程序编译能够通过，但程序获得的结果不符合预期时，就需要一步步跟踪程序，看程序的变量是如何变化的。
一个办法是在可能发生编错的地方之前写个 cout ，但显然比较费劲；
另一个办法就是一般说的调试（debug），在程序中设置断点（breakpoint），程序运行到断点时暂停，且可以查看各个变量的具体值。

1. 介绍

程序运行过程为：
预处理 -> 编译和优化 -> 生成目标文件 -> 链接 -> 可执行文件

- 预处理：不对源程序进行解析，仅做些预处理，如宏的替换、删除注释、处理预处理指令（如#include、#ifdef 等）。
- 编译和优化：解析源代码，查看语法语义是否有错误，有则停止程序同时报错，没有则会生成译文。
- 生成目标文件：生成二进制代码。
- 链接：如某个文件用了其它文件的变量或函数，则需要进行链接。
- 可执行文件。

Code Blocks 中以下几个图标的含义（从左到右）：

- Build：执行编译和链接操作，不会运行程序。
- Run：运行最近一次 Build 的可执行文件。
- Build and run：连续执行以上两步。
- Debug / Continue：开始调试/继续调试，执行到断点处（需要先执行 Build）。
- Run to cursor：运行到光标处（需要先执行 Build）。
- Step into：执行到某个断点处，此时不希望直接执行到下一个断点，而是希望按照代码一步步执行，点击该图标。

设置断点：在该行左端标有该行序号的位置点击鼠标右键，选择“添加断点”。
删除断点：删除一个断点可以将鼠标移动到断点处，点击右键，选择“移除断点”。删除所有断点，可以点击"调试（Debug）"——"移除所有断点（Remove all breakpoints）"。

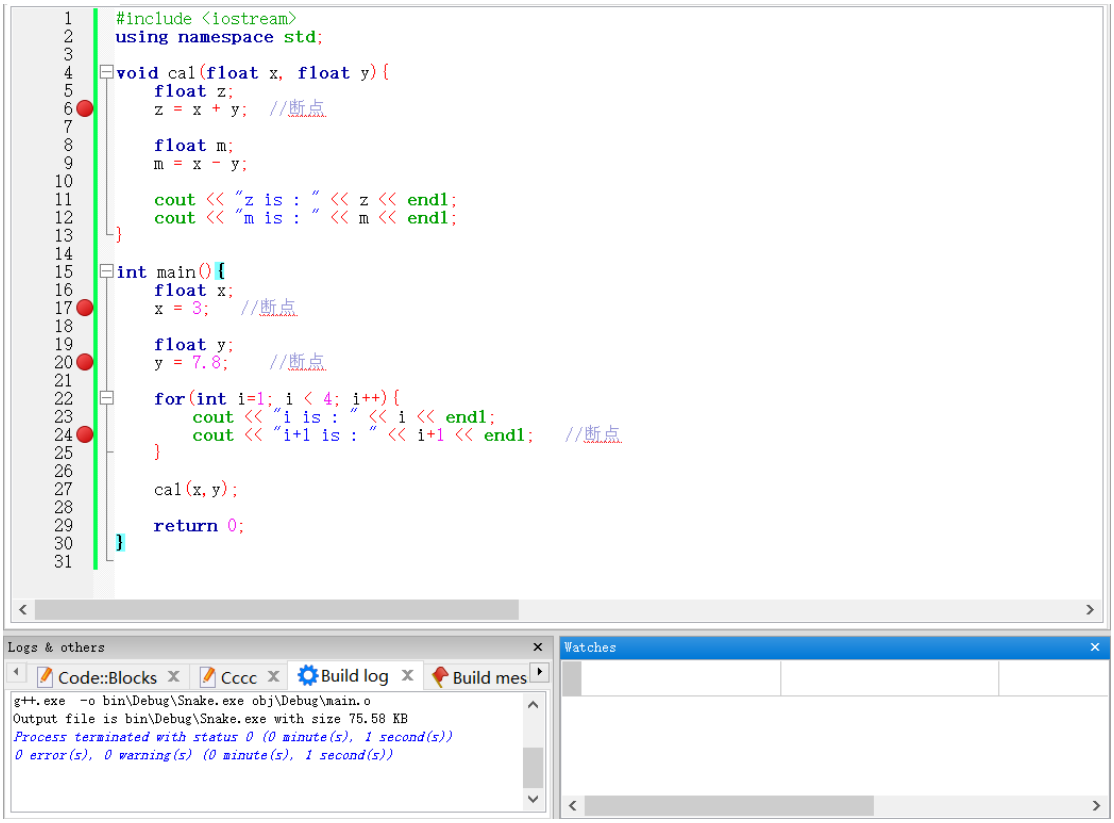
2. 使用

下面的程序在几个地方设置了断点（用注释标记），在编程软件里会显示为红色的圆点。

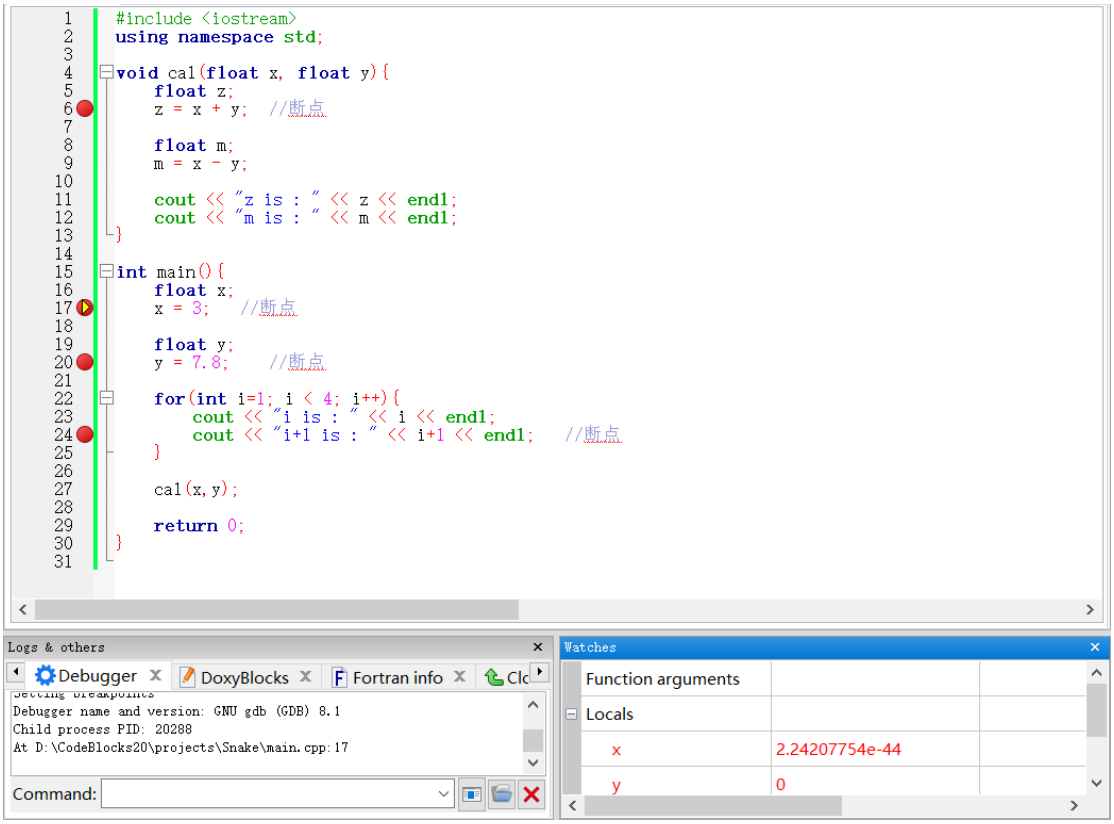
```
#include <iostream>
using namespace std;

void cal(float x, float y){
    float z;
    z = x + y;   //断点

    float m;
    m = x - y;
```

```
    cout << "z is : " << z << endl;
    cout << "m is : " << m << endl;
}

int main(){
    float x;
    x = 3;     //断点

    float y;
    y = 7.8;       //断点

    for(int i=1; i < 4; i++){
        cout << "i is : " << i << endl;
        cout << "i+1 is : " << i+1 << endl;     //断点
    }

    cal(x,y);

    return 0;
}
```

在调试前先进行编译或 Build，看看有没有语法语义错误。



然后再开始调试，执行到第一个断点处，



如果**监视窗口**（即上图中右下方的 Watches 窗口）没有显示，
可以通过“调试（Debug）”——“调试窗口（Debugging windows）”——勾选“监视窗口（Watches）”把它调出来。如果它出来之后不在右下方，可以拖拽到右下方。

监视窗口可以显示调试过程中各个变量的值的变化。
它显示的是执行到当前断点之前时的状态，即还未执行该断点所在行的语句。
继续点击 Debug / Continue 按键就执行到下一个断点处。

3. 退出

在调试模式下，点击这行图标中最右边的红色叉按键即可退出调试模式。