C語言語法使用時機以及進階知識

這裡不講 C語言語法整理裡常用函數的使用時機，只講選擇結構、重複結構以後的使用時機，不懂?

往下看就知道了!

P.S. 點超連結之前請先確認是否有連到網路喔!

變數的使用時機

當要暫時儲存資料時，或者資料要做運算時，可以宣告變數來儲存它(資料)，再經由該變數取得資料。

進階:宣告變數把資料放進變數裡面等同於把資料放進該變數的記憶體位址

下面兩張投影片整理每個變數的儲存範圍以及可使用的型態

P.S. 以下變數型態跟範圍只供參考，實際還是要視編譯器而定!

整數型態(位元大小)	範圍
char (8 bits)	-128 ~ 127
unsigned char (8 bits)	0 ~ 255
short int (16 bits)	-32768 ~ 32767
unsigned short int (16 bits)	0 ~ 65535
int (32 bits)	-2147483648 ~ 2147483647
unsigned int (32 bits)	0 ~ 4294697295
long long int (64bits)	-9223372036854775808 ~ 9223372036854775807
unsigned long long (64 bits)	0 ~ 18446744073709551615

浮點數型態(位元大小)	小數點範圍
float (32 bits)	小數點後第 6 ~ 7位
double (64 bits)	小數點後第 15 ~ 16位
long double (128 bits)	儲存位數要看編譯器

進階:
浮點數在儲存時，有可能因為其他因素，而導致過程不正確，影響到結果也不正確，有興趣者，可以參考下列網站:

1. 使用浮點數最最基本的觀念 (基本)
2. Back to Basic: 談浮點數的比較 (基本)
3. [浮點數] IEEE754 , C/C++ 浮點數誤差 (進階)

以上文章都需要花費一段時間觀看，請小心服用!

if 跟 switch使用時機

當有很複雜的條件式，例如 (value == 20 && value != 0) 時，一定要使用if條件判斷式，當只要判斷數字或字元時，就可以使用switch條件式。

那如果只判斷數字或字元時，switch 跟 if 哪個執行效率比較好呢?

下面一張投影片告訴您!

```
if (a == 1)
{
  ….
}
else if (a == 2)
{
  ….
}
else if (a == 3)
{
  ….
}
else
{
  ….
}
```

```
switch (a)
{
  case 1:
      ….
      break;
  case 2:
      ….
      break;
  case 3:
      ….
      break;
  default:
      ….
      break;
}
```

結論:
左邊兩個程式碼執行結果一樣，但差在每做一次條件式就需要取一次a的資料來做比較，switch只要取一次a的資料與下面的比較，故switch的執行效率較高。

for 、while 、do…while 使用時機

迴圈種類	使用時機
for	當確定要重複N次的時候使用 例如:1+2+3+….+100
while	當不確定要重複幾次的時候使用 例如:10進位整數轉2進位整數時，模擬短除法一直除以2，除到商數為0為止
do…while	

自訂函數使用時機

當需要一直重複一段程式碼在主程式時，雖然可以複製貼上該段程式碼在主程式的其他地方，但整體程式碼會看起來很冗長，如果把它寫成自訂函數時，每次使用只要呼叫該自訂函數名稱以及丟進參數處理，需要回傳值時在return就好了

如果不懂上面一大串再說什麼，下面一張投影片請點超連結到雲端硬碟觀看範例程式碼!

自訂函數 (範例)

輾轉相除法 (使用迴圈且不寫成自訂函數)

輾轉相除法 (使用迴圈但寫成自訂函數)

遞迴使用時機

當問題很複雜，或者根本沒辦法用迴圈寫的時候，例如:河內塔問題、排列組合等....

遞迴 V.S. 迴圈

那如果迴圈也可以寫出來的話，到底用遞迴比較好? 還是迴圈比較好?

先看看輾轉相除法的遞迴跟迴圈的程式碼吧!

下面一張投影片請點超連結到雲端硬碟觀看範例程式碼!

遞迴 vs 迴圈 (範例)

輾轉相除法(遞迴)

輾轉相除法(迴圈)

P.S. 請注意看Greatest_Common_Divisor裡的程式碼

兩個程式碼都看完了嗎? 下一張整理兩個的優缺點!

	遞迴	迴圈
優點	1. 程式碼較少 2. 減少程式編譯後的檔案大小 3. 表達力較強 4. 區域變數與暫存變數較少	1. 較節省執行時間 2. 不需額外的堆疊(stack)空間 3. 容易理解實際執行過程
缺點	1. 執行時參數的存取較費時間 2. 需要額外的堆疊(stack)空間 3. 不易理解實際執行過程	1. 程式碼較多 2. 增加程式編譯後的檔案大小 3. 表達力較弱 4. 區域變數與暫存變數較多

陣列在記憶體的配置

不管維度多少，在記憶體都會轉為一維陣列儲存，
到後面這個觀念非常重要，尤其是第七章的指標。
不信嗎? 我寫好了程式，請注意看記憶體位址的變化!

一維陣列記憶體位址

二維陣列記憶體位址

三維陣列記憶體位址

貼心小提醒:請在電腦執行喔! 如有防毒軟體誤報為有毒軟體的情況請忽略，
不放心的我可以提供程式碼給您!