

# CH06 Debug

除錯 (Welcome to Hell)  
(這不是Bug，這是功能)

主講人：陳靖德  
2014/12/03

# 什麼是Bug ?

9/9

0800 Antam started

1000 " stopped - antam ✓

1300 (032) MP - MC

(033) PRO 2

convd

Relays 6-2 in 033 failed special speed test in relay

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Multy Adder Test.

1545

Relay #70 Panel F (moth) in relay.

First actual case of bug being found.

1630 Antam started.

1700 closed down.

1.2700 9.037847025  
9.037846995 convd  
1.482147000  
2.130476415  
2.130476415  
2.130676415

4.615925059(-2)

Relay 337

# The First "Computer Bug"

# Bug 種類

- 語法錯誤
- 執行階段錯誤
- 邏輯錯誤

# 語法錯誤

- 編譯器會報錯的

```
1  #include <stdio.h>
2
3  int main() {
4      printf("Hello, world!!\n")
5      return 0;
6  }
7
```

錯誤: 必須是 ;

- 這是最好處理的 !!!

# [題外話]警告(warning)

```
1  #include <stdio.h>
2
3  int main() {
4      int x;
5      if (x == 0) {
6          printf("%d", 20);
7      }
8      return 0;
9  }
10
```

## 輸出

顯示輸出來源(S): 建置

1>----- 已開始建置: 專案: Debug, 組態: Debug Win32 -----

1> Source.cpp

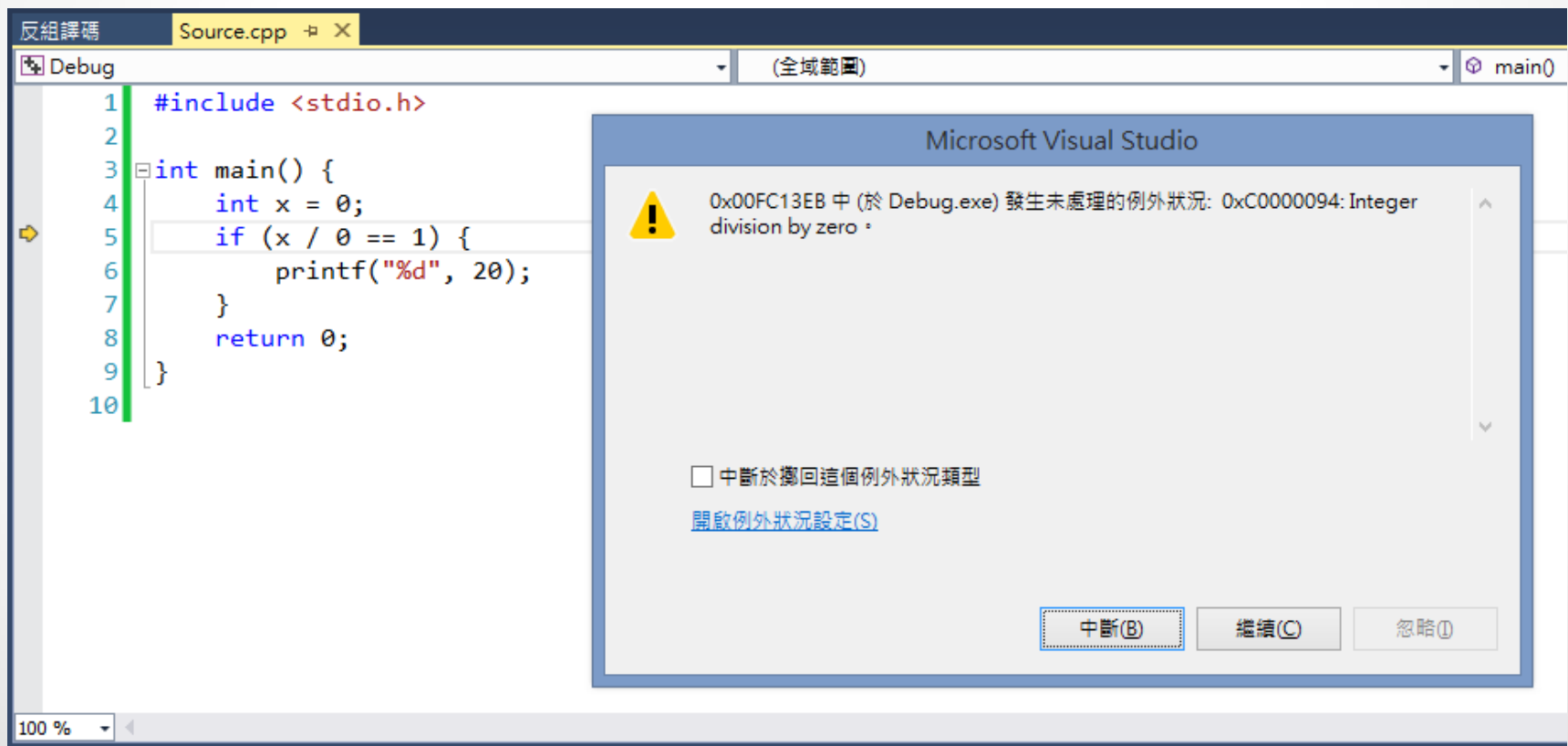
1>c:\users\...\desktop\debug\debug\source.cpp(5): warning C4700: 使用了未初始化的區域變數 'x'

1> Debug.vcxproj -> C:\Users\danny50610\Desktop\Debug\Debug\Debug.exe

===== 建置: 1 成功、0 失敗、0 最新、0 略過 =====

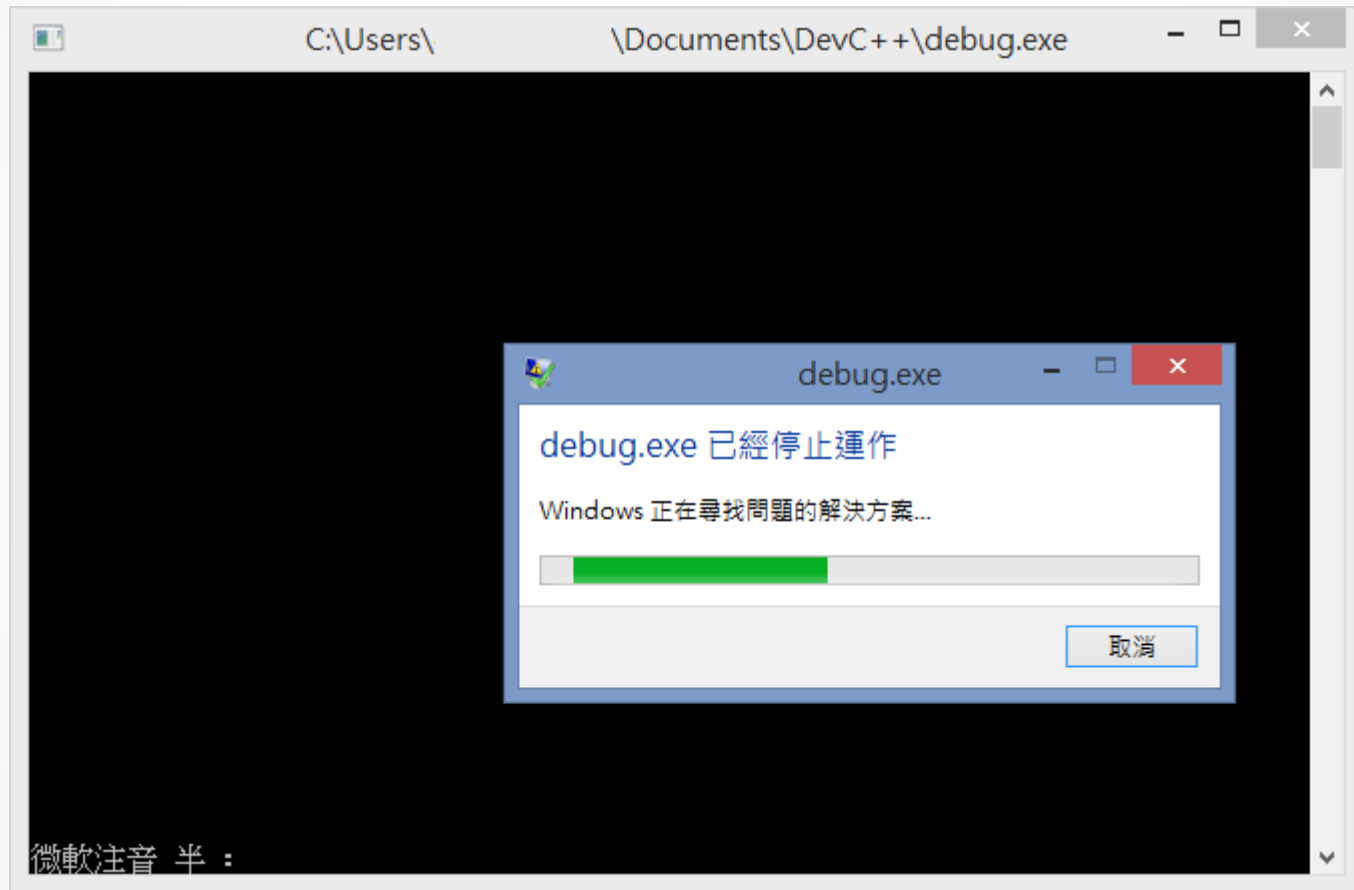
# 執行階段錯誤

- 程式執行時，發生錯誤(像是整數除以0、存取非法記憶體位置.....)



我知道範例很蠢...

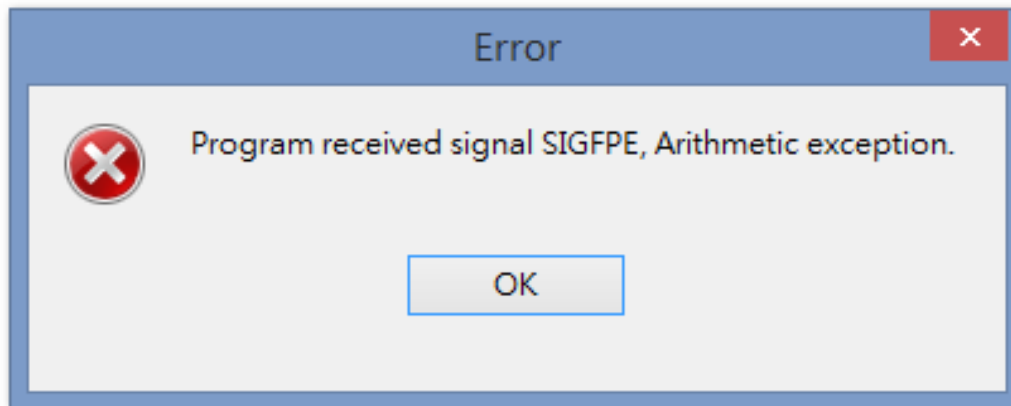
# 執行階段錯誤(續)



# 執行階段錯誤(續)

debug.cpp

```
1  #include <stdio.h>
2
3  int main() {
4      int x = 0;
5      if (x / 0 == 1) printf("hello");
6      return 0;
7  }
```



\*請用Debug模式



# 邏輯錯誤

- 程式沒當掉，但行為不是想要的。**最難處理**

## 印出0到10的整數，錯在哪裡？

```
1 #include <stdio.h>
2
3 int main() {
4     int i;
5     for (i = 0; i = 10; i++) {
6         printf("%d ", i);
7     }
8     return 0;
9 }
10
```

A screenshot of a Windows command prompt window titled "C:\Users\danny50610\Desktop\Debug\Debug.exe". The window contains a large grid of the number "10" repeated many times, arranged in approximately 20 rows and 40 columns. At the bottom left of the window, the text "微軟注音 半：" is visible. The window has standard Windows controls (minimize, maximize, close) at the top right.

# 如何除錯

沒有固定的方法，這是需要學習的

重點是...

多動腦筋，多做思考  
拿起紙筆

# 常用原則

- 了解程式碼
- 尋找可以再次發生錯誤的資料或操作
- 定位錯誤發生的地方
- 小心地修正，避免產生其他Bug
- 驗證假設  
(例如：我覺得這個if會進去，結果事實上一次都沒有)

# 可再現性

觀察Bug

重現

修復

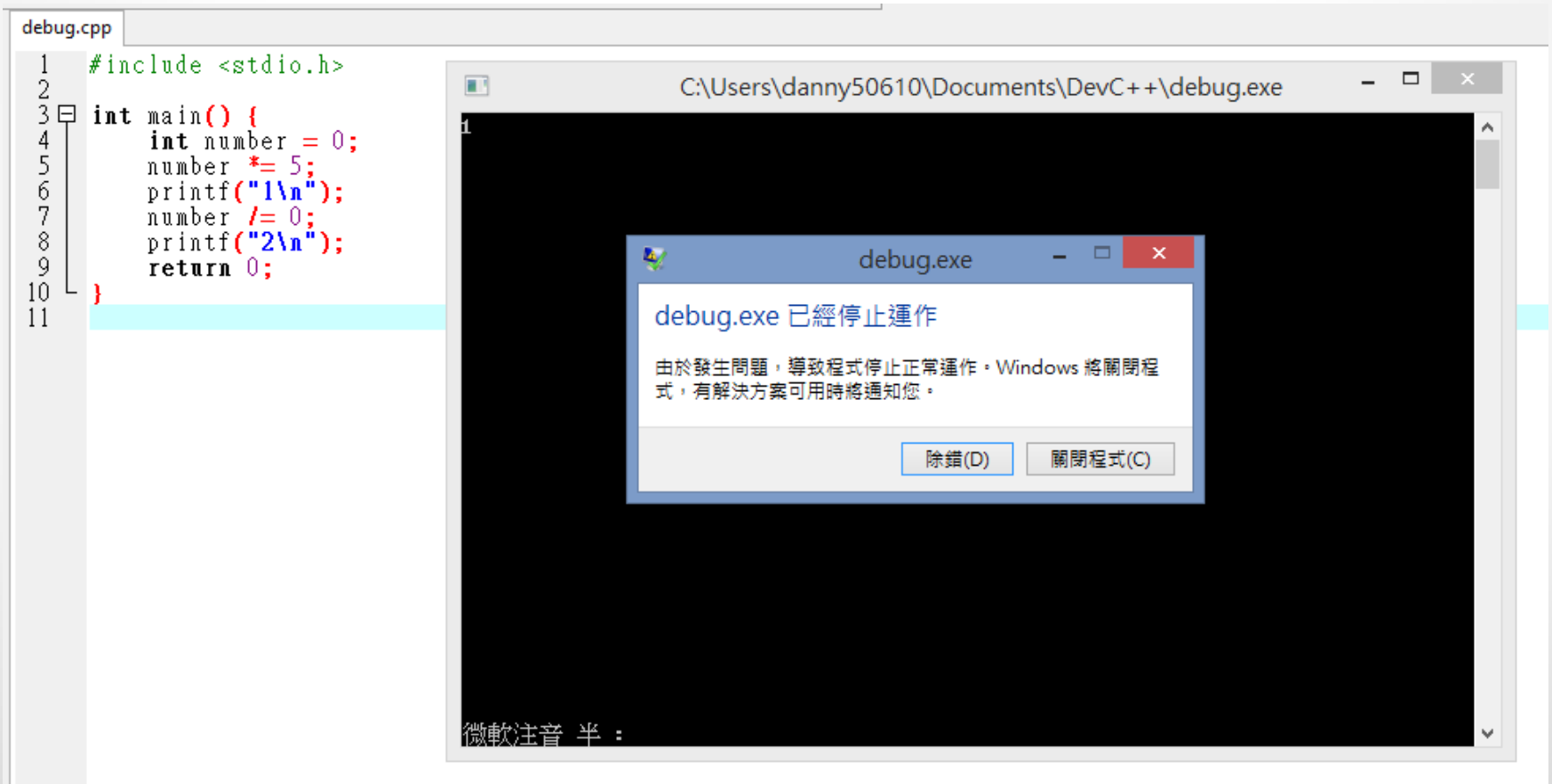
驗證

紀錄



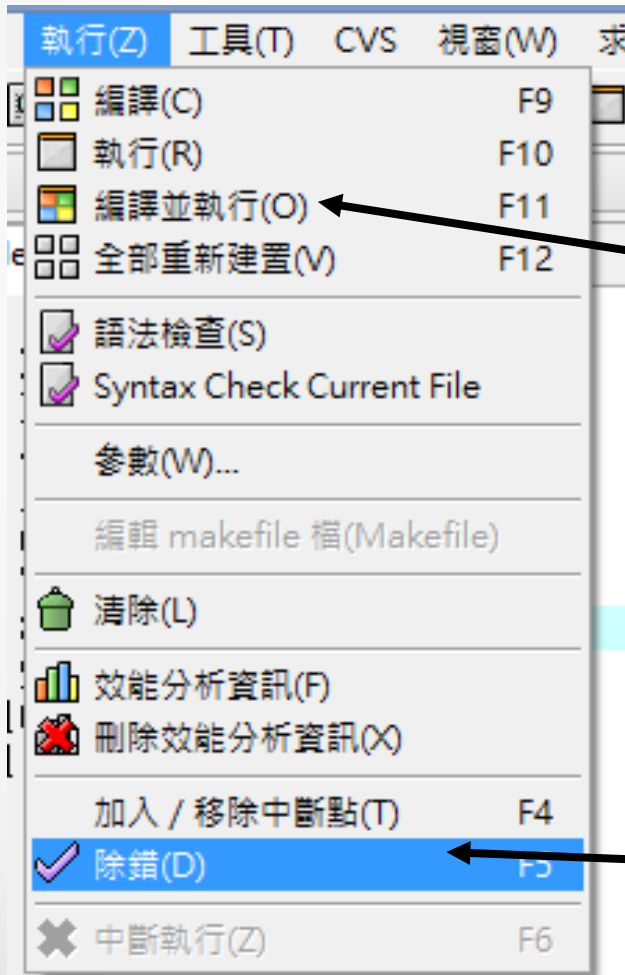
# 常用技巧(一)

- 在適當的地方printf一些東西，來定位錯誤發生的地方



# Debug Tool

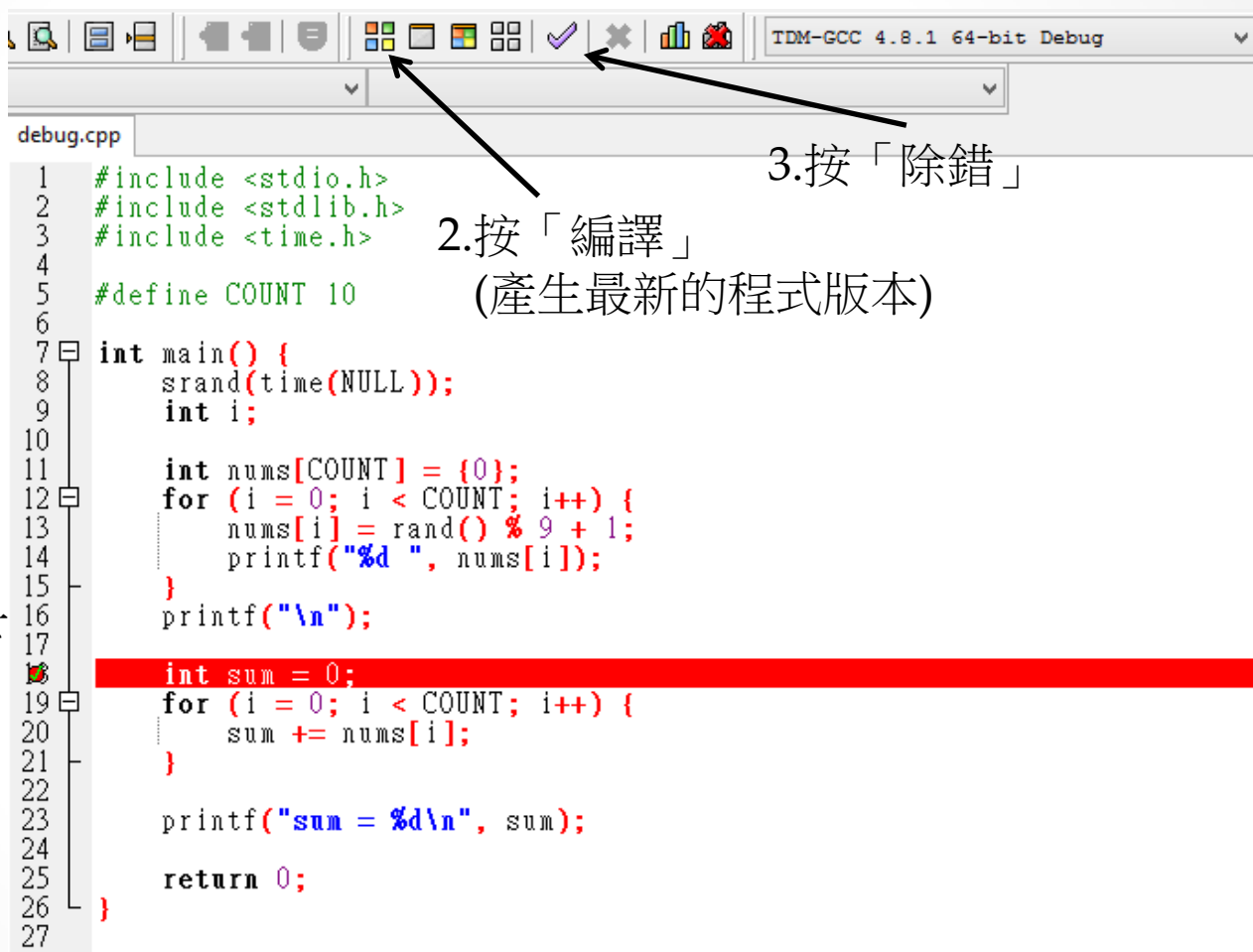
- 這裡先介紹Dev C++的基本除錯



這是平常用的，程式當掉就當掉了

接下來介紹的功能  
程式要在「除錯」模式執行  
除錯前再點

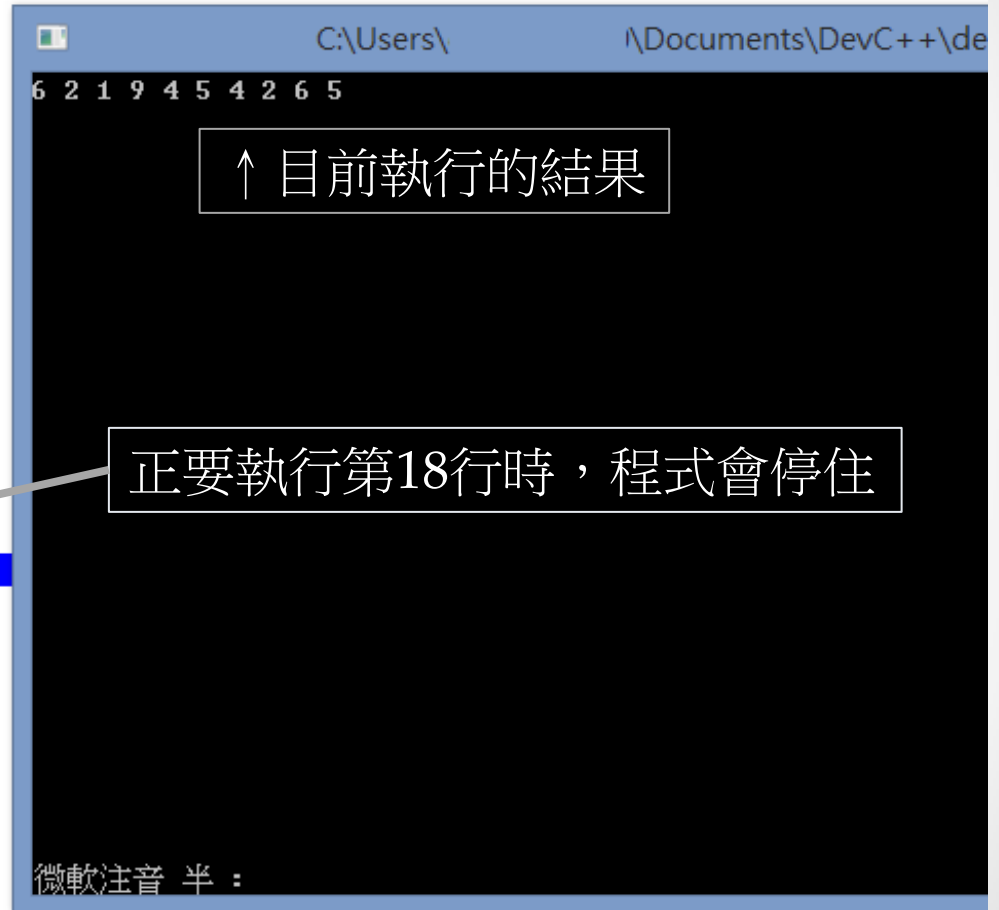
# 中斷點



# 中斷點(續)

debug.cpp

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define COUNT 10
6
7  int main() {
8      srand(time(NULL));
9      int i;
10
11      int nums[COUNT] = {0};
12      for (i = 0; i < COUNT; i++) {
13          nums[i] = rand() % 9 + 1;
14          printf("%d ", nums[i]);
15      }
16      printf("\n");
17
18      int sum = 0;
19      for (i = 0; i < COUNT; i++) {
20          sum += nums[i];
21      }
22
23      printf("sum = %d\n", sum);
24
25      return 0;
26  }
```





# 監看式

The screenshot shows a C++ IDE with a file named `debug.cpp`. The code is as follows:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 #define COUNT 10
6
7 int main() {
8     srand(time(NULL));
9     int i;
10
11     int nums[COUNT] = {0};
12     for (i = 0; i < COUNT; i++) {
13         nums[i] = rand() % 9 + 1;
14         printf("%d ", nums[i]);
15     }
16     printf("\n");
17
18     int sum = 0;
19     for (i = 0; i < COUNT; i++) {
20         sum += nums[i];
21     }
22     printf("sum = %d\n", sum);
23
24     return 0;
25 }
```

A watch window titled "新增監看式" (Add Watch Expression) is open, showing the variable `nums` entered in the "輸入變數名稱:" (Enter variable name) field. The "OK" button is highlighted.

At the bottom of the IDE, the "新增監看式(A)" (Add Watch Expression) button is highlighted with a red box. The command window shows the command `display nums`.

Annotations with arrows indicate the workflow:

- An arrow points from the `nums` variable in the code to the watch window.
- An arrow points from the `nums` variable in the watch window to the "新增監看式(A)" button.

輸入要觀察的變數

# 逐行執行、繼續執行

The screenshot shows the Dev-C++ 5.7.1 IDE with a C++ program open. The program is a simple loop that generates random numbers and calculates their sum. The debugger is active, and the 'Step Through' mode is selected. The 'Step Over' button (逐行執行) and 'Continue' button (繼續執行) are highlighted with red boxes. Arrows point from the Chinese text labels '一行一行執行' and '程式繼續執行' to these buttons respectively.

專案 類別 除錯

nums = {6, 9, 8, 5, 9, 2, 4, 1, 8, 3}  
sum = 1

debug.cpp

```
10  
11  
12 int nums[COUNT] = {0};  
13 for (i = 0; i < COUNT; i++) {  
14     nums[i] = rand() % 9 + 1;  
15     printf("%d ", nums[i]);  
16 }  
17 printf("\n");  
18  
19 int sum = 0;  
20 for (i = 0; i < COUNT; i++) {  
21     sum += nums[i];  
22 }  
23 printf("sum = %d\n", sum);  
24  
25 return 0;  
26  
27 }
```

逐行執行(N) 繼續執行(S)

對 GDB 送出命令: display sum

計算:

一行一行執行

程式繼續執行

行數: 18 列數: 1 反白字數: 0 總行數: 29 檔案長度: 428 插入模式 完成解析 (花了 0.016 秒)

# 進入函式、跳出函式

專案 類別 除錯

nums = {6, 9, 8, 5, 9, 2, 4, 1, 8, 3}  
sum = 1

按「進入函式」會進去函式停住  
按「逐行執行」直接把函式執行完，在下一行停住  
在函式裡按「跳出函數」，會等到跳出函式後停住

```
7  
8 int main() {  
9     //srand(time(NULL)); 拿掉的話，每次程式產生的亂數都一樣  
10    int i;  
11    int nums[COUNT];  
12  
13    for (i = 0; i < COUNT; i++) {  
14        nums[i] = rand() % 9 + 1;  
15    }  
16    printArray(nums, COUNT);  
17  
18    int sum = 0;  
19    for (i = 0; i < COUNT; i++) {  
20        sum += nums[i];  
21    }  
22  
23    printf("sum = %d\n", sum);  
24  
25    return 0;  
26 }  
27  
28 int printArray(int nums[], int n) {  
29    int i;  
30    for (i = 0; i < n; i++) {  
31        printf("%d ", nums[i]);  
32    }  
33    printf("\n");  
34 }  
35
```

編譯器訊息 資源檔 編譯紀錄 除錯 搜尋結果 最小化

除錯(D) 新增監看式(A) 逐行執行(N) 繼續執行(S) 逐指令執行  
中斷執行 檢視 CPU 視窗(V) 進入函數(I) 跳出函數 進入呼叫

計算:

對 GDB 送出命令:

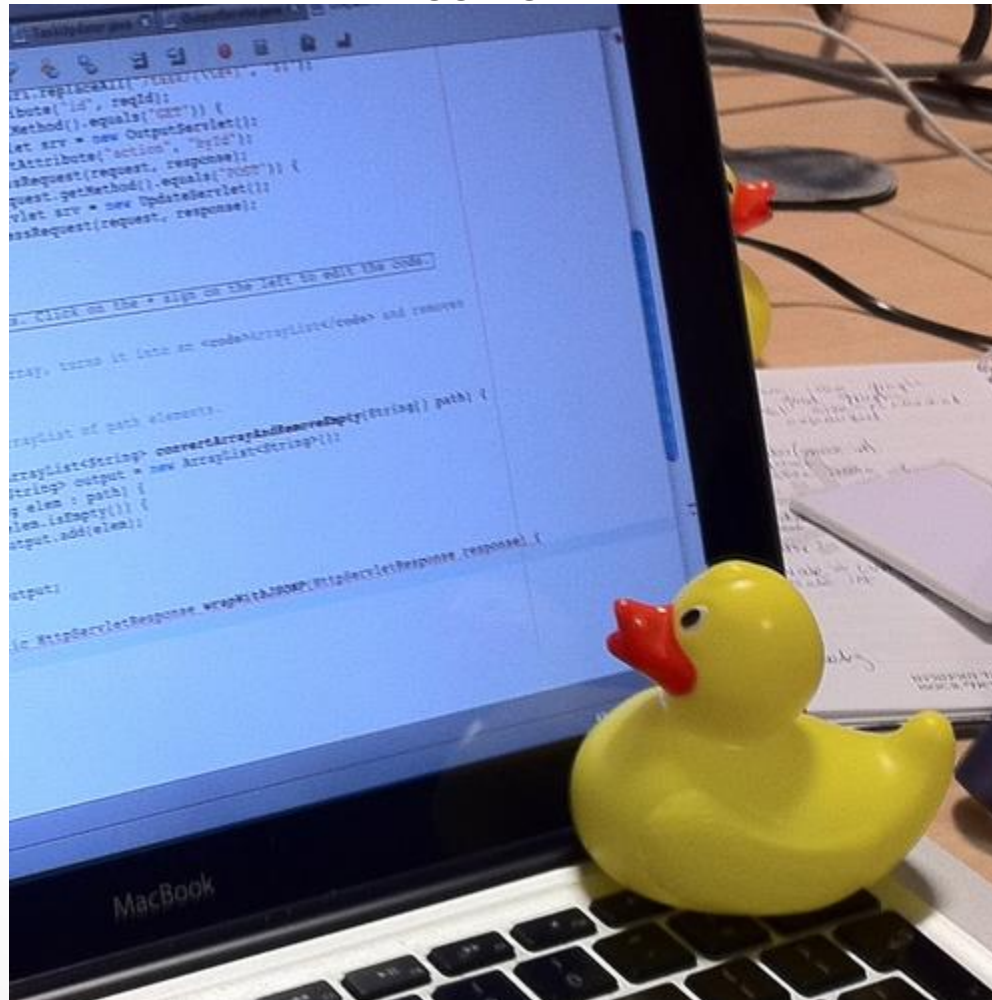
->->value-history-begin 2 -  
\$2 =  
->->value-history-value  
0  
> value-history-end

# 預防勝於治療

- 多考慮各種情況(不要走火入魔)
- 變數立即初始化(`int x = 0;`)
- 每寫完一小段就測試
- .....

# 找人幫助

Rubber duck debugging



# 找人幫助

- 大家都是很有愛心的，有時間就會指點(基本上)
- 給予足夠的資訊
  - 程式碼
  - 能夠產生錯誤的操作
  - 錯誤的結果、正確的結果(不然別人不知道你要甚麼)
  - 錯誤訊息(哪一行當掉、程式或編譯器丟出的訊息)
  - 環境資訊(函式庫版本、程式語言、編譯器...)

# 延伸主題

- bug tracker
  - Bug 回報範例：  
<https://github.com/Azanor/thaumcraft/issues/1041>
- 單元測試
- 版本控制系統(git...)
- assert

# 參考書目、資料

- 編成創藝：編寫出卓越的程式碼(Code Craft : The Practice of Writing Excellend code), Pete Goodliffe / 蔡學庸 譯, 2009
- [http://en.wikipedia.org/wiki/Rubber\\_duck\\_debugging](http://en.wikipedia.org/wiki/Rubber_duck_debugging)