我們已經學會用 **printf()**，**scanf()**，**putchar()**，**getchar()** 來做輸入輸出。這些用來處理 I/O 的 functions 並不是 C 語言內建的，而是由 C library 提供，我們接著就來解釋輸入輸出的運作機制。另外我們也會介紹如何過濾輸入的資料，讓讀進來的資料格式能符合程式需求，這樣才能按照我們設計的方式順利處理資料。

使用 getchar() 與 putchar()

這兩個 functions 每次只能讀取或輸出一個字元，功能看似簡單，但是足以讓我們拿來處理各式各樣的輸入輸出狀況。用底下的範例來複習一下 **getchar()** 和 **putchar()**。

[範例]

```
#include <stdio.h>
int main(void)
{
   char ch;

   while ((ch = getchar()) != '@')
      putchar(ch);

   return 0;
}
```

這個程式執行起來像底下這樣：

```
Hi, there. My email address
Hi, there. My email address
is mac@apple.com.
is mac
```

迴圈 **while** 處理的過程是用 **getchar()** 讀取一個字元，然後用 **putchar()** 輸出一個字元。這個範例照理說應該是輸入一個字元就會回應一個字元例如

```
HHii,,  tthheerree..
```

但是程式執行起來卻是輸入了一整行之後 (按下 Enter 鍵)，才會把輸出的字元顯示出來。我們必須先了解 C 程式如何處理鍵盤輸入，以及所謂的 buffering 機制。由於 **getchar()** 會透過 buffer 來緩衝輸入的字元，所以要等到換行字元出現，才會把 buffer 裡的東西送到程式做處理。所以我們會看到鍵盤輸入的字元先被直接回應到螢幕上 (系統為了要讓我們知道究竟打了哪些字)，同時這些輸入的字元會被存在 buffer 中，等到使用者打了 Enter 才把 buffer 裡的字元送到程式，然後程式的迴圈就把字元一一處理。
另一方面，使用特殊符號 (譬如 **'@'**) 當作結束輸入的字元，也會讓那個字元原本的用途變得無法被正常使用 (有時候我們還是會需要用到它，譬如要打 email address)。所以最好是能用某個絕對不會出現在輸入裡的字元，當作輸入終止的判斷條件。C 確實也提供了一個專門用來表示輸入終止的符號讓我們使用。
我們先暫時跳開，先簡介一個概念：在 C 裡面，輸入和輸出的裝置被當成檔案來看待，輸入就像在讀檔，輸出就像是在寫入檔案，這麼做的好處是可以讓程式可以有統一的介面來處理輸入和輸出，不會因為換到不同系統和不同裝置就變得不能執行。在 stdio.h 裡定了兩個標準的檔案串流叫做 **stdin** 和 **stdout**，分別對應到標準輸入裝置 (通常假設是鍵盤) 和標準輸出裝置 (通常假設為螢幕)。所以如果我們從 **stdin** 讀取資料，就等於是從鍵盤讀取使用者輸入的內容，如果把資料寫入 **stdout** 就等於把資料送到螢幕上。我們之前學過的 **scanf()** 和 **getchar()** 都是預設從 **stdin** 讀資料，而 **printf()** 和 **putchar()** 都預設為寫入 **stdout**。在 stdio.h 裡面定了一個常數 **EOF** (End of File)，用來代表檔案結束。既然輸入被當作檔案來處理，我們在讀取 **stdin** 的資料的時候，就可以判斷讀入的字元是否等於 **EOF**，用這個檔案結束符號來判斷使用者是否已經結束輸入的動作。所以前面的範例程式我們可以改寫成

```
#include <stdio.h>
int main(void)
{
   int ch;

   while ((ch = getchar()) != EOF)
      putchar(ch);

   return 0;
}
```

我們剛才說過，**EOF** 在 stdio.h 裡面有定義，通常它的值被定成 **-1**，但是可能因作業系統而有不同。另外，配合 **EOF** 我們把 **ch** 的型別改成 **int** (其實 **getchar()** 和 **putchar()** 的回傳值和參數本來就都是 **int** 型別)。最後我們要知道的是如何用鍵盤輸入 **EOF** (不是輸入 **E O F** 三個字母，也不是直接打 **-1**)。這在不同作業系統下會有不同的方式，在 DOS 下的作法是在新的一行的開頭按 Ctrl+Z (在Unix 則是Ctrl+D)，這樣就會相當於是 End of File。

假設我們把前一個範例改成 EOF 的版本，取名為 reflect.c，然後 compile 並產生執行檔，假設執行檔的名稱叫做 reflect.exe。在 command line 模式下試試看下面的指令

D:\code\> **reflect.exe < reflect.c**

你會發現 reflect.c 的內容會被顯示到螢幕上。不論是在 DOS 或是 Unix, Linux 都提供了所謂 redirection 的功能。這項功能可以把作業系統裡一般的檔案 (譬如純文字檔)，導向成為標準輸入裝置 (對我們的程式來說就是 **`stdin`**)。用法很簡單，就是像上面的例子用 < 符號，看起來就像把 reflect.c 餵給 reflect.exe。所以 reflect.exe 不再是從鍵盤讀取資料，而是從檔案 reflect.c 讀取資料。對我們的程式來說完全沒差，因為我們的程式本來就是把 **`stdin`** 當作檔案來處理。由於 reflect.c 本身是純文字檔，它的檔案結束正是靠 **`EOF`** 來標記，所以 reflect.exe 讀了整個檔案直到碰到 **`EOF`** 就結束。
除了輸入可以導向，輸出也同樣可以，用法是

D:\code\> **reflect.exe > ten_words.txt**

也就是用 > 符號接上我們要輸出的檔案名稱。執行程式之後我們可以輸入一堆字元，輸入完畢後，在新的一行的開頭按 Ctrl+Z 結束程式。這時候如果去檢查目錄會發現多了一個檔案叫做 ten_words.txt。我們可以看看它的內容是什麼 。我們就直接用剛學會的方式 (reflect.exe < ten_words.txt) 來看檔案內容，會發現它的內容就是我們剛才用鍵盤輸入的那些字元。所以 reflect.exe > ten_words.txt 這個指令的效果相當於把資料寫入檔案 ten_words.txt 裡面。因為我們用 **>** 符號把 **`stdout`** 導向到檔案 ten_words.txt，所以資料不再送到螢幕上，而是送到 ten_words.txt。
最後還可以把兩種導向合併，變成

D:\code\> **reflect.exe < reflect.c > reflect_copy.c**

這個動作相當於把檔案 reflect.c 複製到檔案 reflect_copy.c。
學會了 redirection 這項小技巧之後，處理檔案資料就變得很方便。我們可以先把資料編輯好，再用 < 導向送入程式作處理，不需要每次都在線上打字輸入。而程式輸出的資料也可以用 > 導向，送到檔案裡儲存起來。

[補充] 用字元陣列儲存數字

```
int main(void)
{
   char str1[8];
   int str2[8]; /* char str2[8]; */
   int i;

   scanf("%7s", str1);

   for (i=0;i<strlen(str1);i++)
      str2[i] = str1[i] - '0';
   ...
   return 0;
}
```

str1[0] str1[4]

'4'	'5'	'0'	'3'	'\0'			

str1[0] str1[4]

52	53	48	51	0			

```
str2[0] = str1[0] - '0'
str2[1] = str1[1] - '0'
str2[2] = str1[2] - '0'
str2[3] = str1[3] - '0'
```

str2[0] str2[4]

4	5	0	3	???			

用 **scanf("%7s", str1)** 方式讀進來的資料是字串，存入字元陣列中每個陣列的元素都是一個字元。由於在 C 裡面字元是用 ASCII 值來表示，所以 **str1** 的內容就像上圖第二列顯示的一樣。我們為了要把這些阿拉伯數字字元當做 **0** 到 **9** 的整數值來運算，必須先把他們從數字字元換算成整數，方法是減去 **'0'** 這個字元，或是說減去 **48**。

[註] 雖然 **int str2[8];** 宣告成 **int** 型別的陣列，但是其實也可以宣告成字元陣列，也就是 **char str2[8];**，這樣對於後續把 **str2** 拿來做整數運算完全沒有影響，因為 **char** 型別其實也是整數，只是長度只能記錄到一個 byte (8 bits) 而已，可以表示的整數範圍只在 **-128** 到 **127** 之間。

請試試看底下幾種顯示方式各自會得到什麼結果

顯示 **str1** 的第四個元素

```
printf("%d\n", str1[3]);
printf("%c\n", str1[3]);
```

顯示 **str2** 的第四個元素

```
printf("%d\n", str2[3]);
printf("%c\n", str2[3]);
```

把 **str1** 當作字串顯示

```
printf("%s\n", str1);
```

把 **str2** 當作字串顯示

```
printf("%s\n", str2);
```