深入淺出 C 語言

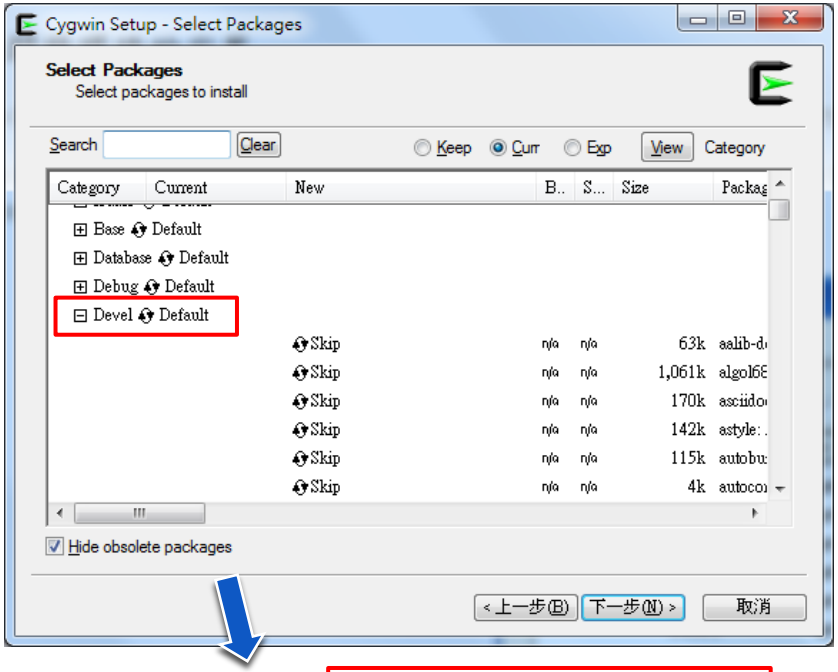
Chien-Jung Li

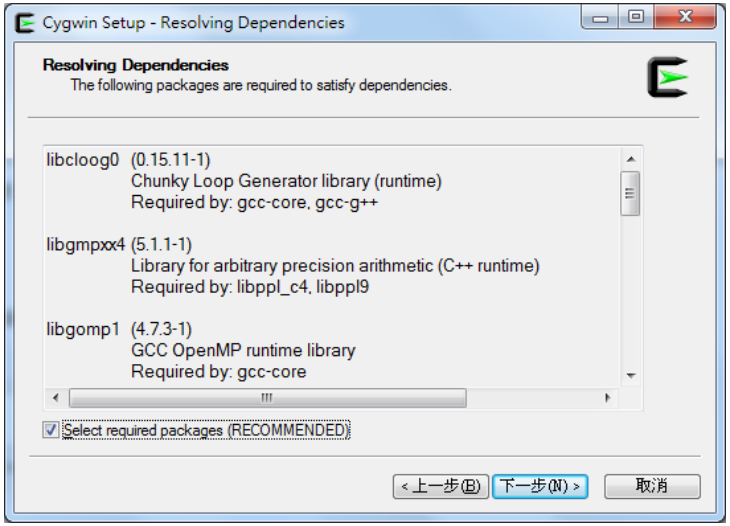
July 2013

準備工作

Cygwin可在Windows模擬類Linux環境

- 先到http://www.cygwin.com/下載Cygwin
- 選裝 Devel下的binutils, gcc, gcc-core, gcc-c++, make



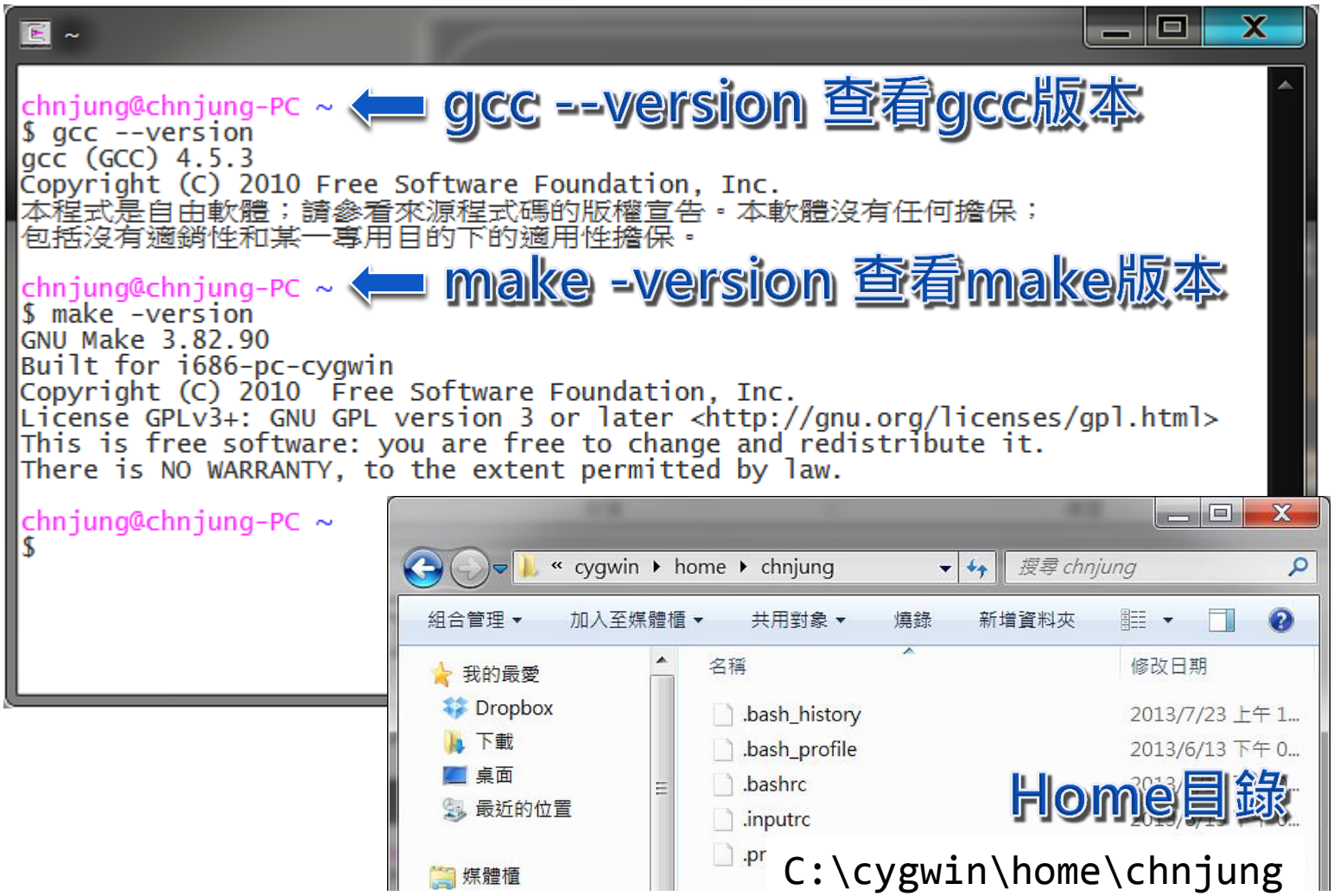


安裝程式會自動找出相關建議套件，按下一步一併安裝。

2.23.51-1	☒	☒	27,622k	binutils	The GNU assembler, linker and binary utilities
4.7.3-1	☒	☒	81,268k	gcc	GNU Compiler Collection
Skip	n/a	n/a	17,893k	gcc-ada	GNU Compiler Collection (Ada)
4.7.3-1	☒	☐	16,159k	gcc-core	GNU Compiler Collection (C, OpenMP)
Skip	n/a	n/a	5,907k	gcc-fortran	GNU Compiler Collection (Fortran)
4.7.3-1	☒	☐	8,430k	gcc-g++	GNU Compiler Collection (C++)
3.82.90-1	☒	☒	1,725k	make	The GNU version of the 'make' utility

執行Cygwin

- 執行C:\cygwin\cygwin.bat (一般安裝完桌面有捷徑)



其他實用軟體

- notepad++ 看/寫程式碼的編輯器
  http://notepad-plus-plus.org/

- FileSeek 可以搜尋「檔案」內容的好工具
  http://www.fileseek.ca/

- Dropbox 雲端硬碟，資料分享最方便
  https://www.dropbox.com/

- Printkey 2000 好用的螢幕抓圖小軟體。
  https://dl.dropboxusercontent.com/u/32423012/Printkey_2000_5.10.zip

寫程式這檔事

電腦只看得懂機器碼
(01010011....)

處理器廠商會規定指令集
(組合語言)

C語言更方便描述我要電腦做甚麼

更高階的其他語言可能更友善

- 寫程式的目的
  - ✓ 想要機器幫我做事
- 如果我想要請人做事
  - ✓ 直接做(像+, -, *, /)
  - ✓ 如果怎麼樣就...(if, case)
  - ✓ 如果要重複做(for, while)
  - ✓ 如果: 真與假(1與0)
- 大多數的程式，寫起來都是為了實現一套邏輯
- 不只這樣，如果我們能多了解一些語言本身的性質跟規則，就可寫出更好的程式以實現邏輯

程式要這樣學

- **不要背語法**

  程式語言那麼多種類，其實寫法都很類似，可是又有一點點不同。只要忘記怎麼寫，上網查一查或是工具書拿起來看一下就可以了。

- **了解語言的觀念**

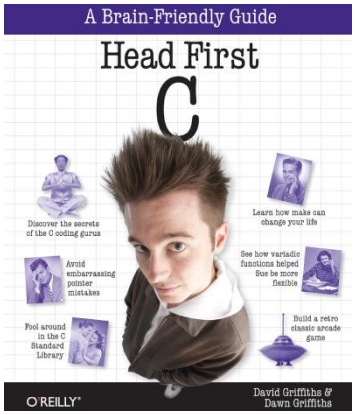
  程式語言的一些規則，往往是有一些理由存在的。觀念有時候比「程式寫作」的本身更重要。解決問題的方法很多種，一開始我們都會用暴力的方法來解決問題，但隨著觀念的進步，程式就能越寫越好。

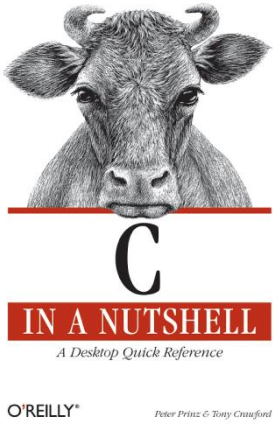
- **站在巨人的肩膀上**

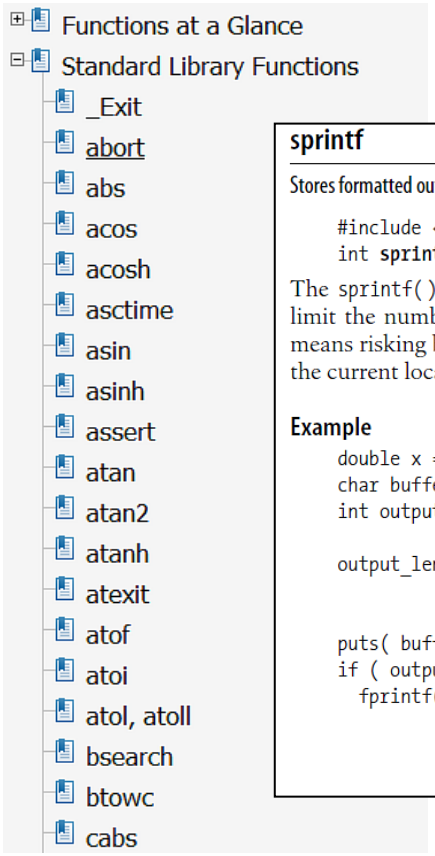
  不管程式語言有多少種類，總是有使用它們的人。既然有人使用，那一定會有程式範例，甚至也有現成的函式庫。觀念清楚了，我們就可以更容易爬上巨人的肩膀。

參考資料

- D. Griffiths, *A Brain-Friendly Guide – Head First C*, O'Reilly. (中文版為「深入淺出C」)
- P. Prinz, *C in a Nutshell*, O'Reilly.







sprintf

Stores formatted output in a string buffer

```
#include <stdio.h>
int sprintf( char * restrict dest, const char * restrict format, ... );
```

The sprintf( ) function is similar to snprintf( ), except that it has no parameter to limit the number of characters written to the destination buffer. As a result, using it means risking buffer overflows, especially because the length of the output depends on the current locale as well as input variables. Use snprintf( ) instead.

Example

```
double x = 1234.5, y = 678.9, z = -753.1, a = x * y + z;
char buffer[80];
int output_len = 0;

output_len = sprintf( buffer, "For the input values %lf, %lf, and %lf, "
                      "the result was %lf.\n",
                      x, y, z, a );
puts( buffer );
if ( output_len >= 80 )
  fprintf( stderr, "Output string overflowed by %d characters.\n"
                   "The variables x, y, z and a may have been corrupted:\n"
                   "x now contains %lf, y %lf, z %lf, and a %lf.\n",
                    output_len - 79, x, y, z, a );
```

Ch1 快速探索C語言

快速探索C語言

- 以現今的眼光來看，C語言已算是低階語言的一種。
- Why C?

```c
#include <stdio.h>
int main()
{
  puts("C Rocks!");
  return 0;
}
```

原始碼 rocks.c

```
> gcc rocks.c -o rocks
>
```

編譯 (gcc)

執行檔 rocks

執行

```
>./rocks
C Rocks!
```

典型的C語言程式

```
/* ← 註解：  /* */中間是註解 或 // 之後是註解
 * Program to calculate the number of cards in the shoe.
 * This code is released under the Vegas Public License.
 * (c)2014, The College Blackjack Team.
 */

#include <stdio.h>   ← 引入標頭檔、外部程式(庫)
int main()           ← 一定有主函數main()
{
    int decks;
    puts("Enter a number of decks");
    scanf("%i", &decks);
    if (decks < 1) {
        puts("That is not a valid number of decks");
        return 1;
    }
    printf("There are %i cards\n", (decks * 52));
    return 0;
}
```

關於printf()與scanf()

- **printf()展示格式化輸出：**

```
printf("%s says the count is %i", "Ben", 21);
```

呼叫printf()函式時，你想包含多少參數都可以，但要確認每個參數都有對應的%格式字元

格式字符	說明
%d	從鍵盤輸入十進制整數
%o	從鍵盤輸入八進制整數
%x	從鍵盤輸入十六進制整數
%c	從鍵盤輸入一個字符
%s	從鍵盤輸入一個字符串
%f	從鍵盤輸入一個實數

- **puts()只顯示字串輸出：**

```
puts("這裡只能顯示字串");
```

- **scanf ()從標準輸入讀取格式化資料：**

```
scanf("%i", &decks);
```

小程式 – 判斷撲克牌點數

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char card_name[3];
    puts("Enter the card_name: ");
    scanf("%2s", card_name);
    int val = 0;
    if (card_name[0] == 'K') {
        val = 10;
    } else if (card_name[0] == 'Q') {
        val = 10;
    } else if (card_name[0] == 'J') {
        val = 10;
    } else if (card_name[0] == 'A') {
        val = 11;
    } else {
        val = atoi(card_name);
    }
    printf("The card value is: %i\n", val);
    return 0;
}
```

為何要3字元? 我們待會就會學到

```
> gcc cards.c -o cards
> ./cards
Enter the card_name:
Q The card value is: 10
> ./cards
Enter the card_name:
A The card value is: 11
> ./cards
Enter the card_name:
7 The card value is: 7
```

如何在電腦上執行程式

- 使用免費的gcc(GNU Compiler Collection)來將C程式編譯為執行檔。

(1) 將上一頁程式碼存成cards.c

(2) 用gcc編譯：

```
> gcc cards.c -o cards
>
```

(3) 執行

```
> ./cards
Enter the card_name:
```

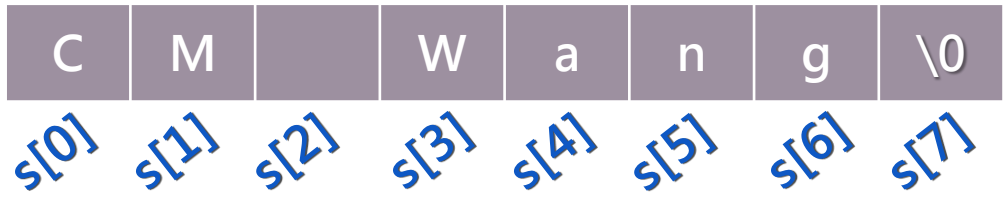
```
> gcc cards.c -o cards && ./cards
```

編譯成功並執行

C語言中的字串

- 原始C語言**只有字元**而沒有字串的觀念。其實字串只是字元的陣列。
- s = “CM Wang”
- s = {‘C’, ‘M’, ‘ ’, ‘W’, ‘a’, ‘n’, ‘g’}

C語言用哨兵字元「\0」來判斷字串的結尾



- **宣告**： char s[8] ← 宣告8個字元的陣列，從s[0]~s[7]
- ★ C語言的本身不會知道你的陣列有多長，我們得自己追蹤和紀錄我們的陣列。
- 字串實字是Constant，經宣告就不可改變個別字元。

命令分兩種

(1) 做某事

```
split_hand();
```

陳述式(statement)

```
{
  deal_first_card();
  deal_second_card();
  cards_in_hand();
}
```

區塊陳述式(block statement)

(2) 在某條件成立時，才做某事

```
if (value_of_hand <= 16)
    hit();
else
    stand();
```

```
if (dealer_card == 6) {
    double_down();
    hit();
}
```

評算牌點

cards2.c

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char card_name[3];
    puts("Enter the card_name: ");
    scanf("%2s", card_name);
    int val = 0;
    if (card_name[0] == 'K') {
        val = 10;
    } else if (card_name[0] == 'Q') {
        val = 10;
    } else if (card_name[0] == 'J') {
        val = 10;
    } else if (card_name[0] == 'A') {
        val = 11;
    } else {
        val = atoi(card_name);
    }
    /* Check if the value is 3 to 6 */
    if ((val > 2) && (val < 7))
        puts("Count has gone up");
    /* Otherwise check if the card was 10, J, Q, or K */
    else if (val == 10)
        puts("Count has gone down");
    return 0;
}
```

&&: 檢查兩條件是否都成立
||: 檢查兩條件是否至少一個成立
!: 讓結果相反

檢查條件

- &&檢查兩個條件是否都成立

```
if ((dealer_up_card == 6) && (hand == 11))
    double_down();
```

- ||檢查兩個條件是否至少有一個成立

```
if (cupcakes_in_fridge || chips_on_table)
    eat_food();
```

- ! 讓評算結果相反

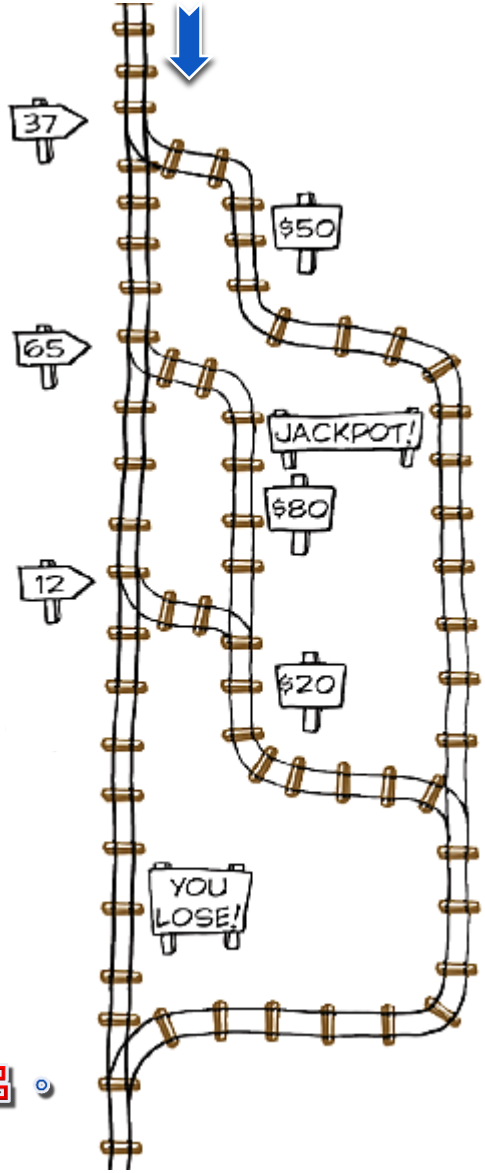
```
if (!brad_on_phone)
    answer_phone();
```

- 能僅使用&與|嗎？ 可以，但它們會把兩個條件都算過。

& 與 | 的另一個用途是做bitwise的操作: 6 & 4 = 4

switch陳述式進行邏輯測試

```
switch(train) {
   case 37:
        winnings = winnings + 50;
        break;
   case 65:
        puts("Jackpot!");
        winnings = winnings + 80;
   case 12:
        winnings = winnings + 20;
        break;
   default:
        puts("You Lose!");
        winnings = 0;
}
```



注意case區段結尾的break，漏掉它很容易讓程式有bug。

switch(x)只能檢查“單一”數值，不能檢查字串。

用switch改寫評算牌點程式

cards3.c

```
int val = 0;
if (card_name[0] == 'K') {
    val = 10;
} else if (card_name[0] == 'Q') {
    val = 10;
} else if (card_name[0] == 'J') {
    val = 10;
} else if (card_name[0] == 'A') {
    val = 11;
} else {
    val = atoi(card_name);
}
```

```
switch(card_name[0]) {
    case 'K':
    case 'Q':
    case 'J':
        val = 10;
        break;
    case 'A':
        val = 11;
        break;
    default:
    val = atoi(card_name);
}
```

重複做某事：while迴圈

- 迴圈是一種控制陳述式(control statement)
- while迴圈：只要某條件成立，就一直重複做某事

```
while ( <some condition> ) {

   ... /* Do something here */

}
```

- do...while迴圈：程式碼至少會被執行一次

```
do {

   /* Buy lottery ticket */

} while(have_not_won);
```

重複做某事N次

使用while來做

```
int counter = 1;
while (counter < 11) {
   printf("%i green bottles, hanging on a wall\n", counter);
   counter++;
}
```

使用for迴圈更容易

```
int counter;
for (counter = 1; counter < 11; counter++) {
   printf("%i green bottles, hanging on a wall\n", counter);
}
```

中斷迴圈與繼續迴圈

```
while(feeling_hungry) {
    eat_cake();
    if (feeling_queasy) {
        /* Break out of the while loop */
        break;
    }
    drink_coffee();
}
```

```
while(feeling_hungry) {
    if (not_lunch_yet) {
        /* Go back to the loop condition */
        continue;
    }
    eat_cake();
}
```

break不能用來中斷if，那是迴圈在用的

再探函式

comp_val.c

```c
#include <stdio.h>

int larger(int a, int b)
{
    if (a > b)
        return a;

    return b;
}

int main()
{
    int greatest = larger(100, 1000);
    printf("%i is the greatest!\n", greatest);
    return 0;
}
```

```c
void complain()
{
    puts("I'm really not happy");
}
```

void不用return

在C語言裡，幾乎所有一切都有回傳值；
所以可以做串接指定(chaining assignment)

```
x = 4;   ==   y = (x = 4);   ==   y = x = 4;
```

按下X離開

cards4.c

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char card_name[3];
    int count = 0;
    do{
        puts("Enter the card_name: ");
        scanf("%2s", card_name);
        int val = 0;
        switch(card_name[0]) {
            case 'K':
            case 'Q':
            case 'J':
                val = 10;
                break;
            case 'A':
                val = 11;
                break;
            case 'X':
                continue;
            default:
                val = atoi(card_name);
                if ((val < 1) || (val > 10)) {
                puts("I don't understand that value!");
                continue;
            }
        }
        if ((val > 2) && (val < 7)){
                count++;
        }else if (val == 10){
                count--;
        }
        printf("Current count: %i\n", count);
    } while (card_name[0]!='X');
    return 0;
}
```

Ch2 記憶體與指標(pointer)

記憶體與指標

addr_show.c

- 指標(pointer)是C語言裡必須要了解的最基礎之一。

```c
#include <stdio.h>
int y = 1;

int main()
{
    int x = 4;

    printf("x is stored at %p\n", &x);
    return 0;
}
```

函式外: 存在globals(全域)

函式內: 存在stack(區域)

格式化位址　取址

```
x is stored at 0x28AC2C
>
```

更多關於指標(I)

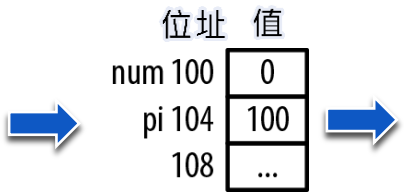
- **指標的讀法：由後往前讀**

1. pci is a variable	`const int *pci;`
2. pci is a pointer variable	`const int *pci;`
3. pci is a pointer variable to an integer	`const int *pci;`
4. pci is a pointer variable to a constant integer	`const int *pci;`

- **位址操作**

```
int num;
int *pi;
```

```
num = 0;
pi = &num;
```




```
num 為 0
pi 為 100
```

```
num 為 0
*pi 為 0
```

```
num = 0;
pi = num;
```

➡ 會出現編譯錯誤　error: invalid conversion from 'int' to 'int*'

```
num = 0;
pi = (int *)num;
```

➡ 不會有編譯錯誤，但pi會指向錯的地方

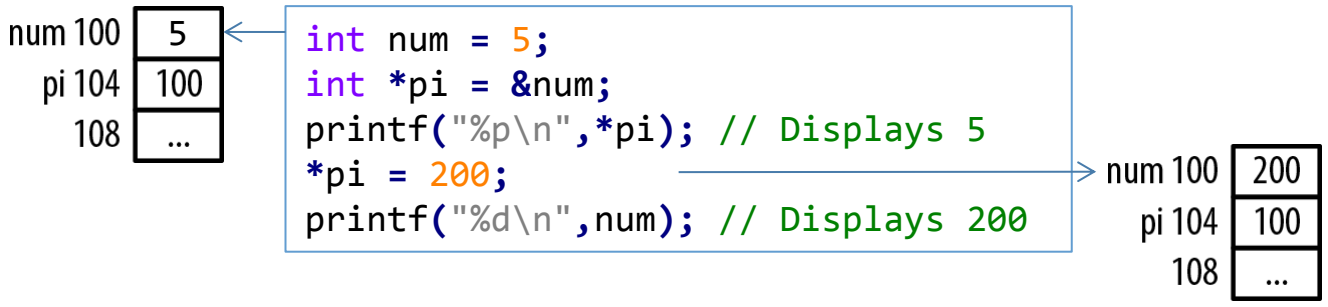
更多關於指標(II)

```
int num = 0;
int *pi = &num;
printf("Address of num: %d Value: %d\n",&num, num);
printf("Address of pi: %d Value: %d\n",&pi, pi);
```

```
Address of num: 4520836 Value: 0
Address of pi: 4520824 Value: 4520836
```

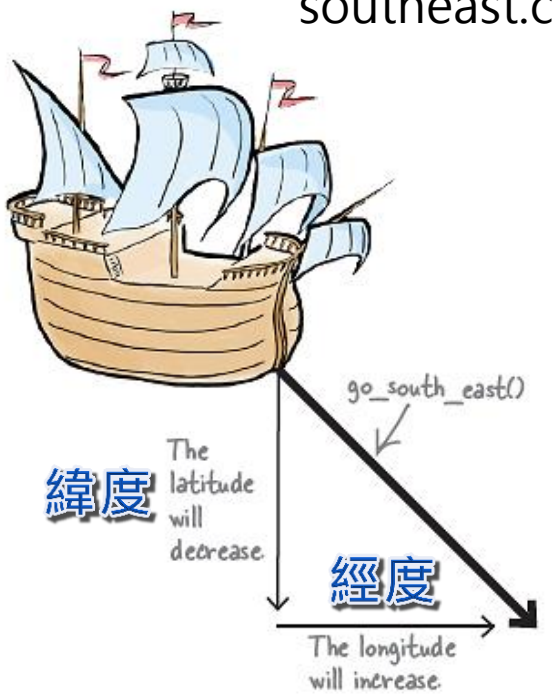
```
Address of pi: 4520824 Value: 4520836      // %d
Address of pi: 44fb78 Value: 44fb84        // %x
Address of pi: 21175570 Value: 21175604    // %o
Address of pi: 0044FB78 Value: 0044FB84    // %p
```

- **指標所指到的值**



航行吧!

southeast.c



Call-by-value (傳值呼叫):

```c
#include <stdio.h>
void go_south_east(int lat, int lon)
{
    lat = lat - 1;
    lon = lon + 1;
}

int main()
{
    int latitude = 32;
    int longitude = -64;
    go_south_east(latitude, longitude);
    printf("Avast! Now at: [%i, %i]\n", latitude, longitude);
    return 0;
}
```

船沒動啊?! 怎麼回事?

```
> gcc southeast.c -o southeast
> ./southeast
Avast! Now at: [32, -64]
>
```

傳值呼叫: 改變的是副本

航行吧!

Call-by-reference (傳址呼叫):

```c
#include <stdio.h>
void go_south_east(int *lat, int *lon)
{
    *lat = *lat - 1;
    *lon = *lon + 1;
}

int main()
{
    int latitude = 32;
    int longitude = -64;
    go_south_east(&latitude, &longitude);
    printf("Avast! Now at: [%i, %i]\n", latitude, longitude);
    return 0;
}
```

指標(pointer)

```c
int x = 4;
printf("x lives at %p\n", &x);
```

```c
int *address_of_x = &x;
```

```c
int value_stored = *address_of_x;
```

```c
*address_of_x = 99;
```

```
> gcc southeast.c -o southeast
> ./southeast
Avast! Now at: [31, -63]
>
```

如何傳遞字串給函式

```c
#include <stdio.h>

void fortune_cookie(char msg[])
{
    printf("Message reads: %s\n", msg);
}

int main()
{
    char quote[] = "Cookies make you fat";
    fortune_cookie(quote);
    return 0;
}
```

引數定義像陣列，但你永遠不知道傳入的字串有多長

```c
void fortune_cookie(char msg[])
{
    printf("Message reads: %s\n", msg);
    printf("Msg occupies %i bytes\n", sizeof(msg));

}
```

sizeof運算子可以告訴你某東西佔用多少bytes

```
> ./fortune_cookie
Message reads: Cookies make you fat
msg occupies 4 bytes X
>
```

有些機器可能會傳回8

陣列變數就像指標

where_is_quote.c

```
char quote[] = "Cookies make you fat";
```


```
printf("The quote string is stored at: %p\n", quote);
```

```
> ./where_is_quote
The quote string is stored at: 0x28AC1B
>
```

但陣列變數不完全是指標

```
char s[] = "How big is it?";
char *t = s;
```

陣列的位址就是「陣列的位址」

```
&s == s
&t != t
```

陣列變數不能指向任何其他地方：

```
s = t
```

會得到編譯錯誤

指標退化(pointer decay)： `t = s`

傳遞陣列給函式時，C語言會將其退化成指標。

陣列從0開始索引

drinks.c

```c
int drinks[] = {4, 2, 3};
printf("1st order: %i drinks\n", drinks[0]);
printf("1st order: %i drinks\n", *drinks);

printf("3rd order: %i drinks\n", drinks[2]);
printf("3rd order: %i drinks\n", *(drinks + 2));
```

drinks[0] == *drinks

drinks[2] == *(drinks+2)


```
> ./drinks
1st order: 4 drinks
1st order: 4 drinks
3rd order: 3 drinks
3rd order: 3 drinks
```

指標有型別

print_nums.c

- 指標運算在檯面下進行，為char指標加1，該指標就會指向下一個記憶體位址，因為一個char字元剛好等於1 byte。
- 在32位元的電腦，int是以4-bytes表示，所以為int的指標加1，編譯出的程式就會把位址增加4到新的記憶體位址。

```
int nums[] = {1, 2, 3};
printf("nums is at %p\n", nums);
printf("nums + 1 is at %p\n", nums + 1);
```

在32位元電腦執行的結果

```
> ./print_nums
nums is at 0x28AC24
nums + 1 is at 0x28AC28
```

跳了4 bytes

指標要指定型別，編譯器才知道如何計算位址。

把指標用於資料輸入

name_test.c

```
char name[40];
printf("Enter your name: ");
scanf("%39s", name);
```

讀進39字元，最後一個保留給「/0」

使用scanf()輸入數字

```
int age;
printf("Enter your age: ")
scanf("%i", &age);
```

存入

將數值變數的位址傳進函式，scanf()可以更新該變數的內容。

```
char first_name[20];
char last_name[20];
printf("Enter first and last name: ");
scanf("%19s %19s", first_name, last_name);
printf("First: %s Last:%s\n", first_name, last_name);
```

```
> ./name_test
Enter your name: chnjung
Enter your age: 34
Your name is chnjung, and you are 34 years old.
Enter first and last name: chnjung Li
First: chnjung Last:Li
```

小心使用scanf()

```
char food[5];
printf("Enter favorite food: ");
scanf("%s", food);
printf("Favorite food: %s\n", food);
```

沒有設定掃入的字元數

```
> ./food
Enter favorite food: orange-tea-with-cake-and-cookies
Favorite food: orange-tea-with-cake-and-cookies
Segmentation fault (core dumped)
>
```

程式因緩衝區溢位而當掉，因為你鍵入的字元數超過food[5]陣列所配置到的記憶體空間囉

```
char food[5];
printf("Enter favorite food: ");
fgets(food, sizeof(food), stdin);
```

改用fgets就不會忘記限制長度(包含/0，不需像scanf()要將長度減1)

```
char *food;
printf("Enter favorite food: ");
fgets(food, 5, stdin);
```

food若是指標，則要指明長度

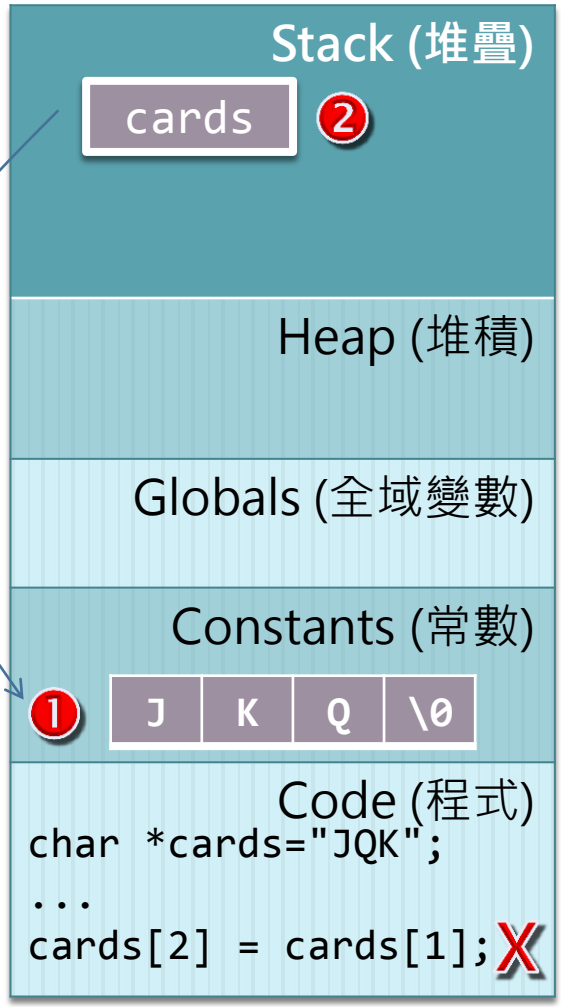
字串實字永遠不能被更新

指向字串實字的變數，不能被用來改變該字串的內容：

```
char *cards = "JQK";
```

為什麼？ 這全都是因為C語言使用記憶體的方式!

❸cards變數所存的位址指向❶




```
char *cards = "JQK";
```

這樣宣告本身並沒有錯誤，而是錯誤只會發生在你要試圖修改字串實字時。如果真的要這樣宣告，請養成好習慣，在前面加const關鍵字，這樣你在試圖修改實自字串時，編譯器就會給你一個錯誤！

```
const char *cards = "JQK";
```

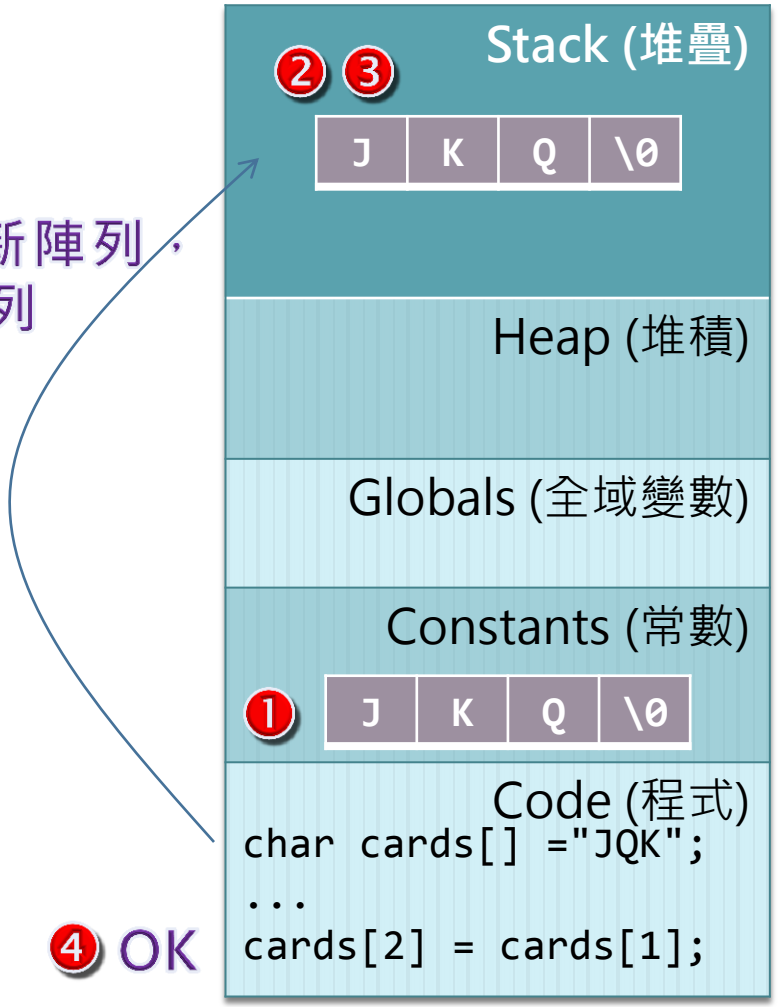
❹ 行不通

要改變字串，就產生一份副本

如果你從字串實字重新建立一個陣列，就能夠修改該陣列：

```
char cards[] = "JQK";
```

在stack建立新陣列，並初始化該陣列



cards[] 或 *cards?

`char cards[]` 此宣告的意思?

這取決於你在何處看到它...

一般的變數宣告，表示cards是陣列：

```
int my_function()
{
    char cards[] = "JQK";
    ...
}
```

未提供陣列尺寸，你必須馬上把它設定為某個值。

若是函數的引數，表示cards是一個指標：

```
void stack_deck(char cards[])
{
    ...
}
```

指標退化!

同義!

```
void stack_deck(char *cards)
{
    ...
}
```

Ch3 字串理論

字串不只用來讀取

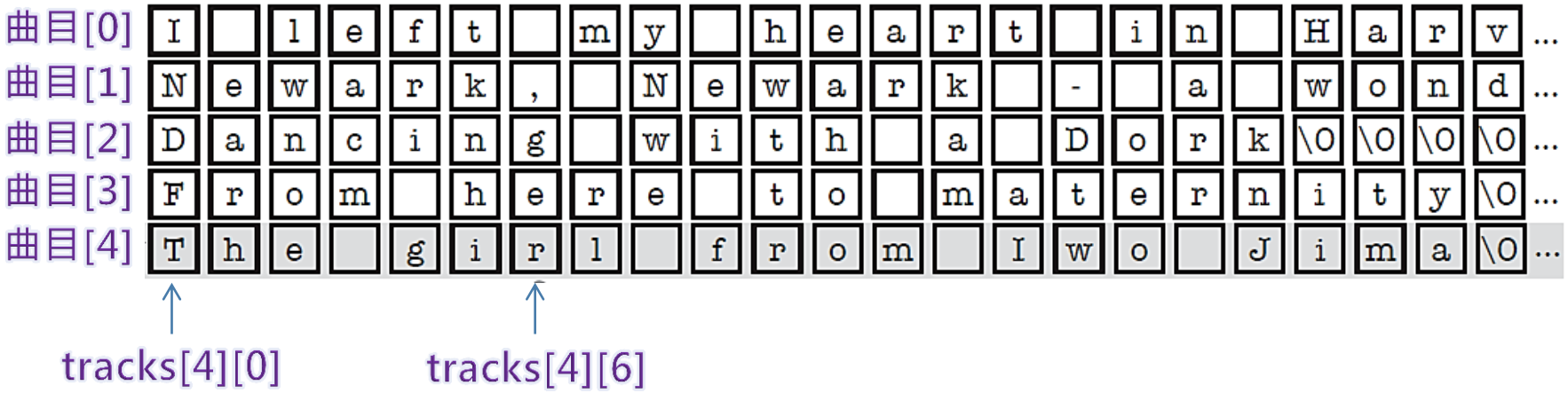
- 在C語言裡，字串實際上是字元陣列。
- 如果我們想操作的對象是「字串」，例如串接、比較、複製字串等，可以想見要做複雜的「字元」處理肯定很惱人。所幸，C語言的標準程式庫string.h可以幫得上忙。

自動點唱機的曲目

建立陣列的陣列：(假設每一首歌的名字都不會超過79個字元)

字串陣列　　個別字串的字元陣列

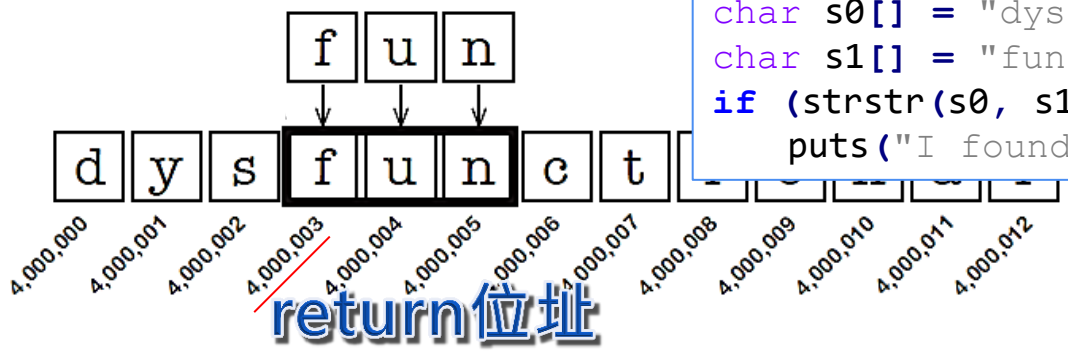
```
char tracks[][80] = {
    "I left my heart in Harvard Med School",
    "Newark, Newark - a wonderful town",
    "Dancing with a Dork",
    "From here to maternity",
    "The girl from Iwo Jima",
};
```



string.h提供的函式

- strchr(): 在字串裡搜尋某個字元的位置
- strcmp(): 比較兩個字串
- strstr(): 在另一個字串裡尋找某個字串的位置
- strcpy(): 將一個字串複製到另一個
- strlen(): 判斷字串長度
- strcat(): 串聯兩個字串

```
strstr("dysfunctional", "fun")
```




```
char s0[] = "dysfunctional";
char s1[] = "fun";
if (strstr(s0, s1))
    puts("I found the fun in dysfunctional!");
```

傳回非0，表示有找到

使用strstr()尋找曲目

text_search.c

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

char tracks[][80] = {
    "I left my heart in Harvard Med School",
    "Newark, Newark - a wonderful town",
    "Dancing with a Dork",
    "From here to maternity",
    "The girl from Iwo Jima",
};

void find_track(char search_for[])
{
    int i;
    for (i = 0; i < 5; i++) {
        if (strstr(tracks[i], search_for))
            printf("Track %i: '%s'\n", i, tracks[i]);
    }
}
```

```c
int main()
{
    char search_for[80];
    printf("Search for: ");
    scanf("%s", search_for);
    find_track(search_for);
    return 0;
}
```

Ch4 小工具

小工具

任務：某GPS所記錄的行車軌跡資料如下

GPS原始資料
(gps.csv)

緯度	經度	速度
42.363400,	-71.098465,	Speed = 21
42.363327,	-71.097588,	Speed = 23
42.363255,	-71.096710,	Speed = 17

我們希望能將其轉換成google map能顯示的格式(json)

```
data=[
   {latitude: 42.363400, longitude: -71.098465, info: 'Speed = 21'},
   {latitude: 42.363327, longitude: -71.097588, info: 'Speed = 23'},
   {latitude: 42.363255, longitude: -71.096710, info: 'Speed = 17'},
   ...
]
```

請寫一支小工具程式來完成這件任務吧!

寫一支處理程式：geo2json.c

```c
#include <stdio.h>
int main()
{
    float lat;
    float lng;
    char info[80];
    int started = 0;

    puts("data=[");
    while (scanf("%f,%f,%79[^\r]",  &lat, &lng, info) == 3) {

        if (started){
          printf(",\n");
        }else{
          started = 1;
        }

        printf("{latitude: %f, longitude: %f, info: '%s'}", lat, lng, info);
    }
    puts("\n]");
    return 0;
}
```

不同系統的換行
Windows: /r/n
Linux或Mac: /r或/n

給我最多到該行的結尾

scanf傳回讀值之數量

前一行有東西時，才接，

執行結果

每組資料要自己一個一個key進去，
key完馬上顯示新格式排列結果。

```
>./geo2json
data=[
42.363400,-71.098465,Speed = 21
{latitude: 42.363400, longitude: -71.098465, info: 'Speed = 21'}42.363327,-
71.097588,Speed = 23
,
{latitude: 42.363327, longitude: -71.097588, info: 'Speed =23'}42.363255,-
71.096710,Speed = 17
,
{latitude: 42.363255, longitude: -71.096710, info: 'Speed =17'}42.363182,-
71.095833,Speed = 22
,
...
...
...
{latitude: 42.363182, longitude: -71.095833, info: 'Speed = 22'}42.362385,-
71.086182,Speed = 21
,
{latitude: 42.362385, longitude: -71.086182, info: 'Speed =21'}^D
]
```

資料一筆一筆keyin太麻煩，如果程式能直接處理檔案資料就太好了。

處理程式(用檔案)：使用重導向

從標準輸入收進data

data由標準輸出送出

gpsdata.csv

```
42.3...00,-71.098465,Speed = 21
42.3...27,-71.097588,Speed = 23
42.3...10,Speed = 17
42.363182,-71.095833,Speed = 22
42.363110,-71.094955,Speed = 14
42.363037,-71.094078,Speed = 16
42.362965,-71.093201,Speed = 18
42.362892,-...
42.362820,-...
42.362747,-...
42.362675,-...
42.362602,-...
42.362530,-...
42.362457,-...
42.362385,-...
```

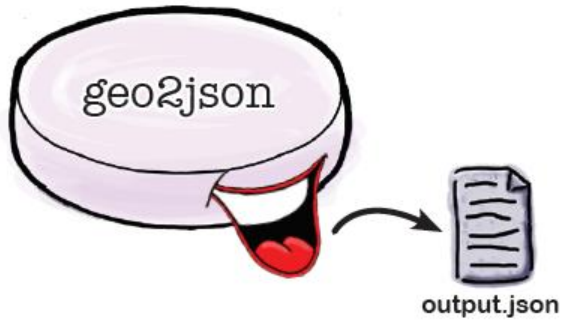
將資料檔重導向，作為geo2json程式的輸入

```
> ./geo2json < gpsdata.csv
data=[
{latitude: 42.363400, longitude: -71.098465, info: 'Speed = 21'},
{latitude: 42.363327, longitude: -71.097588, info: 'Speed = 23'},
{latitude: 42.363255, longitude: -71.096710, info: 'Speed = 17'},
{latitude: 42.363182, longitude: -71.095833, info: 'Speed = 22'},
{latitude: 42.363110, longitude: -71.094955, info: 'Speed = 14'},
{latitude: 42.363037, longitude: -71.094078, info: 'Speed = 16'},
...
...
{latitude: 42.362385, longitude: -71.086182, info: 'Speed = 21'}
]
>
```

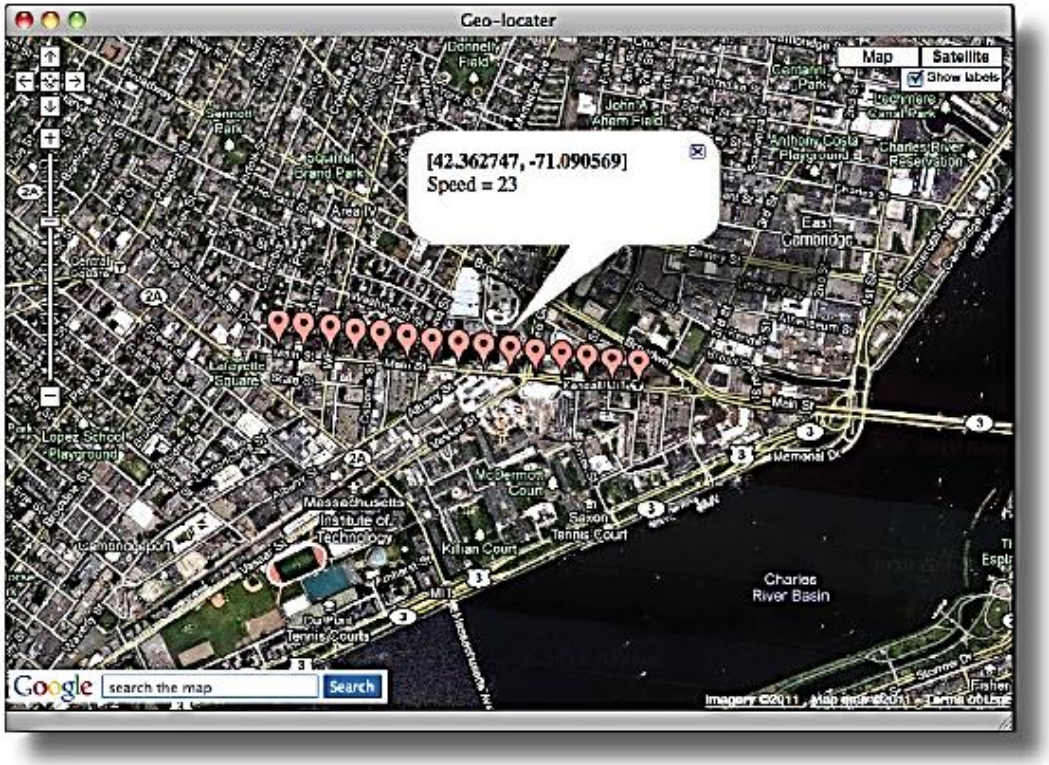
將輸出重導向至檔案

```
> ./geo2json < gpsdata.csv > output.json
>
```

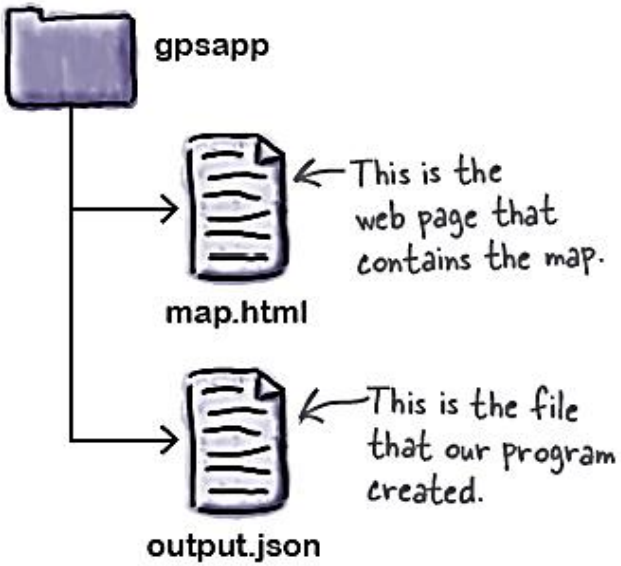
```
data=[
{latitude: 42.363400, longitude: -71.098465, info: 'Speed = 21'},
{latitude: 42.363327, longitude: -71.097588, info: 'Speed = 23'},
{latitude: 42.363255, longitude: -71.096710, info: 'Speed = 17'},
{latitude: 42.363182, longitude: -71.095833, info: 'Speed = 22'},
{latitude: 42.363110, longitude: -71.094955, info: 'Speed = 14'},
{latitude: 42.363037, longitude: -71.094078, info: 'Speed = 16'},
{latitude: 42.362965, longitude: -71.093201, info: 'Speed = 18'},
{latitude: 42.362892, longitude: -71.092323, info: 'Speed = 22'},
{latitude: 42.362820, longitude: -71.091446, info: 'Speed = 17'},
{latitude: 42.362747, longitude: -71.090569, info: 'Speed = 23'},
{latitude: 42.362675, longitude: -71.089691, info: 'Speed = 14'},
{latitude: 42.362602, longitude: -71.088814, info: 'Speed = 19'},
{latitude: 42.362530, longitude: -71.087936, info: 'Speed = 16'},
62457, longitude: -71.087059, info: 'Speed = 16'},
62385, longitude: -71.086182, info: 'Speed = 21'}
```



以google map顯示導航軌跡圖



Download the web page from *http://oreillyhfc.appspot.com/map.html*.



使用管線(pipe)

```
#include <stdio.h>
int main()
{
    float latitude;
    float longitude;
    char info[80];
    while (scanf("%f,%f,%79[^\n]", &latitude, &longitude, info) == 3){
    if ((latitude > 26)&&(latitude < 36))
      if ((longitude > -72)&&(longitude < -62))
        printf("%f,%f,%s\n", latitude, longitude, info);
    }
    return 0;
}
```

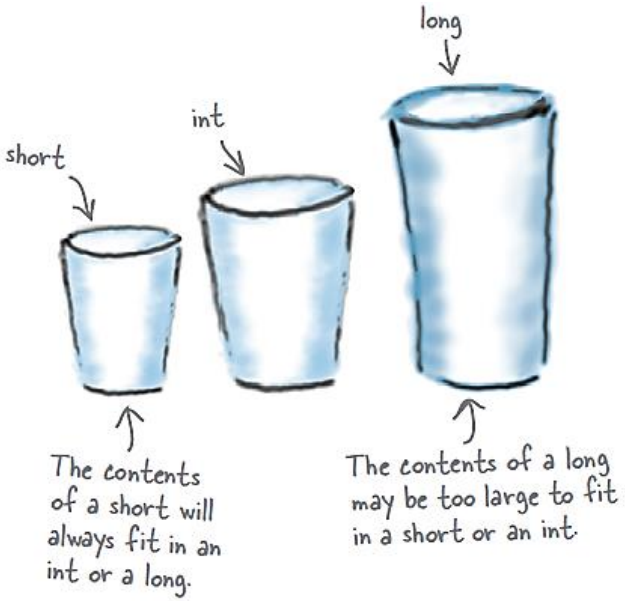
```
30.685163,-68.137207,Type=
28.304380,-74.575195,Type=
29.132971,-71.136475,Type=
28.343065,-62.753906,Type=
27.868217,-
68.005371,Type=Goatsucker
30.496017,-
73.333740,Type=Disappearance
26.224447,-71.477051,Type=UFO
```


```
(./bermuda | ./geo2json) < spooky.csv > output2.json
```

Ch5 使用多個原始檔

資料型別

- 資料型別：char、int、short、long、float、double



別把大東西塞進小容器

```
short x = 15;
int y = x;
printf("The value of y = %i\n", y);
```

➡ The value of y =15

```
int x = 100000;
short y = x;
print("The value of y = %hi\n", y);
```

➡ The value of y = -31072

強制轉型(cast)

```
int x = 7;
int y = 2;
float z = x / y;
printf("z = %f\n", z);
```

z =3

用強制轉型將整數轉成浮點數

```
int x = 7;
int y = 2;
float z = (float)x / (float)y;
printf("z = %f\n", z);
```

z =3.5

也可

```
float z = (float)x / y;
```

帳單程式: totaller.c

```c
#include <stdio.h>

float total = 0.0;
short count = 0;
short tax_percent = 6;

int main()
{
    float val;
    printf("Price of item: ");
    while (scanf("%f", &val) == 1) {
          printf("Total so far: %.2f\n", add_with_tax(val));
          printf("Price of item: ");
    }
    printf("\nFinal total: %.2f\n", total);
    printf("Number of items: %hi\n", count);
    return 0;
}

float add_with_tax(float f)
{
    float tax_rate = 1 + tax_percent / 100.0;
    total = total + (f * tax_rate);
    count = count + 1;
    return total;
}
```

此程式編譯會出錯，因為add_with_tax()沒有宣告在main()之前。

將定義與宣告分開

- 請把函式按順序排好，並且在你於main()中呼叫它之前先宣告(或定義)好。

函式宣告：

```
float add_with_tax(float f);
```

宣告只是一種函式簽名：記錄著函式被稱做甚麼、將接受何種參數以及將傳回甚麼資料型別。

- 若不想管函式排列的順序，可在C程式開頭處先做一系列的宣告。

```
float do_something_fantastic();
double awesomeness_2_dot_0();
int stinky_pete();
char make_maguerita(int count);
```

宣告可獨立成標頭檔(header)再引入使用

建立自己的標頭檔

totaller.h

```
float add_with_tax(float f);
```

totaller.c

```
#include <stdio.h>
#include "totaller.h"
...
```

```
> gcc totaller.c -o totaller
> ./totaller
Price of item: 1.23
Total so far: 1.30
Price of item: 4.57
Total so far: 6.15
Price of item: 11.92
Total so far: 18.78
Price of item: ^D
Final total: 18.78
Number of items: 3
```

通用的功能：共享程式碼

程式一：檔案加密
file_hider

程式二：訊息加密
message_hider

都會用到

加密程式 encrypt()

```
void encrypt(char *message)
{
    char c;
    while (*message) {
        *message = *message ^ 31;
        message++;
    }
}
```

共用程式碼需要自己的標頭檔

encrypt.h

```
void encrypt(char *message)
```

message_hider

```
#include <stdio.h>
#include "encrypt.h"
int main()
{
    char msg[80];
    while (fgets(msg, 80, stdin)) {
        encrypt(msg);
        printf("%s", msg);
    }
}
```

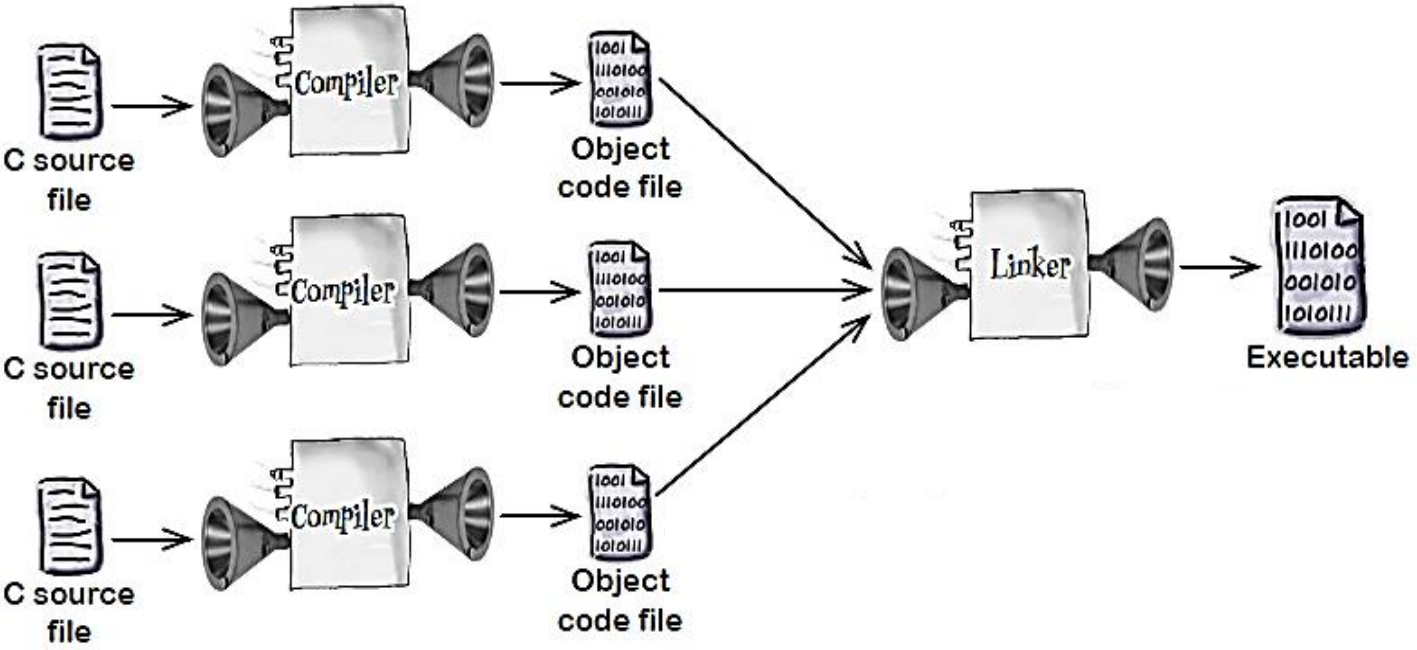
```
#include "encrypt.h"
void encrypt(char *message)
{
    char c;
    while (*message) {
        *message = *message ^ 31;
        message++;
    }
}
```

encrypt.c

有用到的原始碼都要編入

```
> gcc message_hider.c encrypt.c -o message_hider
> ./message_hider
I am a secret message
V?~r?~?lz|mzk?rzll~xz
> ./message_hider < encrypt.h
ipv{?zq|mfok7|w~m5?rzll~xz6$
>
```

編譯



若只有其中一個「原始碼」更動過，就要重編所有檔案會很浪費時間。要怎麼做才不需重新編譯所有檔案呢？

`gcc -c *.c` 把原始碼編譯成目的檔 (用參數 -c)

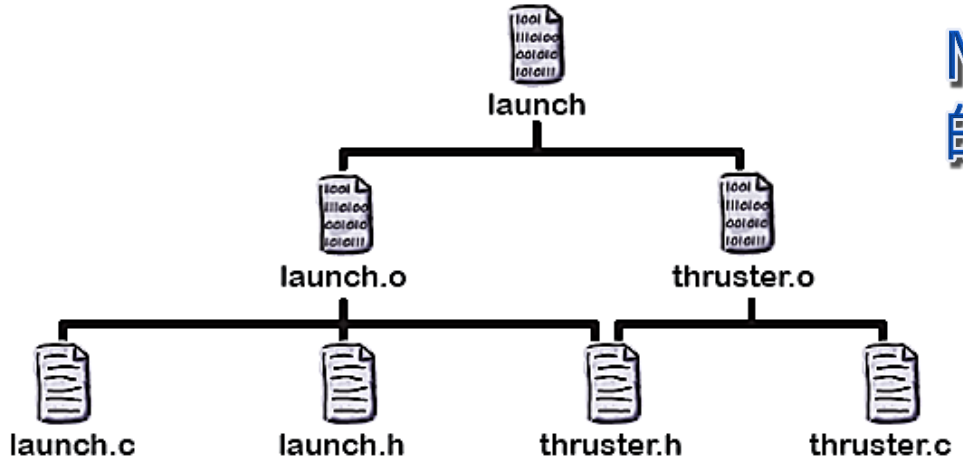
`gcc *.o -o launch` 將他們連結起來 (參數 -o)

只有thruster.c更動過

`gcc -c thruster.c` 只要重編修改過檔案的目的檔

`gcc *.o -o launch` 然後重新連結所有目的檔成為執行檔即可

make：自動化建置工作



Makefile（提供make的資訊給make）

Rule1
Rule2
Rule3

```
launch.o: launch.c launch.h thruster.h
	gcc -c launch.c
thruster.o: thruster.h thruster.c
	gcc -c thruster.c
launch: launch.o thruster.o
	gcc launch.o thruster.o -o launch
```

Target

Receipt一定要以tab開頭(規定)

```
> make launch
gcc -c launch.c
gcc -c thruster.c
gcc launch.o thruster.o -o launch
```

只有被改變的才會重編譯

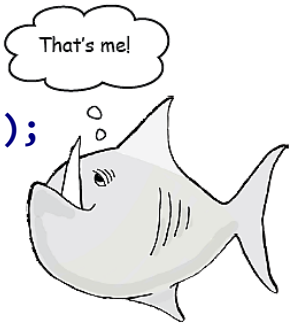
```
> make launch
gcc -c thruster.c
gcc launch.o thruster.o -o launch
```

Ch6 struct、union與bitfield

結構

- strcut 讓你透過撰寫自己的結構來塑模真實世界的複雜性。有時候你必須處理大量資料：

```
/* Print out the catalog entry */
void catalog(const char *name, const char *species, int teeth, int age)
{
  printf("%s is a %s with %i teeth. He is %i\n",name, species, teeth, age);
}

/* Print the label for the tank */
void label(const char *name, const char *species, int teeth, int age)
{
  printf("Name:%s\nSpecies:%s\n%i years old, %i teeth\n",name, species, teeth, age);
}
```

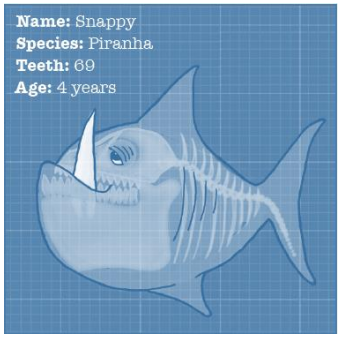



```
int main()
{
  catalog("Snappy", "Piranha", 69, 4);
  label("Snappy", "Piranha", 69, 4);
  return 0;
}
```

如何避免傳遞許多資料

用strcut建立自己的資料型別

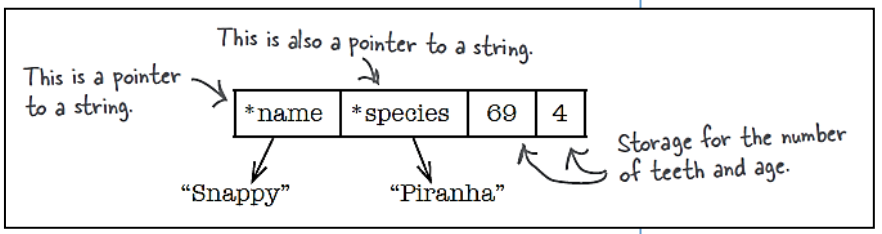
```
struct fish {
    const char *name;
    const char *species;
    int teeth;
    int age;
};
```



將不同資料片段包成一個新的大資料型別

```
/* Print out the catalog entry */
void catalog(struct fish f)
{
    ...
}
/* Print the label for the tank */
void label(struct fish f)
{
    ...
}

int main()
{
    struct fish snappy = {"Snappy", "Piranha", 69, 4};
    catalog(snappy);
    label(snappy);
    return 0;
}
```

傳struct給函式



用新型別建立一隻fish

按順序填入

使用struct的好處

```
struct fish {
    const char *name;
    const char *species;
    int teeth;
    int age;
    int favorite_music;
};
```

你可以改變struct的內容而不需改變使用它的函式

以struct包裹參數，可以讓你的程式更加穩固！

使用「.」運算子讀取欄位

```
struct fish snappy = {"Snappy", "piranha", 69, 4};
printf("Name = %s\n", snappy.name);
```

```
> gcc fish.c -o fish
> ./fish
Name = Snappy
>
```

```
void catalog(struct fish f)
{
    printf("%s is a %s with %i teeth. He is %i\n", f.name, f.species, f.teeth,
    f.age);
}

int main()
{
    struct fish snappy = {"Snappy", "Piranha", 69, 4};
    catalog(snappy);
    /* We're skipping calling label for now
    printf("Name = %s\n", snappy.name);
    printf("Species = %s\n", snappy.species);
    printf("Teeth = %i\n", snappy.teeth);
    printf("Age = %i\n", snappy.age);

    return 0;
}
```

```
> make fish2 && ./fish2
gcc -c fish2.c
gcc fish2.o -o fish2
Snappy is a Piranha with 69 teeth. He is 4
Name = Snappy
Species = Piranha
Teeth = 69
Age = 4
```

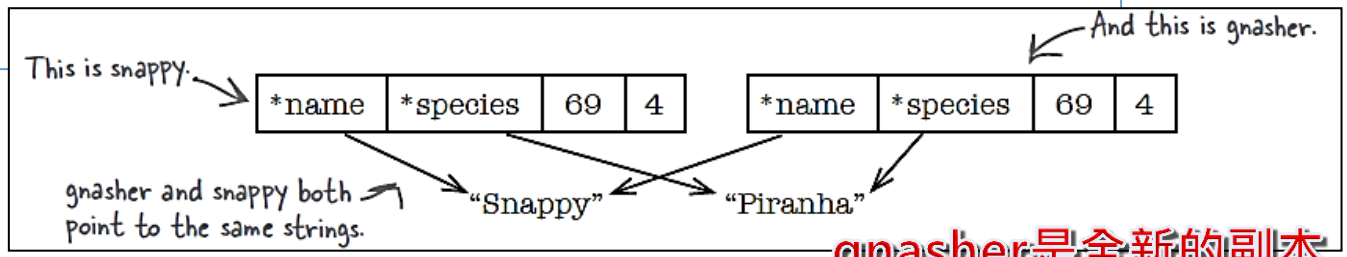
資料會被複製

```
struct fish {
    const char *name;
    const char *species;
    int teeth;
    int age;
};
```

當你指定struct變數時，就是在叫電腦複製資料

```
struct fish snappy = {"Snappy", "Piranha", 69, 4};
struct fish gnasher = snappy;
```

```
int main()
{
    struct fish snappy = {"Snappy", "Piranha", 69, 4};
    catalog(snappy);
    label(snappy);
    return 0;
}
```



gnasher是全新的副本

把struct放進另一個struct

這是魚喜歡的東西

```
struct preferences {
    const char *food;
    float exercise_hours;
};
```

```
struct fish {
    const char *name;
    const char *species;
    int teeth;
    int age;
    struct preferences care;
};
```

這欄位是一個struct，這稱作巢狀結構(nesting)

建立一隻名字為snappy的fish

```
struct fish snappy = {"Snappy", "Piranha", 69, 4, {"Meat", 7.5}};
```

```
printf("Snappy likes to eat %s", snappy.care.food);
printf("Snappy likes to exercise for %f hours", snappy.care.exercise_hours);
```

用typedef簡化

新技術：使用typedef賦予你的struct一個適當的名稱

```
typedef struct cell_phone {
    int cell_no;
    const char *wallpaper;
    float minutes_of_charge;
} phone;
```

phone是struct cell_phone的別名(alias)

宣告時，編譯器看到 phone = struct cell_phone

```
phone p = {5557879, "sinatra.png", 1.35};
```

```
typedef struct {
    int cell_no;
    const char *wallpaper;
    float minutes_of_charge;
} phone;

phone p = {5557879, "s.png", 1.35};
```

編譯器很樂意看到你略過struct的名稱

如何更新struct：烏龜生日

turtle.c

```c
#include <stdio.h>
typedef struct {
    const char *name;
    const char *species;
    int age;
} turtle;

void happy_birthday(turtle t)
{
    t.age = t.age + 1;
    printf("Happy Birthday %s! You are now %i years old!\n", t.name, t.age);
}

int main()
{
    turtle myrtle = {"Myrtle", "Leatherback sea turtle", 99};
    happy_birthday(myrtle);
    printf("%s's age is now %i\n", myrtle.name, myrtle.age);
    return 0;
}
```

clone，t是複製品

```
> gcc turtle.c -o turtle && ./turtle
Happy Birthday Myrtle! You are now 100 years old!
Myrtle's age is now 99
>
```

WTF??!

你需要指向struct的指標

```
void happy_birthday(turtle *t)
{
    ...
}
...
happy_birthday(&myrtle);
```

t是一個指向烏龜struct的pointer

所以呼叫時，要傳入的是myrtle的位址

```
void happy_birthday(turtle *t)
{
    (*t).age = (*t).age + 1;
    printf("Happy Birthday %s! You are now %i years old!\n", (*t).name, (*t).age);
}
```

(*t).age 改以 t->age 表示，可以增加可讀性

```
void happy_birthday(turtle *t)
{
    t->age = t-> age + 1;
    printf("Happy Birthday %s! You are now %i years old!\n", t->name, t->age);
}
```

(*t).age? *t.age? 使用struct時務必小心括號( )，它們真的有影響

同構(union)

- 記錄某種東西的「量」，那個量**可能**是個數、重量或體積。 **什麼阿?**

```
typedef struct {
    ...
    short count;
    float weight;
    float volume;
    ...
} fruit;
```

用struct浪費記憶體，而且可能會被設定超過一個值

```
typedef union {
    short count;
    float weight;
    float volume;
} quantity;
```

使用union，各欄位都存在相同的記憶體空間

union提供一種方式讓你建立「支援不同資料型別」的變數

指定初始化(也可用於struct)

```
quantity q = {.weight=1.5};
```

```
quantity q;
q.volume = 3.7;
```

Union經常搭配struct使用

```
typedef union {
    short count;
    float weight;
    float volume;
} quantity;
```

```
typedef struct {
    const char *name;
    const char *country;
    quantity amount;
} fruit_order;
```

會用到哪一個「量」？

蘋果使用了「weight」來定義他的「量」。
所以union裡面的count與volume都失效。
看到union，就想到要「多選一」。

```
fruit_order apples = {"apples", "England", .amount.weight=4.2};
printf("This order contains %2.2f lbs of %s\n", apples.amount.weight, apples.name);
```

不能知道union儲存的型別

注意不同型別

```
#include <stdio.h>
typedef union {
    float weight;
    int count;
} cupcake;

int main()
{
    cupcake order = {2};
    printf("Cupcakes quantity: %i\n", order.count);
    return 0;
}
```

```
> gcc bad_union.c -o bad_union && ./bad_union
Cupcakes quantity: 1073741824
```

太多蛋糕了吧!

沒有指定初始化，C語言會直接認為是設定union的第一個量。

你指定到weight，但讀的是count。count裡面是甚麼沒人知道喔~

你可以在union裡面儲存許多可能的值，然後一旦儲存之後，你沒有辦法知道它是甚麼型別!

需要一種可以記錄我們儲存在union裡的值：建立列舉(enum)

追蹤紀錄我們存在union的值

列舉(enum)儲存符號

- 列舉讓你建立一系列的符號

```
enum colors {RED, GREEN, PUCE};
enum colors favorite = PUCE;
enum colors favorite = PUSE;
```

編譯不過，PUSE沒有在清單中

那麼我們到底要如何利用enum來幫助我們追蹤紀錄union呢?

範例

enumtest.c

```c
#include <stdio.h>

typedef enum {
    COUNT, POUNDS, PINTS
} unit_of_measure;
```

```c
typedef union {
    short count;
    float weight;
    float volume;
} quantity;
```

```c
typedef struct {
    const char *name;
    const char *country;
    quantity amount;
    unit_of_measure units;
} fruit_order;
```

```c
void display(fruit_order order)
{
    printf("This order contains ");
    if (order.units == PINTS)
        printf("%2.2f pints of %s\n", order.amount.volume, order.name);
    else if (order.units== POUNDS)
        printf("%2.2f lbs of %s\n", order.amount.weight, order.name);
    else
        printf("%i %s\n", order.amount.count, order.name);
}
```

```c
int main()
{
    fruit_order apples = {"apples", "England", .amount.count=144, COUNT};
    fruit_order strawberries = {"strawberries", "Spain", .amount.weight =17.6, POUNDS};
    fruit_order oj = {"orange juice", "U.S.A.", .amount.volume=10.5, PINTS;
    display(apples);
    display(strawberries);
    display(oj);
    return 0;
}
```

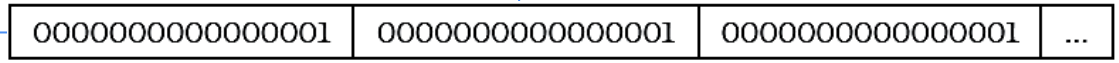
```
> gcc enumtest.c -o enumtest && ./enumtest
This order contains 144 apples
This order contains 17.60 lbs of strawberries
This order contains 10.50 pints of orange juice
```

Bitfield儲存自訂數目的字元

這些變數只想儲存True(1)或False(0)

```
typedef struct {
    short low_pass_vcf;
    short filter_coupler;
    short reverb;
    short sequential;
    ...
} synth;
```

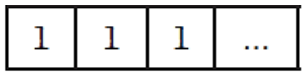
浪費記憶體空間



bitfield一定要宣告成unsigned int

```
typedef struct {
    unsigned int low_pass_vcf:1;
    unsigned int filter_coupler:1;
    unsigned int reverb:1;
    unsigned int sequential:1;
    ...
} synth;
```

如果你有8個單一位元的bitfield，電腦就可以用一個byte把它們儲存下來

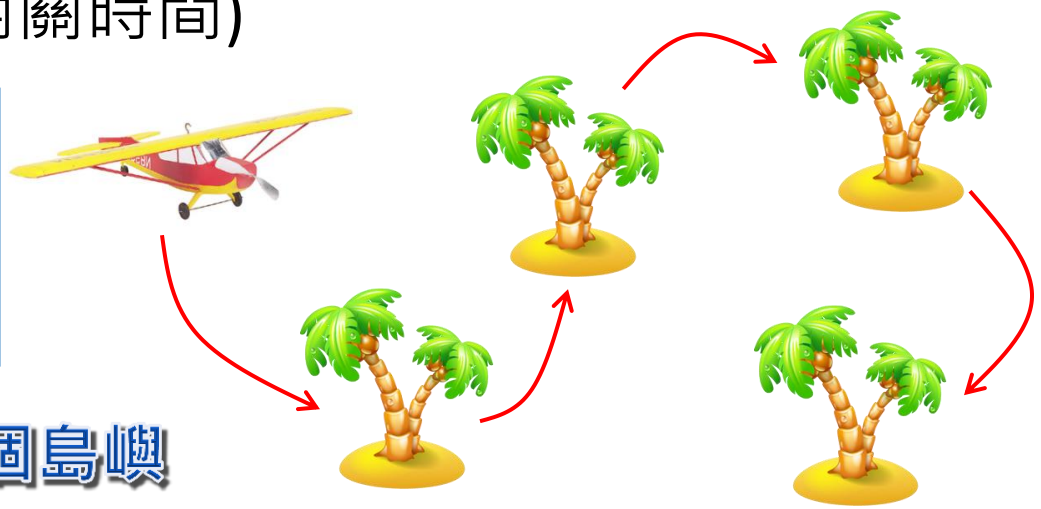


Ch7 建立橋接：資料結構和動態記憶體

資料結構

- 有時候單一struct並不夠用，我們可以用struct指標(struct pointer)將自訂資料型別連結成複雜龐大的資料結構。
- 島嶼旅遊(機場開關時間)

```
typedef struct {
    char *name;
    char *opens;
    char *closes;
} island;
```



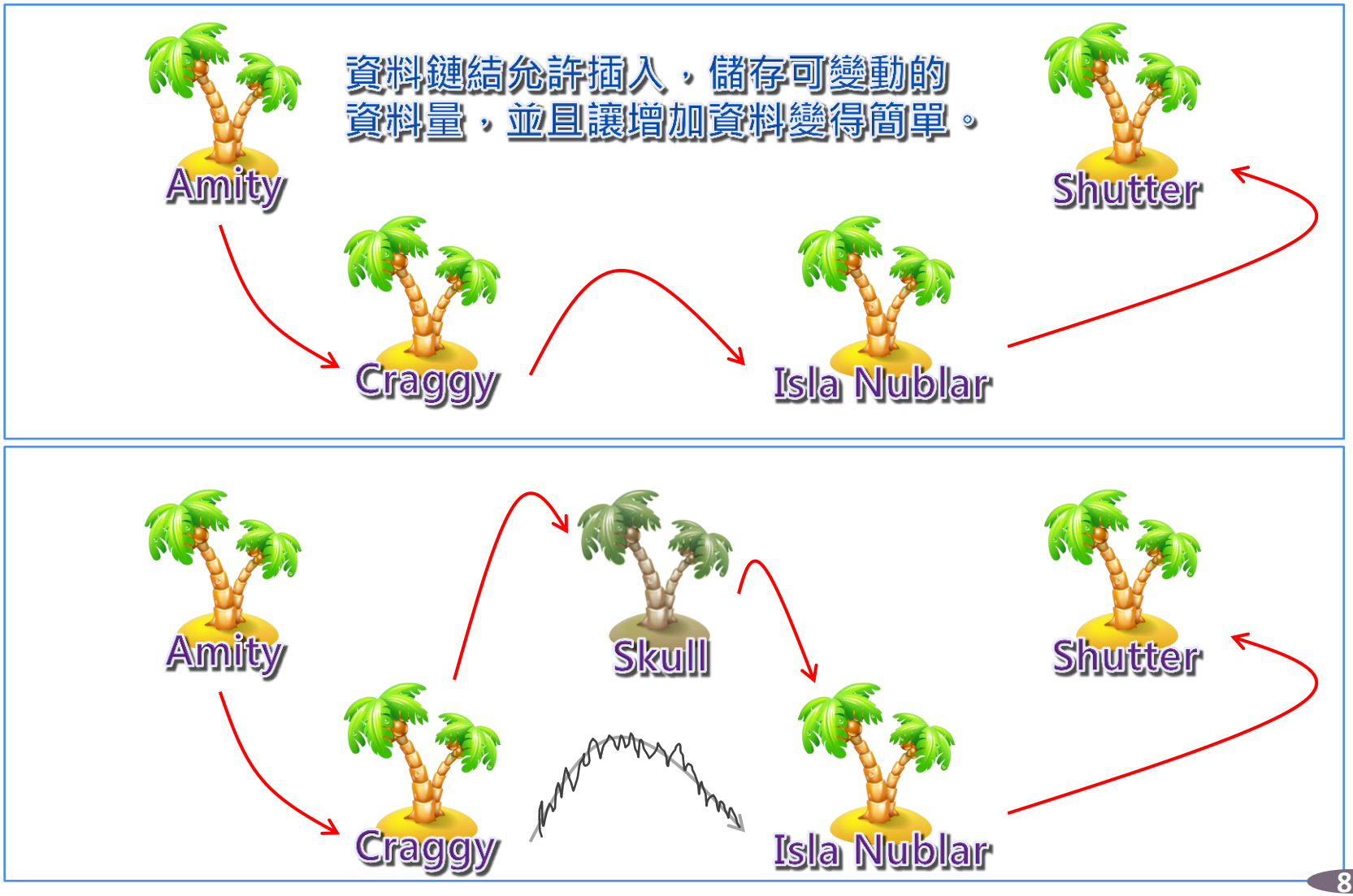
這個行程要跑4個島嶼

```
island tour[4];
```

陣列是固定長度(4個)，沒有彈性。如果中間突然要插入行程呢?

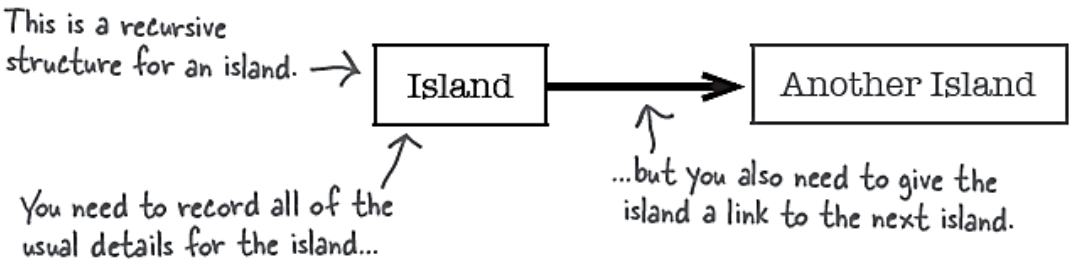
鏈結串列(Linked List) = 資料鏈

- 鏈結串列儲存資料片段及指向另一資料片段的link



建立遞迴結構

- 每一個struct都連結到下一個相同型的的struct，稱為遞迴結構(recursive structure)。



遞迴結構一定需要名稱


```
typedef struct island {
    char *name;
    char *opens;
    char *closes;
    struct island *next;
} island;
```

指向下一個島嶼的指標

用C語言來建造島嶼吧!

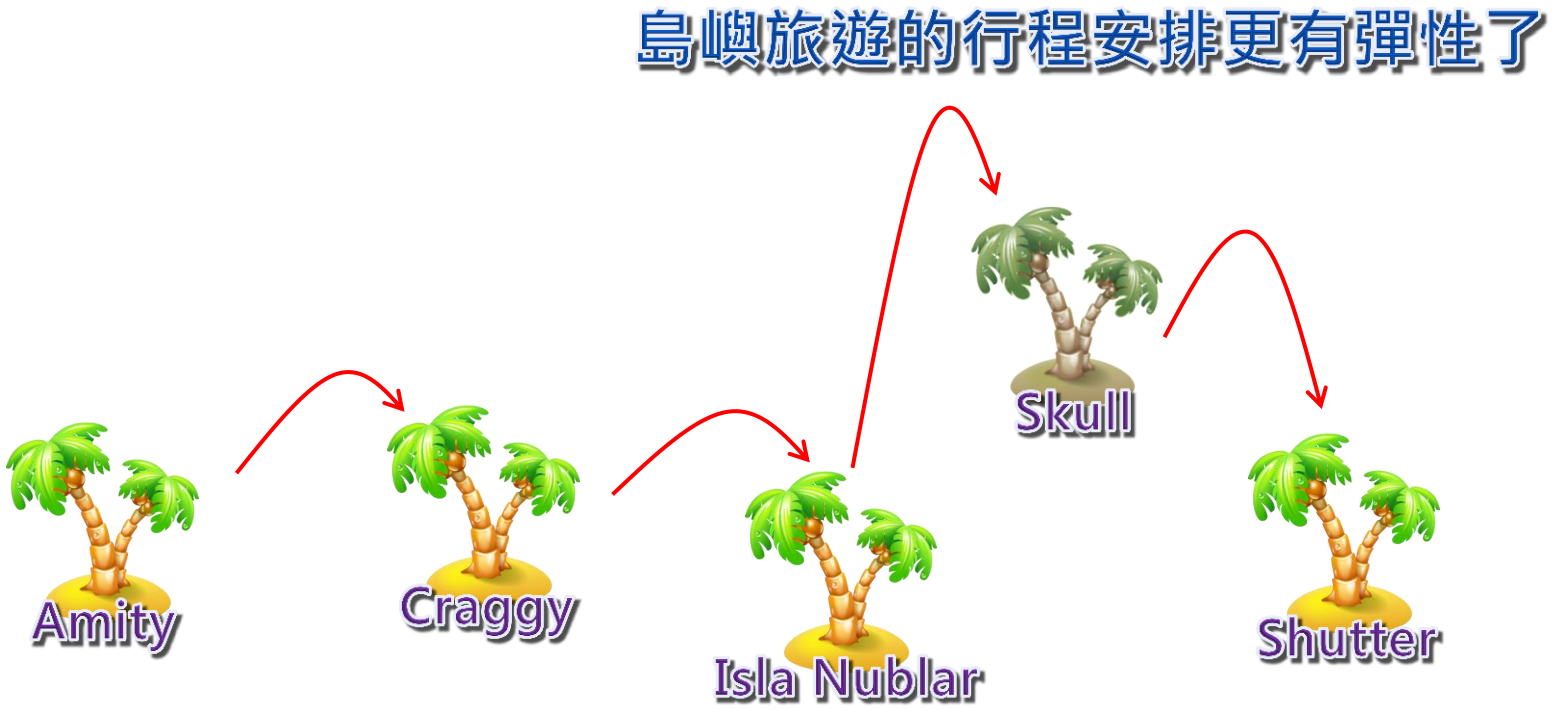
```
island amity = {"Amity", "09:00", "17:00", NULL};
island craggy = {"Craggy", "09:00", "17:00", NULL};
island isla_nublar = {"Isla Nublar", "09:00", "17:00", NULL};
island shutter = {"Shutter", "09:00", "17:00", NULL};
```


```
amity.next = &craggy;
craggy.next = &isla_nublar;
isla_nublar.next = &shutter;
```

用指向下一個島嶼的指標，
串連出快樂的島嶼行程。

插入行程

```
island skull = {"Skull", "09:00", "17:00", NULL};
isla_nublar.next = &skull;
skull.next = &shutter;
```



Tour程式

tour.c

```
void display(island *start)
{
    island *i = start;
    for(; i != NULL; i = i->next ){
        printf("Name: %s open: %s-%s\n",  i->name,  i->opens, i-> closes);
    }
}
```

```
int main()
{
    island amity = {"Amity", "09:00", "17:00", NULL};
    island craggy = {"Craggy", "09:00", "17:00", NULL};
    island isla_nublar = {"Isla Nublar", "09:00", "17:00", NULL};
    island shutter = {"Shutter", "09:00", "17:00", NULL};

    amity.next = &craggy;
    craggy.next = &isla_nublar;
    isla_nublar.next = &shutter;

    island skull = {"Skull", "09:00", "17:00", NULL};
    isla_nublar.next = &skull;
    skull.next = &shutter;
    display(&amity);
    return 0;
}
```

```
> gcc tour.c -o tour && ./tour
Name: Amity open: 09:00-17:00
Name: Craggy open: 09:00-17:00
Name: Isla Nublar open: 09:00-17:00
Name: Skull open: 09:00-17:00
Name: Shutter open: 09:00-17:00
```

在編譯時已知要儲存多少資料量(本地變數存在stack)，但往往有些時候我們不會事先知道要儲存資料的量到底有多少，這時候我們就需要程式有動態記憶體儲存(在heap)。

使用heap做動態儲存

- 目前為止，我們一直在使用的大多數記憶體都是stack(堆疊)，stack是用來儲存本地變數的記憶體區域，每個資料片段皆被儲存在變數裡，並且在你離開函式時馬上消失。
- 麻煩的是，執行期間你很難從stack取得更多儲存空間，而這正是heap(堆積)可以幫得上忙的地方。
- 在heap中，程式可以將資料儲存得更久，他不會自動被清除，那表示他最適合用來儲存像我們的linked list那樣的資料結構。

heap儲存就像是在儲物櫃裡存放貴重物品。你可以用malloc()函式向系統申請一把開啟儲物櫃的鑰匙。

malloc()與free()取得與釋放記憶體

- 程式突然有大量資料在執行期間必須被儲存，我們可以使用malloc()告訴作業系統要騰出多少記憶體出來給我們。malloc()會傳出一個指向那個記憶體空間的pointer，它讓你存取該記憶體並且追蹤已經被配置好的儲物櫃。
- 使用stack時，你不需擔心歸還記憶體，一切都是自動發生，當你離開函式，本地儲存就會從stack釋放。
- Heap不一樣，一旦你要求了一塊heap空間，該空間就不會再挪作他用，直到你告訴系統你已經使用完畢，我們可以呼叫free()來完成這件事情。假使你沒有釋放記憶體，很容易發生memory leak的問題。
- Heap只有固定數量的儲存空間，因此務必要明智的使用他。

malloc()與free()

- 記憶體配置(Memory allocation)

```
#include <stdlib.h>
   ...
malloc(sizeof(island));
```

malloc()與free()在stdlib.h

告訴系統：「給我一塊足以儲存island struct的空間」

將要到的記憶體起始位址指標存放在p變數

```
island *p = malloc(sizeof(island));
```

```
free(p);
```

malloc()實際回傳的是一種通用的指標，具有型別void*

記住：如果你在程式裡的某部分使用malloc()配置記憶體，稍後就一定得在另一個地方用free()釋放它。總之，malloc()和free()要成雙成對就是了。

消失的島嶼之謎

test_flight.c

島嶼名稱以char指標傳送

```
island* create(char *name)
{
    island *i = malloc(sizeof(island));
    i->name = name;
    i->opens = "09:00";
    i->closes = "17:00";
    i->next = NULL;
    return i;
}
```

使用malloc()在heap建立空間，也就是在heap建立island struct

設定欄位

回傳新struct的位址

```
    char name[80];
    fgets(name, 80, stdin);
    island *p_island0 = create(name);
    fgets(name, 80, stdin);
    island *p_island1 = create(name);
    p_island0->next = p_island1;
    display(p_island0);
```

```
> ./test_flight
Atlantis
Titchmarsh Island

Name: Titchmarsh Island
open: 09:00-17:00
Name: Titchmarsh Island
open: 09:00-17:00
```

WTF?!

第一個島嶼消失了!因為它們共用了本地char陣列來儲存島嶼名稱。

使用strdup()修正程式

在string.h

strdup()會將字元陣列複製到heap (不是在stack)

```
island* create(char *name)
{
   island *i = malloc(sizeof(island));
   i->name = strdup(name);
   i->opens = "09:00";
   i->closes = "17:00";
   i->next = NULL;
   return i;
}
```

```
char *s = "MONA LISA";
char *copy = strdup(s);
```

```
> ./test_flight
Atlantis
Titchmarsh Island

Name: Atlantis
open: 09:00-17:00
Name: Titchmarsh Island
open: 09:00-17:00
```

有沒有覺得哪裡怪怪的？

是的，我們有malloc()但是卻沒有free()釋放記憶體。

建立島嶼的程式

test_flight2.c

```c
int main()
{
    island *start = NULL;
    island *i = NULL;
    island *next = NULL;
    char name[80];

    for(; fgets(name, 80, stdin) != NULL; i = next) {
      next = create(name);
        if (start == NULL)
          start = next;
        if (i != NULL)
          i->next = next;
    }
    display(start);
    release(start);

    return 0;
}
```

```c
island* create(char *name)
{
    island *i = malloc(sizeof(island));
    i->name = strdup(name);
    i->opens = "09:00";
    i->closes = "17:00";
    i->next = NULL;
    return i;
}
```

```c
void release(island *start)
{
    island *i = start;
    island *next = NULL;
    for(; i != NULL; i = next){
          next = i->next;
          free(i->name);
          free(i);
    }
}
```

Ch8 函式進階

函式進階 – 發揮函式的強大威力

find.c

- 透過把函式當成參數來傳遞而提高程式碼的智商

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

int NUM_ADS = 7;
char *ADS[] = {
    "William: SBM GSOH likes sports, TV, dining",
    "Matt: SWM NS likes art, movies, theater",
    "Luis: SLM ND likes books, theater, art",
    "Mike: DWM DS likes trucks, sports and bieber",
    "Peter: SAM likes chess, working out and art",
    "Josh: SJM likes sports, movies and theater",
    "Jed: DBM likes theater, books and dining"
};
```

```
> gcc find.c -o find && ./find
Search results:
-----------------------------------
William: SBM GSOH likes sports, TV, dining
Josh: SJM likes sports, movies and theater
-----------------------------------
>
```

過濾程式：喜歡運動但是對小賈斯汀沒興趣者

```
int main()
{
    find();
    return 0;
}
```

```
void find()
{
    int i;
    puts("Search results:");
    puts("-----------------------------------");
    for (i = 0; i<NUM_ADS; i++) {
      if (strstr( ADS[i], "sports") && !strstr(ADS[i], "bieber")) {
          printf("%s\n", ADS[i]);
      }
    }
    puts("-----------------------------------");
}
```

你可以拷貝程式碼，修改一下就可以進行任何的搜尋。但這樣做會很沒效率(重複程式碼)。

告訴find()關於函式的名稱

```
int sports_no_bieber(char *s)
{
    return strstr(s, "sports") && !strstr(s, "bieber");
}
```

有某種方式可以把函式名稱當成參數傳遞給find()，你就能注入該測試

如何指明某個參數儲存函式名稱？如果你有函式名稱，要如何使用它來呼叫該函式？

```
void find( int (*match)(char*) )
{
    int i;
    puts("Search results:");
    puts("-----------------------------------");
    for (i = 0; i < NUM_ADS; i++) {
        if (match(ADS[i])) {
            printf("%s\n", ADS[i]);
        }
    }
    puts("-----------------------------------");
}
```

```
find(sports_no_bieber);
find(sports_or_workout);
find(ns_theater);
find(arts_theater_or_dining);
```

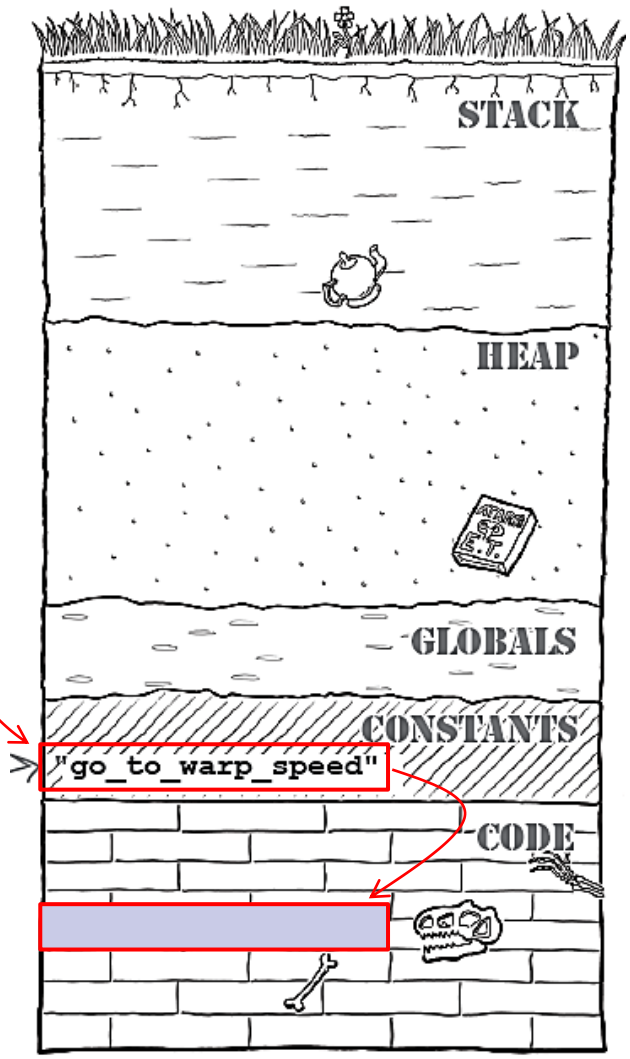
函式名稱 = 指向該函式的指標

```
int go_to_warp_speed(int speed)
{
    dilithium_crystals(ENGAGE);
    warp = speed;
    reactor_core(c, 125000 * speed, PI);
    clutch(ENGAGE);
    brake(DISENGAGE);
    return 0;
}
```

每當你建立函式時，也同時建立了相同名稱的函式指標。

```
go_to_warp_speed(4);
```

當你呼叫該函式時，你正在使用該函式的指標。



沒有函式這種資料型別

`int *a;` V.S. `function *f;` 沒有function這種型別

```
int go_to_warp_speed(int speed)
{
  ...
}
```

因為函式可能傳回任何型別的結果

那要如何建造函式指標呢？

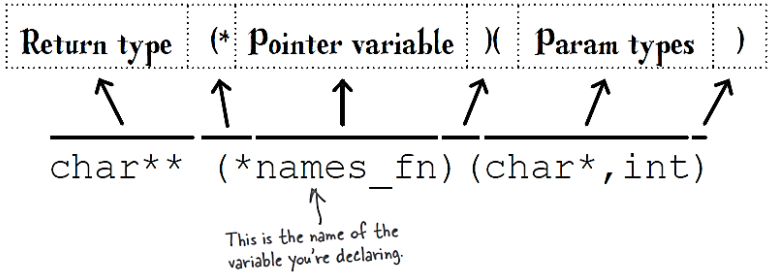
```
int (*warp_fn)(int);
warp_fn = go_to_warp_speed;
warp_fn(4);
```

這會建立名為wrap_fn的變數，它可以儲存go_to_wrap_speed()函式的位址。

指向字串陣列

```
char** (*names_fn)(char*,int);
names_fn = album_names;
char** results = names_fn("Sacha Distel", 1972);
```

你需要告訴C該函式的回傳型別與接受參數。一旦你宣告了函式指標變數，你就能夠像任何變數一樣地使用它。你可以指定值給它，把它增加到陣列，並且還能把它傳遞給函式。



範例：不同功能的搜尋函式

```c
int sports_no_bieber(char *s)
{
    return strstr(s, "sports") && !strstr(s, "bieber");
}
int sports_or_workout(char *s)
{
    return strstr(s, "sports")|| strstr(s, "working out");
}
int ns_theater(char *s)
{
    return strstr(s, "NS") && strstr(s, "theater");
}
int arts_theater_or_dining(char *s)
{
    return strstr(s, "arts") || strstr(s, "theater") || strstr(s, "dining");
}
```

```c
void find( int (*match)(char*) )
{
    int i;
    puts("Search results:");
    puts("-----------------------------------");
    for (i = 0; i < NUM_ADS; i++) {
        if (match(ADS[i])) {
            printf("%s\n", ADS[i]);
        }
    }
    puts("-----------------------------------");
}
```

```c
int main()
{
    find(sports_no_bieber);
    find(sports_or_workout);
    find(ns_theater);
    find(arts_theater_or_dining);
    return 0;
}
```

```
> ./find
Search results:
-----------------------------------
William: SBM GSOH likes sports, TV, dining
Josh: SJM likes sports, movies and theater
-----------------------------------
Search results:
-----------------------------------
William: SBM GSOH likes sports, TV, dining
Mike: DWM DS likes trucks, sports and bieber
Peter: SAM likes chess, working out and art
Josh: SJM likes sports, movies and theater
-----------------------------------
Search results:
-----------------------------------
Matt: SWM NS likes art, movies, theater
-----------------------------------
Search results:
-----------------------------------
William: SBM GSOH likes sports, TV, dining
Matt: SWM NS likes art, movies, theater
Luis: SLM ND likes books, theater, art
Josh: SJM likes sports, movies and theater
Jed: DBM likes theater, books and dining
-----------------------------------
>
```

排序程式qsort()

```
qsort(void *array, size_t length, size_t item_size, int (*compar)(const void *, const void *));
```

指向陣列的指標　陣列的長度　陣列元素的尺寸　指向比較陣列中兩個項目的函式

傳回正數：(值1 > 值2)　傳回負數：(值1 < 值2)　傳回0：值1 = 值2

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

typedef struct {
    int width;
    int height;
} rectangle;
```

```
int compare_scores(const void* score_a, const void* score_b)
{
    int a = *(int*)score_a;
    int b = *(int*)score_b;
    return a - b;
}
```

```
int compare_scores_desc(const void* score_a, const void* score_b)
{
    int a = *(int*)score_a;
    int b = *(int*)score_b;
    return b - a;
}
```

```
int compare_areas(const void* a, const void* b)
{
    rectangle* ra = (rectangle*)a;
    rectangle* rb = (rectangle*)b;
    int area_a = (ra->width * ra->height);
    int area_b = (rb->width * rb->height);
    return area_a - area_b;
}
```

```c
int compare_names(const void* a, const void* b)
{
    char** sa = (char**)a;
    char** sb = (char**)b;
    return strcmp(*sa, *sb);
}
```

```c
int compare_areas_desc(const void* a, const void* b)
{
    return compare_areas(b, a);
}
```

```c
int compare_names_desc(const void* a, const void* b)
{
    return compare_names(b, a);
}
```

```c
int main()
{
    int scores[] = {543,323,32,554,11,3,112};
    int i;
    qsort(scores, 7, sizeof(int), compare_scores_desc);
    puts("These are the scores in order:");
    for (i = 0; i < 7; i++) {
          printf("Score = %i\n", scores[i]);
    }

    char *names[] = {"Karen", "Mark", "Brett", "Molly"};
    qsort(names, 4, sizeof(char*), compare_names);
    puts("These are the names in order:");
    for (i = 0; i < 4; i++) {
          printf("%s\n", names[i]);
    }
    return 0;
}
```

Dear John Letter (I)

send_dear_johns.c

```
enum response_type {DUMP, SECOND_CHANCE, MARRIAGE};
typedef struct {
    char *name;
    enum response_type type;
} response;
```

三種不同的訊息類型

紀錄回信類型，搭配每一段回應資料。

```
void dump(response r)
{
    printf("Dear %s,\n", r.name);
    puts("Unfortunately your last date contacted us to");
    puts("say that they will not be seeing you again");
}
```

打槍

```
void second_chance(response r)
{
    printf("Dear %s,\n", r.name);
    puts("Good news: your last date has asked us to");
    puts("arrange another meeting. Please call ASAP.");
}
```

還有機會

```
void marriage(response r)
{
    printf("Dear %s,\n", r.name);
    puts("Congratulations! Your last date has contacted");
    puts("us with a proposal of marriage.");
}
```

可以結緣

Dear John Letter (II)

```
int main()
{
    response r[] = {
        {"Mike", DUMP}, {"Luis", SECOND_CHANCE},
        {"Matt", SECOND_CHANCE}, {"William", MARRIAGE}
    };

    int i;
    for(i = 0; i < 4; i++){
        switch(r[i].type){
        case DUMP:
          dump(r[i]);
          break;
        case SECOND_CHANCE:
          second_chance(r[i]);
          break;
        default:
          marriage(r[i]);
        }
    }
    return 0;
}
```

繞行整個陣列

每次都檢查type

對應的回信

```
./send_dear_johns
Dear Mike,
Unfortunately your last date contacted us to
say that they will not be seeing you again
Dear Luis,
Good news: your last date has asked us to
arrange another meeting. Please call ASAP.
Dear Matt,
Good news: your last date has asked us to
arrange another meeting. Please call ASAP.
Dear William,
Congratulations! Your last date has contacted
us with a proposal of marriage.
>
```

如果要增加第四種回信類型?

建造函式指標陣列

```
replies[] = {dump, second_chance, marriage};
```

用一個陣列來儲存一群函式名稱，但這樣子在C語言中是行不通的。

你必須確切告訴編譯器即將儲存在該陣列中的函式是甚麼樣子。

```
void (*replies[])(response) = {dump, second_chance, marriage};
```

跟列舉的排列的順序要一樣

```
enum response_type {DUMP, SECOND_CHANCE, MARRIAGE};
```

0 1 2

所以

```
replies[DUMP] == dump
replies[SECOND_CHANCE] == second_chance
replies[MARRIAGE] == marriage
```

Dear John Letter更聰明的寫法

```
int main()
{
    void (*replies[])(response) = {dump, second_chance, marriage};
    response r[] = {
          {"Mike", DUMP}, {"Luis", SECOND_CHANCE},
          {"Matt", SECOND_CHANCE}, {"William", MARRIAGE}
    };
    int i;
    for(i = 0; i < 4; i++){
          (replies[r[i].type])(r[i]);
    }
    return 0;
}
```

你只需使用這一行，就取代了整個switch陳述式。

```
> ./dear_johns
Dear Mike,
Unfortunately your last date contacted us to
say that they will not be seeing you again
Dear Luis,
Good news: your last date has asked us to
arrange another meeting. Please call ASAP.
Dear Matt,
Good news: your last date has asked us to
arrange another meeting. Please call ASAP.
Dear William,
Congratulations! Your last date has contacted
us with a proposal of marriage.
>
```

增加第四種回信類型更方便，函式指標陣列讓你的程式碼更容易管理、精簡、更易擴充!

```
enum response_type {DUMP, SECOND_CHANCE, MARRIAGE, LAW_SUIT};
void (*replies[])(response) = {dump, second_chance, marriage, law_suit};
```

Ch9 靜態與動態程式庫
可熱切換的程式碼

靜態與動態函式庫

- 能夠用在銀行的程式碼：安全防護程式庫

```
#include "encrypt.h"
void encrypt(char *message)
{
    while (*message) {
        *message = *message ^ 31;
        message++;
    }
}
```

encrypt.c

```
void encrypt(char *message);
```

加密程式

encrypt.h

```
#include "checksum.h"
int checksum(char *message)
{
    int c = 0;
    while (*message) {
        c += c ^ (int)(*message);
    message++;
    }
    return c;
}
```

checksum.c

```
int checksum(char *message);
```

錯誤檢查程式

checksum.h

安全防護程式

test_code.c

< > 是給標準檔頭用的，放在 /usr/local/include /usr/include

" "給自訂的本地檔頭用的

```
#include <stdio.h>
#include "encrypt.h"
#include "checksum.h"
int main()
{
    char s[] = "Speak friend and enter";
    encrypt(s);
    printf("Encrypted to '%s'\n", s);
    printf("Checksum is %i\n", checksum(s));
    encrypt(s);
    printf("Decrypted back to '%s'\n", s);
    printf("Checksum is %i\n", checksum(s));
    return 0;
}
```

```
> ./test_code
Encrypted to "Loz~t?ymvzq{?~q{?zqkzm"
Checksum is 89561741
Decrypted back to "Speak friend and enter"
Checksum is 89548156
>
```

如何共用程式碼

- **兩種檔案是你想在各程式間共用的：**

  .h標頭檔與.o目的檔

- **共用.h檔**

(1)把它放在標準目錄: `#include <encrypt.h>`

(2) include時放進完整路徑: `#include "/my_header_files/encrypt.h"`

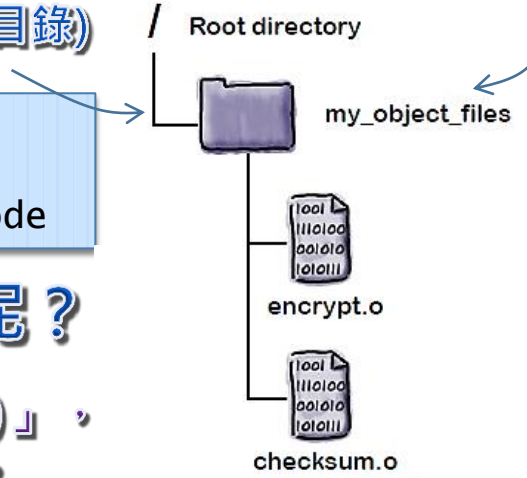
(3) 編譯時告訴編譯器去哪裡找他們:

```
gcc -I/my_header_files test_code.c ... -o test_code
```

- **使用完整路徑共用.o檔** (建一個.o檔的共用目錄)

```
gcc -I/my_header_files test_code.c
      /my_object_files/encrypt.o
      /my_object_files/checksum.o -o test_code
```

目的檔的中央儲存機制

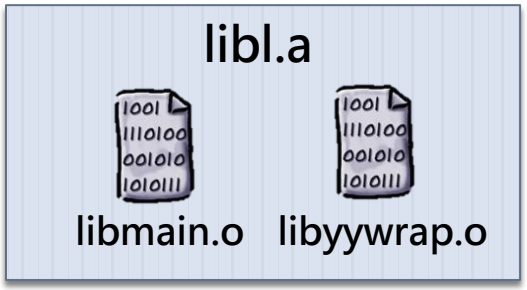


如果你有一拖拉庫的目的檔要共用呢？

你可以建立「由目的檔組成的收藏檔(Archive)」，
你一次就能夠告訴編譯器有一整組目的檔要用。

收藏檔(Archive)包含.o目的檔

- /usr/lib 裡有一堆.a檔



用指令nm可列出.a檔裡的檔案名稱

```
> nm libl.a
libl.a(libmain.o):
00000000000003a8     s EH_frame0
                     U _exit
0000000000000000     T _main
00000000000003c0     S _main.eh
                     U _yylex
libl.a(libyywrap.o):
0000000000000350     s EH_frame0
0000000000000000     T _yywrap
0000000000000368     S _yywrap.eh
>
```

T _main表示libmain.o包含main()函式

用ar命令建立收藏檔：打包多個.o檔

```
ar -rcs libhfsecurity.a encrypt.o checksum.o
```

建好.a檔可丟入/usr/lib或自己的/my_lib目錄下

r：若.a檔已存在則更新

c：不須顯示建立訊息

s：在.a檔開處建立索引

檔名一律要用lib作開頭，表示他們是靜態程式庫(static library)

最後，編譯你的其他程式

原始檔

```
gcc test_code.c -lhfsecurity -o test_code
```

若需要數個.a檔，可用多個 –l 選項

叫compiler去尋找libhfsecurity.a

```
gcc test_code.c -L/my_lib -lhfsecurity -o test_code
```

若.a檔放在其他目錄，可用 –L 指定特定目錄

撰寫makefile

```
#include <stdio.h>
#include <encrypt.h>
#include <checksum.h>
```

原始檔中用 < > include 進 encrypt.h 與 checksum.h。若它們沒有在系統目錄中的話，你必須用 -I 選項告訴編譯器它們位在哪裡。

```
encrypt.o: encrypt.c
        gcc -c encrypt.c -o encrypt.o
checksum.o: checksum.c
        gcc -c checksum.c -o checksum.o
libhfsecurity.a: encrypt.o checksum.o
        ar -rcs libhfsecurity.a encrypt.o checksum.o
bank_vault: bank_vault.c libhfsecurity.a
        gcc bank_vault.c -I . -L . -lhfsecurity -o bank_vault
```

. 表示「當前目錄」

可熱交換的程式碼?

- NTUT健身房正邁向全球化：計算卡路里程式

hfcal.h只包含display_calories()的宣告

```
#include <stdio.h>
#include <hfcal.h>
void display_calories(float weight, float distance, float coeff)
{
    printf("Weight: %3.2f lbs\n", weight);
    printf("Distance: %3.2f miles\n", distance);
    printf("Calories burned: %4.2f cal\n", coeff * weight *
distance);
}
```

hfcal.c

測試程式

```
#include <stdio.h>
#include <hfcal.h>
int main()
{
    display_calories(115.2, 11.3, 0.79);
    return 0;
}
```

elliptical.c

```
Weight: 115.20 lbs
Distance: 11.30 miles
Calories burned: 1028.39
cal
```

事情若更複雜：如果健身器材不只有elliptical(橢圓機)，還有treadmill(跑步機)和exercise bike(飛輪車)；如果器材要賣到使用公制單位kg與km的國家呢?甚至還有不同語言呢!

先看一下建造執行程式的步驟

(1) 建造目的檔 (假設hfcal.h放在 ./icludes目錄下)

```
gcc -I./includes -c hfcal.c -o hfcal.o
```

(2) 建造測試程式的目的檔

```
gcc -I./includes -c elliptical.c -o elliptical.o
```

(3) 建造收藏檔 (假設放到 ./lib目錄下)

```
ar -rcs ./libs/libhfcal.a hfcal.o
```

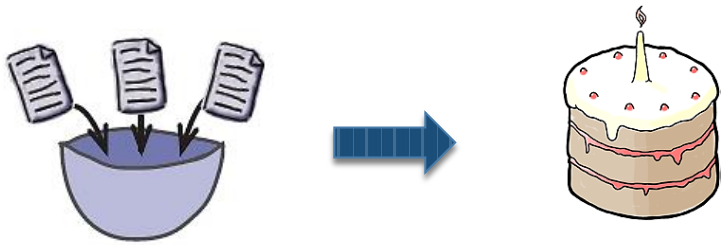
(4) 建立elliptical執行檔

```
gcc elliptical.o -L./libs -lhfcal -o elliptical
```

```
>./elliptical
Weight: 115.20 lbs
Distance: 11.30 miles
Calories burned: 1028.39 cal
>
```

換成kg和km表示

- 靜態連結

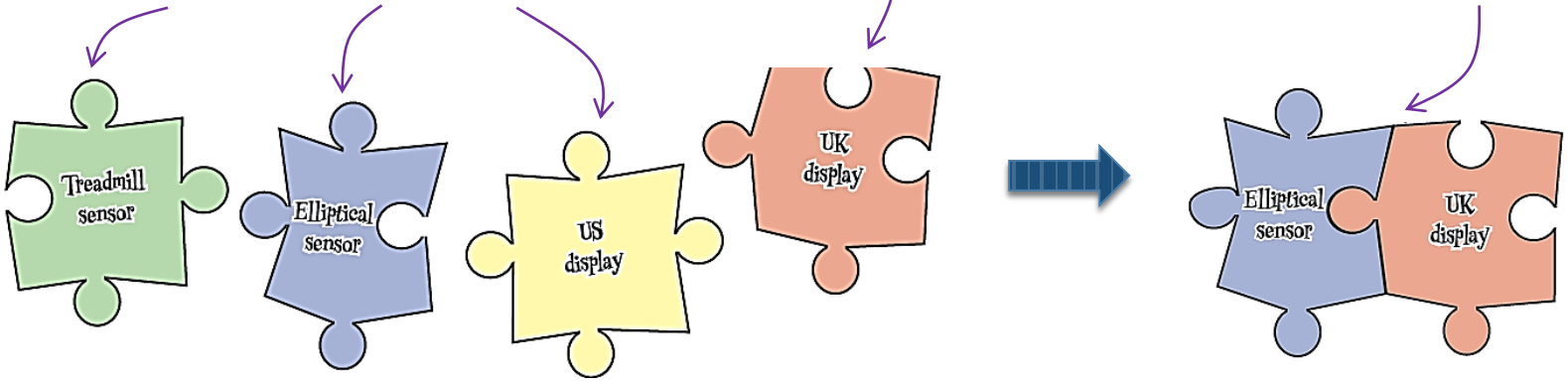


目的檔一旦連結成執行檔，就不能夠改變它們(修改程式後，只能重新再連結)

- 動態連結發生在執行期間

每個程式碼片段存在獨立的片段

在執行期間把相關的檔案結合起來



找到方法將目的碼片段存放在獨立的檔案，並且僅於程式執行時，再把他們動態連結再一起。

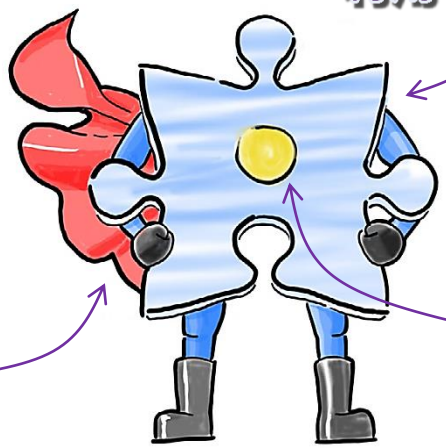
可以在執行期間連結.o或.a檔嗎?

- 事情沒有那麼簡單，因為.o檔與.a檔中並沒有足夠的資訊讓他們可以在執行期間被連結在一起。
- 事實上，我們的動態程式庫檔案還需要其他資訊，像是它們必須連結的其他檔案的名稱。

動態程式庫是服用禁藥的超級目的檔

帶有中介資料(metadata)的可重新安置(relocatable)目的檔

動態程式庫包含額外資訊，OS需要利用它將程式庫連結到其他東西。



程式庫從一個或多個.o檔被建造

動態程式庫的核心是單一目的碼片段

建造動態程式庫

(1) 首先，建造「與位址無關」的目的檔

```
gcc -I./includes -fPIC -c hfcal.c -o hfcal.o
```

hfcal.h位在./includes

-fPIC 告訴編譯器建立「與位址無關」的程式碼

-c表示「別連結程式碼」

(2) 建造動態連結檔

-share告訴編譯器要將 .o檔轉換為動態程式庫

```
gcc -shared hfcal.o -o  C:\libs\hfcal.dll
                        /libs/libhfcal.dll.a
                        /libs/libhfcal.so
                        /libs/libhfcal.dylib
```

MinGW

Cygwin

Linux或Unix

Mac

Windows: 副檔名為 .dll，稱為動態連結程式庫
Linux (Unix): 副檔名為 .so，稱為共用目的檔
Mac: 副檔名為 .dylib，稱為動態程式庫

都是一樣的東西

編譯並執行elliptical程式

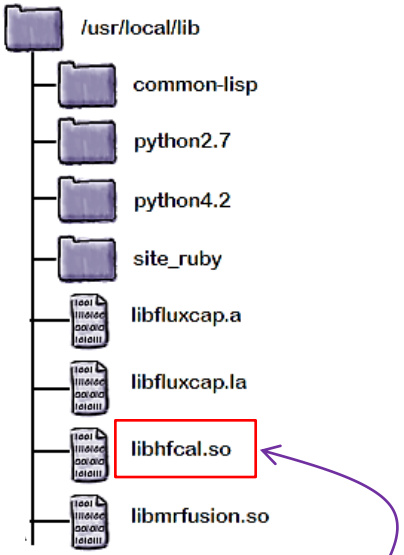
```
gcc -I./includes -c elliptical.c -o elliptical.o
gcc elliptical.o -L./libs -lhfcal -o elliptical
```

跟靜態編譯的命令相同，但是因為hfcal為動態程式庫，因此編譯的運作方式其實是不同的。編譯器不會將該程式庫程式碼包含到執行檔裡，相反地，編譯器將插入一些佔位器程式碼(placeholder code)，那會追蹤該程式庫，並且在執行期間連結它。

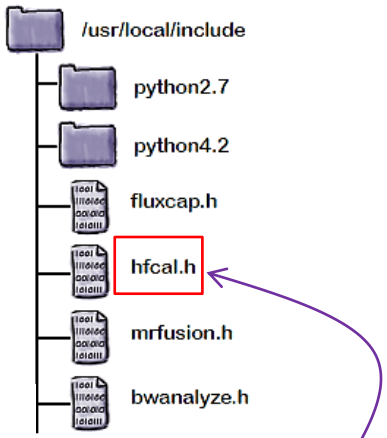
```
> PATH="$PATH:./libs"
> ./elliptical
Weight: 115.20 lbs
Distance: 11.30 miles
Calories burned: 1028.39 cal
>
```

確保電腦能夠找到該動態程式庫(在windows裡，開發者通常將DLL保存在與執行檔相同的目錄裡；在Linux與Mac則通常放在/usr/lib或/usr/local/lib之類的目錄裡。

美國版(mile)換成英國版(km)



hfcal程式庫安裝在這



hfcal標頭檔在這

```
Weight: 53.25 kg
Distance: 15.13 km
Calories burned: 750.42 cal
```

換成新的libhfcal.so

不需要 -I，因為標頭檔在標準目錄裡。

```
gcc -c -fPIC hfcal_UK.c -o hfcal.o
gcc -shared hfcal.o -o /usr/local/lib/libhfcal.so
```

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

void display_calories(float weight, float distance, float coeff)
{
    printf("Weight: %3.2f kg\n", weight / 2.2046);
    printf("Distance: %3.2f km\n", distance * 1.609344);
    printf("Calories burned: %4.2f cal\n", coeff * weight * distance);
}
```

hfcal_UK.c

Ch10 行程與系統呼叫

系統呼叫(System Call) – 打破疆界

- 在電腦中，C程式幾乎一切都仰賴作業系統，如果他們想要跟硬體溝通，就必須透過系統呼叫。
- 系統呼叫就是存在於作業系統核心(kernel)裡的函式，C標準程式庫裡的大多數函式都依靠它們而運作。

指令日誌：guard_log.c

```
> ./guard_log
ls
> ./guard_log
gcc --version
```

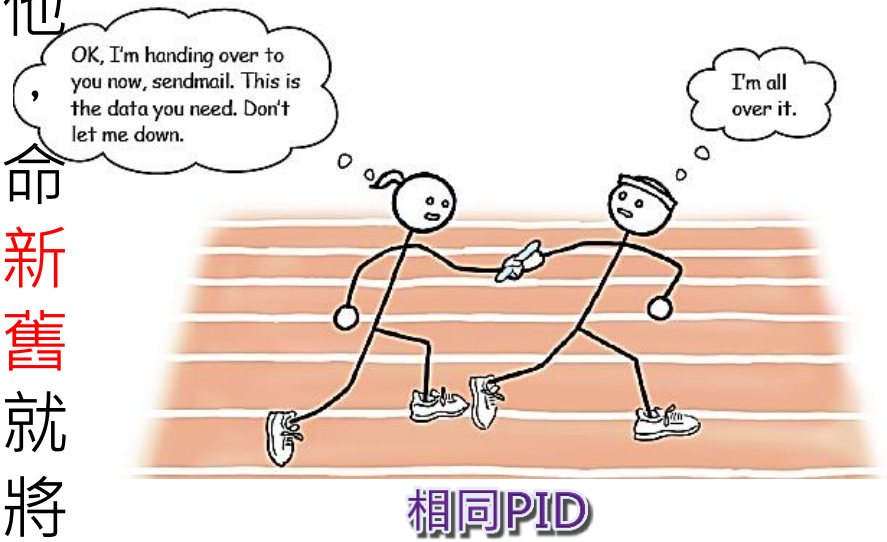
```
1  ls
2   Mon Aug  5 21:30:16 2013
3
4  gcc --version
5   Mon Aug  5 21:30:23 2013
6
```

reports.log

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
char* now()
{
    time_t t;
    time (&t);
    return asctime(localtime (&t));
}
/* Master Control Program utility.
Records guard patrol check-ins. */
int main()
{
    char comment[80];
    char cmd[120];
    fgets(comment, 80, stdin);
    sprintf(cmd, "echo '%s %s' >> reports.log", comment, now());
    system(cmd);

    return 0;
}
```

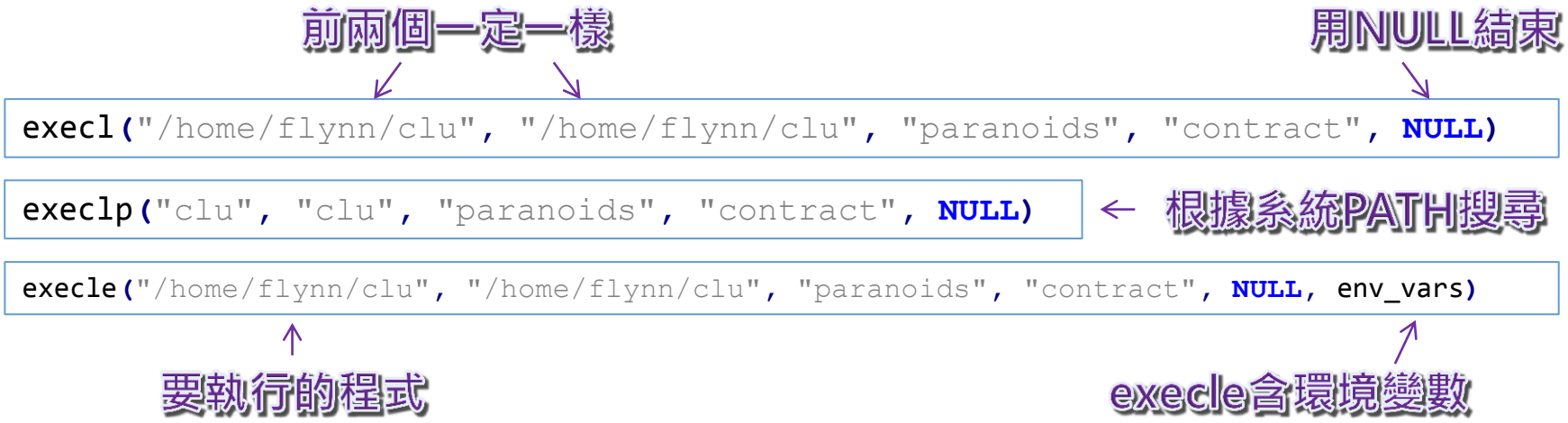
執行檔呼叫：exec()

- 使用exec()確切告訴作業系統要執行哪個程式
- 行程(process)即是在記憶體裡執行的程式
- 在windows下鍵入taskmgr (其他系統 ps –ef)，你就可以看到目前正在系統中執行的行程，作業系統使用行程識別符(PID)來追蹤每個行程。
- exec()透過執行某個其他程式來替換當前的行程，你可以指明要使用哪些命令引數或環境變數。當新程式啟動時，他擁有與舊程式完全相同的PID，就像接力賽跑，你的程式將他的行程交給新程式。

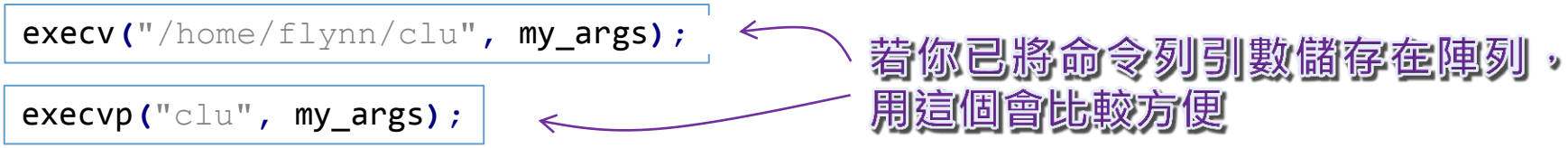


串列型與陣列型的exec()

- 串列型函式：execl()、execlp()、execle()



- 陣列型函式：execv()、execvp()、execve()



建立新行程並傳遞環境變數(I)

- 每個行程有一組環境變數，就是你在命令列輸入set或env會看到的那些值。
- C程式能夠利用stdlib.h中的getenv()系統呼叫來讀取環境變數。

diner_info.c

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("Diners: %s\n", argv[1]);
    printf("Juice: %s\n", getenv("JUICE"));
    return 0;
}
```

若你想用命令列引數和環境變數執行程式，可以這樣做

```c
char *my_env[] = {"JUICE=peach and apple", "PATH=/usr/bin", NULL};
execle("dinner_info", "dinner_info", "4", NULL, my_env);
```

execle()會設定引數和環境變數，然後以dinner_info替換當前的行程

注意：如果你在Cygwin上傳遞環境變數，務必包含PATH變數(PATH=/usr/bin)。

建立新行程並傳遞環境變數(II)

my_exec.c

```c
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>
int main()
{
    char *my_env[] = {"JUICE=peach and apple", "PATH=/usr/bin", NULL};
    execle("diner_info", "diner_info", "4", NULL, my_env);
    puts("Dude - the dinner info_code must be busted");
    puts(strerror(errno));
    return 0;
}
```

```
> ./my_exec
Dude - the dinner info_code must be busted
No such file or directory
```

上面程式有點小問題喔~

```
> ./my_exec
Diners: 4
Juice: peach and apple
```

錯誤控制

- 假如呼叫程式發生問題，既有程式會繼續執行。那表示你可以從錯誤中回復。C標準程式庫提供一些內建程式可以提供使用者更多關於何事出錯的資訊。
- 大多數系統呼叫皆以同樣的方式處理失敗

```
execle("dinner_info", "dinner_info", "4", NULL, my_env);
puts("Dude - the dinner info_code must be busted");
```

若execle執行成功，則下一行不會被執行

- errno變數是定義在errno.h的全域變數

```
EPERM=1          Operation not permitted
ENOENT=2         No such file or directory
ESRCH=3          No such process
EMULLET=81       Bad haircut
```

系統失敗處理：
- 盡可能清理乾淨
- 為errno變數設定錯誤值
- 回傳 -1

```
puts(strerror(errno));
```
← 用stderror查詢標準錯誤訊息

```
No such file or directory
```
例如系統找不到要執行的程式將顯示

範例

- 顯示網路組態指令：
  Linux和Mac機器上有/sbin/ifconfig
  Windows下有ipconfig
- 寫一個程式，當執行/sbin/ifconfig失敗時，改執行ipconfig

check_net.c

exec()

errno變數

strerror()顯示錯誤

若傳回 -1就表示失敗了

```c
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
int main()
{
    if (execl("/sbin/ifconfig", "/sbin/ifconfig", NULL) == -1)
      if (execlp("ipconfig", "ipconfig", NULL) == -1) {
          fprintf(stderr, "Cannot run ipconfig: %s", strerror(errno));
          return 1;
    }
    return 0;
}
```

Newshund.c讀取RSS

```
> export RSS_FEED=http://www.cnn.com/rss/celebs.xml
> python rssgossip.py 'pajama death'
Pajama Death launch own range of kitchen appliances.
Lead singer of Pajama Death has new love interest.
"I never ate the bat" says Pajama Death's Hancock.
```

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main(int argc, char *argv[])
{
    char *feeds[] = {"http://www.cnn.com/rss/celebs.xml",
                     "http://www.rollingstone.com/rock.xml",
                     "http://eonline.com/gossip.xml"};
    int times = 3;
    char *phrase = argv[1];
    int i;
    for(i = 0; i < times; i++){
      char var[255];
      sprintf(var, "RSS_FEED=%s", feeds[i]);
      char *vars[] = {var, NULL};

      if (execl("/usr/bin/python", "/usr/bin/python", "./rssgossip.py", phrase, NULL,
vars) == -1) {
            fprintf(stderr, "Can't run script: %s\n", strerror(errno));
            return 1;
      }
    }
    return 0;
}
```

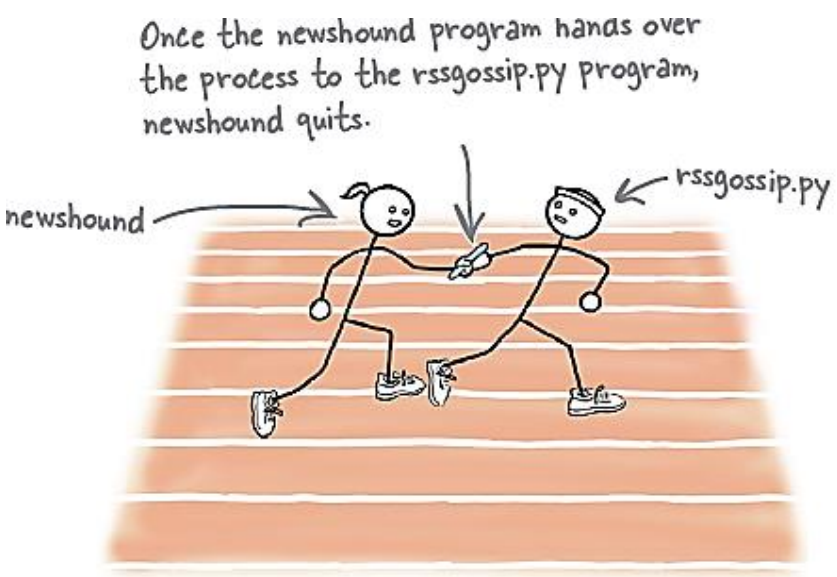
第一個繞行，行程就交棒給rssgossip.py了

```
> ./newshound 'pajama death'
Pajama Death ex-drummer tells all.
New Pajama Death album due next month.
```

exec()是程式的結尾

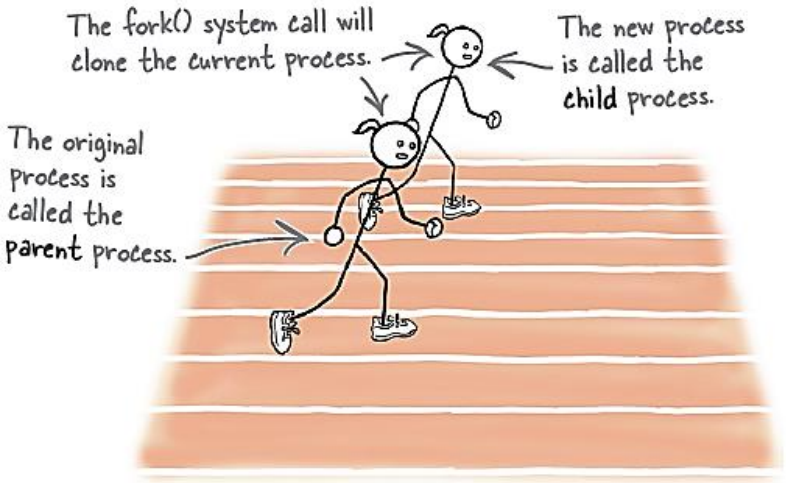
```
for (i = 0; i < times; i++) {
    ...
    if (execle("/usr/bin/python", "/usr/bin/python","./rssgossip.py", phrase, NULL, vars) == -1) {
    ...
    }
}
```

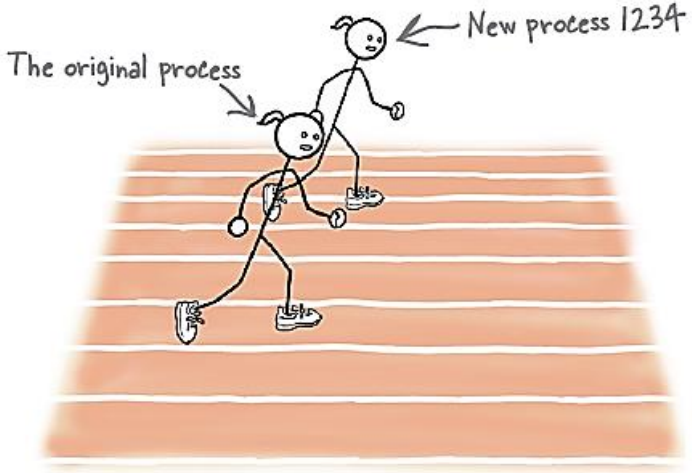
此迴圈只被run一次，整個程式就跳出了

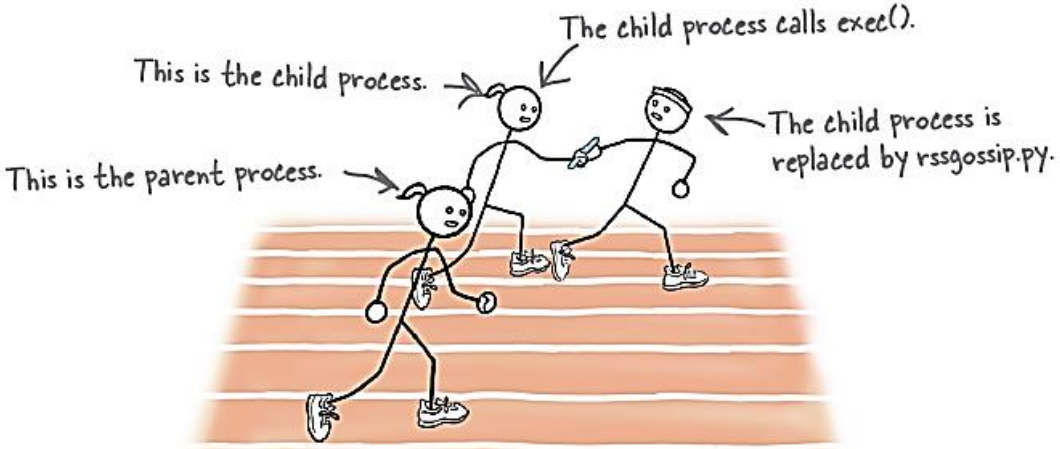


使用fork()複製行程

```
You call fork() like this:
pid_t pid = fork();
```







fork()傳回0給子行程，傳回正的整數值給父行程，父行程會收到子行程的PID

讓子行程交棒給其他行程就OK了

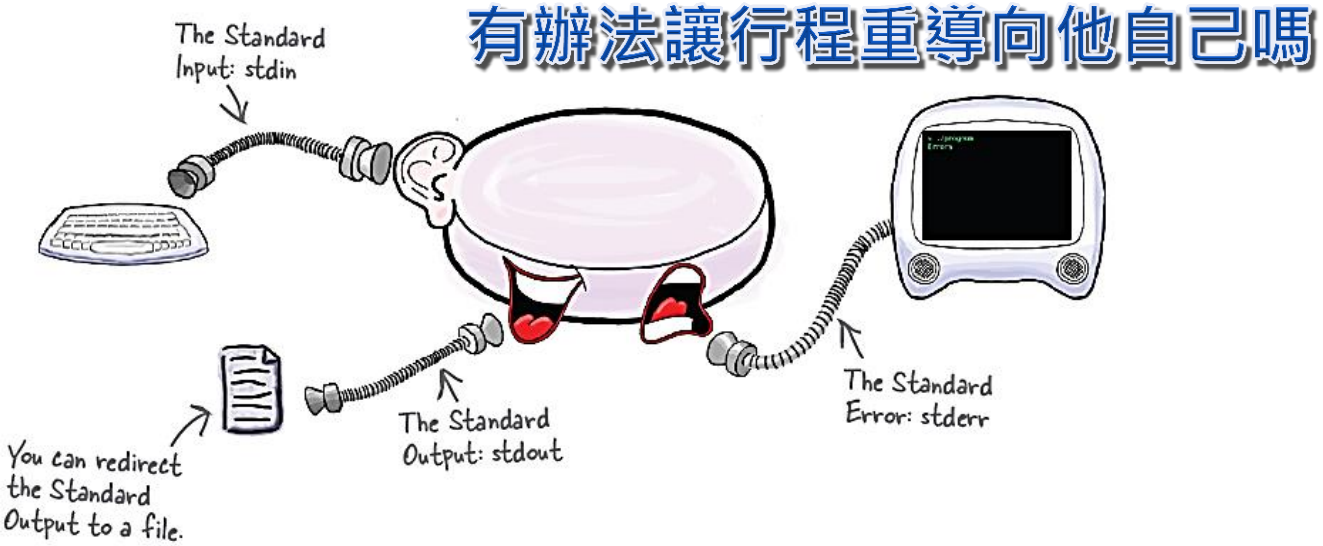
```
for (i = 0; i < times; i++) {
    char var[255];
    sprintf(var, "RSS_FEED=%s", feeds[i]);
    char *vars[] = {var, NULL};
    pid_t pid = fork();

    if (pid == -1) {
      fprintf(stderr, "Can't fork process: %s\n", strerror(errno));
      return 1;
    }

    if (!pid) {
      if (execle("/usr/bin/python", "/usr/bin/python", "./rssgossip.py",phrase, NULL, vars) == -1) {
            fprintf(stderr, "Can't run script: %s\n", strerror(errno));
            return 1;
          }
    }
}
```

```
> ./newshound 'pajama death'
Pajama Death ex-drummer tells all.
New Pajama Death album due next month.
Photos from the surprise Pajama Death concert.
Official Pajama Death pajamas go on sale.
"When Pajama Death jumped the shark" by HenryW.
Breaking News: Pajama Death attend premiere.
```

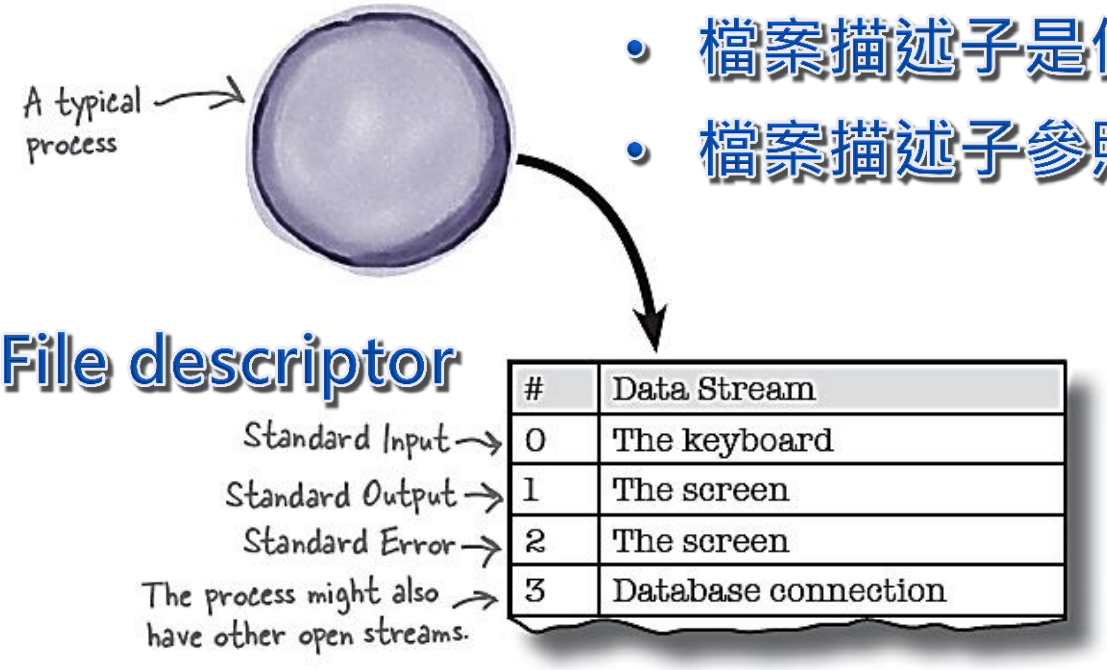
行程間通訊

- 產生行程只是一半的故事
- 行程間通訊(IPC, Interprocess communication)讓各個行程協同合作。
- 透過IPC跟系統上的其他程式交談，讓你的程式碼威力倍增。
- 重導向：

```
python ./rssgossip.py Snooki > stories.txt
```



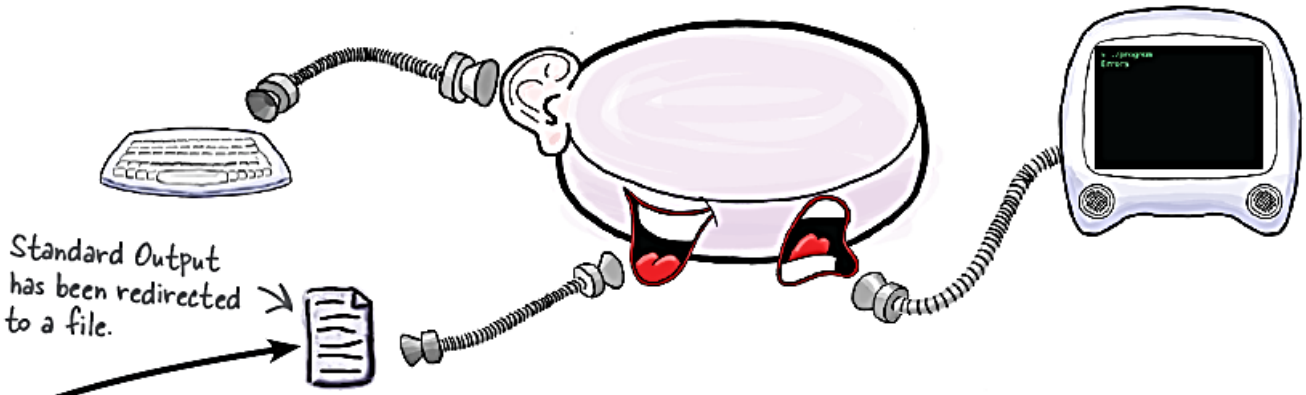
典型的行程



- 檔案描述子是代表資料串流的數字
- 檔案描述子參照的不一定是檔案

表格前三個總是相同，slot0是stdin，slot1是stdout，slot2是stderr

重導向只是替換資料流



That means if you want to redirect the Standard Output, you just need to switch the data stream against descriptor 1 in the table.

#	Data Stream
0	The keyboard
1	~~The screen~~ File stories.txt
2	The screen
3	Database connection

在指令列作重導向：

```
./myprog > output.txt 2> errors.log
```

```
./myprog 2>&1
```

將標準錯誤重導向到標準輸出去

行程可以透過重寫描述子表格來進行他們自己的重導向工作

fileno()告訴你描述子/dup2()複製串流

當你要開啟檔案，OS會找到空的slot，傳回一個指向它的指標。


```
FILE *my_file = fopen("guitar.mp3", "r");
```

```
int descriptor = fileno(my_file);
```

這會傳回 4

#	Data Stream
0	The keyboard
1	The screen
2	The screen
3	Database connection
4	File guitar.mp3

使用dup2()複製資料串流

```
dup2(4, 3);
```

#	Data Stream
0	The keyboard
1	The screen
2	The screen
3	~~Database connection~~ File guitar.mp3
4	File guitar.mp3

錯誤處理程式碼

一大堆錯誤檢查，都是重複的程式碼

```
pid_t pid = fork();
if (pid == -1) {
    fprintf(stderr, "Can't fork process: %s\n", strerror(errno));
return 1;
}
```

```
if (execle(...) == -1) {
    fprintf(stderr, "Can't run script: %s\n", strerror(errno));
return 1;
}
```

改用exit()，不必return就直接終止程式，也可使程式寫得較簡單。

```
void error(char *msg)
{
    fprintf(stderr, "%s: %s\n", msg, strerror(errno));
    exit(1);
}
```

```
pid_t pid = fork();
if (pid == -1) {
    error("Can't fork process");
}
```

```
if (execle(...) == -1) {

    error("Can't run script");
}
```

重新導向

```
int main(int argc, char *argv[])
{
    char *phrase = argv[1];
    char *vars[] = {"RSS_FEED=http://www.cnn.com/rss/celebs.xml", NULL};
    FILE *f = fopen("stories.txt", "w");
    if (!f) {
          error("Can't open stories.txt");
    }

    pid_t pid = fork();
    if (pid == -1) {
          error("Can't fork process");
    }
    if (!pid) {
      if (dup2(fileno(f), 1)==-1) {
          error("Can't redirect Standard Output");
      }
      if (execle("/usr/bin/python", "/usr/bin/python", "./rssgossip.py", phrase, NULL, vars) == -1) {
          error("Can't run script");
      }
    }
    return 0;
}
```

#	Data Stream
0	The keyboard
1	File stories.txt
2	The screen
3	File stories.txt

將#1指向stories.txt

預期

```
> ./newshound2 'pajama death'
> cat stories.txt
Pajama Death ex-drummer tells all.
New Pajama Death album due next month.
```

實際

```
> ./newshound2 'pajama death'
> cat stories.txt
>
```

檔案內是空的

因為子行程慢，父行程有時必須等待

```
#include <sys/wait.h>

int main(int argc, char *argv[])
{
    char *phrase = argv[1];
    char *vars[] = {"RSS_FEED=http://www.cnn.com/rss/celebs.xml", NULL};
    FILE *f = fopen("stories.txt", "w");
    if (!f) {
        error("Can't open stories.txt");
    }

    pid_t pid = fork();
    if (pid == -1) {
        error("Can't fork process");
    }
    if (!pid) {
      if (dup2(fileno(f), 1)==-1) {
        error("Can't redirect Standard Output");
      }
      if (execle("/usr/bin/python", "/usr/bin/python", "./rssgossip.py", phrase, NULL, vars) == -1) {
        error("Can't run script");
      }
    }

    int pid_status;
    if (waitpid(pid, &pid_status, 0) == -1) {
        error("Error waiting for child process");
    }
    return 0;
}
```

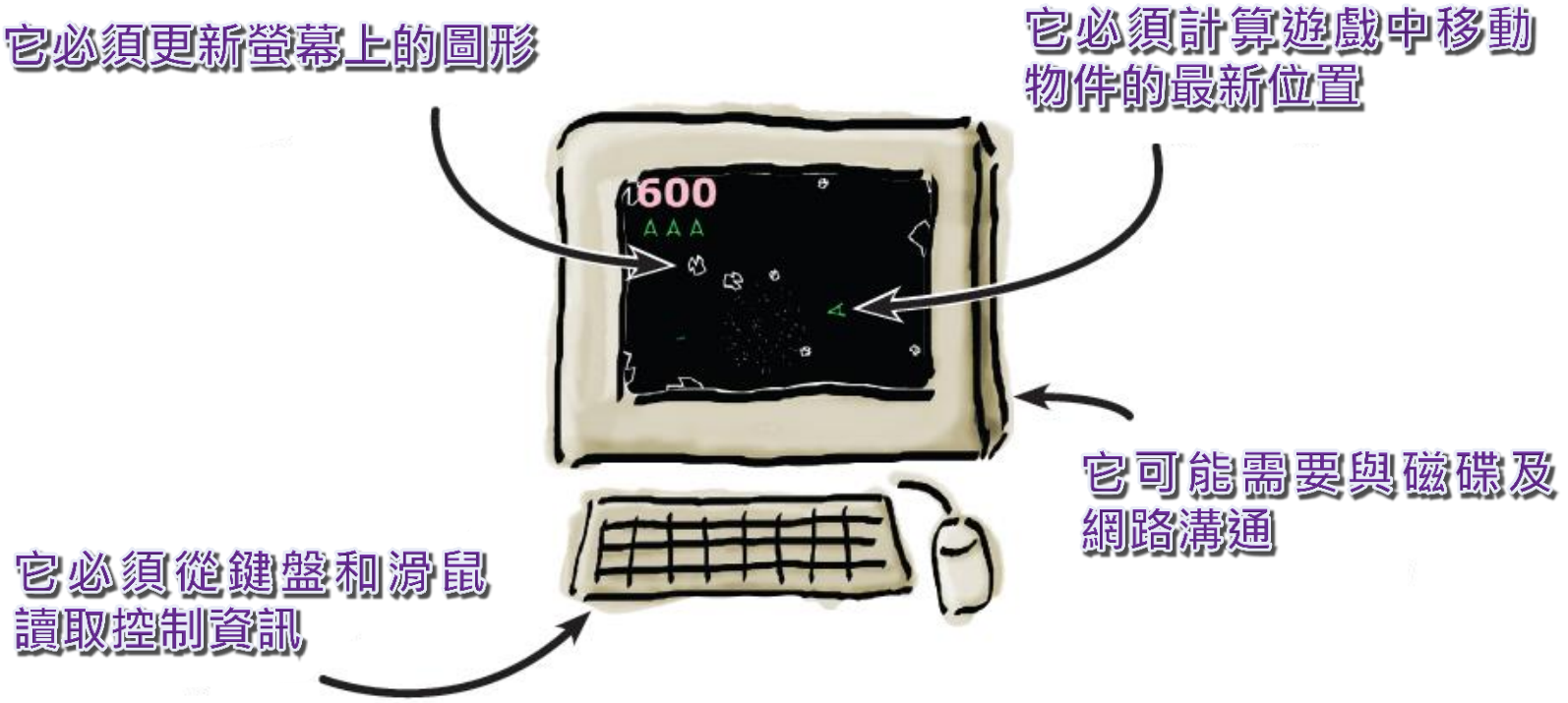
讓行程等待，以得到正確結果

```
> ./newshound2 'pajama death'
> cat stories.txt
Pajama Death ex-drummer tells all.
New Pajama Death album due next month.
```

0表示該函式會等到該行程結束

Ch11 並行發展的世界
執行緒

執行緒(Thread)：遊戲動起來



- 所有的事情，必須同時做
- 你的程式碼如何同時執行幾項不同的任務?

行程不總是解法

- 之前我們學到，透過行程你可以讓電腦同時做幾件事情。但行程不總是好辦法，因為：
  - **行程需要時間建立**

    如果你想執行的額外任務耗時很短，那麼每次建立行程來處理就不是很有效率。
  - **行程不能輕易共用資料**

    當你建立子行程時，它會自動從父行程為所有資料產生完整的副本；然而，那只是資料副本。如果子行程需要將資料送回給父行程，那麼就需要使用pipe之類的機制來完成。
  - **行程就是比較棘手**

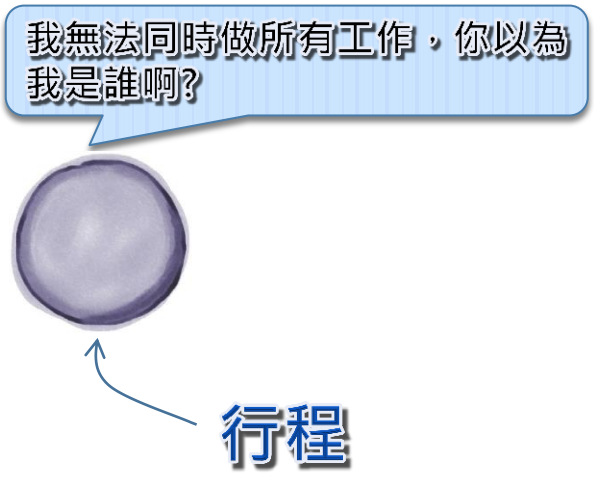
    你必須建立程式碼區塊來產生行程，那會讓你的程式既冗長又凌亂

➡ **你需要的是執行緒(thread)**

簡單的行程一次只做一件事

衝浪店

- 操作收銀機
- 補貨
- 替衝浪板打蠟
- 接電話
- 修屋頂
- 記帳



- 雇用額外人力：多個執行緒

所有執行緒都能夠存取相同的heap記憶體、讀寫相同的檔案並且在相同的網路socket交談。

如何建立執行緒： pthread程式庫

執行緒函式必須具有 void* 回傳型別

```c
void* does_not(void *a)
{
    int i = 0;
    for (i = 0; i < 5; i++) {
        sleep(1);
        puts("Does not!");
    }
    return NULL;
}
```

```c
void* does_too(void *a)
{
    int i = 0;
    for (i = 0; i < 5; i++) {
        sleep(1);
        puts("Does too!");
    }
    return NULL;
}
```

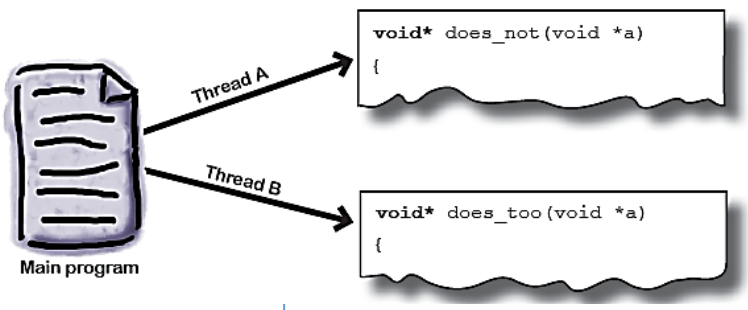
沒有東西要回傳，就傳回NULL吧!

- 用pthread_create建立執行緒

要引用這個!

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <pthread.h>
void error(char *msg)
{
    fprintf(stderr, "%s: %s\n", msg, strerror(errno));
    exit(1);
}
```

Multi-thread



用pthread_create建立執行緒

argument.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <pthread.h>
void error(char *msg)
{
    fprintf(stderr, "%s: %s\n", msg, strerror(errno));
    exit(1);
}

void* does_not(void *a)
{
    int i = 0;
    for (i = 0; i < 5; i++) {
        sleep(1);
    puts("Does not!");
    }
    return NULL;
}

void* does_too(void *a)
{
    int i = 0;
    for (i = 0; i < 5; i++) {
        sleep(1);
        puts("Does too!");
    }
    return NULL;
}
```

```
> gcc argument.c -lpthread -o argument
```

```
> ./argument
Does too!
...
Does not!
>
```

```
int main()
{
    pthread_t t0;
    pthread_t t1;
    if (pthread_create(&t0, NULL, does_not, NULL) == -1)
        error("Can't create thread t0");
    if (pthread_create(&t1, NULL, does_too, NULL) == -1)
        error("Can't create thread t1");

    void* result;
    if (pthread_join(t0, &result) == -1)
        error("Can't join thread t0");
    if (pthread_join(t1, &result) == -1)
        error("Can't join thread t1");
}
```

建立執行緒

pthread_join()等待執行緒結束

啤酒倒數

beer.c

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

int beers = 2000000;

void* drink_lots(void *a)
{
    int i;
    for (i = 0; i < 100000; i++) {
        beers = beers - 1;
    }
    return NULL;
}

int main()
{
    pthread_t threads[20];
    int t;
    printf("%i bottles of beer on the wall\n%i bottles of beer\n", beers, beers);
    for (t = 0; t < 20; t++) {
        pthread_create(&threads[t], NULL, drink_lots, NULL);
    }
    void* result;
    for (t = 0; t < 20; t++) {
        pthread_join(threads[t], &result);
    }
    printf("There are now %i bottles of beer on the wall\n", beers);
    return 0;
}
```

```
> ./beer
2000000 bottles of beer on the wall
2000000 bottles of beer
There are now 0 bottles of beer on the wall
> ./beer
2000000 bottles of beer on the wall
2000000 bottles of beer
There are now 883988 bottles of beer on the wall
> ./beer
2000000 bottles of beer on the wall
2000000 bottles of beer
There are now 945170 bottles of beer on the wall
>
```

WTF?

建立20個執行緒做倒數

程式碼不是執行緒安全

```
beers = beers - 1;
```

← 想像兩個執行緒同時執行這行程式碼

1. 首先，兩個執行緒必須讀取beers變數的當前值

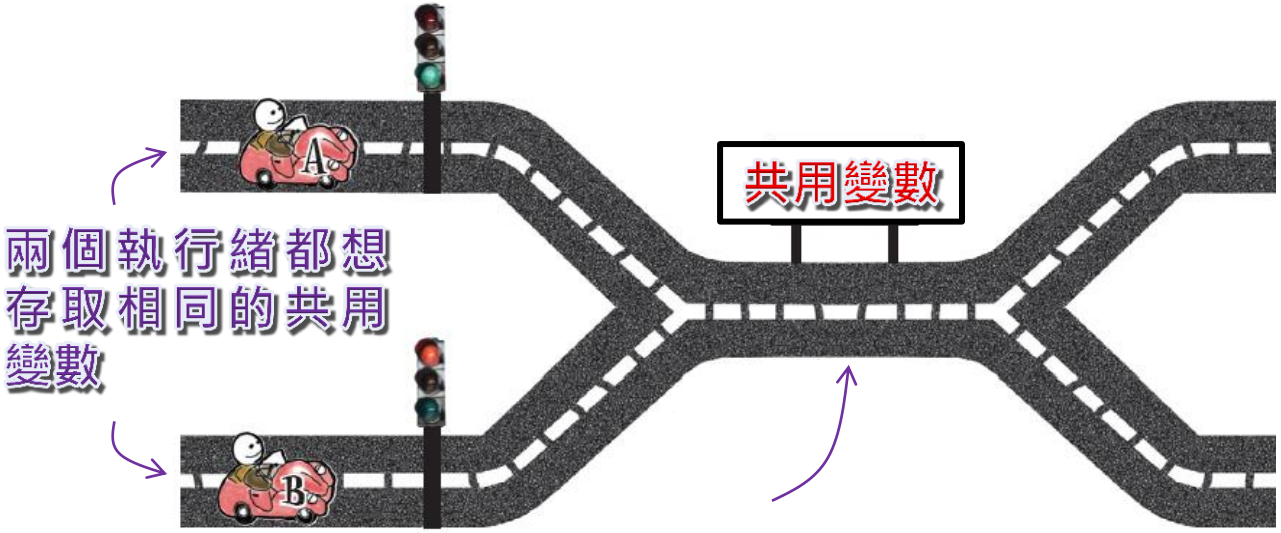
2. 然後，兩個執行緒都會把該數值減1

3. 最後，每個執行緒將 (beers – 1) 的值儲存回beers變數

結果beers並沒有被減2，因為執行緒會互相干擾。

你需要增加交通號誌：mutex

多執行緒(multi-thread)的程式威力強大，但是它可能會以出乎你預料的方式運作，你必須在裡頭安排一些適當的控制：mutex。



- 建立mutex (有時也稱為lock)

```
pthread_mutex_t a_lock = PTHREAD_MUTEX_INITIALIZER;
```

宣告為全域變數(放在函式外)：因為所有執行緒都要看見它

建立mutex

❶ 首先，建立一個mutex

```
pthread_mutex_t a_lock = PTHREAD_MUTEX_INITIALIZER;
```

❷ 紅燈停：在敏感程式碼開頭，放置第一個交通號誌

```
pthread_mutex_lock(&a_lock);
/* Sensitive code starts here... */
```

pthread_mutex_lock()將只讓一個thread通過，其他thread都必須乖乖等待

❸ 綠燈行：在敏感程式碼尾端解鎖，交通號誌變綠燈讓下一個thread通過

```
/* ...End of sensitive code */
pthread_mutex_unlock(&a_lock);
```

傳送long值給執行緒函式之探究

param.c

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

void* do_stuff(void* param)
{
    long thread_no = (long)param;
    printf("Thread number %ld\n", thread_no);
    return (void*)(thread_no + 1);
}

int main()
{
    pthread_t threads[20];
    long t;
    for (t = 0; t < 3; t++) {
        pthread_create(&threads[t], NULL, do_stuff, (void*)t);
    }

    void* result;
    for (t = 0; t < 3; t++) {
        pthread_join(threads[t], &result);
        printf("Thread %ld returned %ld\n", t, (long)result);
    }
    return 0;
}
```

轉回long

將long變數t轉成void指標

將回傳值轉為long

```
> ./param
Thread number 0
Thread number 1
Thread number 2
Thread 0 returned 1
Thread 1 returned 2
Thread 2 returned 3
```

在Beer.c不同的地方鎖定程式碼

版本A　beer_a.c

```
void* drink_lots(void *a)
{
    int i;
    pthread_mutex_lock(&beers_lock);
    for (i = 0; i < 100000; i++) {
      beers = beers - 1;
    }
    pthread_mutex_unlock(&beers_lock);
    printf("beers = %i\n", beers);
    return NULL;
}
```

```
> ./beer_a
2000000 bottles of beer on the wall
2000000 bottles of beer
beers = 1900000
beers = 1800000
beers = 1700000
beers = 1600000
beers = 1500000
beers = 1400000
beers = 1300000
beers = 1200000
beers = 1100000
beers = 1000000
beers = 900000
beers = 800000
beers = 700000
beers = 600000
beers = 500000
beers = 400000
beers = 300000
beers = 200000
beers = 100000
beers = 0
There are now 0 bottles of beer on the wall
>
```

在Beer.c不同的地方鎖定程式碼

版本B　beer_b.c

```
void* drink_lots(void *a)
{
    int i;

    for (i = 0; i < 100000; i++) {
      pthread_mutex_lock(&beers_lock);
      beers = beers - 1;
      pthread_mutex_unlock(&beers_lock);
    }
    printf("beers = %i\n", beers);
    return NULL;
}
```

```
> ./beer_b
2000000 bottles of beer on the wall
2000000 bottles of beer
beers = 92953
beers = 9786
beers = 8196
beers = 6625
beers = 5070
beers = 4072
beers = 3171
beers = 2615
beers = 2049
beers = 1607
beers = 1225
beers = 942
beers = 714
beers = 505
beers = 341
beers = 232
beers = 92
beers = 66
beers = 13
beers = 0
There are now 0 bottles of beer on the wall
>
```

總結

- 我們所介紹的東西，雖然不能涵蓋C語言的全部，但相信你已經對C語言的輪廓有很基本的認識了。
- 擁有基本觀念之後，接下來就是各自揮灑的時候了。有些人程式寫的簡潔漂亮又有效率，有人寫的擁腫凌亂又沒效率。別擔心，看多、做多了，你也可以。
- 接下來的8051嵌入式系統設計課程，你會看到很多我們在這裡所學到的東西。
- 假使忘了，沒關係，隨時翻書、上網找資料。有很多東西千萬不要用背的，因為把它們背起來其實並沒有多大的用處。