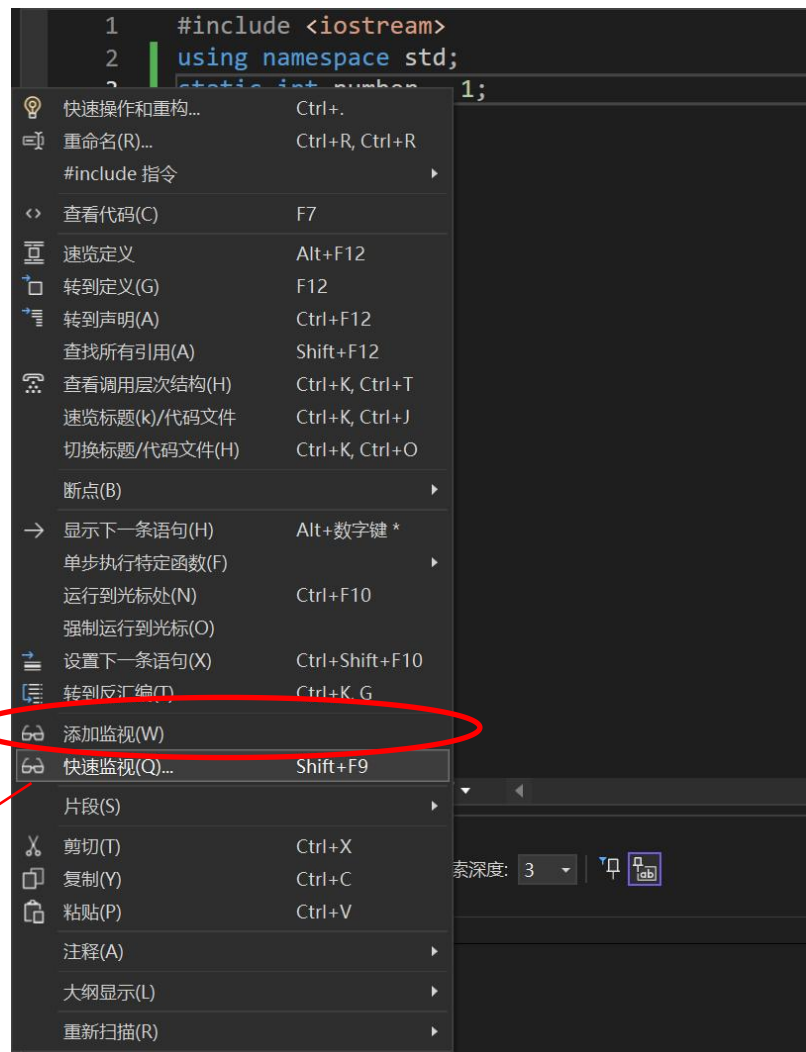
调试 1.cpp

```
#include <iostream>
using namespace std;
static int number = 1;
int fun(int x);
int main()
{
    int a = 1;
    a = fun(a);
    return 0;
}
```

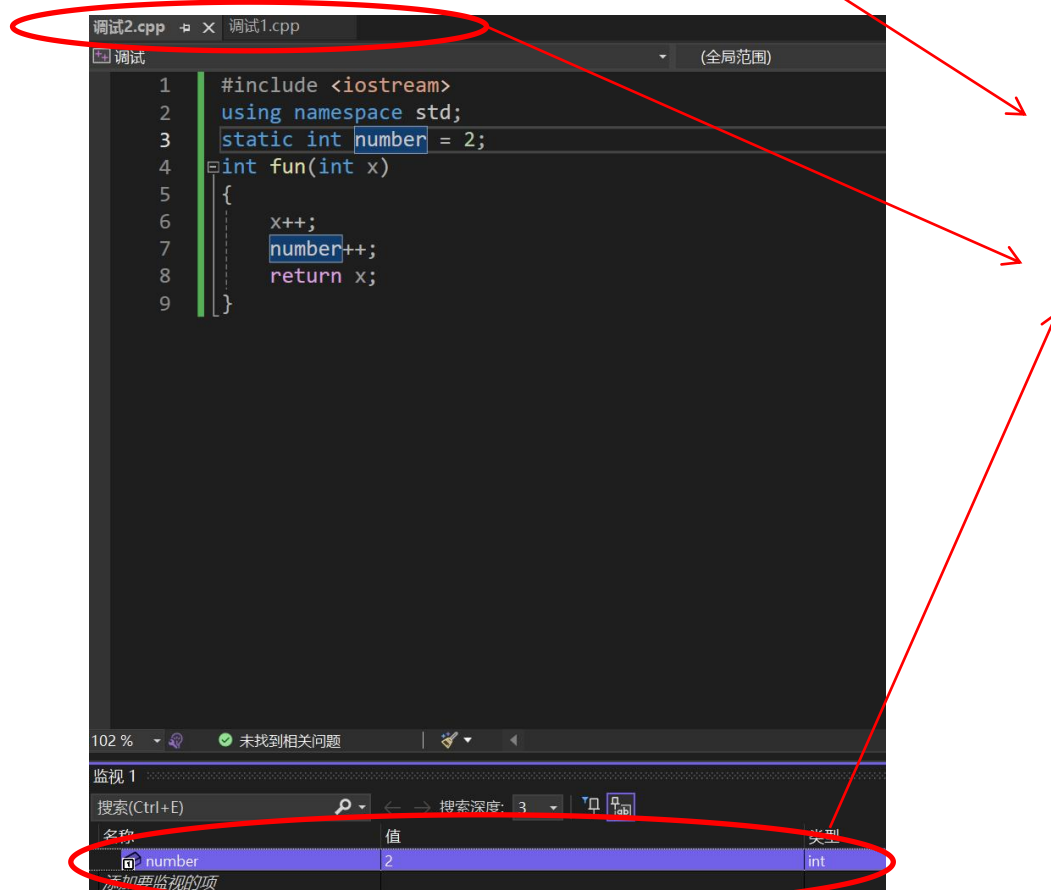
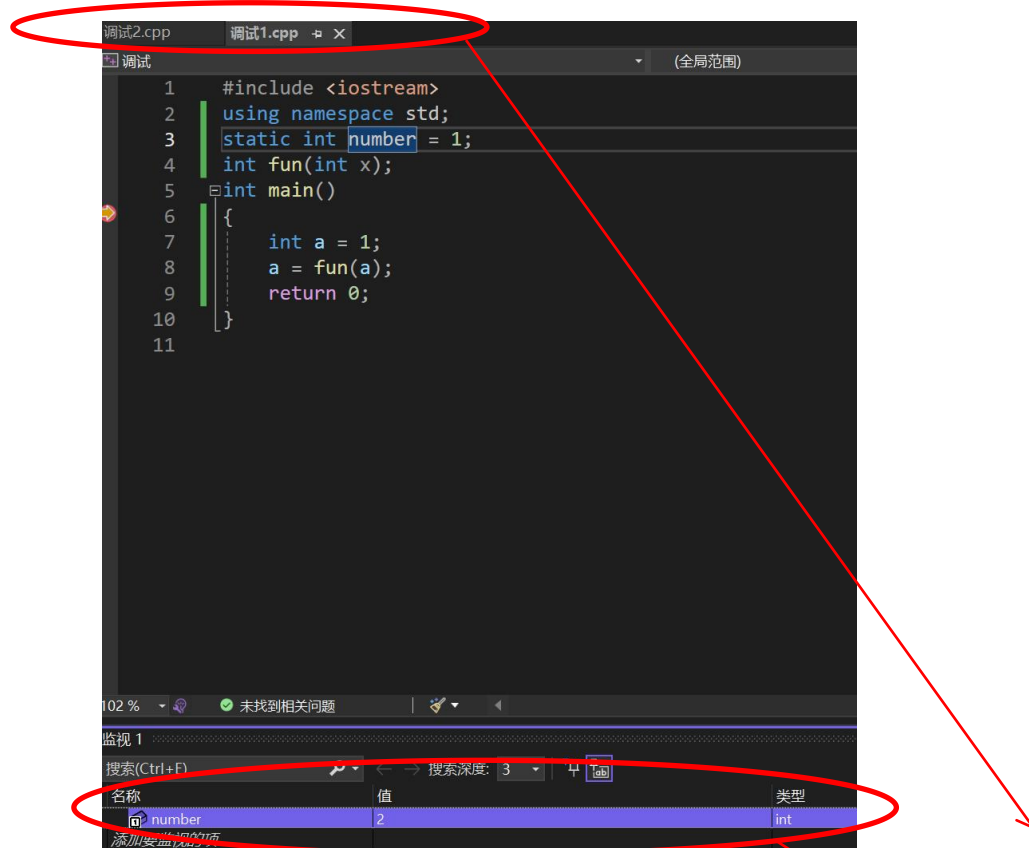
调试 2.cpp

```
#include <iostream>
using namespace std;
static int number = 2;
int fun(int x)
{
    x++;
    number++;
    return x;
}
```

全局变量需要手动添加监视，添加监视后在监视窗口显示当前程序执行处添加的监视的变量名相同的变量



本页涉及知识点 2.3



在这两个程序中都添加静态全局变量 `number`，本程序的监视窗口中的 `number` 都是调试 2.cpp 中的 `number` 的监视值。这样的做法非常容易搞混，比较难以区分。

本页涉及知识点 2.3

2.4 外部全局变量

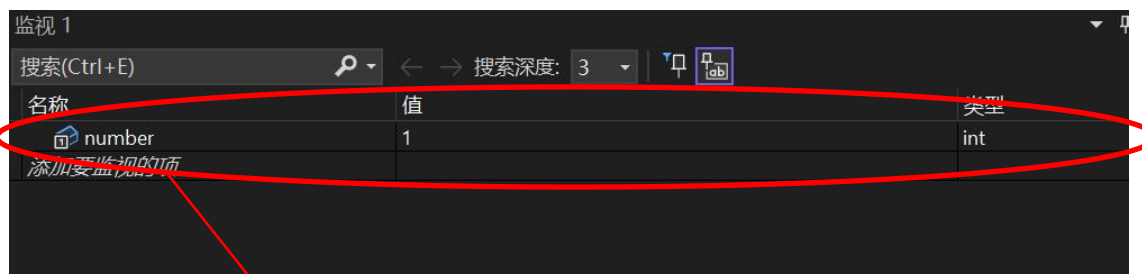
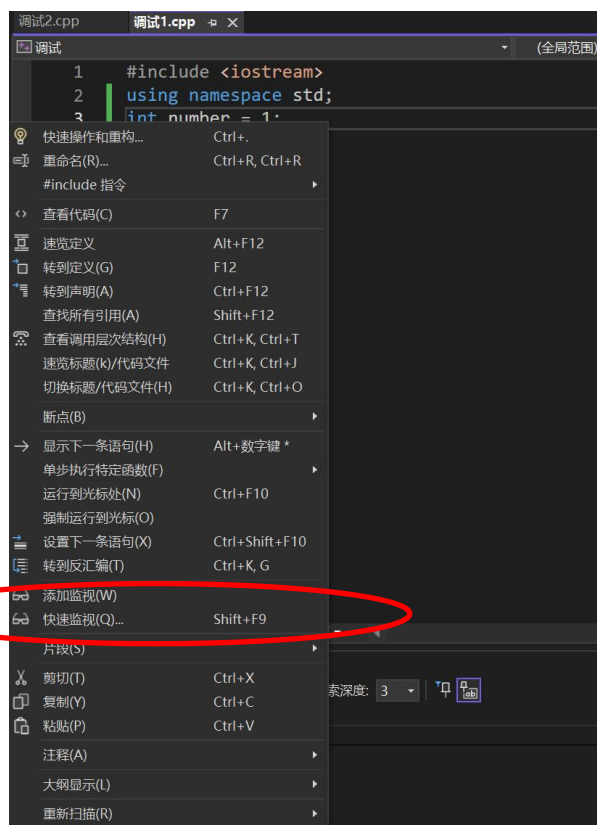
调试 1.cpp

```
#include <iostream>
using namespace std;
int number = 1;
int fun(int x);
int main()
{
    int a = 1;
    a = fun(a);
    return 0;
}
```

调试 2.cpp

```
#include <iostream>
using namespace std;
extern int number;
int fun(int x)
{
    x++;
    number++;
    return x;
}
```

外部全局变量需要手动添加监视，添加监视后在监视窗口显示当前程序执行处添加的监视的变量名相同的变量



当一个 cpp 有定义、另一个 cpp 中有 extern 说明时，在下方的监视窗口中的 number 是同一个正常查看即可

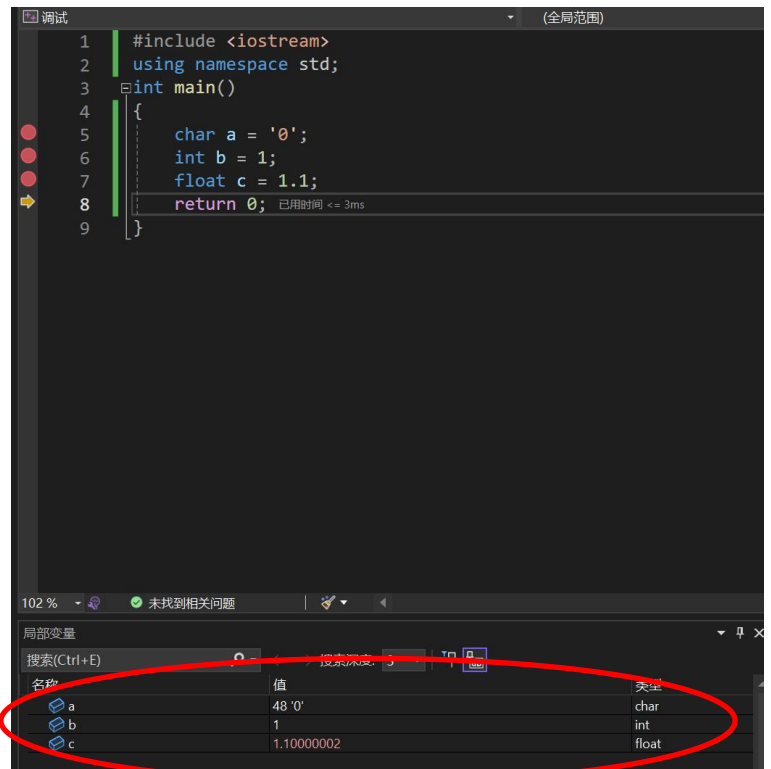
本页涉及知识点 2.4

3. 查看不同类型的变量

3.1 简单变量

```
#include <iostream>
using namespace std;
int main()
{
    char a = '0';
    int b = 1;
    float c = 1.1;
    return 0;
}
```

对于简单类型的变量,在下方的窗口内直接查看其显示的值。



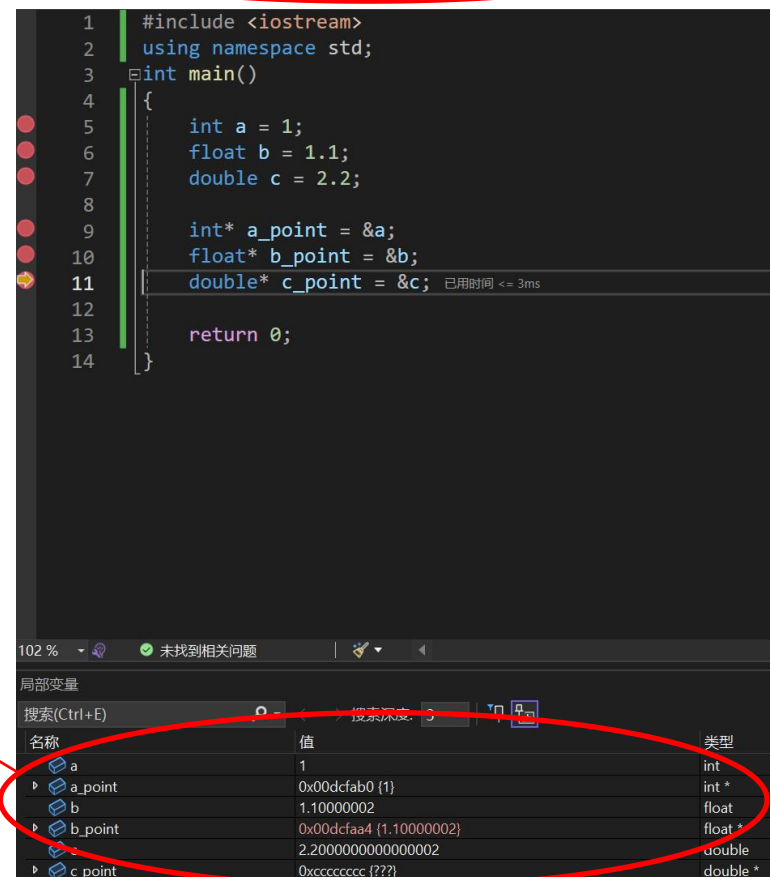
3.2 指向简单变量的指针

```
#include <iostream>
using namespace std;
int main()
{
    int a = 1;
    float b = 1.1;
    double c = 2.2;

    int* a_point = &a;
    float* b_point = &b;
    double* c_point = &c;

    return 0;
}
```

对于指向简单类型的变量的指针,在下方的窗口内查看 {} 内显示的即为其所指向地址位置的值;如果还未访问到或者越界访问,则 {} 内为???



本页涉及知识点 3.1,3.2

3.3,3.4 一维数组及其指针

```
#include <iostream>
using namespace std;
int main()
{
    int a[] = { 0,1,2,3,4 };
    int* b = a;

    return 0;
}
```

对于一维数组,在下方的窗口内会显示其首地址值并在后面的 {} 内显示一维数组内显示的所有值;对于指向一维数组的指针,在下方的窗口内显示该数组的首地址和数组内的第一个元素值;如果访问越界,则其显示的值不可信。

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

名称	值	类型
a	0x00f3fae8 {0, 1, 2, 3, 4}	int[5]
b	0x00f3fae8 {0}	int *

3.5 二维数组

```
#include <iostream>
using namespace std;
int main()
{
    int a[][3] = {0,1,2,3,4};

    return 0;
}
```

对于二维数组(数组名仅带有一个下标的情况),在下方的窗口内会显示其首地址的所有值,并且在随后的 {} 内按照前文一维数组所示的格式显示所含有的所有值。

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

名称	值	类型
a	0x004ff89c {0x004ff89c {0, 1, 2}, 0x004ff8a8 {3, 4, 0}}	int[2][3]

本页涉及知识点 3.3,3.4,3.5

3.6 一维数组作为指针值实参

```
#include <iostream>
using namespace std;
void fun(int* a)
{
    int temp = 1;
    cout << temp << endl;
}
int main()
{
    int a[] = {0,1,2,3,4};
    fun(a);
    return 0;
}
```

一维数组作为指针值实参和将数组首元素地址赋值给形参效果一致，在下方的窗口内显示数组首元素的地址值和首元素的值

```
1 #include <iostream>
2 using namespace std;
3 void fun(int* a)
4 {
5     int temp = 1;
6     cout << temp << endl; 已用时间 <= 1ms
7 }
8 int main()
9 {
10     int a[] = {0,1,2,3,4};
11     fun(a);
12     return 0;
13 }
```

名称	值	类型
a	0x00c2fc80 (0)	int *
temp	1	int

3.7 指向字符串常量的指针变量

```
#include <iostream>
using namespace std;
int main()
{
    char* str;
    str = (char*)"I love programming.";
    return 0;
}
```

指向字符串常量的指针变量，在下方的窗口内会显示其字符串常量的首地址，在后面会跟上整个字符串常量的具体的值。

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     char* str;
6     str = (char*)"I love programming.";
7
8     return 0; 已用时间 <= 1ms
9 }
10
```

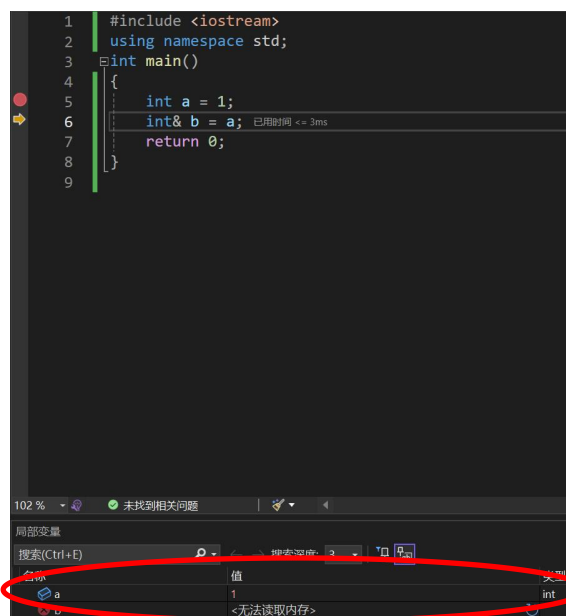
名称	值	类型
str	0x00069bd8 "I love programming."	char *

本页涉及知识点 3.6,3.7

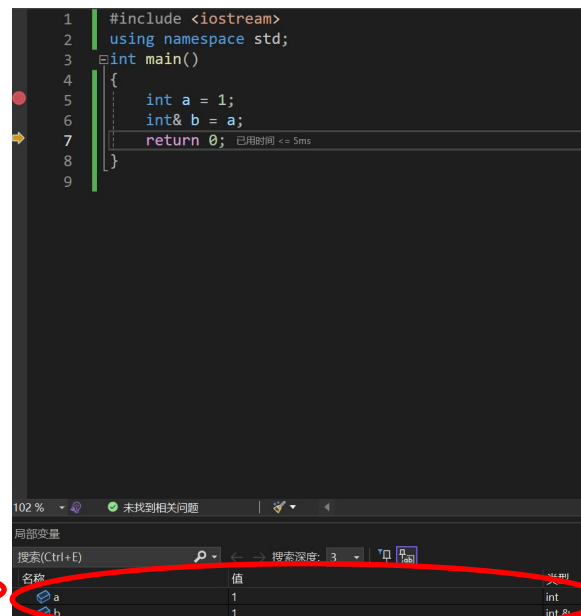
3.8 引用

引用和指针有所区别：类型不同，引用的类型是(int)&型(()内依据所引用的数据类型变化)；引用的值是所用的赋值变量的值，而指针的值是地址。引用在赋值之前会无法读取内存。

```
#include <iostream>
using namespace std;
int main()
{
    int a = 1;
    int& b = a;
    return 0;
}
```



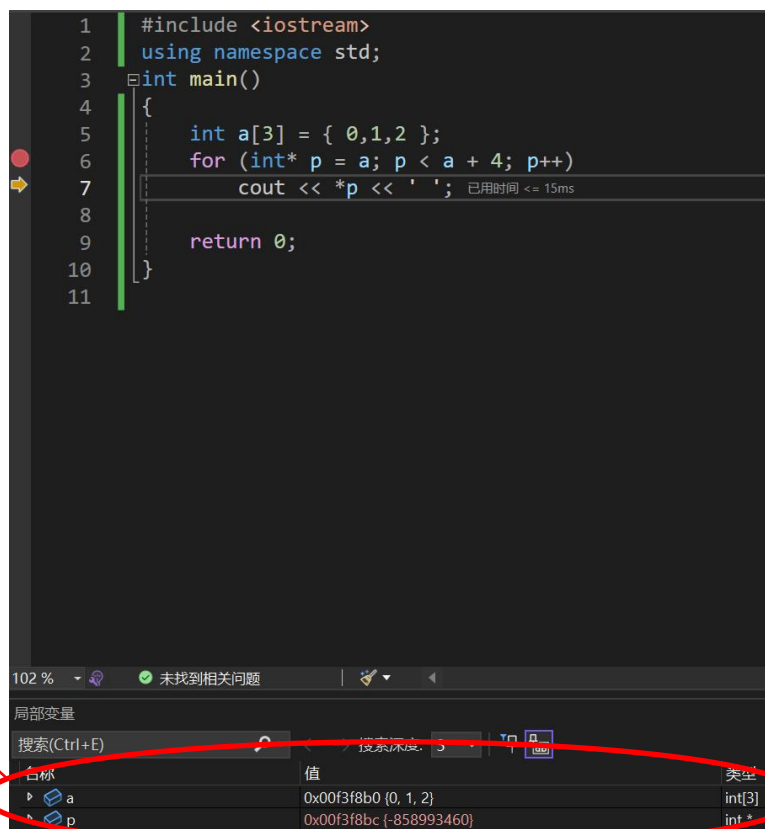
引用在赋值前无法的读取内存



引用在赋值后直接赋值变量的值

3.9 指针越界访问

```
#include <iostream>
using namespace std;
int main()
{
    int a[3] = { 0,1,2 };
    for (int* p = a; p < a + 4; p++)
        cout << *p << ' ';
    return 0;
}
```



指针发生越界访问，其在下方窗口显示的数值不可信

本页涉及知识点 3.8,3.9

```
#include <iostream>
using namespace std;
int main()
{
    char str[] = "I love programming";
    for (char* p = str; p < str + 30; p++)
        cout << *p;
    return 0;
}
```

当字符数组的指针访问发生越界，下面的窗口内会显示<字符串中的字符无效>，并且输出的值不可信

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      char str[] = "I love programming";
6      for (char* p = str; p < str + 30; p++)
7          cout << *p; 已用时间 <= 12ms
8      return 0;
9  }
```

局部变量

名称	值	类型
p	0x00affb4c <字符串中的字符无效.>	char *
str	0x00affb38 "I love programming"	char [19]

本页涉及知识点 3.9