

## 第二章



2.1	簡介
2.2	一個簡單的C程式：列印一行文字
2.3	另一個簡單的C程式：將兩個整數相加
2.4	記憶體觀念
2.5	C的算術運算
2.6	判斷：等號運算子和關係運算子
2.7	安全程式設計
	摘要

Wireless Access



2.1  
2.2  
2.3  
2.4  
2.5  
2.6

### 2.1 簡介

- C語言使得程式設計者能以結構化且有條理的方法來設計電腦程式。本章將簡單介紹C程式的設計，並舉出數個例子來說明C語言的一些重要特性。



2.3  
2.4  
2.5  
2.6  
2.7  
摘要

Wireless Access



## 2.2 一個簡單的C程式：列印一行文字

```
1 // Fig. 2.1: fig02_01.c
2 // A first program in C.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main( void )
7 {
8     printf( "Welcome to C!\n" );
9 } // end function main
```

Welcome to C!

- » 註解: `//, /*xxxxxx*/`
- » 前置處理指令 `#include < xxx.h >`
- » 主函式 `main(){ 程式主體 }`



2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要

### 一 註解

- 請養成習慣程式前幾行要先說明該程式處理目標與作者姓名學號

```
// Homework 1 in week 2
// Try to print out my Resume
// By AM 04xxxx 黃XX
```
- 以 `//` 開頭的行表示這行是註解 (comment)。
- 也可以 `/* */` 進行多行註解 (multi-line comments)，從 `/*` 開始且以 `*/` 結尾之間的任何內容都被會視為註解。
- 建議使用 `//`



2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要

2.1

2.2

2.3

2.4

2.5

2.6

2.7

摘要

## – #include 前置處理器指令

- `#include <stdio.h>`
- 是個**C前置處理器 (C preprocessor)** 指令。在程式編譯之前，前置處理器會處理以#開頭的每一行。第3行告訴前置處理器將**標準輸入／輸出標頭檔 (standard input/output header) (<stdio.h>)** 引入到程式裡面。

Wireless Access



## 前置處理指令 – #include

- #include 引入標頭檔可分為下列兩種格式：

### – #include <xxx.h>

- xxx.h 為 C 編譯器提供的標頭檔，並且存放在編譯器內定的目錄中，使用此種格式，前置處理器會自動到內定目錄中找到標頭檔。

### – #include "ooo.h"

- ooo.h 不是編譯器提供的標頭檔，所以程式設計師必須標明該檔案所在目錄，以便前置處理器取得該檔案。

原始程式

前置處理器

處理前置指令

程式碼

編譯

目的碼

連結

可執行碼

執行

Wireless Access



## 前置處理指令 — #include

ANSI C 函式庫  
標頭檔

函式庫標頭檔名稱	函式種類
<stdio.h>	標準輸入與輸出
<ctype.h>	字元分類測試
<string.h>	字串處理與轉換
<math.h>	數學函式
<stdlib.h>	標準函式庫，提供各類基本函式
<assert.h>	例外偵測，有助於除錯
<stdarg.h>	引數串列的測試
<setjmp.h>	非區域跳躍
<signal.h>	訊號偵測
<time.h>	提供各類時間函式
<limits.h> 及 <float.h>	float.h 提供浮點數的精確位數定義， limits.h 提供某些極限值的定義

2.3  
2.4  
2.5  
2.6  
2.7  
摘要

Wireless Access

### main 函數

```
int main (void)
{

}
```

- 是每個C程式都有的部分。
- 每個函式本體 (body)
  - 以左大括號 { 表示開始
  - 以右大括號 } 表示每個函式的結束位置
  - 這兩個括號以及之間的程式稱為一個區塊 (block)。

2.1  
2.2  
2.3  
2.4  
2.5  
2.6  
2.7  
摘要

WATSE

2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要

- ❖ C語言屬於**模組化**設計的一種語言，而C語言的模組則是以「**函式**」來加以表示。
- ❖ 換句話說，C程式是由各個不同功能的函式所組成，並且函式與函式之間可透過呼叫及回傳值方式加以聯繫，一個函式的基本定義格式如下：

```

函式回傳值型態 函式名稱(參數)
{
    函式內容 (敘述群)
}


```
- ❖ **main()**函式是命令列(Console Mode)C程式的**進入點**，換句話說，當我們在命令列式的作業系統中執行由C所撰寫的應用程式時，會先從**main()**函式開始執行。

## 敘述

2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要

- 指示電腦執行一個**動作 (action)**，包括指令、括號、括號裡的**引數 (argument)** 及分號 (semicolon「;」) 等，稱為一道**敘述式 (statement)**。每行敘述須以分號 做結束。
- ❖ C語言的敘述除了函式呼叫之外，還有以下幾種類型，我們將在後面章節中加以說明：

▪ 算式敘述	(Expression Statement)
▪ 複合敘述	(Compound Statement)
▪ 選擇敘述	(Selection Statement)
▪ 迴圈敘述	(Iteration Statement)
▪ 標籤敘述	(Labeled Statement)
▪ 跳躍敘述	(Jump Statement)




- 2.1
- 2.2
- 2.3**
- 2.4
- 2.5
- 2.6
- 2.7
- 摘要


■ 輸出敘述

- 第8行

```
printf( "Welcome to C!\n" );
```


- 指示電腦執行一個**動作 (action)**，將雙引號內的**字串 (string)** 顯示在螢幕上。字串有時稱做**字元字串 (character string)**。
- 跳脫序列: 請注意字元 `\n` 並不會顯示在螢幕上。反斜線符號 (`\`) 稱為**跳脫字元 (escape character)**。它表示**printf**應印出一些特殊的字元。跳脫序列 `\n` 表示**新增一行 (newline)**。其他一些常見的跳脫序列請參閱圖2.2。





- 2.3
- 2.4**
- 2.5
- 2.6
- 2.7
- 摘要

» 跳脫序列	說明
» <code>\n</code>	換行。將游標移到下一行開始處。
» <code>\t</code>	水平tab。將游標移到下一個tab定位點。
» <code>\a</code>	警告。讓系統發出聲音，或是顯示警告但不改變游標位置。
» <code>\\</code>	反斜線符號。在字串中插入反斜線字元。
» <code>\"</code>	雙引號。在字串中插入雙引號。
»	圖2.2 一些常見的跳脫序列



## 自由格式與空白字元



2.1  
2.2  
2.3  
2.4  
2.5  
2.6  
2.7  
摘要

### ■ 空行與空白

- 第4行僅僅只有一個空行。你可以使用空行、空格及定位字元 (tabs) 使程式較容易閱讀。這些字元統稱為**空白 (white space)**，空白通常會被編譯器略過

- C語言採用自由格式撰寫，換句話說，您可以去除程式中各敘述間的所有空白字元(spaces、tabs…等等)及換行符號 ( carriage、return )，編譯器仍會正確編譯程式，例如：

```
main(void){ printf("歡迎使用C語言!\n"); printf("這是一個簡單的C程式.\n");  
            system("pause");}
```



## 自由格式與空白字元



2.1  
2.2  
2.3  
2.4  
2.5  
2.6  
2.7  
摘要

雖然省略空白字元以及換行符號能夠使得程式行數減少，但並不會加速程式的執行效率，因為不論是範例的表示方法或上述表示方法，編譯器都將產生相同的輸出結果。不過上述的撰寫格式，則比範例2-1更難以閱讀。為了日後維護的方便，**強烈建議**讀者應該培養程式碼縮排及適當換行的習慣。

- ❖ 『"』內的空白字元並不會被編譯器忽略，因為『"』在C語言中，是用來表示字串。
- ❖ C程式和Unix/Linux作業系統一樣，對於字元的大小寫是有所區別的，因此您**不能**將main(void)改寫為Main(void)或MAIN(VOID)。



## ■ 連結器與可執行程式

- 標準函式庫的函式 (如**printf**或**scanf**) 並不是C語言的一部分。所以編譯器無法發現他們是否拼字錯誤。當編譯器在編譯**printf**敘述式時，它只是在目的碼中空出一塊空間來「呼叫」函式庫函式。編譯器不知道函式庫函式在哪裡——但是**連結器**知道。



- 一直要到連結器進行連結的時候，才會找出函式庫函式的位置，然後在目的碼中插入對函式庫函式正確的呼叫。此時，目的碼便是完整且可執行的了。因此，已經連結的程式通常稱為**可執行 (executable)** 的程式。





## ■ 使用多個 printf

- **printf** 函式能以數種不同的方式印出 **Welcome to C!** 舉例來說，圖2.3的程式就可以產生與圖2.1程式相同的輸出。這是因為每個**printf**都會從上一個**printf**結束列印的地方開始列印。第一個**printf** (第8行) 印出**Welcome**及一個空格，而第二個**printf** (第9行) 便從同一行的空格之後開始列印。



2.1  
2.2  
2.3  
2.4  
2.5  
2.6  
2.7  
摘要



```
1 // Fig. 2.3: fig02_03.c
2 // Printing on one line with two printf statements.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main( void )
7 {
8     printf( "Welcome " );
9     printf( "to C!\n" );
10 } // end function main
```


Welcome to C!

圖2.3 以兩個**printf**敘述式列印一行文字



2.1  
2.2  
2.3  
2.4  
2.5  
2.6  
2.7  
摘要






- 2.1
- 2.2
- 2.3
- 2.4
- 2.5
- 2.6
- 2.7
- 摘要

```
1 // Fig. 2.4: fig02_04.c
2 // Printing multiple lines with a single printf.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main( void )
7 {
8     printf( "Welcome\nto\nC!\n" );
9 } // end function main
```

Welcome  
to  
C!



## 2.3 另一個簡單的C程式：將兩個整數相加

- 我們的下一個程式使用了標準函式庫函式 **scanf**，來讀進使用者鍵入的兩個整數，然後計算這兩個值的和，再以 **printf** 將結果印出來。程式和輸出範例如圖2.5所示。

```

1 // Fig. 2.5: fig02_05.c
2 // Addition program.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main( void )
7 {
8     int integer1; // first number to be entered by user
9     int integer2; // second number to be entered by user
10    int sum; // variable in which sum will be stored
11
12    printf( "Enter first integer\n" ); // prompt
13    scanf( "%d", &integer1 ); // read an integer
14
15    printf( "Enter second integer\n" ); // prompt
16    scanf( "%d", &integer2 ); // read an integer
17
18    sum = integer1 + integer2; // assign total to sum
19
20    printf( "Sum is %d\n", sum ); // print sum
21 } // end function main

```

```

Enter first integer
45
Enter second integer
72
Sum is 117

```

圖2.5 一個有關加法的程式

## ■ 變數與定義

```

int integer1; // first number to be entered by user
int integer2; // second number to be entered by user
int sum; // variable in which sum will be stored

```

- 都是**定義 (definitions)**。integer1、integer2和sum是**變數 (variables)** 名稱。變數就是電腦記憶體中，存放數值供程式使用的位置。
- 前述的定義也可以合併成為單一定義敘述式，如下：

```
int integer1, integer2, sum;
```

## ■ 識別字和大小寫區別

- C語言當中變數的名稱可以是任意合法的**識別字 (identifier)**。識別字是由**字母**、**數字**和**底線 ( \_ )**組成的一連串字元，但第一個字元不可以是數字。識別字的長度沒有限制，不過C標準中規定編譯器只需辨認前31個字元。C會**區分大小寫 (case sensitive)**，所以**a1**與**A1**是不同的識別字。



2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要



## ■ 提示訊息

- 第12行

```
printf( "Enter first integer\n" ); // prompt
```

- 如字面印出**Enter first integer**，並將游標移至下一行的開頭。這個訊息就叫作**提示 (prompt)**，因為它指示使用者進行某特定動作。



2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要






2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要

### ■ Scanf函式與格式化輸入

- 下一個敘述式

```
scanf( "%d", &integer1 ); // read an integer
```

- 使用**scanf**從使用者處讀取一個數值。scanf函式由標準輸入 (通常是鍵盤) 讀取輸入值。



2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要

### ■ 指定敘述句

- 第18行的**指定敘述句 (assignment statement)**

```
sum = integer1 + integer2; // assign total to sum
```

- 計算了變數**integer1**和**integer2**的和，並使用指定運算子 **= (assignment operator)** 將結果設定給變數**sum**。大部分的計算都是用指定敘述式來執行。運算子 **=** 和 **+** 稱為二元運算子 (binary operator)，因為它們具有二個**運算元 (operands)**。

## ■ 使用格式控制字串列印

- 第20行

```
printf( "Sum is %d\n", sum ); // print sum
```

- 呼叫函式**printf**，在螢幕上印出字面常數**Sum is**以及變數**sum**的值。



2.1

2.2

2.3

2.4

2.5

2.6

2.7

摘要



## ■ 在**printf**敘述式裡執行計算

- 我們也可以將計算的執行放在**printf**敘述式裡。上面兩道敘述式可以合併成

```
printf( "Sum is %d\n", integer1 + integer2 );
```

- 第21行的右大括號 ( ) 表示**main**函式結束。



2.1

2.2

2.3

2.4

2.5

2.6

2.7

摘要



## 2.4 記憶體觀念

- 像 `integer1`、`integer2` 和 `sum` 這樣的變數名稱都會對應到電腦裡的記憶體位置 (locations)。每個變數都具有一個名稱 (name)、一個**型別 (type)**、及一個**數值 (value)**。
- 在圖2.5的加法程式中，當敘述式 (第13行)

```
scanf( "%d", &integer1 ); // read an integer
```



2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要

- 執行時，使用者鍵入的數值會被放在 `integer1` 所被指定的記憶體位置。假設使用者輸入數字 **45** 作為變數 `integer1` 的值。



圖2.6 顯示變數名稱及數值的記憶體位置



2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要

2.1

2.2

2.3

2.4

2.5

6

2.7

摘要

- 回到我們的加法程式，當下列敘述式 (第16行)

```
scanf( "%d", &integer2 ); // read an integer
```

- 執行時，假設使用者輸入**72**，這個數值就會存到 **integer2**的位置，其記憶體配置如圖2.7所示。

integer1	45
integer2	72

圖2.7 輸入兩個變數之後的記憶體位置

Wireless Access



2.1

2.2

2.3

2.4

2.5

2.6

2.7

摘要

- 在程式讀取 **integer1**和**integer2**的值之後，它會將這兩個值相加，然後將其和放到變數**sum**。敘述式 (第18行)

```
sum = integer1 + integer2; // assign total to sum
```

- 會執行加法，並取代目前**sum**的值。

integer1	45
integer2	72
sum	117

情形





## 2.5 C的算術運算

C 的運算	算術運算子 (arithmetic operator)	代數運算式	C 運算式
加法	+	$f + 7$	$f + 7$
減法	-	$p - c$	$p - c$
乘法	*	$bm$	$b * m$
除法	/	$x / y$ or $\frac{x}{y}$ or $x \div y$	$x / y$
模數除法	%	$r \bmod s$	$r \% s$

圖2.9 算術運算子

### ■ 整數除法與模數運算子

- **整數除法 (Integer division)** 得到的結果會是個整數。如7/4會等於1，而17/5會等於3。C還提供了**模數運算子 % (remainder operator)**，它的運算結果是整數相除後的餘數。模數運算子是個整數運算子，它的運算元必須是整數。運算式 $x \% y$ 會產生x除以y的餘數。因此 $7 \% 4$ 等於3，而 $17 \% 5$ 等於2。

### ■ 橫行形式的算術運算式

- C的算術運算式須以**橫行形式 (straight-line form)**輸入電腦，方便將程式輸入電腦。因此，像是「a除以b」的運算式就必須寫成 $a/b$ ，如此所有的運算子和運算元都會排成橫行。

### ■ 用小括號將子運算式分群

- 在C運算式中，小括號的用法跟代數運算的用法很像。例如，要將a乘以 $b + c$ 的值，就寫成 $a * (b + c)$ 。



### ■ 運算子優先順序規則

- 1.在同一對小括號中的運算子先進行計算。小括號具有「最高優先權」。若為**巢狀 (nested)**，或稱**嵌入 (embedded)**的**小括號 (parentheses)**，如

$$((a+b)+c)$$

– 最裡面那對小括號中的運算子會先計算。

- 2.接著會處理乘法、除法和模數運算。若運算式有多個乘法、除法和模數運算，則從左到右進行計算。因此乘法、除法和模數運算具有相同的優先權層級。

- 3.接下來進行加和減的運算。若運算式有多個加減法運算，則從左到右進行計算。加和減的優先順序等級是相同的，但是優先次序低於乘法、除法以及模數運算子。



#### 4. 最後處理的是賦值運算子。

運算子	運算	計算的順序 (優先順序)
( )	小括號	優先計算。如果小括號是巢狀的，則會先計算最內層的。若有數個此類型的運算，則會從左到右運算。
*	乘法	第二個計算。若有數個此類型的運算，則會從左到右運算。
/	除法	
%	模數除法	
+	加法	第三個計算。若有數個此類型的運算，則會從左到右運算。
-	減法	
=	指定	最後計算。

圖2.10 算術運算子的優先順序


#### ■ 簡單的代數與C運算式

- 現在看看幾個遵照運算子優先順序的運算式。每個範例都會列出一個代數運算式和對等的C運算式。下面的算式計算了5個數的算術平均數：

$$\text{Algebra: } m = \frac{a+b+c+d+e}{5}$$

$$\text{C: } m = (a + b + c + d + e) / 5;$$

- 假如遺漏了其中的括號，便會得到  $a + b + c + d + e/5$ ，運算式就算錯了



- 2.1
- 2.2
- 2.3**
- 2.4
- 2.5
- 2.6
- 2.7
- 摘要

- 下面的算式是直線的方程式：

Algebra:  $y = mx + b$



C:  $y = m * x + b;$


- 這裡不需要小括號。程式會先計算乘法運算，因為乘法的優先權高於加法的優先權。
- 以下算式包括模數運算 (%)、乘法、除法、加法、減法和賦值等運算：

Algebra:  $z = pr \% q + w/x \quad y$

C:  $z = p * r \% q + w / x - y;$

6
1
2
4
3
5



- 2.1
- 2.2
- 2.3**
- 2.4
- 2.5
- 2.6
- 2.7
- 摘要



### ■ 二次多項式計算

- 為了進一步了解運算子優先順序，我們看C如何計算一個二次多項式。

$y = a * x * x + b * x + c;$

6
1
2
4
3
5

- 在運算式底下圈起來的數字，表示C執行這些運算的順序。


2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要

- 跟代數運算一樣，在運算式中加入非必要的小括號可讓運算式的計算順序更清楚。這些括號稱為**多餘括號 (redundant parentheses)**。例如，前述的敘述可加上以下的小括號：

$$y = ( a * x * x ) + ( b * x ) + c;$$

## 2.6 判斷：等號運算子和關係運算子

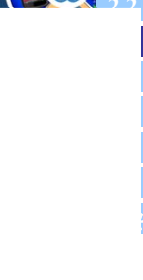
標準代數的等號或關係運算子	C 的等號或關係運算子	C 的條件式範例	C 條件式的意義
<b>等號運算子</b>			
=	==	x == y	x 等於 y
≠	!=	x != y	x 不等於 y
<b>關係運算子</b>			
>	>	x > y	x 大於 y
<	<	x < y	x 小於 y
≥	>=	x >= y	x 大於或等於 y
≤	<=	x <= y	x 小於或等於 y



```

1 // Fig. 2.13: fig02_13.c
2 // Using if statements, relational
3 // operators, and equality operators.
4 #include <stdio.h>
5
6 // function main begins program execution
7 int main( void )
8 {
9     int num1; // first number to be read from user
10    int num2; // second number to be read from user
11
12    printf( "Enter two integers, and I will tell you\n" );
13    printf( "the relationships they satisfy: " );
14
15    scanf( "%d%d", &num1, &num2 ); // read two integers
16
17    if ( num1 == num2 ) {
18        printf( "%d is equal to %d\n", num1, num2 );
19    } // end if
20


```



```

21    if ( num1 != num2 ) {
22        printf( "%d is not equal to %d\n", num1, num2 );
23    } // end if
24
25    if ( num1 < num2 ) {
26        printf( "%d is less than %d\n", num1, num2 );
27    } // end if
28
29    if ( num1 > num2 ) {
30        printf( "%d is greater than %d\n", num1, num2 );
31    } // end if
32
33    if ( num1 <= num2 ) {
34        printf( "%d is less than or equal to %d\n", num1, num2 );
35    } // end if
36
37    if ( num1 >= num2 ) {
38        printf( "%d is greater than or equal to %d\n", num1, num2 );
39    } // end if
40 } // end function main

```


2.1

Enter two integers, and I will tell you the relationships they satisfy: 3 7

3 is not equal to 7

3 is less than 7

3 is less than or equal to 7

Enter two integers, and I will tell you the relationships they satisfy: 22 12

22 is not equal to 12

22 is greater than 12



22 is greater than or equal to 12

Enter two integers, and I will tell you the relationships they satisfy: 7 7

7 is equal to 7

7 is less than or equal to 7

7 is greater than or equal to 7



2.1
2.2
2.3
2.4
2.5
2.6
2.7
摘要

除了指定運算子 = 之外，其他所有運算子的結合性都是由左至右。  
指定運算子 (=) 是從右至左結合

運算子	結合性
()	從左到右
*   /   %	從左到右
+   -	從左到右
<   <=   >   >=	從左到右
==   !=	從左到右
=	從右到左

圖2.14 目前討論過的運算子優先順序和結合性




[2.1](#)  
[2.2](#)  
[2.3](#)  
[2.4](#)  
[2.5](#)  
[2.6](#)  
[2.7](#)  
[摘要](#)

**關鍵字**

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

C99的關鍵字

`_Bool` `_Complex` `_Imaginary` `inline` `restrict`

C11 草案標準的關鍵字

`_Alignas` `_Alignof` `_Atomic` `_Generic` `_Noreturn` `_Static_assert` `_Thread_local`

[2.1](#)  
[2.2](#)  
[2.3](#)  
[2.4](#)  
[2.5](#)  
[2.6](#)  
[2.7](#)  
[摘要](#)

► C程式設計藝術(第七版)

## 2.7 安全程式設計

- 我們在序章提到C安全程式標準 ( The CERT C Secure coding Standard )，其中提出了一些指導原則，幫助你免於在程式實作時，使系統暴露在攻擊之下。

全華
第50頁，共85頁



### ◆避免單一參數的printf函式

- 指導原則的其中一項，就是避免在使用**printf**時，只用一個字串當作參數。如果你要顯示一個以換行 (newline) 結束的字串，請使用**puts函式 (puts function)**，它將參數裡的字串後加上換行字元並顯示。如圖2.1的第8行。

```
printf( "Welcome to C!\n" );
```

- 應該寫成：

```
puts( "Welcome to C!" );
```

- 我們字串後面沒有**\n**，因為**puts**會幫自動加上。
- 如果你要顯示一個字串且不使用換行符號結尾，請在使用**printf**時帶入兩個參數。**轉換指定詞 (conversion specifier) %s** 可用於顯示一個字串，例如圖2.3第8行。

```
printf( "Welcome " );
```

- 應該寫成：

```
printf( "%s", "Welcome " );
```