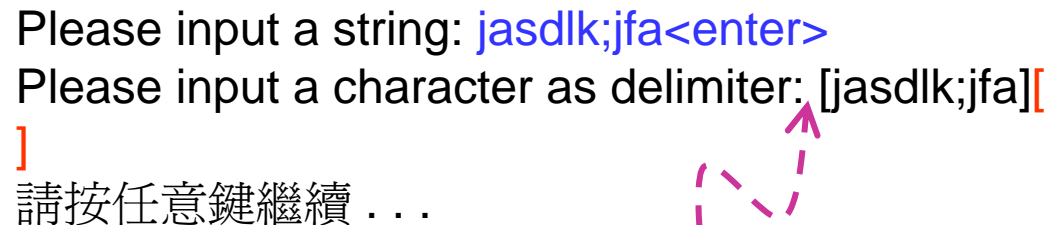


scanf 的緩衝區問題  
與  
scanf 支援的資料剖析功能

丁培毅

# 問題 1 描述

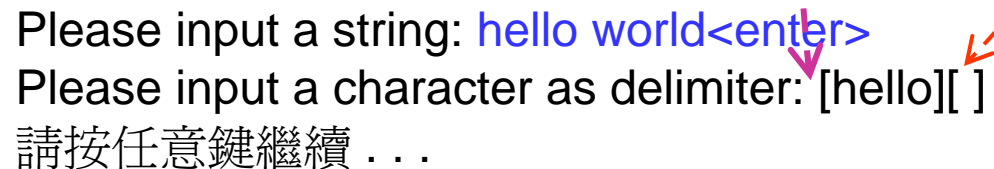
```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 int main() {
05     char str[100];
06     char symbol='\0';
07
08     printf("Please input a string: ");
09     scanf("%s",str);
10     printf("Please input a character as delimiter: ");
11     scanf("%c", &symbol);
12     printf("[%s][%c]\n", str, symbol);
13
14     system("pause");
15     return 0;
16 }
```



```
Please input a string: jasdlk;jfa<enter>
Please input a character as delimiter: [jasdlk;jfa][
]
```

請按任意鍵繼續...

爲什麼程式執行起來第  
11 列怎麼沒有停下來讓  
操作者輸入一個字元?



```
Please input a string: hello world<enter>
Please input a character as delimiter: [hello][ ]
]
```

請按任意鍵繼續...

是系統表現不穩定還是你誤會它的表現了?

慢速 I/O 裝置和快速 cpu  
中間調節速度的區域

# 鍵盤緩衝區的問題

- 首先 手⇒鍵盤⇒stdio函式庫 (scanf) 的鍵盤緩衝區⇒變數
- 你在鍵盤上打 `hello world<enter>`, 都會在按下 `<enter>` 後被低階驅動程式搬進鍵盤緩衝區, 不會有東西消失了, 你也要特別注意換列字元 `'\n'`, 只要按一次 `<enter>` 就會有一個 `'\n'` 字元進入緩衝區
- 其次, 所有在緩衝區裡的資料都是由程式中 `scanf()`, `getchar()`, `getc()`, `gets()` 來處理的 [請注意 `getch()`, `getche()`, 和 `kbhit()` 不是 `stdio` 函式庫裡的函式, 處理不到這個緩衝區裡面的資料]
- 接下來請注意需要完全了解 `scanf()` 每一個控制命令 `scanf()` 所做的動作, 例如 `%s` 是“跳過0或多個 white space, 由鍵盤緩衝區裡讀取連續不是 white space 的字元”, 所謂 white space 包括 空格, `'\t'`, 和 `'\n'` 三個字元
- 又例如 `%c` 是“不跳過任何字元, 直接由鍵盤緩衝區裡讀取單一一個字元”;
- `%d` 是跳過所有 white space, 由鍵盤緩衝區裡讀取連續 0~9 之間的十進位數字, 轉換為二進位, 如果除了 white space 之外只看到不是 0~9 的字元, `scanf("%d",&x)` 回傳 0 (注意是回傳 0 代表這個命令沒有成功, x 的數值不變)
- 了解上面這些以後你才會知道 `scanf("%s",str); scanf("%c",&symbol);`當輸入 `hello world<enter>`時 "hello" 會進入 str 陣列, 接下來的 空格 就進到 symbol 了,所以才會覺得 `scanf("%c",&symbol);` 怎麼沒有停下來等候輸入

- 運用 `scanf()` 的 "`%c`" 格式字串, 控制 `scanf` 讀取 `symbol` 時需要 “跳過所有的 white spaces”

```
printf("Please input a string: ");  
scanf("%s",str);  
printf("Please input a character as delimiter: ");  
scanf("%c", &symbol);
```

- 執行結果  也可以用 `\n` 或 `\t` 取代

```
Please input a string: asdfasdf  
Please input a character as delimiter: a  
[asdfasdf][a]  
請按任意鍵繼續...
```

程式表現符合預期

```
Please input a string: hello world  
Please input a character as delimiter: [hello][w]  
請按任意鍵繼續...
```

讀到空格後面的 'w' 了



怎麼沒有讀到 world



還是沒有停下來讓  
操作者輸入一個字元?



- 運用 `gets(str)` 函式由鍵盤緩衝區讀取包含 **white spaces** 在內的一整列資料 (到第一個 `'\n'` 為止), `str` 陣列不包含 `'\n'`, 但是 `'\n'` 會由鍵盤緩衝區中移除

兩者功能一模一樣

```
printf("Please input a string: ");
gets(str);
printf("Please input a character as delimiter: ");
scanf("%c", &symbol);
```

讀出所有不是 `'\n'` 的資料, `'\n'` 還在緩衝區; 如果沒有任何資料的話, `str` 維持是空的

或是 {

```
str[0] = 0;
scanf("%[^\n]", str);
scanf("%*1[\n]");
```

由緩衝區讀出1個 `'\n'` 字元, 不存入任何變數

- 執行結果

```
Please input a string: hello world
Please input a character as delimiter: GG
[hello world][G]
請按任意鍵繼續...
```

請注意想要跳過 `'\n'` 不要用 `scanf("\n");`; 這樣是跳過所有的 **white space**

程式表現完全符合預期

- 運用 `getchar()` 及 `fflush()` 函式取代 `scanf("%c",...)`; 其中 `fflush(stdin)` 是由鍵盤緩衝區中移除所有資料, 此處就是 `"\nworld\n"` 但是和 `scanf("%c",...)`; 的效果還是有一點點差別, 就是 `getchar()` 不會跳過額外的空格字元

```
printf("Please input a string: ");
scanf("%s",string);
fflush(stdin);
printf("Please input a character as delimiter: ");
symbol = getchar();
```

請注意: `fflush(stdin)` 在線上測試系統 e-Tutor 上, 在 linux GNU gcc/g++, 在 Mac clang gcc/g++ 都沒有作用, `fflush(輸出串流)` 是有明確定義的動作, `fflush(輸入串流)` 則沒有明確的定義, 請避免使用

- 執行結果

```
Please input a string: hello\nworld
Please input a character as delimiter: G
[hello][G]
請按任意鍵繼續...
```

程式只能讀到 **hello**,  
但是可以讀到 **G**

# 不同鍵盤緩衝區問題

- conio 的 getch(), getche(), kbhit() 是沒有緩衝區的輸入函式  
iostream 的 cin >>, cin.get(), cin.getline(). cin.ignore()  
stdio 的 scanf(), getchar(), gets(), fflush()

兩個函式庫各有自己的鍵盤緩衝區

- 雖然可以藉由 ios\_base::sync\_with\_stdio(true) 來同步, 但是一般來說 iostream 和 stdio 不要混著使用, 例如

```
std::ios_base::sync_with_stdio(false);  
scanf("%s", str1);  
std::cin >> str2;
```

- 在鍵盤上輸入 hello world<enter>every body<enter> 之後, (g++ only)  
str1 的內容是 **hello**, str2 的內容是 **every** 而不是 **world**

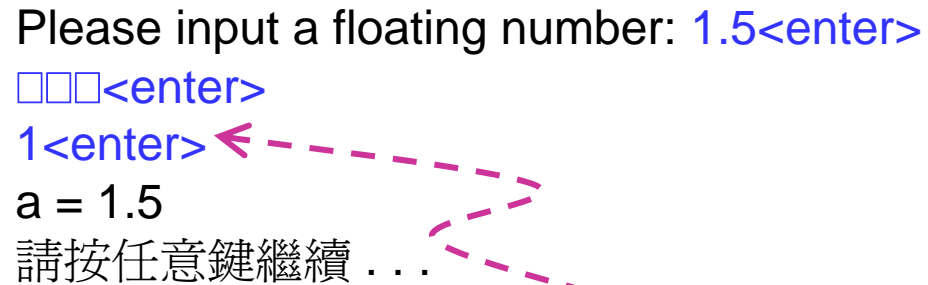
- 下面範例解釋 conio 和 stdio 共用的狀況

```
scanf("%s", str); c = getche();
```

- 在鍵盤上輸入 hello world<enter>d之後, (both g++ 4.8.1 and vc2010)  
str1 的內容是 **hello**, c 的內容是 **d** 而不是 **w**

# 問題 2 描述

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 int main() {
04     float a;
05     printf("Please input a floating number: ");
06     scanf("%f\n", &a);
07     printf("a = %f\n", a);
08     system("pause");
09     return 0;
10 }
```



```
Please input a floating number: 1.5<enter>
□□□<enter>
1<enter>
a = 1.5
請按任意鍵繼續 . . .
```

為什麼程式執行起來第06列讀了1.5進去以後一直停在那裡等候輸入不繼續執行呢？一直按<enter>或是空白都沒有用

仔細看一下程式，很多同學都會修改，可以讓程式正確運作，可是能不能給一個正確的解釋呢？這樣子以後才不會又遇見一樣的錯誤啊！換成這樣呢？

```
06     scanf("%f□", &a);
```

沒有辦法解釋的話，可能還是你誤會它的表現了？



# 功能超級強大的 scanf

- scanf 這樣的函式不是從 C 才開始有的, Algol 68 裡面就有 readf 這樣的 **輸入解析 (input parsing)** 函式, 大部分人都只知道 **%d %u %lf %s %c** 這些格式命令, 覺得需要和實際參數一一對應很麻煩, 而且需要用 **&** 運算子取得變數位址作為參數是不太好瞭解、很容易出錯的
- 所以一些入門的書乾脆不用 C 的 **stdio** 函式庫而用 C++ 裡面的 **iostream** 函式庫, 理由是**簡單**, 不需要解釋太多東西就可以順利運作
  - `int x; double y; char z; char w[100];`  
`cin >> x >> y >> z >> w;` 就打發掉一切輸入了
  - 可是其實這並不是 **iostream** 函式庫的用意, 它的設計是以**物件化**為主要目的, 使用它的話程式可以很容易地**擴充**任意物件**輸入輸出序列化**的功能, **維持封裝**的完整性 (如果在使用的時候不知道上面的這些的話, 那就只是是爲了簡單而用它, 也就需要接受它比 **scanf** 功能少很多很多的事實, 當然所有的功能都可以自己寫, 自己加上去的, 可是相信我在知道 **scanf** 強大的功能以後, 你不會想要這麼做的)

# 資料剖析範例

- 範例一: 請讀取下列資料到三個浮點數陣列裡

```
double mandarin[2][2],  
       math[2][2],  
       english[2][2];
```

```
1年1班學生國文成績平均為 76.80  
1年1班學生 數學 成績平均為 68.00  
1年1班學生英文 平均為 67.40  
1年2班學生英文平均為 68.80  
1年2班學生國文成績平均為 61.60  
1年2班學生 數學 成績平均為 57.00  
2年1班學生英文平均為 69.40  
2年1班學生國文成績平均為 71.80  
2年1班學生 數學 成績平均為 73.00  
2年2班學生國文成績平均為 75.60  
2年2班學生 數學 成績平均為 80.20  
2年2班學生英文平均為 59.80
```

請注意假設資料量很大, 格式與順序有點不太整齊, 不要手動編輯資料檔案來修改資料的格式, 是程式需要考量這些資料的變異性的

- 範例二: 請讀取右側程式設定資料到下列陣列裡

```
char id[2][20];  
char nickname[2][20];  
int loginTimes[2];  
int lastLoginYear[2];  
int lastLoginMonth[2];  
int lastLoginDay[2];
```

```
[user]  
ID=giddens  
nickname=九把刀  
loginTimes=868  
lastLogin=20101226
```

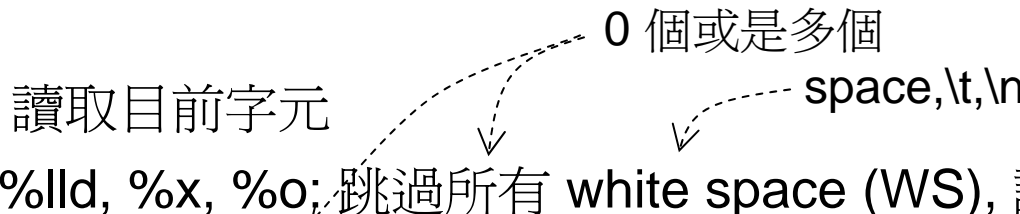
```
[user]  
ID=bonddealer  
nickname=總幹事  
loginTimes=32493  
lastLogin=20100210
```

如果你曉得結構的用法的話, 也可以把這些資料讀到下面結構陣列裡

```
struct {  
    char id[20], nickname[20];  
    int loginTimes, lastLoginYear,  
        lastLoginMonth, lastLoginDay;  
} user[2];
```

請注意資料裡面 = 號以及前面的字串是給其它編輯器使用的, 讀進程式時比對正確即可, 不需記錄下來, 格式錯誤就直接結束程式

# scanf 格式命令用法

- `%c` 讀取目前字元
  - `%d, %lld, %x, %o`: 跳過所有 white space (WS), 讀取 10/16/8 進位整數
  - `%f, %lf`: 跳過所有 WS, 讀取 10 進位浮點數, 例如 123.456e5
  - `%s`: 跳過所有 WS, 讀取任意非 WS 字串
- 上述命令若目前字元為 WS 則一直等候輸入; 不是指定格式資料就提前結束
- `%n`: 把此次 `scanf` 呼叫在緩衝區裡已經處理過(讀入或是跳過)的字元數轉換為整數
  - **return value**: 此次呼叫 `scanf` 成功讀取轉換為數值的資料筆數 (不包括 `%n`), 如果已經到達串流結尾則回傳 EOF (-1)
  - `%wc`: 由目前字元讀取 `w` 個字元, 第一個字元存入變數中, 不夠 `w` 個字元時會等待使用者輸入
  - `%wd, %wx, %wo, %ws, %wf`: 跳過所有 WS, 讀取其後 `w` 個字元, 遇見不合法字元時, 將已讀入之資料轉換好, 提前結束
  - 沒有 `%w.pf` 這種格式命令
- 

# scanf 格式命令用法 (cont'd)

- space, \t, \n: 跳過所有 WS (請注意後兩者很容易以為是比對單一字元)
- 非 WS 的字元 c: 比對目前字元是否為指定字元, 是則跳過繼續處理其他格式命令, 否則提前結束此次 scanf (如何得知成功與否? 在命令之後加上 %n 命令, 檢查有沒有讀入目前字元數), c 與 %\*1[c] 效果相同
- %% 代表單一個百分號
- %w[a-zA-Z0-9,/] 讀入符合規則的最多 w 個字元到字串變數中  
%\*w[^0-9] 讀入最多 w 個 0-9 的字元, 不存到任何變數中  
%\*w[^ \t\n] 讀入最多 w 個不是 white space 的字元, 不存到任何變數中

注意 1. int ivar, char cvar;

① scanf("%d", &ivar); scanf("%c", &cvar);

② scanf("%d%c", &ivar, &cvar);

當串流裡資料格式正確  
使得 %d 命令順利完成  
時, ①② 兩寫法是等效的

2. scanf 所有的參數都是記憶體位址, 連格式字串那個參數也是

3. fscanf() 和 sscanf() 所接受的命令和 scanf() 一樣, 一個由檔案串流裡讀取/剖析資料, 一個是由字元陣列 (字串) 裡讀取/剖析資料

# 資料格式命令範例

1. `char c; scanf("%c",&c);` // 不可以用 `int c; scanf("%c",&c);`;
2. `int d; scanf("%d",&d); scanf("%x",&d); scanf("%o",&d);`;
3. `long long lld; scanf("%lld",&lld);`;
4. `char buf[100]; scanf("%s",buf);` // 請不要用 `scanf("%s",&buf);`;
5. `char buf[100]; scanf("%[a-z]",buf);` // 讀入所有小寫字母的字元  
請注意用 `%s` 命令或是 `%[xyz]` 時變數一定要是字元陣列, 不可以是整數、浮點數、字元之類的, 編譯器不會檢查到, 可以運作但是會造成執行錯誤, 以及接續的記憶體內容被破壞
6. 上面是基本的 `scanf` 用法, 所有用法都可以在 `%` 之後加上 `*`, 告訴 `scanf`: 請根據命令處理資料串流, 但是讀到的資料不要放進任何變數裡
7. 所有的用法也可以在 `%` 和 `*` 之後加上一個數字來限定最多處理幾個輸入字元, 可以利用來處理特別的資料格式, 也可以用 `char buf[51]; scanf("%50s",buf);` 來避免 `scanf` 讀進來的東西寫到錯誤的記憶體位置去, 避免所謂的 `buffer overflow attack`

# 資料格式命令範例 (cont'd)

8. `char c; scanf("□%c",&c);` 目的是爲了讀取串流中接下來不是 `white space` 的一個字元, 有幾個等效的寫法讓你參考 `"\n%c"`, `"\t%c"`, `"□%1c"`, `"%*[□\t\n]%1c"`, `"%*[□\t\n]%1[^\t\n]"`, 你不需要真的用這些怪怪的命令, 但是如果你看懂爲什麼它們都有相同的功效, `scanf` 這些命令的精神你就掌握了一大半
9. 請注意 `scanf("□%d",&d);` 或是 `scanf("□%s", buf);` 命令裡的空格都是多餘的; `scanf("%d□",&d);` 或是 `scanf("%s□", buf);` 命令裡的空格會使得程式一直想要跳過 `white space`, 所以在讀到需要的資料之後, 你會發現一直按 `<enter>`, 空格時, `scanf` 都一直不結束, 不會繼續往下執行下一列的程式, 直到你按下不是 `white space` 的任意字元再加上 `<enter>`, `scanf` 才結束, 繼續往下執行下去, 當然 `scanf("%d\n",&d);` 也有相同的效果, 不要用錯了, 無法解釋程式的表現而以爲電腦有問題

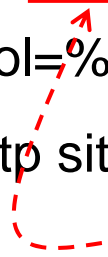
# 資料格式命令範例 (cont'd)

10. `char id[21]; sscanf("ID=aB1_Ab-9","ID=%20[-_a-zA-Z0-9]",id);`  
這個範例裡一開始的 `ID=` 會比對成功所以繼續執行 `%20[...]` 的命令, 讀取最多 20 個 `a-z` 之間或是 `A-Z` 之間或是 `0-9` 之間或是減號或是底線的字元到陣列 `id` 中, 所以陣列裡的資料會是 `aB1_Ab-9`, 如果 `ID=` 沒有比對到, `scanf` 會提早結束, 回傳數值會是 0, 如果接下來完全沒有合法的文字 `scanf` 也會提早結束, 回傳數值會是 0, 如果合法的文字小於或是等於 20 個, 就會把合法的文字放到陣列 `id` 裡面, 如果合法的文字超過 20 個, 只會把前 20 個放到陣列 `id` 裡面, 只要有讀到一個以上的字元到陣列 `id` 中, 回傳數值就是 1
11. `int a,nItems,nChars=-1; nItems=scanf("%d%n",&a, &nChars);`  
如果成功讀到了 `a` 的數值, `nItems` 的數值就是 1, 同時 `nChars` 裡面會是這一次 `scanf` 已經處理的字元數, 如果 `a` 沒成功讀取, 則 `nItems` 會是 0, `nChars` 會維持 -1, 如果讀取資料到變數 `a` 之前串流已經結束, `nItems` 會是 EOF (-1), `nChars` 會維持 -1

# 資料剖析範例

```
char name[20], tel[50], code[20], protocol[10], site[50], path[50];  
int age;
```

- `sscanf("name:john age:40 tel:(0912)123456",  
"name:%s age:%d tel:(%[0-9])%[0-9]", name, &age, code, tel);  
printf("%s %d %s-%s\n", name, age, code, tel); // john 40 0912-123456`
- `sscanf("name:john age:40 tel:082-313530",  
"%*[^:]:%s %*[^:]:%d %*[^:]:%s", name, &age, tel);  
printf("%s %d %s\n", name, age, tel); // john 40 082-313530`
- `sscanf("http://ccckmit.wikidot.com/cp/list/hello.txt",  
"%[^:]:%*2[/]%[^/]/%[a-zA-Z0-9._/-]", protocol, site, path);  
printf("protocol=%s site=%s path=%s\n", protocol, site, path);  
// protocol=http site=ccckmit.wikidot.com path=cp/list/hello.txt`

 應該用 //