C＋＋教學上課講義

授課教師：周敬斐
講義編號：A

1.1、 為何使用 C＋＋(5 min)

1.1.1、C＋＋是目前使用最普遍的語言。

1.1.2、在 1998 由 ANSI（American National Standards Insitute）與 ISO（International Standards Organzation）所制訂標準統一化。

1.1.3、C/C++，C＋＋包含了 C 以前的功能並加入許多更多的功能。

1.1.4、學習 C＋＋對於將來各種程式語言學習均有幫助，例如 JAVA，一種跨越平台（PLATFORM）的語言就是以 C＋＋為基礎而開發出來的新一代語言。

1.2、 什麼是電腦(5 min)

1.2.1、一種可以在很快的速度下做邏輯決策的裝置。

1.2.2、新型的電腦運算能力是數十年前的幾百萬倍。

1.2.3、其他支援裝置，輸入裝置如鍵盤、滑鼠。輸出裝置如螢幕、印表機等等。

1.3、 電腦的組織(5 min)

1.3.1、Input Device

1.3.2、Output Device

1.3.3、Memory Unit

1.3.4、Arithmetic Unit

1.3.5、Central processing Unit (CPU)

1.3.6、Secondary storage Unit

1.4、 機器語言、組合語言、高階語言(5 min)

1.4.1、Machine Language 機器語言

+1300042774
+1400593419
+1200274027

所有電腦的指令都是以代號查表進行

1.4.2、Assembly Language 組合語言

ADD AX,2
MOV dx,ax

有基礎的指令，靠移動電腦記憶體中的值來達成命令

1.4.3、High-level Language 高階語言

薪水 ＝ 底薪 ＋ 獎金

這樣子的一個算是由編譯器組合成組合語言在連結成機器語言。

Ex1.4

進入 DOS **模式，輸入** DEBUG。

執行 a 100

輸入 add ax,2

按 ENTER **離開**

輸入 u100

輸入 Q **離開**

1.5、 典型 C＋＋基本環境(5 min)

1.5.1、C＋＋**程式的副檔名通常為** .cpp ; .cxx ; .c 。

1.5.2、C＋＋**程式開發過程**

C＋＋	**電腦**	**備註**
Editor	DISK	**編寫程式**
Preprocessor	DISK	**前置處理程式**
Compiler	DISK	**編譯**
Linker	DISK	**連結**
Loader	Primary Memory	**把程式讀到記憶體中**
CPU	Primary Memory	**執行**

1.6、 Microsoft Visual C++ 環境介紹(5 min)

講解如何在 MS VC++**中開一個新的程式**

1.7、 最簡單的程式-印出一行字(15~30 min)

```
//Prg. 1.7.1:    prg1.7.1.cpp
//第一個程式
#include <iostream.h>

int main()
{
    cout << “I love u all baby!!” ;

    return 0;      //程式結束傳回 0
}
```

1.7.1、**符號「//」的功能是註解**。

1.7.2、**程式行「**int main()」

int **是數值型態的意思、**MAIN **是表示這裡面寫的是主程式**。

1.7.3、**指令「**cout」**用法**。

1.7.4、**補充格式化輸出功能**

\n	**換新行**
\t	**輸出至定位點**

\r	將游標換回原來的定位點
\a	發出聲音
\\	後退鍵
\”	雙位元

1.8、另一個簡單的程式-兩數相加(15~30 min)

```
//Prg. 1.8.1:    prg1.8.1.cpp
//兩數相加
#include <iostream.h>

int main()
{
     cout << 2 + 3 ;

     return 0;      //程式結束傳回 0
}
```

1.8.1、簡單的數字相加
1.8.2、設定變數，讓變數和變數相加
1.8.3、輸入（cin）指令的使用。
1.8.4、設計一個簡單的兩數相加程式，由使用者輸入第一個數字和第二個數字，然後寫程式讓結果印出。

1.9、 變數的觀念(Memory Concept)(5~30 min)

1.9.1、電腦中變數的處理方式

薪資 ＝ 底薪 ＋ 獎金

電腦的處理方式是先處理等號右方的，再將等號右邊的結果放入左邊。

1.10、 算數(Arithmetic)(5~30min)

1.10.1、運算符號

C＋＋運算	數學符號	數學表示式	C＋＋表示式
加	+	F+7	F + 7
減	-	p-c	P – c
乘	*	Bm	B * m
除	/	X/y	X / y
餘數	%	R mod s	R % s

運算子	順序
( )	優
*, / , %	中

+ , -	**後**

1.10.2、**把自己當成電腦**

試著去翻譯日常的數學運算式成為電腦看得懂得語法。

例如：

數學：y = ax + b

電腦；y = a * x + b

1.11、 決策制訂：同等、相關運算子(Equality and Relational Operator)(5~30min)

1.11.1、**決策運算**

相等運算			
數學運算符號	C＋＋**運算符號**	C＋＋**範例**	**說明**
＝	==	X==Y	X **等於** Y
≠	!=	X!=Y	X **不等於** Y
關係運算子			
>	>	X > Y	X **大於** Y
<	<	X < Y	X **小於** Y
≧	>=	X >= Y	X **大於等於** Y
≦	<=	X <= Y	X **小於等於** Y

1.11.2、if **判斷式**

```
//Prg. 1.11.2:    prg1.11.2.cpp
//判斷誰大
#include <iostream.h>
int main()
{
    int x,y;
    cout << “Input X:” ;
    cin >> x;
    cout << “Input Y” ;
    cin >> y;

    if (x>y)
       {
          cout << “First is BIG” ;
       }

    if(x<y) {cout << “Second is BIG” ;}


    return 0;      //程式結束傳回 0
```

```
}
```

2.1、基本的 if/else 選擇結構(20 min)

2.1.1、基本的 if/else 結構

if 我不夠帥
    我就上網找老婆
else
    我去參加相親

```
//Prg 2.1.1 : prg2_1_1
//if / else 結構範例

#include <iostream.h>

void main()
{
    int grad;
    cout << "請輸入成績；" ;
    cin >> grad ;

    if (grad >=60)
        cout << "及格！！" ;
    else
        cout << "當掉" ;
}
```

2.2、多層結構判斷(20 min)

```
if 我成績 90
    印出 A
else
    if 我成績 80
        印出 B
    else
        if 我成績 70
            印出 C
        else
            if 我成績 60
                印出 D
            else
                印出不及格
```

◎請同學將上面的敘述翻譯成程式碼

2.3 、while 重複結構(20 min)

2.3.1、while 的說明

while 當有更多的促銷品
    我繼續購買

範例程式：

```
int product = 2;
```

```
while ( product <= 1000 )
      product = product + 1 ;
```

2.3.2、寫寫看幾個使用 WHILE 的程式

1.**寫依個用來計算兩數相加的程式，當第一個數字輸入**-1 **的時候，程式就自動跳開**。

2.**寫一個** 99 **乘法表，只要印出** 2 **的那一欄**。

2 * 1 = 2
2 * 2 = 4
….
….
2 * 9 = 18

3.**寫一個計算平均的程式**。
例如輸入十次不同的數，程式最後輸出平均值。

3.1、工作運算子 (20~40 min)

3.1.1、基本的工作運算子

工作運算子	表示式	解釋
+=	C += 7	C = C + 7
-=	D -= 4	D = D – 4
*=	E *= 5	E = E * 5
/=	F /= 3	F = F / 3
%=	G %=9	G = G % 9

3.1.2、增加/減少 運算子

運算子	表示式	解釋
++	A++	A = A + 1
++	++A	A = A + 1
--	B--	B = B –1
--	--B	B = B –1

```
//Prg. 3.1.2:    prg3_1_2.cpp
//++運算子前置、後置差別
#include <iostream.h>

int main()
{
     int c ;
     c = 5 ;
     cout << c << endl ;
     cout << c++ << endl ;
     cout << c << endl << endl ;

     c = 5 ;
     cout << c<< endl;
     cout << ++c << endl ;
     cout << c << endl ;

     return 0;      //程式結束傳回 0
}
```

3.2、計次控制(10~20 min)

3.2.1、計次控制需要條件

1.計次用的變數。
  (int counter;)
2.計次用的變數初始值。
  (counter = 1;)
3.計次用的便數值的增加。
  (++counter;)
4.測試迴圈結束的值。
   while (counter <= 10)

```
//Prg. 3.2.1:    prg3_2_1.cpp
//計次控制
#include <iostream.h>

int main()
{
     int counter ;
     counter = 1;

     while (counter <= 10) {
          cout << counter << endl ;
          ++counter;
     }

     return 0;      //程式結束傳回 0
}
```

4.1、for 迴圈 (20~60 min)

4.1.1、for 的用法

for (*expression1; expression2; expression*3)
    statement

相等於

```
expression1
while (expression2){
    statement
    expression3
}
```

4.1.2、for 迴圈幾乎處理了所有的計次迴圈系統，請看以下的範例。

```
//Prg. 4.1.1:    prg4_1_1.cpp
//使用 FOR 來處理計次迴圈
#include <iostream.h>

int main()
{
     for (int counter = 1 ; counter <=10 ; counter++ )
          cout << counter << endl ;
     return 0;      //程式結束傳回 0
}
```

4.2、應用 for 迴圈 (20~60 min)

4.2.0、FOR 結構與計次結構比較

```
for( int number = 2;                        //計次變數初始並賦予值
     number <= 100 ;                        //判斷計次變數
      sum += numbner , number += 2)         //改變計次變數的值。
```

4.2.1、例如從一到一百的結構

```
for(int i = 1; i <= 100 ; i++)
```

4.2.2、從一百到一的結構

```
for(int i = 100; i >= 1 ; i--)
```

4.2.3、從 7 到 77 的 FOR 結構。

4.2.4、從 20 到 2 每一次少 2。

4.2.5、依照下面的順序 2,5,8,11,14,17,20

4.2.6、依照下面的順序 99,88,77,66,55,44,33,22,11,0

4.3、switch 應用(20~60 min)

4.3.1、switch 的格式

```
switch (變數){
    case 條件 1
        命令
        break;

    case 條件 2
        命令
```

```
            break;
        …
        …
    }
```

4.3.2、**將** switch **應用在** if/else **結構中**

如讓使用者輸入分數，然後依照分數列出分數的英文等級，如 90 **分就是** A**，** 80 **分就是** B..**等**。

4.4、do/while 重複結構(20~60 min)

4.4.1、**基本的** while **重複結構**

```
while( condition)
```

4.4.2、do / while **重複結構**

```
do
    statement
while(condition);
```

或是

```
do {
    statement
}while (condition);
```

```
//Prg. 4.4.2:    prg4_2_2.cpp
//用 do/while 重複結構
#include <iostream.h>

int main()
{
    int counter = 1;
    do {
        cout << counter << "    ";
        counter ++ ;
    } while (counter <= 10) ;
    cout << endl ;
    return 0 ;
}
```

4.5、邏輯運算子

4.5.1、**邏輯運算子 ＆＆**

相等於英文中的 AND **，也就是中文中的且。**

如果要連結兩個判斷式就必須使用＆＆運算子。

例如：

if **年齡大於** 20 **且 性別為男生**

```
if age >= 20 && sex = 1 ;
```

4.5.2、**邏輯運算子 ｜｜**

相等於英文中的 OR

例如：

```
if age >= 20 or age <= 12
```

5.1、函數簡介（5min）

5.2、叫用數學函數(5~10 min)

5.2.1、**叫用數學函數**

在呼叫數學函數的時候需要引入另外一個表頭檔 math.h

5.2.2、**可用的數學函數**

函數名稱	**解釋**	**範例**
ceil(x)	**調整到最接近的整數**	ceil(9.2) is 10.0
cos(x)	Cosine	cos(0.0) is 1.0
exp(x)	**科學記號** e	exp(1.0) = 2.71282
pow(x , y)	X **的幾次方**	pow(2 , 3) is 8

5.3、函數定義(10~30min)

5.3.1、**簡單的函數應用**

```
//Prg. 5.3.1:    prg5_3_1.cpp
//簡單的函數應用

#include <iostream.h>

int square( int ) ; //函數宣告

int main()
{
    for ( i = 1 ; i <= 10 ; i ++)
    {
        cout << square ( i ) << "\t" ;
    }
    cout << endl ;
    return 0;      //程式結束傳回 0
}

int square( int y)
{
    return y * y ;
}
```

5.3.2、**函數宣告**

int square(int) ;

int **是函數的型態**

square ()**是函數的名稱，（）中的** int **是表示接收值的型態是整數**。

5.3.3、**函數本體**

return-value-type function-name(parameter-list)

{

declarations and statements

}

5.3.4、return **是幹什麼用的？！**

return expression;

5.4、設計一個函數，可以傳入 a,b,c 三個數，並且比較，然後傳回最大的那一個數。

這個地方由老師親自示範函數的寫作，並帶領同學寫作衍生的函數。

給同學建議，因為函數對於 C++來說是最重要的部分，不會很難，要多練習，因為將來任何大型的程式其實都是靠著一段一段小小的函數結合起來成為一個大型的程式。所以請同學加油，有問題要問肥老師！！

6.1、函數型態

6.1.1、函數型態

所有在 C++中的函數都必須事先宣告型態，這些型態包含了以下資訊。

型態	說明
long double	倍精度的浮點
double	單精度的浮點
float	浮點
unsigned long int	無正負號長整數
long int	長整數
unsigned int	無正負號整數
int	整數
unsigned short int	無正負號短整數
short int	短整數
unsigned char	無號字元
char	字元

6.2、表頭檔(Header File)

6.2.1、表頭檔是做什麼的呢？

6.2.2、有什麼表頭檔可以用？

6.2.3、表頭檔放在哪？

6.3、產生隨機數字(Random Number Generation)

6.3.1、如何產生隨機數字

i = rand() ;

rand() 是一產生隨機數字的函數，在使用前必須引入表頭檔 stdlib.h

6.3.2、如何使用 rand()產生隨機數字？

Rand 會產生從 0 到 32767 之間的任何數字。

所以你可以應用這之間的數字來做一些事情，例如統計，我們來寫一個程式看待這個函數。

6.3.3、試著寫程式產生一個隨機數字。

```
//Prg. 6.3.3:    prg6_3_3.cpp
//產生隨機數字

#include <iostream.h>
#include <stdlib.h>
int main()
{
    for(int i = 1; i <= 20 ; i ++) {
        cout << (1 + rand() % 6) << "\t" ;
        if ( i % 5 == 0 )
            cout << endl ; //換行
    }
```

```
    return 0;      //程式結束傳回 0
}
```

6.3.4、設計一個程式來統計一個骰子投擲 6000 次的結果。

同學先自己想想看，老師可能挑會寫的人上台。

最後老師寫一遍給大家看。

6.3.5、變動隨機數字

另用一個指令叫做 srand()**可以使電腦產生不同的隨機數字。**

用法：srand(x) , x **必須為** unsigned int **的型態**

```
//Prg. 6.3.5:    prg6_3_5.cpp
//隨機產生隨機數字

#include <iostream.h>
#include <stdlib.h>
int main()
{
    unsigned seed ;
    cout << "請輸入：" ;
    cin >> seed ;
    srand ( seed ) ;
    for(int i = 1; i <= 20 ; i ++) {
        cout << (1 + rand() % 6) << "\t" ;
        if ( i % 5 == 0 )
            cout << endl ; //換行
    }
    return 0;      //程式結束傳回 0
}
```

6.4、產生隨機數字的應用

老師的話。隨機數字的應用層面可以很廣泛，光一個統計學上的用處就已經說不盡了，所以說還有許多數學系的學生利用 C++**來對他們的研究進行考驗。否則真的叫你投擲** 6000 **次骰子可能嗎？**

7.1、函數覆載（overloading）

7.1.1、當在程式中定義函數的時候，妳必須去指定一個回傳的型態，不過當我們如果寫一個函數用來做兩數相加的動作，而我們想要將三個整數作加法運算的時候就又必要去設計另外的函數來達成我們需要的目的。

為了減少複製函式的動作，C＋＋可以讓妳使用相同的名稱來定義函式。當妳編譯成式的時候，C＋＋會先檢查每個函式所使用的參數，然後呼叫正確的函式。

函式覆載是 C＋＋的一項特色，這在 C 語言中沒有，而且函式覆載是很方便的技巧。

7.1.2、如何使用函式的覆載？

```
#include <iostream.h>

int add_values(int a, int b)
{
      return (a + b) ;
}

int add_values(int a, int b,int c)
{
      return ( a + b + c) ;
}

void main()
{
      cout << “200 + 100 = “ << add_values(200 ,100) << endl;
      cout << “200 + 100 + 2 = “ << add_values(200 ,100, 2) << endl;
}
```

7.2、函數的循環（Recursion）

7.2.1、何謂函數的循環

一個循環函數就是一個自己呼叫自己的函數，函數的循環一般都是在高等程式設計的課程中討論，不過我們現在可以用一些簡單的程式來讓自己瞭解何謂(Recursion)。

7.2.2、在什麼種情況下要使用（Ｒｅｃｕｒｓｉｏｎ）函數？

假設下面這一種狀況：

當程式需要執行下面這種計算，

n * (n-1) * (n-2) …. * 1

n!

7.2.3、請大家試著去用 FOR 迴圈寫出 讓使用者輸入 n 結果輸出 n!。

7.2.4、用(Recusion)來寫出 N！。

```
#include <iostream.h>

unsigned long n( unsigned long)

void main()
{
```

```
    for (int i=1 ; i <=10 ; i++)
    {
        cout <<    "\t" << i << "! =" << n(i) << endl ;
    }
}

unsigned long n( unsigned long i)
{
    if (i<=1)
          return 1 ;
    else
          return i * n(i – 1)
}
```

7.3、**函數傳值傳址**(call by value , or , call by refrerence)

7.3.1、**函數傳值**(call by value)

```
#include <iostream.h>

int add_value(int a, int b) {return (a + b) ;}
void main()
{
  int x , y ;
  x = 2 , y = 3;
  cout << add_value(x,y) << endl ;
  cout << x << endl;
  cout << y << endl;
}
```

Add_valuec **函數只是把** MAIN **給他的兩個的** x,y **的** VALUE **放到** add_value **中的** A、B **變數中，而這樣子的傳遞手法就叫做** CALL BY VALUE**，所以在** MAIN（）**中的** X、Y **變數的值並未改變**。

7.3.2、**函數傳址**(call by reference)

```
#include <iostream.h>

void add_value(int &a, int &b) {a = (a + b) ;}
void main()
{
  int x , y ;
  x = 2 , y = 3;
  add_value(x,y) ;
  cout << x << endl;
  cout << y << endl;
}
```

由於函數傳的不適值而是該變數的位置，所以 X **的值就會被改變**。

7.4、**函數的預設值**

7.4.1、**當妳呼叫函式的時候，如果沒有設定傳入參數值的話，**C＋＋**就會以妳預設的** DEFAULT **進行處理**。

```
#include <iostream.h>

void show(int a=1, int b=2, int c=3)
{
    cout << “a” << a << “b” << b << “c” << c << endl;
}

void main()
{
    show();
    show(1001);
    show(1001,2002);
    show(1001;2002;3003);
}
```

8.1、陣列的結構

8.1.1、陣列儲存的方式

C[0]	2
C[1]	3
C[2]	0
C[3]	-123
C[4]	3
C[5]	1310
C[6]	41
C[7]	23
C[8]	54
C[9]	467
C[10]	-2342

8.1.2、陣列的使用方法

```
cout << c[0] << c[1] << c[10] ;
x = x[3] / 10;
```

8.2、陣列的宣告

8.2.1、陣列的宣告

```
int b[10] ;
int x[9] , y[9] ;
```

8.2.2、陣列的使用

```
#include <iostream.h>

void main()
{
    int a, n[10] ;
    for (a = 0; a< 10 ; a++)
    {
        n[a] = a ;
    }
    cout << “Elements Value”;

    for (a=0 ; a<10 ; a++)
    {
        cout << a << “\t” << n[a] << endl;
    }
}
```

這是利用 FOR **迴圈來賦予每一個陣列元素一個值，當然我們也可以事先指定陣列中的值。**

```
#include <iostream.h>

void main()
{
    int n[10] = {32, 26, 54, 12, 23, 44, 56, 78, 77, 88};
```

```
    for (int j=0 ; j<10 ; j++)
    {
        cout << “no.” << j << “is    “ << n[j] << endl ;
    }
}
```

8.3、constant varible **固定變數**

8.3.1、**固定變數的使用時機主要在於想要宣告一個值不變的的變數**。

```
#include <iostream.h>

void main()
{
    const int x = 7 ;
    cout << “const variables x is : “ << x ;
}
```

8.3.2、**錯誤的固定變數宣告方式**

```
#include <iostream.h>

void main()
{
    const int x ;
    x=7;
    cout << “const variables x is : “ << x ;
}
```

8.4、**利用陣列的基本應用**

8.4.1、**下列是一個計算總和的程式**。

```
#include <iostream.h>

void main()
{
    const int arraySize    = 12 ;
    int a [ arraySize ] = { 1 , 3 , 5 , 7, 4 , 12, 24 , 44, 54, 53, 98, 12};

    int total = 0 ;
    for (int k=0; k< arraySize; k++)
    {
          total += a [ k] ;
    }
    cout << “The sum is:” << total ;
}
```

8.4.2、**另一個另類的陣列應用（統計）**

```
#include <iostream.h>

void main()
{
      const int responseSize = 40, frequencySize = 11;
      int responses[ responseSize ]
```

```
            = { 1, 2, 6, 4, 8, 5, 9, 7, 8,10,
                  1, 6, 3, 8, 6,10, 3, 8, 2, 7,
                  6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
                  5, 6, 7, 5, 6, 4, 8, 6, 8,10};

      int frequency[ frequencySize ] = { 0 } ;

      for (int answer = 0; answer < responseSize ; answer++)
      {
            ++frequency[ responses[ answer ] ];
      }
      cout << "Rating" << "\t" << "Frequency" << endl;

      for (int rate=1 ; rate <frequencySize ; rate++)
      {
            cout << rate << "\t" << frequency[rate] << endl;
      }
}
```

上面這一個程式有點小小的複雜，要仔細研究喔！

10.1、排序陣列

在電腦應用程式中，我們有很多的機會來做排序的動作，例如，如果我們需要將學生的名單依照字母先後順序排序等等，所以我們現在就來討論有關於陣列的排序。

```
#include <iostream.h>

void main()
{
	const int arraySize = 10 ;
	int a [ arraySize ] = {2 ,6 , 4, 8, 10, 12, 89, 68, 45, 37 };
	int i, hold ;

	cout << "資料原本的順序如下：\n";

	for (i=0 ; i<arraySize; i++)
		cout << "	" << a [ i ] ;

	for (int pass = 0 ; pass < arraySize - 1; pass++)
	{
		for (i=0 ; i<arraySize ; i++)
		{
			if ( a[i] > a[i+1] )
			{
				hold = a[i] ;
				a[i] = a[i+1];
				a[i+1] = hold ;
			}
		}
	}

	cout << "\n 資料改變後的順序如下：\n";

	for (i=0 ; i<arraySize; i++)
		cout << "	" << a [ i ] ;

}
```

10.2、專案探討 C＋＋陣列在統計學上的應用

在統計學上我們常常會使用到下面幾個計算，一個是平均數 MEAN，中位數 MEDIAN，和眾數 MODE，藉由 C＋＋運用陣列可以來對這些樣本空間進行計算。

下面的程式很多很複雜，不過詳加端倪可以看出其實運作並不難，下面就是一個很好的例子來做陣列的應用。

```
#include <iostream.h>
void mean( const int answer[] , int arraySize);
void median( int answer[], int size);
void mode( int freq[], int answer[], int size);
void bubbleSort( int a[], int size);
void printArray( const int a[], int size);

void main()
{
     const int responseSize = 99 ;
     int frequency[10] = { 0 },
          response[ responseSize ] =
               {6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
                7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
                6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
                7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
                6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
                7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
                5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
                7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
                7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
                4, 5, 6, 1, 6, 7, 8, 7 };
     //可以在下面三行程式前加上//，好一次觀察一種函數。
     mean( response, responseSize) ;
     median( response, responseSize );
     mode( frequency, response, responseSize );
}

void mean( const int answer[] , int arraySize)
{
     int total = 0;
     cout << "\n********** Mean **********\n" ;

     for(int j=0 ; j< arraySize; j++)
          total += answer[ j ] ;

     cout << "mean 就是所有資料的平均：" << total / arraySize << endl;
}

void median( int answer[], int size)
{
     cout << "\n********** Median **********\n" ;
     cout << "為排序的陣列資料內容為：\n" ;
```

```
    printArray( answer , size );
    bubbleSort( answer , size );
    cout << "\n\n 排序過後的陣列為：" ;
    printArray( answer , size );
    cout << "\nMedian 就是中位數，也就是樣本經過排序後出現在最中間的數："
         << answer[ size / 2] << "\n\n" ;
}

void mode( int freq[], int answer[], int size)
{
    int rating, largest = 0, modeValue = 0;
    cout << "\n********** Mode 眾數 **********\n";

    for(rating =1 ; rating <= 9; rating++)
         freq[rating] = 0 ; //設定初始值

    for(int j=0; j< size; j++)
         ++freq[ answer[j] ];

    cout << "數字\t 出現次數\t 圖表\n";

    for( rating=1 ; rating <= 9 ; rating++)
    {
         cout << "\t" << rating << "\t" << freq[ rating ] << "    \t";

         if(freq[rating] > largest)
         {
              largest = freq[ rating ];
              modeValue = rating ;
         }

         for(int h=1; h<=freq[rating]; h++)
              cout << "*" ;
         cout << '\n' ;
    }
    cout << "mode 就是眾數，也就是出現最多次的數：" << modeValue ;
}

void bubbleSort( int a[], int size)
{
    int hold;
    for(int pass = 1; pass < size ; pass++)
    {
         for(int j=0; j<size -1; j++)
         {
              if(a[j]>a[j+1])
              {
                   hold = a[j];
                   a[j] = a[j+1];
                   a[j+1] = hold;
              }
         }
```

```
    }
}

void printArray( const int a[], int size)
{
    for(int j=0; j<size; j++)
    {
        if (j%20 ==0)
            cout << endl ; //換行

        cout << "  " << a[j] ;
    }
}
```

10.3、**搜尋陣列（線性搜尋**(linear Search)）

在很多的應用程式中都必須使用的查詢的動作，例如，我們如果把客戶資料放在一個名為 CUSTOM **的陣列的時候，假設陣列中的原素有兩千個，那麼如果今天我們必須要尋找一位客戶，我們就得要一個元素一個元素的去看，所以我們必須使用搜尋的技術**。

Linear Search

C＋＋**程式一開始就賦予了一個從** 2 **開始，以** 2 **為乘底到** 100 **的陣列，所以我們只要輸入** 2 **的倍數，程式就會輸出符合這個條件的元素是哪一個**。

```
#include <iostream.h>
int linearSearch( const int array[], int key, int sizeOfArray );

void main()
{
    const int arraySize = 100;
    int a[ arraySize ] , searchKey, element;

    for(int x=0; x<arraySize; x++)
        a[x] = 2 * x;

    cout << "請輸入數值的搜尋關鍵字(Search Key)：" ;
    cin >> searchKey ;
    element = linearSearch( a, searchKey, arraySize ) ;

    if( element != -1 )
        cout << "找到了:" << element << endl ;
    else
        cout << "找不到！" << endl ;
}

int linearSearch( const int array[], int key, int sizeOfArray )
{
    for(int n=0; n<sizeOfArray; n++)
    {
        if( array[n] == key )
```

```
                return n ;

        }
        return -1 ;
}
```

10.4、多維陣列

如果今天我們需要使用到一個陣列分別可以存放姓名、電話兩個欄位的時候，就必須使用到多維陣列。

	Colum0	Colum1	Colum2	Column3
Row0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
Row1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
Row2	A[2][0]	A[2][1]	A[2][2]	A[2][3]

上面的表格是陣列中元素的排列方式

多維陣列的使用

```
#include <iostream.h>
void printMArray(int a[][3]);

void main()
{
        int array1[2][3] = {{1,2,3},{4,5,6}},
                array2[2][3] = {1,2,3,4,5,6},
                array3[2][3] = {{1,2},{4}};

        cout << "陣列 array1 的元素為：" << endl ;
        printMArray( array1 );

        cout << "陣列 array2 的元素為：" << endl ;
        printMArray( array2 );

        cout << "陣列 array3 的元素為：" << endl ;
        printMArray( array3 );
}

void printMArray(int a[][3])
{
        for(int i=0;i<2;i++)
        {
                for(int j=0;j<3;j++)
                        cout << a[i][j] << ' ' ;
                cout << endl;
        }
}
```

11.0、字串函式庫的使用

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

void main()
{
      char x[] = "Happy Birthday to You" ;
      char y[25], z[15];

      cout << "The string in array x is:" << x
            << "\nThe string in array y is:" << strcpy(y,x)
            << '\n' ;
      strncpy(z,x,14); //並不會複製 NULL 字元
      z[14] = NULL;
      cout << "The String in array z is: " << z << endl ;
}
```

11.1、指標的宣告與初始化

C＋＋程式碼會將變數儲存在記憶體中，而一個指標就是一個記憶體位址，他會址向或參考到一個特定的位置，就好向先前提過的函數傳址的方式一樣。同樣地，如果程式中使用到陣列或字元字串，通常程式也會使用指標來處理陣列的元素，使用指標可以減少程式碼，也可以增加程式的可讀性。

指標就像其他變數一樣，在使用之前必須要先宣告。

```
int *countPtr, count;
```

其中 int 的意義就是讓指標指到一個是整數的值。當然妳也可以宣告其他種形態的指標。

```
Float *xPtr, *yPter;
```

而當妳宣告的時候在變數前面加上一個＊號，就表示這個變數是一個指標。

11.2、**指標運算子**

符號&，也就是位址運算子，是一個自反的運算子，他會傳回指標中的值。

Int y =5;

Int *yPtr;

yPtr = &y ;

這樣子的動作會將 y **的位址傳給** yPtr。

```
#include <iostream.h>

void main()
{
        int a;        //a  是一個整數
        int *aPtr; //aPtr  是一個指標

        a=7;
        aPtr = &a; //將 a 變數的位址傳給 aPtr 變數。

        cout << "a  的位址為：" << &a ;
        cout << "\naPtr  的值為：" << aPtr;

        cout << "\n\na  的值為：" << a ;
        cout << "\n*aPtr  的值為：" << aPtr;

        cout<< "\n\n 所以＊與＆是相互顛倒的。"
                << "\n&*aPtr = " << &*aPtr
                << "\n*&aPtr = " << *&aPtr << endl ;
}
```

11.3、**利用參考來做泡沫排序**(Call-By-Reference)

泡沫排序的作法是將兩個不同的值做比較，如果第一個值比第二個大的話，就不變，否則就將兩個值的位置相互對調。這就叫做泡沫排序。

```
#include <iostream.h>
void bubbleSortR(int *array, const int size);

void main()
{
	const int arraySize = 10;
	int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
	int i;

	cout << "Data Item in origrinal order\n" ;

	for(i=0;i<arraySize;i++)
		cout << "	" << a[i] ;

	bubbleSortR(a, arraySize);

	cout << "\nData items in ascending order\n" ;

	for(i=0;i<arraySize;i++)
		cout << "	" << a[i] ;
	cout << endl ;
}

void bubbleSortR(int *array, const int size)
{
	void swap(int *,int *);
	for(int pass=0;pass<size-1;pass++)
	{
		for(int j=0;j<size-1;j++)
			if(array[j] > array[j+1])
				swap(&array[j],&array[j+1]);
	}
}

void swap(int *element1Ptr, int *element2Ptr)
{
	int hold = *element1Ptr;
	*element1Ptr = *element2Ptr;
	*element2Ptr = hold;
}
```

11.4、陣列應用專題討論（撲克牌的洗牌與交易處理）

在現在的課程中我們要來進行寫作一個洗牌的程式，這個程式在將來可以應用在很多撲克牌的遊戲上。而目前我們用從頭到尾一步一步的方式分解動作來帶領各位寫一個這樣子的程式。

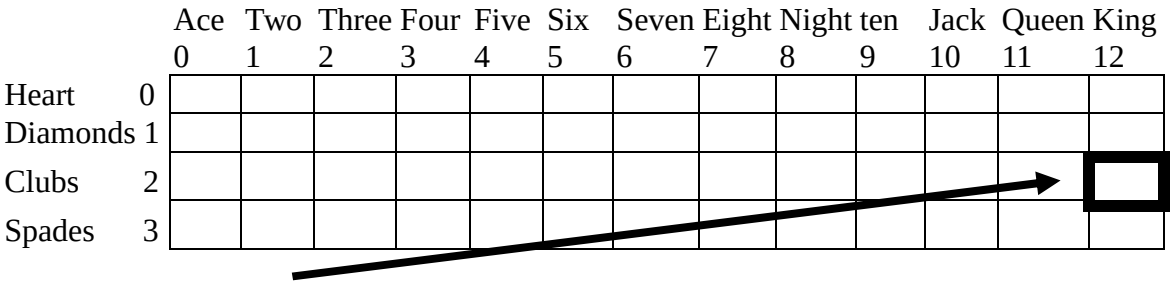
首先有幾個陣列必須要瞭解：

首先 suit[0] = 'H' 'e' 'a' 'r' 't' 's' '\0'

suit[1] = 'D' 'i' 'a' 'm' 'o' 'n' 'd' 's' '\0'

suit[2] = 'C' 'l' 'u' 'b' 's' '\0'

suit[3] = 'S' 'p' 'a' 'd' 'e' 's' '\0'



Deck[2][12] 所呈現的就是 梅花 CLUB KING

Club King

這個洗牌程式首先必須要將 deck 陣列設成為零，然後利用程式將出牌順序依次的填入到這個 DECK 陣列之中，所以以共會有五十二個號碼在這個陣列之中。

而 suit 陣列是用來陳述紙牌的花色，而 face 陣列是用來呈現紙牌的大小。所以剛剛好 suit[row]和 face[column]合起來就是形成類似梅花 2、紅心 9 等等。

寫作程式邏輯

建立 suit 陣列

建立 face 陣列

建立 deck 陣列

洗牌➔就是將隨機數字擺到未使用的 deck 陣列中

出牌➔從陣列中找出牌面並且印出

程式碼在後面^_^

```
#include <iostream.h>
#include <iomanip.h>
#include <time.h>
#include <stdlib.h>

void shuffle(int wDeck[][13]);
void deal(const int wDeck[][13], const char *wFace[], const char *wSuit[]);

void main()
{
     const char *suit[4] = {"Hearts","Diamonds","Clubs","Spades"};
     const char *face[13] =
     { "Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
     "Eight", "Nine", "Ten", "Jack", "Queen", "King"};

     int deck[4][13] = {0};
     srand(time(0));

     shuffle(deck);
     deal(deck,face,suit);
}
void shuffle(int wDeck[][13])
{
     int row, column;
     for(int card=1;card<=52;card++)
     {
          do
          {
               row = rand() % 4;
               column = rand() % 13;
          } while (wDeck[row][column] != 0); //為了避免用到已經用過的元素存放位置

          wDeck[row][column] = card;
     }
}

void deal(const int wDeck[][13], const char *wFace[], const char *wSuit[])
{
     for(int card=1;card<=52;card++)

          for(int row=0;row<=3;row++)

               for(int column=0;column<=12;column++)

                    if (wDeck[row][column] == card)
                         cout << setw(5) << setiosflags(ios::right)
                              << wFace[ column ] << " of "
                              << setw(8) << setiosflags(ios::left)
                              << wSuit[row]
                              << (card % 2 == 0 ? '\n' : '\t' );
}
```

12.1、資料結構定義

瞭解資料結構，並定義使用資料結構。請看以下的程式碼。

首先，基本的資料結構定義的方法如下：

```
struct
{
    變數;
    函數;
};
```

注意，結構的最後一定要加上分號，這是很多人都會犯下的錯！

```
#include <iostream.h>

struct Time
{
    int hour;
    int minute;
    int second;
};

void printStandard( const Time &t );
void printMilitary( const Time &t );

void main()
{
    Time dinnerTime;        //宣告一個 TIME 形態的資料結構 dinnerTime

    dinnerTime.hour = 18;   //為資料結構中的值初始化
    dinnerTime.minute = 30;
    dinnerTime.second = 0;

    cout << "The dinner will br held at ";
    printMilitary( dinnerTime );

    cout << "mlitary time, \nwhich is ";
    printStandard( dinnerTime );
    cout << "standard time.\n";

    //另外設定值給 dinnerTime
    dinnerTime.hour = 29;
    dinnerTime.minute = 73;

    cout << "\nTime with invalid values: " ;
    printMilitary( dinnerTime );
    cout << endl ;
}
void printMilitary( const Time &t )
{
    cout << ( t.hour < 10 ? "0" : "") << t.hour << ":"
         << ( t.minute < 10 ? "0" : "") << t.minute ;
}
```

```
void printStandard( const Time &t )
{
      cout << ((t.hour == 0 || t.hour == 12 ) ? 12 : t.hour % 12 )
           << ":" << (t.minute < 10 ? "0" : "" ) << t.minute
           << ":" << (t.second < 10 ? "0" : "" ) << t.second
           << (t.hour < 12 ? "AM" : "PM" ) ;
}
```

12.2、資料結構之 CLASS

class 和 struct 本身是很類似的功能，不過 CLASS 有 STRUCT 所不能做到的功能，所以我們可以用 CLASS 來完全取代 STRUCT。

```
class TimeClass {
public:
      TimeClass();
      void setTime(int ,int ,int);
      void printMilitary();
      void printStandard();
private:
      int hour;
      int minute;
      int second;
};
```

以上是一個 CLASS 的定義方式。

PUBLIC：是一個標籤，用來區分說這是一個公共的成員，也就是說可以被 C＋＋的程式直接叫用。

PRIVATE：也是一個標籤，不過 PRIVATE 中的成員只能被 CLASS 中的成員所呼叫。

其中有一個很特別的函數叫做 CONTRUCTOR，這個函數的工作是在一個資料被建立的時候執行。

12.2.1、CLASS 的使用

```
#include <iostream.h>

class STUDENT //建立一個結構叫做 STUDENT
{
public:
      int no;
      int age;
};

void main()
{
      STUDENT curtis;

      cout << "請輸入學生編號：";
      cin >> curtis.no ;
```

```
        cout << "請輸入學生年齡：" ;
        cin >> curtis.age ;
        cout << curtis.no << endl
                << curtis.age ;
}
```

12.2.2、globale function & member function 總體函數與成員函數

在 C＋＋中，所有函數的地位都是相等的，不過因為我們現在學到了 CLASS 或 STRUCT 等資料結構，所以我們必須學習到另一種規格的函數，我們稱為 MEMBER FUNCTION，因為這一些函數均存在於結構資中，所以我們才稱他們為成員函數。

```
#include <iostream.h>

class STUDENT //建立一個結構叫做 STUDENT
{
public:
        void print();
        int no;
        int age;
};

//編寫成員函數的本體，這個 PRINT 函數主要的功能就是印出結構的內容
void STUDENT::print()
{
        cout << "[" << no << "," << age << "]" << endl ;
}

void main()
{
        STUDENT curtis;

        cout << "請輸入學生編號：";
        cin >> curtis.no ;
        cout << "請輸入學生年齡：" ;
        cin >> curtis.age ;

        curtis.print();          //叫用成員函數
}
```

12.2.3、public 與 private 標籤

就在我前面的講義中就說明過，PUBLIC 標籤是所謂的公共成員，這 PUBLIC 下的所有東西都可以被其他函數直接呼叫，而 PRIVATE 標籤中必須使用 PUBLIC 中有定義到的 MEMBER FUNCTION 才能讀取。這樣子一來可以避免資料不小被其他函數破壞。

```
#include <iostream.h>

class STUDENT //建立一個結構叫做 STUDENT
{
public:
	void print();
	void input(int,int);
private:
	int no;
	int age;
};

//編寫成員函數的本體，這個 PRINT 函數主要的功能就是印出結構的內容
void STUDENT::print()
{
	cout << "[" << no << "," << age << "]" << endl ;
}

//編寫成員函數的本體，這個函數主要的功能是去輸入學生的資料
//而且因為現在 no 和 age 變數都是歸類到 PRIVATE 的標籤之下,
//所以必須透過 MEMBER 才能讀寫。

void STUDENT::input(int i_no, int i_age)
{
	no = i_no;
	age = i_age;
}

void main()
{
	STUDENT curtis;
	int age,no ;

	cout << "請輸入學生編號：";
	cin >> no ;
	cout << "請輸入學生年齡：" ;
	cin >> age ;

	curtis.input(no, age);	//叫用成員函數 input
	curtis.print();		//叫用成員函數 print
}
```

12.2.4、constructor 建構函數

所謂 constructor 就是在資料結構一被建立的時候，就會自動叫用 constructor 函數，例如，當我們建立一個學生資料結構的同時，我們可以先將學生的所有資訊都先設定成為 0。這個時候我們就必須使用到 contructor 了！

```
#include <iostream.h>

class STUDENT //建立一個結構叫做 STUDENT
{
public:
	STUDENT();  //建構函數  constructor
	void print();
	void input(int,int);
private:
	int no;
	int age;
};

//編寫建構函數
//目前這個建構函數的功能主要就是當結構一被創立的時候自動賦予
//其 private 成員中的所有變數的初始值為零。
STUDENT::STUDENT()
{
	no = 0;
	age = 0;
}


//編寫成員函數的本體，這個 PRINT 函數主要的功能就是印出結構的內容
void STUDENT::print()
{
	cout << "[" << no << "," << age << "]" << endl ;
}

//編寫成員函數的本體，這個函數主要的功能是去輸入學生的資料
//而且因為現在 no 和 age 變數都是歸類到 PRIVATE 的標籤之下,
//所以必須透過 MEMBER 才能讀寫。

void STUDENT::input(int i_no, int i_age)
{
	no = i_no;
	age = i_age;
}

void main()
{
	STUDENT curtis;
	int age,no ;
```

```
        cout << "目前結構中的初始值為：";
        curtis.print();
        cout << "請輸入學生編號：";
        cin >> no ;
        cout << "請輸入學生年齡：" ;
        cin >> age ;

        curtis.input(no, age);      //叫用成員函數 input
        curtis.print();             //叫用成員函數 print
}
```

12.3、利用 CLASS 來設計一個完整的應用

我們將綜合以上用到的所有技巧來設計一個 CLASS 的應用程式。

請看下面的程式碼：

```
#include <iostream.h>

class TimeClass{
public:
        TimeClass();
        void setTime(int, int, int);
        void printMilitary();
        void printStandard();
private:
        int hour;
        int minute;
        int second;
};

TimeClass::TimeClass(){ hour=minute=second=0;}

void TimeClass::setTime(int h,int m,int s)
{
        hour = (( h>=0 && h<24) ? h : 0);
        minute = ( m>=0 && m< 60 ) ? m : 0;
        second = ( s>=0 && s< 60 ) ? s : 0;
}

void TimeClass::printMilitary()
{
        cout << ( hour < 10 ? "0" : "") << hour << ":"
                << ( minute < 10 ? "0" : "") << minute ;
}

void TimeClass::printStandard()
{
        cout << ((hour == 0 || hour == 12 ) ? 12 : hour % 12 )
                << ":" << (minute < 10 ? "0" : "" ) << minute
```

```
            << ":" << (second < 10 ? "0" : "" ) << second
            << (hour < 12 ? "AM" : "PM" ) ;
}

void main()
{
      TimeClass t;          //  建立一個結構叫 t

      cout << "The initial militay time is: ";
      t.printMilitary();
      cout << "\nThe initial standard time is: ";
      t.printStandard();

      t.setTime(13,27,6);

      cout << "\n\nThe militay time after set is: ";
      t.printMilitary();
      cout << "\nThe standard time after set is: ";
      t.printStandard();

      t.setTime(99,99,99);

      cout << "\n\nThe militay time after invalid set is: ";
      t.printMilitary();
      cout << "\nThe standard time after invalid set is: ";
      t.printStandard();

      cout << endl ;

}
```

12.4、CLASS 實做與應用

試寫一個 CLASS **名稱為：**GOOD**，這個** GOOD **中包含了三個** private **成員，第一個成員是貨品編號** no **形態為整數，第二個成員是貨品價格** price **為整數形態，第三個成員為貨品數量** quan **為整數。每個結構內的變數初始直都要為零。**

1. **其中妳必須要寫一個** constructor**，用來將結構內的成員設定初始值，每個結構內的變數初始直都要為零。**
2. **皆下來寫一個** input **成員函數，這個函數用來將使用輸入的值擺到結構中的變數** no,quan,price**。**
3. **寫一個** print **的成員函數，將結構內的變數輸出在螢幕上。**
4. **寫一個** cal **函數用來輸出總金額，使用者可以輸入折數，函數會將傳回打折過後的金額。假設如果輸入超過** 1 **的話，則將打折變數的值設定成為** 0

13.1、CONST（CONSTANT）物件與 CONST 成員函數（MEMBER FUNCTION）

在程式設計的過程中，妳會建立一些資料結構，我們現在通稱這些東西為物件（OBJECT），而在設計過程妳會發現有些物件的內容必須要改變，有些則不需要改變，就如十二點整是一個吃飯時間，這樣一段話就表示是不可以被更改的，如果妳更改了則 C＋＋就會告訴你語法錯誤。

const Time noon(12,0,0);
指定一個常態的 TIME 形態的物件 NOON 並設定初始值。

C＋＋允許成員函數去呼叫這個物件，不過這個成員函數本身也必須是要 CONST 形態。

下面這一個程式範例會導致語法錯誤。

```
#include <iostream.h>
#include "time5.h"

//注意開始建構函示就賦予了初始值零。
Time::Time(int hr,int min,int sec){setTime(hr,min,sec);}

void Time::setTime(int h,int m,int s)
{
      setHour(h);
      setMinute(m);
      setSecond(s);
}

void Time::setHour(int h)
{ hour = ( (h>=0 && h<=24) ? h:0 ); }

void Time::setMinute(int m)
{ minute = ( (m>=0 && m<60) ? m:0); }

void Time::setSecond(int s)
{ second = ( (s>=0 && s<60) ? s:0); }

int Time::getHour() const { return hour;}

int Time::getMinute() const { return minute;}

int Time::getSecond() const { return second; }

void Time::printMilitary() const
{
      cout << ( hour < 10 ? "0" : "") << hour << ":"
      << ( minute < 10 ? "0" : "") << minute ;
}

void Time::printStandatd()
{
```

```
        cout << ( (hour == 12) ? 12: hour % 12) << ":"
             << ( minute < 10 ? "0" : "") << minute << ":"
             << ( second < 10 ? "0" : "") << second << ":"
             << ( hour < 12 ? "AM" : "PM" );
}

//主程式在這裡
void main()
{
        Time wakeUp(6,45,0);          //non-constant object
        const Time noon(12 , 0, 0);   //constant object

                                          //Member Function      Object
        wakeUp.setHour(18);               //non-const                 non-const

        noon.setHour(12);             //non-const                 const

        wakeUp.getHour();             //const                     non-const

        noon.getMinute();             //const                     const
        noon.printMilitary();         //const                     const
        noon.printStandatd();         //non-const                 const
}
```

這個程式一旦執行就會被發現兩個錯誤

```
Compiling...
paper13_1.cpp
D:\C++\taiwan\test_for_paper13\paper13_1.cpp(52) : error C2662: 'setHour' :
cannot convert 'this' pointer from 'const class Time' to 'class Time &'
                    Conversion loses qualifiers



D:\C++\taiwan\test_for_paper13\paper13_1.cpp(58) : error C2662:
'printStandatd' : cannot convert 'this' pointer from 'const class Time' to 'class
Time &'
        Conversion loses qualifiers
Error executing cl.exe.

test_for_paper13.exe - 2 error(s), 0 warning(s)
```

首先出現在 noon.setHour（）中，因為 noon 物見識 CONST 形態，而 setHour（）函數是 NON-CONST 形態，所以導致錯誤。

再來是 noon.printStandard()中，所發生的問題和上一個一樣。所以 CONST 物件一定要被 CONST 形態的 MEMBER FUNCTION 存取。

13.2、**運算子覆載**(operator overloading)

前面我們已經提過，一個變數的形態指定這個變數可以儲存的值，並且可以對這變數中的值進行運算處理。在這一章節中，妳將定義一個運算子。

下面的運算子是不可以被覆載的（. . ＊ :: ?: sizeof）

下面是一個不用運算子覆載的方式來做字串的加減。

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

class string
{
public:
	string(char *); //Constructor
	void str_append(char *);
	void chr_minus(char);
	void show_string(void);
private:
	char data[256];
};

string::string(char *str)
{
	strcpy(data,str);
}

void string::str_append(char *str)
{
	strcat(data,str);
}

void string::chr_minus(char letter)
{
	char temp[256];
	int i,j;
	for(i=0,j=0;data[i];i++)
		if(data[i] != letter)
			temp[j++] = data[i];
	temp[j] = NULL;
	strcpy(data,temp);
}

void string::show_string()
{
	cout << data << endl ;
}

void main()
{
```

```
	string title("Rescued By C++");
	string lesson("Understanding Operator Overloading");

	title.show_string();
	title.str_append(" rescued me!");
	title.show_string();
	lesson.show_string();
	lesson.chr_minus('n');
	lesson.show_string();
}
```

接下來是使用運算子覆載的情況

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

class string
{
public:
	string(char *); //Constructor
	void operator +(char *);
	void operator -(char);
	void show_string(void);
private:
	char data[256];
};

string::string(char *str)
{
	strcpy(data,str);
}

void string::operator +(char *str)
{
	strcat(data,str);
}

void string::operator -(char letter)
{
	char temp[256];
	int i,j;
	for(i=0,j=0;data[i];i++)
		if(data[i] != letter)
			temp[j++] = data[i];
	temp[j] = NULL;
	strcpy(data,temp);
}

void string::show_string()
{
	cout << data << endl ;
```

```
}

void main()
{
	string title("Rescued By C++");
	string lesson("Understanding Operator Overloading");

	title.show_string();
	title + " rescued me!";
	title.show_string();

	lesson.show_string();
	lesson - 'n';
	lesson.show_string();
}
```

13.3、覆載輸出的運算子

```
#include <iostream.h>

struct XY
{
	void setValue(int ,int);

	int x;
	int y;
};

void XY::setValue(int a,int b)
{
	x=a;
	y=b;
}

ostream &operator <<(ostream &ooo,const XY &t)
{
	ooo << "[" << t.x << "," << t.y << "]" << endl ;
	return ooo ;
}

void main()
{
	XY test1;

	test1.setValue(1,1);
	cout << test1;
}
```