C Programming Language

劉和師

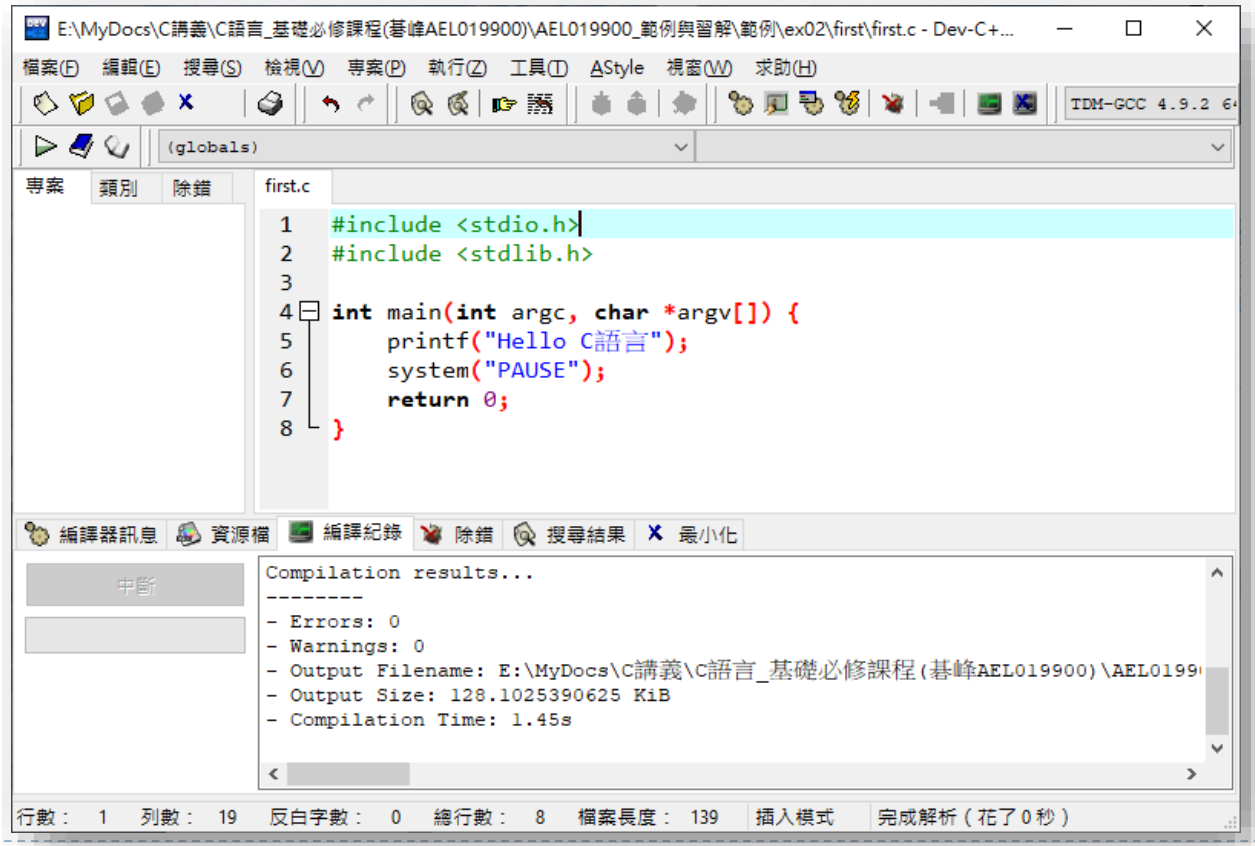
2024/05/24

C語言

- 它是現在一些熱門語言的「阿公」，經久不衰，許多語言都是由它延伸發展而來的。
- 它的適用範圍廣，包含軟體工程師、硬體工程師都會用到。
- 如果你熟悉了C語言，那學習其它語言都不是問題。
- 它不適合寫視窗程式，但用來控制裝置很方便。如果你要寫視窗程式可以考慮微軟的C#或Python的PyQt6套件。

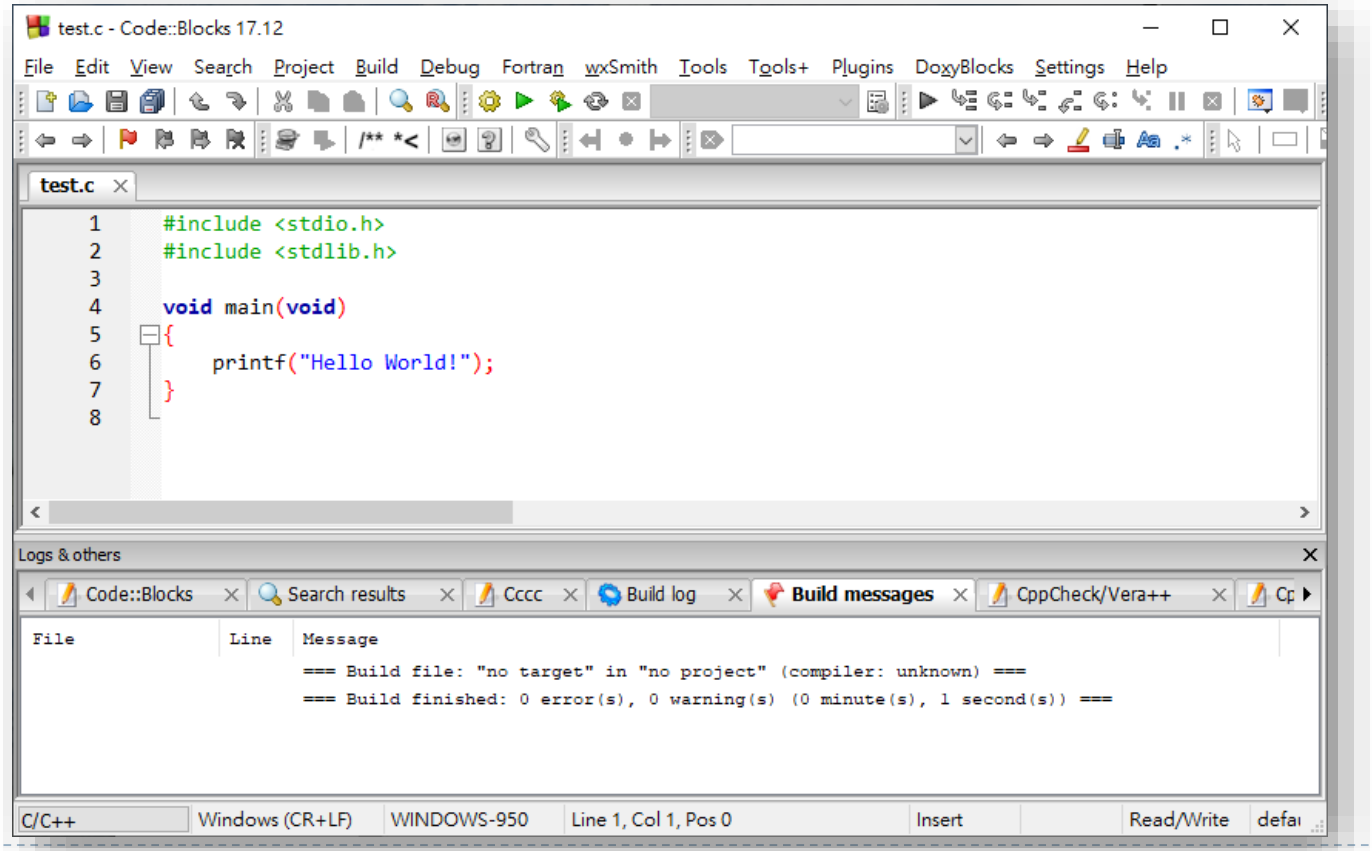
編輯環境建立及操作

- 建議的簡易編譯環境(一)：
  - Dev C++：



編輯環境建立及操作

- 建議的簡易編譯環境(二)：
  - CodeBlocks：



編輯環境建立及操作

- 關於編譯環境：
  - 你可以在谷哥大神找到這些免費軟體的下載。
  - 在學習的過程中，你可以使用任一個編譯環境來執行。
  - 微軟的Visual Studio是一個很不錯功能又強的IDE(整合發展環境)，但對現階段的你來說太龐大，不需要。
  - 安裝過程請看老師上課安裝。
  - 課程內容的程式請一定要親自去試試。

休息一下~

- 去準備編譯及執行的環境~

C語言架構

- C語言的架構
  - 是由一個個模組所組成，包含:
    - 1.前端處理程式
    - 2.全域變數
    - 3.主程式模組
    - 4.函數式模組

- C語言的概念是以函數(Function)為主，連主程式main( )都是一個函數。

C語言架構

前端處理程式

```
#include   標頭檔  //引進所需要之庫存函數
           :
#define    識別字或常數    //增加可讀性
           :
全域變數宣告
           :
```

主程式模組，只能有一個

```
main( ) //程式進入點,由此開始執行
{
        主程式區域變數宣告
        敘述…
}
```

函數模組，可以有很多個

```
functions( )
{
        函數區域變數宣告

        敘述…
}
```

C語言架構

- 識別字(Identifiers)：
  - 程式中所用到的各種名稱、如變數名稱、常數名稱、函數名稱等。
  - 要自己取名字，不可以用中文。
  - 最好是和其相關的有意義的名字。

C語言架構

- 識別字命名規則：
  - 第一個字一定是英文字母或底線字元“_”。
  - 其他字元可以是英文字母、數字、底線字元或“$”符號。
  - 長度不可超過255個字。
  - 大小寫字母所代表意義不同(即大小寫是嚴格區分的)。
  - 關鍵字或庫存函數不可當識別字。
  - **一些例子：**

正確的	錯誤的	說 明
abc	3abc	第一個字母不可為數字
_ab	for	不可使用關鍵字
A12	A#12	不可使用特殊符號

C語言架構

- 識別字命名風格：
  - 當識別字需要好幾個單字來表達時，可採用：
    - 加底線，例如：your_name、get_score。
      - 用底線隔開幾個單字。
    - 駝峰式，例如：yourName、getScore。(推薦)
      - 將單字首予以大寫來區分。

C語言架構

- 關鍵字(Key words)：
  - 或稱保留字，是具有特殊意義的識別字。
  - 程式中不可以使用關鍵字當作變數名稱、常數名稱或函數名稱。
  - 關鍵字在不同的Compiler或不同的版本中可能有差異。

C語言架構

- C語言常用關鍵字：

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

註解

- 一個好的程式會有適當的註解。
- 註解是給人看的，編譯器會予以忽略。
  - 單行註解：

```
// one line comment
```

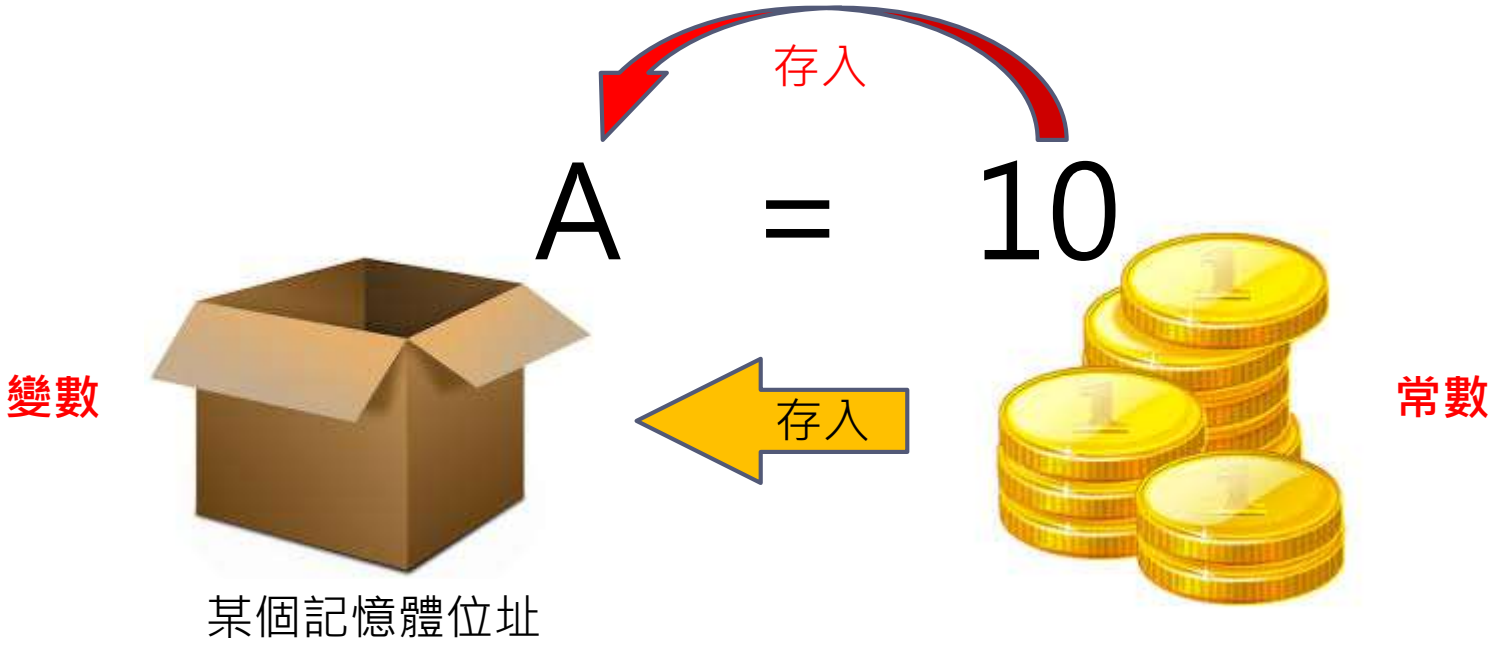
  - 多行註解：

```
/* a comment */  或
/*----------------------------
    another comment
  ----------------------------*/
```

- 適當的空白行可以有助程式閱讀，編譯器會忽略空白行及空白字元。

常數與變數

- **變數**：代表一個可存放資料的記憶體位置，是程式中不可或缺的部分。
- **常數**：程式執行時的各項資料的值。

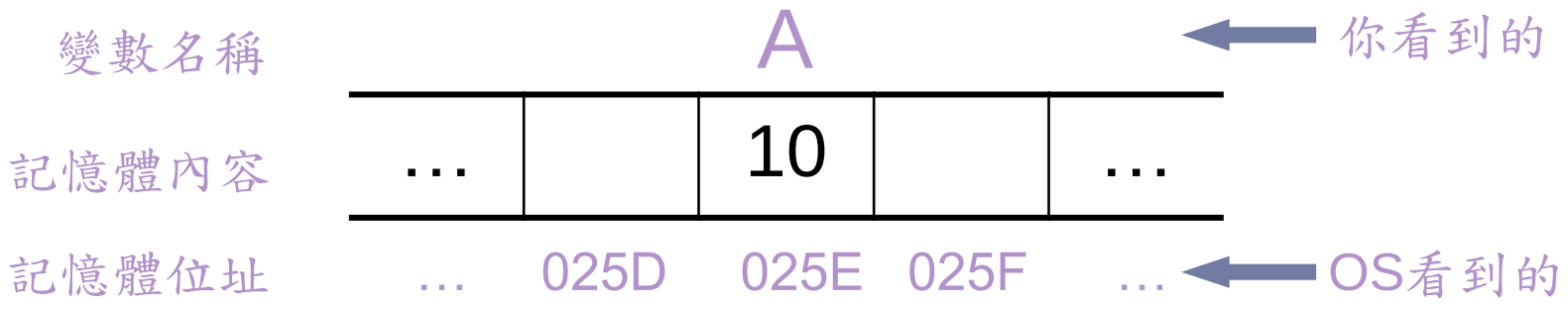


常數與變數

- 變數：
  - Ex: 敘述 `A = 10;` （把10存到A裡面）

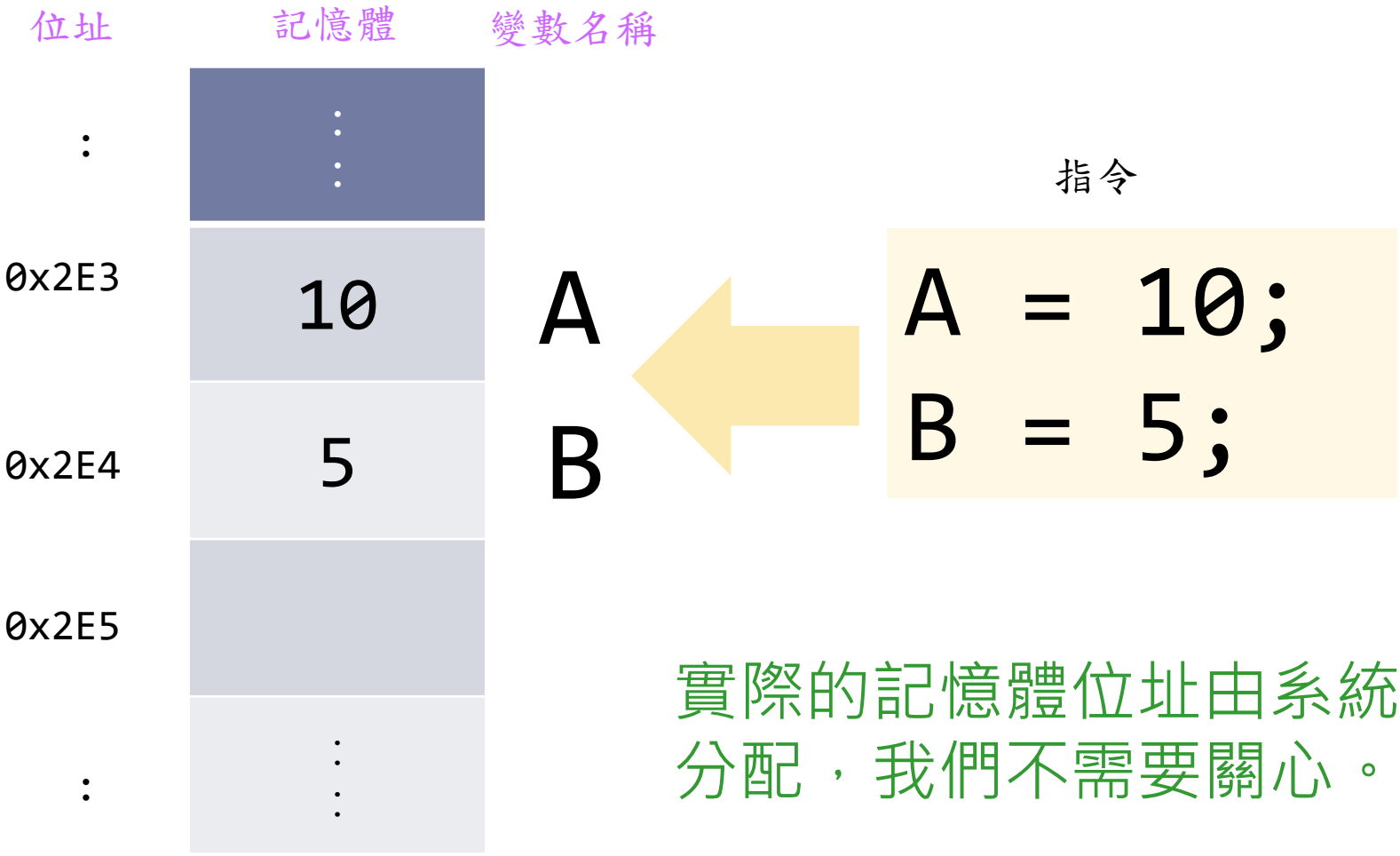
✸注意：C語言每一道敘述後面都要加上一個分號";"當結尾，不可遺漏。

- 實際情況：



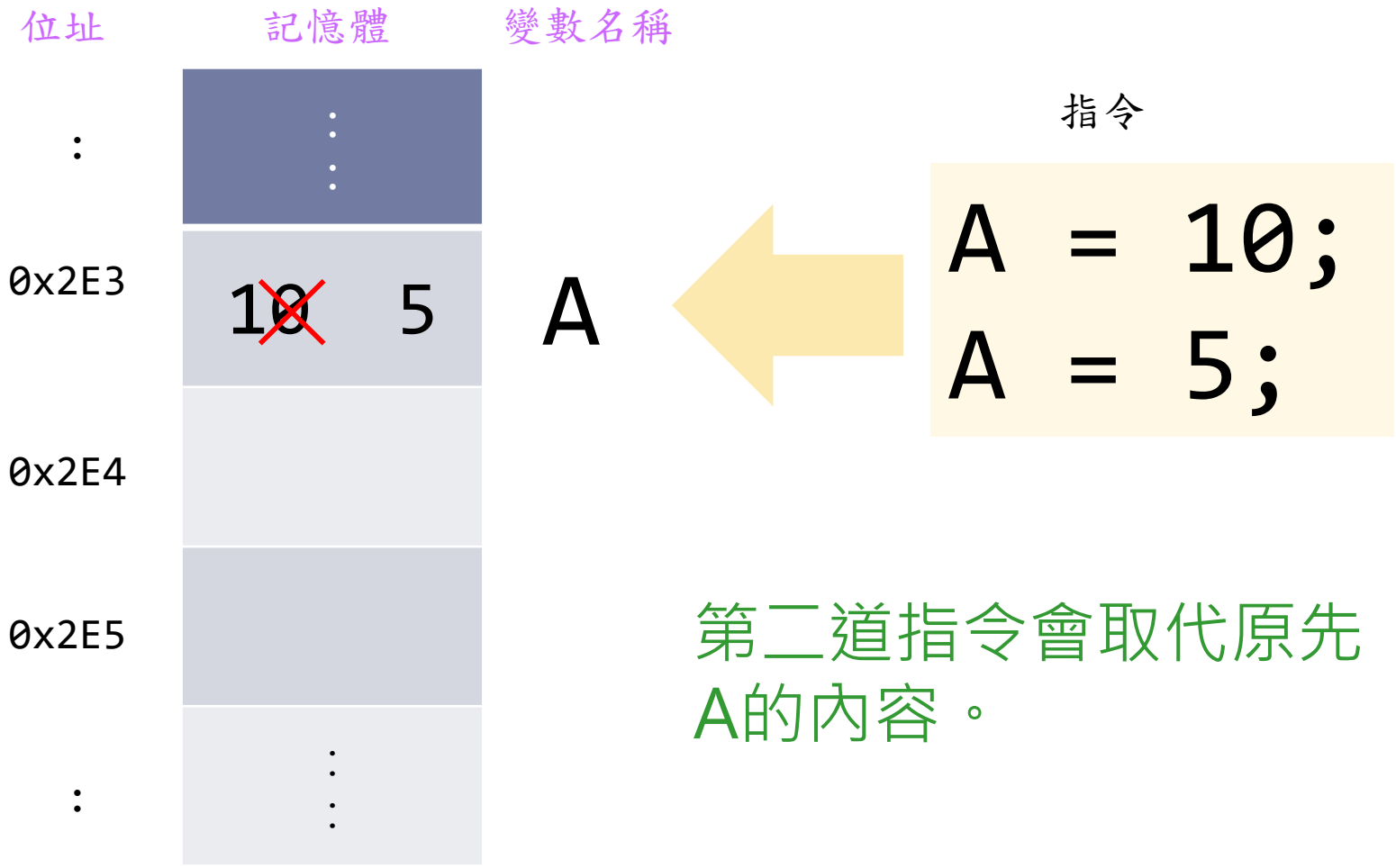
(不能自己指定，由OS分配)

常數與變數



實際的記憶體位址由系統分配，我們不需要關心。

常數與變數



常數與變數

- 在程式語言中，一個變數即代表記憶體中一個儲存空間，佔一至數byte不等，依指定之資料型態而定。
- 在整個程式執行期間，變數名稱是不變的，但其所儲存的值可能會一直改變。
- 所佔用之記憶體位置由作業系統(Operating System)決定。

資料型別

- 整數：不帶小數之數，有10、8、16、2進制四種表示法。

進制	正數	負數
10	20	-20
8	024	-020
16	0x14	-0x14
2	0b01100001	沒有
註：0為數字零,不是字母O		

資料型別

- 浮點數：帶小數點的數。
  - 如: 123.4,　1.234e+02(表示1.234×10^2)
- 字元：括在一對單引號之間的字元。
  - 如：'a'。
- 字串：括在一對雙引號之間的一串字元。
  - 如："My name is Orion"。

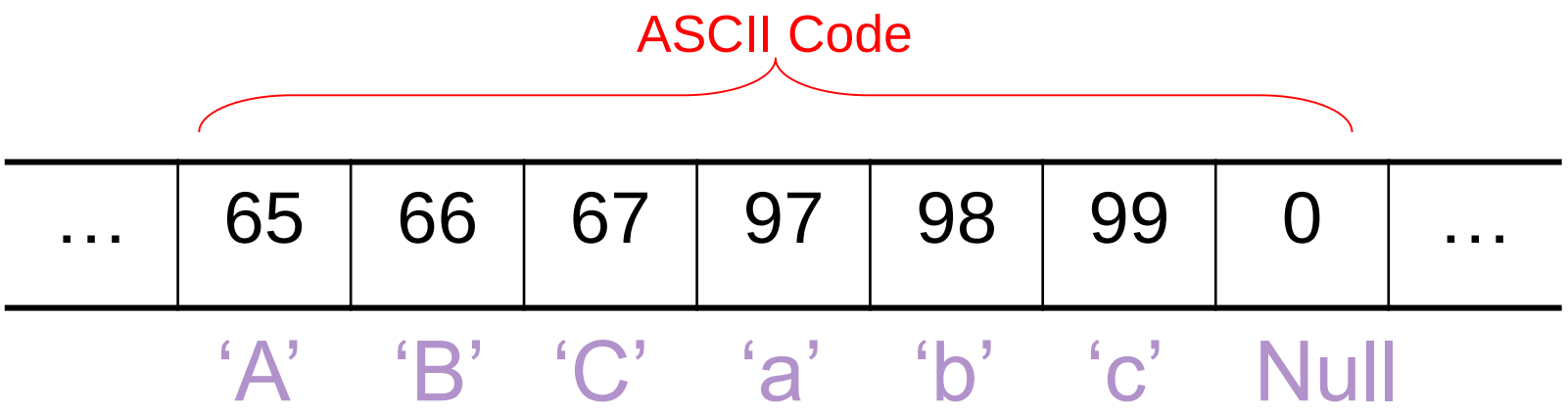
資料型別

- 字串：
  - 其長度在C語言中沒有限制(僅受限於記憶體大小)。
  - 字串的最後會自動加上一個Null byte(空字元),以表示字串的結束。
  - 一個英數字佔一個byte，中文兩個byte。

    (現在有許多編碼方式，ASCII、BIG5、UTF8、UTF16...，一般來說一個中文兩個byte，如果必要的話還是要自己確定一下)

資料型別

- 例如字串"ABCabc"在記憶體中的情形：



- 由n個字元組成的字串佔用n+1個byte的記憶體。

變數及資料型別

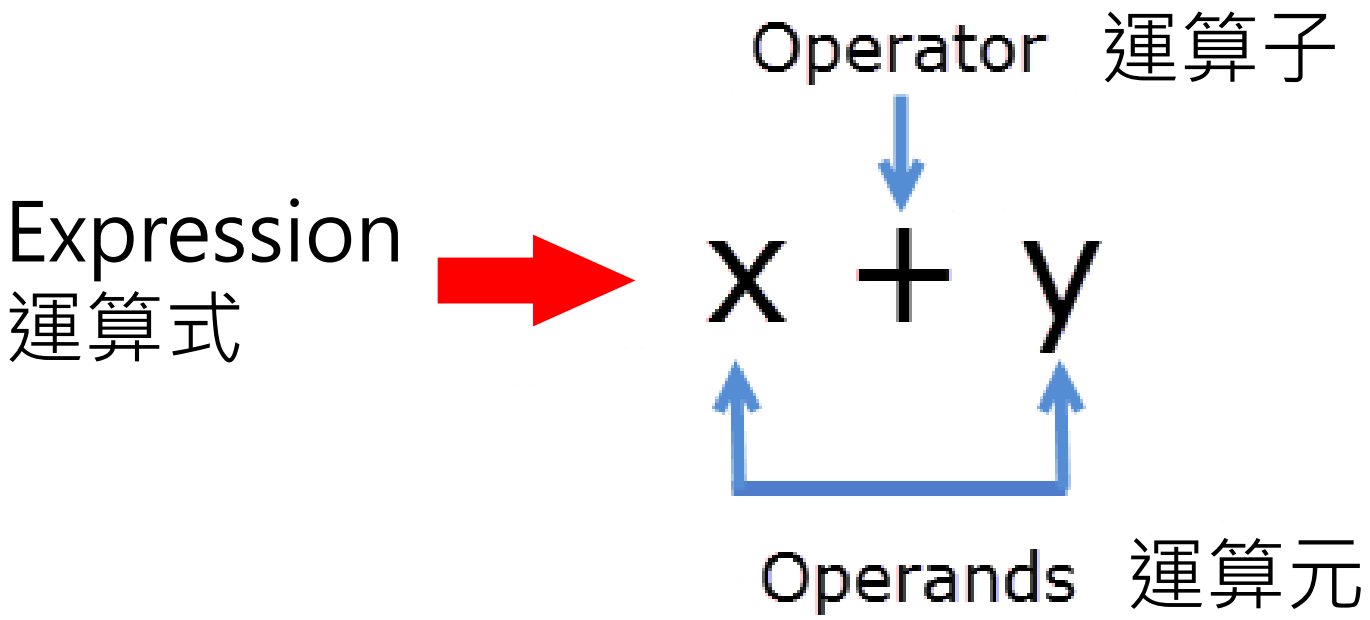
- 資料型別及表示範圍參考：(依作業系統會稍有不同)

Data type	Size(bytes)	Range	Format string
Char	1	128 to 127	%c
Unsigned char	1	0 to 255	%c
Short or int	2	-32,768 to 32,767	%i or %d
Unsigned int	2	0 to 65535	%u
Long	4	-2147483648 to 2147483647	%ld
Unsigned long	4	0 to 4294967295	%lu
Float	4	3.4 e-38 to 3.4 e+38	%f or %g
Double	8	1.7 e-308 to 1.7 e+308	%lf
Long Double	10	3.4 e-4932 to 1.1 e+4932	%lf

休息一下~

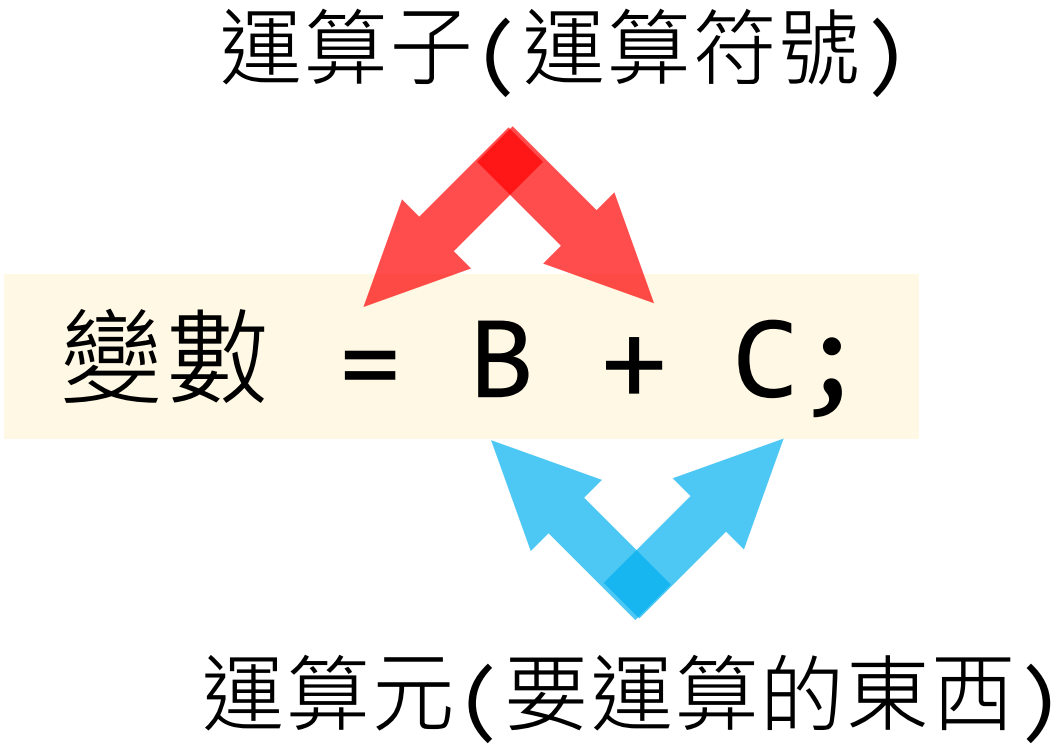
運算式與運算符號

- 運算子(Operator)與運算元(Operand)



運算式與運算符號

- 一個運算式是由運算子及運算元構成的：



運算式與運算符號

- 算術運算：

運算子	功能說明	範例	結果(假設 y=4)
+	加法	x = y + 2	x 變數值為 6
-	減法	x = y - 2	x 變數值為 2
*	乘法	x = y * 2	x 變數值為 8
/	除法	x = y / 2	x 變數值為 2
%	取整數的餘數	x = y % 2	x 變數值為 0
>>	右移位元	x = y >> 2	x 變數值為 1(等於除以 4)
<<	左移位元	x = y << 2	x 變數值為 16(等於乘以 4)

- 注意：整數除整數仍然會得到整數，不會有小數。

運算式與運算符號

- 複合運算：

運算子	功能	範例	結果(x 變數值原為 3)
=	指定	x = 3	x 變數值為 3
+=	相加後再指定	x += 2 (x = x + 2)	x 變數值為 5
-=	相減後再指定	x -= 2 (x = x - 2)	x 變數值為 1
*=	相乘後再指定	x *= 2 (x = x * 2)	x 變數值為 6
/=	相除後再指定	x /= 2 (x = x / 2)	x 變數值為 1
%=	相除取餘數後再指定	x %= 2 (x = x % 2)	x 變數值為 1

  - 複合運算可以簡化運算式。

運算式與運算符號

- 遞增/遞減運算：

遞增、減運算式	一般運算式寫法	結果(若 x=2)
y = ++x ;	x = x + 1 ; y = x ;	x = 3 , y = 3
y = x++ ;	y = x ; x = x + 1 ;	x = 3 , y = 2

  - 可以簡化運算式，這個運算在C語言很常用。

運算式與運算符號

- 關係運算：

關係運算子	功能	數學表示式	範例	結果(若 x=1、y=2)
==	等於	x = y	x == y	0(假)
!=	不等於	x ≠ y	x != y	1(真)
>=	大於等於	x ≥ y	x >= y	0(假)
<=	小於等於	x ≤ y	x <= y	1(真)
>	大於	x > y	x > y	0(假)
<	小於	x < y	x < y	1(真)

  - 「=」是指定運算，「==」是關係運算。

運算式與運算符號

- 邏輯運算：

邏輯運算子	功能	範例	結果(若 x=1、y=0)	說明
&&	且	x && y	0(假)	兩者皆真結果才為真
||	或	x || y	1(真)	只要一個為真結果就為真
!	非	!x	0(假)	相反(原為真結果為假)

- 真值表：

x	y	x && y	x || y	!x	!y
1	1	1	1	0	0
1	0	0	1	0	1
0	1	0	1	1	0
0	0	0	0	1	1

運算式與運算符號

- 運算符號的優先序：

優先次序	運算子(Operator)	運算次序
1	()(呼叫函數)、[](陣列註標)、++(後置)、--(後置)、()	由左至右
2	!(非)、+(正號)、-(負號)、++(前置)、--(前置)	由右至左
3	*(乘)、 / (除)、 %(取餘數)	由左至右
4	+(加)、 -(減)	由左至右
5	<< (左移) 、 >> (右移)	由左至右
6	< 、 <=、 > 、 >= (關係運算子)、	由左至右
7	== (相等) 、 != (不等於)	由左至右
8	&& (且邏輯運算子)	由左至右
9	|| (或邏輯運算子)	由左至右
10	？： (條件運算子)	由右至左
11	= 、 += 、 -= 、 *= 、 /= 、 %= 、 <<= 、 >== 、 &= 、 ^= 、 != (指定、複合指定運算子)	由右而左
12	, (逗號)	由左至右

運算式與運算符號

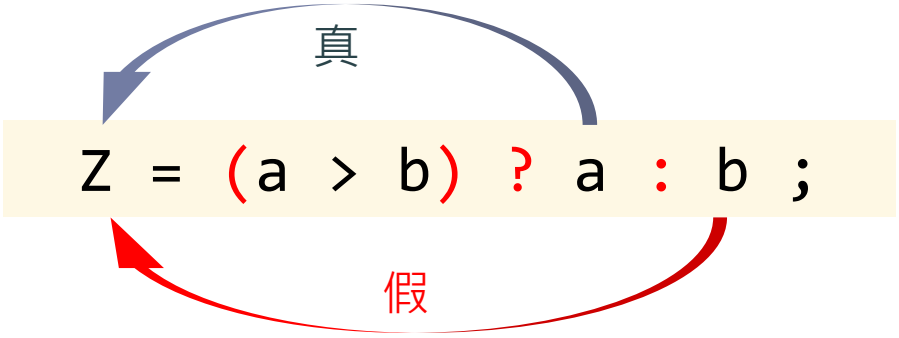
- 位元運算：

位元運算子	符號	範例	a	b	c
AND	&	c = a & b	1010	1001	1000
OR	|	c = a | b	1010	1001	1011
XOR	^	c = a ^ b	1010	1001	0111
NOT	~	c = ~a	1010		0101

  - 依真值表位元對位元運算，沒有進位或借位的問題。
  - 此類運算我們比較少用到，但8051單晶片的C常用到。

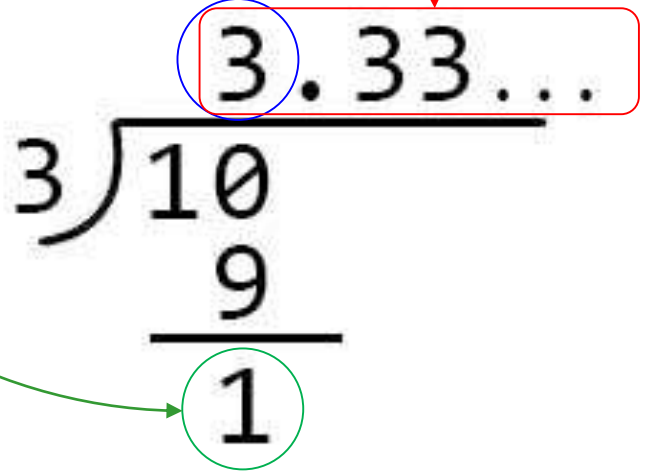
運算式與運算符號

- 條件運算子：
  - 格式：　(關係運算式)?運算式1:運算式2;
  - 說明：若關係運算式之值為真，則取運算式1之值，否則取運算式2之值。
  - 例：比較a,b之值，較大者指定給Z。



運算式與運算符號

- 注意C的三種"除法"運算：
  - `10 / 3.0 = 3.333333` (有一邊是浮點數)
  - `10 / 3 = 3` (兩邊都是整數)
  - `10 % 3 = 1` (求餘數)



型別轉換

- 當兩個不同型態的運算元作運算時，C會根據一些轉換法則將兩個運算元轉換成相同型態，原則為:
  - 所有字元型態(char)和短整數(int)→皆轉換為int。
  - 所有浮點型態(float)→皆轉換為double型態。
  - 一但有double運算元→其它運算元皆轉換為double。
  - 一但有long運算元→其它運算元皆轉換為long。
  - 一但有long double運算元→其它運算元皆轉換為long double。
  - 一但有unsigned運算元→其它運算元皆轉換為unsigned。

型別轉換

- C語言提供一個型態轉換運算符號，用以強迫改變運算元之型態，格式為:

(型態)運算元;

- Ex:

x = 10/3 → x得到3

x = (float)10/3 → x得到3.333333

- 一個良好的程式應避免讓運算式自行轉換型態，才能確保程式的正確與效率。

休息一下~

基本輸出入函式

- 主函式main()架構：
  - 格式一：

```
void main(void)
{
      敘述;
        :
}
```

程式結束時傳回給作業系統的型態。void表示沒有傳回值。

沒有命令列輸入參數

  - 格式二：

```
int main(int argc, char *argv[])
{
      敘述;
        :
      return 0;
}
```

程式結束時傳回一個整數給作業系統，這是比較好的做法。

取得命令列輸入參數個數及其值

傳回一個值，讓作業系統可以知道程式是否正常結束

基本輸出入函式

- 輸出函式printf()：
  - 格式：

```
printf("字串");
```

    - 單純輸出一個字串。

  - 或：

```
printf("格式字串",引數1,引數2, ...);
```

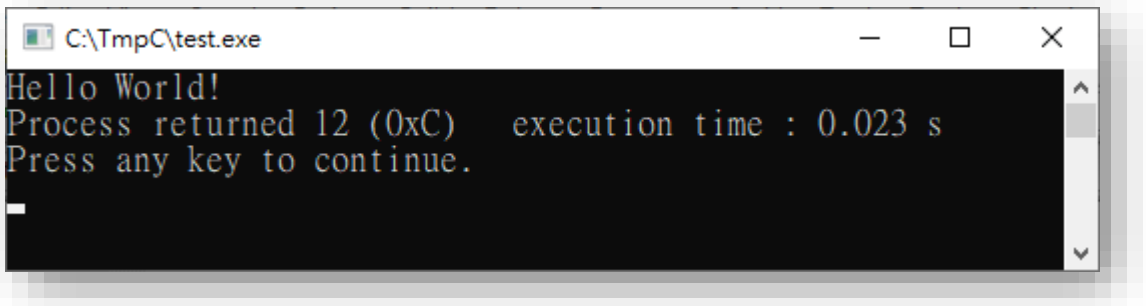
    - 依指定的格式輸出。

基本輸出入函式

- 輸出函式printf()：
  - 例：

```c
#include <stdio.h>     //引入標準輸出入函式庫
#include <stdlib.h>    //引入標準函式庫

void main(void)     //void表示沒有東西
{
    printf("Hello World!");    //不要忘了結尾分號
}
```

  - 輸出結果：



基本輸出入函式

- 在Dev C++和CodeBlocks環境，輸出結果會出現在一個命令列視窗(DOS視窗)，並會自動停止，等你按Enter後返回。
- 其它環境若輸出視窗一閃而過，就要加上一行讓程式暫停的指令，以便檢視結果。

- 暫停指令：

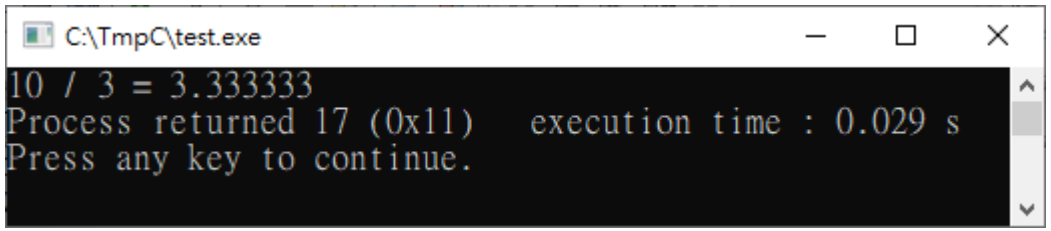
```
system("PAUSE");
```

基本輸出入函式

- 輸出函式printf()：
  - 例：

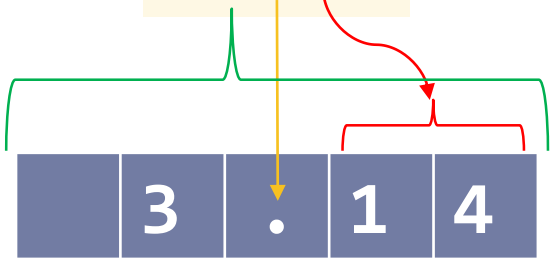
```
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
    int x = 10;    //宣告變數並指定初值
    int y = 3;
    float z;
    z = (float)x / y;    //依序將引數交給格式字串顯示
    printf("%d / %d = %f", x, y, z);
}
```

  - 輸出結果：



基本輸出指令

- printf()的%格式化的控制方式：
- Ex： %5.2f 表示總共留5格，小數佔用2格。
  (小數點固定占用1格)



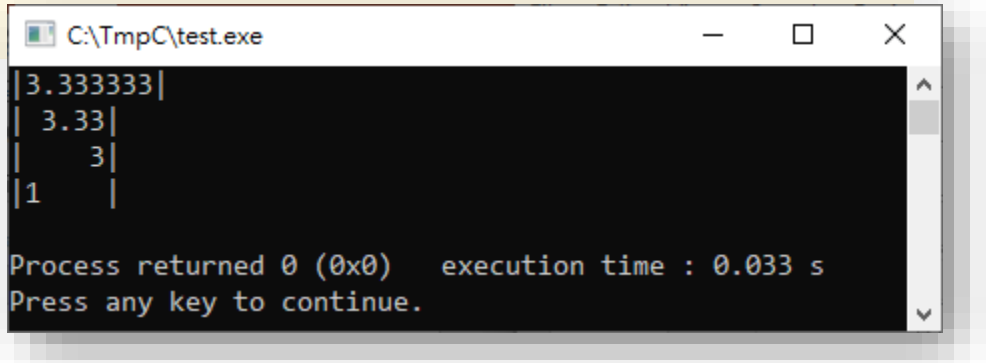
（若未填滿則留出空白）

- Ex： %5d 表示總共留5格，向右靠齊。
- Ex： %-5d 表示總共留5格，向左靠齊。

基本輸出入函式

- printf()的格式化控制範例：

```c
#include <stdio.h>
#include <stdlib.h>

int main(void){
    printf("|%f|\n", 10/3.0);   //依內定長度顯示，向右靠齊
    printf("|%5.2f|\n", 10/3.0); //指定顯示格式，小數2位
    printf("|%5d|\n", 10/3);     //指定寬度，靠右對齊
    printf("|%-5d|\n", 10%3);    //指定寬度，靠左對齊
    return 0;
}
```



基本輸出入函式

- 輸出格式字串：

資料型別	%型別字元	對應引數資料型別	指定輸出格式
字元	%c	char	字元顯示。
字串	%s	char *	字串顯示。
整數	%d	int	以含正負號 10 進位整數顯示。
	%o	int	以無正負號八進位整數顯示。
	%x	int	以小寫無正負號 16 進位整數顯示。
	%X	int	以大寫無正負號 16 進位整數顯示。
	%u	int	以無正負號 10 進位整數顯示。
	%l		以長整數顯示，可以%ld、%lu、%lo、%lx 四種方式表示。
浮點數	%f	float double	數值含小數來顯示。預設小數位數有 6 位。
	%e	float double	以[-]m.nnnnnne[+] 指數型式來表示 float 或 double 的資料。n 位數預設 6 位。
	%E	float double	以[-]m.nnnnnnE[+] 指數型式來表示 float 或 double 的資料。n 位數預設 6 位。
	%g	float double	以輸入值的位數決定使用%f 或%e 輸出數值。若整數位數 7 位以上，使用指數型式。
	%G	float double	以輸入值的位數決定使用%f 或%E 輸出數值。若整數位數 7 位以上，使用指數型式。

基本輸出入函式

- 跳脫字元：有特殊意義的字元組合。

逸出序列	使用說明
\a	發出警告聲。
\b	倒退鍵，會由目前游標所在位置向左刪除一個字元。
\f	換頁。
\n	換行，游標會由目前所在位置跳到下一行的行首。
\r	移到行首，會刪除掉該行游標所在位置前面的所有字元。
\t	水平跳格，每個間格為 8 格字元。
\v	垂直跳格。
\\	顯示倒斜線「\」字元。
\'	顯示單引號「'」字元。
\"	顯示雙引號「"」字元。
\nnn 或 \ooo	以八進制輸出。 例：printf("\101");　　// 顯示 A 字元
\xhh	以十六進制輸出。 例：printf("\x41");　　// 顯示 A 字元
\0(零)	Null 字元，代表字串的結尾

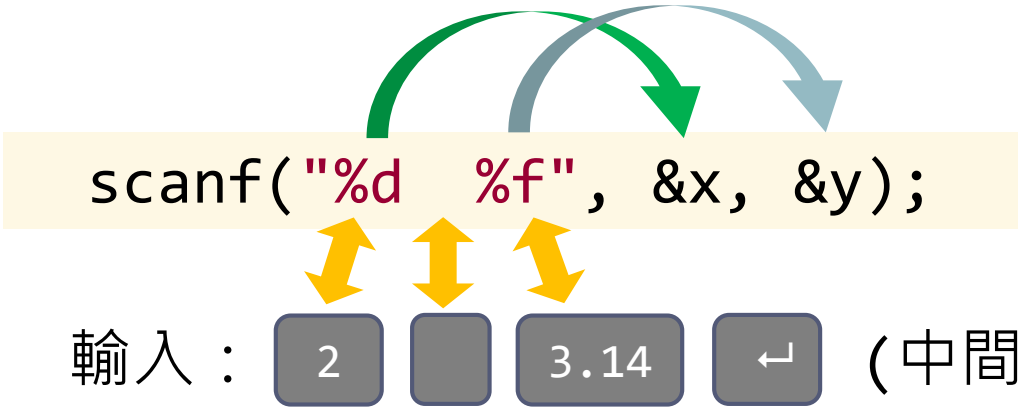
基本輸出入函式

- 輸入函式scanf()：
  - 格式：

```
scanf("格式字串", &變數1, &變數2, ...);
```

    - 依格式字串指定的格式取得輸入，並依序放入變數。

  - 例：

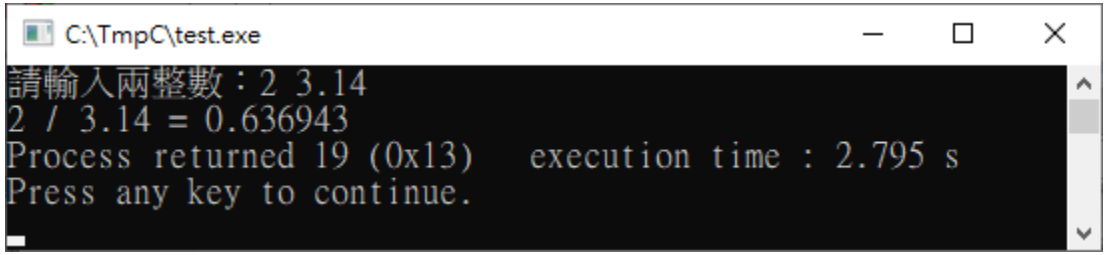


基本輸出入函式

- 輸入函式scanf()：
  - 例：

```c
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
    int x;
    float y, z;
    printf("請輸入一個整數及一個浮點數：");   //提示訊息
    scanf("%d %f",&x, &y);
    z = x / y;
    printf("%d / %.2f = %f", x, y, z);
}
```

  - 執行結果：



基本輸出入函式

- 輸出入格式字串：

%type	輸入資料型別	%type	輸入資料型別
%c	字元	%f	單精確浮點數
%s	字元陣列(字串)	%lf	雙精確浮點數
%d	十進位整數	%e	學科記號
%u	無負號十進位整數	%o	八進位整數
%ld	長整數	%x	十六進位整數

- 輸入間隔字元：

常用的 隔開字元	scanf()函式	鍵入資料方式
空格 (用Δ表示)	scanf("%dΔ%d",&n1,&n2);	24Δ31↵
逗點「,」	scanf("%d,%d",&n1,&n2);	24,31↵

休息一下~

運算式與輸出入練習

- 練習：

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
  int x;
  x = -6 + 3 % 4 - 3;
  printf("%d\n", x);
}

執行結果: -6
```

Ans:

x=-6+3%4-3

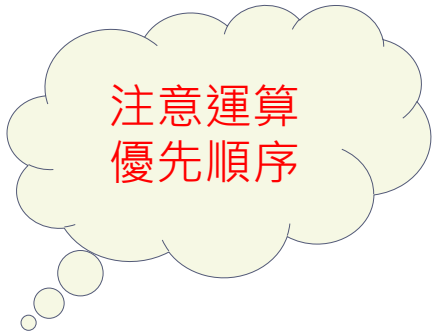
=(-6)+3-3

=-3-3

=-6

運算式與輸出入練習

- Ex:設 int x, y;
  - (a) y=x=(2+3)/4;　結果：x=1, y=1
    - x=(2+3)/4 =5/4 =1,　y=x
  - (b) y=3+2*(x=7/2);結果：x=3, y=9
    - x=7/2 =3, y=3+2*(x=7/2) =3+2*3 =9
  - (c) x=(int)3.8+3.3;　結果：x=6
    - x=3+3.3 =6.3 =6(x為int)
  - (d) x=3/5*22.0;　結果：x=0
    - x=0*22.0 =0
  - (e) x=22.0*3/5;　結果：x=13
    - x=66.0/5 = 13



運算式與輸出入練習

- 練習：

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
    int x, y=3;
    x = y == 2;
    printf("%d\n",x);
}
```

執行結果：0 (假)

Ans:

x = y == 2

判斷y是否等於2

y = 3 ≠ 2,故傳回0

x=0

運算式與輸出入練習

- 練習：

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
    int x, y=3, z;
    x = y <= (z = 2);
    printf("%d\n",x);
}
```

執行結果：0

Ans:

(y=3) <= (z=2)

不成立,故傳回0

x = 0

運算式與輸出入練習

- 練習：

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
    int x=3, y=2, z=1;
    x = x && y || z;
    printf("%d\n",x);
}
```

執行結果：1

Ans:

因為x,y皆不為0

x = x && y || z

=(True) && (True) || z

=(True) || z

有一者為真即不再做||運算

= True = 1

運算式與輸出入練習

- 練習：

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
    int x=2, y=0, z=1;
    x = !x || y && z;
    printf("%d\n",x);
}
```

執行結果：0

```
Ans:
x = !x || y && z
  = 0 || (0 && 1)
  = 0 || 0
  = 0
```

運算式與輸出入練習

- C常用的++(加1運算子)及--(減1運算子)。

- ++與--可放在變數之前，也可放在變數之後，可分成：
  - 前置運算子：
    - 置於變數前,例如： ++n, --n
  - 後置運算子：
    - 置於變數後,例如： n++, n--

運算式與輸出入練習

- 前置運算子：

(a) x = ++n 可寫成 {n=n+1; x=n;}

先將n+1再指定給x

Ex:若執行前x=2,n=6,執行x=++n之後

x=7, n=7

(b) y = --n 可寫成 {n=n-1; y=n;}

先將n-1再指定給y

Ex:若執行前y=2,n=6,執行y=--n之後

y=5, n=5

運算式與輸出入練習

- 後置運算子：

(a) x = n++ 可寫成 {x=n; n=n+1;}

先將n給x,再將n值加1

Ex:若執行前x=2,n=6,執行x=n++之後

x=6, n=7

(b) y = n-- 可寫成 {y=n; n=n-1;}

先將n給y,再將n值減1

Ex:若執行前y=2,n=6,執行y=n--之後

y=6, n=5

運算式與輸出入練習

- 練習:
- 若 w=5, x=8, y=11, z=13 則

```
a = w-- + --x + y++ - ++z  ?
```

Ans:

```
a = (5--)+(--8)+(11++)-(++13)
  =  5   +   7 + 11   -   14
  = 9
```

- 執行後：a=9, w=4, x=7, y=12, z=14

運算式與輸出入練習

- 練習:
- 若 a=4, b=6 則

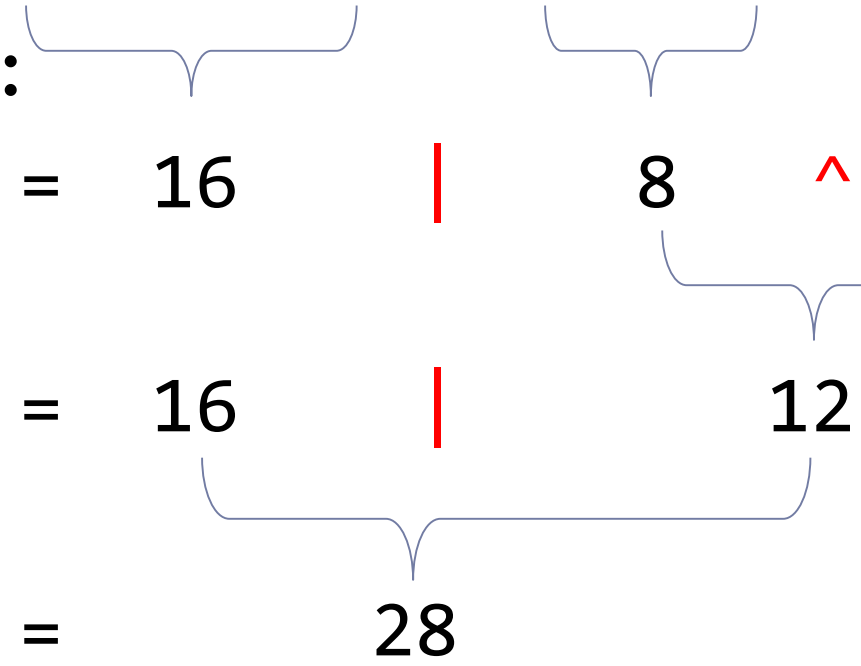
```
x = 39/-++a+-29%b--  ?
```

Ans:

```
x = 39 / (-(++a)) + ( -(29) % (b--))
  = 39 /     -5   +    -29  %  6
  =    -7         +        -5
  =  -12
```

運算式與輸出入練習

- 練習(位元運算)：
- `2 * 5 + 6 | 15 - 7 ^ 16 * 9 % 5`
- Ans:



*注意運算優先順序

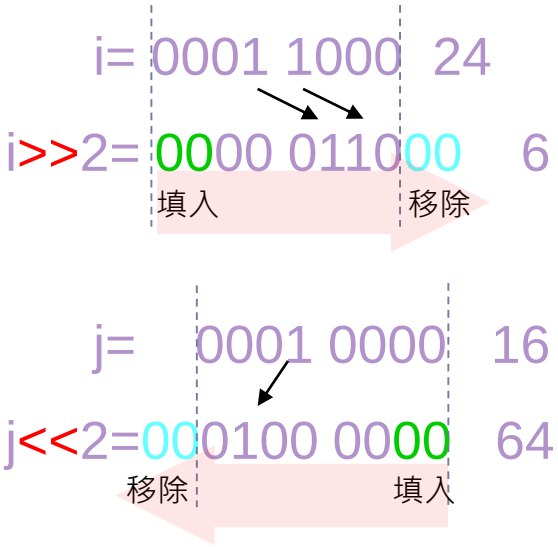
	0000 1000	8
^ XOR)	0000 0100	4
	0000 1100	12
| OR)	0001 0000	16
	0001 1100	28

運算式與輸出入練習

- 範例：

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
  int i=24, j=16, p, q;
  p = i >> 2;
  q = j << 2;
  printf("p=%d,q=%d\n",p,q);
}
```

執行結果：p=6, q=64



運算式與輸出入練習

- 範例：

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
    int x=2, y=5, z;
    z=(x < y)? ++x : --y ;
    printf("%d\n",z);
}
```

執行結果：3

x < y 成立，

故取 ++x 之值

休息一下~

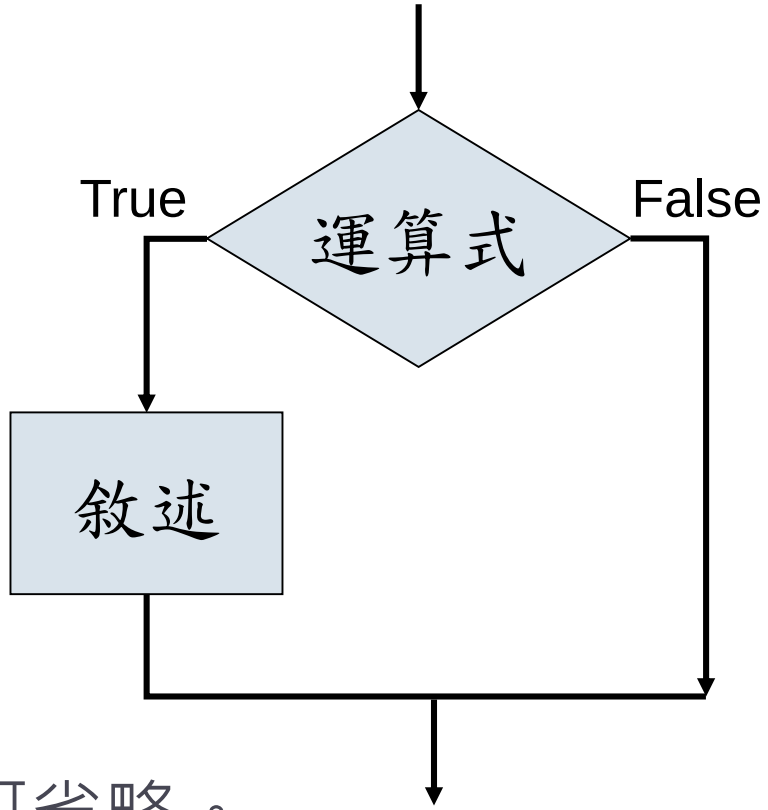
控制結構

- 1.順序結構(sequence structure)
- 2.選擇結構(selection structure)
  - if – else
  - switch
- 3.重複結構(loop structure)
  - for
  - while
  - do - while

選擇結構：if – else 敘述

- 格式一：

```
if(運算式)
{
      敘述;
        :
        :
}
```

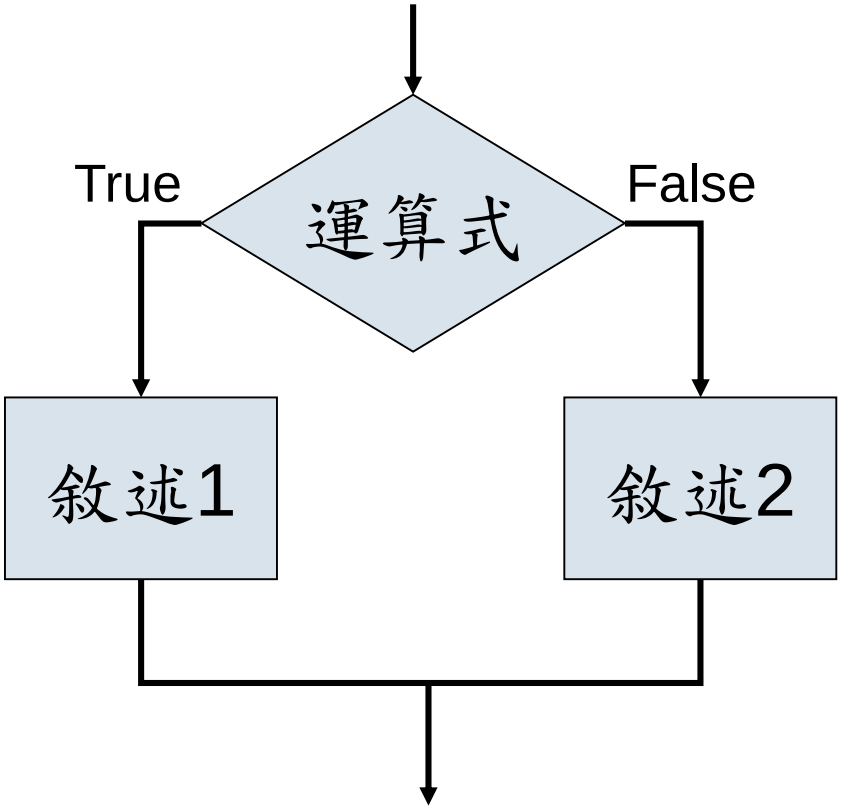


  - 若只有一行敘述，{}可省略。

選擇結構：if – else 敘述

- 格式二：

```
if(運算式)
{
      敘述1;
        :
}
else
{
      敘述2;
        :
}
```

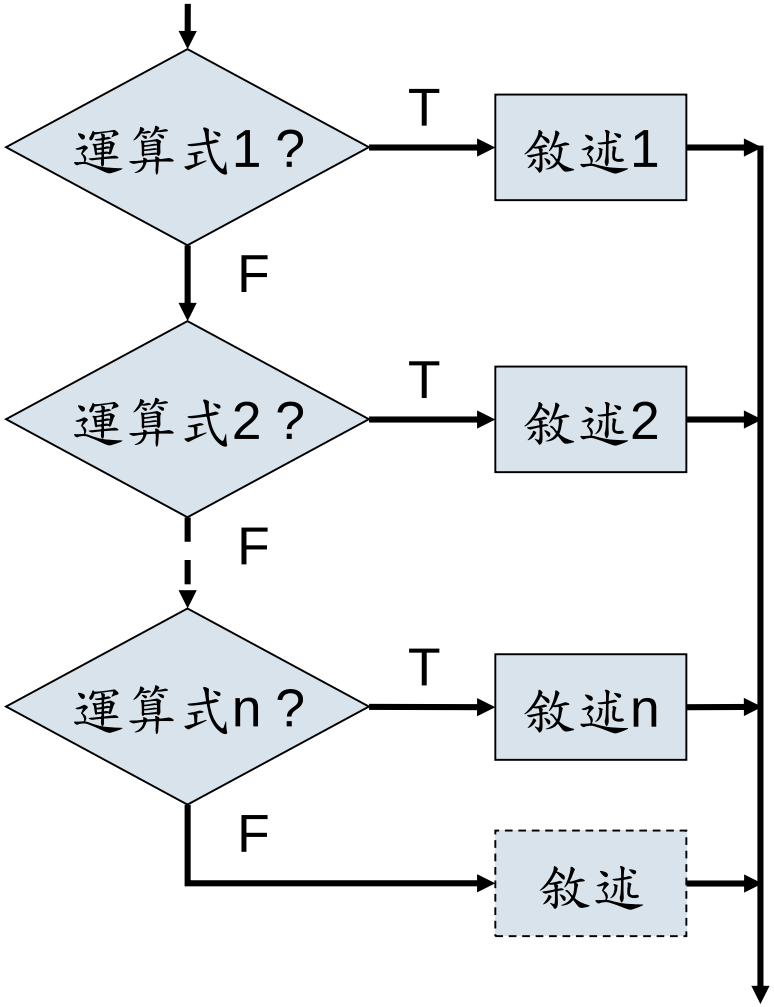


  - 若只有一行敘述，{}可省略。

選擇結構：if – else 敘述

- 格式三：

```
if(運算式1){
      敘述1;
        :
}else if(運算式2){
      敘述2;
        :
}else if(運算式n){
      敘述n;
        :
}else{
      敘述;
}
```



選擇結構：if – else 敘述

- 範例：輸入一整數，若大於0則印出"這是正數"的訊息。

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
    int num;   //宣告變數
    printf("請輸入數值：");   //提示訊息
    scanf("%d",&num);      //請求輸入

    if( num > 0 )
      printf("這是正數");
}
```

之後為節省畫面空間，這兩行會省略，請實作程式時自己加進去。

選擇結構：if – else 敘述

- 範例：輸入一整數，若大於0則印出"這是正數"的訊息，否則印出"這不是正數"。

```
void main(void)
{
    int num;   //宣告變數
    printf("請輸入數值：");   //提示訊息
    scanf("%d",&num);      //請求輸入

    if( num > 0 )
      printf("這是正數");
    else
      printf("這不是正數");
}
```

選擇結構：if – else 敘述

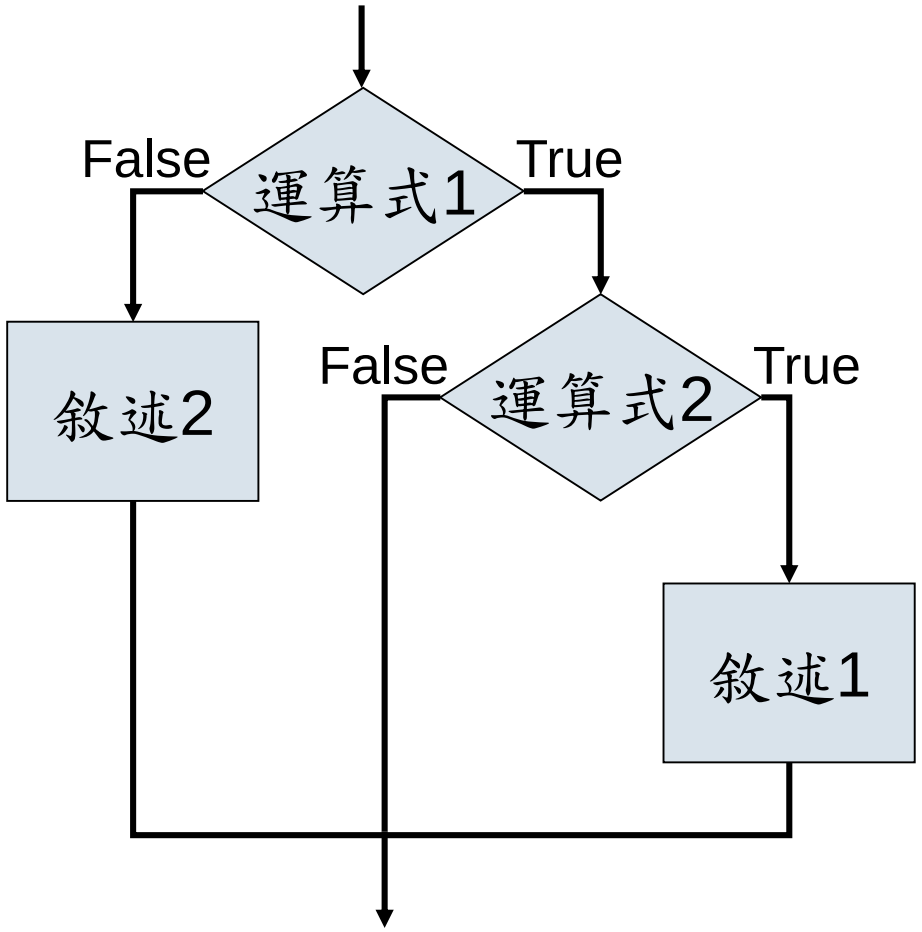
- 範例：演唱會門票一張$100元，一次購買五張(含)以上打九折，五張以下不打折，計算出總價。

```
void main(void) {
    int price=100, count=0;     //宣告變數並指定初值

    printf("每張票定價100元\n");  //顯示提示訊息
    printf("請輸入購買票數(1張以上):");
    scanf("%d",&count);    //等待輸入

    if(count>=5) {    //判斷是否五張以上
        printf("購買五張以上，打九折優惠!!\n");
        printf("總價為%f\n", count*price*0.9);
    }

    if(count>=1 && count<=4)  //介於1~4張之間
        printf("總價為%d\n", count*price);

    if(count<=0)  //要判斷每一種可能發生的狀況
        printf("很抱歉，您未輸入正確票數\n");
}
```

選擇結構：if – else 敘述

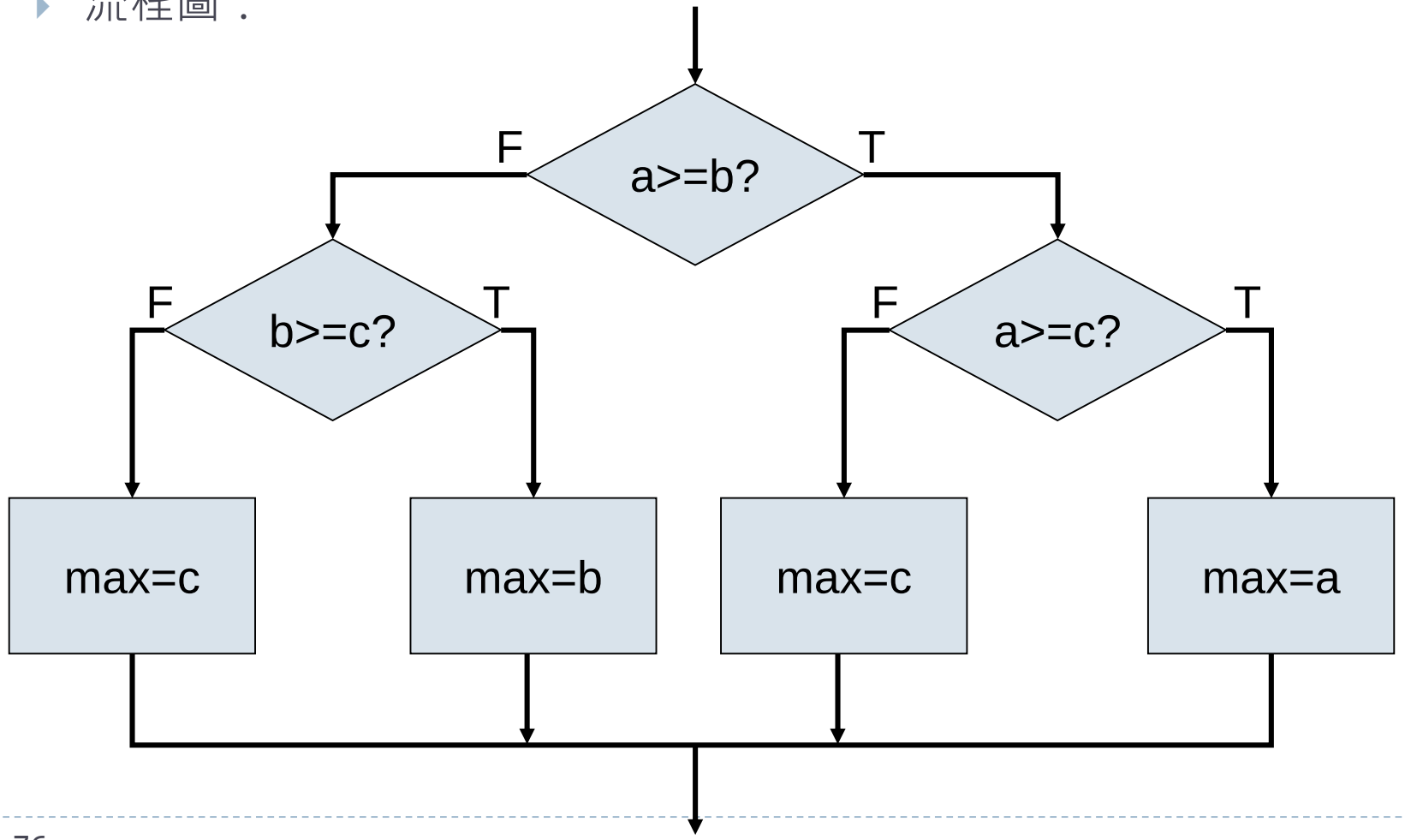
- 巢狀結構：if裡面還有if。
- 格式：

```
if(運算式1)
{
    if(運算式2)
        敘述1;
        :
}
else
    敘述2;
```



選擇結構：if – else 敘述

- 練習：找出a、b、c三數中的最大值，存入變數max。
  - 流程圖：



選擇結構：if – else 敘述

- 範例：找出a、b、c三數中的最大值，存入變數max。
  - 程式1:

```
if(a >= b)
    if(a >= c)
        max=a;
    else
        max=c;
else
    if(b >= c)
        max=b;
    else
        max=c;
```

程式2:

```
max = (a>=b) ? a : b;
max = (max>c) ? max : c;
```

選擇結構：if – else 敘述

- 範例：請使用者輸入三個整數，印出最大值。

```
void main(void) {
    int a, b, c, max;
    printf("請輸入三個整數(以空白隔開)：");
    scanf("%d %d %d", &a, &b, &c);

    if(a >= b)
        if(a >= c)
            max=a;
        else
            max=c;
    else
        if(b >= c)
            max=b;
        else
            max=c;
    printf("最大值是：%d\n", max);
}
```

選擇結構：if – else 敘述

- 善用條件運算子：
  - 判斷一輸入為正數、零或負數。

```
int main(void){
    int n;
    printf("請輸入一整數:");
    scanf("%d", &n);

    printf("%s",(n>0)?"正數":((n==0)?"零":"負數"));

    return 0;
}
```

多重選擇結構：if 敘述

- 寫一程式判斷成績等第。90~100分為優等，80~89分為甲等，70~79為乙等，60~69分為丙等，60分以下為丁等。

```
int main(void){
    int n;
    printf("請輸入成績:");
    scanf("%d", &n);
    if(n>=90 && n<=100)
        printf("優等");
    else if(n>=80 && n<=89)
        printf("甲等");
    else if(n>=70 && n<=79)
        printf("乙等");
    else if(n>=60 && n<=69)
        printf("丙等");
    else if(n>=0 && n<=59)
        printf("丁等");
    else
        printf("不正確的分數");
    return 0;
}
```

一個使用if-else的多重選擇的範例。

多重選擇結構：switch 敘述

- 在進行多重選擇時，過多的if-else常會造成困擾，可用switch敘述取代。

- 格式：

```
switch(運算式)
{
   case 常數1:
      敘述1;
      break;
   case 常數2:
      敘述2;
      break;
         :
   case 常數n:
      敘述n;
      break;
   default :
      敘述d;
}
```

運算式的結果其值必須為整數或字元。

而case後面的常數必須為整數或字元。

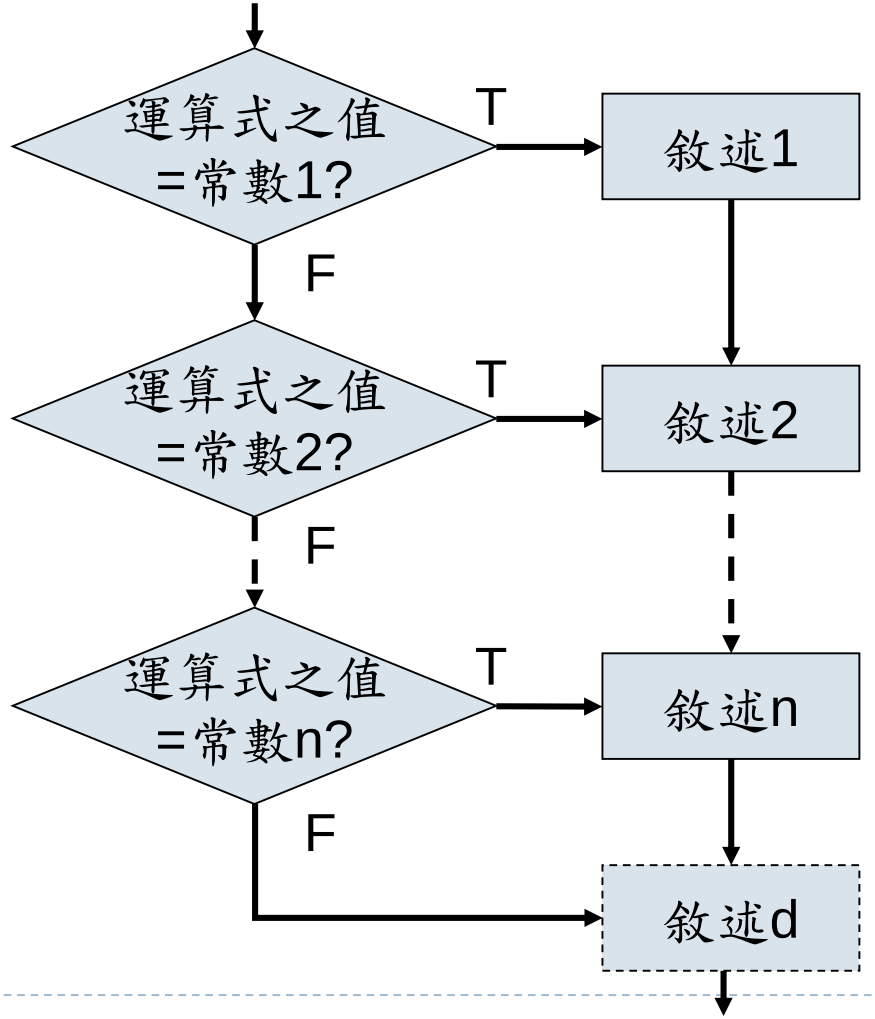
default敘述可省略

多重選擇結構：switch 敘述

- switch敘述執行過程：
  - 先求出運算式的值，然後與case後的常數進行比對。
  - 若找到相同的值，則執行其後面的敘述，執行過後底下的每一個case內之敘述也將陸續被執行，除非有break敘述才會結束。
  - 若找不到吻合的常數值，則：
    - 有default則執行default後的敘述。
    - 沒有default則跳出switch敘述。
  - 繼續switch之後的下一個敘述。

多重選擇結構：switch 敘述

- switch敘述執行過程：



一但進入敘述執行，則以下的所有敘述都將被執行到

default敘述可有可無

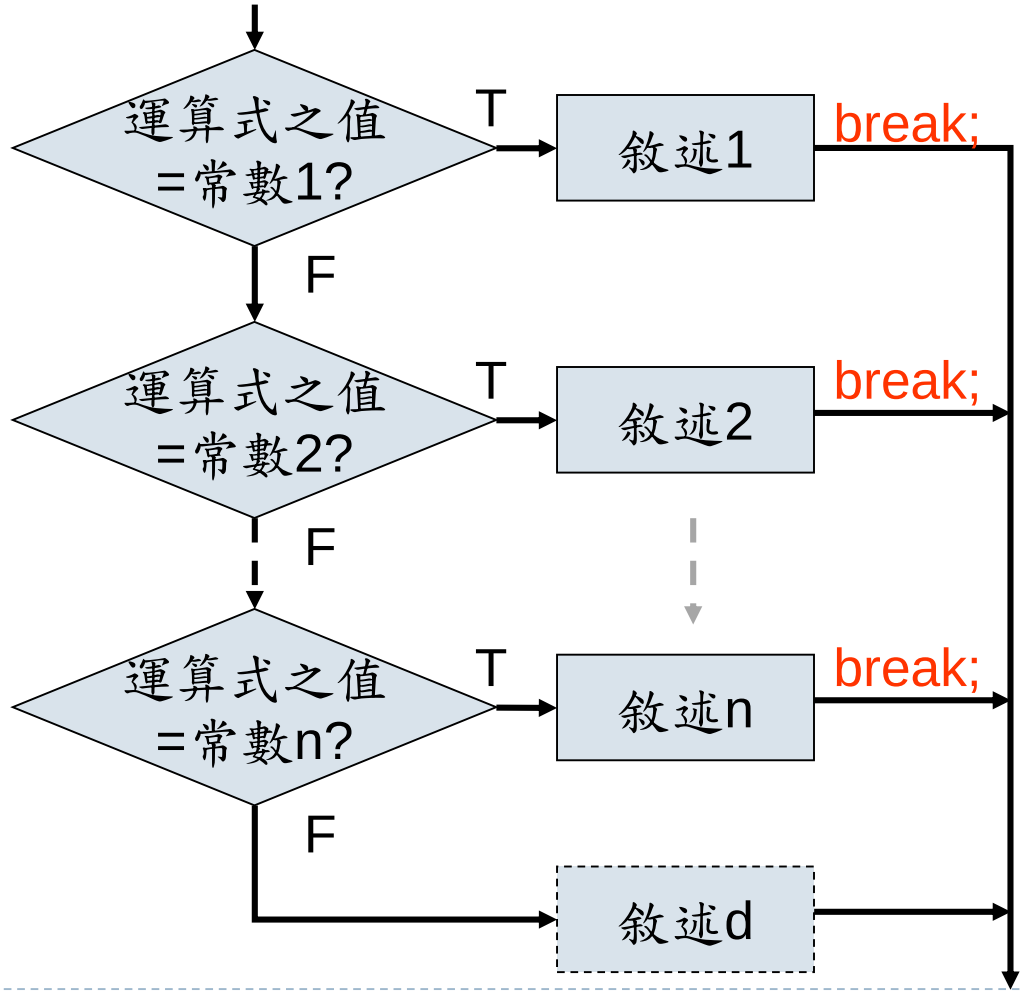
多重選擇結構：switch 敘述

- 範例：


```
void main(void)
{
    char ch;
    printf("請輸入一個字母(A/B/C):");
    scanf("%c",&ch);

    switch(ch)
    {
       case 'A': printf("It is an Apple\n");
       case 'B': printf("It is a Bed\n");
       case 'C': printf("It is a Cat\n");
       default: printf("輸入錯誤\n");
    }
}
```

多重選擇結構：switch 敘述

- 加入break敘述：

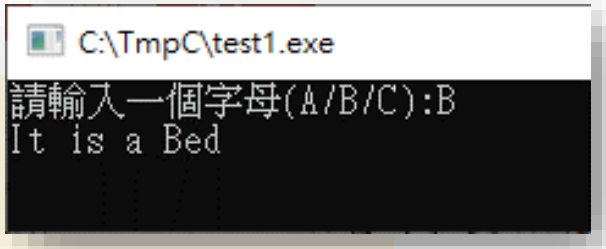


一但有break敘述,執行完後即離開switch,不再往下執行

default敘述可有可無,由於是最後一道敘述,故不需加break

多重選擇結構：switch 敘述

- 範例：加上break。




```
void main(void) {
    char ch;
    printf("請輸入一個字母(A/B/C):");
    scanf("%c",&ch);
    switch(ch)
    {
        case 'A':
            printf("It is an Apple\n"); break;
        case 'B':
            printf("It is a Bed\n"); break;
        case 'C':
            printf("It is a Cat\n"); break;
        default: printf("輸入錯誤\n");
    }
}
```

多重選擇結構：switch 敘述

- 依輸入判斷是第幾季：

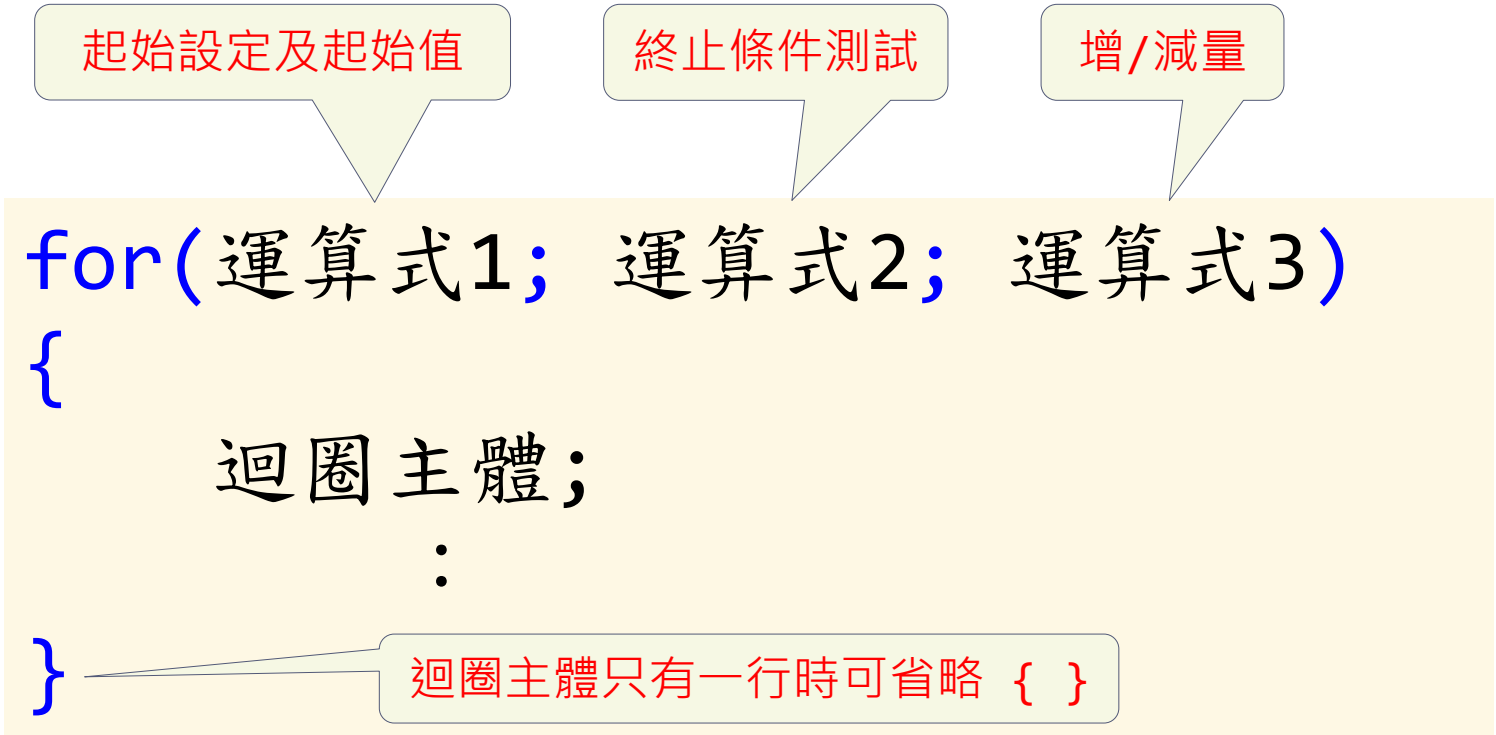
```
void main(void) {
    int m;
    printf("請輸入月份:");
    scanf("%d",&m);
    switch(m) {
        case 1:
        case 2:
        case 3: printf("第一季\n"); break;
        case 4:
        case 5:
        case 6: printf("第二季\n"); break;
        case 7:
        case 8:
        case 9: printf("第三季\n"); break;
        case 10:
        case 11:
        case 12: printf("第四季\n"); break;
        default: printf("輸入錯誤\n");
    }
}
```

case後面只能是整數或字元

休息一下~

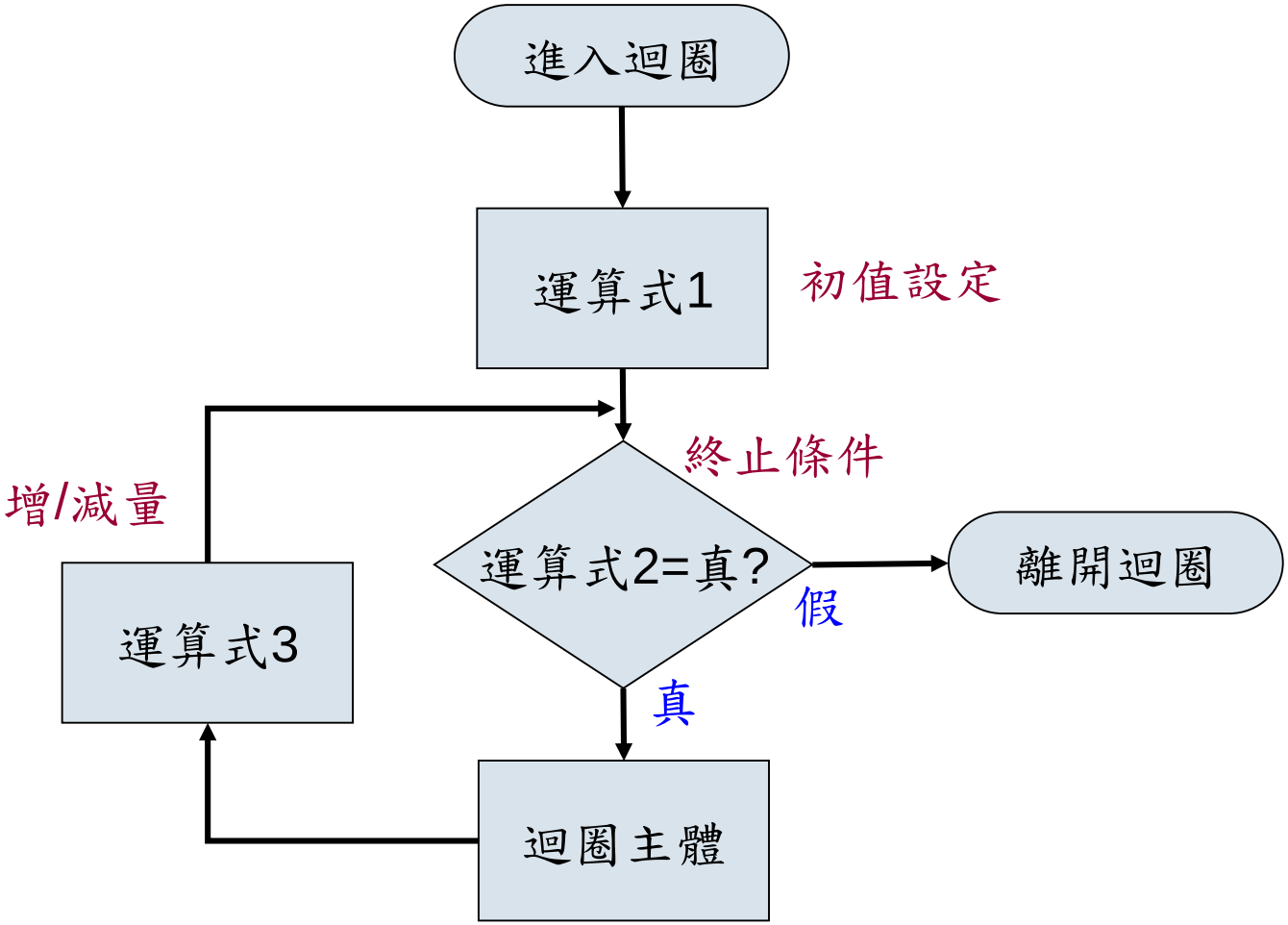
重複結構：for 迴圈

- 迴圈：當程式需要重覆執行某一段落時。
- 格式：



重複結構：for 迴圈

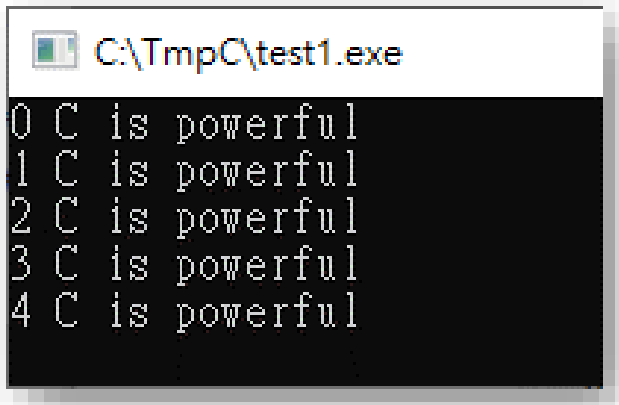
- for迴圈流程圖：



重複結構：for 迴圈

- 範例：在螢幕上列印5次"C is powerful"。

```
void main(void)
{
    int n;
    for(n=0; n<5; n++)
        printf("%d C is powerful\n", n);
}
```



重複結構：for 迴圈

- 下列迴圈各執行幾次?

① `for(i=0; i<10; i++)`

Ans: i=0,1,2,…9　計10次(當i=10時跳出迴圈)

② `for(j=10; j>0; j-=3)`

Ans: j=10,7,4,1　計4次

③ `for(k=0;  ; k++)`

Ans: 可接受無終止條件的情況,為無盡迴圈

④ `for(m=0; m<10;  )`

Ans: 可接受無增/減量,為無盡迴圈

⑤ `for(m=1,n=10; n>0; m+=3,n-=2)`

Ans: 由n控制迴圈次數，n=10,8,6,4,2　計5次。

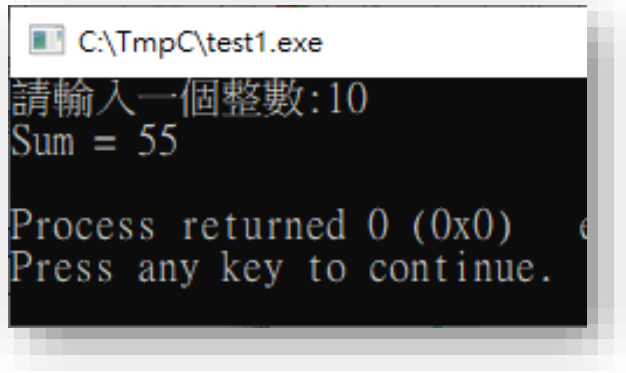
m無控制上的意義，但會隨迴圈變更其值。

重複結構：for 迴圈

- 範例：寫一程式累加至使用者指定之數目。

```
void main(void)
{
    int s=0,n;

    printf("請輸入一個整數:");
    scanf("%d", &n);

    for(int i=1; i<=n; i++)
        s += i;

    printf("Sum = %d\n", s);
}
```

如果i只在for迴圈用到，可以在for裡設定初值時宣告，迴圈結束即丟棄。

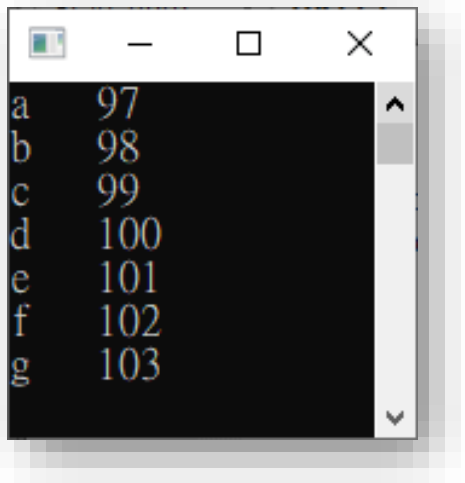


重複結構：for 迴圈

- 範例：寫一程式印出字元'a'到'g'的ASCII值

```
void main(void)
{
    char ch;
    for(ch='a'; ch<='g'; ch++)
        printf("%c    %d\n",ch, ch);
}
```

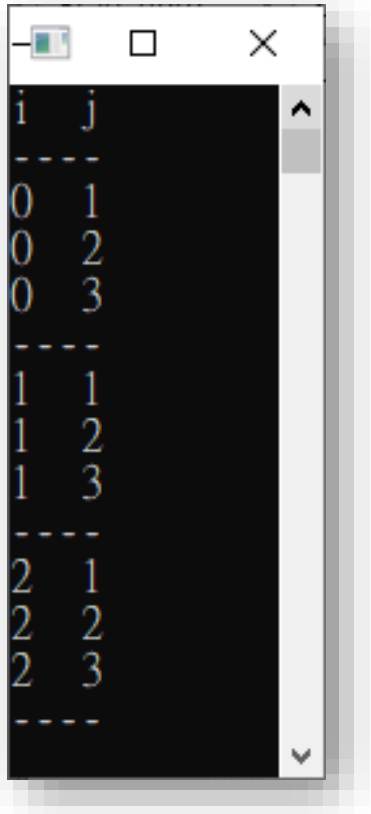
同一個變數,
第一次以**文字**方式印出,
第二次以**數值**方式印出



重複結構：for 迴圈

- 範例：寫一程式印出迴圈變數的變化過程。

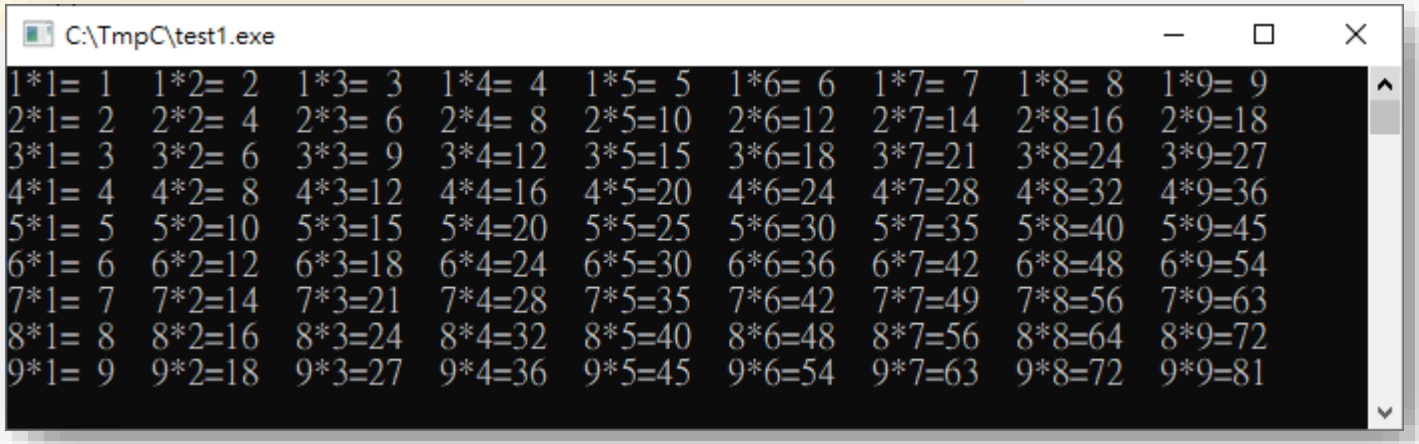
```
void main(void)
{
    int i,j;
    printf("i  j\n----\n");
    for(i=0; i<3; i++)
    {
        for(j=1; j<=3; j++)
            printf("%d  %d\n", i, j);
        printf("----\n");
    }
}
```



重複結構：for 迴圈

- 範例：經典的9X9乘法表來了~

```
void main(void)
{
    for(int i=1; i<=9; i++)
    {
        for(int j=1; j<=9; j++)
            printf("%d*%d=%2d  ", i, j, i*j);
        printf("\n");
    }
}
```



重複結構：while 迴圈

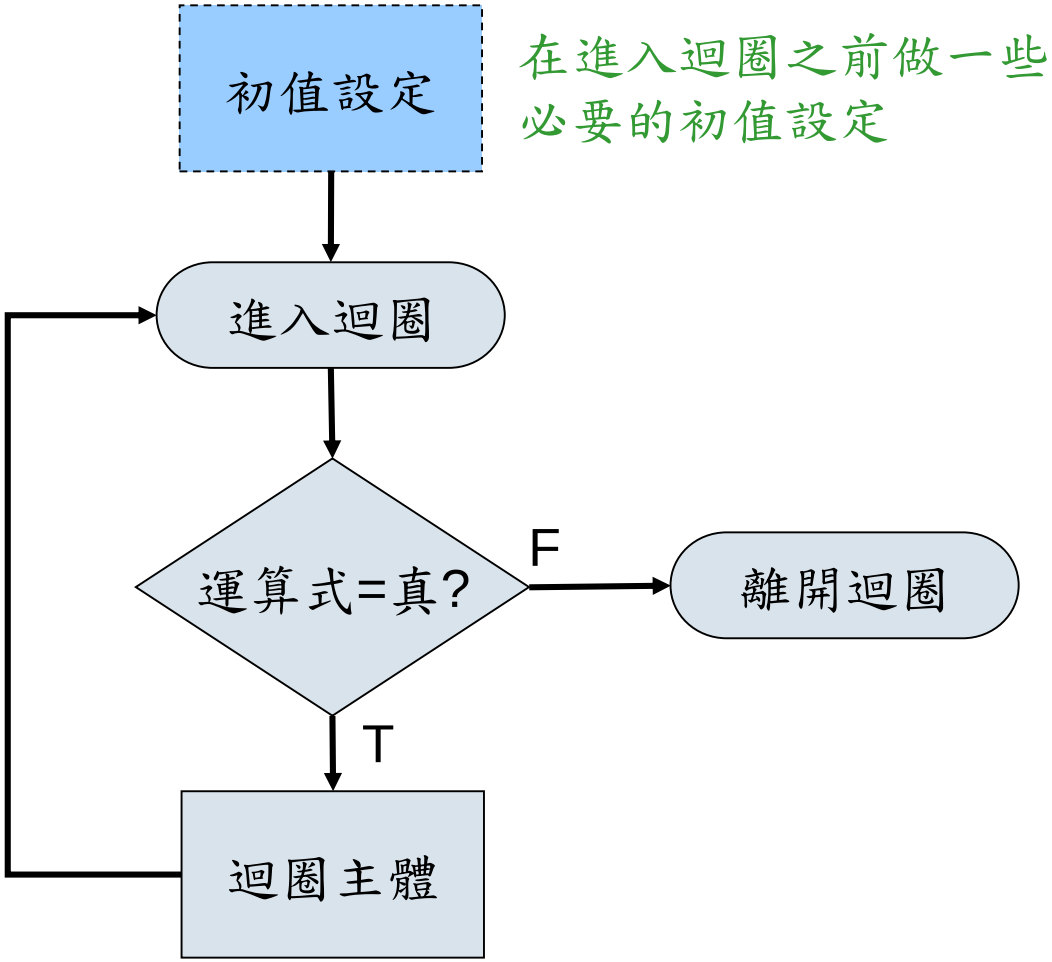
- 若不確定要重複執行幾次，可使用while指令。
  - 格式：

```
while( 運算式 )
{
      迴圈主體;
          :
}
```

  - 執行步驟：
    - Step 1: 測試運算式的值。
    - Step 2: 若結果為真→執行迴圈主體→回到Step1。
      若結果為假→跳出迴圈。

重複結構：while 迴圈

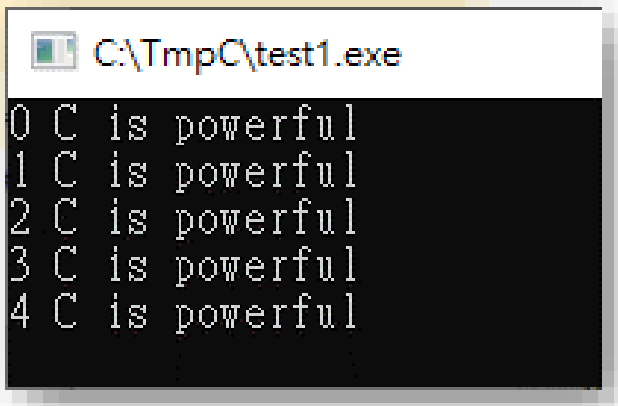
- while迴圈流程圖：



重複結構：while 迴圈

- 範例：在螢幕上列印5次"C is powerful"。

```
void main(void)
{
    int n=0;    //設定初值
    while( n<5 )    //結束條件
    {
        printf("%d C is powerful\n", n);
        n++;    //增減量
    }
}
```



重複結構：while 迴圈

- 範例：由鍵盤輸入n個數值，直到輸入999為止(999不算)，將輸入之數值加總印出。

```
void main(void)
{
    int x=0, sum=0;

    //x=999就會結束迴圈
    while( x!=999 )
    {
        sum += x;
        printf("請輸入數值：");
        scanf("%d", &x);
    }
    printf("Sum = %d", sum);
}
```

重複結構：while 迴圈

- while的無盡迴圈：

```
while( 1 )
{
      迴圈主體;
          :
}
```

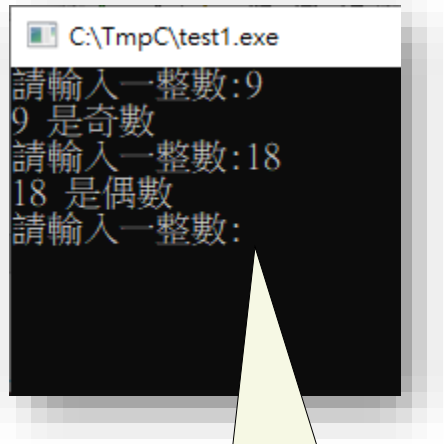
非零的常數恆為真，故while迴圈將無止境的循環執行下去

(看來奇怪，卻還蠻常用的)

重複結構：while 迴圈

- 範例：輸入一整數並判斷是奇數還是偶數。

```
void main(void)
{
    int num;
    while( 1 )  //無盡迴圈
    {
        printf("請輸入一整數:");
        scanf("%d",&num);
        if( num % 2 == 0 )
            printf("%d 是偶數\n", num);
        else
            printf("%d 是奇數\n", num);
    }
}
```



在此處等待輸入，程式不會結束。
(可用Ctrl+C中斷程式)

重複結構：do - while迴圈

- 先執行一次迴圈主體，再判斷要不要重複。
  - 格式：

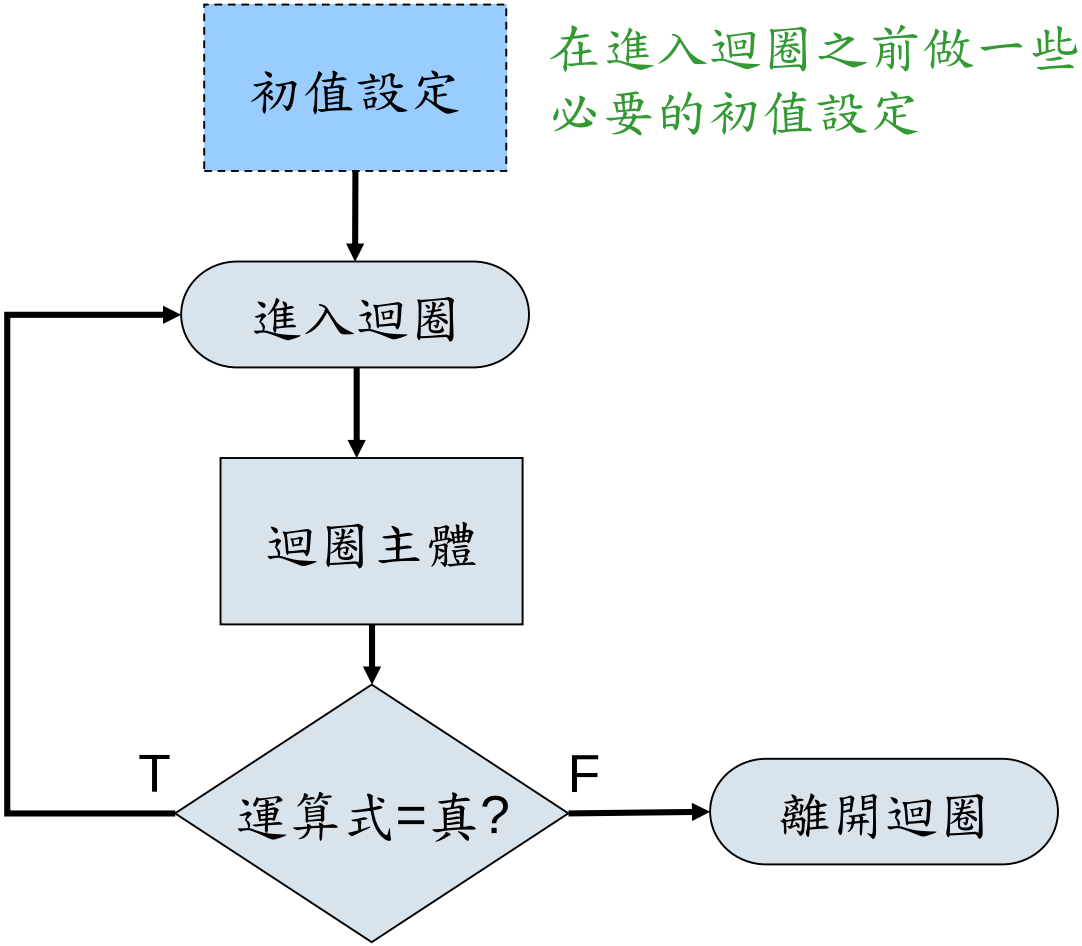
```
do
{
      迴圈主體;
          :
} while( 運算式 );
```

注意結尾要有分號

  - 執行步驟：
    - Step 1: 執行迴圈主體(至少執行一次)。
    - Step 2：測試條件運算式的值。
    - Step 3: 若為真→回到Step1。
      若為假→跳出迴圈。

重複結構：do - while迴圈

- while迴圈流程圖：

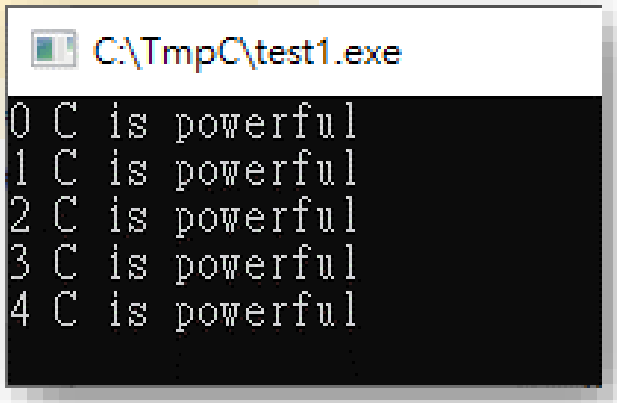


重複結構：do - while迴圈

- 範例：在螢幕上列印5次"C is powerful"。

```
void main(void)
{
    int n=0;
    do
    {
        printf("%d C is powerful\n",n);
        n++;
    }while(n<5);
}
```

重要，不要漏掉了!



重複結構：do - while迴圈

- 範例：由鍵盤輸入n個數值，直到輸入999為止(999不算)，將輸入之數值加總印出。

```
void main(void)
{
    int x=0, sum=0;

    //x=999就會結束迴圈
    do
    {
        sum += x;
        printf("請輸入數值：");
        scanf("%d", &x);
    } while( x!=999 );
    printf("Sum = %d", sum);
}
```

重複結構：do - while迴圈

- 我們做了三次印出"C is powerful"，請比較有何不同？
  - for
  - while
  - do while

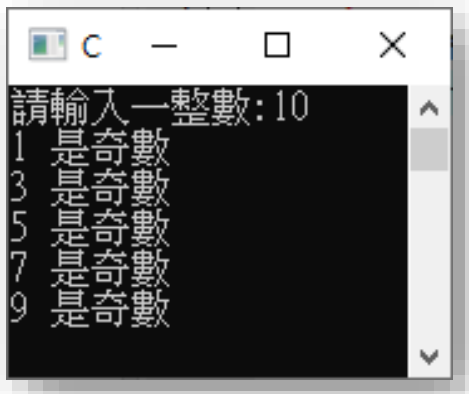
continue與break敘述

- 與重複結構配合的指令：
  - break敘述：用以跳出所在的那層迴圈(也用於結束switch敘述)。
  - continue敘述：當執行到continue敘述時，程式會略過迴圈中其餘的部分而直接回到該層迴圈的起始處。

continue與break敘述

- 範例：輸入一整數，印出其中的奇數。

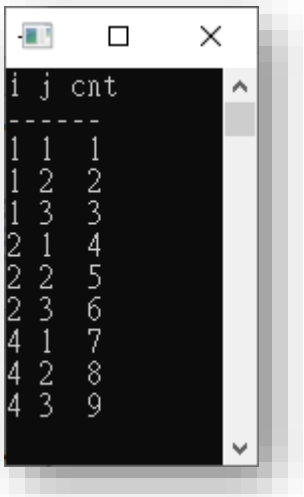
```
void main(void)
{
    int num;
    printf("請輸入一整數:");
    scanf("%d", &num);
    for(int i=0; i<=num; i++)
    {
        if(i%2 == 0) continue;
        printf("%d 是奇數\n", i);
    }
}
```



continue與break敘述

- 範例：追蹤下列程式，最後變數cnt為何值？

```
void main(void)
{
    int i,j,cnt=0;
    printf("i j cnt\n------\n");
    for(i=1;i<=4;i++)
    {
        for(j=1;j<=5;j++)
        {
            if(i==3) continue;
            if(j>3) break;
            cnt++;
            printf("%d %d  %d\n",i,j,cnt);
        }
    }
}
```



continue與break敘述

- 注意continue在while指令與do...while指令返回位置稍有不同：

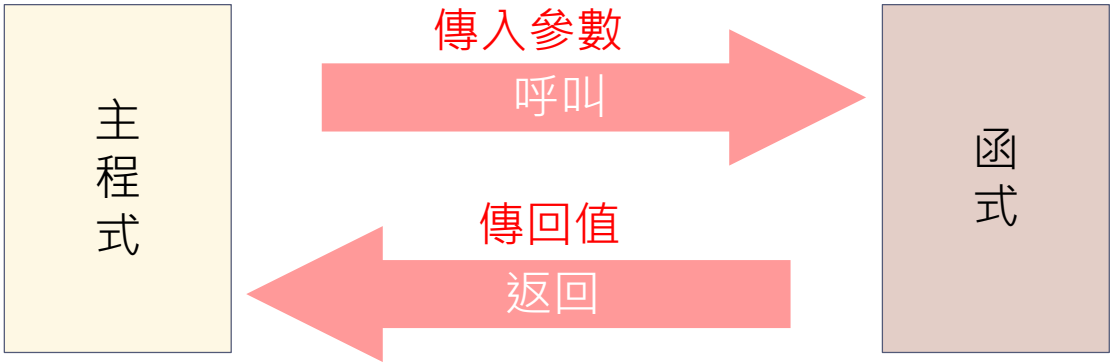
```
while(運算式)
{
        :
    continue;
        :
}
```

```
do
{
        :
    continue;
        :
}while(運算式);
```

休息一下~

函式(Function)

- 函式(Function)，亦稱函數。
- 分為：
  - 內建函式：C語言提供可立即呼叫使用的函式。
  - 自訂函式：自己寫的函式。
- 函式的好處是可以讓程式簡潔，易於維護。



內建函式

- C語言編譯器提供的已定義好的函式集合稱為「標準函式庫」，像最常用的printf()、scanf()等，使用時用#inlude命令引入即可。
- 因為太常用，所以我們固定會引入這兩項：

```
#include <stdio.h>
#include <stdlib.h>
      :
```

- 有些編譯器你若不寫這兩行它也會自動引入。

內建函式

- 常用函式庫：

標頭檔	函式種類
stdio.h	輸出入函式 printf, scanf, gets, puts, getchar, putchar, fclose, feof, fgets, fputs, fread, fopen, fwrite, rewind, abs,
stdlib.h	轉換函式 atof, atoi, atol, log, itoa, pow, rand, random, randomize, srand
string.h	字串函式 strcmp, strcpy, strcat, strlen, strtok
ctype.h	字元函式 isalnum, isalpha, isdigit, ispunct, isspace
math.h	數學函式 acos, asin, atan, atof, poly, ceil, cos, div, exp, floor, sin, sqrt, tan
time.h	時間函式 time, gettime, getdate,settime, setdate
io.h	檔案輸出入函式 eof, open, close, filelength, write, read
conio.h	本文視窗顯示函式 getche, getch, clrscr

內建函式

- 數學函式(太多了，僅示範部分)：

函式名稱	功能說明
fabs	語法：double fabs(double x) 說明：傳回浮點數 x 的絕對值。 例：fabs(-12.345) 傳回 12.345
pow	語法：double pow(double x, double y); 說明：傳回 x 的 y 次方(x^y)值。 例：pow(2, 3) 傳回 8
sqrt	語法：double sqrt(double x); 說明：傳回 x 的平方根($\sqrt{x}$)值。 例：sqrt(25) 傳回 5
hypot	語法：double hypot(double x, double y); 說明：傳回 $\sqrt{x^2 + y^2}$ 的值。 例：hypot(3, 4) 傳回 $\sqrt{3^2 + 4^2} = \sqrt{25} = 5$

sin
cos
tan
	語法：double sin(double x);
	double cos(double x);
	double tan(double x);
	說明：傳回三角函數值，x 以弳度量(弧度)為單位。
	弳度量 = 角度量*(π/180)
	例：90° (角度量)= 90 * (π/180) = 1/2 π ≒ 1.5708 弳(弳度量)

內建函式

- 亂數函式(這個好用)：

函式名稱	功能說明
rand	語法：int rand(void); 說明：產生介於 0 到 32767 之間的隨機亂數。
srand	語法：void srand(unsigned seed); 說明：亂數產生器的種子，在使用 rand 函式之前要先啟動。若 seed 值沒更換，則每次產生的亂數是一樣的。一般會配合時間值的變化當亂數種子，以求得每次產生的亂數是不同的，如下例： 例：srand((unsigned) time(NULL));　/* 須引用 time.h 標頭檔 */

內建函式

- 時間函式：

函式名稱	功能說明
time	語法：time_t time(time_t *timeptr); 說明：取得系統目前的時間，time() 函式傳回自 1970/1/1 00:00:00 到目前系統所經過的秒數。由於數目會很大，所以使用長整數型別。
clock	語法：clock_t clock(void); 說明：取得 CPU 自從程式啟動所到的 Ticks(振盪時間) 。 1 秒 = 1000 Ticks，1 Ticks = 0.001 秒 = 10^{-3} 秒
difftime	語法：double difftime(time_t t2, time_t t1); 說明：取得 t2-t1 的時間差，單位為秒數。
ctime	語法：char *ctime(const time_t *timeptr); 說明：將時間秒數轉換成時間字串。

內建函式

- 轉換函式：

函式名稱	功能說明
atof	語法：double atof(const char *string); 標頭檔：math.h 說明：將字串常數 string 轉換成倍精確浮點數。
atoi	語法：int atoi(const char *string); 標頭檔：stdlib.h 說明：將字串常數 string 轉換成整數。
atol	語法：long atol(const char *string); 標頭檔：stdlib.h 說明：將字串常數 string 轉換成長整數。
toasscii	語法：int toasscii(int c); 標頭檔：ctype.h 說明：將字元轉換成 ASCII 字元。
toupper	語法：int toupper(int c); 標頭檔：ctype.h 說明：將字元轉換成大寫英文字母。
tolower	語法：int tolower(int c); 標頭檔：ctype.h 說明：將字元轉換成小寫英文字母。

內建函式

- 以上只是各類函式中的一點點而已，C的內建函數有一本書那麼多，去買書查閱或問谷歌大神吧。
- 呼叫方式：

```
變數 = 內建函式名稱(參數1, 參數2, ...);
```

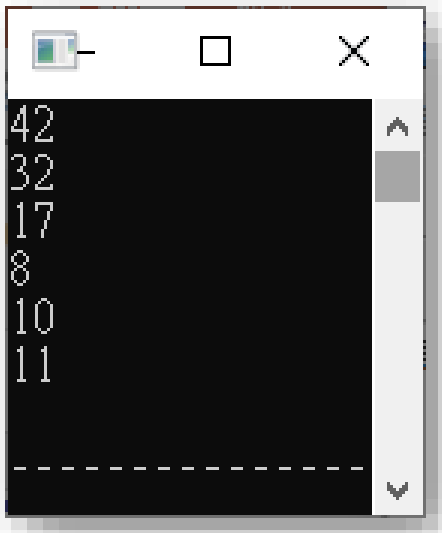
- 例如：

```
x = abs(-5);    //x會得到-5的絕對值
y = tolower('A')  //y會得到小寫a
```

內建函式

- 範例：取得6個1至49之間的亂數。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void main(void)
{
    int x;
    srand(time(NULL));  //亂數起始種子
    for(int i=1; i<=6; i++)
    {
        x = rand() % 49+1;  //取得亂數
        printf("%d\n", x);
    }
}
```



內建函式

- 亂數範圍的公式：
  - 如果要產生n1(頭)~n2(尾)之間的整數，其公式為：

```
rand() % (n2 - n1 + 1) + n1
```

  - 例：要產生20~59之間的亂數：

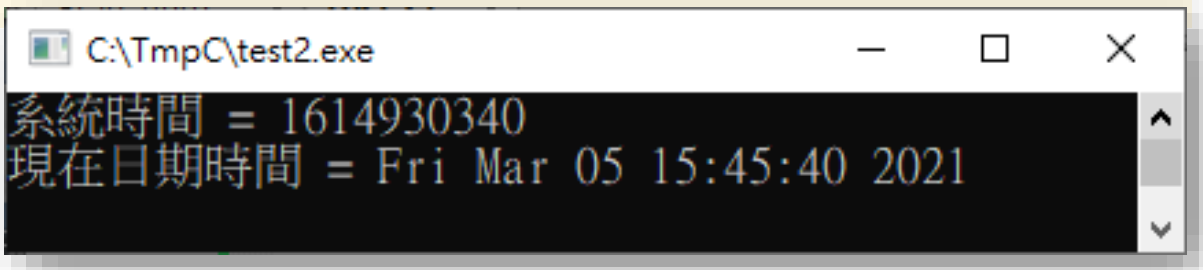
```
rand() % (59 - 20 + 1) + 20
```

```
= rand() % 40 + 20
```

內建函式

- 範例：取得系統現在日期時間。

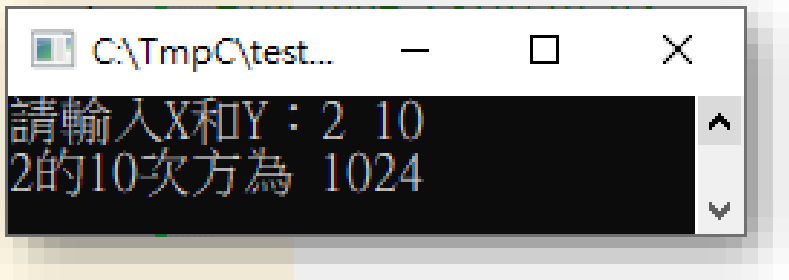
```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void main(void)
{
    time_t now;     //now變數宣告為時間型態
    now = time(NULL);  //取得現在的日期時間
    printf("系統時間 = %d\n", now);
    printf("現在日期時間 = %s", ctime(&now));
}
```



內建函式

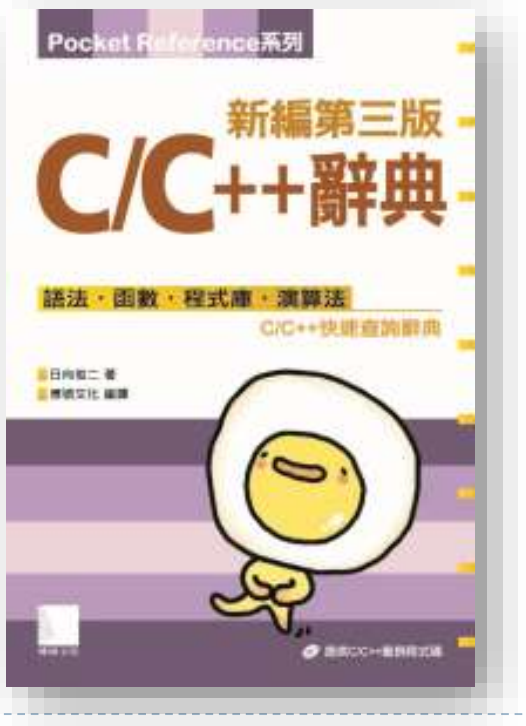
- 範例：輸入X和Y，計算X^Y。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

void main(void)
{
    int X, Y;
    double Z;
    printf("請輸入X和Y：");
    scanf("%d %d", &X, &Y);
    Z = pow(X, Y);
    printf("%d的%d次方為 %.0f",X, Y, Z);
}
```



內建函式

- 內建函式多不勝數，早期會在手邊準備一本類似辭典的書來查閱，現在谷歌一下應該就會有相關使用資訊了。



(範例，非廣告)

自訂函式

- 是指自己定義的函式。
  - 格式：

```
傳回值型態 函式名稱(參數1,參數2,...)
參數宣告;
{
        函式主體;
}
```

  - 例：

```
void test(x, y)
int x, y;
{
   printf("%d", x+y);
}
```

或（參數宣告位置不同）

```
void test(int x, int y)
{
   printf("%d", x+y);
}
```

自訂函式

- 範例：無傳入參數亦無傳回值的函數。

主程式

```
void main()
{
    star();
    printf("IBM-PC");
    star();
}
```

呼叫

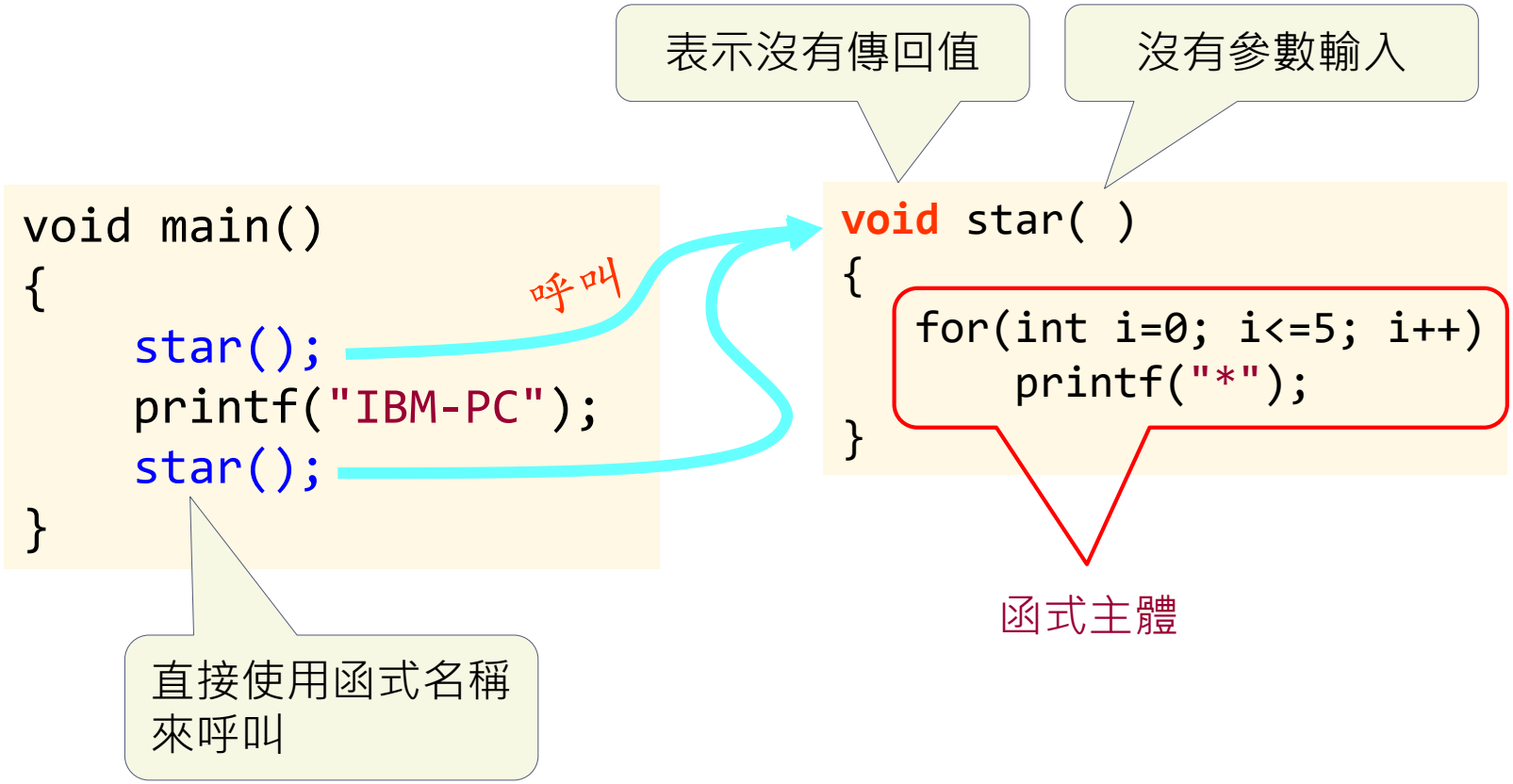
自訂函式

```
void star()
{
   for(int i=0; i<=5; i++)
       printf("*");
}
```

- 執行結果：

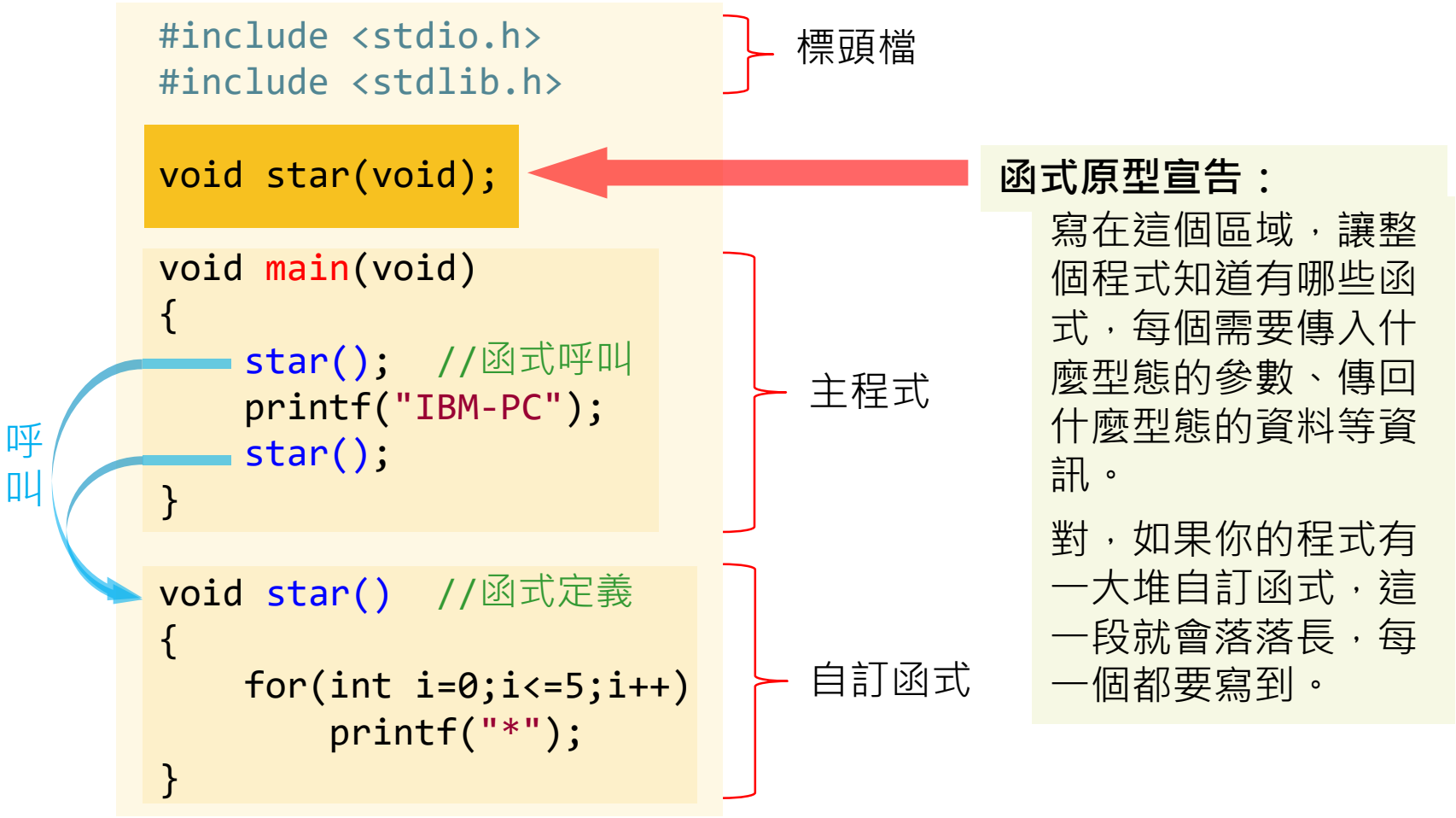
自訂函式

- 說明：


表示沒有傳回值
沒有參數輸入
void main()
{
star();
printf("IBM-PC");
star();
}
呼叫
void star( )
{
for(int i=0; i<=5; i++)
printf("*");
}
函式主體
直接使用函式名稱來呼叫

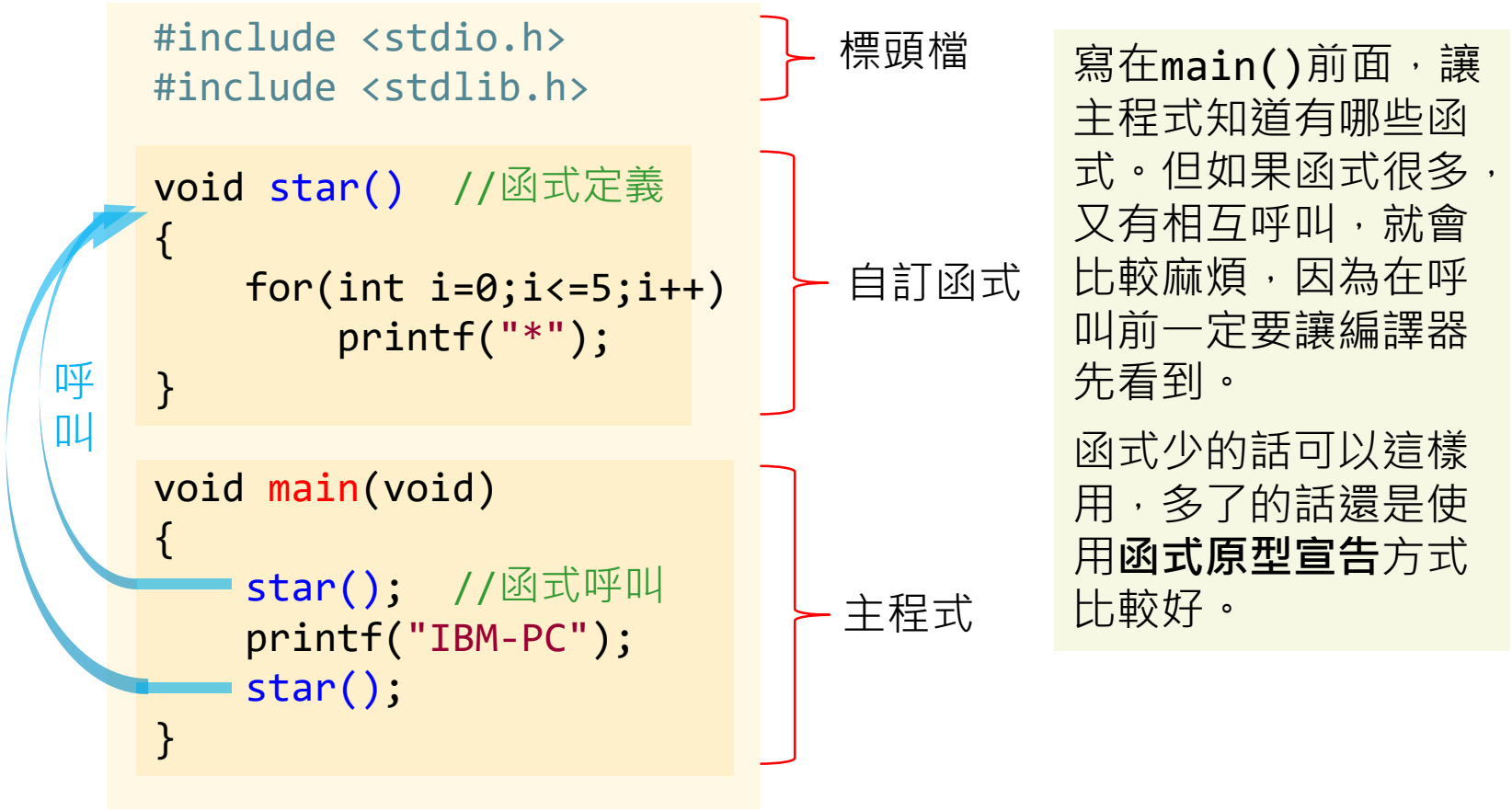
自訂函式

- 比較嚴謹的寫法：



自訂函式

- 如果不寫函式原型宣告，那函式就必須放在main()前面：



寫在main()前面，讓主程式知道有哪些函式。但如果函式很多，又有相互呼叫，就會比較麻煩，因為在呼叫前一定要讓編譯器先看到。

函式少的話可以這樣用，多了的話還是使用**函式原型宣告**方式比較好。

自訂函式

- 範例：傳入參數。

```
#include <stdio.h>
#include <stdlib.h>

void star(int);

main()
{
    for(int i=1;i<=10;i++)
        star(i);
}

void star(int num)
{
    for(int j=0; j<num; j++)
        printf("*");
    printf("\n");
}
```

函式原型宣告
函式名稱：star()
傳入參數：一個整數(int)
傳回值：無(void)

每次呼叫時將 i 的值
傳過去

由 num 接收
傳入的參數

呼叫

傳遞

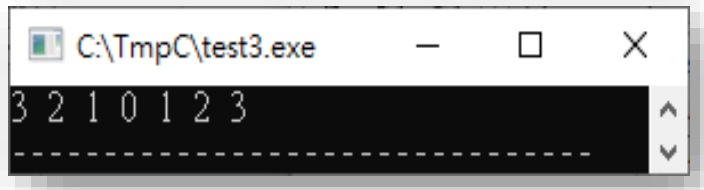
自訂函式

- 具傳回值的函式。
  - 在某些情況下，可能需要在函數中進行某些運算，並將結果傳回到主程式，可使用具有傳回值的函數。
  - 必須有變數或相關的敘述來接收函數的傳回值，以進行後續處理。
  - 使用return敘述來傳回值。
  - 由於return敘述只能傳回一個值，所以若要傳回多個值，可使用陣列或指標(後面會介紹)。

自訂函式

- 範例：計算絕對值的函數。

```
int ABS(int);  //函式原型宣告

void main(void)
{
    for(int i=-3; i<=3; i++)
        printf("%d ", ABS(i));
}
```



呼叫

傳遞

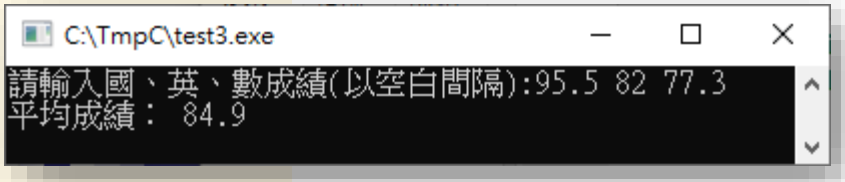
```
int ABS(int num)  //宣告傳回值是整數
{
    return (num<0) ? -num : num;
}
```

由return敘述
傳回結果

自訂函式

- 範例：輸入國英數成績，計算其平均值。

```
//函式原型宣告，傳回值為浮點數
float avg(float, float, float);

void main(void)
{
    float chi,eng,math;
    printf("請輸入國、英、數成績(以空白間隔):");
    scanf("%f %f %f",&chi, &eng, &math);
    printf("平均成績：%5.1f\n",avg(chi,eng,math));
}
```



呼叫　傳　遞

```
float avg(x, y, z)
float x, y, z;  //傳入之參數型態宣告
{
    float avescore = (x+y+z)/3;
    return(avescore);
}
```

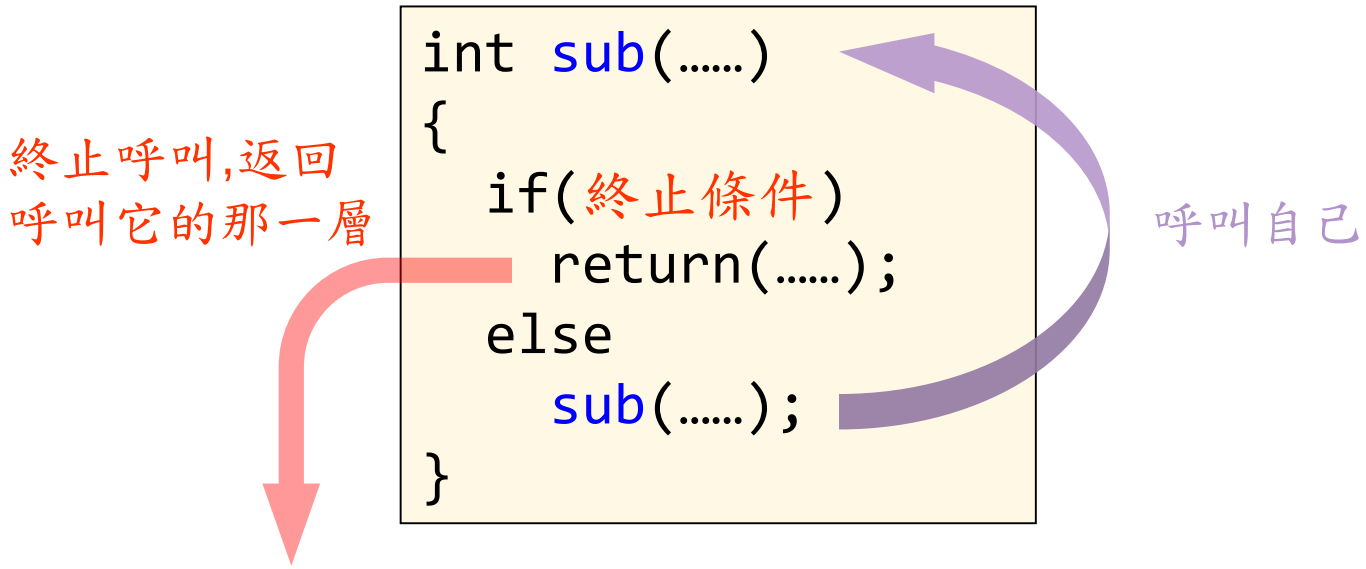
由return敘述
傳回結果

遞迴函式(Recursive)

- 當函式裡又呼叫自己時，這種情況稱為遞迴。
- 有些問題用遞迴可以得到更簡潔的程式，但不一定會有高效率。
- 使用遞迴時要注意：
  - 1.傳入的參數：要能符合要解決的問題。
  - 2.終止條件：錯誤或沒有終止條件，將造成無盡迴圈甚至當機。
- 經典應用：最大公因數 (GCD)、費波納西數列(Fibonacci Sequence)、河內塔 (Hanoi Tower)、N個字元的排列組合等。

遞迴函式(Recursive)

- 遞迴函數大多是這個模式：
  - (注意，它不是迴圈的概念!)

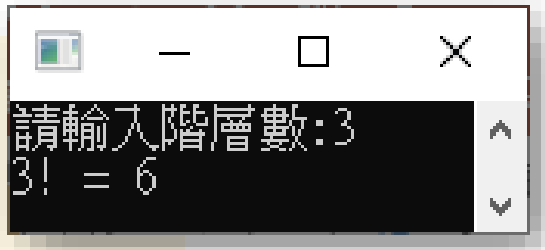


遞迴函式(Recursive)

- 範例：計算階層。

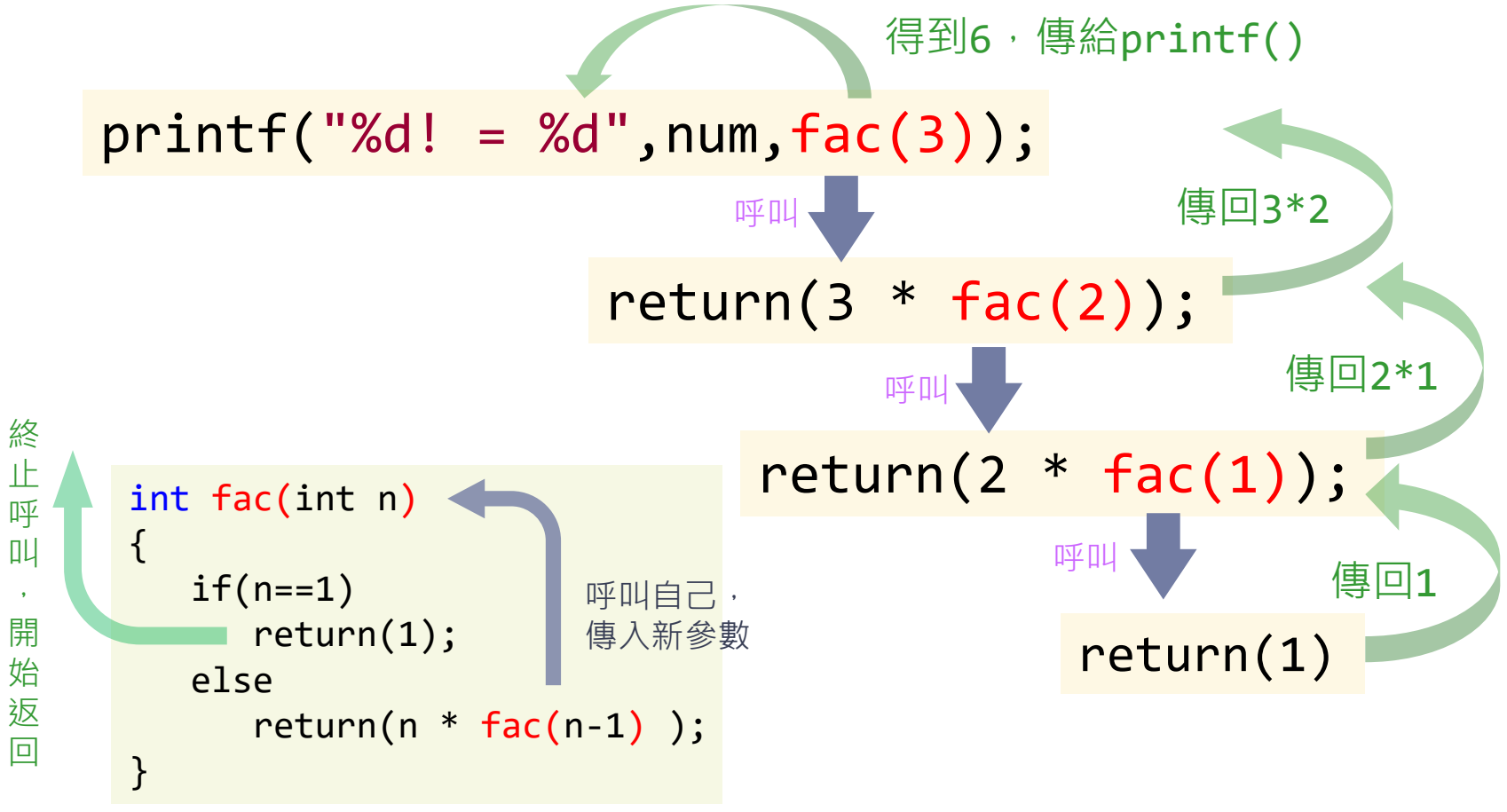
```
int fac();  //函式原型宣告

void main(void)
{   int num;
    printf("請輸入階層數:");
    scanf("%d", &num);
    printf("%d! = %d", num, fac(num));
}

int fac(int n)
{
    if(n==1)
        return(1);
    else
        return(n * fac(n-1) );
}
```

呼叫自己，
並傳入新參數值



遞迴函式(Recursive)

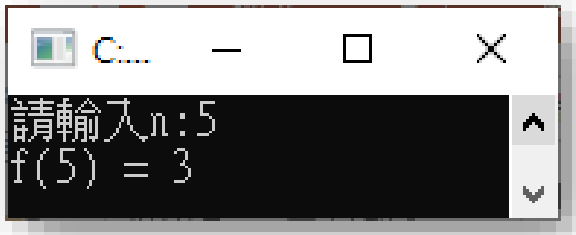
- 範例：計算階層遞迴過程說明(設num=3)。



遞迴函式(Recursive)

- 範例：求第n個費氏級數值(0,1,1,2,3,5,8,...)。

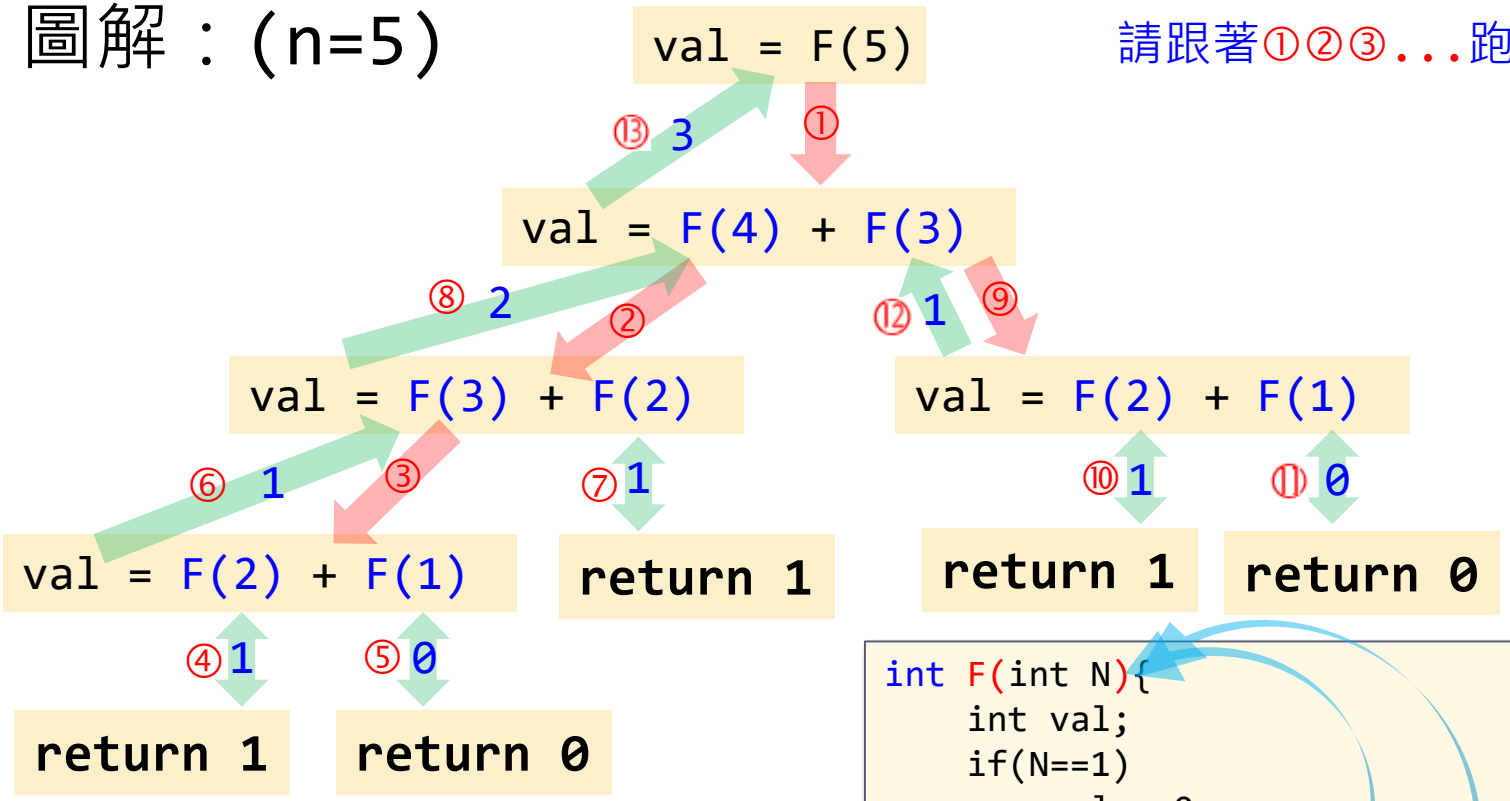
```
int F();  //函式原型宣告
void main(void){
    int n;
    printf("請輸入n:");
    scanf("%d", &n);
    printf("f(%d) = %d", n, F(n));
}
int F(int N)
{
    int val;
    if(N==1)
        val = 0;
    else if(N==2)
        val = 1;
    else
        val =  F(N-1) + F(N-2);
    return(val);
}
```



會分兩次呼叫自己

遞迴函式(Recursive)

- 圖解：(n=5)




```
int F(int N){
    int val;
    if(N==1)
        val = 0;
    else if(N==2)
        val = 1;
    else
        val =  F(N-1) + F(N-2);
    return(val);
}
```

- 最後傳回3

休息一下~

前置命令處理程式

- 置命令處理程式是對編譯器(Compiler)所下的指令，讓編譯器知道程式中的一些設定。

- 格式說明:
  - 1.以 # 字開頭。
  - 2.可放在程式任何地方，但通常在main()之前或某個函數之前。

- (前面出現過的#include就是)。

#define命令

- #define會使編譯器以標記字串取代原始程式檔中每個出現的識別碼。

- 目的：
  - 1.使程式容易閱讀
  - 2.容易修改

- 格式：

#define命令

- 例：

```
#define  PI  3.14159
```

  - 程式中有出現字串"PI"的地方，在編譯時都將被"3.14159"取代後再送去編譯。

- 例：

```
#define  MAX  100
```

  - 程式中有出現字串"MAX"的地方，都將被"100"取代。

#define命令

- 範例：計算球面積。

```
#include <stdio.h>
#define  PI  3.14159    //定義PI為3.14159

float area(float rad){
    return(4 * PI * rad * rad);
}   //球面積 = 4πr²

float area(float); //函式原型宣告

void main(void)
{
    float radius;
    printf("請輸入球半徑:");
    scanf("%f",&radius);
    printf("球面積是：%.2f",area(radius));
}
```

編譯時"3.14159"會取代"PI"

所以編譯器看到的會是:

```
return(4 * 3.14159 * rad * rad)
```

#define巨集

- 利用字串替代的特性定義簡單的巨集指令。
- 巨集會產生較多的程式碼，但執行的速度較函數快。

- 格式：　`#define　巨集名稱　單行敘述;`

- Ex:　
```
#define   ERROR   printf("\nError\n");
```

➔指令執行時ERROR將被printf()指令取代

#define巨集

- 範例：

原始碼

```
#define ERROR printf("\nError\n");
main()
{
        :
    if(x != y)
        ERROR
    else
        :
}
```

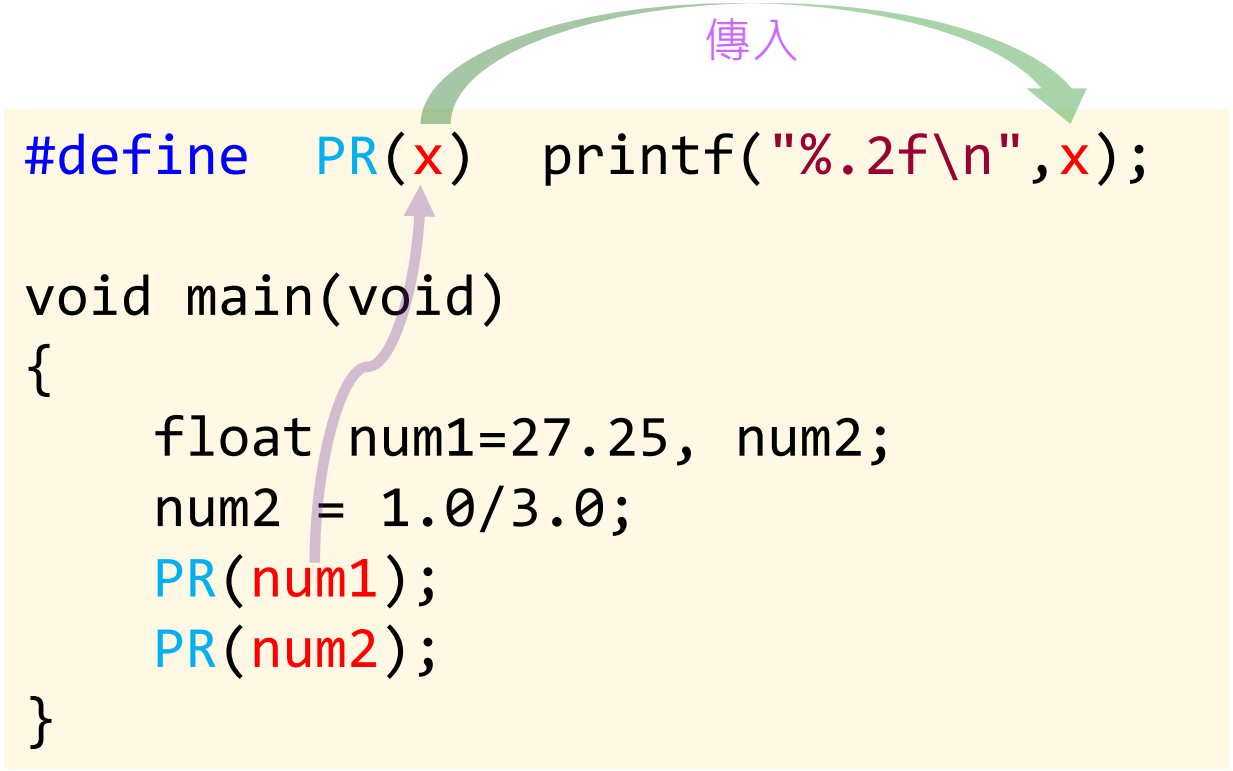
被取代成

編譯器看到的

```
#define ERROR printf("\nError\n");
main()
{
        :
    if(x != y)
        printf("\nError\n");
    else
        :
}
```

#define巨集

- 使#define巨集接受引數。



#define巨集

- 範例：取得兩整數較大者，利用巨集簡化程式。

```
#include <stdio.h>
#define MAX(a,b) ((a)>(b)?(a):(b))

void main(void)
{
    int x, y;
    printf("請輸入兩個整數：");
    scanf("%d %d", &x, &y);
    printf("%d 比較大", MAX(x,y));
}
```

#include前置命令

- 此命令讓程式可以載入另一個檔案，通常是事先定義好的標頭檔(Header file)。
- C語言對相關的應用及內定函數已經定義好許多標頭檔，可直接include進來，其副檔名為 *.h。
- 格式：

```
#include "檔名.h"
```

  - → 在目前路徑尋找該檔,找不到再去系統目錄找。

```
#include <檔名.h>
```

  - → 直接到系統設定的目錄找。

#include前置命令

- 我們已經使用過#include很多次了。放在程式最前面即可引入。

```
#include <stdio.h>    //標準輸出入函式庫
#include <stdlib.h>   //標準函式庫
#include <math.h>     //數學函式庫
#include <time.h>     //日期時間函式庫
      :

main( )
{
      :
}
```

#include前置命令

- #include使用範例：引入自訂標頭檔。

檔案: area.h

```
#define  PI  3.14159
#define  AREA_CIRCLE(radius)  (PI*radius*radius)
#define  AREA_SQUARE(length,width)  (length*width)
```

主程式:

引入

```
#include "area.h"
#include <stdio.h>  //標準函式庫
void main(void)
{   float R,L,W;
    printf("請輸入R L W：");
    scanf("%f %f %f",&R, &L, &W);
    printf("%f\n", AREA_CIRCLE(R));
    printf("%f\n", AREA_SQUARE(L,W));
}
```

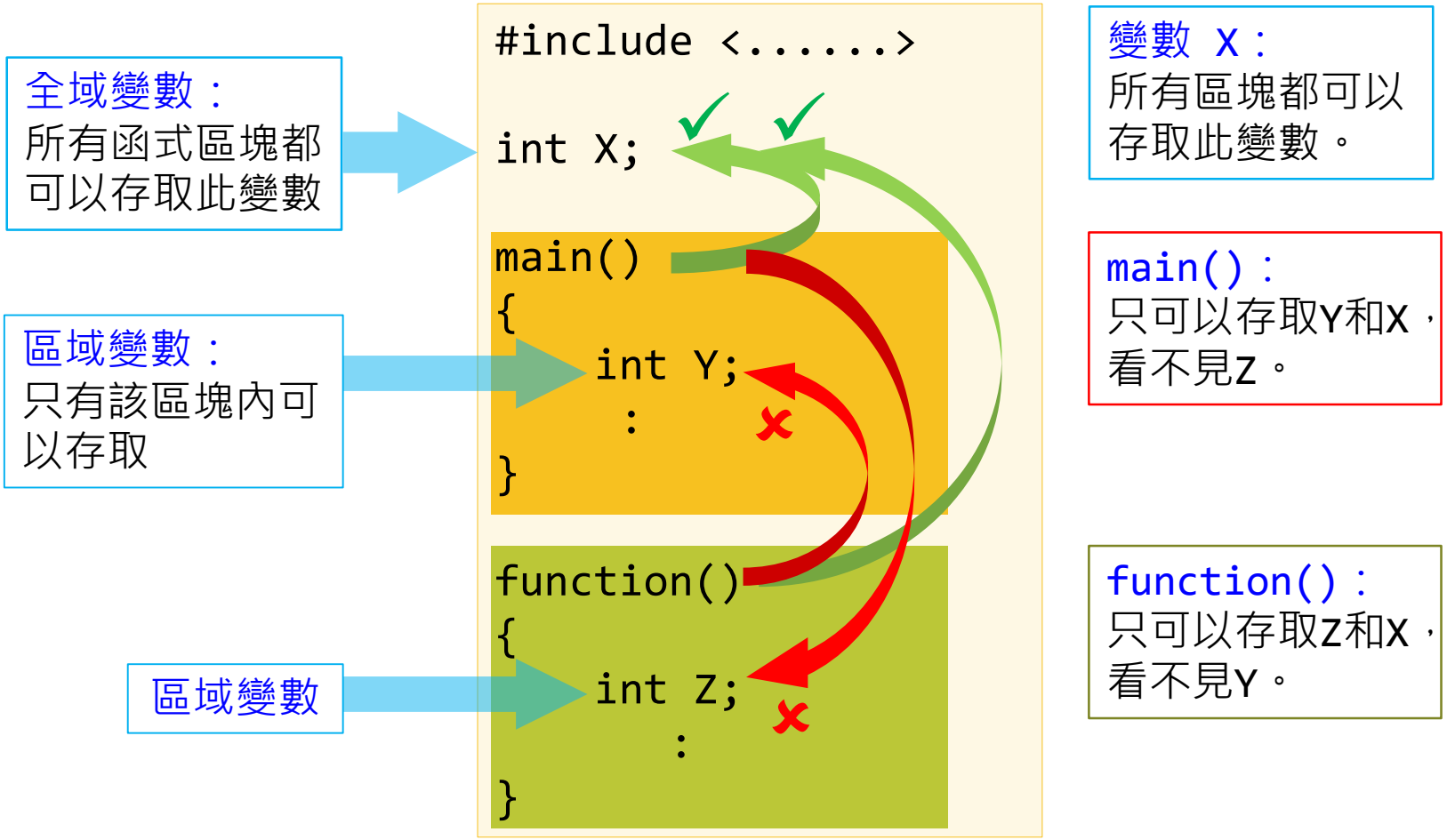
休息一下~

變數的使用範疇及生命週期

- 在C語言中，資料項目(或變數)的儲存類別，關係著該項資料的使用範疇(Scope)與生命週期(Lifetime)。
  - 使用範疇(Scope)：又稱可見範圍，程式中能"**看得見**"此資料項目的部分。
    - **全域變數**：宣告在所有區塊之外的變數。
    - **區域變數**：宣告在某個區塊內的變數。
  - 生命週期(Lifetime)：資料項目存在主記憶體中之時間。

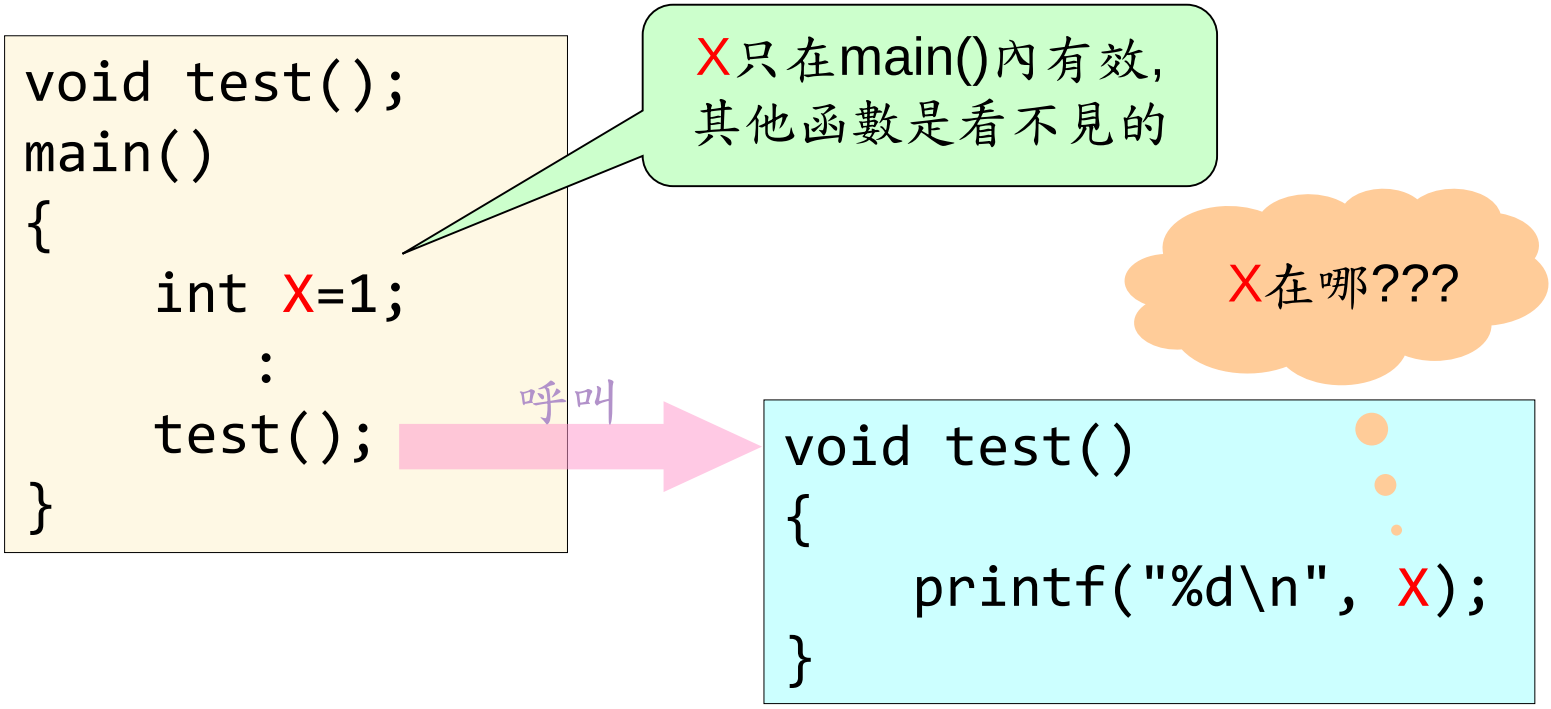
變數的使用範疇及生命週期

- 說明Scope之重要性:



變數的使用範疇及生命週期

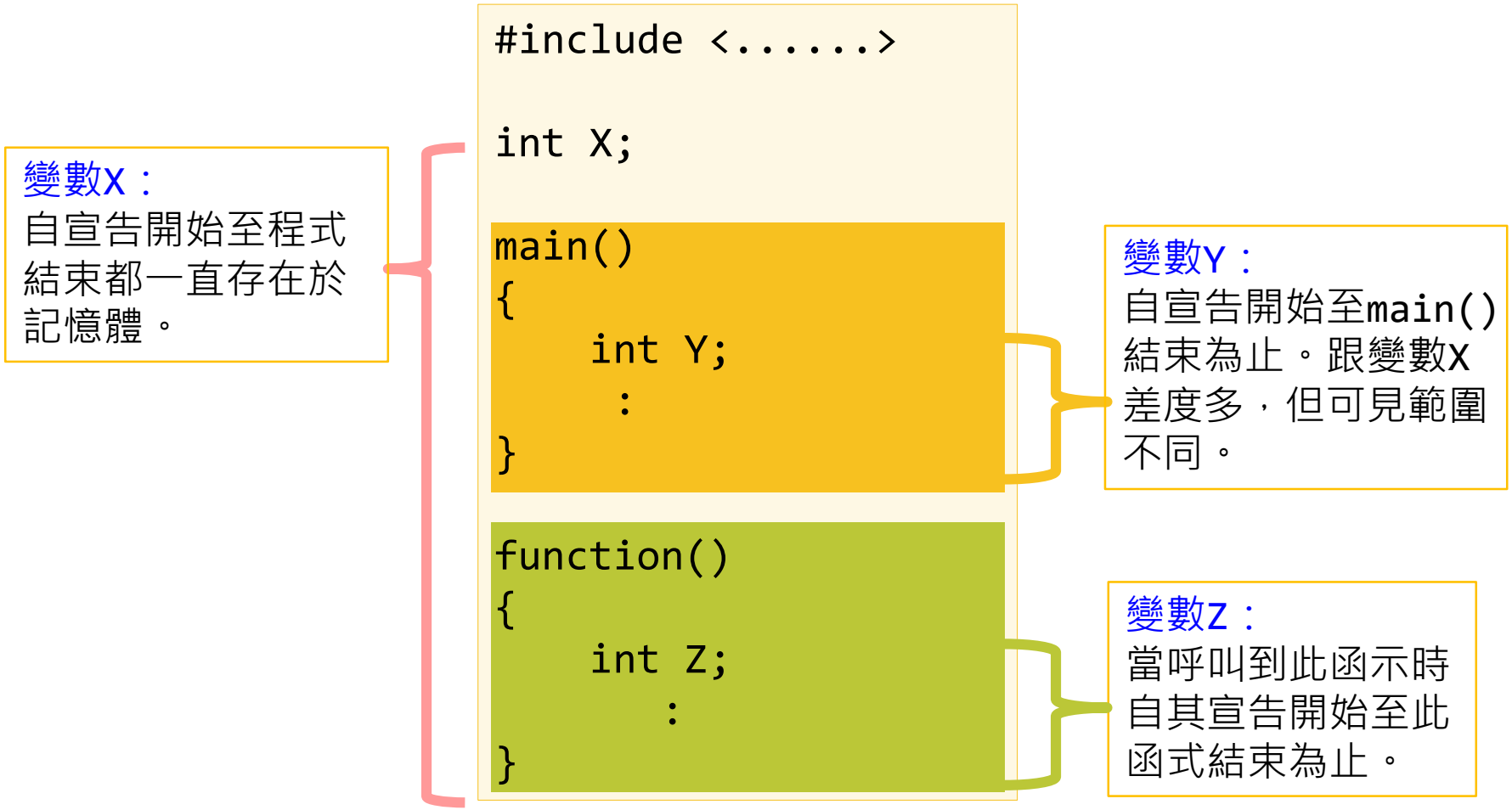
- 說明Scope之重要性:



變數X的scope並沒有涵蓋函數test()，printf()無法正常進行!

變數的使用範疇及生命週期

- 說明生命週期Lifetime之重要性:



變數的使用範疇及生命週期

- 說明生命週期Lifetime之重要性:

```
main()
{
   int sum(), result, i;
   for(i=1; i<=5; i++)
      result = sum(i);
   printf("%d", result);
}
```

執行結果：5（並不是1+2+3+4+5=15）

呼叫

```
int sum(int n)
{
    int temp=0;
    temp += n;
    return(temp);
}
```

傳回temp

因 temp 之生命週期只為 sum() 函數執行期間，故每次執行 sum() 之後，其記憶體空間將被釋放，當再次被呼叫時，將重新配置 temp，先前的值就不存在了。

儲存類別

- 變數的儲存類別分為：
  - auto(自動)
  - register(暫存器)
  - static(靜態)
  - external static(外部靜態)
  - extern(外部)
- 適當的選擇儲存類別可以：
  - 提高記憶體使用效率。
  - 增加程式執行效率。
  - 減少程式之錯誤。

儲存類別

- 儲存類別(Storge class)：

保留字	Scope	Lifetime
auto	區塊	區塊
register	區塊	區塊
static(內部靜態)	區塊	程式
static(外部靜態)	一個檔案	程式
extern	多個檔案	程式

- **區塊**：即由"{"與"}"所括住的程式段。
- **程式**：該程式執行的期間。
- **一個檔案**：該程式所有函數均可看見。
- **多個檔案**：所有正在執行的程式間皆可看見。

儲存類別

- auto與register儲存類別：
- 例：

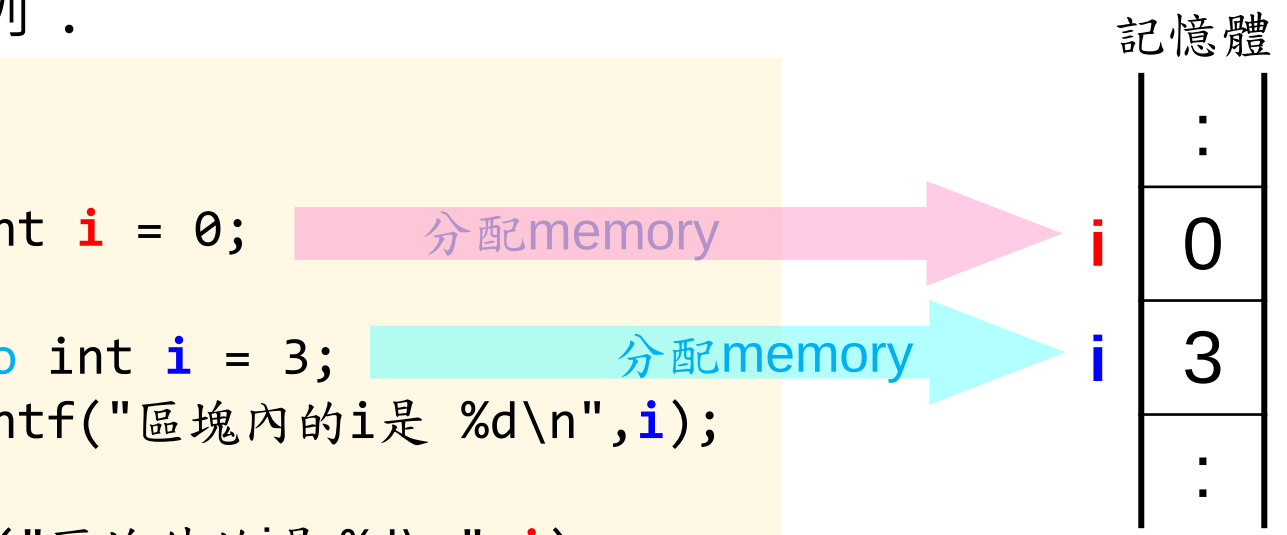
```
auto int a;         //a為auto類別之變數
register int r;     //r為暫存器類別之變數
int i;              //i未設定儲存類別,以auto視之
```

- auto與register之區別
  - auto宣告之變數是存在主記憶體中。
  - register宣告之變數是存在暫存器中，以減少記憶體耗損，提高存取度，若已無暫存器可用，則自動轉換為auto類別。

儲存類別

- auto範例：

```
main()
{
   auto int i = 0;
   {
      auto int i = 3;
      printf("區塊內的i是 %d\n",i);
   }
   printf("區塊外的i是 %d\n",i);
}
```



執行結果：
區塊內的i是 3
區塊外的i是 0

- 說明：
  - auto可省略。
  - 兩個i位於不同區塊，所以是不同的，第二個i只在其區塊內可見，離開該區塊後即消失。

儲存類別

- static儲存類別：
  - 在函數執行後，變數的值若要保留，已備下次呼叫時使用，則可宣告成static(靜態變數)。
  - static分成兩種：
    - 內部靜態(internal static)：宣告於函數內部。
    - 外部靜態(external static)：宣告於函數外部。
    - 兩者都是用 static 宣告，只是宣告之位置不同。

儲存類別

- static範例：

```
main()
{
   int sum(), result, i;
   for(i=1; i<=5; i++)
      result = sum(i);
   printf("%d", result);
}
```

呼叫

```
int sum(int n)
{
    static int temp=0;
    temp += n;
    return(temp);
}
```

傳回temp

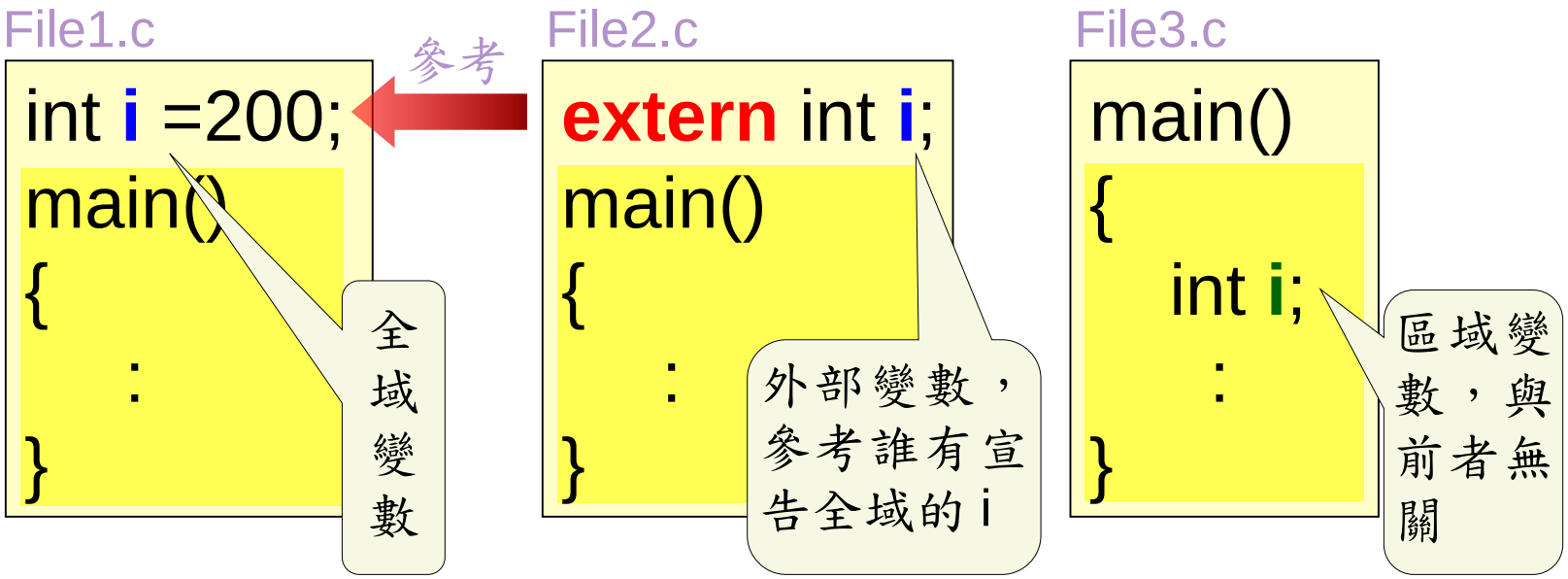
執行結果：15

變數temp經static宣告後，即使函式結束，其內容仍保留在記憶體中，直到再次被呼叫時繼續使用。

- static變數的初值只設定一次，第二次即不會再做 temp=0 的動作。

儲存類別

- extern儲存類別：
  - extern儲存類別的作用在於分享其他檔案內的全域變數。
  - 例：



休息一下~

陣列(Array)

- 用途：
  - 陣列(Array)係用來儲存一組具有相同資料型態的資料。
- 目的：
  - 減少變數的個數，使程式的編寫更容易。
  - 陣列可配合迴圈來處理大量資料。

陣列(Array)

- 為何要使用陣列?
  - 若要將一組有相關性意義的資料(如全年中各月份的薪資)存放在記憶體中，若每一份資料宣告一個變數，如：

```
long int salary1;       //一月份薪水
long int salary2;       //二月份薪水
       :                        :
long int salary12;      //十二月份薪水
//一共要宣告12個獨立的變數
```

  - 如此宣告使得程式的編寫變得冗長而不具效率。
  - 故應使用陣列來改進此種缺失。

陣列(Array)

- 宣告：任何陣列在使用前必須先宣告。
  - 格式為：

```
型態    陣列名稱[陣列長度];
```

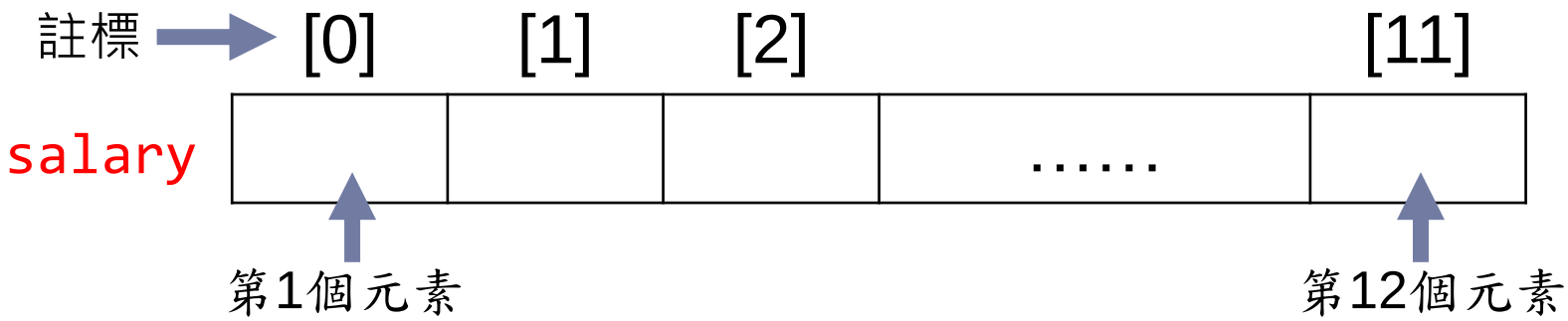
  - 型態：陣列中所有元素之共同資料型態。
  - 陣列名稱：用以識別不同陣列，命名規則同一般變數。
  - 陣列長度：陣列中可存放的總元素個數(通常為1~65535)。

陣列(Array)

- 一維陣列宣告：
  - 例：

```
Long int  salary[12];
```

元素的形態　陣列名稱　陣列長度



注意：陣列註標為正整數,而且是從0開始

陣列(Array)

- 陣列元素內容存取：
  - 存入至某元素：

```
陣列名稱[註標] = 值;      例：salary[0] = 369;
```

  - 取出某元素內容：

```
變數 = 陣列名稱[註標];     例：x = salary[0];
```

陣列(Array)

- 範例：將一年中各月份的薪資填入陣列中，並顯示總和(使用for迴圈)。

```
void main(void)
{
   long int sum=0, salary[12]; //宣告陣列
   for(int m=0; m<12; m++)
   {
      printf("請輸入%d月薪資:", m+1);
      scanf("%ld", &salary[m]); //存入陣列
      sum += salary[m];  //自陣列取出元素值
   }
   printf("總和：%ld", sum);
}
```

陣列(Array)

- 範例：將一年中各月份的薪資填入陣列中，並顯示總和(使用do-while迴圈)。

```
void main(void)
{
    long int sum=0, m=0, salary[12];
    do
    {
        printf("請輸入%d月薪資:", m+1);
        scanf("%ld", &salary[m]);
        sum += salary[m];
        m++;
    }while(m<12);
    printf("總和：%ld", sum);
}
```

陣列(Array)

- 例：將前述陣列中所有元素依序取出。

```
for(int m=0; m<12; m++)
{
    printf("%d月薪資為%d\n", m+1, salary[m]);
}
```

- 例：將每月薪資均加上1000元。

```
for(int m=0; m<12; m++)
{
    salary[m] += 1000;
}
```

陣列(Array)

- 陣列界限檢查：
  - C語言中對陣列不做界限的檢查。
  - 當存取陣列元素時，若所引用的註標超過所宣告之陣列長度時，C語言在compiler時並不提供任何警告，但在執行時會存取到其他記憶體空間的資料，而造成不可預期的錯誤。
- 例：

```
long int x, salary[12];
salary[15] = 1000;
x = salary[16];
```

宣告

已在陣列之外,這個記憶體空間會被覆蓋,造成程式錯誤

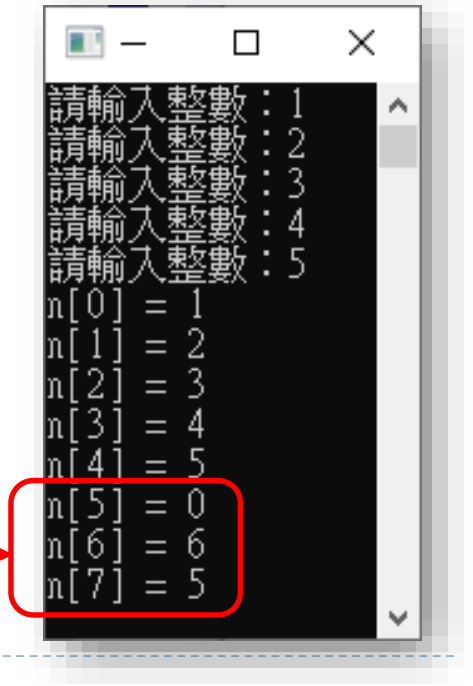
取到未知的值

陣列(Array)

- 範例：錯誤邊界存取。

```
void main(void)
{
    int n[5];  //宣告陣列，有5個元素
    for(int i=0; i<5; i++)
    {
        printf("請輸入整數：");
        scanf("%d", &n[i]);  //i=0~4
    }
    for(int j=0; j<8; j++)
        printf("n[%d] = %d\n",j ,n[j]);
}                                  //j=0~7
```

這三個是甚麼鬼？
(是其它程式空間的記憶體內容)



陣列(Array)

- 一維陣列初值設定：
  - 在宣告陣列時也可同時指定初值。
  - 陣列初值設定方式：

```
型態   陣列名稱[陣列長度] = {初值1, 初值2, …};
```

以逗點分隔的一些常數

  - 例：

```
int x[4] = {10,20,30,40};
```

註標	[0]	[1]	[2]	[3]
陣列 x	10	20	30	40

陣列(Array)

- 陣列初值設定注意事項：
  - 設定陣列初值時，可以不須宣告陣列之長度。
  - 例：

```
int arr[ ] = {1, 2, 3, 4, 5};
```

空白即可

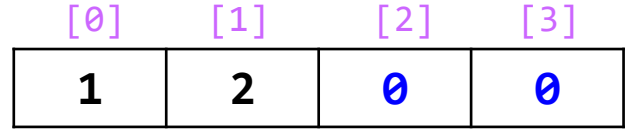
  - 編譯器會依初值之個數自行訂出陣列長度。
  - 同以下敘述：

```
int arr[5] = {1, 2, 3, 4, 5};
```

陣列(Array)

- 陣列初值設定注意事項：
  - 設定陣列初值時，若所宣告之陣列長度：
    - 大於初值個數：編譯器會將陣列中其餘元素初值設為0。

```
int X[4] = {1, 2};
```



    - 小於初值個數：編譯器會發出錯誤訊息。

```
int X[2] = {1, 2, 3, 4};  //會有錯誤訊息
```

  - 宣告陣列但未設定初值時，則陣列之初值為：
    - 為auto或register時→初值無法預測，為所分配之記憶體區塊內殘存之值。
    - 為global或static時→初值自動設為0。

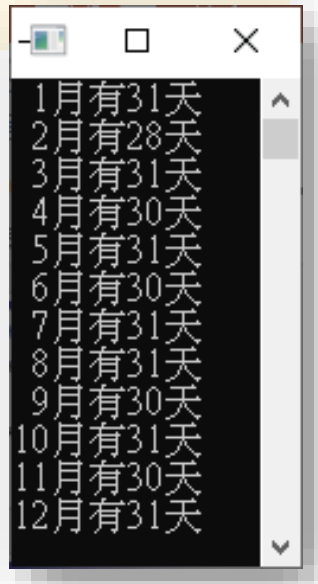
陣列(Array)

- 範例：將一年中每一月份的天數印出來。

```
int days[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
void main(void)              //陣列直接設定初值
{   int month;
    for(month=0; month<12; month++)
        printf("%2d月有%d天\n", month+1, days[month]);
}
```

宣告陣列時即指定初值

[0]	[1]	[2]	[3]	…	…		…	[10]	[11]
31	28	31	30	31	30	31	31	30	31	30	31



多維陣列

- 即兩個維度以上之陣列。C語言之最大維數沒有限制，視編譯程式而定。
- 宣告方式：

```
型態　陣列名稱[陣列長度].....[陣列長度];
```

- 例：

```
int arr[2][3][4];
```

  - 宣告為三維陣列，型態為整數，三維之長度分別為2、3、4，共有2*3*4=24個元素。

多維陣列

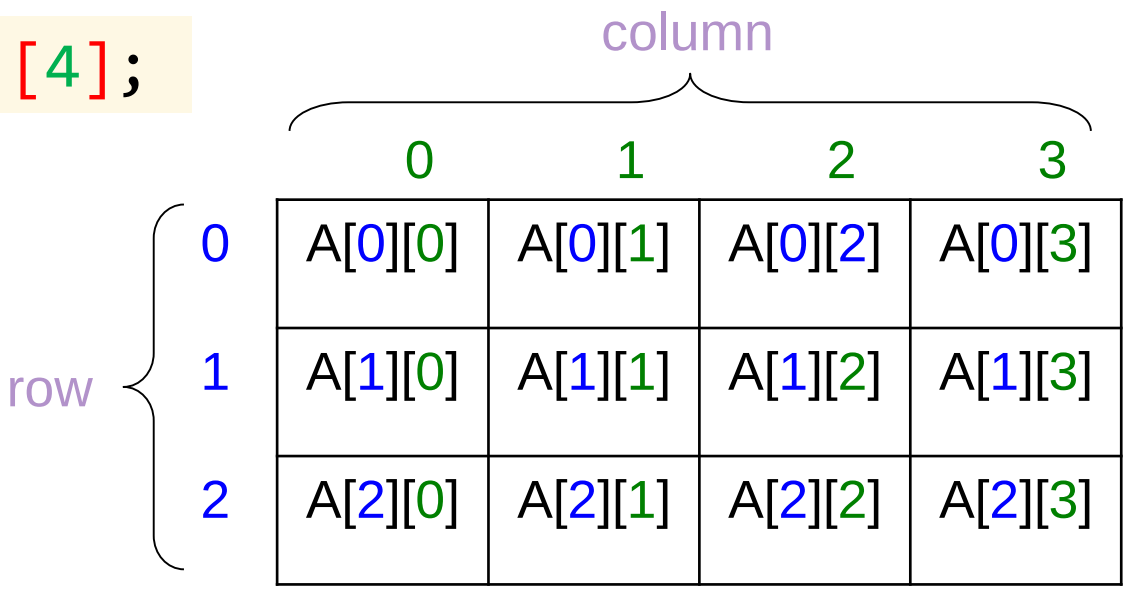
- 二維陣列：
  - 宣告方式：

```
型態　陣列名稱[列數(row)][行數(column)];
```

  - 例：

```
int A[3][4];
```

  - 示意圖：



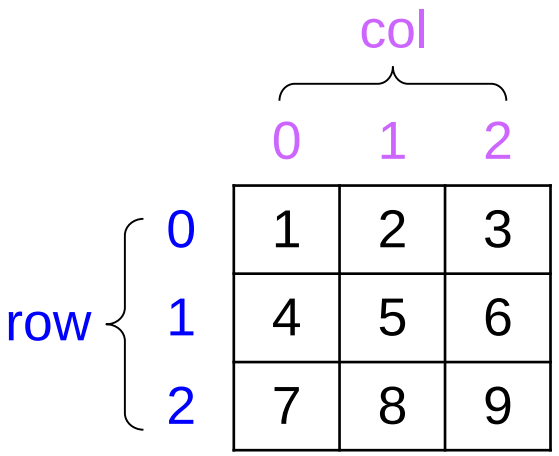
多維陣列

- 二維陣列初值設定：
  - 方式一：

```
int A[3][3] = { {1,2,3}, {4,5,6}, {7,8,9} };
```

  - 方式二：

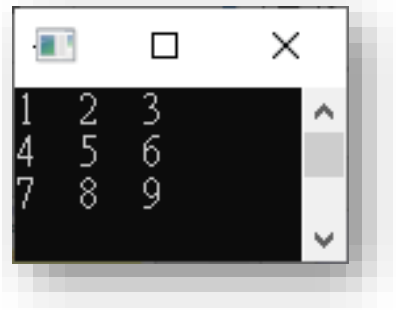
```
int A[3][3] = { {1,2,3},
                {4,5,6},
                {7,8,9} };
```



多維陣列

- 範例：將數字1~9如圖示依序填入二維陣列。

```
void main(void)
{
    int arr[3][3];  //宣告一個3*3的二維陣列
    int x = 1;
    for(int row=0; row<3; row++)
    {
        for(int col=0; col<3; col++)
        {
            arr[row][col] = x++;
            printf("%d  ", arr[row][col]);
        }
        printf("\n");
    }
}
```



row \ col	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

以陣列作為引數傳遞

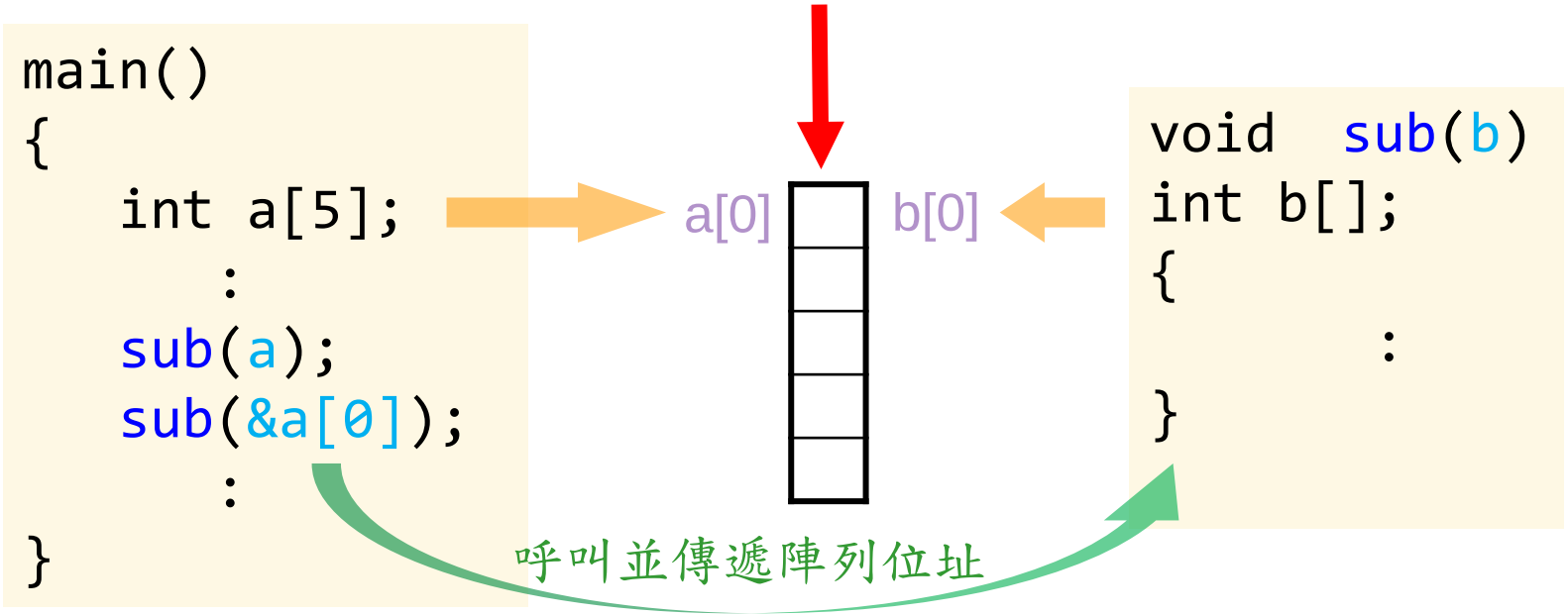
- 將陣列傳給函數時，只需陣列名稱即可，當陣列名稱單獨使用時，即代表該陣列的位址，也就是陣列中第一個元素的位址。

```
List  == &List[0]
```

陣列名稱　　第一個元素的位址

以陣列作為引數傳遞

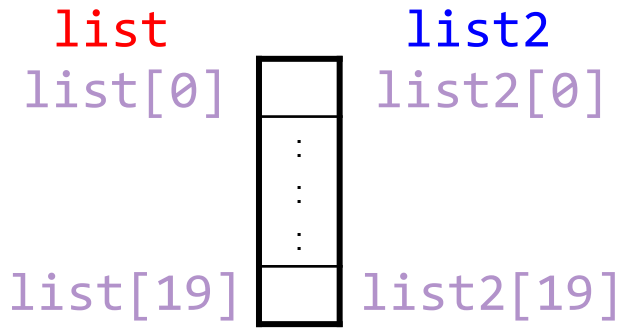
- 利用陣列來當函數的引數，相當於一次傳入多個參數函數，也可獲得多個傳回結果。
- 傳遞陣列給函數時，並不會將此陣列複製一份，函數與主程式都是**看到同一個陣列**。



以陣列作為引數傳遞

- 使用者輸入一組數目，程式會把最大的印出來。

```
#define  MAXSIZE  20
int max();

void main(void)
{
   int list[MAXSIZE];
   int size=0, num;
   do
   {
      printf("請輸入整數(0結束):");
      scanf("%d", &list[size]);
   }while(list[size++]!=0);
   size--;
   num = max(list,size);
   printf("最大值是%d\n",num);
}
```




```
int max(list2, size2)
int list2[], size2;
{
   int dex, max;
   max = list2[0];
   for(dex=1; dex<size2; dex++)
      if(max<list2[dex])
         max = list2[dex];
   return(max);
}
```

以陣列作為引數傳遞

- 陣列名稱與變數名稱：
  - 如果list是一個陣列，則list就代表它的起始位址。
  - 如果list是一個簡單的變數，則list就代表它的內容。
  - 同一個函式中，陣列名稱與變數名稱不可相同。
- 例：

這是一個陣列

```
int arr[5], x;
    :
x = arr;
```

x得到arr陣列的起始位址

這是一個變數

```
int arr, x;
    :
x = arr;
```

x得到arr的內容

字串(String)

- 字串只是一連串的字元，也就是char型態的陣列。
  - 例：
    ```
    printf("APPLE");
    ```
    - 雖然只有五個字母,但真正存於記憶體共六個字。

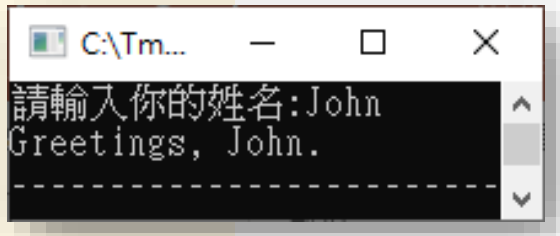
    - 所有的字串都必須以空字元"\0"為結束，此字元之值為零，稱Null字元。

  - (老實說，我覺得C的字串處理能力蠻弱的，它連string型態都沒有，其它語言都有...)

字串(String)

- 輸入字串範例：
  - 雖然陣列長度為15，但只能輸入14個字，最後一個要保留給null字元，若超過14個字，將會把記憶體中其它的資料覆蓋過去，造成無法預期的後果。
  - scanf()中fname不用加&，因其為陣列，只需使用陣列名稱即代表其位址。

```
void main(void)
{
    char fname[15];
    printf("請輸入你的名字:");
    scanf("%s",fname);
    printf("Greetings, %s.", fname);
}
```



字串輸出入函數

- puts()：字串輸出
  - puts()比printf()簡單，但只能輸出一個字串，無法包含變數，若只是要單純的輸出一字串，puts()較方便。

- gets()：字串輸入
  - gets()可接受空白及跳格鍵(Tab)，直到按下Enter才會結束輸入。

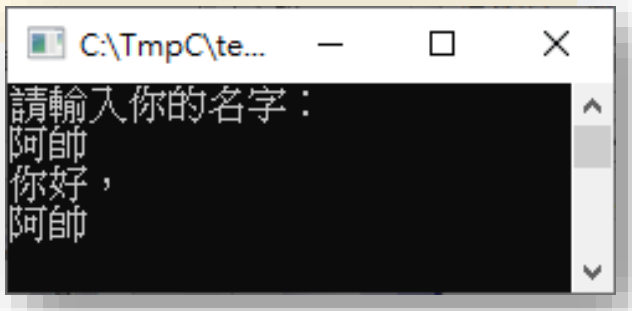
字串輸出入函數

- puts與gets範例：

將要輸出的字串放入puts函數即可。puts最後會自動加上換行。

將字串陣列名稱放入gets即可，輸入之字串會放入該陣列。

```
void main(void)
{
    char name[81];
    puts("請輸入你的名字：");
    gets(name);
    puts("你好， ");
    puts(name);
}
```



設定字串之初值

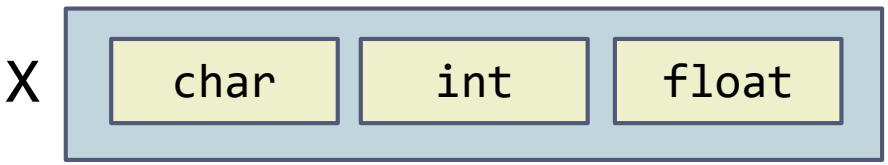
- 字串初值設定方式：
  - 方式一：
```
char name[] = {'O','r','i','o','n'};
```
  - 方式二：
```
char name[] = "Orion";
```

- 兩種方式都是一樣的，陣列不用指定大小，程式會自動依字元數決定，並自動補上null字元。

休息一下~

結構(Structure)

- 結構(Structure)概念：
  - 變數只能存放一個資料。
  - 例： `int X;` → X [int]
  - 陣列只可以存放多個相同型態的資料。
  - 例： `int X[3];` → X [int | int | int]
  - 結構可以存放一組不同型態的資料。
  - 例：一個變數包含三個儲存空間，其形態可不同。



結構(Structure)

- 結構宣告(定義結構方式)：

```
struct 結構名稱
{
    資料型態  變數名稱;
        :
        :
    資料型態  變數名稱;
}
```

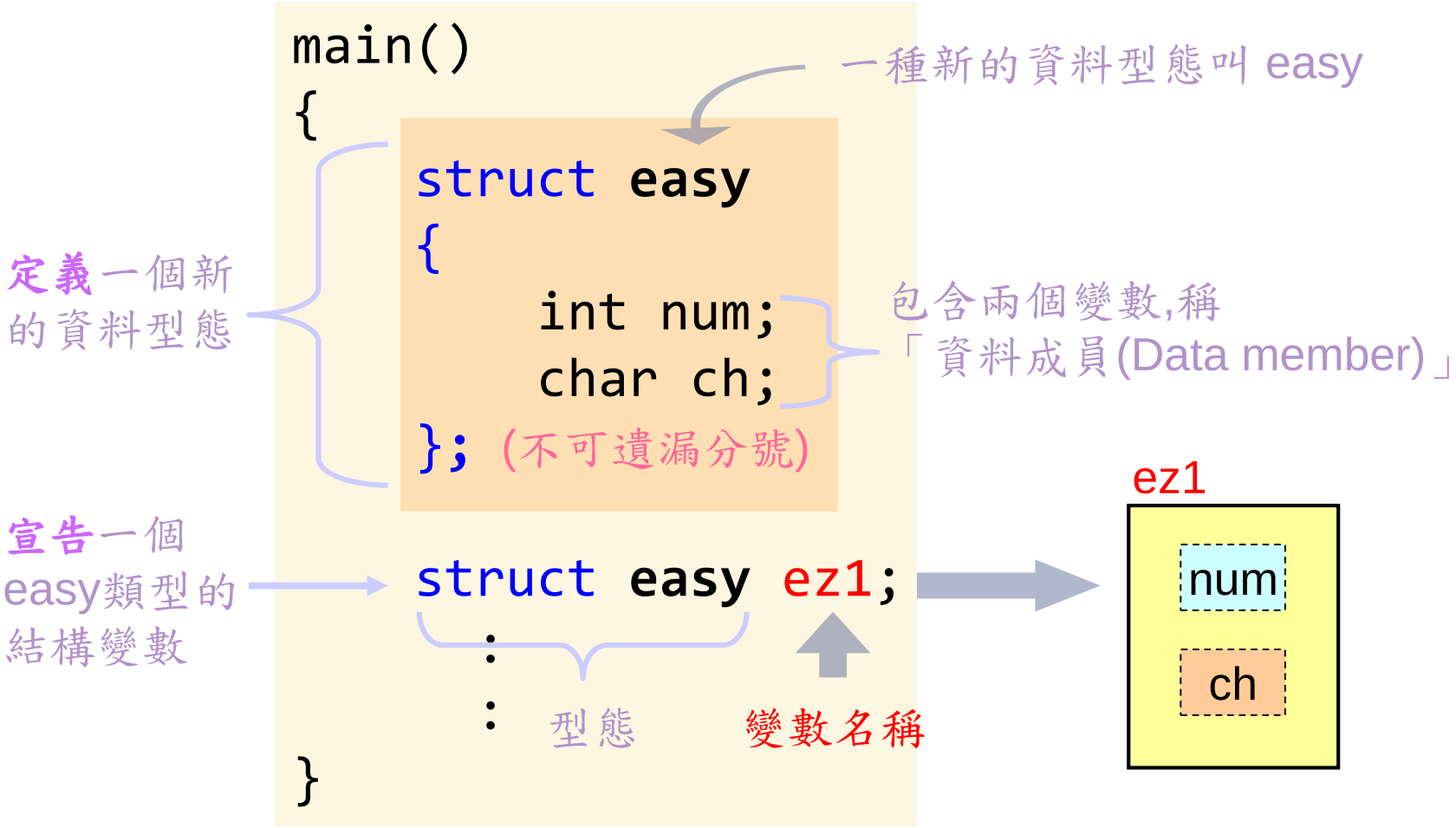
例：

```
struct mydata
{
    char name[20];
    int age;
    float score;
}
```

- 結構也可以說是一個由程式設計者自訂的資料型態。

結構(Structure)

- 結構定義及宣告範例：



結構(Structure)

- 設定結構初值：
  - 在宣告結構變數時指定初值。
  - 依結構成員之型態順序指定，以大括符包圍，並以逗點分開。
  - 依前例：

```
struct easy ez1 = { 3 , 'a' };
struct easy ez2 = {15 , 'b' };
```

結構(Structure)

- 結構成員存取：
  - 利用「.」運算子來連接結構變數與結構成員。
  - 例：

```
struct easy ez1;
ez1.num= 3;
ez1.ch= 'a';
```

  - 若變數為指向結構的指標時，則可使用「->」運算符號(指標稍後介紹)。

```
struct easy ez1,*ptr;
ptr = &ez1;
ptr->num = 3;
ptr->ch = 'a';
```

結構(Structure)

- 結構存取範例：

```
main()
{    struct easy
     {
          int num;
          char ch;
     };

     struct easy ez1;
     ez1.num = 3;
     ez1.ch = 'a';
     printf("%d  %c\n", ez1.num, ez1.ch);
}
```

定義結構

宣告結構變數

結構(Structure)

- 將定義與宣告結合：
  - 在定義結構後直接宣告結構變數。
  - 例：

```
struct easy
{
    int num;
    char ch;
} ez1, ez2;
```

定義結構

以逗點隔開,分號結束

宣告結構變數ez1和ez2

結構(Structure)

- 複製結構內容：
  - 多個宣告成相同結構型態的變數之間，可視同一般變數做運算，如：

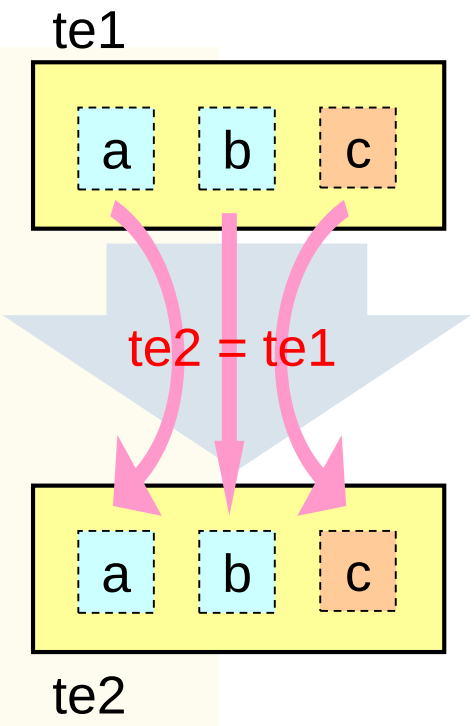
```
struct easy ez1, ez2;

ez1 = ez2;
```

  - ez2內所有成員變數的內容會被指定給ez1內相同成員的變數。

結構(Structure)

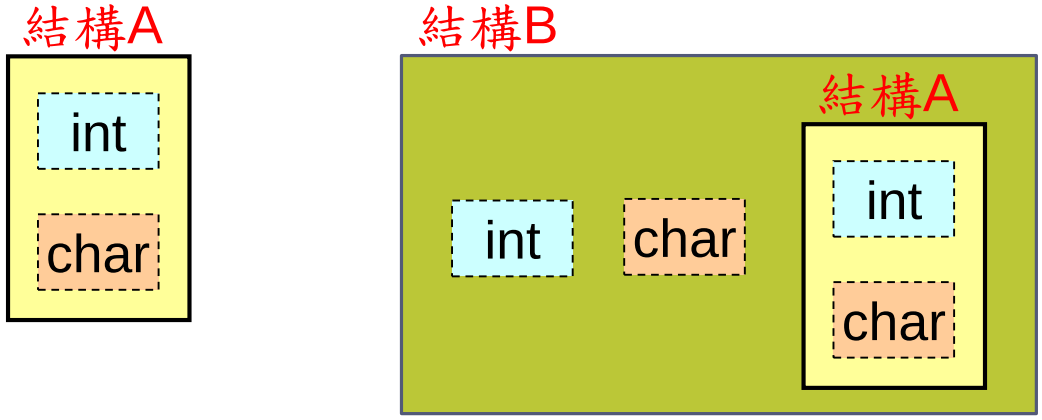
- 複製結構內容：

```
void main(void)
{
    struct test
    {
        int a, b;
        char c;
    }te1, te2;

    printf("請輸入一個字元：");
    scanf("%c", &te1.c);
    printf("請輸入兩個數字：");
    scanf("%d %d", &te1.a, &te1.b);
    te2 = te1;
    printf("%d %d %c\n", te2.a, te2.b, te2.c);
}
```



巢狀結構

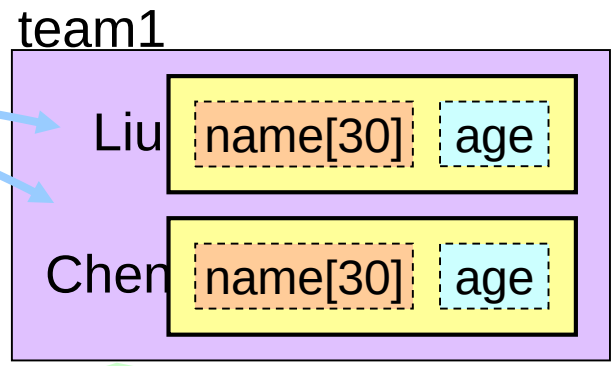
- 即結構中包含結構。
- 結構一經定義完畢，即可視為新的資型態，故另再定義新結構時亦可包含此種資料型態，成為巢狀結構。
  - 例：結構B例包含了一個結構A。



巢狀結構

- 巢狀結構範例：

```
struct personal
{    char name[30];
     int age;
};
```




```
struct team
{    struct personal Liu;
     struct personal Chen;
};
```

結構team包含兩個結構變數

```
struct team team1={{"Orion Liu",30 },{"Becky Chen",20}};
```

存取方式

```
void main(void)
{    printf("Liu:");
     printf("Name:%s, age:%d\n",team1.Liu.name,team1.Liu.age);
     printf("Chen:");
     printf("Name:%s, age:%d\n",team1.Chen.name,team1.Chen.age);
}
```

休息一下~

指標(Pointer)

- 指標是C語言提供直接指引在記憶體某一個指定位址的一種類似於路標的設施。

- 其實指標變數的內容就是一個記憶體位置，我們透過它可以直接存取某個記憶體位置的內容。

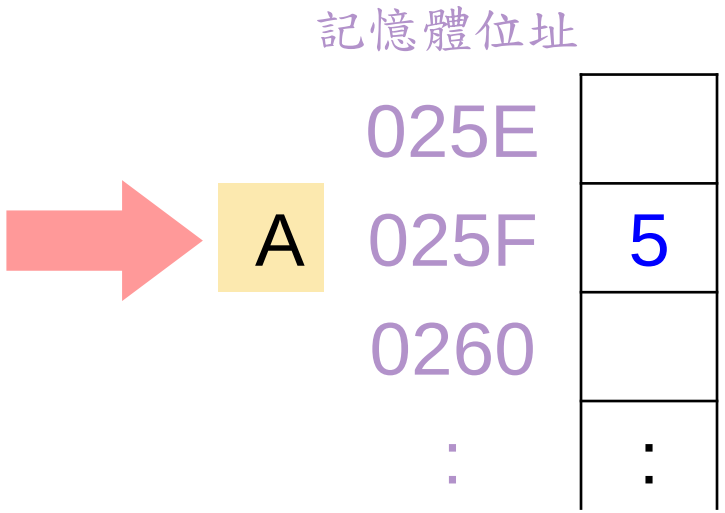
- 這個功能太危險，所以之後的語言(C++、Java、Python..等)都把這個功能拿掉了。

指標(Pointer)

- 指標之內容即為記憶體位址。

✸變數和它的位址：

變數A由作業系統自行分配位址，本例假設在記憶體位址025F的地方，其內容值為5。



- (讀取A得到5，讀取&A得到025F。)

指標(Pointer)

- 指標變數的宣告(使用*號)：
  - 格式：
    ```
    資料型態   *指標變數名稱;
    ```
  - 例：
    ```
    char   *ptr1;
    int    *ptr2, *ptr3;
    ```
    - 宣告了3個指標變數，其中ptr1用以存放某一個字元變數的位址，而ptr2與ptr3用以存放某整數變數的位址。
    - 換句話說，ptr1指到一個字元變數，ptr2與ptr3各指到一個整數變數。

指標(Pointer)

- 指標使用範例：

```
main()
{    int i= 3;
     int j= 5;
     char c= 'a';
     char *ptr1;
     int *ptr2, *ptr3;
     ptr1= &c;
     ptr2= &i;
     ptr3= &j;
}
```

PS：要記得記憶體位址是隨機分配的，每次執行時都不盡相同。

指標(Pointer)

- & 與 * 運算符號：
- & 運算符號是用以取得變數的位址，稱為「取址運算符號(Address operator)」。
- * 運算符號是用來取得指標所指向的變數的內含值，稱為「間接定位運算符號」。

&a	取得變數a的位址
*b	取得指標變數b指到的位址的內容

指標(Pointer)

- &運算符號使用法：
  - 其作用是將變數名稱之所在位址取出，而後可以將該位址值指定給指標，例：

```
int x, *ptr;   //宣告整數變數x及指標變數ptr
ptr = &x;      //將變數x的位址存入指標ptr中
                  (稱為ptr指向x)
```

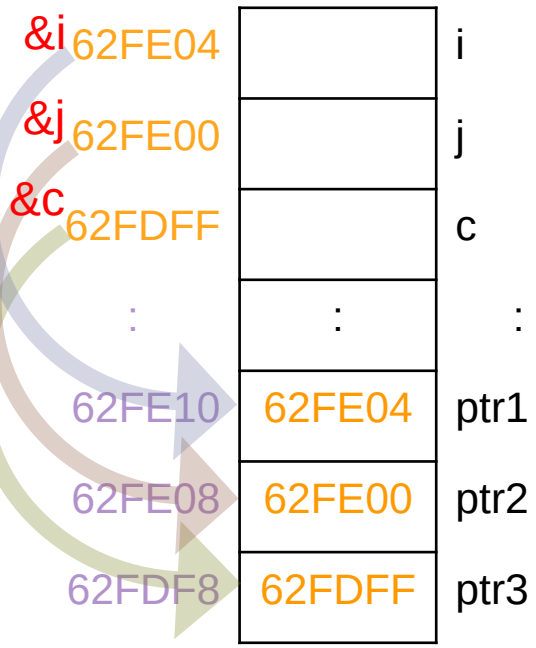
  - &只對變數名稱作運算，故&(i+1)或&(5)均是錯誤的，&也不可以對register變數作運算，因為register在CPU內，沒有位址值。

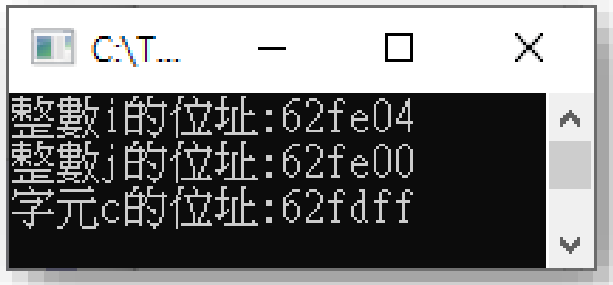
指標(Pointer)

- &符號使用範例：

```
void main(void)
{    int i, j, *ptr1, *ptr2;
     char c, *ptr3;
     ptr1= &i;
     ptr2= &j;  //取出變數位址交給指標變數
     ptr3= &c;
     printf("整數i的位址:%lx\n", ptr1);
     printf("整數j的位址:%lx\n", ptr2);
     printf("字元c的位址:%lx\n", ptr3);
}
```

(無號長整數或16進位數)

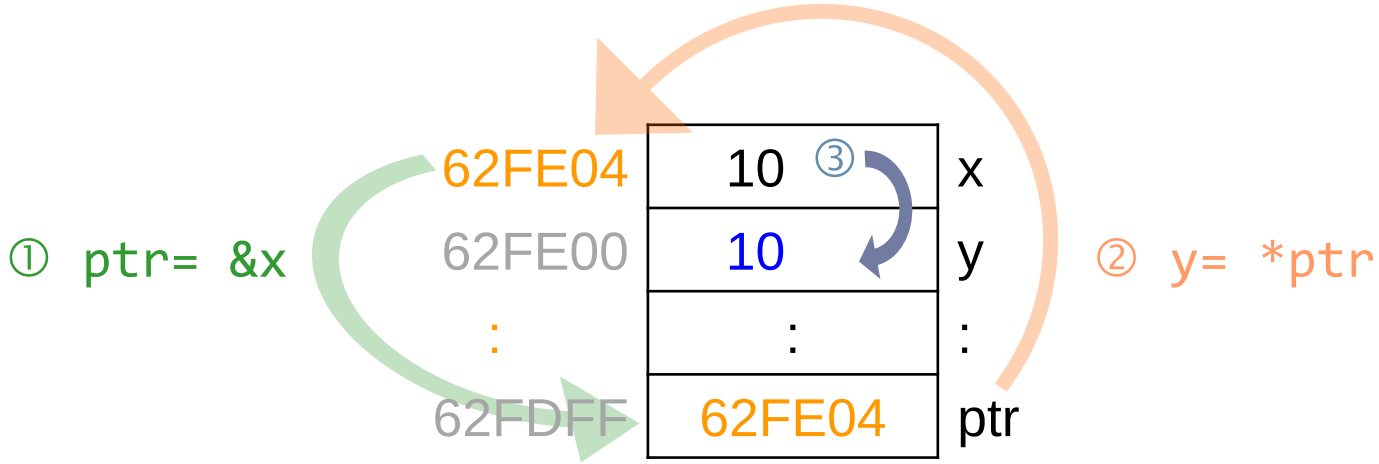




指標(Pointer)

- *運算符號之使用：
  - 作用：將指標所指向的變數的內容取出。
  - 例：

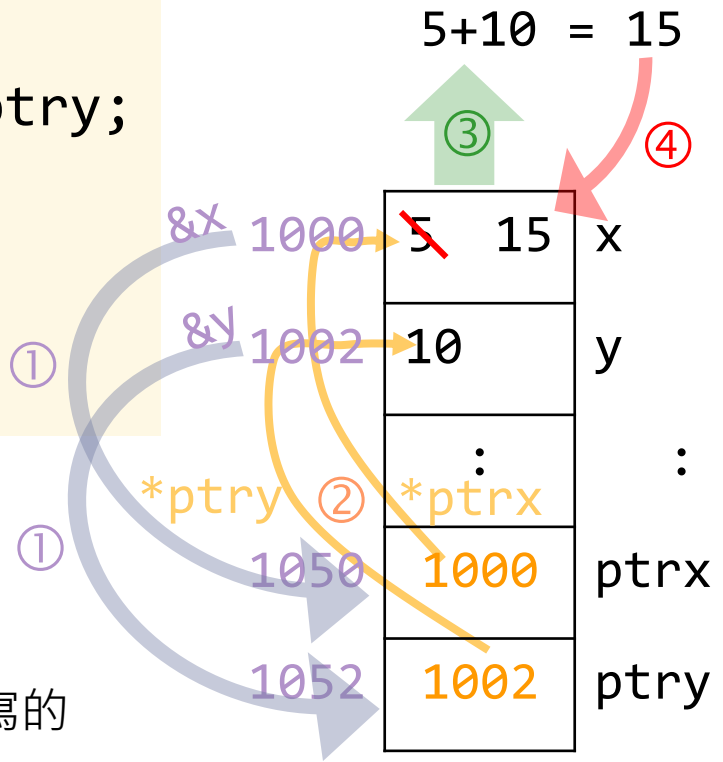
```
int x=10, y, *ptr;
ptr= &x;    //將x的位址取出,放入ptr中
y= *ptr;    //將ptr所指向的位址之內含值取出放入y
```



指標(Pointer)

- 範例：利用指標指向兩個整數x、y，並將x、y之內容值相加，結果放回x中。

```
main()
{
    int x=5, y=10, *ptrx, *ptry;
  ① ptrx= &x;
  ① ptry= &y;
    x= *ptrx + *ptry;
}   ④     ②   ③    ②
```

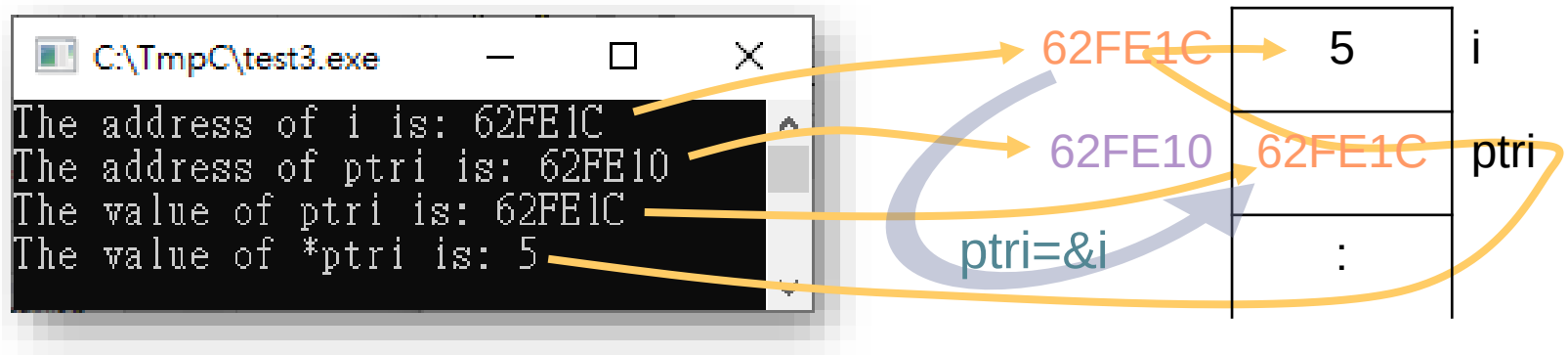


PS:記憶體位址是隨便寫的

指標(Pointer)

- 範例：指標測試程式(一)

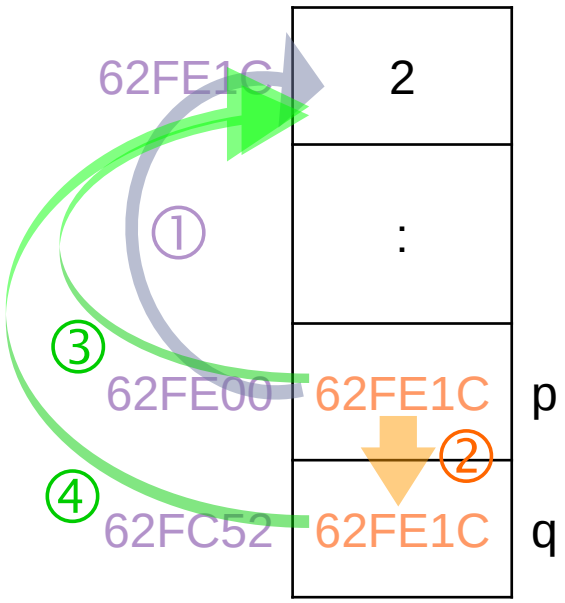
```
void main(void)
{   int i=5, *ptri;
    ptri= &i;
    printf("The address of i is: %X\n", &i);
    printf("The address of ptri is: %X\n", &ptri);
    printf("The value of ptri is: %X\n", ptri);
    printf("The value of *ptri is: %X\n", *ptri);
}
```



指標(Pointer)

- 指標測試程式(二)：

```
main()
{
    int *p, *q; //宣告兩個指標變數
        :
        :
① *p= 2;    //p與q初值設定
②  q= p;
    printf("%d %d\n", *p, *q);
}                          ③    ④
```



執行結果會顯示： 2 2

PS:本程式片段僅供說明,不能執行,因指標變數未做初值設定

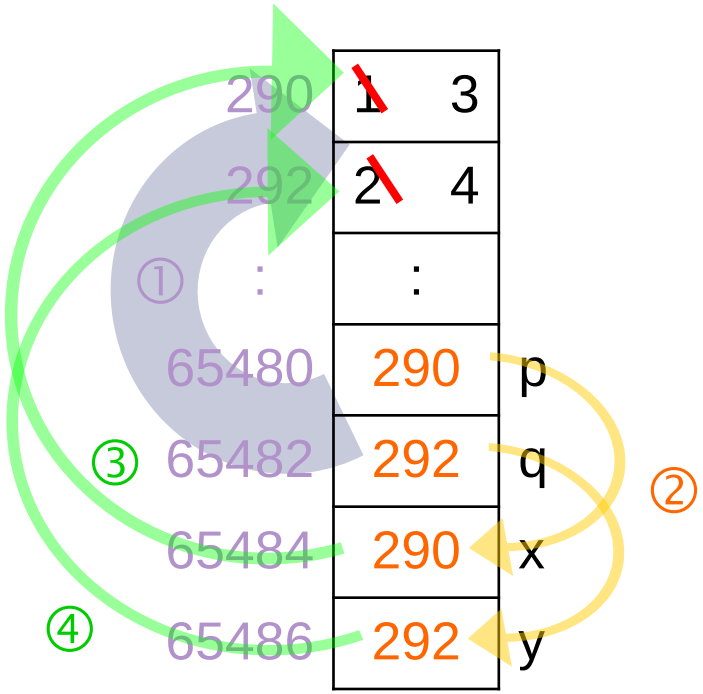
指標(Pointer)

- 傳遞指標給函數：

```
void sub(int*, int*);
main()
{
    int *p, *q;
        :
        :  (p與q初值設定,假定為290及292)
    *p=1; *q=2;  ①
    sub(p,q);    ②
    printf("%d %d\n",*p,*q);
}
```

傳遞指標

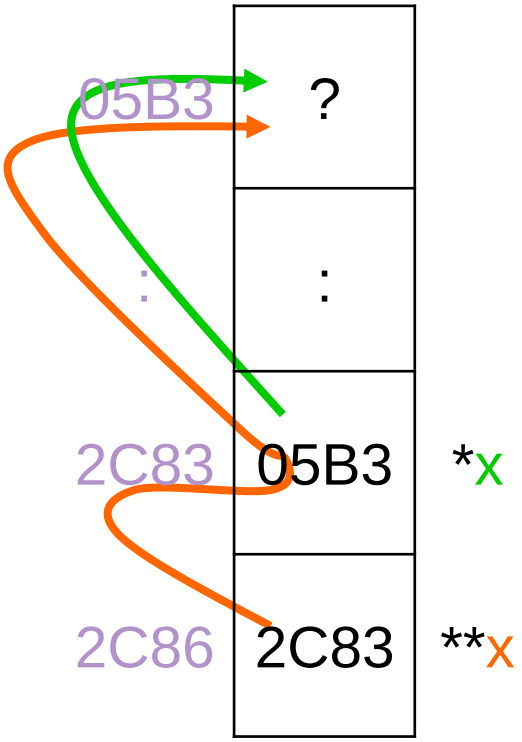
```
void sub(int *x, int *y){
    *x= 3;  ③
    *y= 4;  ④
}
```



執行結果： 3 4

指標(Pointer)

- 多重指標：
  - 多重指標指的是有兩個以上的*運算。
  - 例如：
    - *x：表示x指標所指到的位址之內容。
    - **x：表示指標變數*x的內容所指到的位址之內容。



- 因為指標的功能後來的語言都不支援了，所以就此簡單的介紹，有興趣的同學可以自行鑽研。

休息一下~

- 若指標不熟也不用緊張，除了C++，其它語言都沒有指標了(有類似的功能，但更安全)。

檔案存取

- 基本上檔案I/O和標準I/O運作原理相同，只是對象不同而已。
  - 標準I/O：對螢幕及鍵盤。
  - 檔案I/O：對檔案。

- 檔案讀寫流程：
  - Step1:定義指向FILE結構的指標。
  - Step2:開啟檔案。
  - Step3:呼叫檔案處理相關函數進行檔案操作。
  - Step4:關閉檔案。

檔案存取

- 常用檔案存取函式：

函式	功能說明
fopen	語法：FILE *fopen(const char *filename, const char *mode) 功能：使用 mode 模式來開啟指定的 filename 資料檔。若開啟檔案成功會傳回一個具有 FILE 檔案結構的指標；若開啟失敗，則會傳回 NULL。
fclose	語法：int fclose(FILE *fptr) 功能：將檔案指標 fptr 所指的檔案關閉。若傳回 0 表示關檔成功。
fgetc	語法：int fgetc(FILE *fptr) 功能：讀取檔案指標 fptr 所指位置中的字元後，檔案指標即往後移一個字元位置，若檔案指標已指到 EOF(檔案結束符號)，則傳回-1。
fputc	語法：int fputc(int ch, FILE *fptr) 功能：將字元變數 ch 寫入檔案指標 fptr 所指向檔案的位置。
fgets	語法：char *fgets(char *str, int n, FILE *fptr) 功能：在檔案指標 fptr 所指檔案位置讀取 n 個字元並放入 str 字串(即 str 指標所指的位址)。
fputs	語法：int fputs(const char *str, FILE *fptr) 功能：將字串 str(即 str 指標所指位址內的字串)寫入檔案指標 fptr 所指向檔案的位置。
fscanf	語法：int fscanf(FILE *fptr, "格式化字串", [引數串列]) 功能：以指定的格式由緩衝區中 fptr 所指位址讀取資料並放入引數串列對應的變數，適用循序檔。
fprintf	語法：int fprintf(FILE *fptr, "格式化字串", [引數串列]) 功能：以指定的格式將引數串列對應的變數資料分別寫入至 fptr 指標所指的檔案，適用於循序檔。
Fileno	語法：int fileno(FILE *) 功能：傳回所開啟檔案對應的檔案處理代碼(handle_no)。

檔案存取

- 常用檔案存取模式：

模式	功能說明
"w"	開啟一個可以寫入資料的檔案。若檔案存在，則該檔內的資料都會被刪除變成一個空檔；如果檔案不存在，則會自動建立一個新檔。
"a"	開啟一個可以寫入資料的檔案。若檔案存在，則新增的資料會附加在該檔案的最後面；如果檔案不存在，則會自動建立一個新檔。
"r"	開啟一個已經存在的檔案，若檔案不存在，則 fopen()函式會傳回 NULL。
"w+"	開啟一個可以讀寫資料的檔案。若檔案存在，則該檔內的資料都會被刪除變成一個空檔；如果檔案不存在，自動建立一個新檔。
"a+"	開啟一個可以讀寫資料的檔案。若檔案存在，則新增的資料會附加在該檔案的最後面；如果檔案不存在，則會自動建立一個新檔。
"r+"	開啟一個可以讀寫資料的已存在檔案，若檔案不存在，則 fopen()函式會傳回 NULL。
"wb+"	開啟一個可以讀寫資料的二進位檔案。若檔案存在，則該檔內的資料都會被刪除變成一個空檔；如果檔案不存在，則會自動建立一個新檔。
"ab+"	開啟一個可以讀寫資料的二進位檔案。若檔案存在，則新增的資料會附加在該檔案的最後面；如果檔案不存在，則會自動建立一個新檔。
"rb+"	開啟一個可以讀寫資料的已存在二進位檔案，若檔案不存在，則 fopen()函式會傳回 NULL。

檔案存取

- 檔案字元I/O(寫入)：

```
#include <stdio.h>
#include <conio.h>
void main(void)
{
   FILE *fptr;
   char ch;
   fptr= fopen("D:\\test.txt","w");
   if(fptr == NULL)
     printf("檔案開啟失敗");
   else{
     printf("請輸入字串，空白結束：");
     while((ch=getche()) != ' ')
       fputc(ch, fptr);
     fclose(fptr); }
}
```

檔案結構定義在此

檔案指標,準備指向一個檔案

檔案結構關鍵字

檔案名稱

開啟檔案

判斷檔案開啟是否成功

將字元送至檔案

關閉檔案

將鍵入的字元依序送至檔案內,直到輸入空白為止

模式	作用
r	讀取
w	寫入
a	附加

檔案存取

- 檔案字元I/O(讀出)：

```
#include <stdio.h>

void main(void)
{
    FILE *fptr;
    char ch;
    fptr= fopen("D:\\test.txt","r");
    if(fptr == NULL)
      printf("檔案開啟失敗");
    else{
      while((ch=fgetc(fptr)) != EOF)
        printf("%c",ch);
     fclose(fptr); }
}
```

檔案結構關鍵字

檔案指標,準備指向一個檔案

檔案名稱

開啟檔案

自檔案讀出字元

關閉檔案

一次讀入一個字元,然後印出,直到檔尾(EOF)為止

模式	作用
r	讀取
w	寫入
a	附加

檔案存取

- 檔案字串I/O(寫入)：

```
#include <stdio.h>
#include <string.h>
void main()
{
    FILE *fptr;
    char str[81];
    fptr= fopen("D:\\test.txt","w");
    if(fptr == NULL)
      printf("檔案開啟失敗");
    else{
      printf("請輸入字串，空白行結束：");
      while(strlen(gets(str)) > 0){
        fputs(str, fptr);
        fputs("\n", fptr);
      }
      fclose(fptr); }
}
```

使用字串必須引入此標頭檔

開啟檔案

寫入字串

關閉檔案

一次寫入一個字串,並加上換行字元,直到輸入字串長度為0止

檔案存取

- 檔案字串I/O(讀出)：

```
#include <stdio.h>
#include <string.h>
void main()
{
    FILE *fptr;
    char str[81];
    fptr= fopen("D:\\test.txt","r");
    if(fptr == NULL)
      printf("檔案開啟失敗");
    else{
      while(fgets(str,80,fptr) != NULL)
        printf("%s", str);
      fclose(fptr);
    }
}
```

開啟檔案

自檔案讀入一行字串

關閉檔案

一次讀出一行(最大長度為80字元),直到讀到空行(NULL)為止

檔案存取

- 讀出檔案時利用feof()函式來判斷是否以讀取完畢。

```
#include <stdio.h>
#include <string.h>
void main()
{
    FILE *fptr;
    char str[81];
    fptr= fopen("D:\\test.txt","r");
    if(fptr == NULL)
      printf("檔案開啟失敗");
    else
    {  while(!feof(fptr))
       {  fgets(str,80,fptr);
          printf("%s",str);
       }
       fclose(fptr);
    }
}
```

開啟檔案

使用feof()函數檢查是否已到檔尾

關閉檔案

檔案存取

- 檔案的傳回值：
  - 以fopen()開啟檔案成功會傳回檔案指標，失敗則會傳回NULL值。
  - fgetc()讀取成功會傳回字元，失敗則會傳回EOF。
  - fgets()讀取成功會傳回字串，失敗則會傳回NULL。
  - (應注意所使用之讀取函數及結束時傳回值。)

檔案存取

- 檔案的格式化I/O：
  - 當輸入的資料有各種型態時，就必須使用fprintf()及fscanf()函數。
  - 使用方式與printf()及scanf()相同，只是多了一個檔案指標的參數。
  - 例：

```
fprintf( fptr, "%s %d\n", a, b);
```

檔案指標　格式　變數

```
fscanf( fptr, "%s %d\n", a, b);
```

檔案存取

- 檔案格式化I/O(寫入)：

```
#include <stdio.h>
#include <string.h>
int main()
{   FILE *fptr; char id[10]; int score;
    if((fptr=fopen("D:\\test.txt","w"))==NULL)
    {   printf("檔案無法開啟!");
        return(1);
    }
    do
    {   printf("請輸入學號和成績(分數大於100時結束):");
        scanf("%s %d", id, &score);
        if(score<=100)
            fprintf(fptr,"%10s %3d\n", id, score);
    }while(score<=100);
    fclose(fptr);
}
```

檢查檔案是否開啟成功

結束程式

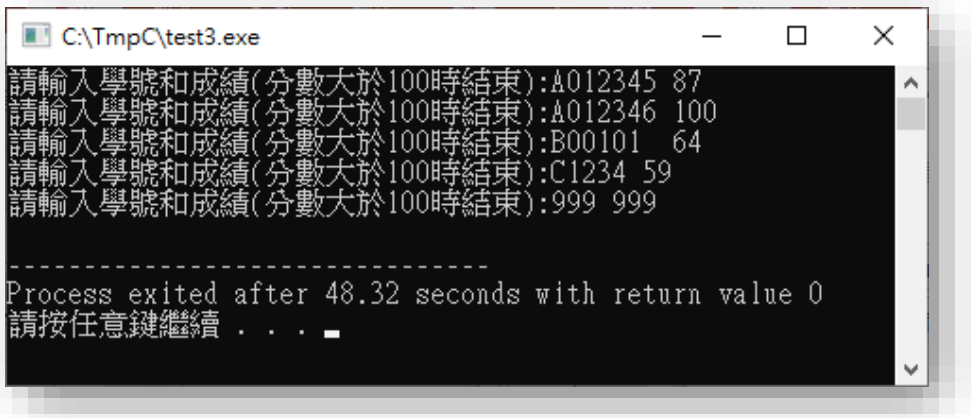
寫入前先確認資料是有效的

依指定格式寫入檔案

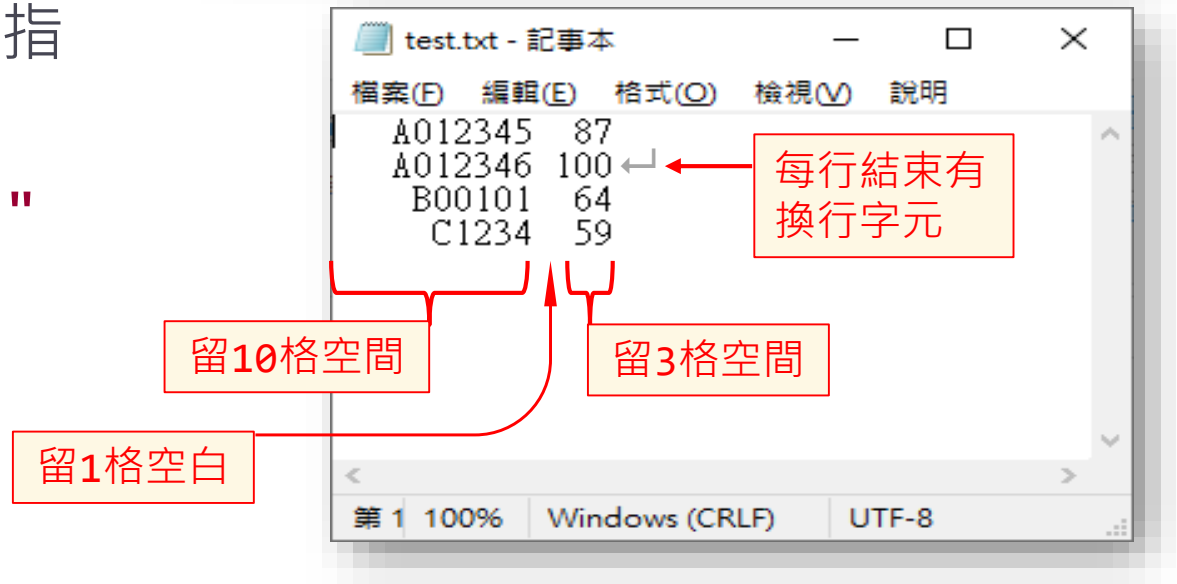
使用fprintf()函數依序將資料寫入檔案

檔案存取

- 前例執行結果：
  - 執行畫面。



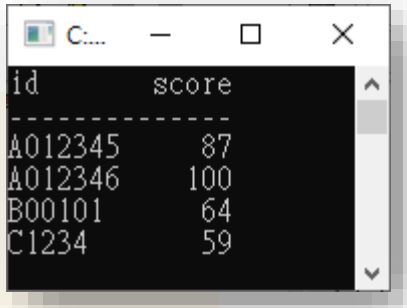
  - 所謂格式化是指依指定格式 `"%10s %3d\n"` 寫入檔案。



檔案存取

- 檔案格式化I/O(讀出)：

```
#include <stdio.h>
#include <string.h>
int main()
{   FILE *fptr; char id[10]; int score;
    if((fptr=fopen("D:\\test.txt","r")) == NULL)
    {
        printf("檔案無法開啟!");
        return(1);
    }
    printf("id        score\n--------------\n");
    while(fscanf(fptr,"%s %d", id, &score) != EOF)
       printf("%-10s %3d\n", id, score);
    fclose(fptr);
}
```



檢查檔案是否開啟成功

結束程式

依指定格式讀入每一行的資料

靠左對齊

使用fscanf()函數依序將資料讀出直到檔尾

實例練習

- 另發，請參考作業及講義。

下課~