GLOBAL EDITION

Computer Programming and Practice (Ⅰ)
計算機程式設計與實習(一)
- Ch03 -

109年上學期
國立臺南大學 電機工程系
梁家銘

3 結構化程式的開發

3.1 簡介

- 準備撰寫程式解決某個問題前，須充分了解問題，並仔細規劃解決此問題之方法。
  - 本章將介紹有關開發結構化電腦程式設計及技巧。

3.2 演算法(Algorithms)

- 任何計算問題，皆可歸納以特定順序(order)來執行一系列動作。
- 因此，我們可列出解決問題之**程序 (procedure)** 如下：
  1.將要執行的**動作 (actions)**
  2.執行這些動作的**順序 (order)**。
- 這就稱為是**演算法 (algorithm)**。
  - 註：可視為有順序性的執行步驟。

3.3 虛擬程式碼

- **虛擬程式碼 (Pseudocode)** 是一種方便開發者規劃的非正規語言(類似程式草稿)，以協助設計演算法。
  - 有助於轉換成結構化C程式之演算法。
- 虛擬程式碼完全由文字所組成，因此可以利用文書編輯軟體輕易地撰寫。
- 虛擬程式碼只包含動作敘述
  - 當程式由虛擬碼轉換成C程式後被執行部分。
  - 宣告部分 (非可執行敘述)-供編譯器看的簡易訊息。

3.4 控制結構

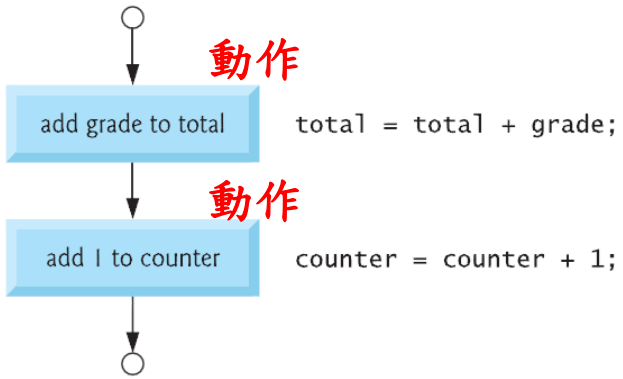
- 程式中的敘述式(statement)是根據在程式中之順序依序地被執行。
  - 此稱為**循序式執行(sequential execution)**。
- 有些C程式敘述式能指定下一個執行的敘述式(非依序式)。
  - 此稱為**控制權移轉 (transfer of control)**。
- 根據Bohm和Jacopini (兩位學者)的研究指出，所有程式均可由三種**控制結構 (control structure)** 撰寫，分別為：
  - 1) 循序結構(sequence structure)
  - 2) 選擇結構 (selection structure)
  - 3) 重複結構 (repetition structure)。
- 其中，循序結構是C語言內建特性，一般電腦會自動地按照撰寫的C敘述式之順序，逐行執行，除非變更程式執行流程。
  - 註：結構化的程式較清楚、易除錯及不易產生錯誤。

(精簡程式碼：很重要!)

1. 流程圖 (flowchart)

- **流程圖**是整個演算法或部分演算法的圖形表示法。
- 流程圖使用具有特殊涵義的標誌繪製，像是
  - 1) 矩形 □：執行動作(如：計算)
  - 2) 菱形 ◇：條件判斷
  - 3) 圓角矩形 □：演算法開始或結束 (通常標示「Begin」及「End」)
  - 4) 圓形 ○：片段程式之開始或結束
- 這些標誌以稱為**流向(flowline)**的箭頭→連接起來。
  - 其中，**菱形符號(diamond symbol)**最重要，它被稱為**判斷符號(decision symbol)**，表示將進行某項判斷。

練習看看!



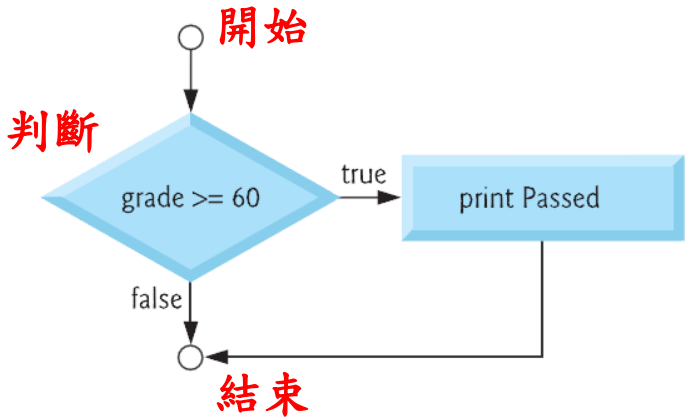
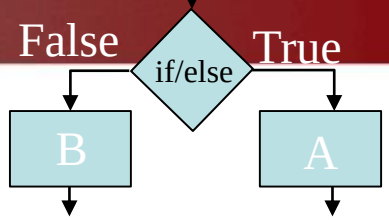


圖3.1　C語言依序結構的流程圖



2. 選擇結構(Selection Statements)

- C語言提供了三種選擇結構：
  1) **if** 選擇結構(單選)：當條件為真/成立(True)時，執行某項動作；反之，當條件為偽/不成立(False)時，跳過(不執行動作)。
  2) **if...else** 選擇敘述式(雙選)：當條件為真/成立(True)時，執行某項動作；反之，當條件為偽/不成立(False)時，執行另一項動作。
  3) **switch** 選擇敘述式(多選)：依符合的內容，選擇執行許多動作中的其中一項。

3. 重複敘述(Iteration Statements)

- C語言提供三種重複結構(迴圈結構)，分別是**while迴圈**、**do...while迴圈**和**for迴圈**(在第4章介紹)。
- 因此，C語言共有七種控制敘述式：
  - 循序、三種選擇和三種重複，C程式可根據演算法需要，應用並組合這七種結構。

3.5 if 選擇敘述式(單選)

- if(...) 選擇敘述：可用來選取不同功能動作。
  - 例如：假設考試中及格成績為60分，以下虛擬碼敘述式

If students grade is greater than or equal to 60
Print Passed

寫法：
if(...){...}

  - 意思：將判斷條件「student's grade is greater than or equal to 60」是真(True)或偽(False)。
  - 若為真(成立)，則印出「Passed」，然後繼續下個虛擬碼敘述式。
  - 若條件為偽(不成立)，則不執行動作，然後繼續下個虛擬碼敘述式。

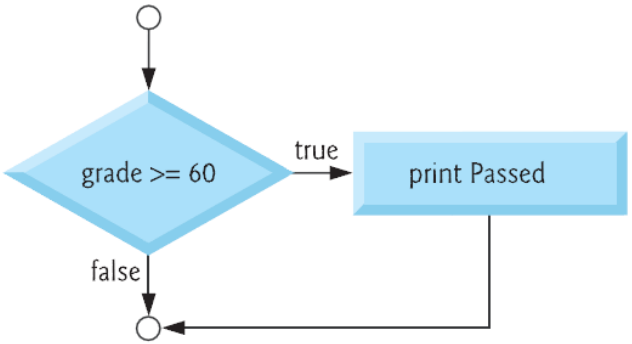


圖3.2 (單一選擇)if敘述式的流程圖

```
if ( grade >= 60 ) {
   puts( "Passed" );
} // end if
```

C程式

註：{...敘述式...}

3.6 if…else選擇敘述式(雙選)

- **if…else** 選擇敘述式：可根據條件真、偽，來執行兩種不同動作。
  - 例如：下方虛擬碼敘述式

If students grade is greater than or equal to 60
Print Passed
else
Print Failed

寫法：
if(…){…}
else {…}

  - 當學生成績高於或等於60時，印出**Passed**；而小於60時，則印出**Failed**，對應兩種不同動作「執行」。
- 下方流程圖表示**if…else**結構的控制流程。
  - 其中，矩形為動作，而菱形為判斷。

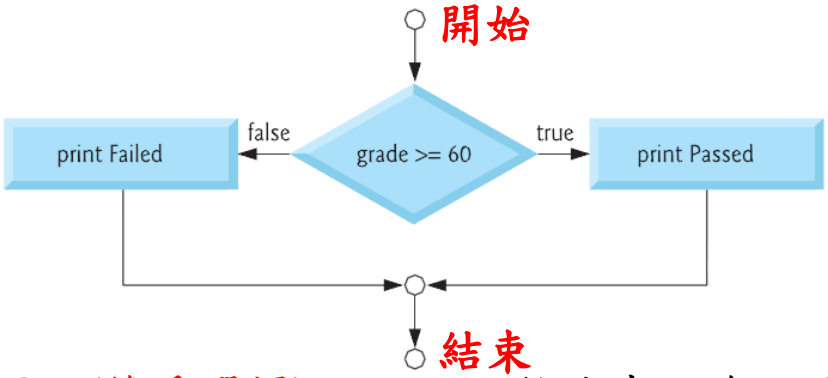


圖3.3 (雙重選擇)if…else敘述式之流程圖

```
if ( grade >= 60 ) {
   puts( "Passed" );
} // end if
else {
   puts( "Failed" );
} // end else
```

C程式

- 註：C語言也提供if…else敘述式的簡化運算子"… ? … : …"，例如：判斷條件?成立動作:不成立動作，範例如下：

```
grade >= 60 ? puts( "Passed" ) : puts( "Failed" );
```

4. 巢狀的if...else敘述式

- 透過將**if...else** 敘述插入另一個**if...else**敘述裡，構成**巢狀的if...else 敘述式 (nested if...else statements)**， //一層一層
  - 用以建構多重選擇之狀況。
  - 例如：以下虛擬碼會在考試成績大於等於**90**時印出**A**；大於等於**80**（但小於**90**）時印出**B**；大於等於**70**（但小於**80**）時印出**C**；大於等於**60**（但小於**70**）時印出**D**；而其他成績則印出**F**。

看看虛擬碼:

```
If student's grade is greater than or equal to 90
    Print "A"
else
    If student's grade is greater than or equal to 80   //80~89
        Print "B"
    else
        If student's grade is greater than or equal to 70   //70~79
            Print "C"
        else
            If student's grade is greater than or equal to 60   //60~69
                Print "D"
            else
                Print "F"   // <59
```

- 此虛擬碼可撰寫成C程式(如右圖)

條件

```
if ( grade >= 90 ) {
   puts( "A" );
} // end if
else {
   if ( grade >= 80 ) {
      puts( "B" );
   } // end if
   else {
      if ( grade >= 70 ) {
         puts( "C" );
      } // end if
      else {
         if ( grade >= 60 ) {
            puts( "D" );
         } // end if
         else {
            puts( "F" );
         } // end else
      } // end else
   } // end else
} // end else
```

- 若變數**grade**大於等於90時，只有第一個{...}內的**puts**敘述會執行。
  - 當此**puts**執行後，最「外層」之**if...else**敘述的**else**部分，將不被執行(直接跳過)。
  - 註：也可將前述**if...else**敘述改寫成

```
if ( grade >= 90 ) {
   puts( "A" );
} // end if
else if ( grade >= 80 ) {
   puts( "B" );
} // end else if
else if ( grade >= 70 ) {
   puts( "C" );
} // end else if
else if ( grade >= 60 ) {
   puts( "D" );
} // end else if
else {
   puts( "F" );
} // end else
```

(對照前例)

```
if ( grade >= 90 ) {
   puts( "A" );
} // end if
else {
   if ( grade >= 80 ) {
      puts( "B" );
   } // end if
   else {
      if ( grade >= 70 ) {
         puts( "C" );
      } // end if
      else {
         if ( grade >= 60 ) {
            puts( "D" );
         } // end if
         else {
            puts( "F" );
         } // end else
      } // end else
   } // end else
} // end else
```

  - 此兩種方式是相同的，但後者較好(可讀性較高)是由於其可避免過深縮排(程式向右傾)。

3.7 while重複敘述式

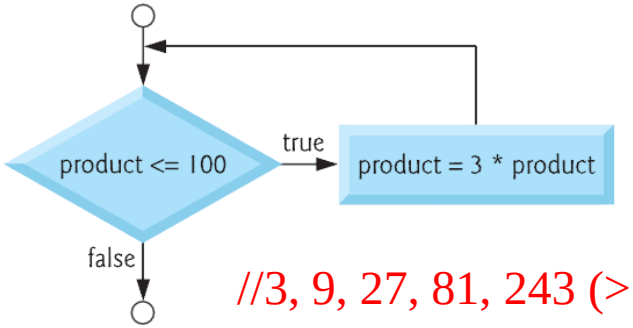
- **重複敘述式(repetition statement)**，也稱為**循環敘述(iteration statement)**，可讓指定在某種條件為真(True)時，重複執行同一項動作，如下方虛擬碼 (描述購物行程中的重複動作)：

條件

While there are more items on my shopping list
Purchase next item and cross it off my list

  - 其中條件「there are more items on my shopping list (購物欄有項目)」若為真，則執行動作「Purchase next item and cross it off my list (購買此項目並刪除它)」。
  - 當此條件持續為真，此動作就會重複執行。
  - 當條件變成偽時 (即：購買shopping list中的最後一項並將之刪除後)，便停止動作，並進行後面虛擬碼。
- 再舉一例 (價格每次乘3)，如下：

寫法：while (…) {…}




```
product = 3;                          條件
while ( product <= 100 ) {
   product = 3 * product;
}
```

//3, 9, 27, 81, 243 (> 100, 跳出迴圈)

C程式

圖3.4 while重複敘述式的流程圖

3.8 建構演算法案例研究1：計數器控制的重複結構

- 以下說明如何設計一個演算法，我們以數個方式來解決”計算全班平均成績的問題”。
- 請看下列的問題描述：
  - 有一個班級共有10位學生，已進行一次測驗，並取得測驗成績(分數為0到100的整數)。
  - 請求出此班的平均成績。
- 思考：全班平均成績即等於所有成績總和除以學生人數。
- 策略：在電腦上解決此問題的演算法(程式)，須可以
  - 1) 輸入每位學生成績
  - 2) 計算平均值
  - 3) 結果印出

- 此例，我們利用**計數器控制之重複結構(counter-controlled repetition)**，一次一筆輸入這些成績。
  - 此例，我們利用一個稱為**counter(計數器)**變數，指定某一組敘述應被執行之次數。
  - 當counter超過10時，結束重複動作。

程式碼

寫下 Pseudo code:

```
1   Set total to zero
2   Set grade counter to one
3
4   While grade counter is less than or equal to ten
5       Input the next grade
6       Add the grade into the total
7       Add one to the grade counter
8
9   Set the class average to the total divided by ten
10  Print the class average
```

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

```
1  // Fig. 3.6: fig03_06.c
2  // Class average program with counter-controlled repetition.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main( void )
7  {
8     unsigned int counter; // number of grade to be entered next
9     int grade; // grade value
10    int total; // sum of grades entered by user
11    int average; // average of grades
12
13    // initialization phase
14    total = 0; // initialize total
15    counter = 1; // initialize loop counter
16                 //變數若沒初始化，會出錯 (garbage value)
17    // processing phase
18    while ( counter <= 10 ) { // loop 10 times
19       printf( "%s", "Enter grade: " ); // prompt for input
20       scanf( "%d", &grade ); // read grade from user
21       total = total + grade; // add grade to total
22       counter = counter + 1; // increment counter
23    } // end while
24
25    // termination phase
26    average = total / 10; // integer division
27
28    printf( "Class average is %d\n", average ); // display result
29 } // end function main
```

圖3.5 計數器控制之重複結構來解決全班平均問題之虛擬碼演算法

3.9 以從上而下逐步改進的方式建構演算法(案例研究2：警示訊號控制的重複結構)

- 警示值 (sentinel value)，亦稱為信號值、虛值或旗標值，用於告知控制迴圈是否結束。
  - 警示控制的重複結構，通常也稱為不確定次數的重複結構 (indefinite repetition)，因為在迴圈開始執行前並不知道重複的次數。
  - 注意：警示值不可與輸入值混用。

- 接下來，我們將全班平均問題一般化(任意筆數)：
  - 請看下面所述的問題：
  - 設計一個求全班平均之程式，可處理任意筆數之成績
- 思考：前例計算全班平均成績，成績筆數(10筆) 為固定且已知。
- 差異：本例程式必須能夠處理任意筆數成績。

1. 從上而下(top-down)逐步精確修正的技術

- 透過一種**從上而下逐步修正之技術(top-down, stepwise refinement)**，來設計前述之全班平均程式。
  - 首先，從代表總敘述式 (**top**) 的虛擬碼開始思考：

Determine the class average for the quiz　← 從一句話，具體化列出欲進行之動作

  - 觀察：總敘述式是單一敘述，涵蓋程式全部功能
  - 修改：將總敘述式分割成數項、較小的工作，並列出執行順序(步驟)，取得**初步改進 (first refinement)**

Initialize variables (步驟1)
Input, sum, and count the quiz grades (步驟2)
Calculate and print the class average (步驟3)

  - 以上改寫，使用了循序結構(C程式的7種結構之一)。

2. 第二個改進(精確化)

```
1   Initialize total to zero
2   Initialize counter to zero
3
4   Input the first grade
5   While the user has not as yet entered the sentinel
6       Add this grade into the running total
7       Add one to the grade counter
8       Input the next grade (possibly the sentinel)
9
10  If the counter is not equal to zero
11      Set the average to the total divided by the counter
12      Print the average
13  else
14      Print No grades were entered
```

(逐漸將步驟具體化)

(避免除法分母為0, 會出錯 (run time error!))

圖3.7　利用警示訊號控制的迴圈結構來解決全班平均問題的虛擬碼演算法

- 上圖的虛擬碼演算法，解決了前述問題。
  - 此演算法經過兩階層改進後設計完成。
  - 有些情況，可能需要更多階層改進。

- 下圖為對應的C程式及執行範例。

程式碼

(練習trace code)

```
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
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
```

//警示訊號：-1

//考慮各種情況(避免除法的分母為0)

//暫時轉成浮點數(小數)

//小數點後2位

＊型態補充(連結)

```
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 以警示值控制迴圈結構來計算全班平均(任意筆數)

- 其中，第38行

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

  - 為對運算子(total)做強制轉換(cast)浮點數型別(float)，暫時轉換成浮點型別。
  - 注意：原存放total內數值，仍是整數型別，這種方式為**精確轉換 (explicit conversion)**。

//主動轉換　＊型態補充(連結)

- 此時，這項運算轉變成一個浮點數型別(total) 除以無號整數型別(unsigned integer)的counter值。
  - 注意：C語言只能執行相同運算型別的運算式。
  - 因此，編譯器在某些情況，會偷偷地對某些運算元執行**隱含式轉換(implicit conversion)**，以確保所有運算元型別相同。

//以前例來說，若沒加(float), 結果會轉成整數型態

3. 格式化浮點數

- 上例中，程式使用**printf**的轉換指定詞 **%.2f** 來印出**average**的值。

```
printf( "Class average is %.2f\n", average );
```

  - **f**表示以浮點數型別被輸出
  - **.2** 表示其輸出的**精準度(precision)**為小數點後2位
  - 若只用 **%f**（未指定精準度），則**預設精準度(default precision)** 為小數點後6位，如同**%.6f**
- 當輸出浮點數型別時，其值會**被四捨五入(rounded)**成指定精準度。
  - 註：此值在記憶體內未改變(只影響輸出結果)。
  - 舉例：以下兩敘述執行後，將印出3.45和3.4。

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

4. 浮點數注意事項

- 浮點數並非「100%精確」(並非無窮多位數)，但已經可以應用在許多地方。
  - 例如：「正常」的體溫為30.6度(攝氏)時，通常不會刻意計較精準度到太多位數 (實際值可能為30.5999473210643)，視實用性為主。
- 無窮小數：當兩數相除時，可能會產生無窮小數的浮點數
  - 例如：當10除以3，得到無窮小數3.3333333...。
  - 注意：電腦只配置固定大小的記憶體空間來存放此數值，因此只能算是近似值。

3.10 以從上而下逐步改進的方式建構演算法(案例研究 3：巢狀的控制敘述)

//案例：迴圈內加 if..else

- 請看下列問題描述：
  - 目前有一門課程，想要知道學生們在這次考試的表現如何。
  - 目標：希望寫程式來統計考試結果。
  - 假設：已知有10位學生及他們的考試結果 (紀錄1代表通過，2代表未通過)。
- 程式要求：應按照下列規定來統計此次考試結果：

1. 輸入每位考試結果 (即1或2)，並提示「Enter result」以要求輸入下一筆考試結果。
2. 計算每一筆考試結果的個數(包含：通過、不通過)。
3. 列出考試結果，並告知有多少位學生通過，多少位學生未通過。
4. 若超過8位學生通過，印出「Raise tuition」此訊息。

- ＊撰寫前(思考及觀察)：
  1. 有10筆考試結果，可使計數器控制的迴圈。
  2. 每一筆結果：1或2，只需判斷它是否為1 (若不是1，則簡單假設它是2)。
  3. 兩個計數器：計算通過的學生數，計算未通過的學生數。
  4. 最後，判斷是否有8位以上的學生通過。
- 右圖虛擬碼
  - 空行是區隔**while**結構，增進程式可讀性。

Initialize variables (簡易虛擬碼)
Input the ten quiz grades and count passes and failures
Print a summary of the exam results and decide whether i
instructor should receive a bonus

(精確版虛擬碼)

```
1  Initialize passes to zero
2  Initialize failures to zero
3  Initialize student to one
4
5  While student counter is less than or equal to ten
6        Input the next exam result                    (左式1, 2)
7
8        If the student passed                         (左式3)
9              Add one to passes
10       else
11             Add one to failures
12
13       Add one to student counter
14
15 Print the number of passes
16 Print the number of failures
17 If more than eight students passed                  (左式4)
18       Print Bonus to instructor!
```

圖3.9　考試結果問題的虛擬碼

- 接著，將虛擬碼轉換成C程式(如下)：

程式碼

```c
// Fig. 3.10: fig03_10.c
// Analysis of examination results.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   // initialize variables in definitions
   unsigned int passes = 0; // number of passes
   unsigned int failures = 0; // number of failures
   unsigned int student = 1; // student counter
   int result; // one exam result

   // process 10 students using counter-controlled loop
   while ( student <= 10 ) {

      // prompt user for input and obtain value from user
      printf( "%s", "Enter result ( 1=pass,2=fail ): " );
      scanf( "%d", &result );

      // if result 1, increment passes
      if ( result == 1 ) {
         passes = passes + 1;
      } // end if
      else { // otherwise, increment failures
         failures = failures + 1;
      } // end else

      student = student + 1; // increment student counter
   } // end while

   // termination phase; display number of passes and failures
   printf( "Passed %u\n", passes );
   printf( "Failed %u\n", failures );

   // if more than eight students passed, print "Bonus to instructor!"
   if ( passes > 8 ) {
      puts( "Bonus to instructor!" );
   } // end if
} // end function main
```

```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Passed 6
Failed 4
```

```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Passed 9
Failed 1
Bonus to instructor!
```

(>8, 獎勵指導者)

圖3.10　考試結果的分析

3.11 指派運算子(Assignment Operators)

- C語言提供數種指派運算子，並可用縮寫替代
  - 例如：

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

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

```
c += 3;
```

  - 其中，+= 運算子會將右邊運算式的值，加上此運算子左邊變數，並儲存到左邊變數。
  - 規則如下：

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

  - 縮寫成下式：

```
variable operator= expression;
```

  - 下表為常見指派運算子：

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

3.12 遞增和遞減運算子

- 為了方便計算，C語言提供單元遞增運算子”++“ (increment operator)和單元遞減運算子”--“(decrement operator)

運算子	範例運算式	說明
++	++a //當行加1	先將 a 遞增 1，再以 a 的新值進行運算
++	a++ //隔行才加1	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

圖3.12 遞增和遞減運算子

- 下圖程式，示範前置遞增與後置遞增運算子的差異。

程式碼

```
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
```

//隔行才加1

```
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
```

//當行加1

```
5
5
6
5
6
6
```

//隔行才加1

//當行加1

- 下圖為運算子的運算優先順序(precedence)和結合性(associativity)。
  - 優先順序：從上往下由高至低
  - 結合性(由右至左)：條件運算子(?:)、遞增(++)、遞減(--)、正(+)、負(-)及型別轉換運算子及指定運算子(=、+=、-=、*=、/= 和 %=)
  - 其他：由左至右

(高)

優先權

(低)

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

// 負號

圖3.14　常見運算子之優先順序及結合性

3.13 安全程式設計

＊型態補充(連結)

1. 觀念：算數溢位(arithmetic overflow)

- 當計算兩整數總和，如下：
  - `int integer1, integer2, sum;`
  - `sum = integer1 + integer2; //assign total to sum`
  - 以上可能潛在問題：兩整數相加會產生過大值，造成**算數溢位(arithmetic overflow)**，易衍伸不可預知行為，導致系統面臨攻擊危險。
- 程式好習慣：記得確認計算是否可能超出系統最大值(或低於系統最小值)，通常以INT_MAX和INT_MIN來表示，其定義於標頭檔<limits.h>。

2. 觀念：無號整數(unsigned integer)

- 前例中，counter宣告變數為無號整數(unsigned integer)
  - 因為次數不應該出現負值
  - 無號類型的數值範圍從0到有號兩倍範圍，有興趣的同學可以確認<limits.h>中UINT_MAX所使用最大無號整數範圍。