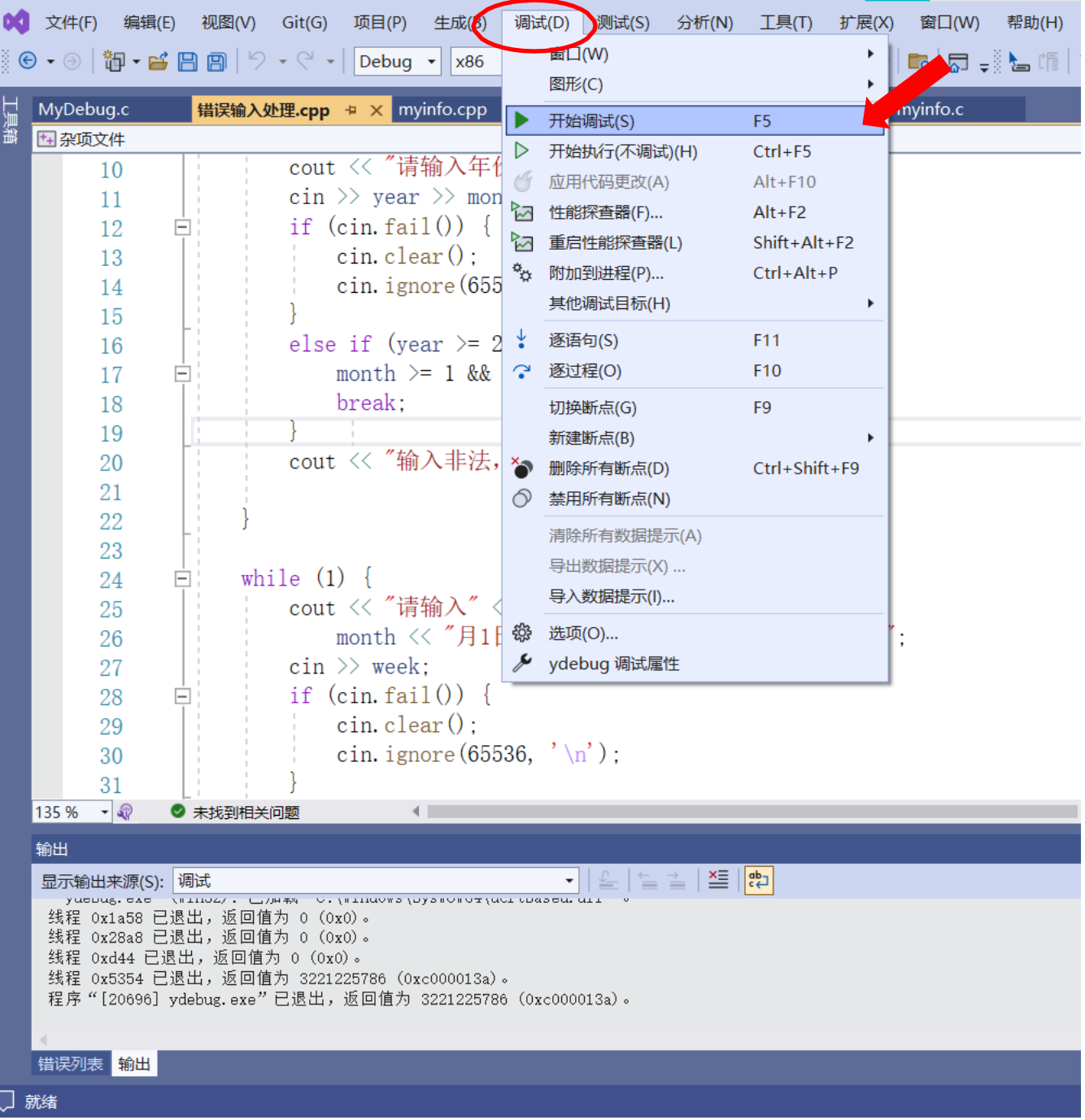
VS2022调试工具的使用

班级：计科

学号：2152988

姓名：杨恺铭

完成日期：2022.11.30



本图涉及知识点
1.1

开始调试：

1.快捷键F5

2.点击“调试”—“开始调试”

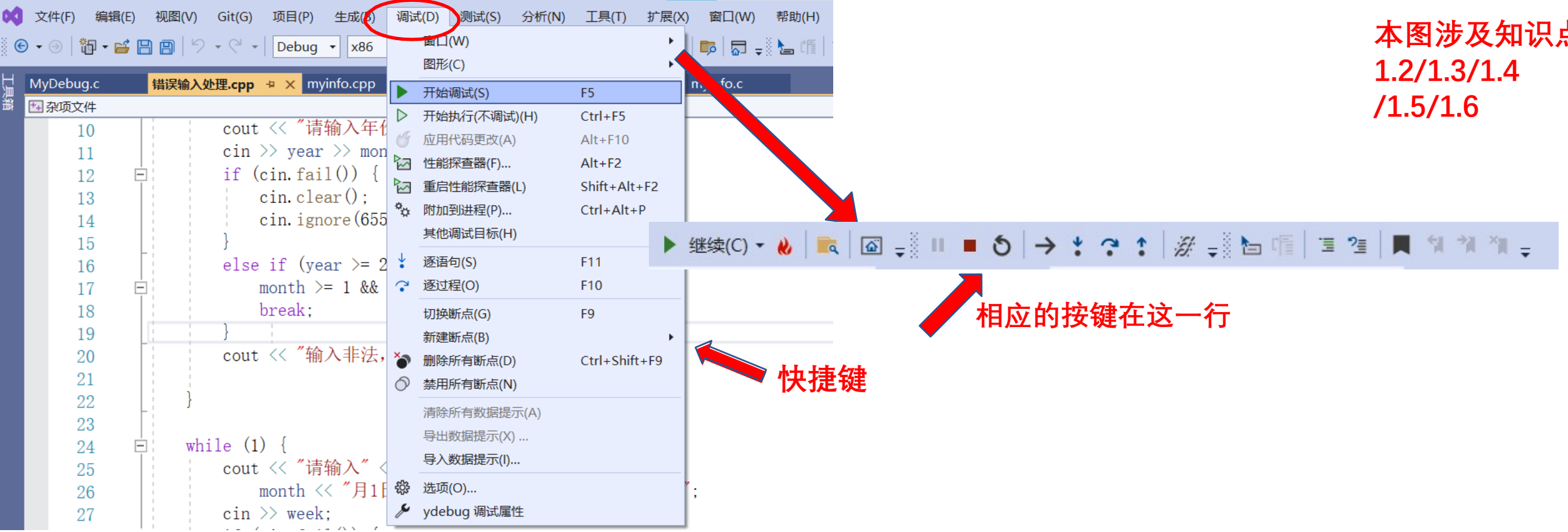
3.点击“本地windows调试 器”

结束调试：

1.快捷键：shift+F5

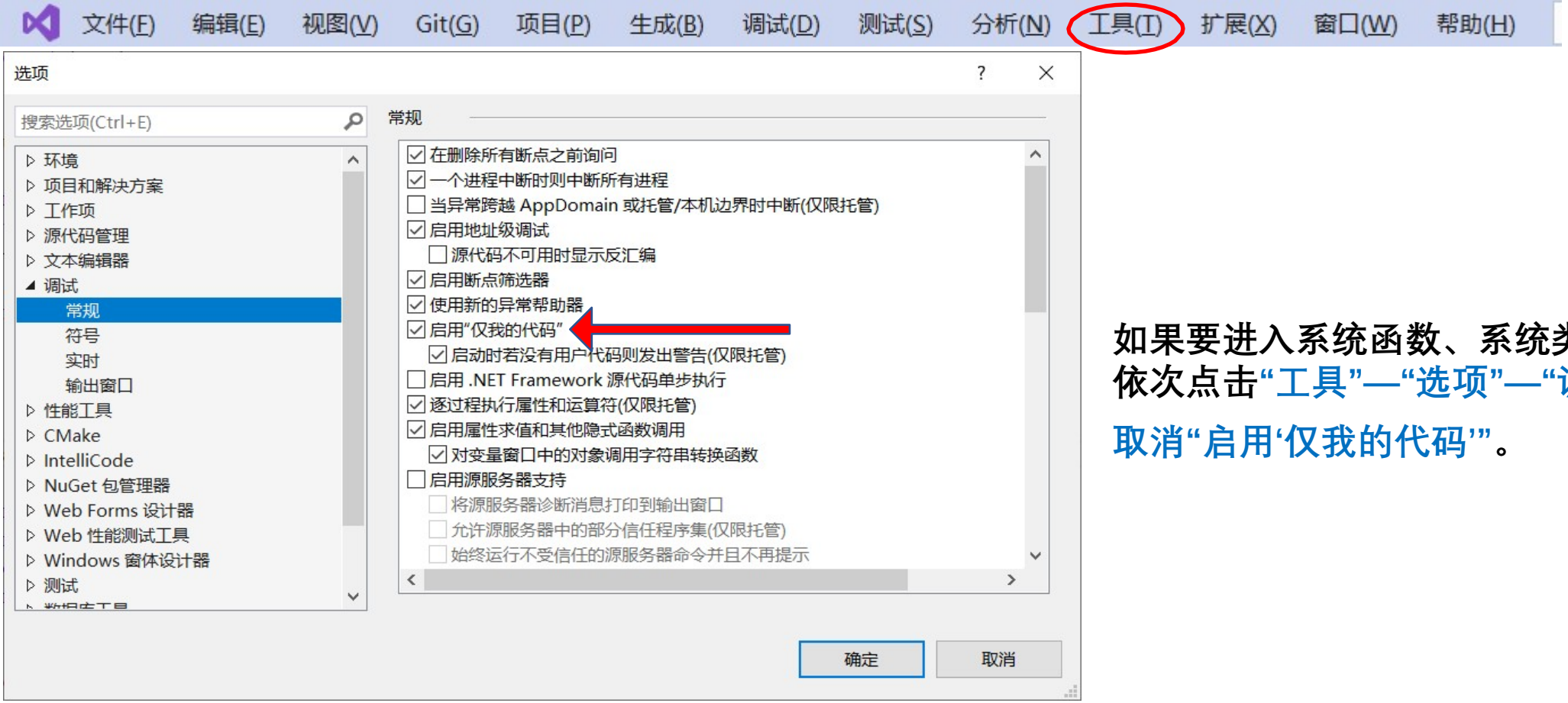
2.点击“调试”—“结束调试”

3.直接点击结束调试的按键



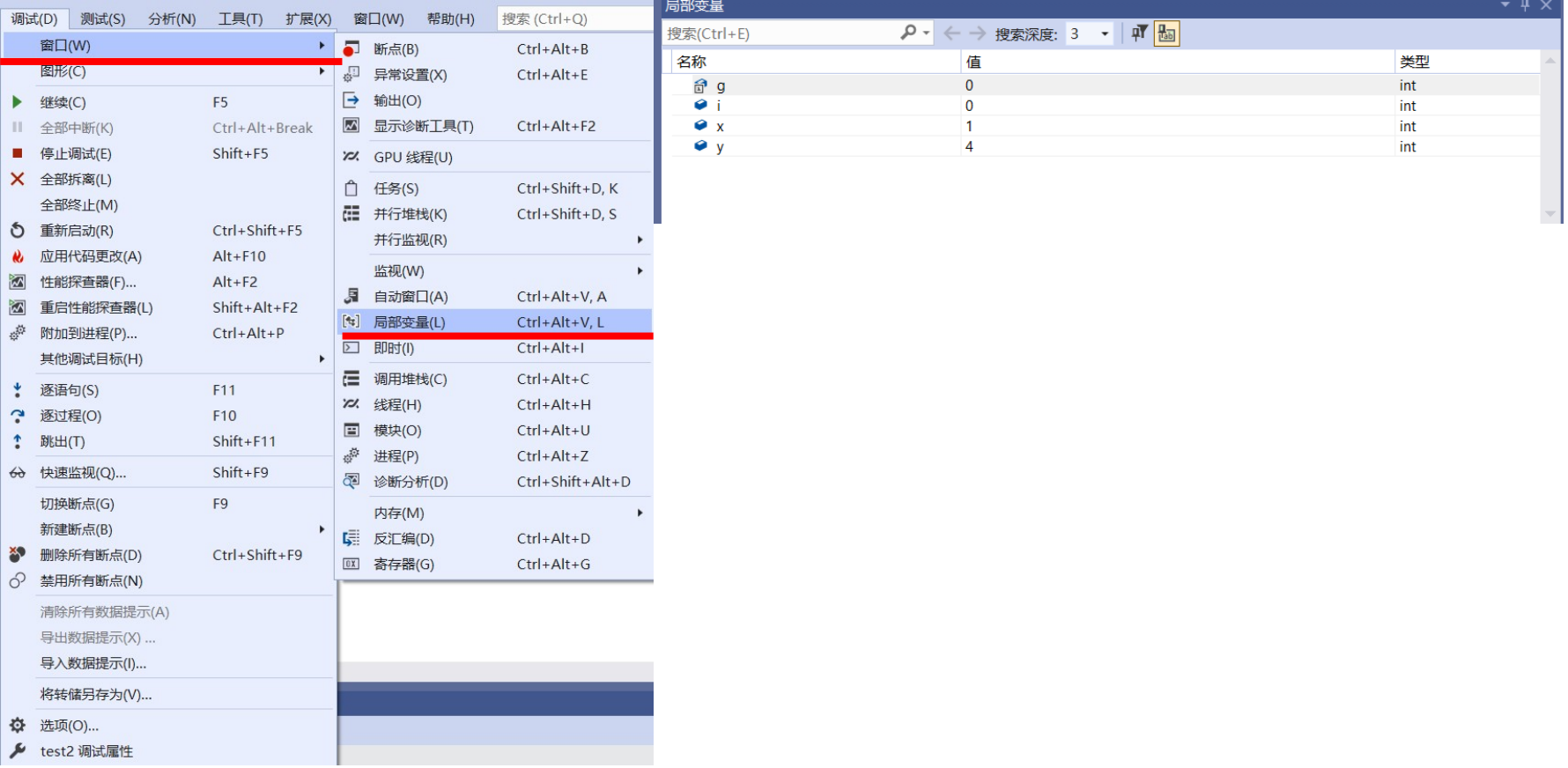
每个语句单步执行：选择“调试”—“逐语句”，或者开始执行后在点击逐语句的按键（此时需要有断点，否则调试会一步执行完）

在碰到系统函数、系统类、自定义函数时：
①选择“逐过程”，会一步完成执行而不会进入到其内部。
②选择“逐语句”会进入到其内部执行。（进入系统函数或者系统类中要更改编译器选项，详见下页）
③如果进入了系统函数、系统类、自定义函数中，选择“跳出”会跳出该过程，回到上一过程中。（例如本图中在fun中执行选择跳出会回到main）
（以上调试选项均可直接点击按键或使用快捷键）



如果要进入系统函数、系统类中逐语句执行需要依次点击“工具”—“选项”—“调试”—“常规”。

取消“启用‘仅我的代码’”。

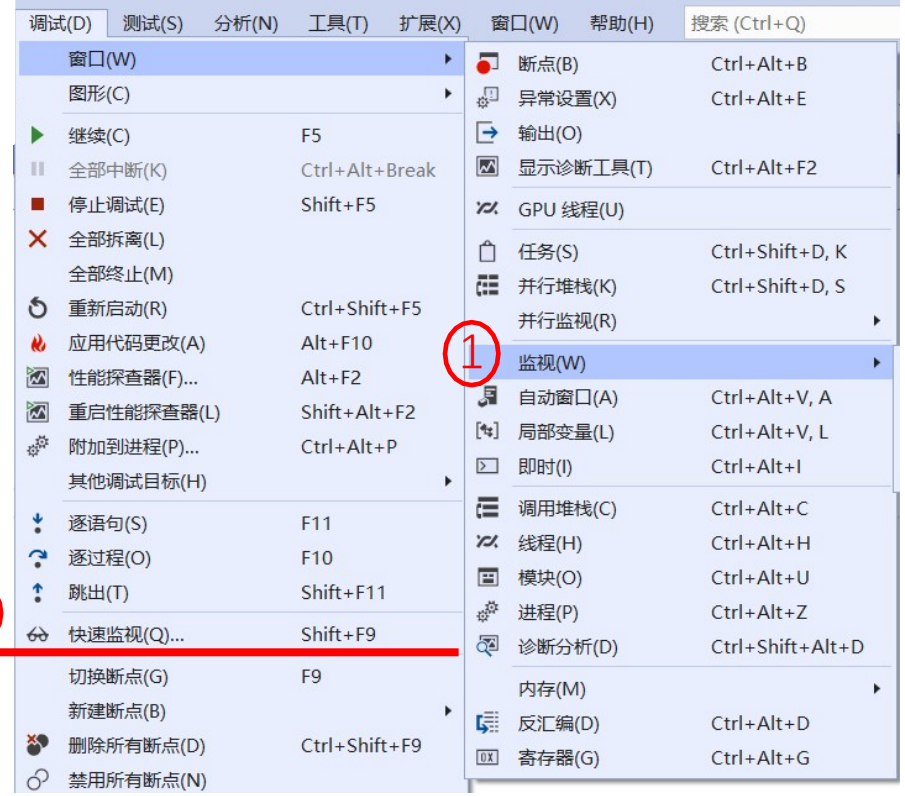


本图涉及知识点2.1

选择“调试”—“窗口”—“局部变量”。
可以从底部看到形参，自动变量的值，类型，以及变化情况。

本图涉及知识点2.2/2.3/2.4

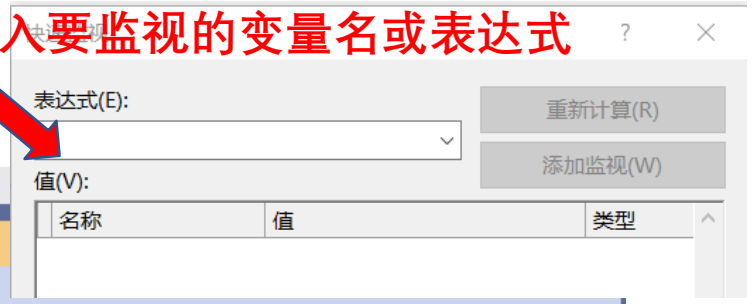
```
3   using namespace std;
4   static int h = 100;
5   int e = 1;
6   extern int f;
7   int fun(int x, int y);
8
9   int main()
10  {
11      int a=1, b, c;
12      double d;
13      b = 4;
14      c = fun(a, b);
15      d = sqrt(b);
16      return 0;
17  }
18
```



选择“调试”—“窗口”—“监视”，或者 “调试”—“快速监视”， 在“添加要监视的项”中输入 变量名即可对相应的变量(包括 静态局部，静态全局，外部全局) 进行监视，查看其变化情况。

或者直接移动光标到变量名上，右键选择添加监视，即可监视查看

在此输入要监视的变量名或表达式



100 % 未找到相关问题

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
e	1	int
f	2	int
h	1	int
添加要监视的项		

在此输入要监视的变量名或表达式

本图涉及知识点
3.1/3.2/3.3/3.4

test2 (全局范围)

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int a = 1, * b = &a;
7       int array1_int[10] = { 0 }, * array1 = array1_int;
8       char c = 'C';
9       float d = 1.234;
10
11      return 0;  已用时间 <= 1ms
12  }
13
```

100 % 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
a	1	int
array1	0x005dfc58 {0}	int *
	0	int
array1_int	0x005dfc58 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}	int[10]
b	0x005dfc94 {1}	int *
c	67 'C'	char
d	1.23399997	float

array1_int	0x005dfc58 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}	int[10]
[0]	0	int
[1]	0	int
[2]	0	int
[3]	0	int
[4]	0	int
[5]	0	int

通过“局部变量”窗口可以对简单变量，一维数组以及指向它们的指针进行查看。

点击一维数组左边的按键即可查看数组中各个元素的值（如上图）。

指向简单变量/一维数组的指针变量的值会显示它指向的地址，后面的{}内为指向的变量的值。

本图涉及知识点
3.5/3.7

```cpp
#include <iostream>
using namespace std;


void fun(int *array,int &a)
{
    cout << *array;
}

int main()
{
    const char* c = "hello world";
    int a = 1;
    int array1[][2] = { 1,2,3,4 };
    int array2[2][3] = { 1,2,3,4,5,6 };
    int array3[3] = { 1,2,3 };
    fun(array3, a); 已用时间 <= 1ms
    return 0;
}
```

100 % 未找到相关问题

局部变量

搜索(Ctrl+E) 搜索深度: 3

名称	值	
a	1	int
array1	0x00aff914 {0x00aff914 {1, 2}, 0x00aff91c {3, 4}}	int[2][2]
array2	0x00aff8f4 {0x00aff8f4 {1, 2, 3}, 0x00aff900 {4, 5, 6}}	int[2][3]
array3	0x00aff8e0 {1, 2, 3}	int[3]
c	0x005d7b30 "hello world"	const char *

array1	0x00aff914 {0x00aff914 {1, 2}, 0x00aff91c {3, 4}}	int[2][2]
[0]	0x00aff914 {1, 2}	int[2]
[0]	1	int
[1]	2	int
[1]	0x00aff91c {3, 4}	int[2]

地址

二维数组：与一维数组类似（如上图中可以在array1中直接查看二维数组中的每个变量）。其中由于二维数组是元素为一 维数组的一维数组，所以其中可以显示一些地址。

指向字符串常量的指针变量，可以看到字符串常量的地址（如上图“hello world”的地址即为0x005d7b30）非常靠前

本图涉及知识点
3.6/3.8

```cpp
#include <iostream>
using namespace std;


void fun(int *array,int &a)
{
    cout << *array;
}

int main()
{
    const char* c = "hello world";
    int a = 1;
    int array1[][2] = { 1,2,3,4 };
    int array2[2][3] = { 1,2,3,4,5,6 };
    int array3[3] = { 1,2,3 };
    fun(array3, a); 已用时间 <= 1ms
    return 0;
}

```

当调试进入fun函数时

局部变量

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
a	1	int &
array	0x00aff8e0 {1}	int *
	1	int

图1

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
a	1	int &
array	0x00aff8e0 {1}	int *
array+1	0x00aff8e4 {2}	int *
添加要监视的项		

图2

引用和指针的区别：

在函数中引用只能显示变量的值 而指针可以看到变量的地址（指针本身的值）和变量的值

实参是一维数组名，形参是指针：

如图一，只能查看到数组中指针指向的元素的值。

如果想要查看数组中其它元素的值可以在监视中手动添加，例如array+1等形式

本图涉及知识点
3.9

```
5   int main()
6   {
7       char str[5] = "ab";
8       char* p = str;
9       for (int i = 0; i < 10; i++) {
10          ++p;
11      }已用时间 <= 5ms
12      return 0;
13  }
14
```

保存的更改

119 % 未找到相关问题

局部变量

搜索(Ctrl+E) 搜索深度: 3

名称	值
i	5
p	0x0058f93a <字符串中的字符无效。>
	-52 '?'
str	0x0058f934 "ab"

指针越界访问：通过对指针变量的值的监视，如果访问越界，则会出现字符串中的字符无效的提示