

C/C++資料結構與程式設計

語法復習

NTU CSIE 張傑帆

C/C++資料結構與程式設計

整合開發環境

NTU CSIE 張傑帆

C++開發工具

- 整合式開發環境
(Integrated Development Environment)
簡稱**IDE**
- 是整合編輯、編譯、測試、除錯、與執行等功能的
程式開發軟體
- 例如Borland公司的C++ Builder、IBM公司的
VisualAge C++、Microsoft公司的Visual C++ 等都是
整合式的C++ 程式開發軟體

下載Dev C++

在 Google 搜尋 “Dev C++”，並選擇搜尋所有網頁，則可找到BloodShed Software Dev C++的官方網站。

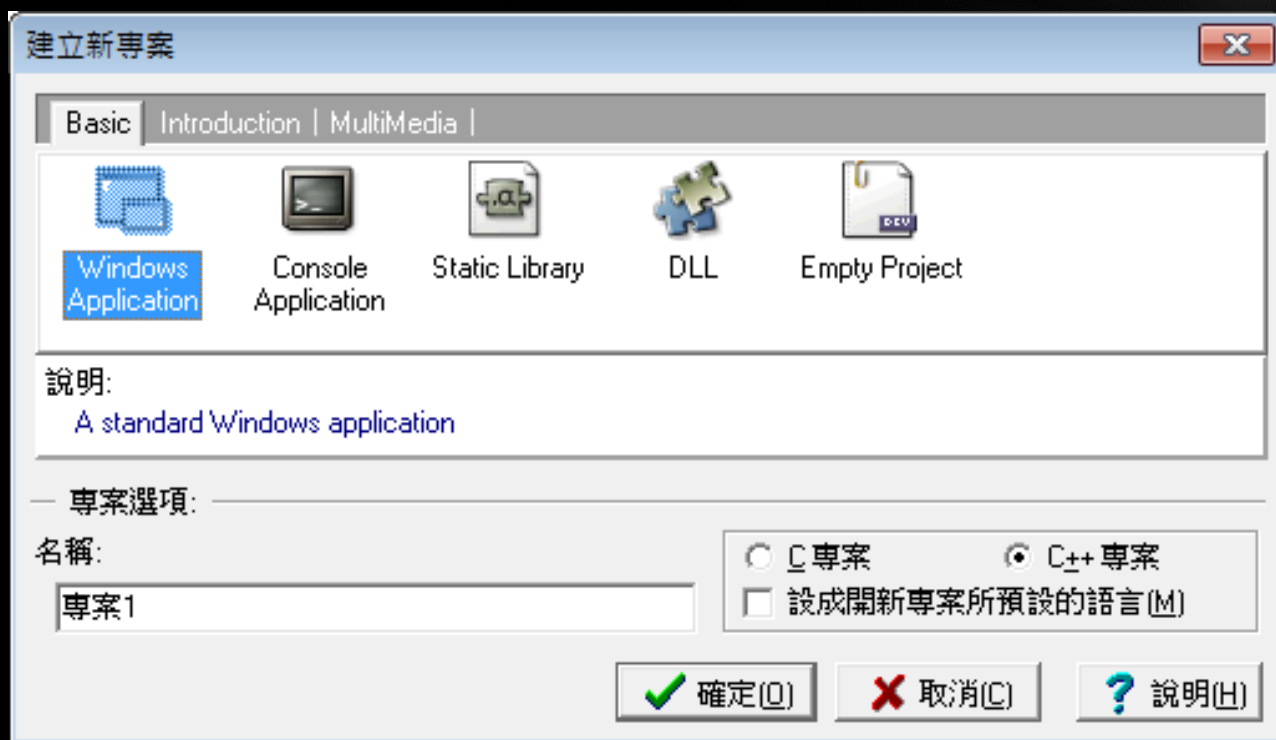
衍生版本

下載最新版的Orwell Dev-C++ 5.11版

阿榮免安裝版

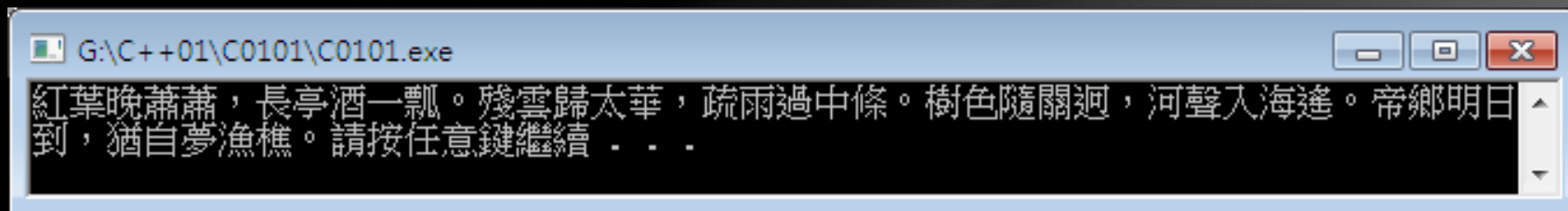
Orwell's Engine 官網

使用Dev C++ 5.0



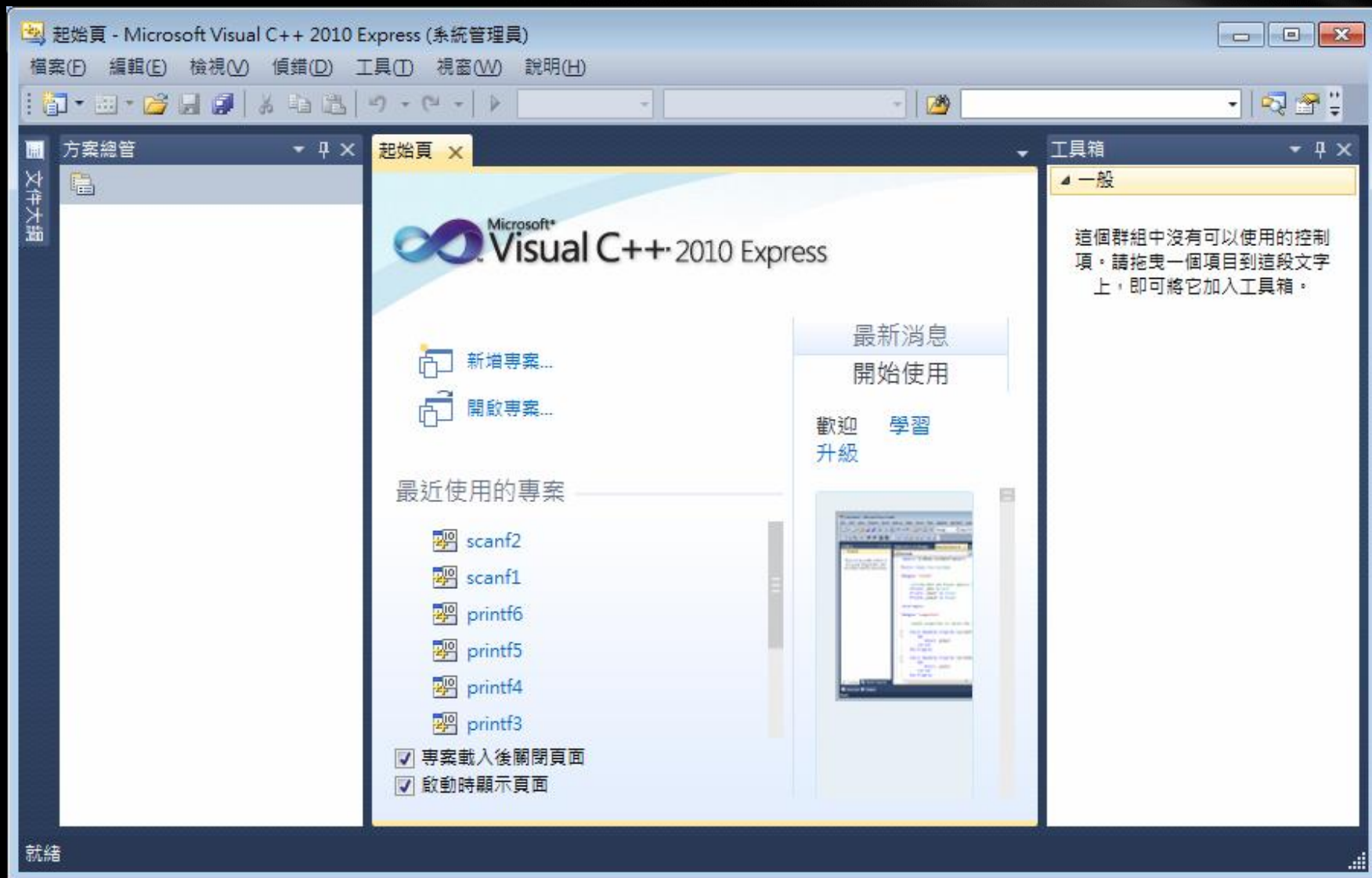
編譯與執行

2. 由於這是新專案，所以Dev-C++ 會要求選擇儲存main.cpp的位置，預設儲存位置是與專案相同的目錄，因此按存檔就可以了。
3. 編譯完成且程式沒有語法錯誤後，出現命令提示字元視窗，並根據cout指令輸出字串如下：



```
G:\C++01\C0101\C0101.exe
紅葉晚蕭蕭，長亭酒一瓢。殘雲歸太華，疏雨過中條。樹色隨關迴，河聲入海遙。帝鄉明日到，猶自夢漁樵。請按任意鍵繼續 . . .
```

使用Visual C++ 2010/2012/2013



Visual C++

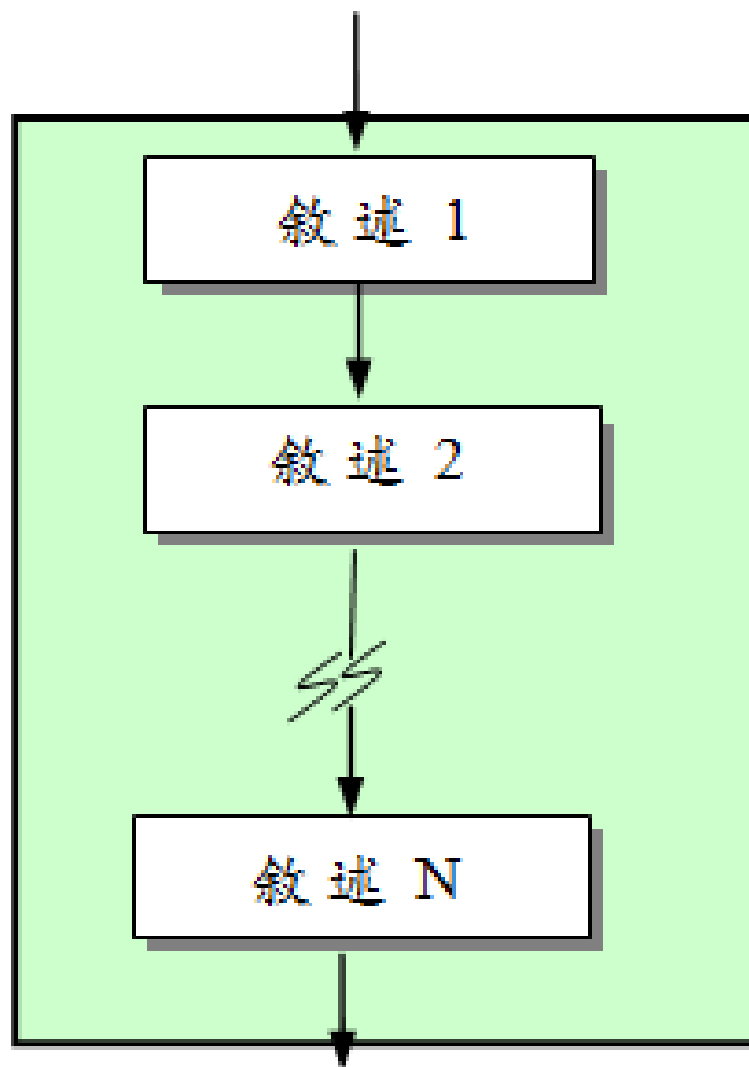
Visual C++ 2010，可以上微軟（ Microsoft ）官網下載及安裝Visual C++ 2010/2012/2013 Express。

C/C++資料結構與程式設計

選擇結構

NTU CSIE 張傑帆

選擇結構簡介



循序結構

start

選擇結構

天氣好？

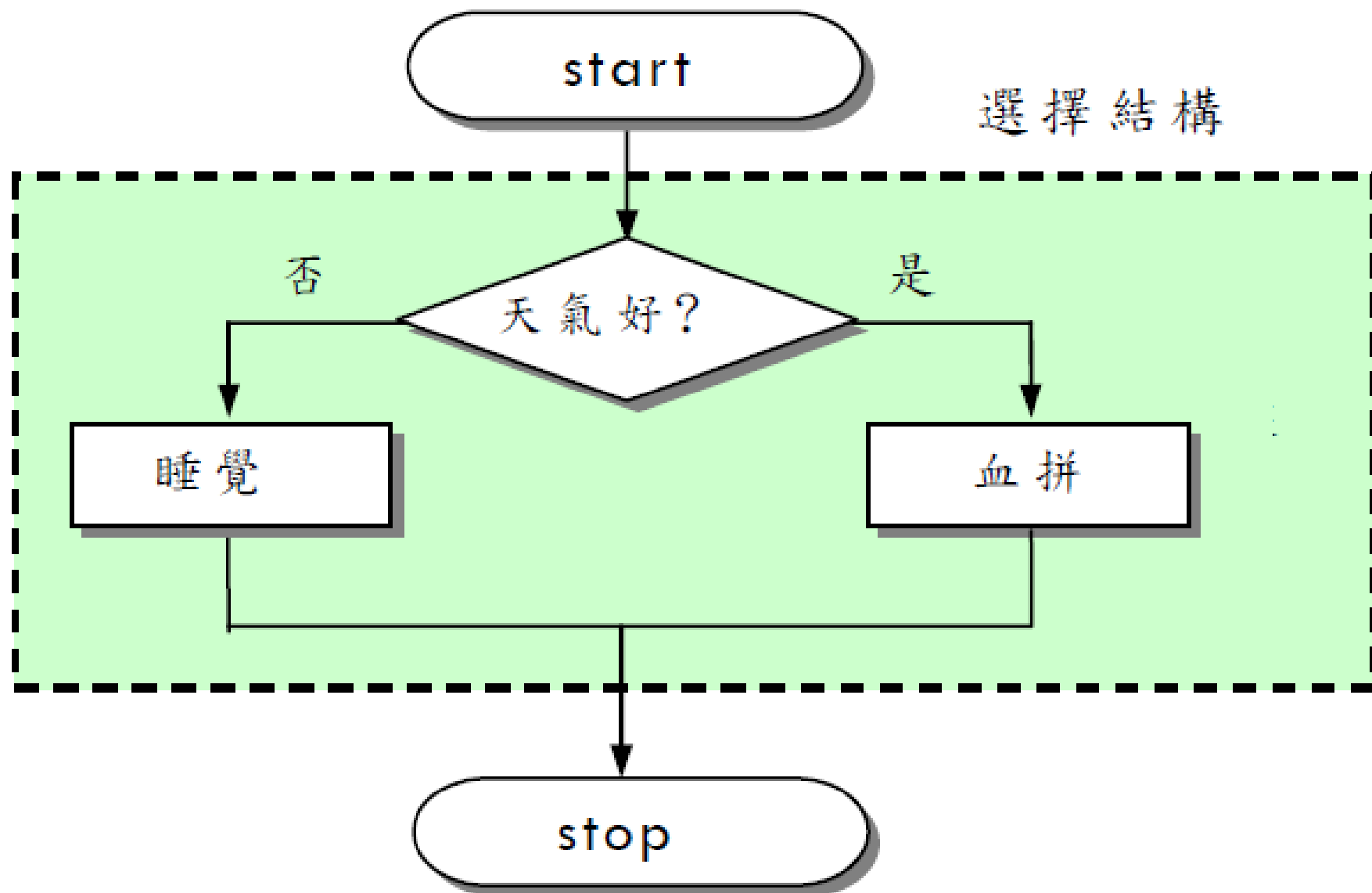
否

是

睡覺

血拼

stop



選擇敘述

if-else敘述

程式中的選擇結構有如口語中的「如果.....就.....否則....」，在C語言中是使用if-else敘述來達成。

如語法

若 條件 成立時

則執行接在if後面的 敘述區段1。

否則(條件不成立)

執行接在else後面的 敘述區段2。

語法 1

```
if (條件)  
    敘述 1;  
[else  
    敘述 2; ]
```

語法 2

```
if (條件)  
{  
    ⋮ 敘述區段 1  
}  
[ else  
    {  
        ⋮ 敘述區段 2  
    } ]
```

說明

1. 如果 if-else 的選擇敘述所要執行的敘述只有一行時，可以使用語法 1，省略左大括號 { 及右大括號 }。
2. 若敘述超過一行以上時，就必須使用語法 2。

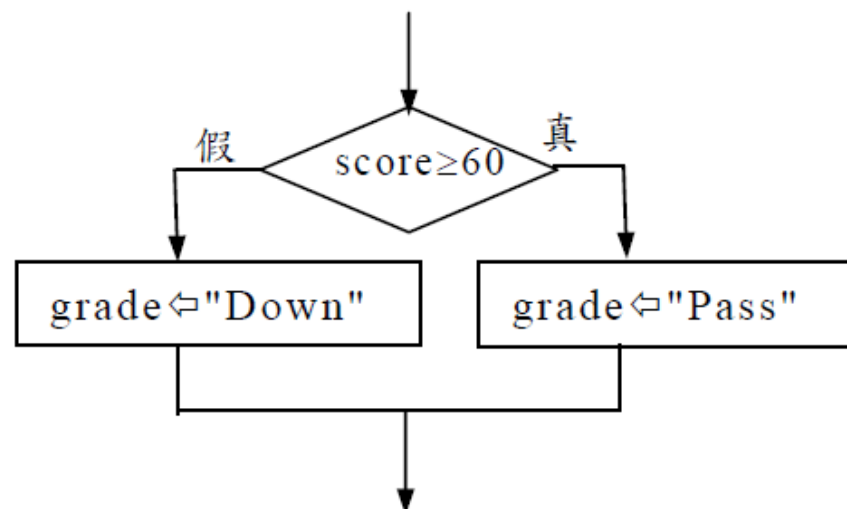
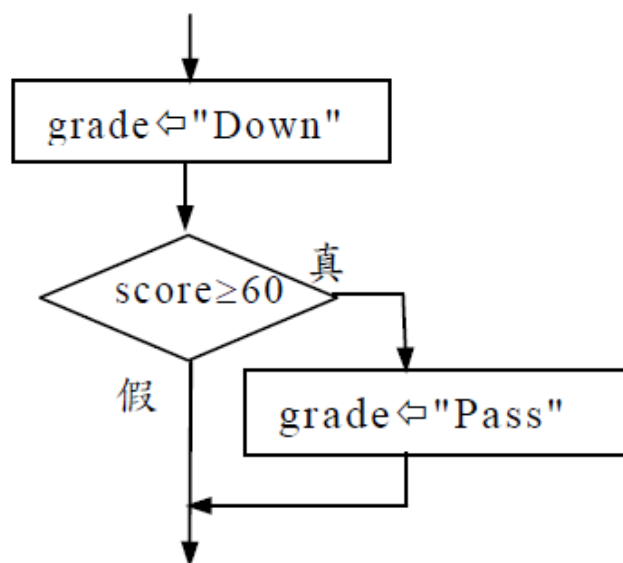
3. 上面語法中的 [...] 中括號內的敘述是當不滿足條件且不執行任何敘述時，此部份可省略。
4. 譬如：由分數 score 來判斷是否 Pass(及格)或 Down(不及格)？若 $\text{score} \geq 60$ 顯示 "Pass"；如果 $\text{score} < 60$ 顯示 "Down"，有下列兩種撰寫方式：

使用單一選擇(省略 else 敘述)

```
grade = "Down" ;  
if (score >= 60)  
    grade = "Pass" ;
```

使用 if...else 敘述：

```
if (score >= 60)  
    grade = "Pass" ;  
else  
    grade = "Down";
```



關係運算子

「關係運算子」(Relational Operator)是用來比較關係運算子左右兩邊的運算式

關係運算子將比較的結果傳回

比較的結果為真，傳回值為1

比較結果不成立時傳回值為0(代表假)

在C語言中的關係運算子是透過大於、小於或等於運算子組合成下表中的六種狀態

運算子	說明	使用例	結果
== (相等)	判斷此運算子左右兩邊運算式的值是否相等。	18 == 18	1(真)
		18 == 35	0(假)
		3+2 == 1+4	1(真)
!= (不相等)	判斷此運算子左右兩邊運算式的值是否不相等。	17 != 18	1(真)
		56 != 56	0(假)
		12*3 != 3*12	0(假)
< (小於)	判斷此運算子左邊運算式的值是否小於右邊運算式的值。	17 < 18	1(真)
		42 < 30	0(假)
		2 < 10-7	1(真)
> (大於)	判斷此運算子左邊運算式的值是否大於右邊運算式的值。	19 > 18	1(真)
		26 > 36	0(假)
		12*3 > 12*2	1(真)
<= (小於等於)	判斷此運算子左邊運算式的值是否小於等於右邊運算式的值。	17 <= 18	1(真)
		18 <= 18	1(真)
		10+3 <= 12	0(假)
>= (大於等於)	判斷此運算子左邊運算式的值是否大於等於右邊運算式的值。	17 >= 18	0(假)
		18 >= 18	1(真)
		12*3 >= 35	1(真)

邏輯運算子

當一個條件中有兩個以上的關係運算式需要一起做判斷時，就必須使用到邏輯運算子來連接。

邏輯運算子是用來判斷兩個以上關係運算式之間的關係，這在程式設計的流程中是很常用的，至於邏輯運算式的表示語法如下：

結果 = 運算式₁ 邏輯運算子 運算式₂ ;

```
result = ( a==b || c>=d )
```


邏輯運算子	說明	真值表		
&& (AND, 且)	此運算子左右兩邊的運算式結果若不為零值，結果為1(真)；否則為零值(假)。	運算式 1	運算式 2	結果
		非 0	非 0	1
		非 0	0	0
		0	非 0	0
		0	0	0
 (OR, 或)	此運算子左右兩邊的運算式結果只要其中有一個不為零值，結果就是 1；兩個都為零結果才是零值。	運算式 1	運算式 2	結果
		非 0	非 0	1
		非 0	0	1
		0	非 0	1
		0	0	0
! (NOT)	此運算子是單一的運算，主要是將敘述結果相反，即 $1 \Rightarrow 0$ ， $0 \Rightarrow 1$ 。	運算式	結果	
		非 0	0	
		0	1	

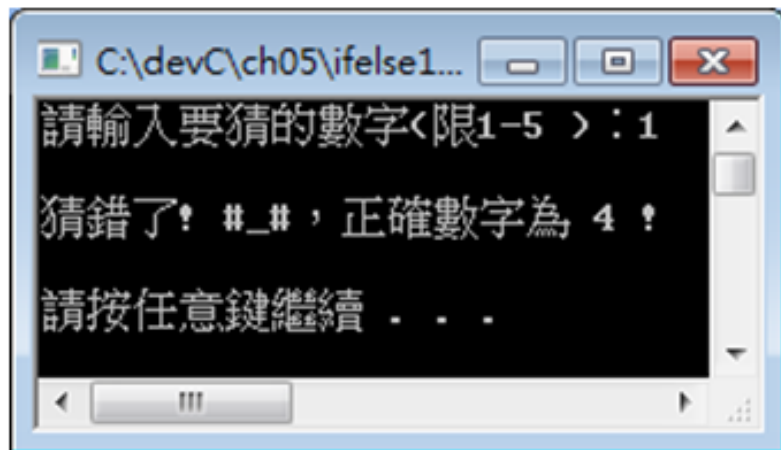
實機練習



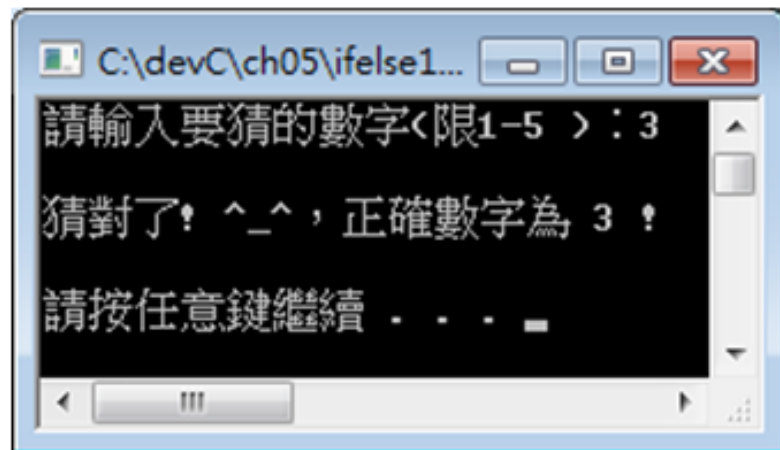
範例：ifelse1.c

製作簡易的猜數字遊戲。程式執行時電腦自動產生 1~5 之間的亂數，接著等待您由鍵盤鍵入猜的數字。若猜對，則顯示「猜對了」相關訊息；若猜錯，則顯示「猜錯了」相關訊息。

執行結果



猜錯畫面



猜對畫面

1. 若在程式中直接設定被猜數字，導致每次執行所猜的數字都一樣，此種作法程式不具彈性。C 語言如下面敘述提供 `srand()` 函式，配合 `rand()` 函式可產生介於 0~32,767 間的亂數，如此每次執行時，所產生被猜的數字會不一樣。可用時間當做亂數產生器的種子，寫法如下：

```
guess=rand();
```

以上面得到的種子產生 0~32,767 的整數置入 `guess` 整數變數，寫法如下：

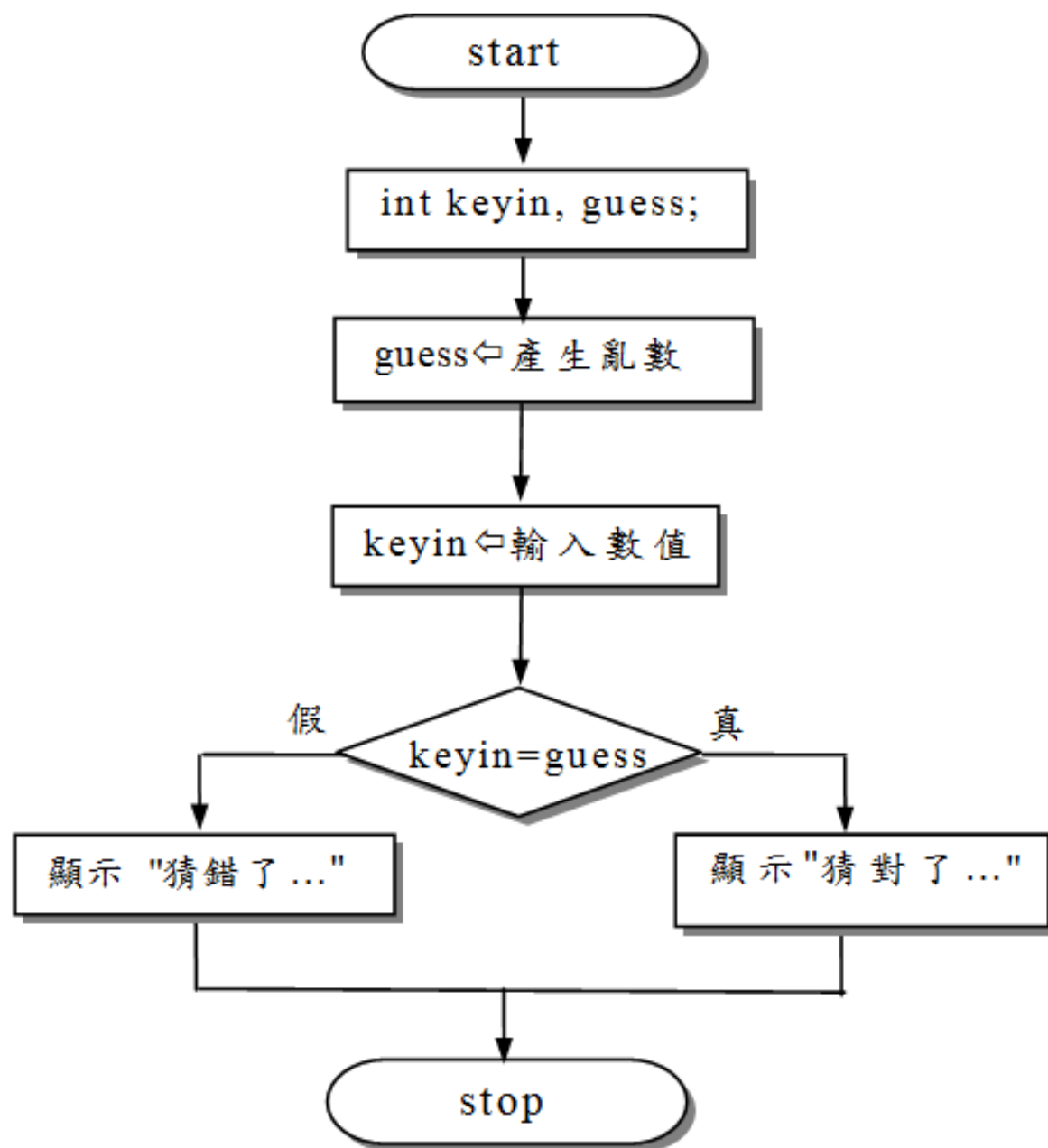
2. 將產生介於 0~32,767 的亂數除以 5 取其餘數，結果所產生的餘數會介於 0~4 之間。寫法如下：

```
rand()%5
```

3. 產生介於 1~5 之間的整數亂數，並指定給 `guess` 整數變數，寫法如下：

```
guess=rand()%5+1;
```

4. 由於 `srand()` 和 `rand()` 兩個函式都宣告在 `stdlib.h` 標頭檔中，`time()` 函式宣告在 `time.h` 標頭檔，因此，在程式開頭要記得含入 `#include <stdlib.h>` 和 `include <time.h>` 兩個敘述。



程式碼 FileName : ifelse1.c

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include <time.h>
04
05 int main(int argc, char *argv[])
06 {
07     int keyin, guess;
08     srand((unsigned)time(NULL));
09     guess=rand()%5+1;
10     printf("請輸入要猜的數字(限 1-5 ) : ");
11     scanf("%d", &keyin);
12     if (keyin==guess)
13         printf("\n 猜對了! ^_^, 正確數字為 %d !\n", guess);
14     else
15         printf("\n 猜錯了! #_#, 正確數字為 %d !\n", guess);
16     printf("\n");
17     system("PAUSE");
18     return 0;
19 }
```

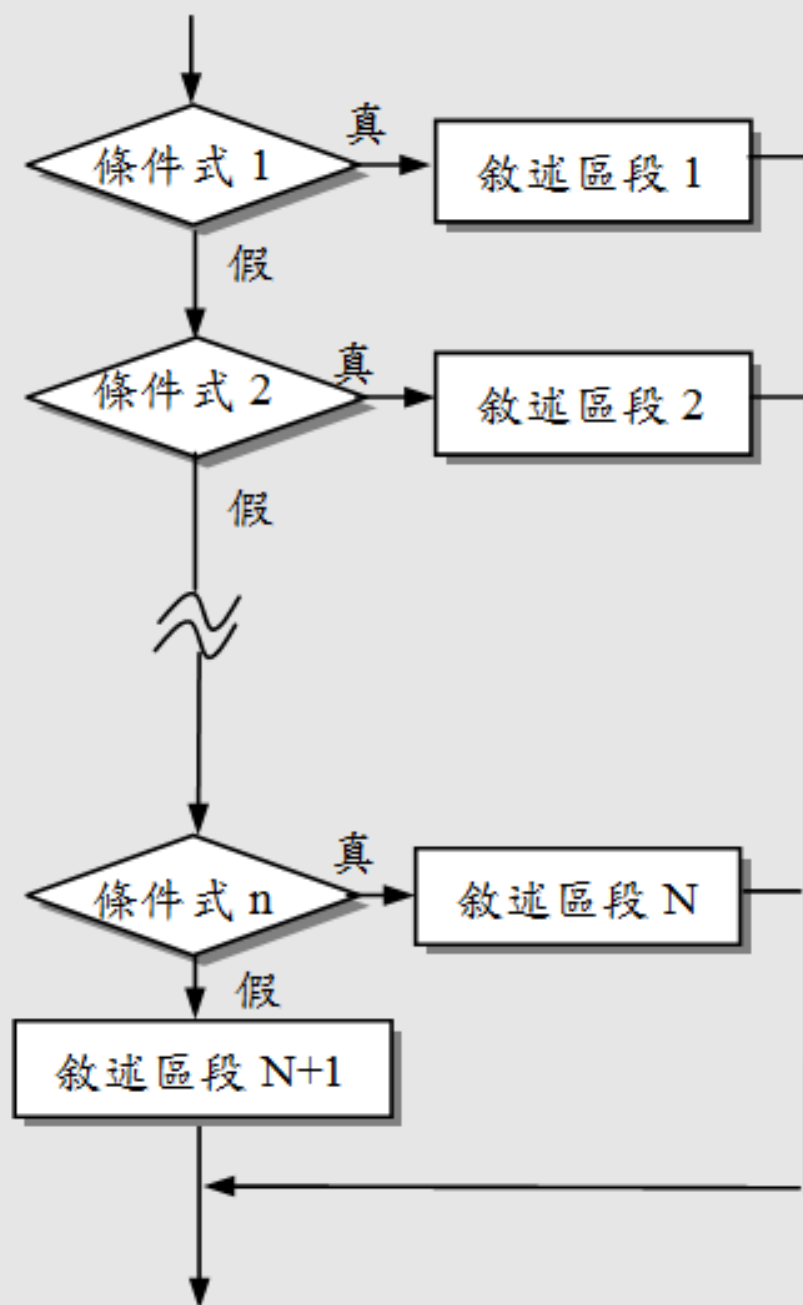
if-else if敘述

如果判斷資料的條件超過兩個以上，就可以加上else if來做其他條件判斷

- 在第一個條件使用if
- 其他條件都使用else if來描述
- 最後再以else 來處理都不滿足以上條件(即剩下的可能性)。

其語法如下：

```
if (條件式 1)
{
    敘述區段 1
}
else if (條件式 2)
{
    敘述區段 2
}
...
else if (條件式 N)
{
    敘述區段 N
}
else
{
    敘述區段 N+1
}
```



小測驗-綜合所得稅試算

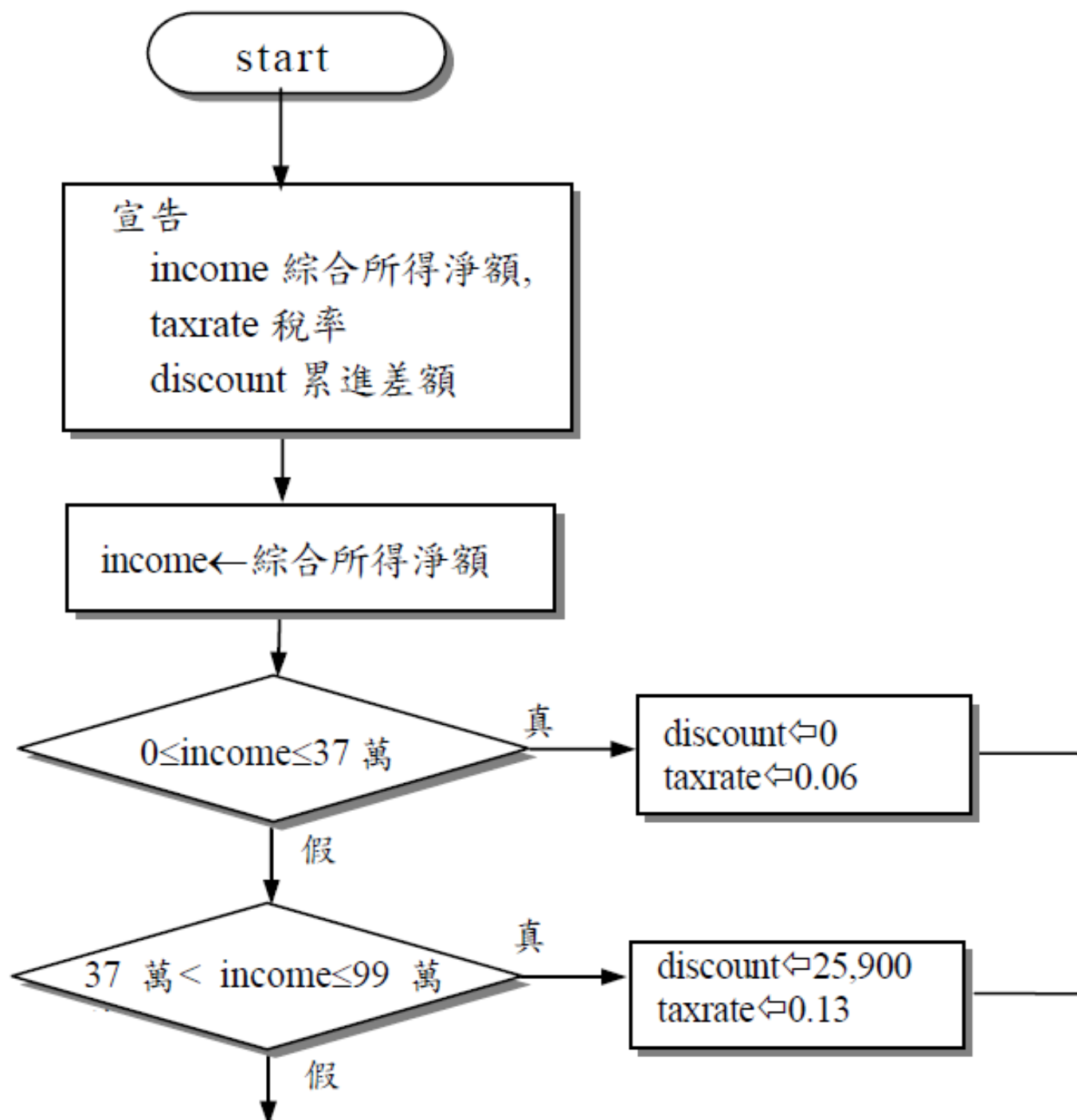


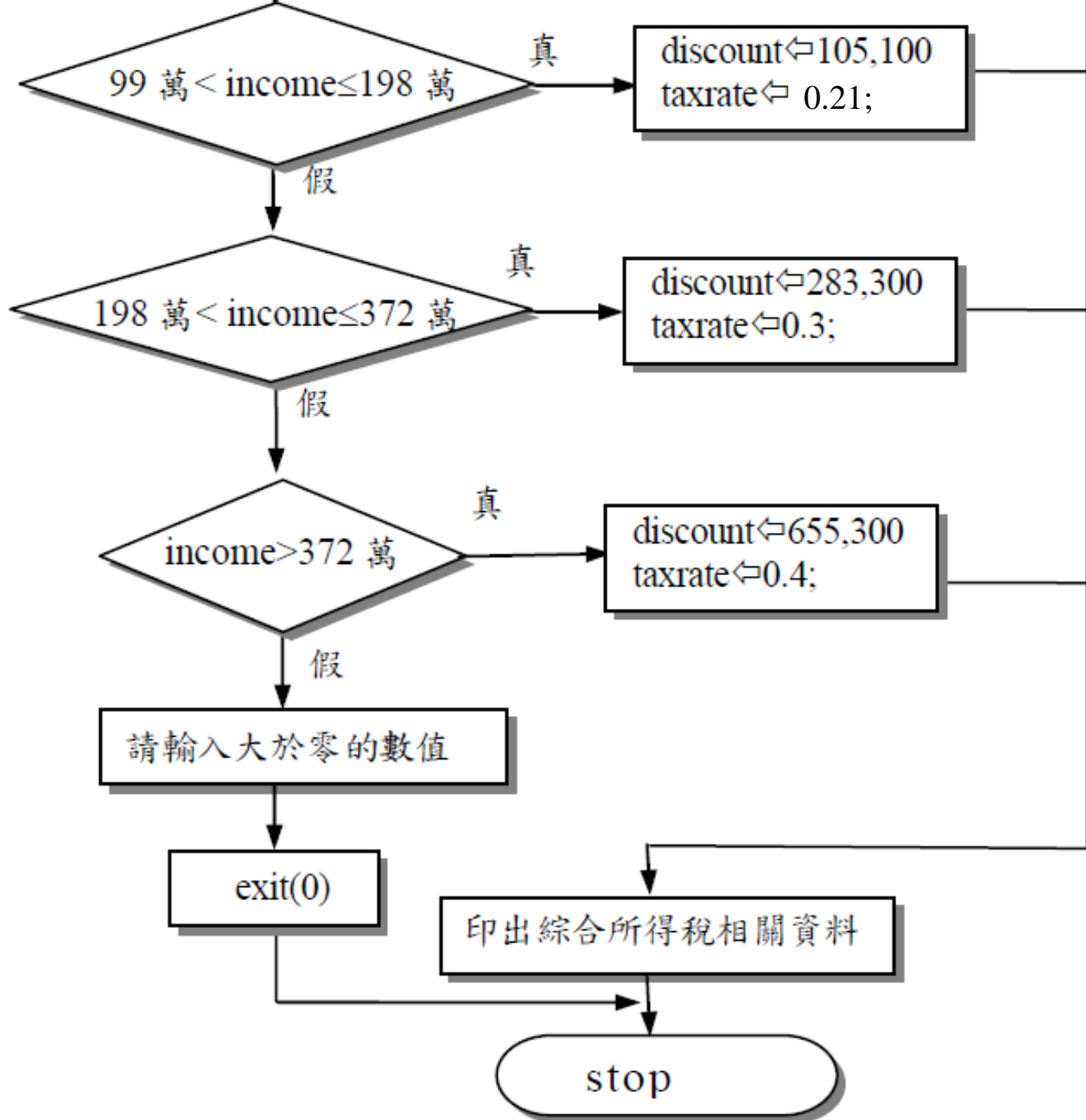
範例：tax.c

請根據下表的綜合所得稅速算公式表，求出使用者輸入綜合所得淨額後，即印出稅率、稅金、累進差額還有今年應納稅額。

綜合所得稅速算公式：

級別	綜合所得淨額	乘	稅額	減	累進差額	等於	全年應納稅額
1	0~370,000	×	6%	—	0	=	?
2	370,000~990,000	×	13%	—	25,900	=	?
3	990,000~1,980,000	×	21%	—	105,100	=	?
4	1,980,000~3,720,000	×	30%	—	283,300	=	?
5	3,720,000 以上	×	40%	—	655,300	=	?





C:\devC\ch05\tax\tax.exe

請輸入綜合所得淨額：1230000

綜合所得淨額： 1230000 元
稅 額： 21 %

稅 金： 258300 元
累 進 差 額： 105100 元

今年應納稅額： 153199 元

請按任意鍵繼續 . . .

多重條件選擇-switch敘述

如果有一個資料，有兩個以上不同的條件需要做判斷，依不同條件給予不同的執行結果，若使用太多的else if可讀性不高，這時候使用switch最為適當。其語法如下：

```
switch (exp)
```

```
{
```

```
    case 常數 1 :
```

```
        ⋮
```

```
        /* 敘述區段 1 */
```

```
        break ;
```

```
    case 常數 2 :
```

```
        ⋮
```

```
        /* 敘述區段 2 */
```

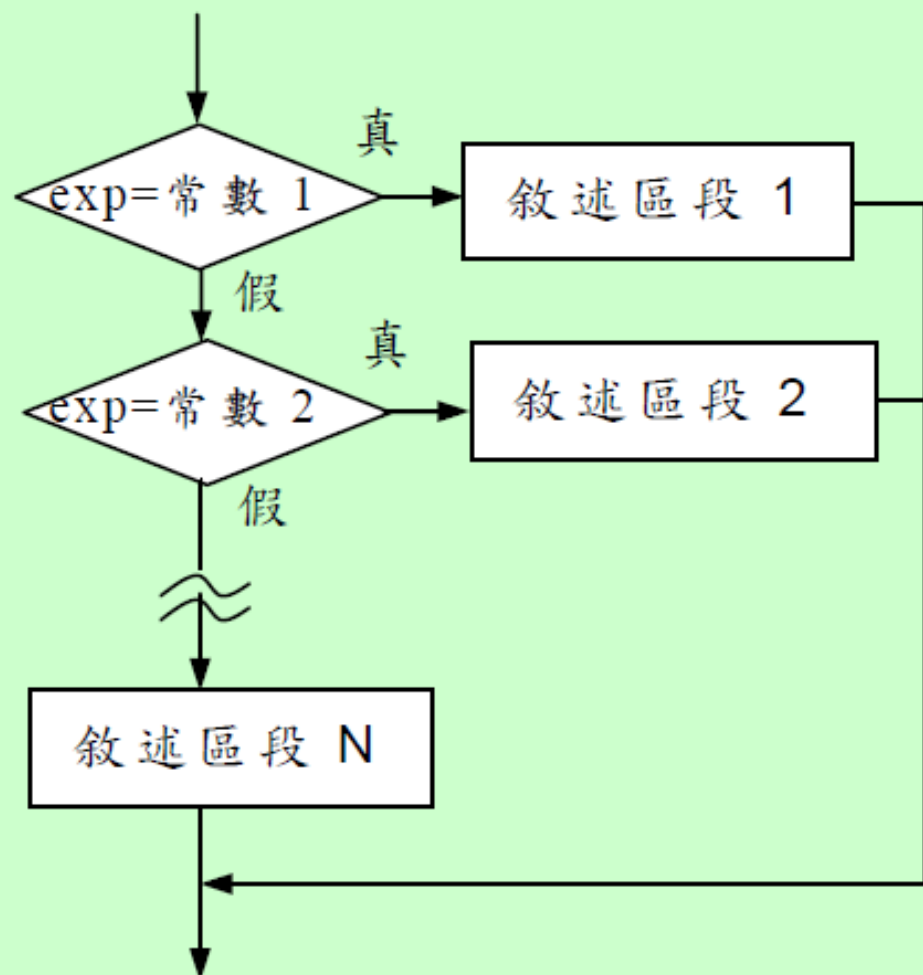
```
        break ;
```

```
        ⋮
```

```
    [default :]
```

```
        /* 敘述區段 N */
```

```
}
```





製作一個簡易的功能表選單。根據下列按鍵顯示對應的訊息：

- ① 按下鍵盤 **A** 鍵顯示："進入新增功能..."。
- ② 按下鍵盤 **D** 鍵顯示："進入刪除功能..."。
- ③ 按下鍵盤 **U** 鍵顯示："進入修改功能..."。
- ④ 按下鍵盤 **Q** 鍵顯示："離開系統^_^--bye"。
- ⑤ 按其他鍵顯示："沒有這個選項 \$#%^&*? "。

執行結果

```
C:\devC\ch05\switch_p1\...
客戶管理維護系統
=====
A : 新增作業
D : 刪除作業
U : 修改作業
Q : 離開作業
=====
請選項 [A,D,U,Q] : A
進入新增作業...
請按任意鍵繼續 . . .
```

問題分析

1. 本範例選項只允許按 **A**、**D**、**U**、**Q** 四個按鍵的大小寫字母，若使用 `(ch=='A' && ch=='a')` 邏輯運算式必須使用 `if-elseif` 敘述，但為了配合使用 `switch` 敘述，必須先使用 `toupper()` 或 `tolower()` 函式，先將輸入的字元 `ch` 一律改成大寫或小寫才能放入 `switch` 敘述中的 `case` 子句，否則大小寫都要使用一個 `case` 來進行比對，如此會增加程式的長度。
2. `toupper()` 和 `tolower()` 函式宣告在 `ctype.h` 標頭檔，因此此標頭檔必須在程式開頭先含入。若欲將輸入的字元轉成大寫，其寫法如下：

```
ch = getchar();  
ch = toupper(ch);  /* 將 ch 字元轉成大寫英文字母 */
```


程式碼

FileName : switch_p1.c

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include <ctype.h>      /* 含入 ctype.h 標頭檔 */
04 #include <conio.h>
05
06 int main(int argc, char *argv[])
07 {
08     char ch;
09     printf("  客戶管理維護系統 \n");
10     printf(" =====\n");
11     printf("      A: 新增作業\n");
12     printf("      D: 刪除作業\n");
13     printf("      U: 修改作業\n");
14     printf("      Q: 離開作業\n");
15     printf(" =====\n");
16     printf("  請選項 [A,D,U,Q]: ");
17     ch = getchar();
18     ch = toupper(ch); /* 將 ch 的值轉成大寫英文字，然後再指定給 ch */
```

```
19     switch(ch)
20     {
21         case 'A':
22             printf("\n 進入新增作業...\n");
23             break ;
24         case 'D':
25             printf("\n 進入刪除作業...\n");
26             break ;
27         case 'U':
28             printf("\n 進入修改作業...\n");
29             break ;
30         case 'Q':
31             printf("\n 離開系統 ! ^_^ ... Bye Bye! \n");
32             break ;
33         default :
34             printf("\n 沒有這個選項$ #%^&*?\n");
35     }
36     printf("\n");
37     system("PAUSE");
38     return 0;
39 }
```

問題：

請問 switch 與 else if 有什麼不同？

C/C++資料結構與程式設計

重複結構

NTU CSIE 張傑帆

for迴圈敘述

C語言提供三種迴圈敘述：for、while、do-while敘述

若迴圈的次數可以預知，for敘述是最好的選擇

若迴圈次數無法確定，則可使用while、do-while敘述來達成

for迴圈

語法

```
for (初始運算式; 條件運算式; 控制運算式)
{
    [敘述區段]
}
```

for(i=1; i<=3; i++)

說明

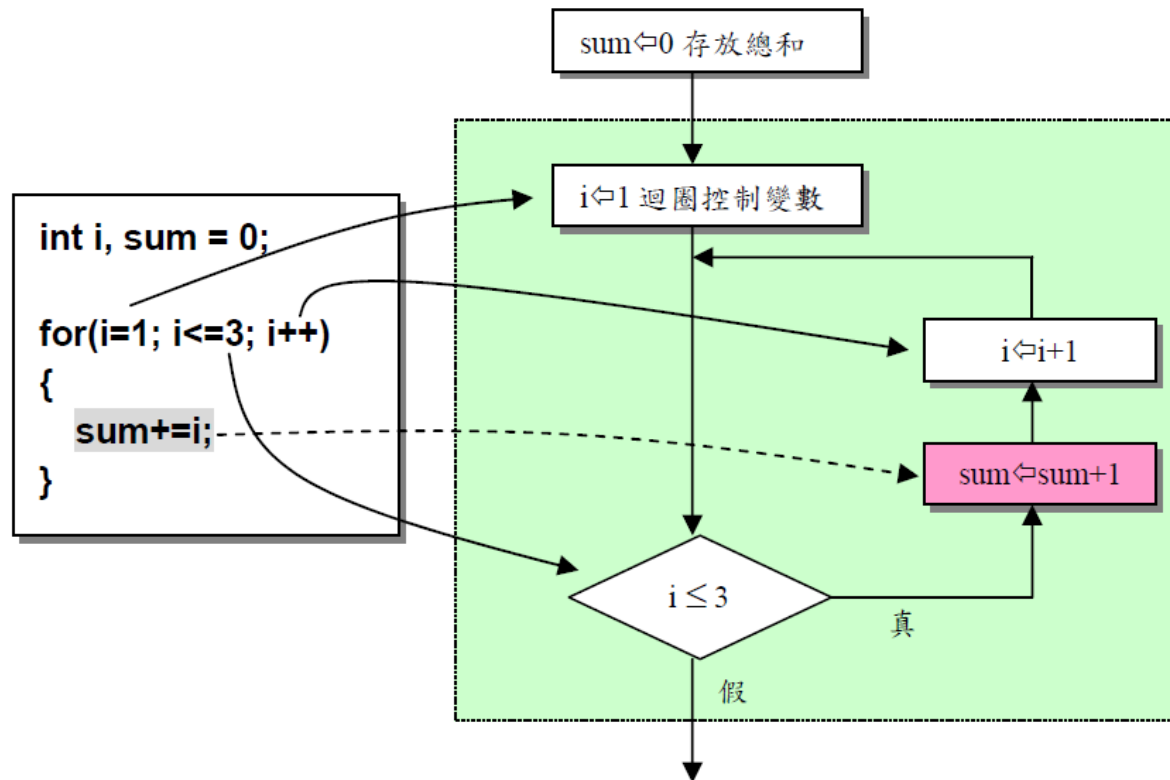
1. 初始運算式

用來設定迴圈控制變數的初始值。這個運算式只有在第一次進入迴圈時才會執行，一直到離開迴圈前都不會再執行。

2. 條件運算式

判斷是否要離開 for 迴圈，此運算式會在每次執行 for 迴圈之前進行判斷。若條件運算式的結果為零(假)表示不滿足條件，則離開 for 迴圈，執行接在 for 迴圈後面的敘述。若條件運算式的結果不為零(真)表示滿足條件，會執行 for 迴圈內的敘述區段一次，再回到 for 敘述中，先執行控制運算式，接著執行條件運算式，再依滿足條件與否決定執行迴圈內的敘述區段或離開迴圈。

5. 下例是利用 for 迴圈，計算 $1+2+3=?$ 總和的流程圖以及執行過程：



程式追蹤

敘述 迴圈	i 值	$i \leq 3$	$sum \leftarrow sum + i$	$i++$
第 1 次	1	$1 \leq 3$ (成立)	$sum \leftarrow 0+1$	2
第 2 次	2	$2 \leq 3$ (成立)	$sum \leftarrow 0+1+2$	3
第 3 次	3	$3 \leq 3$ (成立)	$sum \leftarrow 0+1+2+3$	4
第 4 次	4	$4 \leq 3$ (不成立)	離開迴圈	

前測式迴圈while敘述

由於while迴圈是將條件運算式置於迴圈的最開頭，屬於前測式迴圈，因此若一開始便不滿足條件，迴圈內的敘述區段連一次都不會執行。其語法如下：

語法

```
while (條件運算式)
{
    [敘述區段]
}
```


WHILE迴圈(前測式)

範例：輸入鍵盤按鍵，直到輸入q後程式結束。

```
#include <stdio.h>
#include <conio.h>
int main()
{
    char key=o;

    while(key!='q')
    {
        key=getche();
        printf("\n%c\n", key);
    }
    return o;
}
```


後測式迴圈do-while敘述

do-while迴圈是將條件運算式置於迴圈的最後面，因此一開始就先執行迴圈內的敘述區段一次，接著才檢查接在迴圈最後while的條件運算式，當結果不為零(true)會再執行迴圈內的敘述區段一次，一直到結果為零(false)才會離開迴圈。

要注意的是while(條件運算式)後面必須加上一個「;」號。其語法如下：

語法

```
do
{
    [敘述區段]
}while(條件運算式);
```

DO-WHILE迴圈

範例：輸入鍵盤按鍵，直到輸入q後程式結束。

```
#include <conio.h>
int main()
{
    char key;
    do
    {
        key=getche();
        printf("\n%c\n", key);
    }while(key!='q');

    return 0;
}
```

break與continue敘述

break與continue敘述可在for、while、do-while迴圈內的敘述區段中使用。

當執行到break敘述時，會馬上如圖一跳離迴圈繼續執行接在迴圈後面的敘述。

執行到continue敘述時，會如圖二忽略接在continue後面的敘述區段，直接返回到迴圈的條件式，判斷是否繼續執行迴圈：

while (條件式)

{

[敘述區段 1]

break;

[敘述區段 2]

}

[敘述區段 3]

圖一

while (條件式)

{

[敘述區段 1]

continue ;

[敘述區段 2]

}

[敘述區段 3]

圖二

課堂練習-猜數字遊戲

範例：guess.c

製作猜數字遊戲。程式會先產生 1-99 之間的亂數當作被猜的數字，執行過程中會提示您所猜的數字應該再大一點或再小一點，並縮小猜的範圍，若猜到正確的數字會印出 "賓果!猜對了,答案是 xx" 及 "總共猜了 n 次"。



執行結果

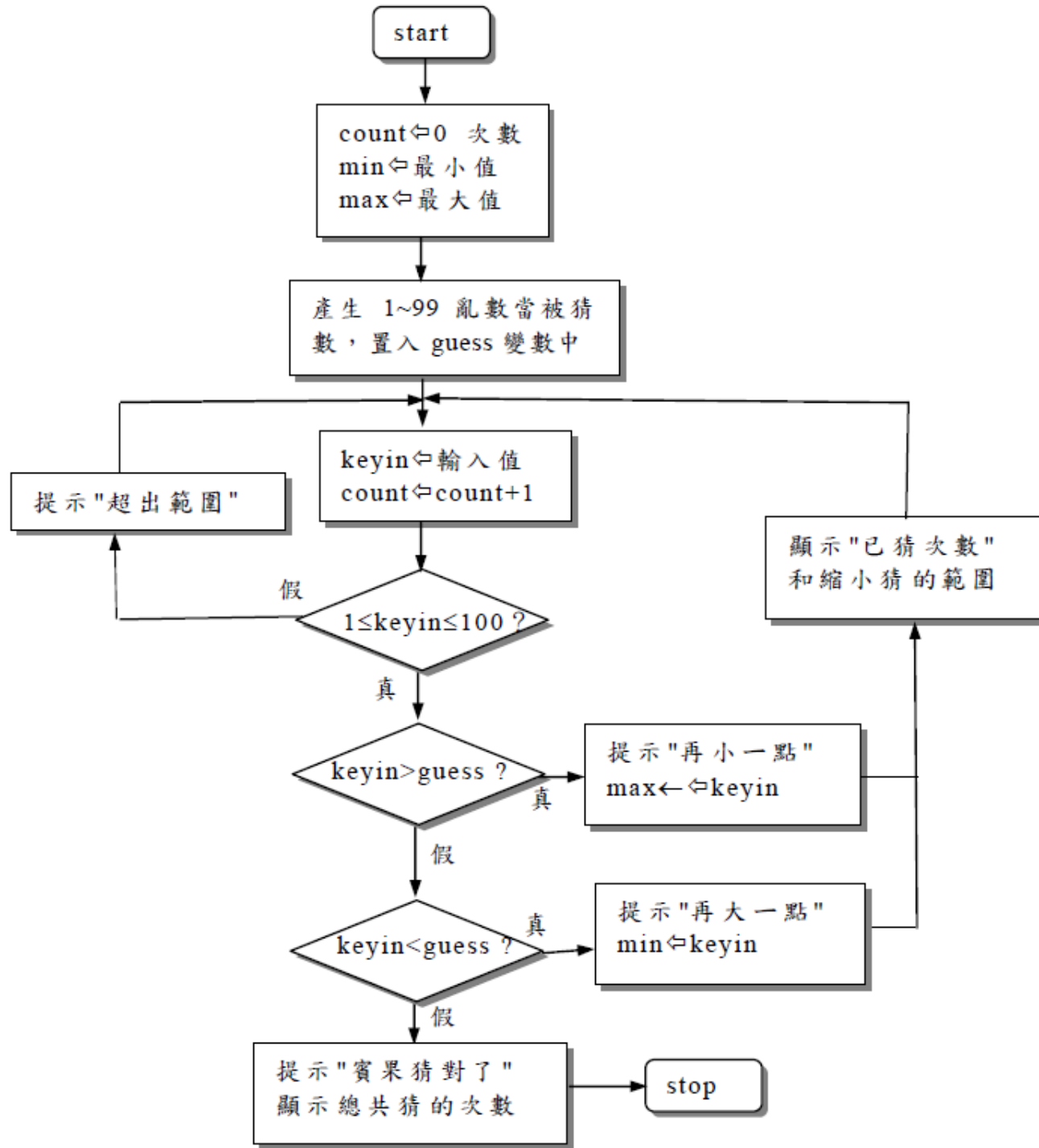
```
=====  
猜數字遊戲=====  
猜數子範圍 0 < ? < 100 : 50  
再小一點!! 您猜了 1 次  
猜數子範圍 0 < ? < 50 : 40  
再大一點!! 您猜了 2 次  
猜數子範圍 40 < ? < 50 : 45  
賓果! 猜對了, 答案是 45  
總共猜了 3 次!  
請按任意鍵繼續 . . .
```


問題分析

1. 若在程式直接設定初值當作被猜數字，每次執行所猜的數字會一樣，不具彈性，為了使每次執行的猜數字能隨機，請在程式最開頭先含入 `stdlib.h` 及 `time.h` 標頭檔，接著再使用下面敘述來產生 1~99 之間的亂數。

```
srand((unsigned)time(NULL)); /*亂數種子器*/  
guess=rand()%99+1 /*產生 1-99 之間的亂數並指定給整數 guess*/
```

2. 本例猜到正確的數字才結束程式的執行，因此猜數字是否正確的程式必須撰寫在無窮迴圈中，若要離開無窮迴圈可按  +  鍵結束程式的執行。



程式碼 FileName : guess.c

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include <time.h>
04 int main(int argc, char *argv[])
05 {
06     int keyin, guess, count, min, max;
07     count=0;
08     min=0;
09     max=100;
10     srand((unsigned)time(NULL));
11     guess=rand()%99+1;
12     printf("=====  
猜數字遊戲  
===== : \n\n");
13     do
14     {
15         printf("猜數字範圍 %d < ? < %d : ", min, max);
16         scanf("%d", &keyin);
17         count++;
18         if(keyin>=1 && keyin<100)
19         {
20             if(keyin==guess)
21             {
```

```
22         printf("賓果！猜對了，答案是 %d\n", guess);
23         break;
24     }
25     else if(keyin>guess)
26     {
27         max=keyin; /*將目前輸入的數字 keyin 指定給 max*/
28         printf("再小一點!!");
29     }
30     else if(keyin<guess)
31     {
32         min=keyin; /*將目前輸入的數字 keyin 指定給 min*/
33         printf("再大一點!!");
34     }
35     printf(" 您猜了 %d 次\n\n", count);
36 }
37 else
38 {
39     printf("請輸入提示範圍內的數字!\n");
40 }
41 }while(1);    /*無窮迴圈*/
42 printf("\n 總共猜了 %d 次!\n\n", count);
43 system("PAUSE");
44 return 0;
45 }
```

C/C++資料結構與程式設計

陣列

NTU CSIE 張傑帆

陣列簡介

設計程式時，輸入的資料、中間結果或最後結果都需用變數存放資料多時，變數命名很重要。不同性質資料，變數名稱必須不一樣。

為方便變數名稱命名，C 對同性質資料提供陣列來存放。

將陣列想像是一組經過編號的變數，一個變數相當於一個車廂號碼的話，那麼一個陣列就是一列火車，而每列火車的車廂長度即陣列的大小(長度)。

欲存取陣列中某個資料內容，只要知道該變數在陣列中是第幾個元素相當於車廂號碼，即可取得該變數的內容相當於該車廂內的東西，寫法是在該陣列名稱後面小括號內填入該變數所對應的數字編號。

陣列的宣告與初值設定

陣列和一般變數一樣，必須先經過宣告，編譯器才能根據所宣告資料的型別和大小來配置合適的記憶體空間。

陣列經過宣告後，便可知道該陣列中到底含有多少個陣列元素，以及透過所宣告的資料型別知道每個陣列元素佔用的記憶體大小。

如何宣告陣列

陣列的宣告方式是以資料型別開頭，其後緊跟著陣列名稱，在陣列名稱後面加上一對 `[]` 中括號所組成，中括號內的註標或稱索引值，代表該陣列的大小(或稱個數)。其語法如下：

資料型別 陣列名稱[索引];

譬如：下面敘述宣告一個陣列名為 `myAry` 的整數陣列：

```
int myAry[10];
```

上面敘述經過宣告後的陣列可得知下列事項：

1. `myAry` 是一個整數陣列，該陣列含有十個陣列元素依序為：`myAry[0]` ~ `myAry[9]`。
2. 索引值由 0 開始一直到 9，索引值必須是整數常值或整數變數或整數運算式。
3. 每個陣列元素相當於一個變數名稱，所存放的資料都是整數。

4. Dev C++每個整數資料是使用 4 Bytes 來存放，若一個記憶體位址只允許存放一個 Byte 資料時，那麼一個整數資料就需佔用 4 個記憶體位址，因此 myAry 陣列中有 10 個陣列元素共需使用 $10 \times 4 = 40$ 個記憶體位址來存放資料，編譯器在編譯時會在記憶體中找出 40 個連續未使用的記憶體位址供此陣列使用，下圖即是陣列元素和對應記憶體位址的關係圖：

```
int myAry[10];
```

記憶體位址	資料	陣列元素
1000~1003		myAry[0]
1004~1007		myAry[1]
1008~1011		myAry[2]
1012~1015		myAry[3]
1016~1019		myAry[4]
1020~1023		myAry[5]
1024~1027		myAry[6]
1028~1031		myAry[7]
1032~1035		myAry[8]
1036~1039		myAry[9]

以下為陣列各資料型別的宣告方式：

```
int score[5];    /* 宣告 score 為整數型別陣列，陣列元素為 score[0] ~ score[4]*/  
char name[4];   /* 宣告 name 為字元型別陣列，陣列元素為 name[0] ~ name[3] */  
byte qty[10];   /* 宣告 qty 為位元組型別陣列，陣列元素為 qty[0] ~ qty[9] */  
float price[10]; /* 宣告 price 為浮點數型別陣列，陣列元素為 price[0]~price[9] */  
double sum[10]; /* 宣告 sum 為倍精確數型別陣列，陣列元素為 sum[0]~sum[9] */
```


C語言並未提供字串資料型別，而是以字元陣列方式來儲存字串。

字元和字串間的差異在當字串存入字元陣列會自動在字元陣列的最後面插入一個『\0』當結束字元，以方便在讀取字元陣列中的字串時，碰到結束字元，便結束讀取動作。

譬如：欲將 "test" 字串置入名稱為ary的字元陣列中，由於字串實際長度為4，必須再多出一個位置存放結束字元。所以，字元陣列ary的大小(長度)應設為5，其宣告方式如下：

```
char ary[5];
```

ary[0]	't'
ary[1]	'e'
ary[2]	's'
ary[3]	't'
ary[4]	'\0'

如何設定陣列的初始值

陣列經過宣告完畢，再使用指定運算子(=)來逐一指定陣列元素的初值。其寫法如下：

Case 1：數值資料

```
int ary[4];  
ary[0] = 10;   ary[1] = 20;   ary[2] = 30;   ary[3] = 40;
```

宣告 `ary` 是一個整數陣列，它含有四個陣列元素 `ary[0]~ary[3]`，每個陣列元素內所存放的是整數資料。第 2 行使用指定運算子(=)來設定四個陣列元素的初值。

Case 2：字元陣列資料

```
char name[4];  
name[0] = 'T';   name[1] = 'o'; name[2] = 'm';
```

宣告 `name` 是一個字元陣列，它含有四個陣列元素 `name[0]~name[3]`，每個陣列元素內所存放的是字元資料。上面敘述設定 `name[0]~name[2]` 的初值，在 C 語言中字元陣列也可以用來存放字串，若以字串方式存入陣列時 `name[3]` 會自動設為 `'\0'`。

方式 2 宣告同時設定初值

在宣告陣列的同時一起指定陣列元素的初值。方法就是緊接在陣列宣告敘述 [] 中括號後面，加上指定運算子(=)，其後再使用 { } 左右大括號，將陣列初始值設定在裡面即可，撰寫時要注意各陣列元素的初值之間必須使用逗號隔開，為避免陣列大小設定錯誤可省略不寫，但各陣列元素的初值必須在使用前先設定後才能存取。其語法如下：

語法

資料型別 陣列名稱 [陣列大小] = {索引 0 的初值, 索引 1 的初值, ...索引 n-1 的初值};

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

如何存取陣列的資料

由於陣列內的註標可以是整數常數、整數變數或是運算式結果為整數的資料。

連續存取陣列中的陣列元素時，可透過迴圈來輸入(存)及輸出(取)資料，其好處是可縮短程式的長度以及提高可讀性。

Case 1：程式中直接設定整數陣列初值

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

或由鍵盤輸入設定整數陣列初值

```
for(i=0; i<4; i++)  
    scanf("%d", ary[i]);
```

Case 2：使用迴圈讀取整數陣列元素的內容

```
for(i=0; i<4; i++)  
    printf("%d \n", ary[i]);
```


字元陣列

Case 3：由鍵盤輸入字串到字元陣列方法

```
char name[10]; /*限 9 個字元，保留一個字元放置結束字元*/  
printf("設定字元陣列 name 中的字串 :");  
gets(name); //字元陣列可用gets()
```

Case 4：延續 case3 顯示字元陣列各陣列元素的內容

```
printf("%s \n ",name);  
puts(name); /* name 為字元陣列 */  
/* 字元陣列可使用 strlen 來取得字元的個數 */  
for(i=0; i<strlen(name); i++)  
{  
    printf(" name[%d] = %c \n", i, name[i]);  
}
```

實機練習-大樂透開獎



範例：biglottery.c

撰寫一個大樂透電腦自動選號程式。程式執行時會以亂數的方式顯示 1~49 之間七個不重複的大樂透號碼。

執行結果

```
C:\devC\ch07\biglottery\...  
本期大樂透 電腦選號 號碼如下：  
35 12 27 20 8 30  
特別號： 17  
請按任意鍵繼續 . . .
```

程式碼 FileName : biglottery.c

```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 int main(int argc, char *argv[])
05 {
06     int lot[49];          /* 陣列元素為 lot[0]~lot[48]  */
07     int choose[7];        /* 陣列元素為 choose[0]~choose[6]  */
08     int min=1, max=49, num=7;
09     int max_dim, choice;
10     int i;
11     max_dim=max-min+1;
12     for(i=0;i<max_dim;i++)
13     {
14         lot[i]=min+i;
15     }
16     srand((unsigned)time(NULL));    /* 亂數種子器 */
```

```
17     for(i=0;i<num;i++)
18     {
19         choice=rand()%max_dim;          /* 產生亂數 */
20         choose[i]=lot[choice];          /* 隨機取一個號碼 */
21         /* 陣列最後一個號碼移到目前選取號碼陣列元素的位置 */
22         lot[choice]=lot[max_dim-1];
23         max_dim--;
24     }
25     printf("\n 本期大樂透 電腦選號 號碼如下:\n\n");
26     for( i=0; i<num-1; i++)              /* 印出大樂透的 6 個號碼 */
27     {
28         printf(" %d", choose[i]);
29     }
30     printf("\n\n 特別號: %d \n", choose[6]);    /* 印出大樂透的特別號 */
31     printf("\n\n");
32     system("PAUSE");
33     return 0;
34 }
```


多維陣列

二維陣列的宣告與初始值的設定

二維陣列含有兩個註標(索引)，維度為2。

每個索引以 [] 中括號括住，其語法如下。

欲連續存取二維陣列內的資料時，必須使用雙層的for迴圈才能達成：

資料型別 陣列名稱[索引₁, 索引₂];

譬如：欲建立一個 3x4 的整數陣列 a(即 3 個水平列和 4 個垂直行)的陣列，其宣告方式如下：

```
int a[3][4];      /* 宣告 a 為 3x4 的二維陣列，資料型別為 int */
```

下圖 a[3][4] 是 3x4 二維陣列，第一個註標(或稱維度)為 3 代表水平列(Row)有三列由第 0~2 列，第二個註標為 4 代表垂直行(Column)有四行由第 0~3 行。此陣列共有 12 個陣列元素，其示意圖如下：

	第 0 行	第 1 行	第 2 行	第 3 行
第 0 列	a[0][0]	a[0][1]	a[0][2]	a[0][3]
第 1 列	a[1][0]	a[1][1]	a[1][2]	a[1][3]
第 2 列	a[2][0]	a[2][1]	a[2][2]	a[2][3]

二維陣列和一維陣列一樣，經過宣告後便可如下表將 3x4 的二維陣列 a 逐一設定初值：

	第 0 行	第 1 行	第 2 行	第 3 行
第 0 列	a[0][0]=0	a[0][1]=1	a[0][2]=2	a[0][3]=3
第 1 列	a[1][0]=4	a[1][1]=5	a[1][2]=6	a[1][3]=7
第 2 列	a[2][0]=8	a[2][1]=9	a[2][2]=10	a[2][3]=11

初始化

【方式一】

```
int a[3][4] ;  
a[0][0]=0 ; a[0][1]=1 ; a[0][2]=2 ; a[0][3]=3 ;  
a[1][0]=4 ; a[1][1]=5 ; a[1][2]=6 ; a[1][3]=7 ;  
a[2][0]=8 ; a[2][1]=9 ; a[2][2]=10 ; a[2][3]=11 ;
```

【方式二】

```
int n[3][4]={ {0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11} } ;
```

第 0 列

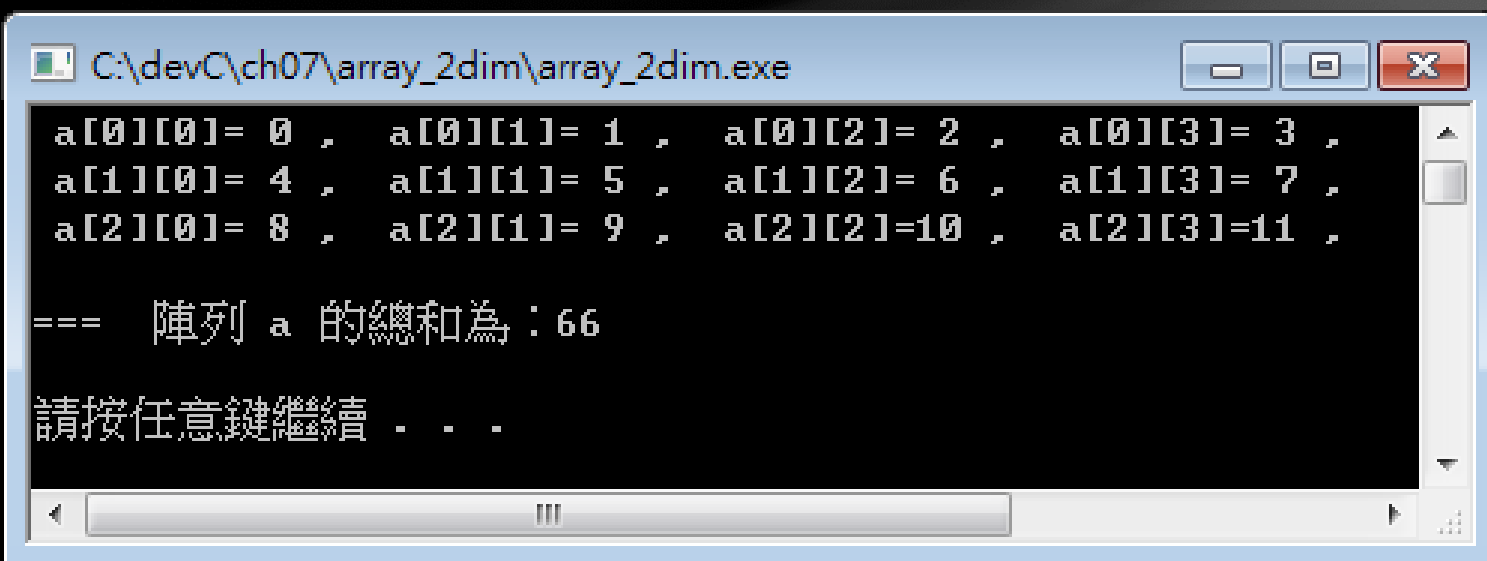
第 1 列

第 2 列

如何存取二維陣列的資料

下面範例是透過巢狀迴圈來取得 3×4 二維陣列所設定的初值，並累加各 $a[i][j]$ 陣列元素的總和。

下表採逐列(Row-Majored)方式取得陣列初值，因此本例雙層巢狀迴圈的外層迴圈是使用整數變數 i ($0 \sim 2$)，內層迴圈是使用整數變數 j ($0 \sim 3$)：



```
C:\devC\ch07\array_2dim\array_2dim.exe


a[0][0]= 0 ,   a[0][1]= 1 ,   a[0][2]= 2 ,   a[0][3]= 3 ,
a[1][0]= 4 ,   a[1][1]= 5 ,   a[1][2]= 6 ,   a[1][3]= 7 ,
a[2][0]= 8 ,   a[2][1]= 9 ,   a[2][2]=10 ,   a[2][3]=11 ,

=== 陣列 a 的總和為：66

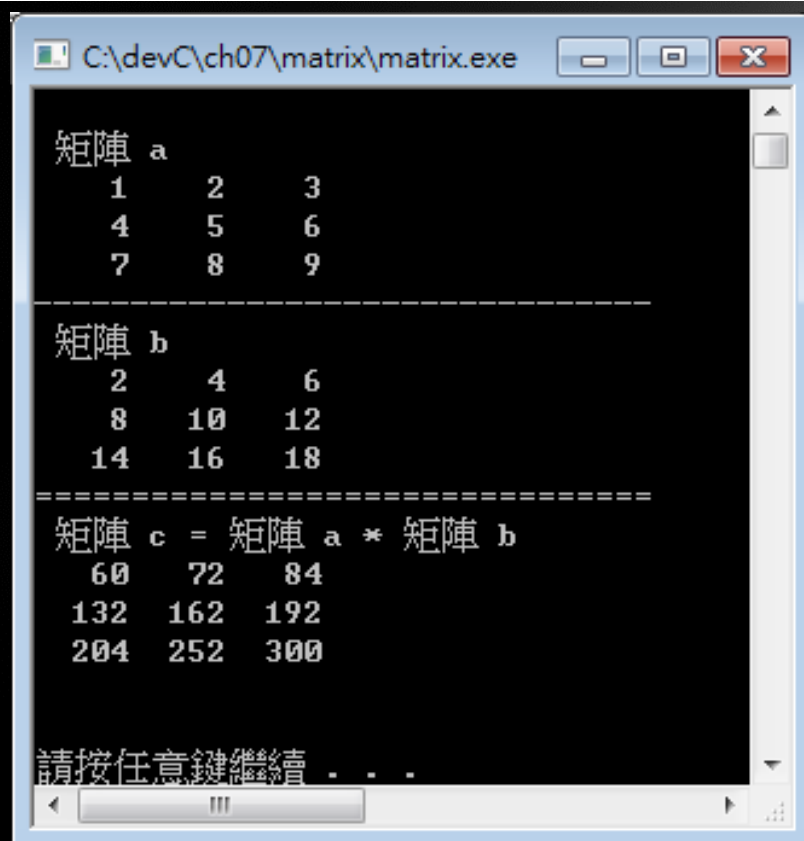
請按任意鍵繼續 . . .
```

```
04 int main(int argc, char *argv[])
05 {
06     int a[3][4]={ {0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}
07     int sum = 0 ;
08     int i, j;
09     // 第一維陣列維度為 3，i=3
10     for (i=0 ; i<3 ; i++)
11     {
12         // 第二維陣列維度為 4，j=4
13         for (j=0 ; j<4 ; j++)
14         {
15             printf(" a[%d][%d]=%2d , ", i, j, a[i][j]);
16             sum += a[i][j];
17         }
18         printf("\n");    /* 將游標移到下一行 */
19     }
20     printf("\n=== 陣列 a 的總和為：%d \n\n", sum);
```

實機練習-矩陣相乘

 範例 : matrix.c

宣告三個二維陣列來模擬數學矩陣相乘，矩陣 a 和矩陣 b 相乘，所得的乘積置入矩陣 c。



```
C:\devC\ch07\matrix\matrix.exe

矩陣 a
  1   2   3
  4   5   6
  7   8   9
-----
矩陣 b
  2   4   6
  8  10  12
 14  16  18
=====
矩陣 c = 矩陣 a * 矩陣 b
 60  72  84
132 162 192
204 252 300

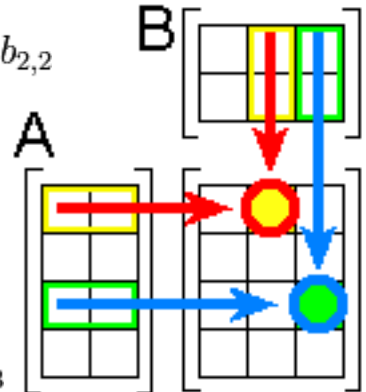
請按任意鍵繼續 . . .
```

矩陣相乘的公式如下：

$$(AB)_{1,2} = \sum_{r=1}^2 a_{1,r}b_{r,2} = a_{1,1}b_{1,2} + a_{1,2}b_{2,2}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix} = \begin{bmatrix} 60 & 72 & 84 \\ 132 & 162 & 192 \\ 204 & 252 & 300 \end{bmatrix}$$

$$(AB)_{3,3} = \sum_{r=1}^2 a_{3,r}b_{r,3} = a_{3,1}b_{1,3} + a_{3,2}b_{2,3}$$



$$\begin{bmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \\ a[2][0] & a[2][1] & a[2][2] \end{bmatrix} \times \begin{bmatrix} b[0][0] & b[0][1] & b[0][2] \\ b[1][0] & b[1][1] & b[1][2] \\ b[2][0] & b[2][1] & b[2][2] \end{bmatrix} = \begin{bmatrix} c[0][0] & c[0][1] & c[0][2] \\ c[1][0] & c[1][1] & c[1][2] \\ c[2][0] & c[2][1] & c[2][2] \end{bmatrix}$$

$$c[i][j] += a[i][k] \times b[k][j]$$

$$c[0][0] = a[0][0] * b[0][0] + a[0][1] * b[1][0] + a[0][2] * b[2][0]$$

$$c[0][1] = a[0][0] * b[0][1] + a[0][1] * b[1][1] + a[0][2] * b[2][1]$$

$$c[0][2] = a[0][0] * b[0][2] + a[0][1] * b[1][2] + a[0][2] * b[2][2]$$

$$c[1][0] = a[1][0] * b[0][0] + a[1][1] * b[1][0] + a[1][2] * b[2][0]$$

$$c[1][1] = a[1][0] * b[0][1] + a[1][1] * b[1][1] + a[1][2] * b[2][1]$$

$$c[1][2] = a[1][0] * b[0][2] + a[1][1] * b[1][2] + a[1][2] * b[2][2]$$

$$c[2][0] = a[2][0] * b[0][0] + a[2][1] * b[1][0] + a[2][2] * b[2][0]$$

$$c[2][1] = a[2][0] * b[0][1] + a[2][1] * b[1][1] + a[2][2] * b[2][1]$$

$$c[2][2] = a[2][0] * b[0][2] + a[2][1] * b[1][2] + a[2][2] * b[2][2]$$


```
06     int a[3][3]={ {1,2,3}, {4,5,6}, {7,8,9} };
07     int b[3][3]={ {2,4,6}, {8,10,12}, {14,16,18} };
08     int c[3][3];
09     int i, j, k;
10     printf("\n 矩陣 a \n");
11     for(i=0;i<=2;i++) /* 印出 a 矩陣(a 陣列) */
12     {
13         for(j=0;j<=2;j++)
14         {
15             printf("%5d", a[i][j]);
16         }
17         printf("\n");
18     }
19     printf("-----");
20     printf("\n 矩陣 b \n");
21     for(i=0;i<=2;i++) /* 印出 b 矩陣(b 陣列) */
22     {
23         for(j=0;j<=2;j++)
24         {
25             printf("%5d", b[i][j]);
26         }
27         printf("\n");
28     }
```

C++資料結構與程式設計

函式

NTU CSIE 張傑帆

函式簡介

- 在撰寫一個大的應用程式時，常會碰到一些具有某些功能的程式片段在程式中多處重複出現
- 若不加以處理會導致程式冗長不具結構化且不易維護，因此我們將這些小功能的程式片段獨立出來撰寫成函式(Function)，所以函式就是指具有某種特定功能的一段程式碼
- 程式中使用函式最主要是用來
 - 精減程式碼
 - 重覆使用
 - 模組化

主程式 - main()

⋮ 計算平均

⋮ 計算平均

計算平均

⋮

myfun

⋮

計算平均

主程式 - main()

呼叫 myfun ;

呼叫 myfun ;

呼叫 myfun ;

內建函式

`main()`本身就是一個函式我們稱為主函式或主程式，
`printf()/scanf()`也是函式，這些都是系統廠商提供的內建函式

我們以內建函式`pow()`計算某數的某次方為例：

語法：`double c = double pow(double a, double b);`

標頭檔：`#include <math.h>`

說明：傳回 a^b 值給 `c`。

自定函式的定義與呼叫

如何宣告與定義自定函式

若您在設計程式時，需要使用到一個小功能，這個功能在內建函式中找不到，那麼只好在程式中自己定義一個具有此小功能的函式，我們稱為「使用者自定函式」(User Defined Function)簡稱「自定函式」。

由於使用者自定函式是無中生有的，因此必須在使用前先定義該函式，此即為該函式的主體。

函式主體的位置通常撰寫在 `#include` 和 `main()` 主函式的之間，即 `main()` 主函式的前面，也允許放在 `main()` 主函式的後面，若使用後者就必須在 `main()` 主函式前面先宣告函式的原型，以告知編譯器此自定函式在程式中有定義。

方法一

宣告函式 1 `int func (int, int);`

主程式-main()
⋮

函式 1

```
int func ( int var1, int var2 )  
{  
    int var3 = var1+var2;  
    return var3;  
}
```

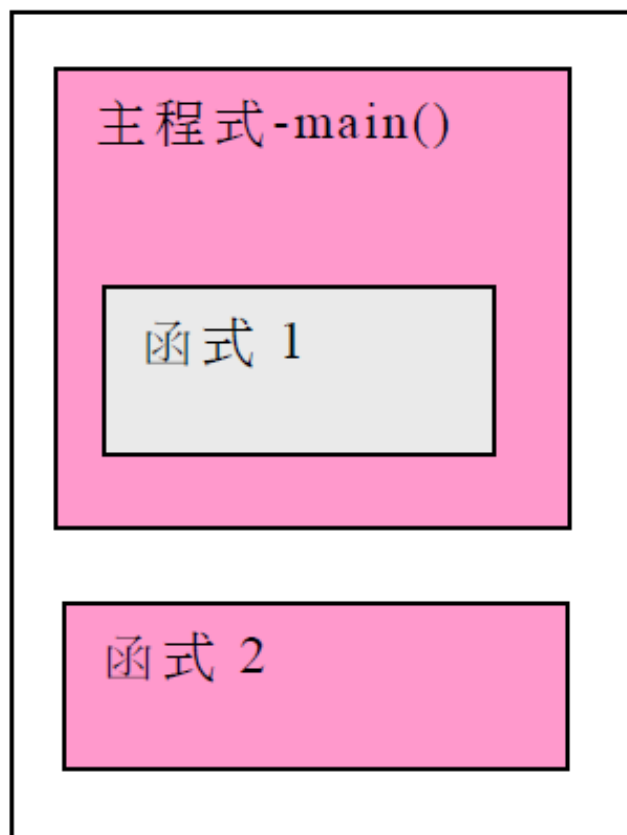
方法二

函式 1

```
int func ( int var1, int var2 )  
{  
    int var3 = var1+var2;  
    return var3;  
}
```

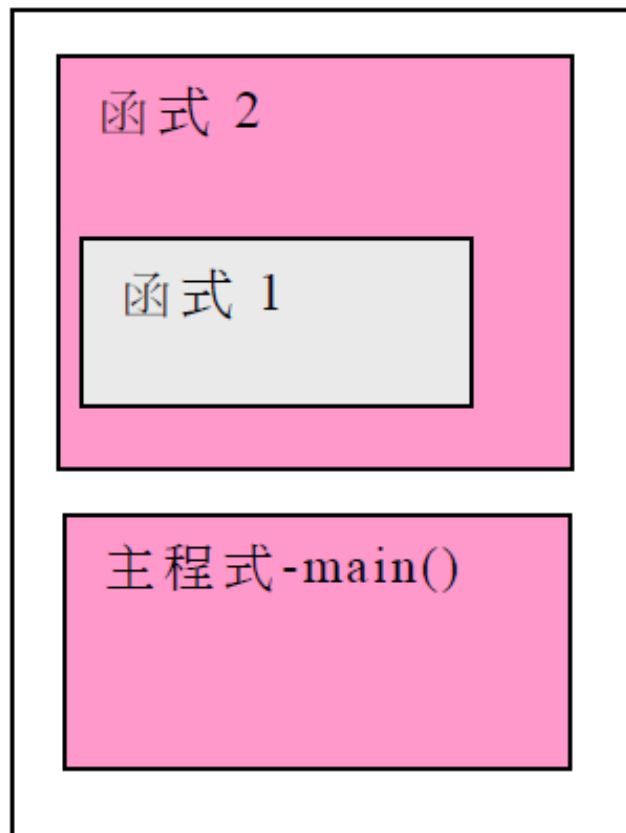
主程式-main()
⋮

5. 任何函式主體不可在主程式內



錯誤

6. 函式主體內不可在其他函式主體內



錯誤

一. 函式原型宣告語法

語法

[傳回值型別] 函式名稱 (資料型別 1, 資料型別 2, ...資料型別 n 引數 n);

說明

1. double cal(int);

宣告 cal 函式，此函式可傳回 double 浮點數資料型別的資料，呼叫時必須傳入一個 int 整數資料型別的資料。

2. int add(int, int);

宣告 add 函式，此函式可傳回 int 整數資料型別的資料，呼叫時必須傳入兩個 int 整數資料型別的資料。

二. 函式定義語法

語法

```
[傳回值型別] 函式名稱 (資料型別 1 引數 1, 資料型別 2 引數 2, ... 資料型別 n 引數 n)
{
    程式區段 ;
    return 運算式 ;
}
```

```
int func ( int var1, int var2)
{
    int var3 = var1+var2;
    return var3;
}
```

說明

1. 傳回值型別

函式的傳回值可以是數值(byte, int, float, ...)、字元、字串、指標...等資料型別。若函式不傳回任何資料，必須將傳回值型別設為 void。

2. 函式名稱

在同一個程式中自定函式的名稱不可以重複定義；函式名稱也不能和系統所提供的內建函式的名稱相同。

函式間傳遞陣列

在設計程式時，時常碰到需要將整個陣列傳給被呼叫的函式，在被呼叫函式內整個陣列經過計算處理完畢，再將整個陣列傳回給原呼叫函式的陣列。其寫法是在原呼叫函式的實引數中，只需寫上陣列名稱即可，後面不必加上[]中括號。

在被呼叫函式的虛引數內對應的虛引數必須是一個陣列名稱，其後面必須加上[]中括號，虛引數內陣列名稱前加上該陣列名稱的資料型別。

```

#include<stdio.h>
void sort(int [ ]);          /* 函式原型宣告 */
int _tmain(int argc, _TCHAR* argv[]) /* 原呼叫函式 */
{
    int myArray[5];
    ...
    sort(myArray);          /* 呼叫函式敘述 */
    ...
}
void sort(int tArray[ ])    /* 被呼叫函式 */
{
    ...
}

```


- 當寫到2維陣列(或多維陣列)時，赫然發現事情已經不是如上所想的簡單了，首先，這樣的陣列傳遞已經無法通過compile了

```
1  void timesTwo( int arr[][] );  
2  int main()  
3  {  
4      int arr2[2][4]={ 1,3,5,7,2,4,6,8 };  
5      ...  
6  }
```

- 如果這樣寫就可以通過compile，但把維度寫死一點都不彈性

```
1  void timesTwo( int arr[][4] ) {};
```

- 所以要換成這樣的寫法較佳

```
1  void timesTwo( int arr[2][4] ) {};
```

- 或這樣

```
1  void timesThree( int (*arr)[4] ) {};
```


小練習

- 將上面的矩陣相乘程式，列印二維陣列及矩陣相乘本體轉換成函式
- `void Print2dAry(int[][3]);`
- `void MatrixMul2dAry(int[][3]);`

遞迴函式

若函式中有一行敘述再呼叫自己本身的函式，即稱為「遞迴函式」(Recursive function)。

使用遞迴函式執行時，會不斷的呼叫自己本身之函式，這樣遞迴函式會形成無窮迴圈，因此必須在遞迴函式內，設定條件來結束該函式的執行，如此當滿足條件時，才結束呼叫自己本身，這樣才能離開此函式。

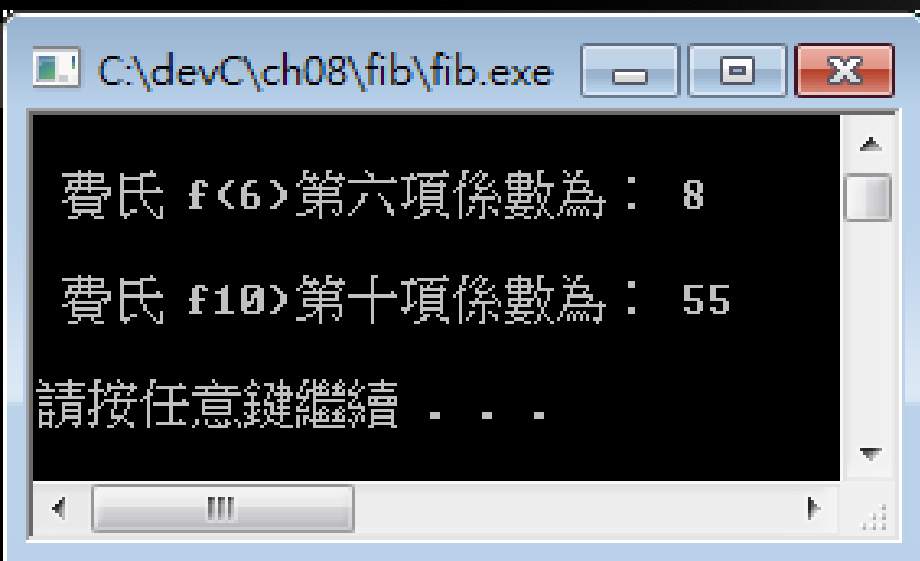
遞迴常使用在具有規則性的程式設計，例如：求最大公因數、排列、組合、階乘、費氏數列。

實機練習-遞迴 費氏數列



範例：fib.c

利用遞迴來算出費波南希級數(簡稱費式級數)的係數。數列最開頭即第一項和第二項為 1，第三項以後的係數是前兩項係數的和。該級數的係數依序為：1，1，2，3，5，8，13 …其他以此類推。



```
C:\devC\ch08\fib\fib.exe
費氏 f(6)第六項係數為: 8
費氏 f(10)第十項係數為: 55
請按任意鍵繼續 . . .
```

$$f(n) = \begin{cases} 1 & n=1,2 \\ f(n-2) + f(n-1) & n>2 \end{cases}$$

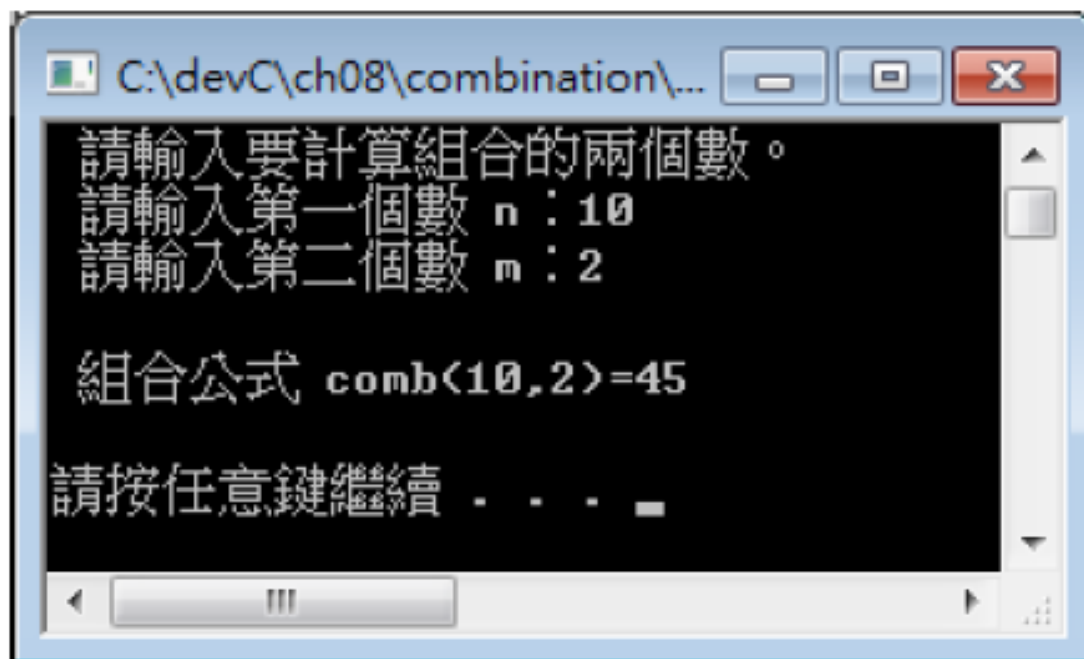
```
04 int fib(int);
05
06 int main(int argc, char *argv[])
07 {
08     printf("\n 費氏 f(6)第六項係數為： %d\n", fib(6));
09     printf("\n 費氏 f(10)第十項係數為： %d\n\n", fib(10));
10     system("PAUSE");
11     return 0;
12 }
13 int fib(int n)
14 {
15     if(n==1 || n==2)                /* 若 n 為 1 或 2，則傳回 1 */
16     {
17         return 1;                    /* f(1)=1    f(2)=1    */
18     }
19     else                             /* 若 n>2，則呼叫函式自己本身 */
20     {
21         return fib(n-1)+fib(n-2);    /* 呼叫函式自己本身 */
22     }
23 }
```


回家作業 數學組合公式求法

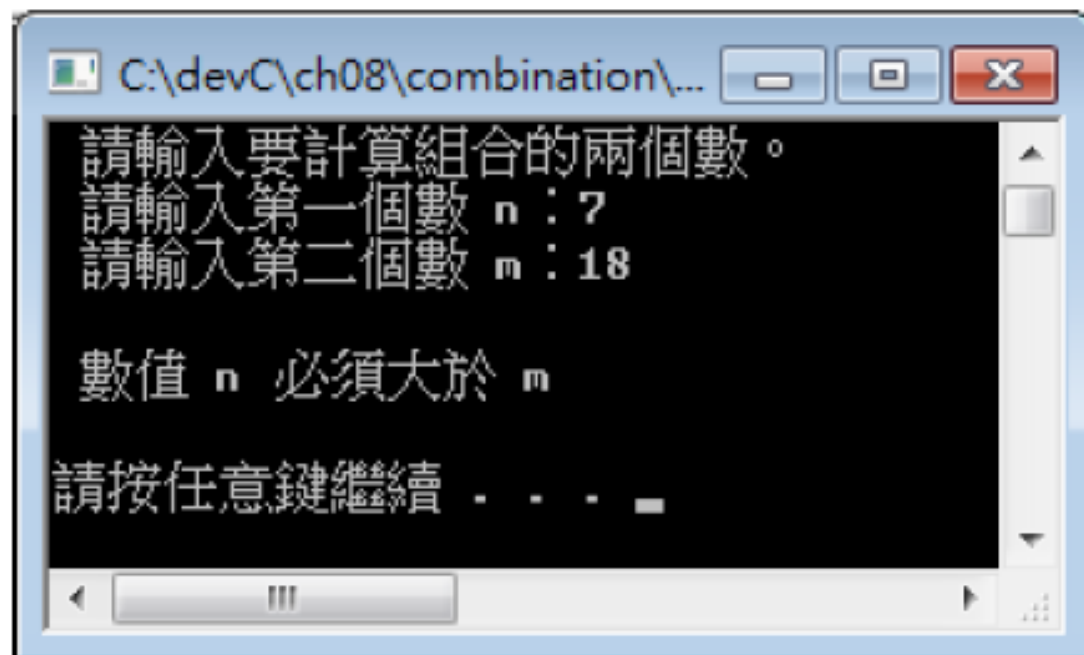


範例：combination.c

使用遞迴函式求數學組合的 C_m^n 之值。使用者可以輸入組合的第一個數 n 和第二個數 m ，程式執行結果會計算數學組合 C_m^n 之值，如下兩圖：



正確操作



錯誤操作

1. 數學組合公式如下：

$$C_m^n = \begin{cases} 1 & /*\text{如果 } n=m \text{ 或 } m=0, \text{ 則傳回 } 1*/ \\ C_m^{n-1} + C_{m-1}^{n-1} & /*\text{如果 } n \geq m \text{ 則傳回此公式的結果值}*/ \end{cases}$$

2. 例如：欲求出數學組合 C_2^5 之值，則計算過程如下：

$$\begin{aligned} C_2^5 &= C_2^4 + C_1^4 \\ &= (C_2^3 + C_1^3) + (C_1^3 + C_0^3) \\ &= (C_2^2 + C_1^2) + (C_1^2 + C_0^2) + (C_1^2 + C_0^2) + 1 \\ &= 1 + (C_1^1 + C_0^1) + (C_1^1 + C_0^1) + 1 + (C_1^1 + C_0^1) + 1 + 1 \\ &= 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 \end{aligned}$$

```
04 int comb(int, int);    /*宣告 comb 函式原型*/
05
06 int main(int argc, char *argv[])
07 {
08     int numN, numM, ans;
09     printf(" 請輸入要計算組合的兩個數。\\n");
10     printf(" 請輸入第一個數 n:");
11     scanf("%d", &numN);
12     printf(" 請輸入第二個數 m:");
13     scanf("%d", &numM);
14     if(numN>=numM)
15     {
16         ans=comb(numN, numM);
17         printf("\\n 組合公式 comb(%d,%d)=%d\\n\\n",numN, numM, ans);
18     }
19     else
20     {
21         printf("\\n 數值 n 必須大於 m \\n\\n");
22     }
23     system("PAUSE");
24     return 0;
25 }
```

C++資料結構與程式設計

指標的活用

NTU CSIE 張傑帆

指標簡介

一般變數要存取該記憶體內的資料，只要書寫其變數名稱即可，存取方式資料是屬於直接取值。

指標變數要存取該記憶體內的資料，必須在指標變數名稱前面加上取值運算子($*$)，才能存取所指記憶體位址的內容，存取資料方式是屬於間接取值。

在程式中使用指標變數的好處是：

1. 透過指標的移動可以存取不同記憶體位址的資料，如字串中的字元或陣列元素等。
2. 函式透過指標可以傳回多個資料。
3. 透過指標變數可在函式間傳遞字串、整個陣列。
4. 指標變數是在程式執行需要時，才動態配置記憶體給該變數使用，不必事先定義或宣告。
5. 支援動態資料結構，如鏈結串列(Linked List)、佇列(Queue)、二元樹(Binary Tree).....等皆可以使用指標變數輕鬆處理。

如何宣告指標

宣告變數時，在變數名稱的**前面**加上「*****」**取值運算子**(Dereference operator或Indirect Operator)，表示該變數可以用來存放所指定資料型別變數的記憶體位址，此類的變數即稱為**指標變數**，其語法如下：

資料型別 *指標變數名稱 , ***指標變數名稱.....;**

指標宣告方式如下：

```
int *ptr;
```

```
/* 宣告ptr是一個指向整數資料的指標 */
```

```
float *ptr;
```

```
/* 宣告ptr是一個指向浮點數資料的指標 */
```

```
char *ptr;
```

```
/* 宣告ptr是一個指向字元資料的指標 */
```

```
void *ptr;
```

```
/* 宣告ptr是一個指向任何(不拘)資料型別的指標 */
```


指標宣告方式如下：

```
int *ptr;
```

```
/* 宣告ptr是一個指向整數資料的指標 */
```

```
float *ptr;
```

```
/* 宣告ptr是一個指向浮點數資料的指標 */
```

```
char *ptr;
```

```
/* 宣告ptr是一個指向字元資料的指標 */
```

```
void *ptr;
```

```
/* 宣告ptr是一個指向任何(不拘)資料型別的指標 */
```

如何存取指標變數

存取指標變數必須先熟悉**取址運算子(&)**和**取值運算子(*)**。

取址運算子是在變數名稱的**前面**加上「&」符號，用來取得變數名稱所對應記憶體의起始位址。

在指標變數前面加上* 取值運算子是用來取得指標變數所指向記憶體位址內的資料，* 也可以稱為**間接運算子**。至於 & 和 * 的使用方式如下：

*指標變數名稱

&指標變數名稱

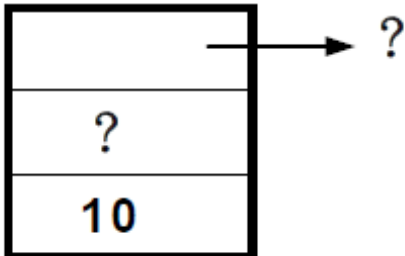
Step 01

宣告 *i* 和 *k* 都是整數變數，並設定 *i* 的初值為 10；*ptr1* 是一個指向整數資料的指標變數，其寫法如下：

```
int i=10 , k ;
```

```
int *ptr1 ;
```

由於一個整數變數和一個指標變數在記憶體中都佔用四個位址，若上面敘述經過宣告後假設變數的記憶體位址由 1000 開始放起，各變數在記憶體中配置如下：

變數名稱	記憶體位址	內容
ptr1	1008~ 1011	
k	1004~1007	?
i	1000~1003	10

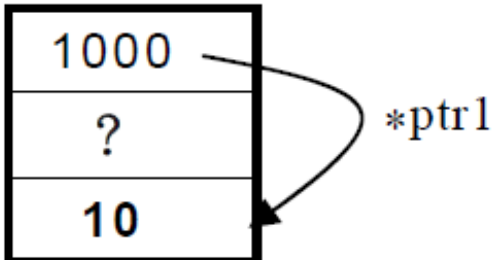
Step 02

將變數 *i* 記憶體位址指定給指標 *ptr1*，其寫法如下：

```
ptr1=&i;
```

執行過上面敘述，會將整數變數 *i* 的位址 1000 存放到 *ptr1* 位址內，相當於 *ptr1* 指到位址 1000。

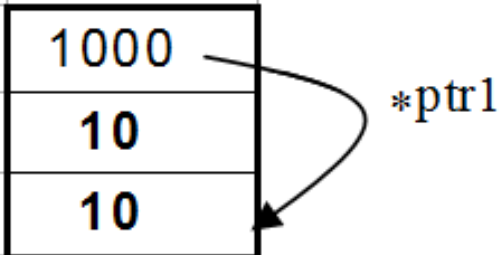
變數名稱	記憶體位址	內容
ptr1	1008 ~1011	1000
k	1004~1007	?
i	1000~1003	10



Step 03 將 ptr1 所指位址內的資料(即 i 的內容)指定給整數變數 k，其寫法如下：

k=*ptr1;

變數名稱	記憶體位址	內容
ptr1	1008~1011	1000
k	1004~1007	10
i	1000~1003	10



在宣告和設定指標變數的內容時，要注意所宣告的資料型別必須與指向變數位址的資料型別相同，否則程式編譯時會發生錯誤。譬如下面寫法是錯誤的：

```
int n=10;  
/*宣告n為整數變數並同時設定初值為10 */
```

```
float *ptr2;  
/*宣告ptr2是一個指向浮點數資料的指標*/
```

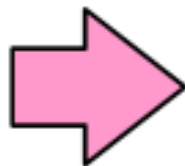
```
ptr2=&n;  
/*指標ptr2所指到資料必須是浮點數與n的資料型別不一致 */
```

當您宣告指標變數的同時也可以和一般變數一樣同時設定初值，其語法如下：

資料型別 *指標變數名稱 = &一般變數名稱;

透過上面語法，我們可以將指標宣告和初值設定由三行敘述改成兩行敘述即可：

```
int k=10;  
int *ptr1;  
ptr1=&k;
```



```
int k=10;  
int *ptr1=&k;
```


指標變數不管宣告成哪一種資料型別，其佔用記憶體大小都一樣。

譬如：您可宣告ptr1是一個指向整數資料的指標，透過sizeof(ptr1)便得知該指標變數佔用記憶體是多少個Bytes。



範例：ptraddress.c

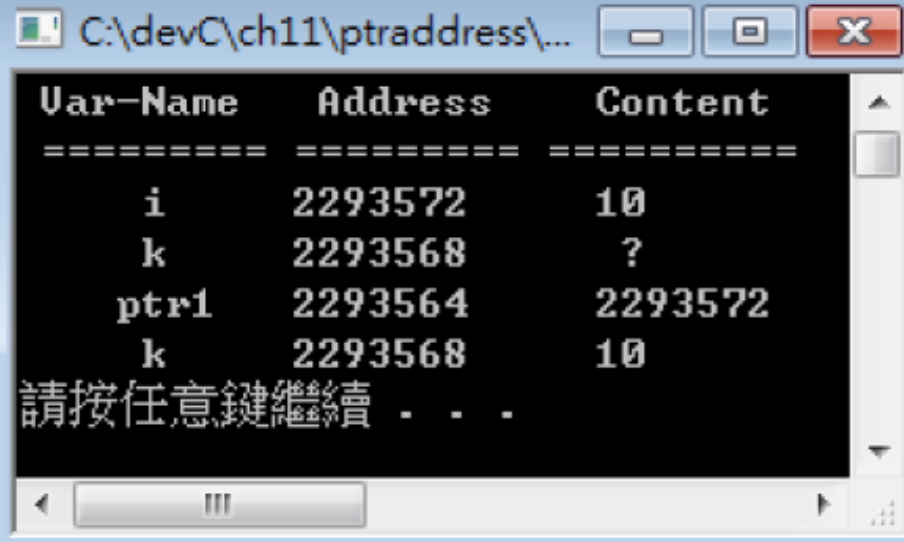
下例顯示各變數及指標變數對應的記憶體位址，以驗證「&」取址運算子和「*」取值運算子的關係。

執行結果

```
C:\devC\ch11\ptraddress\...  
Var-Name    Address    Content  
=====    =====  
      i      2293572      10  
      k      2293568       ?  
    ptr1     2293564    2293572  
      k      2293568      10  
請按任意鍵繼續 . . .
```

程式碼 FileName : ptraddress.c

```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 int main(int argc, char *argv[])
05 {
06     int i=10,k;
07     int *ptr1;
08     printf(" Var-Name   Address   Content   \n");
09     printf(" =====   =====   =====   \n");
10     printf("      i      %d      %d      \n",&i,i);
11     printf("      k      %d      ?      \n",&k);
12     ptr1=&i;
13     printf("    ptr1    %d      %d      \n",&ptr1,ptr1);
14     k=*ptr1;
15     printf("      k      %d      %d      \n",&k,*ptr1);
16     system("PAUSE");
17     return 0;
18 }
```



Var-Name	Address	Content
=====	=====	=====
i	2293572	10
k	2293568	?
ptr1	2293564	2293572
k	2293568	10

請按任意鍵繼續 . . .

動態記憶體配置

C語言無法讓使用者動態決定陣列的大小，往往陣列的大小都必須先宣告其大小在編譯時便被固定，無法在程式執行過程中更改陣列的大小。我們也不能使用下面敘述來宣告陣列的大小，否則程式編譯時會發生錯誤：

```
int n;  
scanf("%d", &n); /* 由輸入值來決定陣列的大小，編譯時會發生錯誤 */  
int myarray[n]; /* 陣列大小宣告時必須是整數常數不能用整數變數 */
```


C語言提供動態記憶體配置(Dynamic Memory Allocation)方式來解決。透過此種方式可以讓使用者在程式執行過程中自行決定隨時能應程式需求配置記憶體空間給程式使用。

做法就是在程式執行的過程中，需要記憶體時才使用 `malloc()` 函式來動態配置一塊記憶體空間，並將此塊配置記憶體空間的起始位址傳回給所指定的指標，接著再移動此指標來存取記憶體位址內的資料，當記憶體不再使用時，再透過 `free()` 函式將此塊記憶體釋放掉歸還給系統。

由於 `malloc()` 函式和 `free()` 函式都定義在 `stdlib.h` 標頭檔內，若程式中有使用到這兩個函式，必須在程式開頭將 `stdlib.h` 標頭檔含入到程式中。

動態配置記憶體-malloc()函式

`malloc()`函式定義於`stdlib.h`標頭檔中，呼叫此函式時會配置您所指定的記憶體空間大小，此函式不像陣列在編譯階段配置記憶體而是在執行階段需要時才配置記憶體。

若配置成功由於此函式無法確知您所要求的記憶空間是要指向那一種資料型別，因此其傳回值設為指向`void`資料型別的指標。

有介紹過此種`void`指標可以強迫它轉換成任何資料型別的指標。

當配置的記憶體空間不足，則會傳回`NULL`值。

由於`malloc()`函式所配置的記憶體是不做初始值設定，使用前建議最好先做清除的工作。

要求配置多少空間來存放資料時，必須要考慮資料的長度，因此指標變數的資料型別必須和配置的記憶體空間所使用的資料型別一致，因此在傳回動態記憶體的起始位址時給指定的指標前必須先將預設的void指標進行指標型別轉換。其語法如下：

指標變數=(資料型別 *)malloc(記憶體空間大小);

[例]

```
int *ptr, n=4;  
ptr=(int*)malloc(sizeof(int)*n);
```

說明

1. sizeof(int)

取得一個整數變數的大小，假設為 4 Bytes。

2. sizeof(int)*n

取得 n 個整數變數的大小，n=4 則 16 Bytes。

3. malloc(sizeof(int)*n)

系統配置連續 16 個記憶體位址給這四個整數使用。並傳回 void 指標指向此塊配置記憶體的起始位址。

4. ptr=(int*)malloc(sizeof(int)*n);

由於 ptr 為指向整數變數的指標，因此使用(int*)將 malloc()函式傳回的 void 指標轉換成 int 型別的整數指標，最後再指定給 ptr 整數指標。

5. 若 ptr 為指向字元變數的指標，則 malloc()函式寫法如下：

```
int *char, n=4;  
ptr=(char*)malloc(sizeof(char)*n);
```

6. 程式中如何檢查是否配置記憶體成功，其寫法如下：

```
int *ptr, n=4;  
ptr=(int*)malloc(sizeof(int)*n);  
if (ptr==NULL)    /* 若 ptr 指標為 NULL，表示配置記憶體失敗 */  
{  
    printf("記憶體空間不足,無法配置 ..... \n");  
    exit(0);  
}  
else  
{  
    .....  
    ..... 配置成功執行  
    free(ptr);    /* 釋放記憶體 */  
}
```


釋放記憶體-free()函式


動態配置的記憶體若不再使用時，必須使用free()函式釋放掉歸還給系統，如此才不會浪費記憶體空間。

使用malloc()函式動態配置記憶體之後，若不用使用記憶體空間時，可以使用stdlib.h標頭檔所定義的free()函式指定要釋放的記憶體指標。其語法如下：

```
free(指標變數);
```

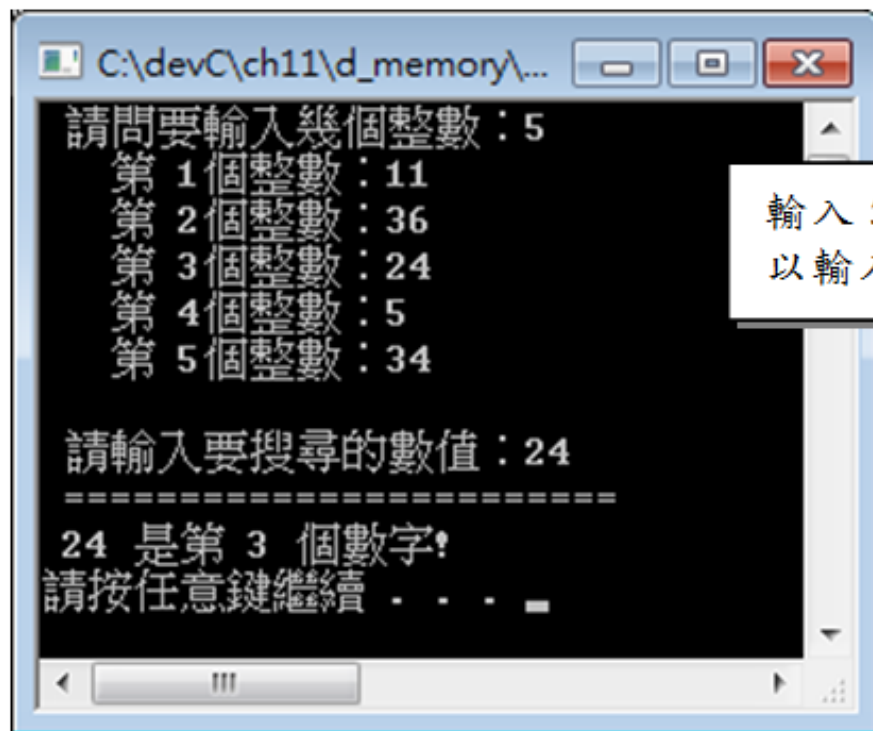
```
int *ptr, n=4;  
/* 動態配置記憶體 */  
ptr=(int*)malloc(sizeof(int)*n);  
/* 釋放記憶體 */  
free(ptr);
```

實機練習-循序搜尋

 範例：d_memory.c

使用動態配置記憶體的方式，讓使用者可以自行決定要輸入幾個整數，當逐一輸入整數之後，最後再使用循序搜尋法搜尋您所要搜尋的數值是在第幾個整數。

執行結果



```
C:\devC\ch11\d_memory\...
請問要輸入幾個整數：5
第 1 個整數：11
第 2 個整數：36
第 3 個整數：24
第 4 個整數：5
第 5 個整數：34

請輸入要搜尋的數值：24
=====
24 是第 3 個數字!
請按任意鍵繼續 . . .
```

輸入 5，使用者可以輸入 5 個整數



輸入 3，使用者可以
輸入 3 個整數

程式碼 FileName : d_memory.c

```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 int main(int argc, char *argv[])
05 {
06     int *ptr, n, i, search_num, num=-1;
07     printf(" 請問要輸入幾個整數：");
08     scanf("%d", &n);
09     ptr=(int*)malloc(sizeof(int)*n);    /* 動態配置記憶體 */
10     for(i=0;i<n;i++)
11     {
12         /* 將使用者輸入的資料放入指標 ptr 目前指向位址的記憶體空間 */
13         printf("    第%2d 個整數：", i+1);
14         scanf("%d", ptr+i);
15     }
16     printf("\n");
17     printf(" 請輸入要搜尋的數值：");
18     scanf("%d", &search_num);
```

```
19     for(i=0;i<n;i++) /* 循序搜尋法 */
20     {
21         if(*(ptr+i)==search_num)
22         {
23             num=i; break;
24         }
25     }
26     printf(" =====\n");
27     if(num== -1)      /* 顯示搜尋的結果 */
28     {
29         printf(" 沒有這個數字-> %d \n", search_num);
30     }
31     else
32     {
33         printf(" %d 是第 %d 個數字! \n", search_num, (num+1));
34     }
35     free(ptr);      /*釋放記憶體*/
36     system("PAUSE");
37     return 0;
38 }
```

小實驗

- 動態陣列與靜態陣列的Size上限差很多嗎？
- 若動態陣列不回收記憶體會怎麼樣？
 - 讓我們來操爆記憶體吧~