

**C**

語言基本架構分成三大區：前置處理 (preprocessing) 區、函數 (function) 定義區及註解 (comment) 區。且至少要有函數定義 main() ，因為 main 為 C 語言[函數](http://pydoing.blogspot.com/2010/06/c-function.html)中預設的保留字，函數為 C 程式執行的單位， main() 函數則為 C 程式執行的起點，這是說，所有可執行的 C 程式編譯時都會尋找名為 main 的函數，然後呼叫函數 main() 開始執行。

C**語言**[程式，包含](http://pydoing.blogspot.com/2010/07/c-ch0100.html)

* [前置處理](http://pydoing.blogspot.com/2010/07/c-ch0101.html)
* [註解](http://pydoing.blogspot.com/2010/07/c-ch0102.html)
* [函數](http://pydoing.blogspot.com/2010/07/c-ch0103.html)
* [變數、宣告與關鍵字](http://pydoing.blogspot.com/2010/07/c-ch0104.html)
* [常數與指派](http://pydoing.blogspot.com/2010/07/c-ch0105.html)
* [呼叫](http://pydoing.blogspot.com/2010/07/c-ch0106.html)
* [回傳](http://pydoing.blogspot.com/2010/07/c-ch0107.html)
* [程式碼的編排](http://pydoing.blogspot.com/2010/07/c-ch0108.html)

前置處理區，如範例第一行。

C 程式在編譯前可以先進行[前置處理 (preprocessing)](http://pydoing.blogspot.com/2010/06/c-preprocessing.html) ，所謂的前置處理包括引入標頭檔 (header file) 、定義常數 (constant) 或巨集 (macro) 、條件式編譯等。若要使用 C 語言[標準函數庫](http://pydoing.blogspot.com/2010/07/c-std.html)中的功能，就需要先引入相關的[標頭檔](http://pydoing.blogspot.com/2010/06/c-header.html)。

此例中，由於我們用到函數 [printf()](http://pydoing.blogspot.com/2010/07/c-printf.html) ，這是 C 語言用來將格式化字串傳送到標準輸出裝置的函數，而這個函數原型宣告在標頭檔 [stdio.h](http://pydoing.blogspot.com/2010/07/c-stdio.html) 中，因此我們要用到這個函數時，就得在前置處理時先引入。

井字號加上 include

|  |
| --- |
| #include <stdio.h> |

這是前置處理器的指令，作為引入標頭檔之用，注意，引入標準函數庫是用角括號 < > 將標頭檔圍起來，若是引入自己寫的標頭檔，就要用雙引號 " " ，例如

|  |
| --- |
| #include "my\_header.h" |

這便會將自己寫的 my\_header.h 引入原始碼當中，好讓編譯器在編譯前先將 my\_header.h 中定義的東東囊括進來。

標頭檔通常會放往後開發程式所共同需要的東東，例如常數、巨集、[結構](http://pydoing.blogspot.com/2010/06/c-structure.html)、[函數原型](http://pydoing.blogspot.com/2010/06/c-prototype.html)的定義放在標頭檔中，由於這些其他程式也可能會用到，因此放在某一標頭檔中，然後就可以讓其他程式以相同方式利用囉！這是重新利用已經完成程式的方法，不需要重新再開發，也不需要重複相同的程式碼。

標準函數庫就是以這樣的概念建置的，許多經常用到的功能都已經先放在標準函數庫的標頭檔之中，因此需要用到這些功能時，只需要引入標準函數庫的標檔就可以使用囉！

註解 (comment) 區**，如**第 21 到 26 行

註解就是用為輔助的說明文字，編譯器編譯時會忽略所有的註解文字。我們在簡單例子中所提供的註解為有關程式的整體資訊，小程式也許不太需要註解就能讀懂，但是當程式發展越來越大的時候，也就是動輒數百到數千、數萬行之時，適當的註解有助於程式後續的發展及維護。

通常軟體開發團隊會有既定的註解模式，諸如哪些地方、什麼時候等等，我們提供的註解是站在教學立場，以提供輔助說明為主。

C 語言的註解方式有兩種，第一種如簡單例子，利用一組斜線加星號 /\* ，然後一直到另一組星號加斜線為止 \*/

|  |
| --- |
| /\* 註解第一行 \*/  /\* 註解第二行 \*/  /\* 註解第三行 \*/ |

這種方式也可跨行

|  |
| --- |
| /\* 註解第一行  註解第二行  註解第三行 \*/ |

也可加在任何程式碼之前或之後

|  |
| --- |
| int month, day, hour, min; /\* 宣告為整數型態的變數 \*/  /\* 注意，這是指標 \*/  char \*name; |

第二種方式為連用兩個斜線 //

|  |
| --- |
| // 註解第一行  // 註解第二行  // 註解第三行 |

這只能做單行的註解，同時只能放在無程式碼的單行或程式碼之後

|  |
| --- |
| int month, day, hour, min; // 宣告為整數型態的變數  // 注意，這是指標 char \*name; |

上例中的[字元型態](http://pydoing.blogspot.com/2010/06/c-datatype.html#1)的[指標](http://pydoing.blogspot.com/2010/06/c-pointer.html)變數 name 被註解化了，因此編譯時會發生錯誤。

註解化可作為發展程式時一種進行測試的技巧，例如除錯時不確定錯誤發生的地方，可將該處先註解化，然後先進行編譯，看是否能編譯成功，以期順利找出錯誤的地方。

[函數 (function)](http://pydoing.blogspot.com/2010/06/c-function.html) 定義區，第 3 到 19 行

C 程式是以函數為執行單位，每個可執行的 C 程式一定要有個函數 main() ，因為 main 為 C 語言函數的保留字，編譯 C 程式時，編譯器 (compiler) 都會先尋找名為 main 的函數作為程式執行的起點。若無函數 main()，編譯會產生錯誤訊息。

函數的定義格式如下

|  |
| --- |
| 回傳值型態 函數名稱(參數列)  {  ....    return 回傳值;  } |

上面第一列為函數的標頭，然後左大括弧 { 一直到右大括弧 } 便是函數定義的區域，右大括弧的上方有一行 return 陳述，這便是函數回傳數值的地方。若以簡單例子的函數 main() 來看，函數標頭與回傳的 return 陳述標示如下

通常自行定義的函數可以先做[函數原型 (prototype)](http://pydoing.blogspot.com/2010/06/c-prototype.html) 的宣告，如

|  |
| --- |
| 回傳值型態 函數名稱(參數列); |

函數原型的宣告會放在 main() 之前，而通常不會對 main() 宣告函數原型。先行宣告函數原型，編譯器會檢查函數原型的回傳值型態及參數列中各參數的型態是否與定義相同。

C 語言的函數可以有回傳值 (return value) ，也可以沒有回傳值。回傳值的意思可以想成函數的計算結果，這跟數學中函數的觀念是一樣的，每個輸入值只會對應到唯一的輸出值，因此 C 語言的函數只能有一個回傳值。另外， C 語言中凡是值都有型態，因此回傳值也有所屬的資料型態，這在定義或宣告時都需要標明。

函數也可以沒有回傳值，沒有回傳值就像只是做一些工作，這些工作並不需要計算結果。 C 語言中沒有回傳值的函數須宣告為 void ， void 也為 C 語言的[關鍵字 (keyword)](http://pydoing.blogspot.com/2010/06/c-keyword.html) 之一

|  |
| --- |
| void 函數名稱(參數列); |

C 語言的函數可以有參數 (parameter) ，也可以沒有參數。所謂的參數就像數學中函數的輸入值， C 語言的函數可以有多個參數，就像多元方程式一樣。函數名稱後的小括弧為宣告參數的地方，可有多個，所以稱之為參數列。因為 C 語言中凡是值都有型態，所以參數也須宣告所屬的資料型態。

同樣的，函數也可以沒有參數，這時候參數列就須宣告為 void

|  |
| --- |
| 回傳值型態 函數名稱(void); |

簡單例子中的函數 main() 不接受參數，所以函數定義的標頭為

|  |
| --- |
| int main(void) |

int 為[整數資料型態](http://pydoing.blogspot.com/2010/06/c-datatype.html#2)，同時 int 也為關鍵字之一，函數 main() 的回傳值定義為回傳一個 int 型態的數值。

注意，函數名稱為[識別字](http://pydoing.blogspot.com/2010/06/c-identifier.html)之一，以上舉例的「函數名稱」僅作為位置標示，通常編譯器只接受英文、數字及底線符號作為識別字，使用中文作識別字名稱會發生編譯錯誤。

[變數 (variable)](http://pydoing.blogspot.com/2010/06/c-identifier.html) 宣告，程式碼的第 5 行及第 6 行

程式中的變數與數學使用的變數相似，都是用某些符號或單字代替某些數值，從而得以方便計算過程。程式語言中的變數屬於識別字 (identifier) ， C 語言對於識別字有一定的命名規則，例如只能用英文大小寫字母、數字以及底線符號

|  |
| --- |
| \_ |
| a | b | c | d | e | f | g | h | i | j | k | l | m |
| n | o | p | q | r | s | t | u | v | w | x | y | z |
| A | B | C | D | E | F | G | H | I | J | K | L | M |
| N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |  |  |

其中，數字不能用作識別字的開頭，單一識別字裡不允許有空格，而如 int 、 char 為 C 語言的[關鍵字 (keyword)](http://pydoing.blogspot.com/2010/06/c-keyword.html) 之一，屬於程式語言的語法保留字，因此也不能用為自行定義的名稱。通常編譯器至少能讀取名稱的前 31 個字元，但外部名稱可能只能保證前六個字元有效。

C 語言的關鍵字有 34 個，如下列表

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

變數使用前要先進行[宣告 (declaration)](http://pydoing.blogspot.com/2010/06/c-declaration.html) ，宣告的主要目的是告訴編譯器這個變數屬於哪一種資料型態，好讓編譯器預先替該變數保留足夠的記憶體空間。宣告的方式很簡單，就是型態名稱後面接空格，然後是變數的識別名稱

|  |
| --- |
| char a; //變數 a 為字元型態  short b; //變數 b 為短整數型態  int c; //變數 c 為整數型態  long d; //變數 d 為長整數型態  long long e; //變數 e 為 long long 型態  float f; //變數 f 為浮點數型態  double g; //變數 g 為倍準浮點數型態  long double h; //變數 h 為 long double 型態 |

C 語言的[基本資料型態](http://pydoing.blogspot.com/2010/06/c-datatype.html)可分為三大類，分別是字元 (character) 、整數 (integer) 及浮點數 (floating-point number) ，有八種，使用六個關鍵字，如下列表

|  |  |  |  |
| --- | --- | --- | --- |
| **識別字** | **種類** | **佔用位元數** | **範圍** |
| char | 字元 | 8 |  |
| short | 整數 | 16 | -32768 to 32767 |
| int | 整數 | 16 或 32 ，實際視機器而定 | 實際視機器而定 |
| long | 整數 | 32 | −2,147,483,648 to 2,147,483,647 |
| long long | 整數 | 64 | 實際視機器而定 |
| float | 浮點數 | 32 |  |
| double | 浮點數 | 64 |  |
| long double | 浮點數 | 128 |  |

除了字元型態，整數及浮點數都有數種，主要由於不同型態佔用的記憶體空間不同，因此不同整數或浮點數型態可儲存的最大或最小值也有所不同，這樣一來，妥善規劃程式所用的資料型態可以更有效的利用記憶體空間。

宣告相同型態的多個變數也可以寫在同一列，例如簡單例子中的

第 5 行 int month, day, hour, min;

第 6 行 char \*name;

這是宣告字元型態的[指標 (pointer)](http://pydoing.blogspot.com/2010/06/c-pointer.html) 變數，所謂的指標是 C 語言的[衍生資料型態](http://pydoing.blogspot.com/2010/06/c-derivedtype.html)之一，用來存放指向某個變數或數值的記憶體位址。宣告指標變數，需要在變數名稱前加上星號 \* 。

[變數 (variable)](http://pydoing.blogspot.com/2010/06/c-identifier.html)，程式碼的第 8 行到第 12 行

將數值 (value) 給變數，程式語言中稱為[指派運算 (assignment operation)](http://pydoing.blogspot.com/2010/06/c-assignment.html) ，其為運算的一種，需要使用等號 = 當作[運算子 (operator)](http://pydoing.blogspot.com/2010/06/c-operator.html) ，因為這過程通常是將表示[常數 (constant)](http://pydoing.blogspot.com/2010/06/c-constant.html) 的數值放到變數所佔用的記憶體空間中。

C 語言可表示為常數的數值有四類，分別是字元 (character) 、字串 (string) 、整數 (integer) 及浮點數 (floating-point number) 。由於 C 語言直接支援 [ASCII](http://zh.wikipedia.org/zh-tw/Ascii) 編碼，所以字元型態的常數皆為表示 ASCII 編碼的符號，凡是用單引號 ' ' 圍住的單一符號皆為字元的常數。例如以下都是字元型態的常數

|  |
| --- |
| 'a' // 小寫英文字母  'H' // 大寫英文字母  '%' // 百分比符號  ' ' // 空格  '[' // 左中括弧  '=' // 等號 |

有一些不可見字元，例如換行、響鈴等稱為跳脫序列 (escape sequence) ，這是用反斜線加上某些字母組成的，如下為跳脫序列的完整列表

|  |  |
| --- | --- |
| **跳脫序列的字元** | **功能** |
| \a | 響鈴 |
| \b | 倒退鍵 |
| \f | 跳頁 |
| \n | 印出新列 |
| \r | 歸位符號 |
| \t | tab 鍵 |
| \v | 垂直定位符號 |
| \\ | 印出反斜線 |
| \? | 印出問號 |
| \' | 印出單引號 |
| \" | 印出雙引號 |

所謂的字串是指用雙引號 " " 圍住的連續字元，以下為字串常數的例子

|  |
| --- |
| "A bird in the hand is worth two in the bush."  "Fine words butter no parsnips."  "God helps those who help themselves."  "You cannot burn the candle at both ends."  "While there's life, there's hope."  "The early bird gets the worm." |

簡單例子中的

第 12 行 name = "kaiching";

由於已經在第 6 行以 char \*name; 宣告 name 為指向字元的[指標 (pointer)](http://pydoing.blogspot.com/2010/06/c-pointer.html)

因此 name 中會存放 "kaiching" 中第 1 個字元 'k' 的記憶體位址。事實上，字串是屬於字元陣列，[陣列 (array)](http://pydoing.blogspot.com/2010/06/c-array.html) 為 C 語言的[衍生資料型態](http://pydoing.blogspot.com/2010/06/c-derivedtype.html)之一，其為相同資料型態的資料體，這意思是說，陣列中可以有很多個相同資料型態的元素。因此將 "kaiching" 指派給變數 name ，雖然實際上 name 中所存放的是字元 'k' 的記憶體位址，但往後用 name 便可直接存取整個 "kaiching" 字串。  
  
  
至於整數及浮點數的常數很容易判別，凡是帶有小數點的數字皆為浮點數，不帶有小數點的數字皆為整數，如

|  |
| --- |
| 0.0065 // 浮點數 0.006500  21.35 // 浮點數 21.350000  6321. // 浮點數 6321.000000  6321 // 整數 6321  55 // 整數 55  0 // 整數 0 |

因此簡單例子中的變數 month 、 day 、 hour 、 min 、name 經過 8 到 12 行的指派運算，可視為各變數中儲存 7 、 27 、 9 、 45 、 "kaiching" 等常數數值。

呼叫 (call) [函數 (function)](http://pydoing.blogspot.com/2010/06/c-function.html) ，程式碼的第 14 行到第 16 行

函數 [printf()](http://pydoing.blogspot.com/2010/07/c-printf.html) 為[標準函數庫](http://pydoing.blogspot.com/2010/07/c-std.html)中 [stdio.h](http://pydoing.blogspot.com/2010/07/c-stdio.html) 所定義的格式化輸出函數，可將格式化字串傳送到標準輸出裝置上，標準輸出裝置通常是指螢幕。所謂的格式化字串是指含有轉換字符的字串，轉換字符如 printf() 第一個參數，也就是格式化字串中的 %d 及 %s 。

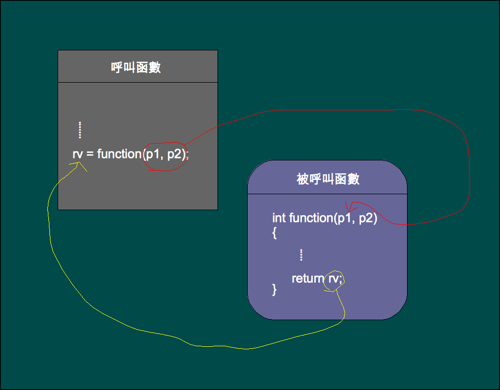
函數 printf() 可用的轉換字符如下列表

|  |  |  |
| --- | --- | --- |
| %d, %i | 有正負號的十進位整數 | int |
| %u | 無正負號的十進位整數 | int |
| %o | 無正負號的八進位整數 | int |
| %x, %X | 無正負號的十六進位整數 | int |
| %c | 字元 | char |
| %s | 字串 | char \* |
| %f | 浮點數 | double |
| %p | 記憶體位址的編碼 | void \* |
| %% | 百分比符號 | % |

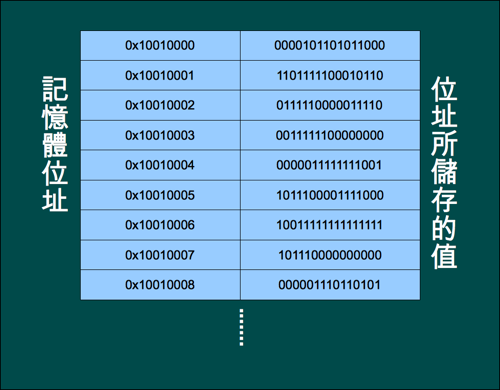
所以 printf() 在第一個參數後依轉換字符的數列接上引數，每個引數都用逗號 , 隔開。第 14 行及第 15 行的格式化字串各有兩個 %d ，所以後面接相對應整數型態的變數，第十六行的格式化字串有一個 %s ，其後接一個字串，由於字串是個比較複雜的字元陣列，所以這裡是個指向字串的[指標](http://pydoing.blogspot.com/2010/06/c-pointer.html)變數。

程式中使用已經寫好的函數，不論是標準函數庫中的函數或是自行定義的函數，使用函數都稱之為呼叫。若所呼叫的函數有定義參數，呼叫時便須提供型態相符的引數，這裡須注意，很多文獻都有用到參數或引數的說法，如英文原文參數為 parameter ，引數為 argument ，然而實際上參數是函數定義所用的詞，引數則是呼叫函數所用的詞，也有人用 formal parameter 及 actual argument 區分兩者。這裡我們提醒引數是指什麼，但我們傾向只使用參數一詞。

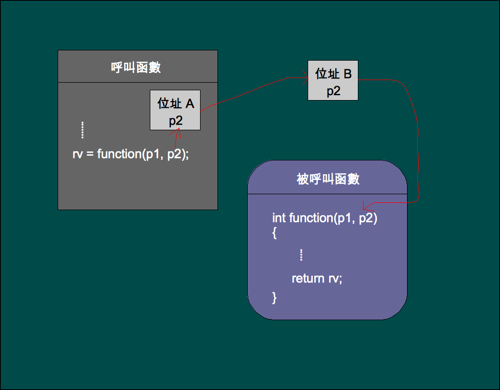
函數與函數之間靠參數與回傳值來傳遞訊息，呼叫函數傳遞參數給被呼叫函數，被呼叫函數則是用回傳值給呼叫函數，如下圖



由於 C 語言是採用傳值呼叫 (call by value) ，因此呼叫函數時所用參數是傳遞該參數數值的副本，怎麼說呢？我們需要了解程式中所用的每一個變數都具有記憶體位址，因此每個變數都具有該變數所儲存的值，以及該變數的記憶體位址



雖然資料依型態所佔用的記憶體空間（位址）多寡不同，然而這方面編譯器會利用類似表格的方式，記錄每個變數的起始位址。所謂的副本就是將原變數的值拷貝到另一個記憶體空間



因此被呼叫函數所具有的參數為原呼叫函數拷貝的副本，被呼叫函數並不能夠直接修改原呼叫函數所具有的變數值。

此外須留意 C 語言只有傳值呼叫，坊間有些傳參考呼叫 (call by reference) 或是傳址呼叫 (call by address) 的說法。前者，也就是傳參考呼叫，其為 C 之後演生出的 C++ 與 JAVA 才有的呼叫方式，但很多時候 C 與 C++ 籠統的合稱 C 語言，因此有 C 語言具有傳參考呼叫的說法。至於後者，也就是傳址呼叫的說法，嚴格說這樣的講法並不對。

會有傳址呼叫的說法是因為 C 語言可以用指標指向變數的記憶體位址，然而指標所儲存的只是記憶體位址的編碼值，甚至可以對指標進行某些程度的算術運算，實際上指標所儲存記憶體位址的編碼值仍是值。如果強行用傳址呼叫來理解 C 語言函數間參數的傳遞，使用雙重指標的部份容易兜不攏，便會對指標有很難、不易掌握的印象。

回傳值 (return value) 的部份，下程式碼的第 18 行

函數可以有回傳值，也可以沒有回傳值，我們的簡單例子在第 3 行

|  |
| --- |
| int main(void) |

已經將函數 main() 定義成回傳一個 int 型態的整數值，所以第 18 行

|  |
| --- |
| return 0; |

便是回傳 0 給叫用這個程式的作業系統。若是成功回傳整數 0 ，表示這個程式順利執行結束，若非，也就是說沒有回傳 0 給作業系統，就表示這個程式並沒有順利執行結束。

return 陳述也可以用運算式，所謂的[運算式](http://pydoing.blogspot.com/2010/06/c-expression.html)是由運算元與[運算子](http://pydoing.blogspot.com/2010/06/c-operator.html)所組成的，運算元可以是常數或是其他變數，運算子如同我們紙筆記算所用的計算符號，例如假、減、乘、除在 C 語言利用如下的符號

|  |  |  |
| --- | --- | --- |
| **運算子** | **功能** | **範例** |
| + | 加 | a + b |
| - | 減 | a - b |
| \* | 乘 | a \* b |
| / | 除 | a / b |

實際可能的程式碼如

|  |
| --- |
| a + b; // 計算兩個變數 a 、 b 相加  a - 8; // 計算變數 a 減掉 8  7 \* b; // 計算 7 乘上變數 b  7 / 8; // 計算 7 除以 8 |

若是 return 後直接加分號，沒有任何變數、常數或運算式，代表程式控制權移轉回原呼叫函數。

## 程式碼的編排

我們重新觀察這個程式，函數 main() 中每一行程式碼的結尾都帶分號，因為每一行以分號結尾的程式碼都稱為陳述，[函數 (function)](http://pydoing.blogspot.com/2010/06/c-function.html) 是 C 程式執行的單位，而陳述 (statement) 是函數裡程式碼執行的單位。

C 語言為自由格式的語言，這意思是說用 C 語言寫程式，程式碼可以自由編排，但有些地方必須照既定的方式撰寫，例如函數名稱後必須接帶有參數列的小括弧，底下函數內容必須用大括弧圍起來，以下程式大括弧放的位置不對，所以編譯時會發生錯誤；陳述必須以分號作為結束，漏了分號編譯時也會發生錯誤

大體上前兩個例子都被稱為語法錯誤 (syntax error) ，由於原始碼中帶有讓編譯器無法解析句型的內容，所以編譯會產生錯誤，也不會產生可執行檔。

其餘只要合乎語法，程式碼可以自由編排，例如，縮排不是必要的，合乎分號作陳述的結尾，分行也不是必要的，然而這樣一來，便不容易閱讀程式碼，因此我們傾向提供容易閱讀，並且加上適當註解的程式作為範例。

### 第二章 來寫個加法器

C 語言的加法運算子跟我們一般所用的加法符號一樣，都是 +

|  |  |  |
| --- | --- | --- |
| **運算子** | **功能** | **範例** |
| + | 加 | a + b |

程式如下

[?](https://pydoing.blogspot.com/2010/08/c-ch0201.html)

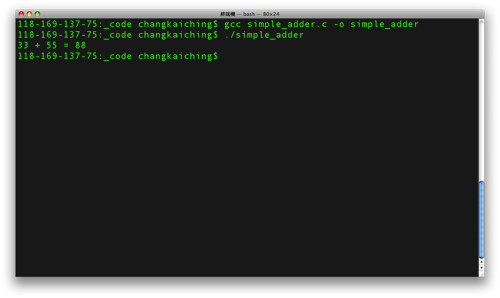
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #include <stdio.h>    int main(void)  {  int a = 33;  int b = 55;  int result = a + b;    printf("這個程式計算兩個整數的和....\n");  printf("%d + %d = %d\n", a, b, result);    return 0;  } |

從第 5 行到第 7 行，我們分別宣告了三個變數，同時直接指派 33 給 a ， 55 給 b ， a + b 的結果給 result

|  |  |
| --- | --- |
| 5  6  7 | int a = 33;  int b = 55;  int result = a + b; |

因此變數 a 中就儲存了整數 33 ，變數 b 中也儲存了整數 55 ，而變數 result 得到的是 a + b 的結果，也就是說， result 中會儲存整數 88 。  
  
  
我們在第 9 行呼叫函數 printf() 將三個變數值以加法算式印出

|  |  |
| --- | --- |
| 9 | printf("%d + %d = %d\n", a, b, result); |

編譯然後執行，不意外的，我們會在螢幕上看到 33 + 55 = 88  
  
  
這樣加法器全部的輸入都得預先在程式碼中寫好，嗯，我們來寫個接受使用者輸入的版本吧！  
  
C 語言[標準函數庫](http://pydoing.blogspot.com/2010/07/c-std.html) [stdio.h](http://pydoing.blogspot.com/2010/07/c-stdio.html) 中的函數 [printf()](http://pydoing.blogspot.com/2010/07/c-printf.html) 用為格式化輸出，另外有個函數 [scanf()](http://pydoing.blogspot.com/2010/07/c-scanf.html) 用為格式化輸入，其中轉換字符與 printf() 大致相同。  
  
我們寫的接受使用者輸入的加法器版本如下

[?](https://pydoing.blogspot.com/2010/08/c-ch0201.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #include <stdio.h>    int main(void)  {  int a, b, result; //宣告需要用到的變數    printf("這個程式計算兩個整數的和....\n");  printf("請輸入第一個整數： "); //提示使用者輸入的文字  scanf("%d", &a);  printf("請輸入第二個整數： "); //提示使用者輸入的文字  scanf("%d", &b);  result = a + b; //計算結果    printf("結果如下\n");  printf("%d + %d = %d\n", a, b, result);    return 0;  } |

第 5 行，我們將所用到的變數放在一起宣告

|  |  |
| --- | --- |
| 5 | int a, b, result; //宣告需要用到的變數 |

這是因為 a 、 b 需要等待使用者輸入，而 result 要等到 a 、 b 都有數值之後才能進行第 12 行的計算

|  |  |
| --- | --- |
| 12 | result = a + b; //計算結果 |

如果變數只有宣告而沒有給值，那麼變數中所具有的值會是記憶體位址的殘值，因此，程式要得到預期的結果，宣告變數後就要在適當的地方指派數值給變數，或是宣告後直接給變數初值。

第 8 行及第 10 行為提示使用者輸入的文字，會額外印出這樣的訊息文字，是因為輸入函數如 scanf() 會等待使用者輸入，直到按下 return 或 Enter 鍵才會結束執行，提示訊息的目的就是告訴使用者應該怎麼跟程式互動。

|  |  |
| --- | --- |
| 8 | printf("請輸入第一個整數： "); //提示使用者輸入的文字 |
| 10 | printf("請輸入第二個整數： "); //提示使用者輸入的文字 |

第 9 行及第 11 行為呼叫函數 scanf()

|  |  |
| --- | --- |
| 9 | scanf("%d", &a); |
| 11 | scanf("%d", &b); |

函數 scanf() 可接受多個參數，其中第一個必須是含有轉換字符的格式化字串，後面則依轉換字符的數量接型態相符的指標變數，也就是所要儲存輸入值變數的記憶體位址。

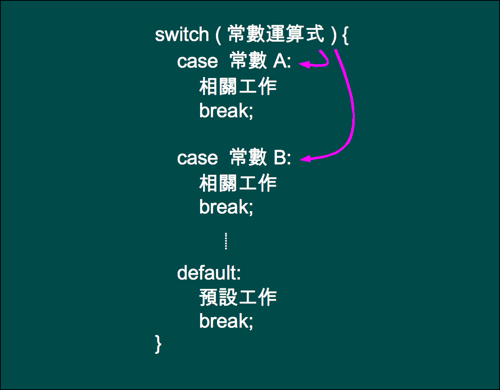
此例中，我們用變數 a 與 b 分別記錄使用者輸入的第一個及第二個整數，利用取址運算子 & 可取得該變數的記憶體位址，因此取址運算子與變數名稱連用便是指標。

### 第二章 將加法器擴展成四則運算器

加法器很簡單，純粹處理加法也很實用，但我們總是希望程式多點功能，直接可以處理所有加減乘除等四則運算的計算工作。有需求，程式就可以隨之推出改進的版本，想一想，該怎麼設計新的程式版本呢？  
  
  
因為我們已經寫出了加法器，依序可以接受使用者輸入兩個整數，那麼我們中間多輸入一個表示加減乘除的字元，然後設計一個機制，讓程式可以判斷是加減乘除的哪個符號不就可以了嗎？依照這個想法寫出的程式如下

[?](https://pydoing.blogspot.com/2010/08/c-ch0202.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | #include <stdio.h>    int main(void)  {  int a, b; //宣告需要用到的變數  char opt;    printf("這個程式會依符號計算兩個整數的和、差、積、商....\n");  printf("請輸入第一個整數： "); //提示使用者輸入的文字  scanf("%d", &a);  printf("請輸入運算符號： "); //提示使用者輸入的文字  scanf("%s", &opt);  printf("請輸入第二個整數： "); //提示使用者輸入的文字  scanf("%d", &b);    //以下會依所輸入的運算子計算並印出結果  printf("結果如下： \n");  switch (opt) {  case '+':  printf("%d + %d = %d\n", a, b, a + b);  break;    case '-':  printf("%d - %d = %d\n", a, b, a - b);  break;    case '\*':  printf("%d \* %d = %d\n", a, b, a \* b);  break;    case '/':  printf("%d / %d = %d\n", a, b, a / b);  break;    default:  printf("輸入錯誤！\n");  break;  }    return 0;  } |

我們利用了 [switch-case](http://pydoing.blogspot.com/2010/06/c-switch.html) 陳述作為分辨[運算子](http://pydoing.blogspot.com/2010/06/c-operator.html)的機制，程式的第 18 行到第 38 行便是 switch-case 陳述的部份。 switch-case 為複合陳述，用法圖解如下  


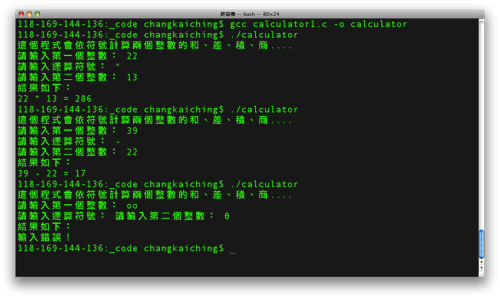
switch-case 為複合陳述，大括弧後不需要以分號結尾。

switch 為[關鍵字](http://pydoing.blogspot.com/2010/06/c-keyword.html)之一，其後接一個小括弧，小括弧內必須是[常數](http://pydoing.blogspot.com/2010/06/c-constant.html)，或是運算結果為常數的運算式。然後接一對大括弧，大括弧裡放多個 case 及後最後一個 default 。 case 及 default 也都是關鍵字， case 的作用就是對情況進行判斷，所謂的情況就是 case 後一個空格，然後接的常數或常數運算式，然後再接一個冒號。  
  
  
如果 switch 的常數與任何一個 case 的常數相同，該 case 底下的陳述就會被執行。每個 case 最後會用一個 break 陳述結尾，這是要中斷 switch 繼續做 case 條件檢查之用。 break 亦為關鍵字之一，用於中斷迴圈或跳出 switch 陳述。 deault 可以看成是預設的 case ，假如以上的 case 都不符合，就會執行 default 底下的陳述。  
  
  
如果沒有用 break 陳述， switch 陳述就會一直檢查 case 情況到最後，因此加入 break 陳述可以早一步離開 switch 。而 我們把 default 放在整個 switch 陳述的最後，也就是右大括弧之前，其實這裡可以不加 break 陳述，因為上面每個 case 都不符合才會執行 default 的部份，之所以也放 break 陳述的原因很簡單，只是讓程式碼看起來對齊一致，既然每個 case 都有 break ，那 default 也就加入 break 囉！  
  
  
另外注意第 6 行，我們宣告一個字元變數 opt 來放使用者輸入的運算符號

|  |  |
| --- | --- |
| 6 | char opt; |

鍵盤支援的加減乘除運算子分別為 + 、 - 、 \* 、 / ，因此使用者需要輸入相對應的符號，我們利用第 12 行的函數 scanf() 取得 使用者輸入的運算符號

|  |  |
| --- | --- |
| 12 | scanf("%s", &opt); |

這裡的轉換字符是 %s ，本來是接收字串，字元的轉換字符為 %c ，我們不是接收一個字元就可以了嗎？會用 %s 而不是 %c 的原因其實也很簡單，因為當我們按下 return 或 Enter 鍵的時候就等於我們輸入了新行字元，因此當上一個數字輸入完按下 return 或 Enter 鍵的時候，輸入緩衝區多儲存了一個新行字元，因此若是我們這裡採用 %c 的方式，會使函數 scanf() 直接從緩衝區讀取上一次按下的新行字元，而造成無法輸入運算符號的問題。  
  
  
另外，我們少宣告一個 result 變數，這是因為我們呼叫 printf() 的時候，直接把運算式當作參數代入。由於運算式會計算出一個數值，因此通常可以直接用型態符合的運算式當成參數。  
  
  
我們來編譯執行看看吧！  
  
  
  
嗯，若是若是沒有輸入整數，程式會發生錯誤。我們先把這個問題擱在一旁，先來想想有沒有其他的設計方式。

### 第二章 新版本的四則運算器

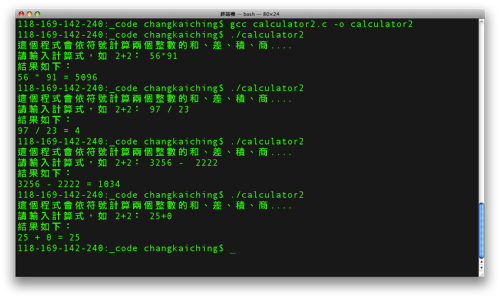
這樣的輸入方式也沒什麼不好哇！就輸入一個項目按一次 return 或 Enter 啊！但是我們手邊小計算機的操作模式確非如此，依數字、運算符號、數字的順序，全部輸入完按一次等號就行了，不是嗎？  
  
  
  
我們能不能規劃像手邊小計算機操作模式的程式呢？畢竟比較跟我們日常的習慣比較接近。其實 [scanf()](http://pydoing.blogspot.com/2010/07/c-scanf.html) 的格式化字串可以用多個轉換字符，就跟函數 [printf()](http://pydoing.blogspot.com/2010/07/c-printf.html) 類似，因此直接要求使用者輸入運算式是可行的。  
  
  
我們提供的新版本如下

[?](https://pydoing.blogspot.com/2010/08/c-ch0203.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | #include <stdio.h>    int main(void)  {  int a, b; //宣告需要用到的變數  char opt;    printf("這個程式會依符號計算兩個整數的和、差、積、商....\n");  printf("請輸入計算式，如 2+2： "); //提示使用者輸入的文字  scanf("%d %c %d", &a, &opt, &b);    //以下會依運算子計算並印出結果  printf("結果如下： \n");  switch (opt) {  case '+':  printf("%d + %d = %d\n", a, b, a + b);  break;    case '-':  printf("%d - %d = %d\n", a, b, a - b);  break;    case '\*':  printf("%d \* %d = %d\n", a, b, a \* b);  break;    case '/':  printf("%d / %d = %d\n", a, b, a / b);  break;    default:  printf("輸入錯誤！\n");  break;  }    return 0;  } |

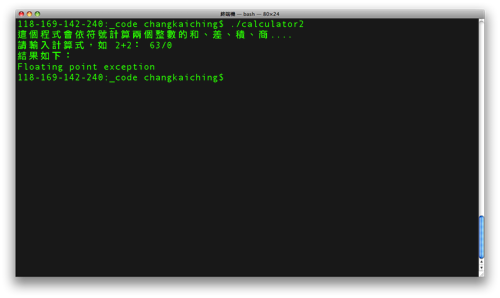
改變的地方就是把原先需要 6 行的輸入部份，包含 3 行的提示訊息，使用者也需要連續輸入然後按 3 次 return 或 Enter 鍵，現在改成兩行，就在第 9 行及第 10 行

|  |  |
| --- | --- |
| 9  10 | printf("請輸入計算式，如 2+2： "); //提示使用者輸入的文字  scanf("%d %c %d", &a, &opt, &b); |

函數 scanf() 直接接受三項輸入值，注意，我們在格式化字串中每一個轉換字符都用空格分開。  
  
  
來編譯執行看看吧！  
  
  
  
只需要一次輸入，似乎好用好多耶！的確，寫出相同功能的程式有很多種不同的設計方式，每一種方式都有優點跟缺點，我們需要學習的是如何在開發軟體時，學會選擇適合開發目標的方法，這在學習程式設計會是一項重要的課題。

就像有些方法省時，但是浪費記憶體空間，有些方法費時，卻能節省記憶體空間

### 第二章 小心！除以 0

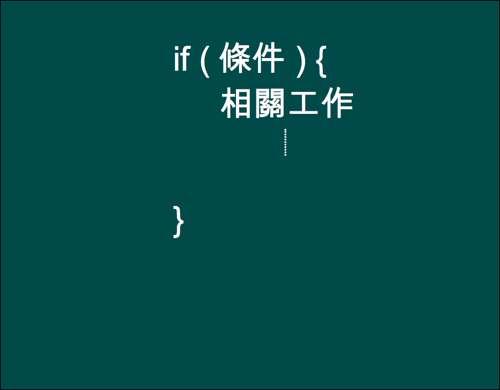
如果在四則運算器中將某個整數除以 0 ，那會發生什麼結果呢？  
  
  
  
  
執行後顯示 Floating point exception 的錯誤，然後中斷程式的執行，現在許多已經開發成熟的作業系統大多會這麼做，可以片面單獨終止某一程式，避免整個系統崩潰。除以 0 對電腦而言是致命錯誤，就是說發生除以 0 的情況，程式變無法繼續運作，導致程式當掉，或是整個系統當機。  
  
  
數學上我們學到 0 不能當分母的規則，這是因為 0 當除數會使算式變得沒有意義，在電腦中除以 0 會無法運算出結果，所以我們得小心使用可能會發生這樣結果的程式，嗯，小心使用固然是個辦法啦！但我們總是不能預期使用者如何使用，因此在程式中加入防範的機制，便是另一個可行的辦法。  
  
  
我們提供一個簡單的防範機制，也就是測試使用者輸入的除數是否為 0 ，如果是 0 ，就把除數改為 1 。程式碼如下

[?](https://pydoing.blogspot.com/2010/08/c-ch0204.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | #include <stdio.h>    int main(void)  {  int a, b; //宣告需要用到的變數  char opt;    printf("這個程式會依符號計算兩個整數的和、差、積、商....\n");  printf("請輸入計算式，如 2+2： "); //提示使用者輸入的文字  scanf("%d %c %d", &a, &opt, &b);    //如果使用者輸入的除數為 0 ，將除數改為 1  if (opt == '/' && b == 0) {  b = 1;  }    //以下會依運算子計算並印出結果  printf("結果如下： \n");  switch (opt) {  case '+':  printf("%d + %d = %d\n", a, b, a + b);  break;    case '-':  printf("%d - %d = %d\n", a, b, a - b);  break;    case '\*':  printf("%d \* %d = %d\n", a, b, a \* b);  break;    case '/':  printf("%d / %d = %d\n", a, b, a / b);  break;    default:  printf("輸入錯誤！\n");  break;  }    return 0;  } |

防範機制設置在第 13 行到第 15 行

|  |  |
| --- | --- |
| 12  13  14  15 | //如果使用者輸入的除數為 0 ，將除數改為 1  if (b == 0) {  b = 1;  } |

第 12 行的註釋說明這段程式碼的作用。這裡引入兩個新觀念，首先是 [if](http://pydoing.blogspot.com/2010/06/c-if.html) 陳述，用法圖解如下  


if 後可接複合陳述，大括弧後不需要以分號結尾。

if 為[關鍵字](http://pydoing.blogspot.com/2010/06/c-keyword.html)之一，主要用為條件選擇，提供程式執行分流之用。 if 後接小括弧，小括弧中即為條件，所謂的條件其實就是[運算式](http://pydoing.blogspot.com/2010/06/c-expression.html)，運算結果為非 0 的值，就會執行 if 底下大括弧的陳述，若運算結果為 0 ，程式就會跳過大括弧，繼續執行大括弧後的程式。

條件的用語是個邏輯概念，若真，做 A 工作，若假，做 B 工作。在 C 語言中，所有非 0 的值皆為真，包括負數，只有 0 是假。

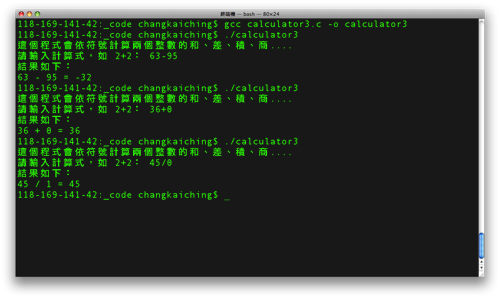
另一個新觀念就是我們用了新的運算子，條件是 opt == '/' && b == 0 ，這個運算式分成兩部份，利用連續兩個 && 連接，我們先來看看 && 之前的 opt == '/' 與之後的 b == 0 。  
  
opt == '/' 的意思就是測試 opt 是否等於 '/' ，而 b == 0 的意思則是測試 b 是否等於 0 。測試結果若相等，計算結果為 1 ，若測試後不相等，，計算結果就為 0 。連續兩個等號 == 為測試相等的運算子，這是 C 語言六個[關係運算子](http://pydoing.blogspot.com/2010/06/c-relational.html)之一

|  |  |  |
| --- | --- | --- |
| **運算子** | **功能** | **範例** |
| < | 小於 | a < b |
| <= | 小於等於 | a <= b |
| > | 大於 | a > b |
| <= | 大於等於 | a <= b |
| == | 相等 | a == b |
| != | 不等 | a != b |

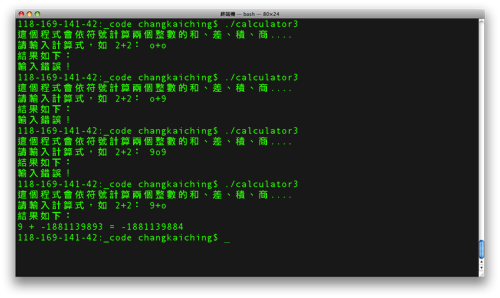
C 語言中，單一個等號 = 用作指派，而連續兩個等號用作 == 測試相等。

連續兩個 && 則是邏輯運算子中的「且」，就是說兩個運算元都為真，邏輯「且」才為真。 C 語言共有三個[邏輯運算子](http://pydoing.blogspot.com/2010/06/c-logical.html)

|  |  |  |
| --- | --- | --- |
| **運算子** | **功能** | **範例** |
| ! | 非 | !a |
| && | 且 | a && b |
| || | 或 | a || b |

所以這個條件 opt == '/' && b == 0 算是比較複雜的運算式，程式會先計算 opt == '/' 的值，然後碰到 && ，這時候，若 opt == '/' 為假，也就是計算結果為 0 ，整個條件運算式的結果就會為假，程式直接跳過 if 底下的大括弧。除非 opt == '/' 為真，程式才會繼續測試 b 是否等於 0 ，若 b 等於 0 ， if 底下的大括弧才會被執行。  
  
C 語言這項特性被稱為 short-circuit evaluation ，中文可稱之為短路徑計算，若妥善利用邏輯運算子與這項特性，對於提升程式效能是很有幫助的。  
  
  
我們來編譯執行看看吧！  


### 第二章 解決輸入型態不相符的問題

我們在設計四則運算器的時候，曾經發現若是函數 [scanf()](http://pydoing.blogspot.com/2010/07/c-scanf.html) 的格式化字串要求 %d ，卻輸入其他字元，造成程式發生錯誤，跳過輸入運算符號的部份，結果顯示「輸入錯誤！」。  
  
  
  
這是我們對輸入第一個整數進行過的測試，現在我們拿四則運算器的例子重新進行測試，分別輸入 o+o 、 o+9 、 9o9 、 9+o   
  
  
  
嗯，前三者都讓 [switch](http://pydoing.blogspot.com/2010/07/c-switch.html) 顯示 default 的「輸入錯誤！」，雖然我們希望 switch 的運作是跟輸運運算符號直接相關，不過輸入第一個整數（運算元）時，也導致 switch 選擇 default ，這樣的運作方式倒是可以接受，當成輸入第一個整數的錯誤防範機制。  
  
  
但是第四個測試卻出現了很奇怪的整數，而且程式能繼續計算，結果還會計算列印出來，這自然不會是我們想要的運作方式，我們總是希望輸入型態不符的話，程式能直接顯示「輸入錯誤！」。類似這樣的錯誤被稱為語意錯誤，就是程式能順利執行，但是結果非預期。  
  
  
語意錯誤跟語法錯誤都是很常發生的錯誤，後者無法順利編譯，同時編譯器會提供錯誤訊息，然而前者，也就是語意錯誤卻能順利編譯，程式能成功執行，但是會得到非預期的結果。通常語意錯誤需要對程式進行反覆測試，例如在與使用者互動的程式中，利用各種不同的輸入來查看執行結果是否正確。  
  
  
那我們該如何修正這個錯誤呢？由於函數 scanf() 會回傳一個整數，表示所輸入的型態相符的輸入值。例如要輸入兩個整數，若使用者正確輸入兩個整數， scanf() 就回傳 2 ，同樣的，我們的例子要求輸入一個整數、一個字元及另一個整數，若是依順序輸入型態相符的參數， scanf() 就會回傳整數 3 。   
  
  
因此我們宣告另一個變數 t 來接收 scanf() 的回傳值，若回傳值不等於 3 ，就把運算符號改為非預期的字元 '!' 。修正後的版本如下

[?](https://pydoing.blogspot.com/2010/08/c-ch0205.html)

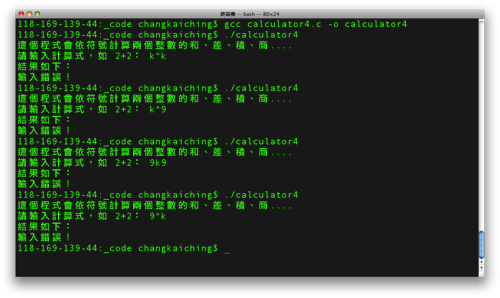
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | #include <stdio.h>    int main(void)  {  int a, b, t; //宣告需要用到的變數  char opt;    printf("這個程式會依符號計算兩個整數的和、差、積、商....\n");  printf("請輸入計算式，如 2+2： "); //提示使用者輸入的文字  t = scanf("%d %c %d", &a, &opt, &b);    //如果使用者輸入的除數為 0 ，將除數改為 1  if (opt == '/' && b == 0) {  b = 1;  }    //如果使用者輸入非整數，便將 opt 改成不合法的運算符號  if (t != 3) {  opt = '!';  }    //以下會依運算子計算並印出結果  printf("結果如下： \n");  switch (opt) {  case '+':  printf("%d + %d = %d\n", a, b, a + b);  break;    case '-':  printf("%d - %d = %d\n", a, b, a - b);  break;    case '\*':  printf("%d \* %d = %d\n", a, b, a \* b);  break;    case '/':  printf("%d / %d = %d\n", a, b, a / b);  break;    default:  printf("輸入錯誤！\n");  break;  }    return 0;  } |

程式改變就是多宣告變數 t ，第 11 行變數 t 接受函數 scanf() 的參數

|  |  |
| --- | --- |
| 11 | t = scanf("%d %c %d", &a, &opt, &b); |

然後第 19 行到第 21 行，檢查 t 是否不等於 3 ，若真就將 opt 設為 '!'

|  |  |
| --- | --- |
| 18  19  20  21 | //如果使用者輸入非整數，便將 opt 改成不合法的運算符號  if (t != 3) {  opt = '!';  } |

重新編譯來測試看看吧！  
  
  
  
嗯，目前看來，的確解決了輸入型態不符的錯誤。

### 第二章 遞增遞減

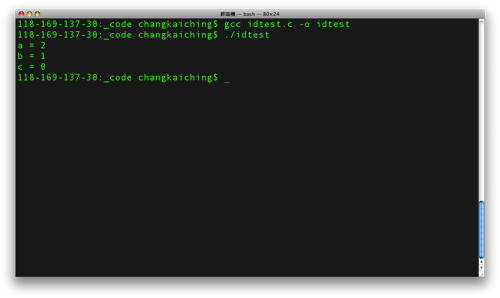
C 也有提供方便的[遞增運算子及遞減運算子](http://pydoing.blogspot.com/2010/06/c-increment.html)，通常用於[控制結構](http://pydoing.blogspot.com/2010/06/c-controlflow.html)中的的各種迴圈

|  |  |  |
| --- | --- | --- |
| **運算子** | **功能** | **範例** |
| ++ | 前遞增 | ++a |
| ++ | 後遞增 | a++ |
| -- | 前遞減 | --a |
| -- | 後遞減 | a-- |

遞增運算子為連續兩個加號，遞減運算子則是連續兩個減號，兩者都是單元運算子，直接施用在連接的運算元上。所謂的遞增運算就是對施用變數加 1 ，遞減就是減 1 ，這裡須注意，為了避免語意上的歧義或語法錯誤，通常我們會把遞增運算子及遞減運算子與施用變數直接連起來，然後與其他運算單位以空格分開。  
  
  
前遞增就是先將運算元遞增，然後才加入運算式計算，後遞增則是運算式先行計算，運算式計算完後才進行遞增，而前後遞減亦同，就是將遞增改成遞減。如下例

[?](https://pydoing.blogspot.com/2010/08/c-ch0206.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #include <stdio.h>    int main(void)  {  int a, b, c;    a = 0;  b = 0;  c = a++ - ++b + ++a;    printf("a = %d\n", a);  printf("b = %d\n", b);  printf("c = %d\n", c);    return 0;  } |

編譯後執行，如下  
  
  
  
計算集中在第 9 行

|  |  |
| --- | --- |
| 9 | c = a++ - ++b + ++a; |

變數 a 與變數 b 最初的值為 0 ， a 後遞增加上 b 前遞增再加上 a 前遞增，因此變數 c 等同下面的等式

c = 0 - 1 + 1;

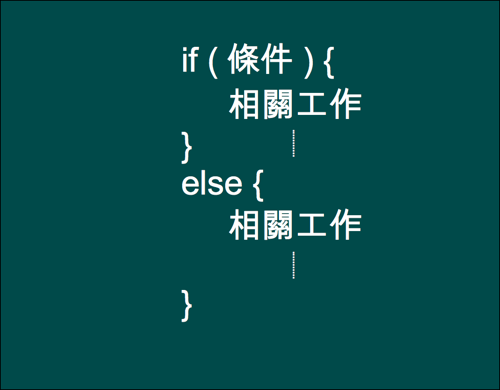
所以 c 等於 0 。 a 總共遞增了兩次， b 遞增了一次，因此 a 等於 2 ， b 等於 1 。  
  
  
我們提供另一個使用遞增運算子與遞減運算子的範例，執行過程中會跟使用者互動，詢問所施用變數目前的值，另外用一個變數累計答對次數，程式如下

[?](https://pydoing.blogspot.com/2010/08/c-ch0206.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68 | #include <stdio.h>    int main(void)  {  int target = 0;  int counter = 0;  int guess;    printf("target = %d\n", target);  printf("target++\n", target++);  printf("--target\n", --target);  printf("target++\n", target++);    printf("請輸入target現在是？ ");  scanf("%d", &guess);    if (target == guess) {  printf("答對囉！\n");  counter++;  }  else {  printf("答錯了!\n");  counter--;  }    printf("target--\n", target--);  printf("++target\n", ++target);  printf("--target\n", --target);  printf("++target\n", ++target);    printf("請輸入target現在是？ ");  scanf("%d", &guess);    if (target == guess) {  printf("答對囉！\n");  counter++;  }  else {  printf("答錯了!\n");  counter--;  }    printf("target++\n", target++);  printf("--target\n", --target);  printf("++target\n", ++target);  printf("--target\n", --target);    printf("請輸入target最後是？ ");  scanf("%d", &guess);    if (target == guess) {  printf("答對囉！\n");  counter++;  }  else {  printf("答錯了!\n");  counter--;  }    if (counter > 0) {  printf("\n共答對 %d 次....\n", counter);  }  else {  printf("\n都沒有答對耶！再加油喔！\n");  }    return 0;  } |

請自行編譯執行這個範例，測試看看自己是否已經了解遞增、遞減的觀念。

### 第二章 指派

我們在上個例子中用到了 if-else 陳述，其結構解析如下  
  
  
  
  
else 為[關鍵字](http://pydoing.blogspot.com/2010/06/c-keyword.html)之一， if-else 陳述為單一 [if](http://pydoing.blogspot.com/2010/06/c-if.html) 陳述的延伸，提供若真則執行某某，若非執行某某的後者，也就是條件為真，執行 if 底下大括弧的程式，條件為假則執行 else 底下大括弧的陳述。  
  
  
利用 if-else 可以讓程式簡化，例如

|  |
| --- |
| if (a > b) {  printf("a > b\n");  }  else {  printf("a < b\n");  } |

這樣程式只需要做單一的條件判斷。沒有用 else 就得寫兩個 if ，程式必須做兩次條件判斷

|  |
| --- |
| if (a > b) {  printf("a < b\n");  }  if (a < b) {  printf("a > b\n");  } |

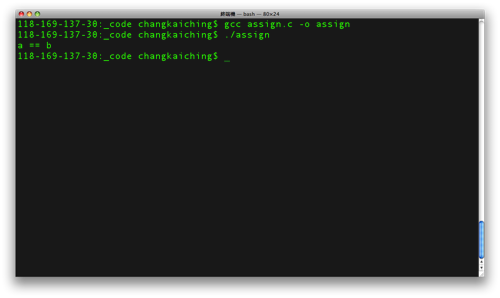
當程式發展規模越來越大的時候，差別就出現了，多餘的條件判斷會降低程式的效率。  
  
  
指派運算子，也就是單一個等號 = ，可以和其他的運算子連用，運算後同時指派，使程式碼更為簡潔

|  |  |  |
| --- | --- | --- |
| **運算子** | **功能** | **範例** |
| += | 相加同時指派 | a += b |
| -= | 相減同時指派 | a -= b |
| \*= | 相減同時指派 | a \*= b |
| /= | 相除同時指派 | a /= b |
| %= | 取餘數同時指派 | a %= b |

簡單說， a += b 就是把 a + b 的結果再指派給變數 a 。到目前為止，我們介紹過的算術運算子都可以和指派運算子連用。這種寫法在許多 C 程式都常出現，因此需要習慣這種表示方式，我們以一個例子示範如下

[?](https://pydoing.blogspot.com/2010/08/c-ch0207.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | #include <stdio.h>    int main(void)  {  int a = 10;  int b = 10;    a += 1;  b++;    a -= 1;  --b;    if (a == b) {  printf("a == b\n");  }  else {  printf("a != b\n");  }    return 0;  } |

編譯後執行，如下  
  
  
  
這個例子也說明

|  |
| --- |
| a += 1; |

與

|  |
| --- |
| a++; // 或++a; |

的計算結果是相同的。同樣的

|  |
| --- |
| a -= 1; |

與

|  |
| --- |
| a--; // 或--a; |

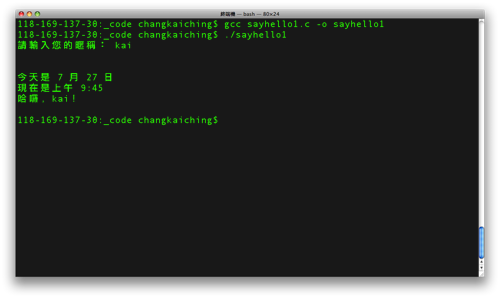
的計算結果也是相同的。

### 第二章 向使用者 Say Hello!

現在我們來寫個向使用者 Say Hello! 的程式吧！  
  
  
  
之前我們直接印出歡迎訊息，有關暱稱的部份使用預設的[字串](http://pydoing.blogspot.com/2010/06/c-string.html)，也就是 "kaiching" ，現在我們則是要利用函數 [scanf()](http://pydoing.blogspot.com/2010/07/c-scanf.html) 接受使用者輸入他們自己的暱稱。程式如下

[?](https://pydoing.blogspot.com/2010/08/c-ch0208.html)

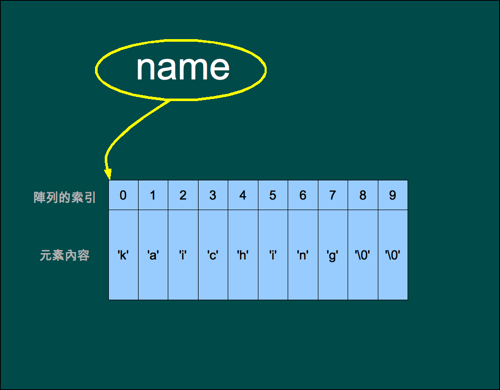
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #include <stdio.h>    int main(void)  {  int month, day, hour, min;  char name[10];    month = 7;  day = 27;  hour = 9;  min = 45;    printf("請輸入您的暱稱： ");  scanf("%s", name);    printf("\n\n今天是 %d 月 %d 日\n", month, day);  printf("現在是上午 %d:%d\n", hour, min);  printf("哈囉, %s！\n\n", name);    return 0;  } |

先來編譯執行看看吧！  
  
  
  
程式的第 6 行

|  |  |
| --- | --- |
| 6 | char name[10]; |

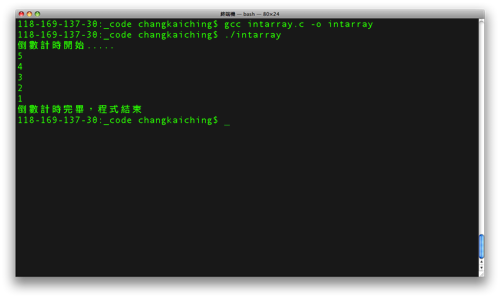
我們宣告了一個字元[陣列](http://pydoing.blogspot.com/2010/06/c-array.html)， C 語言中的陣列是同質的資料結構，意思是說陣列中可以有多筆相同型態的資料，因此字元陣列就是一群字元型態資料的集合體。如果直接指派字串給字元陣列，例如

|  |  |
| --- | --- |
| 6 | char name[10] = "kaiching"; |

此字元陣列中就會儲存字串 "kaiching" 的值，圖示如下  
  
  
  
陣列裡儲存的單筆資料稱為元素，我們可以利用陣列名稱加上中括弧，中括弧內為陣列元素的索引值來個別存取陣列元素。陣列名稱亦為指標，因此陣列名稱 name 就是指向 name[0] 的記憶體位址。   
  
  
這裡我們須注意一點，若是使用指向字元的指標，可以指向某個常數字串，因為使用常數字串，編譯器會直接給該常數字串在記憶體中保留空間，所以指向字元的指標指向常數字串，就等於指向該字串第一個元素的記憶體位址。然而這裡我們接受使用者輸入字串，需要先告訴編譯器要先保留多少空間出來，所以宣告陣列時，中括弧就得同時設置所宣告陣列的元素個數。  
  
  
上例中也可以看出，若是字串裡的字元數少於字元陣列所宣告的元素數量，字元陣列沒有被給值的元素都會填入 '\0' ，這是 ASCII 中的空字元，等同於整數 0 。  
  
  
除了字元陣列外，陣列可以宣告成任何的資料型態。例如，以下例子簡單示範整數陣列的使用

[?](https://pydoing.blogspot.com/2010/08/c-ch0208.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include <stdio.h>    int main(void)  {  int number[5] = {1, 2, 3, 4, 5};    printf("倒數計時開始.....\n");  printf("%d\n", number[4]);  printf("%d\n", number[3]);  printf("%d\n", number[2]);  printf("%d\n", number[1]);  printf("%d\n", number[0]);  printf("倒數計時完畢，程式結束\n");    return 0;  } |

編譯後執行，如下  
  
  
  
第 5 行

|  |  |
| --- | --- |
| 6 | int number[5] = {1, 2, 3, 4, 5}; |

這是宣告陣列同時指派初值的方式，指派陣列的值必須用大括弧圍起來。如果是字串可用這樣

|  |
| --- |
| char \*name = "kaiching"; |

也可以照字元陣列的方法，如

|  |
| --- |
| char name[] = {'k', 'a', 'i', 'c', 'h', 'i', 'n', 'g'}; |

附帶一提，若是陣列宣告同時指派初值，中括弧可留空白，編譯器會自動計算並且配置元素個數。

### 第二章 處理時間問題

Say Hello! 可以向不同使用者打招呼了，可是所顯示的是過去的時間，而非現在的時間。若是要取得現在電腦的時間，我們可以利用作業系統的的 date 指令，可以直接顯示現在的時間日期

[?](https://pydoing.blogspot.com/2010/08/c-ch0209.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #include <stdio.h>  #include <stdlib.h>    int main(void)  {  char name[10];    printf("請輸入您的暱稱： ");  scanf("%s", name);    printf("\n\n");  system("date");  printf("哈囉, %s！\n\n", name);    return 0;  } |

因為我們要使用作業系統的指令，這一方面要利用[標準函數庫](http://pydoing.blogspot.com/2010/07/c-std.html) [stdlib.h](http://pydoing.blogspot.com/2010/07/c-stdutility.html) 的函數 [system()](http://pydoing.blogspot.com/2010/07/c-system.html) ，所以第 2 行，我們要先引入 stdlib.h

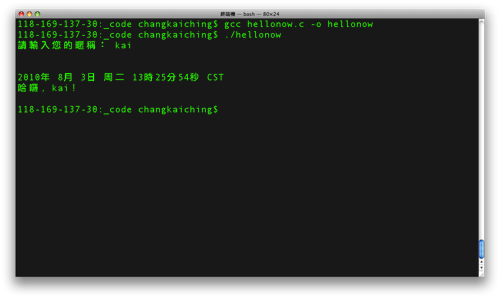
|  |  |
| --- | --- |
| 2 | #include <stdlib.h> |

然後在第 12 行呼叫函數 system()

|  |  |
| --- | --- |
| 12 | system("date"); |

函數 system() 需要用字串當參數，這個字串就是所需要的作業系統指令。在打招呼之前會先執行 date 指令，執行完 date 指令才會繼續執行接下來的打招呼。

UNIX-Like 的作業系統都可以直接用 date 指令，若是用 MS-Windows 系統，可能需要用到 date 及 time 兩個指令，其他作業系統請查詢該系統的手冊。

編譯後執行，如下  
  
  
  
我們繼續提供另一種顯示現在時間的方法，就是利用標準函數庫中 [time.h](http://pydoing.blogspot.com/2010/07/c-stdtime.html) 的函數 [time()](http://pydoing.blogspot.com/2010/07/c-time.html) ，這個函數會回傳自格林威治時間 1970 年 1 月 1 日到目前所經過的總秒數

[?](https://pydoing.blogspot.com/2010/08/c-ch0209.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | #include <stdio.h>  #include <time.h>    int main(void)  {  char name[10];  int s = (int) time(NULL);  int sec, min, hour;    s += 8 \* 60 \* 60;  sec = s % 60;  min = (s / 60) % 60;  hour = (s / 3600) % 24;    printf("請輸入您的暱稱： ");  scanf("%s", name);    printf("\n\n現在是%d點%d分%d秒\n", hour, min, sec);  printf("哈囉, %s！\n\n", name);    return 0;  }    /\* 《程式語言教學誌》的範例程式  <http://pydoing.blogspot.com/>  檔名：hellonowo.c  功能：顯示現在時間的打招呼程式  作者：張凱慶  時間：西元2010年7月 \*/ |

首先在第 2 行，我們引入 time.h

|  |  |
| --- | --- |
| 2 | #include <time.h> |

第 7 行有個 (int)

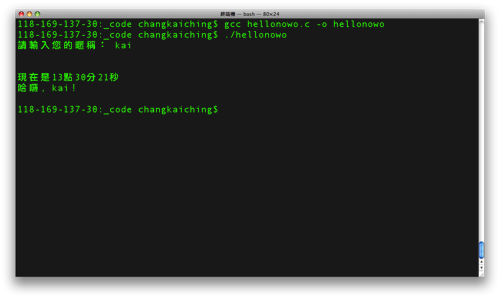
|  |  |
| --- | --- |
| 7 | int s = (int) time(NULL); |

這是種利用小括弧轉換型態的方式，這個小括弧被稱為 [cast 運算子](http://pydoing.blogspot.com/2010/06/c-cast.html)，用來強制型態轉換。因為函數 time() 回傳的是另外定義的算術型態 time\_t ，我們這裡做強制型態轉換的目的是為了避免編譯器出現型態不符的警告訊息。  
  
  
time() 需要一個指向型態 time\_t 的指標當作參數，由於我們直接接收回傳值，所以用 NULL 給 time() 當作參數。 NULL 在 C 語言中是一個常數值，表示空的指標。  
  
  
變數 s 得到的是自格林威治時間 1970 年 1 月 1 日到目前所經過的總秒數，由於格林威治時間與台灣時差八個小時，所以第 10 行，我們將 s 加上時差的秒數

|  |  |
| --- | --- |
| 10 | s += 8 \* 60 \* 60; |

接下來由於計算日期有點麻煩，所以我們暫時只計算出目前時間，也就是現在幾點幾秒幾分，這些計算放在第 11 行到第 13 行

|  |  |
| --- | --- |
| 11  12  13 | sec = s % 60;  min = (s / 60) % 60;  hour = (s / 3600) % 24; |

計算現在時間的秒數，就是把總秒數除以 60 的餘數。計算目前幾分也很簡單，總秒數除以 60 的商會是總分數，將總分數除以 60 取其餘數，就會得到目前幾分。至於計算目前幾點大致相同，就用總秒數除以 3600 的商就是總時數，總實數再除以 60 取其餘數就是目前幾點。  
  
  
來編譯執行看看吧！  


### 第二章 來玩猜數字吧

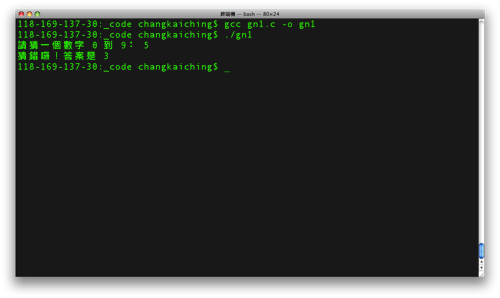
設計各種與使用者互動的程式中，最有趣的莫過與遊戲了，我們來寫個簡單的猜數字遊戲吧！

[?](https://pydoing.blogspot.com/2010/08/c-ch0210.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include <stdio.h>    int main(void)  {  int answer = 3;  int guess;    printf("請猜一個數字 0 到 9： ");  scanf("%d", &guess);    if (guess == answer) {  printf("猜對囉！\n");  }  else {  printf("猜錯囉！答案是 %d\n", answer);  }    return 0;  } |

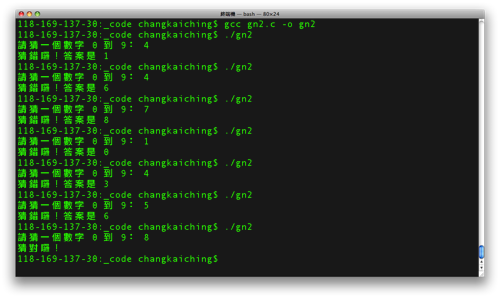
我們一共宣告了兩個變數，變數 answer 直接指派給值 3 作為答案，變數 guess 作為儲存使用者所猜數字的位置。程式很簡單，接受使用者輸入後，第 11 行判斷使用者所猜數字是否是否與答案相符

|  |  |
| --- | --- |
| 11  12  13  14  15  16 | if (guess == answer) {  printf("猜對囉！\n");  }  else {  printf("猜錯囉！答案是 %d\n", answer);  } |

若相符，印出猜對的序息，不相符就印出猜錯訊息附答案。來編譯玩看看吧！  
  
  
  
嗯，自己設計的自己知道答案，不是很有趣，我們再來設計一個不是先設定好答案的猜數字遊戲唄！

[?](https://pydoing.blogspot.com/2010/08/c-ch0210.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #include <stdio.h>  #include <time.h>    int main(void)  {  int answer = (int) time(NULL);  int guess;    answer %= 10;  printf("請猜一個數字 0 到 9： ");  scanf("%d", &guess);    if (guess == answer) {  printf("猜對囉！\n");  }  else {  printf("猜錯囉！答案是 %d\n", answer);  }    return 0;  } |

概念很簡單，就是從[標準函數庫](http://pydoing.blogspot.com/2010/07/c-std.html) [time.h](http://pydoing.blogspot.com/2010/07/c-stdtime.html) 的函數 [time()](http://pydoing.blogspot.com/2010/07/c-time.html) 取得現在的總秒數，然後將其取除以 10 的餘數，這樣我們就會得到一個 0 到 9 之間的整數，以此整數作為遊戲的答案。  
  
  
再來編譯執行玩玩看吧  
  
  
  
嗯，如果猜錯還可以追答案，這樣遊戲是不是有趣一點了呢？

### 第二章 邏輯陳述 目錄

函數 (function) 是 C 程式執行的單位，陳述 (statement) 則是函數裡程式碼執行的單位。 C 語言裡的單一陳述要以分號結尾 ; ，常見的陳述有以下幾類

|  |  |
| --- | --- |
| **種類** | **例子** |
| 運算式陳述 | 指派、呼叫函數 |
| 複合陳述 | 大括弧圍起來的地方 |
| 條件陳述 | if 、 if-else 、 switch |
| 重複陳述 | while 、 do-while 、 for |

最常見的就是運算式陳述了，所謂的運算式具有運算元及運算子，也可以單獨只有運算元，例如以下的程式碼在螢幕上印出 "Hello, world!" ，然後再印出新行

|  |
| --- |
| printf("Hello, world!\n"); |

單獨寫任何常數數值都是合法的常數運算式，如

|  |
| --- |
| 1234;  56.7;  '\n';  "A word to the wise is enough."; |

雖然這樣是合法的，可以成功編譯，但是這樣編譯出的執行檔什麼事也不會做。我們通常都會用變數處理資料，因此將常數數值指派給變數是最常用的陳述之一，例如

|  |
| --- |
| int a = 1234;  float b = 56.7;  char c = '\n';  char \*d = "A word to the wise is enough."; |

宣告變數就可以接上等號 = ，將常數數值直接指派給變數。  
  
  
C 語言具有豐富的運算子，如下表依運算子的優先順序列出 C 所有的運算子

|  |  |
| --- | --- |
| **運算子** | **結合規則** |
| () [] -> . | 左到右 |
| ! ~ ++ -- + - \* & (type) sizeof | 右到左 |
| \* / % | 左到右 |
| + - | 左到右 |
| << >%gt; | 左到右 |
| < <= > >= | 左到右 |
| == != | 左到右 |
| & | 左到右 |
| ^ | 左到右 |
| | | 左到右 |
| && | 左到右 |
| || | 左到右 |
| ?: | 右到左 |
| = += -= /= %= &= | 右到左 |
| ^= |= <<= >>= | 右到左 |
| , | 左到右 |

我們接下來會依需要來討論各種運算子，首先來用常用的算術運算子[來寫個加法器](http://pydoing.blogspot.com/2010/08/c-ch0201.html)

|  |  |  |
| --- | --- | --- |
| **運算子** | **功能** | **範例** |
| + | 加 | a + b |
| - | 減 | a - b |
| \* | 乘 | a \* b |
| / | 除 | a / b |
| % | 取餘數 | a % b |

加法器是很簡單的加法計算，我們再來[將加法器擴展成四則運算器](http://pydoing.blogspot.com/2010/08/c-ch0202.html)。  
  
  
四則運算器得分三次輸入，我們發展[新版本的四則運算器](http://pydoing.blogspot.com/2010/08/c-ch0203.html)，像手邊的小型計算機一樣，一次輸入就 OK 了。  
  
  
有些錯誤必須避免，[小心！除以 0](http://pydoing.blogspot.com/2010/08/c-ch0204.html) ，同時我們也來看看關係運算子

|  |  |  |
| --- | --- | --- |
| **運算子** | **功能** | **範例** |
| < | 小於 | a < b |
| <= | 小於等於 | a <= b |
| > | 大於 | a > b |
| >= | 大於等於 | a <= b |
| == | 相等 | a == b |
| != | 不等 | a != b |

邏輯運算子也很常用

|  |  |  |
| --- | --- | --- |
| **運算子** | **功能** | **範例** |
| ! | 非 | !a |
| && | 且 | a && b |
| || | 或 | a || b |

四則運算器還有些問題需要解決，若是該輸入整數卻輸入字元，會有奇怪的錯誤發生，因此要[解決輸入型態不相符的問題](http://pydoing.blogspot.com/2010/08/c-ch0205.html)。  
  
  
另外[遞增遞減](http://pydoing.blogspot.com/2010/08/c-ch0206.html)也是很常用的概念

|  |  |  |
| --- | --- | --- |
| **運算子** | **功能** | **範例** |
| ++ | 前遞增 | ++a |
| ++ | 後遞增 | a++ |
| -- | 前遞減 | --a |
| -- | 後遞減 | a-- |

[指派](http://pydoing.blogspot.com/2010/08/c-ch0207.html)運算子也可以和算術運算子合用

|  |  |  |
| --- | --- | --- |
| **運算子** | **功能** | **範例** |
| += | 相加同時指派 | a += b |
| -= | 相減同時指派 | a -= b |
| \*= | 相減同時指派 | a \*= b |
| /= | 相除同時指派 | a /= b |
| %= | 取餘數同時指派 | a %= b |

我們接下來繼續改進打招呼的程式，[向使用者 Say Hello!](http://pydoing.blogspot.com/2010/08/c-ch0208.html) 。  
  
  
應該要顯示出現在時刻，所以要[處理時間問題](http://pydoing.blogspot.com/2010/08/c-ch0209.html)。  
  
  
設計與使用者互動程式還是遊戲有趣點，所以最後[來玩猜數字吧](http://pydoing.blogspot.com/2010/08/c-ch0210.html)！