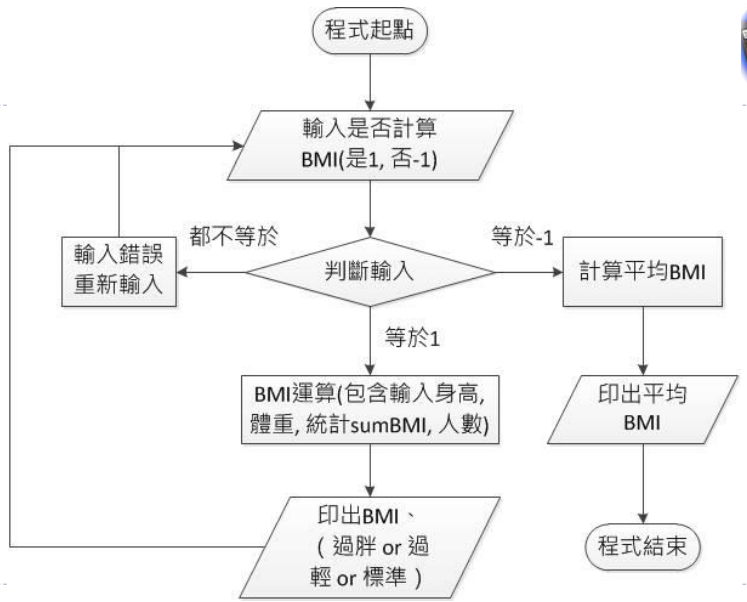
程式設計-第三~四章

洪麗玲

上週作業

- 請寫一支幫忙計算BMI值的程式
  - 先說明程式要做什麼事，並適當指引要做什麼輸入
  - 收到輸入後要再輸出收到的資料以進行確認
  - BMI計算完後輸出計算結果並告知其值是否在標準範圍內
  - 請算出這些被算出的BMI平均值
- 其他要求參照課程要求”程式註解”

Wireless Access Technologies & Software Engineering



整理同學犯的錯誤

- #include <stdio.h>　拼字錯誤
- 資料初始值
  - i = 0　or　i = input/3
  - i = i + k;　(int i, k;　未給定初始值)
- 未讀進資料
  - scanf (" %d, &A");　➔ scanf(" %d", &A);
  - 宣告與讀入的型別不對　int a;　scanf(" %f", &a);

Wireless Access Technologies & Software Engineering

```
1  // Fig. 3.8: fig03_08.c
2  // Class-average program with sentinel-controlled repetition.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main( void )
7  {
8     unsigned int counter; // number of grades entered
9     int grade; // grade value
10    int total; // sum of grades
11
12    float average; // number with decimal point for average
13
14    // initialization phase
15    total = 0; // initialize total
16    counter = 0; // initialize loop counter
17
18    // processing phase
19    // get first grade from user
20    printf( "%s", "Enter grade, -1 to end: " ); // prompt for input
21    scanf( "%d", &grade ); // read grade from user
22
```

圖3.8　以警示值控制重複結構來解決全班平均問題(1/2)

```
23    // loop while sentinel value not yet read from user
24    while ( grade != -1 ) {
25       total = total + grade; // add grade to total
26       counter = counter + 1; // increment counter
27
28       // get next grade from user
29       printf( "%s", "Enter grade, -1 to end: " ); // prompt for input
30       scanf("%d", &grade); // read next grade
31    } // end while
32
33    // termination phase
34    // if user entered at least one grade
35    if ( counter != 0 ) {
36
37       // calculate average of all grades entered
38       average = ( float ) total / counter; // avoid truncation
39
40       // display average with two digits of precision
41       printf( "Class average is %.2f\n", average );
42    } // end if
43    else { // if no grades were entered, output message
44       puts( "No grades were entered" );
45    } // end else
46 } // end function main
```

圖3.8　以警示值控制重複結構來解決全班平均問題(2/2)

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```

```
Enter grade, -1 to end: -1
No grades were entered
```

圖3.8　以警示值控制重複結構來解決全班平均問題

- 明確地類型與隱含地類型轉換
- 第38行

```
average = ( float ) total / counter;
```

- 含有 (**float**) 這個強制型別轉換運算子，它會為它的運算元**total**產生一個暫時的浮點數拷貝。而存放在**total**的值仍然是個整數。以這種方式來使用強制型別轉換運算子稱為**明確地轉換 (explicit conversion)**。

– 格式化浮點數

- 圖3.8的程式使用**printf**的轉換指定詞 **%.2f** (第41行) 來印出**average**的值。其中**f**表示將會有個浮點數要被列印，而 **2**則指定了此值被列印時的**精準度 (precision)**。如果我們用的是 **%f**這個轉換指定詞 (沒有指定精準度)，那麼列印時將使用**預設精準度 (default precision)** 6——就如同我們使用 **% 6f**這樣的轉換指定詞。

- 當列印浮點數時，其值會**被四捨五入 (rounded)** 成所指定的精準度。在記憶體內的值未被改變。如下兩個敘述式執行後，將會印出3.45和3.4。

```
printf( "%.2f\n", 3.446 ); // prints 3.45
printf( "%.1f\n", 3.446 ); // prints 3.4
```

4.11 對於相等運算子 (= =) 和指定運算子 (=) 常見的混淆

- 不管是有沒有經驗的C程式設計師，都很容易犯一種錯誤。我們覺得有必要用一個章節來討論這種錯誤。這種錯誤就是：不小心地混用 == (相等) 運算子和 = (指定) 運算子。這種混用為什麼這麼危險呢？原因在於它通常不會造成編譯錯誤。含有這種錯誤的程式通常可正確地編譯，但程式執行的結果卻可能會因執行時的邏輯錯誤而不正確。
- 引起這個問題的原因是來自於C的兩項特性。第一，C中可以產生值的任何運算式，都可使用在任何控制敘述式的判斷部分裡。如果值是零，便當做偽。如果值不是零，則當做真。第二，C的指定動作會產生值，此值便是指定給指定運算子左邊之變數的值。

- lvalues與rvalues
  - 當我們在撰寫條件式時，最好會將變數名稱寫在左邊，而將常數寫在右邊，如**x == 7**。如果將這個習慣改成常數在左邊，變數在右邊，也就是**7 == x**的話，當你不小心將 == 誤寫成 = 時，編譯器便會將這個錯誤找出來。編譯器會認為這是一個語法錯誤，因為只有變數名稱才能放在指定運算子的左邊。這樣子可避免因不小心所造成的執行時邏輯錯誤。
  - 變數名稱是個**lvalue** (「left value」，即***左邊數值***)，因為他們可放在指定運算子的左邊。常數是個**rvalue** (「right value」，即***右邊數值***)，因為他們只能放在指定運算子的右邊。**lvalues**也可以用做**rvalues**，但反過來則不行。

– 在獨立敘述式中容易混淆的 == 與 =

另外一種情況是將 **=** 誤寫成 **==**。假設原先你想要把一個值設定給變數，如下列的簡單敘述式：

```
x = 1;
```

- 卻不小心地寫成

```
x == 1;
```

- 這不會是個語法錯誤。編譯器只會把它當成是一個比較運算式。如果**x**等於**1**的話，則此條件為真並且運算式會傳回值1。如果**x**不等於**1**，則此條件為偽而運算式會傳回值0。但不論傳回什麼數值，由於這裡沒有指定運算子，數值就只是消失了。因此**x**的值不會改變，這可能會造成一個執行時的邏輯錯誤。

3.11 指定運算子

- C提供了數種指定運算子，使得指定運算式可以縮寫。例如，以下的敘述

```
c = c + 3;
```

- 可利用**加法指定運算子 += (addition assignment operator +=)**縮寫成

```
c += 3;
```

- += 運算子會把在此運算子右邊的運算式的值，加上此運算子左邊變數的值，然後將結果存到運算子左邊的變數。

- 任何如下面格式的敘述式

```
variable = variable operator expression;
```

- 可以寫成下面的格式

```
variable operator= expression;
```


指定運算子	範例運算式	展開式	指定值
假設： `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
+=	`c += 7`	`c = c + 7`	10 到 c
-=	`d -= 4`	`d = d - 4`	1 到 d
*=	`e *= 5`	`e = e * 5`	20 到 e
/=	`f /= 3`	`f = f / 3`	2 到 f
%=	`g %= 9`	`g = g % 9`	3 到 g

圖3.11　算術指定運算子

Wireless Access Technologies & Software Engineering

3.12 遞增和遞減運算子

- C還提供了**單元遞增運算子++ (increment operator)** 和**單元遞減運算子-- (decrement operator)**。

運算子	範例運算式	說明
++	++a	先將 a 遞增 1，再以 a 的新值進行運算
++	a++	以 a 目前的值進行運算，再將 a 遞增 1
--	--b	先將 b 遞減 1 再以 b 的新值進行運算
--	b--	以 b 目前的值進行運算，再將 b 遞減 1

3.12 遞增和遞減運算子

- 圖3.13的程式示範了前置遞增與後置遞增運算子的差異。

```
1  // Fig. 3.13: fig03_13.c
2  // Preincrementing and postincrementing.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main( void )
7  {
8     int c; // define variable
9
10    // demonstrate postincrement
11    c = 5; // assign 5 to c
12    printf( "%d\n", c ); // print 5
13    printf( "%d\n", c++ ); // print 5 then postincrement
14    printf( "%d\n\n", c ); // print 6
15
16    // demonstrate preincrement
17    c = 5; // assign 5 to c
18    printf( "%d\n", c ); // print 5
19    printf( "%d\n", ++c ); // preincrement then print 6
20    printf( "%d\n", c ); // print 6
21 } // end function main
```

```
5
5
6

5
6
```

運算子	結合性	形式
++ (後置) -- (後置)	由右至左	後置
+ - (type) ++ (前置) -- (前置)	由右至左	單元性
* / %	由左至右	乘法
+ -	由左至右	加法
< <= > >=	由左至右	關係
== !=	由左至右	相等
?:	由右至左	條件
= += -= *= /= %=	由右至左	設值

圖3.14 到目前為止所介紹之運算子的運算優先順序

4.1 簡介

- 現在你應該能自在地撰寫簡單的完整C程式了。本章中將會更詳細地考慮有關重複的概念，並且會介紹**for**和**do…while**等兩種用來控制重複的敘述式。也將會介紹**switch**多重選擇敘述式。我們會討論直接和迅速離開某種控制敘述式的**break**敘述式，以及用來跳過重複敘述式本體剩餘部分的**continue**敘述式。

4.2 重複的基本概念

- 大部分的程式都包含了重複的動作或是迴圈 (looping)。迴圈是指當某一個**迴圈繼續條件式(loop-continuation condition)** 為真時，電腦會重複執行的一群指令。我們已經討論過以下這兩種重複的方法：
  1. 計數器控制的重複
  2. 警示值控制的重複
- 在計數器控制的重複當中，必須使用**控制變數 (control variable)** 來計算重複的次數。每次這群指令執行之後，就會遞增控制變數 (通常是1)。當控制變數的值顯示已經執行了正確的重複次數時，就會結束此迴圈，重複敘述式之後的敘述式會繼續執行。
- 在下列兩種情形時，必須用警示值來控制重複的動作：
  1. 不能事先知道重複的次數，以及
  2. 迴圈內含有一個每次迴圈執行時都會取得資料的敘述式。

4.3 計數器控制的重複

- 計數器控制的重複需要有：
  1. 控制變數 (或迴圈計數器) 的**名稱 (name)**。
  2. 控制變數的**初始值 (initial value)**。
  3. 每一次迴圈執行的時候控制變數的**遞增量 (increment)** 或**遞減量 (decrement)**。
  4. 判斷控制變數是否是**終止值 (final value)** 的條件 (也就是檢查迴圈是否應該繼續)。
- 考慮圖4.1列出的簡單程式，該程式會印出從1到10的整數。

```c
// Fig. 4.1: fig04_01.c
// Counter-controlled repetition.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   unsigned int counter = 1; // initialization

   while ( counter <= 10 ) { // repetition condition
      printf ( "%u\n", counter ); // display counter
      ++counter; // increment
   } // end while
} // end function main
```

```
1
2
3
4
5
6
7
8
9
10
```


4.4 for重複敘述式

- **for**重複敘述式會自動處理計數器控制式重複的所有細節。為了說明**for**的強大威力，讓我們重新撰寫圖4.1的程式。輸出結果見圖4.2。

```c
// Fig. 4.2: fig04_02.c
// Counter-controlled repetition with the for statement.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   unsigned int counter; // define counter

   // initialization, repetition condition, and increment
   //  are all included in the for statement header.
   for ( counter = 1; counter <= 10; ++counter ) {
      printf( "%u\n", counter );
   } // end for
} // end function main
```

圖 4.2 以**for**敘述式製作的計數器控制式重複

– for敘述式標頭的組成
  • 圖4.3仔細檢視圖4.2中的**for**敘述式。注意到**for**敘述式「做了每件事」──**for**敘述式指定了含有控制變數的計數器控制重複結構所需要的每個項目。如果**for**本體內的敘述式超過一個，請將他們用大括號包含起來。

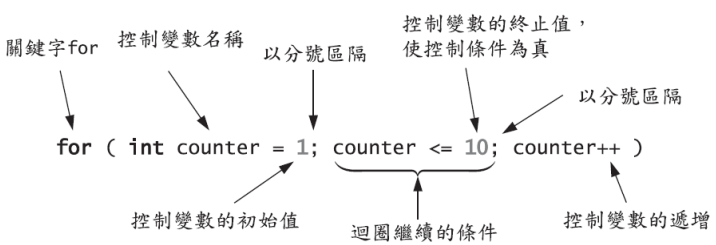


圖4.3 **for**敘述式標頭的組成元件

– 誤差為一的錯誤
  • 請注意圖4.2使用迴圈繼續條件**counter <= 10**。如果你將它誤寫為**counter < 10**，則此迴圈只會執行9次。這是一種常見的邏輯錯誤，稱為**誤差為一的錯誤** (off-by-one error)。
– for敘述式的一般格式
  • **for**敘述式的一般格式如下

```
for ( expression1; expression2; expression3 ) {
   statement
}
```

其中的expression1為迴圈的控制變數指定初始值，expression2是迴圈的繼續條件，而expression3則會遞增控制變數。

- 逗號分隔的連續運算式
  - 通常expression1和expression3是由逗號分隔的一連串運算式。逗號在這裡是當成**逗號運算子 (comma operators)**，它確保這一連串的運算式會由左到右執行運算。
  - 這一連串由逗號所分隔的運算式，其型別和數值是由逗號分隔的一連串運算式中最右邊的運算式來決定。逗號運算子最常用在**for**敘述式中。它的主要功用是讓程式設計者能夠使用多重的初始值指定和 (或) 多重的遞增運算式。例如在同一個**for**敘述式裡，可能會有兩個控制變數必須指定初始值和遞增。
- for敘述式裡標頭運算式是可有可無的
  - **for**敘述式裡的三個運算式都是可有可無的。如果我們省略了expression2的話，則C會認為控制條件永遠為真，因而建立一個無窮迴圈。如果控制變數已在程式其他位置設定好了初始值，則我們可以省略expression1。如果遞增動作在迴圈本體內的敘述式當中執行，或是不需要遞增動作，則expression3便可省略。

- 遞增表示式如同一個獨立敘述式
  - 下列各運算式

```
counter = counter + 1
counter += 1
++counter
counter++
```

  - 對**for**敘述式的遞增部分來說是一樣的。有些C程式設計師較偏好使用**counter++**，因為遞增動作是在迴圈本體執行後才進行的。所以用後置遞增格式似乎較為自然。最後提醒一點，**for**敘述式中的兩個分號是不能省略的。

4.5 for敘述式：要注意的事項以及提示

1. 初始值指定、迴圈繼續條件和遞增的部分都可以包含算術運算式。例如，假設**x=2**、**y=10**，底下的敘述式

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

和以下的敘述式是相等的。

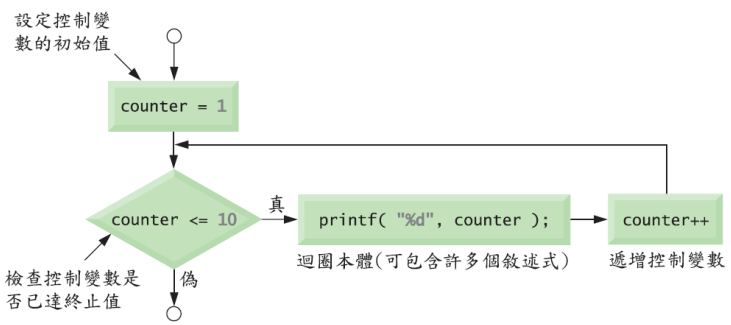
```
for ( j = 2; j <= 80; j += 5 )
```

2. 「遞增量」可以是負的(在這種情況下是為遞減，亦即迴圈是往下計數的)。
3. 如果迴圈繼續條件一開始為偽，則迴圈的本體部分將不會執行。程式接著會由**for**敘述式之後的第一個敘述式開始執行。
4. 控制變數經常會由迴圈本體列印或使用，但並非一定要這麼做。我們也常只用控制變數來控制重複的次數，而在迴圈的本體內並不會使用它。

5. **for**敘述式的流程圖十分類似**while**敘述式。例如，圖4.4展示了下列敘述式的流程圖

```
for ( counter = 1; counter <= 10; ++counter )
   printf( "%u", counter );
```

- 這張流程圖清楚地表示初始值指定動作只進行一次，以及遞增發生在本體敘述式執行之後。



4.6 使用for敘述式的例子

- 下列的範例展示了用來改變**for**敘述式內控制變數的幾種方法。
  1. 將控制變數從1遞增到100 (遞增量為1)。

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

  2. 將控制變數從**100**變到**1**，遞增量為**-1** (也就是遞減量為1)。

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

  3. 將控制變數從**7**遞增到**77**，每次遞增**7**。

```
for ( i = 7; i <= 77; i += 7 )
```

4. 將控制變數從**20**變到**2**，每次遞增**-2**。

```
for ( i = 20; i >= 2; i -= 2 )
```

5. 將控制變數按照下列數列進行改變：**2**、**5**、**8**、**11**、**14**、**17**。

```
for ( j = 2; j <= 17; j += 3 )
```

6. 將控制變數按照下列數列進行改變：**44**、**33**、**22**、**11**、**0**。

```
for ( j = 44; j >= 0; j -= 11 )
```


– 應用：計算由2到100的偶數總和

- 圖4.5的程式利用**for**敘述式算出由**2**到**100**所有偶數的和。迴圈每次執行，都會把控制變數的值加到變數sum。

*W*ireless *A*ccess *T*echnologies & *S*oftware *E*ngineering

```
1  // Fig. 4.5: fig04_05.c
2  // Summation with for.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main( void )
7  {
8     unsigned int sum = 0; // initialize sum
9     unsigned int number; // number to be added to sum
10
11    for ( number = 2; number <= 100; number += 2 ) {
12       sum += number; // add number to sum
13    } // end for
14
15    printf( "Sum is %u\n", sum ); // output sum
16 } // end function main
```

```
Sum is 2550
```

圖4.5　使用**for**將數字加總