

The Very First Course

Expressions and Variables

winston, arbuztw

Hello World

- 第一支程式

```
#include <iostream>

int main() {
    std::cout << "Hello World\n";
    return 0;
}
```

表達式

- 試著用 `std::cout` 輸出結果!

表達式

- 試著用 `std::cout` 輸出結果!

▷ $1 + 2 * 3$

7

表達式

- 試著用 `std::cout` 輸出結果!

▷ $1 + 2 * 3$

7

▷ $(5 + 4)/3$ (division)

3

表達式

- 試著用 `std::cout` 輸出結果!

▷ $1 + 2 * 3$

7

▷ $(5 + 4)/3$ (division)

3

▷ $7 * 7$

49

表達式 (續)

- 算數運算子:
 - 加, `+`
 - 減, `-`
 - 乘, `*`
 - 除, `/`
 - 模(取餘數), `%`
- 值得注意的是，到目前為止我們只討論整數的運算

有關整數除法

- 下面的算式的結果是...

▷ $22/7$

有關整數除法

- 下面的算式的結果是...

$$\begin{array}{r} \triangleright 22/7 \\ 3 \end{array}$$

- 高中數學! (not exactly though...)

$$a = bq + r, \quad a, b, q, r \in \mathbb{N}_0$$

有關整數除法

- 下面的算式的結果是...

$$\begin{array}{r} \triangleright 22/7 \\ 3 \end{array}$$

- 高中數學! (not exactly though...)

$$a = bq + r, \quad a, b, q, r \in \mathbb{N}_0$$

- Exercise: 給定正整數 a 和 b ，試著計算 $\lceil a/b \rceil$

$$\circ \lceil a/b \rceil = \begin{cases} q + 1 & \text{if } a = bq + r \text{ and } r \neq 0 \\ q & \text{if } a = bq \end{cases}$$

表達式 (續)

- 對於下面這些運算子，若條件成立其運算結果會是1，反之則是0.

表達式 (續)

- 對於下面這些運算子，若條件成立其運算結果會是1，反之則是0.
- 比較
 - `<`, `>`, `<=`, `>=`
 - `<=`: 小於等於 (written \leq)
 - `>=`: 大於等於 (written \geq)
 - `==`, `!=`: 判斷是否相等, `x == 5`

表達式 (續)

- 對於下面這些運算子，若條件成立其運算結果會是1，反之則是0.
- 比較
 - `<`, `>`, `<=`, `>=`
 - `<=`: 小於等於 (written \leq)
 - `>=`: 大於等於 (written \geq)
 - `==`, `!=`: 判斷是否相等, `x == 5`
- 邏輯運算子
 - and: `&&`, e.g. `0 ≤ x && x < 10`
 - or: `||`, e.g. `x < 2 || x > 5`
 - not: `!`, e.g. `!x`

表達式 (續)

▷ $13 > 5$

1

表達式 (續)

▷ $13 > 5$

1

▷ $13 < 10$

0

表達式 (續)

▷ $13 > 5$

1

▷ $13 < 10$

0

▷ $13 == 13$

1

表達式 (續)

▷ $13 > 5$

1

▷ $13 < 10$

0

▷ $13 == 13$

1

▷ $!(13 == 13)$

0

表達式 (續)

▷ $13 > 5$

1

▷ $13 < 10$

0

▷ $13 == 13$

1

▷ $!(13 == 13)$

0

▷ $0 < 13 \ \&\& \ (13 < 100 \ || \ 13 > 200)$

1

變數 explanation

- 使用可修改的記憶體區塊

```
int  $x$  = 5;
```

變數 explanation

- 使用可修改的記憶體區塊

```
int x = 5;
```

▷ x

5

變數 explanation

- 使用可修改的記憶體區塊

int $x = 5$;

▷ x

5

▷ $(x + 1)/2$

3

變數 explanation

- 使用可修改的記憶體區塊

```
int x = 5;
```

▷ x

5

▷ $(x + 1)/2$

3

```
x = 8;
```

變數 explanation

- 使用可修改的記憶體區塊

int $x = 5$;

▷ x

5

▷ $(x + 1)/2$

3

$x = 8$;

▷ x

8

變數 explanation

- 使用可修改的記憶體區塊

```
int x = 5;
```

▷ x

5

▷ $(x + 1)/2$

3

```
x = 8;
```

▷ x

8

```
x = x * 2;
```


變數 explanation

- 使用可修改的記憶體區塊

int $x = 5$;

▷ x

5

▷ $(x + 1)/2$

3

$x = 8$;

▷ x

8

$x = x * 2$;

▷ x

16

變數 (續) demo

- `int var = 23;`
-
-

變數 (續) demo

- `int var = 23;`
-
-

`var`

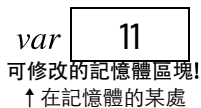
23

可修改的記憶體區塊!
↑ 在記憶體的某處

- `var` 指的同時是該記憶體區塊以及其內所存的值

變數 (續) demo

- `int var = 23;`
- `var = 11;`
-



- `var` 指的同時是該記憶體區塊以及其內所存的值

變數 (續) demo

- `int var = 23;`
- `var = 11;`
- `var = var - 4;`

`var`

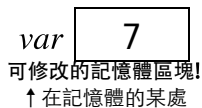
7

可修改的記憶體區塊!
↑ 在記憶體的某處

- `var` 指的同時是該記憶體區塊以及其內所存的值

變數 (續) demo

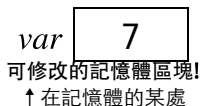
- `int var = 23;`
- `var = 11;`
- `var = var - 4;`



- `var` 指的同時是該記憶體區塊以及其內所存的值
- 以“`var = var - 4;`”這行來說:

變數 (續) demo

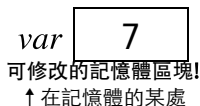
- `int var = 23;`
- `var = 11;`
- `var = var - 4;`



- `var` 指的同時是該記憶體區塊以及其內所存的值
- 以“`var = var - 4;`”這行來說:
 - 在 `var - 4` 中的“`var`”指的是其中所存值

變數 (續) demo

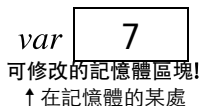
- `int var = 23;`
- `var = 11;`
- `var = var - 4;`



- `var` 指的同時是該記憶體區塊以及其內所存的值
- 以“`var = var - 4;`”這行來說:
 - 在 `var - 4` 中的 “`var` ” 指的是其中所存值
 - `var - 4` 算出來是 7

變數 (續) demo

- `int var = 23;`
- `var = 11;`
- `var = var - 4;`



- `var` 指的同時是該記憶體區塊以及其內所存的值
- 以“`var = var - 4;`”這行來說:
 - 在 `var - 4` 中的 “`var` ” 指的是其中所存值
 - `var - 4` 算出來是 7
 - 將 7 存入 `var`;
這時在`var = ...` 中 “`var` ” 指的是該記憶體區塊

變數 (續)

- 語法:
 - **int** *name*;

變數 (續)

- 語法:
 - **int** *name* = initval;

變數 (續)

- 語法:

- `int x = 5, y, z, w = 2 × (5 - x), my1_var3 = 2;`

變數 (續)

- 語法:
 - `int x = 5, y, z, w = 2 × (5 - x), my1_var3 = 2;`
- "int": 變數的型別/型態(type)
 - `int` 代表 `integer`; 其他一些型別如: `bool`, `char`, `double`

變數 (續)

- 語法:
 - `int x = 5, y, z, w = 2 × (5 - x), my1_var3 = 2;`
- "int": 變數的型別/型態(type)
 - `int` 代表 **integer**; 其他一些型別如: **bool, char, double**
- "*name*", "*x*", ... : 識別字(identifier), 變數的名稱
 - 大小寫不同; 不能使用保留字(reserved words)
 - 以字母或底線開頭, 後面可接任意長度的字母、底線及數字

變數 (續)

- 語法:
 - `int x = 5, y, z, w = 2 × (5 - x), my1_var3 = 2;`
- "int": 變數的型別/型態(type)
 - `int` 代表 **integer**; 其他一些型別如: **bool, char, double**
- "*name*", "*x*", ... : 識別字(identifier), 變數的名稱
 - 大小寫不同; 不能使用保留字(reserved words)
 - 以字母或底線開頭, 後面可接任意長度的字母、底線及數字
- "... = initval": 將變數初始化(initialized)為 `initval` 這個值
 - 未初始化的變數裡的值是**未知的**
 - 建議初始化**每個變數**

變數 (續)

- 為什麼以下的code不能用 (Or even making sense)

```
int x;
```

```
2 * 3 = x + 1;
```


變數 (續)

- 為什麼以下的code不能用 (Or even making sense)

```
int x;  
2 * 3 = x + 1;
```

- 下面的寫法C++中也沒有

```
a, b = b, (a + b)  
(x, y) = (2, 3)
```

變數 (續)

- 為什麼以下的code不能用 (Or even making sense)

```
int x;  
2 * 3 = x + 1;
```

- 下面的寫法C++中也沒有

```
a, b = b, (a + b)  
(x, y) = (2, 3)
```

- Question: 什麼東西可以放在等號左邊?

? = ...

- l-values 和 r-values!

Imperative & Sequencing

- 下面兩者有何不同?
 - $\triangleright x = 5;$
 - $\triangleright x = 5$
- 敘述句(statements) 代表會產生一些「作用」; 而表達式(expressions)則被算為值

```
#include <iostream>

int main() {
    int x = 73;
    std::cout << "line 1\n";
    x = 41;
    std::cout << "line 2\n";
    return 0;
}
```

- 程式會一行接一行按順序執行statements

Imperative & Sequencing (續)

- 任可結尾有分號的expression都是statement
 - 大部份該expression都會「做」一些事，如產生副作用
 - 敘述句也能被串在一起
 - 在之後的課程會談到 `if` 和 `for`

$\text{expr} \rightarrow x \mid 8 \mid 1 + z \mid 2 \times (w - 1)$
 $\mid \text{std}::\text{cout} << \text{"hello world"} \mid \dots$

$\text{statement} \rightarrow \text{expr};$
 $\mid \{ \text{statement statement} \dots \text{statement} \}$
 $\mid \text{if (condition) statement else statement}$
 $\mid \text{for (expr}_0; \text{expr}_0; \text{expr}_0) \text{statement}$

- ```
std::cout << "Hello World"; // ex 01
x = x+5; y = y*3; z = x+y; // ex 02
a = a*3+1; std::cout << a; // ex 03
```

# Hello World, again

- 一支程式的結構:

- `#include <iostream>`

引入需要的標頭檔，如 `iostream`

- ```
int main() {  
    // TODO: your codes here  
}
```

一個名為 `main` 的函數 (see: `main`)

- 程式會由 `main` 函數開始

使用函數 - I: pure functions

- 回憶一些高中熟悉的函數

▷ `sin(1.57)` (a.k.a. $\sin(\pi/2)$; `sin` defined in `cmath`)

1

使用函數 - I: pure functions

- 回憶一些高中熟悉的函數

▷ $\sin(1.57)$ (a.k.a. $\sin(\pi/2)$; \sin defined in `cmath`)
1

▷ $\cos(37 \times 3.14/180)$ (the 37-53-90 triangle; \cos is also defined in `cmath`)^{*1}
0.798832 ($\approx 4/5$)

使用函數 - I: pure functions

- 回憶一些高中熟悉的函數

▷ `sin(1.57)` (a.k.a. $\sin(\pi/2)$; `sin` defined in `cmath`)
1

▷ `cos(37 × 3.14/180)` (the 37-53-90 triangle; `cos` is also defined in `cmath`)^{*1}
0.798832 ($\approx 4/5$)

▷ `abs(3)` (`abs` is defined in `cstdlib`)
3

▷ `abs(-19)` $| - 19 |$
19

- 如同一般數學上的函數，給定一個input值便會得到一個output值

使用函數 - I: pure functions (*1)

- 我們前面都只使用整數，但這裡卻出現了浮點數(*floating point numbers*): 由於浮點數在電腦中是以二元分數(binary fraction)來儲存的，因此用來表達十進位的小數精準度有限。
- double是浮點數的一種型別。於是問題來了，當不同的數值型別在如何交互運算 -- $5 * 3.14$ 的型別是什麼?
- 初學者只要先記得在運算時，所有的數值都會變轉成 double

$2.0 \times 3 + 5$
→ $2.0 \times 3.0 + 5$ (3轉型為double)
→ $6.0 + 5$ (浮點數運算)
→ $6.0 + 5.0$ (5轉型為double)
→ 11.0

使用函數 - II: `main` revisited program structure

```
int main() {
```

- 這裡`int`表示了`main`的回傳值的型別 (i.e. `main()`, `sin(...)`, `cos(...)`, `abs(...)`計算之後的值).
 - `sin`, `cos`會回傳`double`，而`abs`回傳`int`:

```
double sin(...) { ... }  
double cos(...) { ... }  
int abs(...) { ... }
```

- 目前 `main`並沒有使用任何的參數(*parameter*) (不像 `sin`, `cos`, `abs`)。我們用放在`main`後面的“`()`”來表示。
- 呼叫`main`會執行所有在{ }中的statements。

```
}
```

基本輸入/輸出 - `std::cout`

- 使用方式:

```
std::cout << ("string"lvar1) << ("string"lvar2) << ... ;
```

- 字串(string) 是由一連串的被雙引號包住的字元所組成
- 可以用<<將多個變數或字串接起來輸出。

```
int x = 5, y = 31;  
std::cout << "hello\n";  
std::cout << "the value of x is " << x << "\n";  
std::cout << x << " + " << y << " = " << x + y << "\n";
```

* `\n` 代表換行字元. 詳見 [appendix](#).

基本輸入/輸出 - `std::cin`

- 使用方式:

`std::cin >> arg1 >> arg2 >> ... ;`

- ```
int x, y, z;
std::cin >> x;
std::cin >> x >> y;
std::cin >> z >> y >> x;
```

# From Thinking to Programming

# 解二元一次方程組

- 解以下方程式

$$a_1x + b_1y = c_1$$

$$a_2x + b_2y = c_2$$

- 假設
  - 係數皆為整數
  - 恰一組解

# 解二元一次方程組

- 解以下方程式

$$a_1x + b_1y = c_1$$

$$a_2x + b_2y = c_2$$

- 假設
  - 係數皆為整數
  - 恰一組解
- 給定任何實際的方程式，可以直接計算答案

$$\begin{cases} x + 2y = -4 \\ 2x + y = 1 \end{cases} \Rightarrow \begin{cases} x + 2y = -4 \\ -3y = 9 \end{cases} \Rightarrow \begin{cases} x = 2 \\ y = -3 \end{cases}$$

- 相信大家都有學過克拉瑪公式(Cramer's rule)

# 解二元一次方程組

- 在現我們希望電腦能機械化、自動化的來幫我們解這個問題
- 將問題抽象化:
  1. 我們有輸入 (某個未知的方程組)
  2. 我們(會)有輸出 (解)
- 我們要做的事就是寫一個程式讀入資料並輸出結果
- Next step: 列出更詳細的步驟



## 解二元一次方程組

- 假裝你是一個機器人。我們並不知道會是哪個方程組，因此用未知數來表示係數。

$$a_1x + b_1y = c_1$$

$$a_2x + b_2y = c_2$$

## 解二元一次方程組

- 假裝你是一個機器人。我們並不知道會是哪個方程組，因此用未知數來表示係數。

$$a_1x + b_1y = c_1$$

$$a_2x + b_2y = c_2$$

- 用符號來解(假設 $a_1 \neq 0$ )

$$\begin{cases} a_1x + b_1y = c_1 \\ a_2x + b_2y = c_2 \end{cases} \Rightarrow \begin{cases} a_1x + b_1y = c_1 \\ (b_2 - \frac{a_2}{a_1}b_1)y = c_2 - \frac{a_2}{a_1}c_1 \end{cases}$$

$$\Rightarrow \begin{cases} a_1x + b_1y = c_1 \\ y = \frac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1} \end{cases} \Rightarrow \begin{cases} a_1x = c_1 - \frac{(a_1c_2 - a_2c_1)b_1}{a_1b_2 - a_2b_1} \\ y = \frac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1} \end{cases}$$

$$\Rightarrow (x, y) = \left( \frac{c_1b_2 - c_2b_1}{a_1b_2 - a_2b_1}, \frac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1} \right)$$

## 解二元一次方程組

- 於是我們得到了公式!

$$x = \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - a_2 b_1} \quad y = \frac{a_1 c_2 - a_2 c_1}{a_1 b_2 - a_2 b_1}$$

# 解二元一次方程組

- 於是我們得到了公式!

$$x = \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - a_2 b_1} \quad y = \frac{a_1 c_2 - a_2 c_1}{a_1 b_2 - a_2 b_1}$$

- 因此我們能寫出以下的程式。當然必須先將實際的係數存入變數中。

```
int a1, b1, c1;
int a2, b2, c2;

int x = (c1*b2 - c2*b1)/(a1*b2 - a2*b1);
int y = (a1*c2 - a2*c1)/(a1*b2 - a2*b1);
```

## 解二元一次方程組

- 於是我們得到了公式!

$$x = \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - a_2 b_1} \quad y = \frac{a_1 c_2 - a_2 c_1}{a_1 b_2 - a_2 b_1}$$

- 因此我們能寫出以下的程式。當然必須先將實際的係數存入變數中。

```
int a1, b1, c1;
int a2, b2, c2;
int det = a1*b2 - a2*b1;
int x = (c1*b2 - c2*b1)/(a1*b2 - a2*b1);
int y = (a1*c2 - a2*c1)/(a1*b2 - a2*b1);
```

## 解二元一次方程組

- The last step: 輸入及輸出

## 解二元一次方程組

- The last step: 輸入及輸出
- 我們可以用 `std::cin` 來輸入:

```
std::cin >> a1 >> b1 >> c1;
std::cin >> a2 >> b2 >> c2;
```

## 解二元一次方程組

- The last step: 輸入及輸出
- 我們可以用 `std::cin` 來輸入:

```
std::cin >> a1 >> b1 >> c1;
std::cin >> a2 >> b2 >> c2;
```

- 並使用 `std::cout` 來輸出:

```
std::cout << "x = " << x <<< "\ny = " << y << "\n";
```



# 解二元一次方程組

- The last step: 輸入及輸出
- 我們可以用 `std::cin` 來輸入:

```
std::cin >> a1 >> b1 >> c1;
std::cin >> a2 >> b2 >> c2;
```

- 並使用 `std::cout` 來輸出:

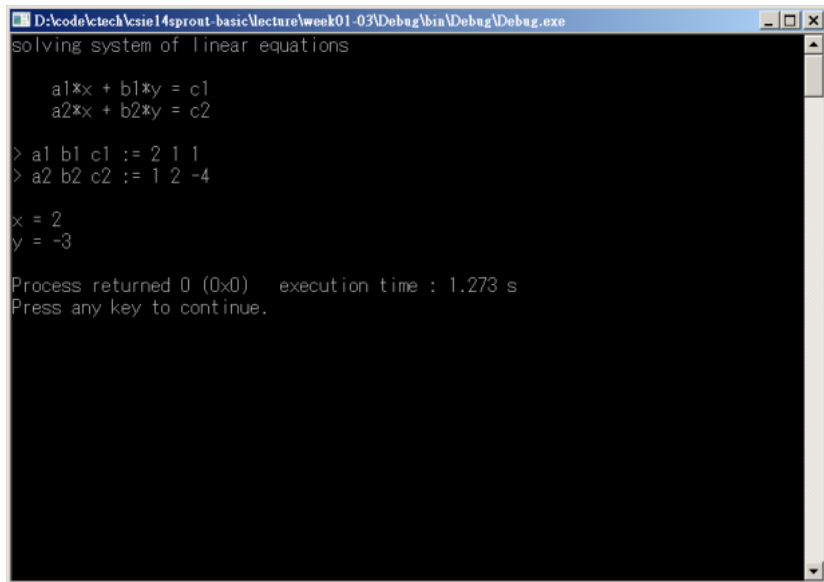
```
std::cout << "x = " << x <<< "\ny = " << y << "\n";
```

- 如果想要也可以多加一些訊息!

```
std::cout << "solving system of linear equations\n\n";
std::cout << " a1*x + b1*y = c1\n";
std::cout << " a2*x + b2*y = c2\n\n";
std::cout << "> a1 b1 c1 := ";
std::cin >> a1 >> b1 >> c1;
std::cout << "> a2 b2 c2 := ";
std::cin >> a2 >> b2 >> c2;
```

# 解二元一次方程組

- There we are!



```
D:\code\ctech\csie14spring-basic\lecture\week01-03\Debug\bin\Debug\Debug.exe
solving system of linear equations

 a1*x + b1*y = c1
 a2*x + b2*y = c2

> a1 b1 c1 := 2 1 1
> a2 b2 c2 := 1 2 -4

x = 2
y = -3

Process returned 0 (0x0) execution time : 1.273 s
Press any key to continue.
```

# 解二元一次方程組

## Full Program

```
#include<iostream>

int main() {
 int a1, b1, c1;
 int a2, b2, c2;

 std::cout << "solving system of linear equations\n\n";
 std::cout << " a1*x + b1*y = c1\n";
 std::cout << " a2*x + b2*y = c2\n\n";

 // read the coefficients from the keyboard
 std::cout << "> a1 b1 c1 := ";
 std::cin >> a1 >> b1 >> c1;

 std::cout << "> a2 b2 c2 := ";
 std::cin >> a2 >> b2 >> c2;

 // Cramer's rule; `det` for determinant
 int det = a1*b2 - a2*b1;
 int x = (c1*b2 - c2*b1)/det;
 int y = (a1*c2 - a2*c1)/det;

 std::cout << "\nx = " << x << "\ny = " << y << "\n";
 return 0;
}
```

## 解二元一次方程組 (Remark)

- 我們在做什麼? 為什麼會想導出公式?

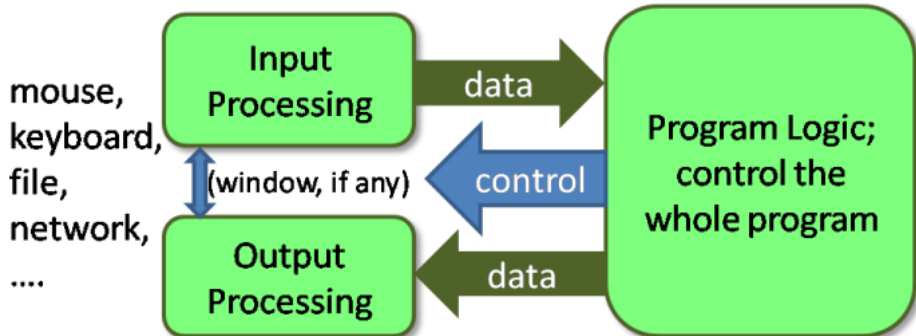
## 解二元一次方程組 (Remark)

- 我們在做什麼? 為什麼會想導出公式?
- 電腦是機器。它們會按照固定的指令來運作。
  - 在這個範例中，我們將抽象的「解方程組」轉換成實際的計算步驟。

## 解二元一次方程組 (Remark)

- 我們在做什麼? 為什麼會想導出公式?
- 電腦是機器。它們會按照固定的指令來運作。
  - 在這個範例中，我們將抽象的「解方程組」轉換成實際的計算步驟。
- 當然我們也可以實作高斯消去法，但會需要用到判斷、迴圈和陣列等知識。但對於二元一次，只要用特定的公式便能解了。

# Programming

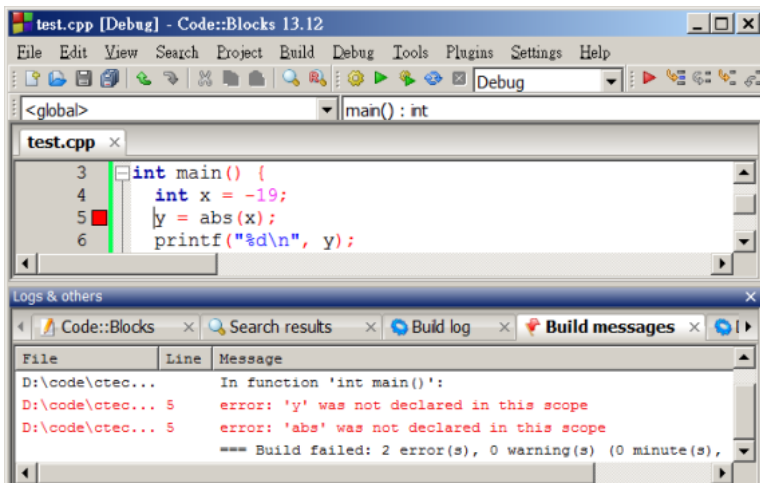


# Appendix



## Appendix 0: 面對Compilation Errors

- ...was not declared in this scope, undefined reference to...
  - 忘記include `iostream`, `cstdlib`, 等等
  - 忘記宣告變數，或拼錯字



```
test.cpp [Debug] - Code::Blocks 13.12
File Edit View Search Project Build Debug Tools Plugins Settings Help
<global> | main(): int
test.cpp x
3 int main() {
4 int x = -19;
5 y = abs(x);
6 printf("%d\n", y);

```

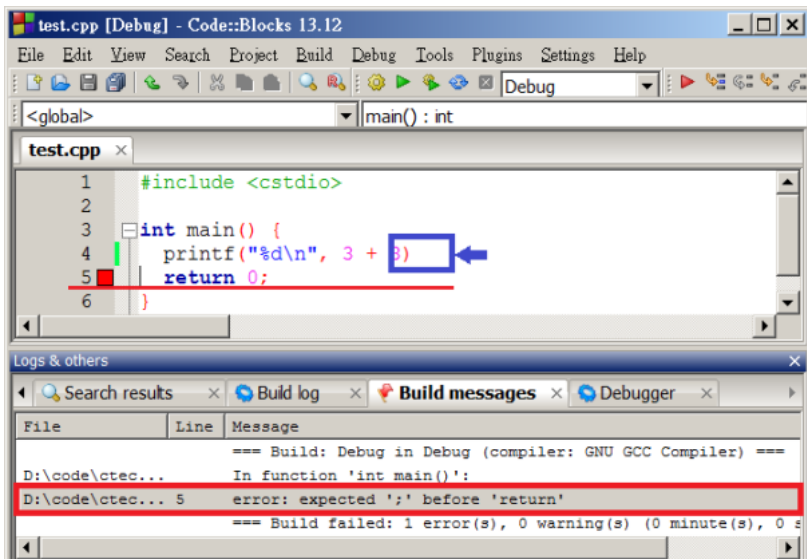
Logs & others

Code::Blocks x Search results x Build log x Build messages x

| File                                                     | Line | Message                                     |
|----------------------------------------------------------|------|---------------------------------------------|
| D:\code\ctec...                                          |      | In function 'int main()':                   |
| D:\code\ctec... 5                                        | 5    | error: 'y' was not declared in this scope   |
| D:\code\ctec... 5                                        | 5    | error: 'abs' was not declared in this scope |
| === Build failed: 2 error(s), 0 warning(s) (0 minute(s), |      |                                             |

# Appendix 0: 面對Compilation Errors

- Expected ... before ...
  - 檢查前一行結尾是否有遺漏；



# Appendix I: 一些運算子

- 賦值(Assignment)、算術(Arithmetic) (& 結合)

```
= // assignment
+ - * / % // usual arithmetic
+= -= *= /= %=
++ -- // increment & decrement
```

- 邏輯運算(Logical)

```
&& || ! and or not
```

- Bitwise

```
& | ^ << >> // bitwise and, or, xor, left shifting and right shifting
```

- 物件(Objects)...

```
. -> ->* .*
```

## Appendix II: Escape Sequences

- 有些字元是無法打出來的，因此我們用特殊的寫法來表示它們
  - 換行字元: `\n`
  - (horizontal) tab 字元: `\t`
  - 雙引號 (字串內): `\"`
  - 單引號 (字元內): `\'`
  - 跳脫字元()本少: `\\`
- `\` 是跳脫字元; 它告訴編譯器要特別處理下個字元

## Appendix II: Escape Sequences

- Example: 下列字串

the quick "brown fox" jumps  
over\ the lazy cat

會被寫成

```
"the quick \"brown fox\" jumps\\nover\\ the lazy cat"
```

- 值及其外部表示方式(external representation):
  - the quick "brown fox" jumps ... 是我們想要的值
  - "the quick \"brown fox\" jumps..." 則是其外部表現方式
- (在C++內部是怎麼表現的呢? (internal representation))

## Appendix III: 保留字(reserved words)

- 下列是C++的保留字

|            |          |           |                  |          |
|------------|----------|-----------|------------------|----------|
| alignas    | continue | friend    | register         | true     |
| alignof    | decltype | goto      | reinterpret_cast | try      |
| asm        | default  | if        | return           | typedef  |
| auto       | delete   | inline    | short            | typeid   |
| bool       | do       | int       | signed           | typename |
| break      | double   | long      | sizeof           | union    |
| case       | mutable  | static    | dynamic_cast     | unsigned |
| catch      | else     | namespace | static_assert    | using    |
| char       | enum     | new       | static_cast      | virtual  |
| char16_t   | explicit | noexcept  | struct           | void     |
| char32_t   | export   | nullptr   | switch           | volatile |
| class      | extern   | operator  | template         | wchar_t  |
| const      | false    | private   | this             | while    |
| constexpr  | float    | protected | thread_local     |          |
| const_cast | for      | public    | throw            |          |