CHAPTER 3 - STRUCTURED PROGRAM DEVELOPMENT

考慮下面的問題

- 問題一：輸入三個整數，然後輸出其總和。

```
#include<stdio.h>
void main()
{
        int x = 0, y;
        scanf(“%d”,&y); x = x + y;
        scanf(“%d”,&y); x = x + y;
        scanf(“%d”,&y); x = x + y;
        printf(“sum: %d\n”,x);
}
```

- 問題二：輸入三十個整數，然後輸出其總和。
  - 你會依照上面的方式來撰寫嗎？如下面例子

```
#include<stdio.h>
void main()
{
        int x = 0, y;
        scanf(“%d”,&y); x = x + y;
        ….
        scanf(“%d”,&y); x = x + y;
        printf(“sum: %d\n”,x);
}
```

  - 有沒有發現這種寫程式的方法怪怪的？

再考慮下面的問題

如果想要印出兩行"Hello world!" ，你會怎麼寫?

```
#include<stdio.h>
int main()
{
    printf("Hello world!\n");
    printf("Hello world!\n");
    return 0;
}
```

再考慮下面的問題

如果想要印出200行"Hello world!" ，你會怎麼寫?

```
#include<stdio.h>
int main()
{
    int u = 0;
    while(u < 200) {
        printf("Hello world\n");
         u = u + 1;
    }
    return 0;
}
```

再考慮下面的問題

如果想要印出任意行"Hello world!" ，你會怎麼寫?

```
#include<stdio.h>
int main()
{
    int u = 0,n;
    scanf("%d",&n);
    while(u < n) {
        printf("Hello world\n");
        u = u + 1;
    }
    return 0;
}
```

再考慮下面的問題

如果想要印出任意行“Hello world!”，而且每10行要印一列----------你會怎麼寫?

```
#include<stdio.h>
int main()
{
    int u = 0,n;
    scanf("%d",&n);
    while(u < n) {
        printf("Hello world\n");
        u = u + 1;
        if (u % 10 == 0) {
            printf("----------\n");
        }
    }
    return 0;
}
```

Control Structures

- **Bohm and Jacopini指出所有的程式皆可由下面這三種控制結構組成**
  - **循序(Sequence structures):**
    - 就是一行一行指令照順序執行啦!
  - **選擇(Selection structures):**
    - 條件成立就執行某些指令不成立則執行另外一些指令
    - C 有四種: **`if`**, **`if/else`**, ? :, and **`switch`**
  - **反覆(Repetition structures):**
    - 當條件成立時重複執行某些指令
    - C 有三種迴圈: **`while`**, **`do/while`** and **`for`**
- 針對某個問題，如何用C語言應用這三種基本控制結構，來描述解決此問題的步驟?

C程式語言控制結構

- **循序結構**（一個指令接著一個指令）
- **選擇結構**(當條件成立執行步驟A否則步驟B)
  - if (條件)
  - if (條件) {} else {}
  - if (條件) {} elseif (條件) {}
  - (條件)? :
  - switch()
- **反覆結構**(重複某些步驟一直到某個條件不成立)
  - for-loop
    - for(初始化變數; 條件; 更動變數) {}
  - while-loop
    - while (條件) {}
  - do-while-loop
    - do {} while (條件)
  - 遞迴函數

C標準函式庫

- C程式可包含許多函式(functions)
  - 程式員可以開發自己的函式
    - 好處:你可以完全知道這個函式的每一處理步驟
    - 壞處: 要花時間寫
  - 程式員將經常使用C 程式庫
    - 應用別人寫好的函式來建構自己的函式
    - 省掉許多程式開發時間
    - C 程式庫裡的函式
      - carefully written, efficient, and portable

如何構思程式

- 一個可行的方式是由上到下逐步精細的方式來設計。
- 用一些較高階方便的方法輔助你構思整個程式：
  - 流程圖
  - 演算法
  - …
- 不管用什麼方法描述解決問題的步驟，對最後你還是要將它轉換成程式。

演算法（Algorithms ）

- Computing problems
  - 其解答步驟可以用一序列的有限的指令描述出來。
- 演算法: procedure in terms of
  - Actions to be executed
  - The order in which these actions are to be executed
- 複雜度(Complexity)
  - 空間複雜度(Space complexity)
  - 時間複雜度(Time complexity)
- 語法錯誤(Syntax errors)
  - Compiler會幫你找到
- 邏輯錯誤(Logic errors)
  - 程式執行時才會發現

Pseudocode

- Pseudocode
  - 一個非正式的程式語言來描述演算法
  - 接近一般的語言(如英文)
  - 不能執行
  - 幫助我們在寫程式前思考如何撰寫
    - 由Pseudocode轉換成程式已不成問題
  - Example

```
void BellmanFord(const int n,const int v)
{ int i,k;
   for(i = 0; i < n; i++) dist[i]=cost[v][i];
   for(k=2; k<=n-1;k++)
      for(each u such that u!=v and u has at least one incoming edge)
               for(each<i,u> in the graph)
                        if (dist[u]>dist[i]+cost[i][u]) dist[u] = dist[i]+cost[i][u];
}
```

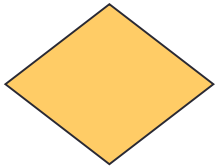
Flowchart

- 流程圖(Flowchart)
  - 用圖示來表示一個演算法
  - 以一些特殊符號表示特定意義

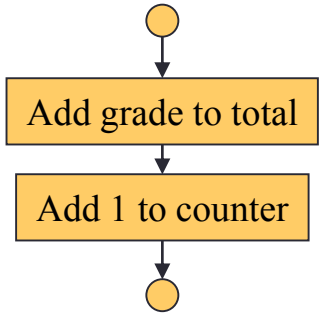
Rectangle symbol (action symbol):
- 表示動作

Oval symbol:
- 表示程式的開始與結束

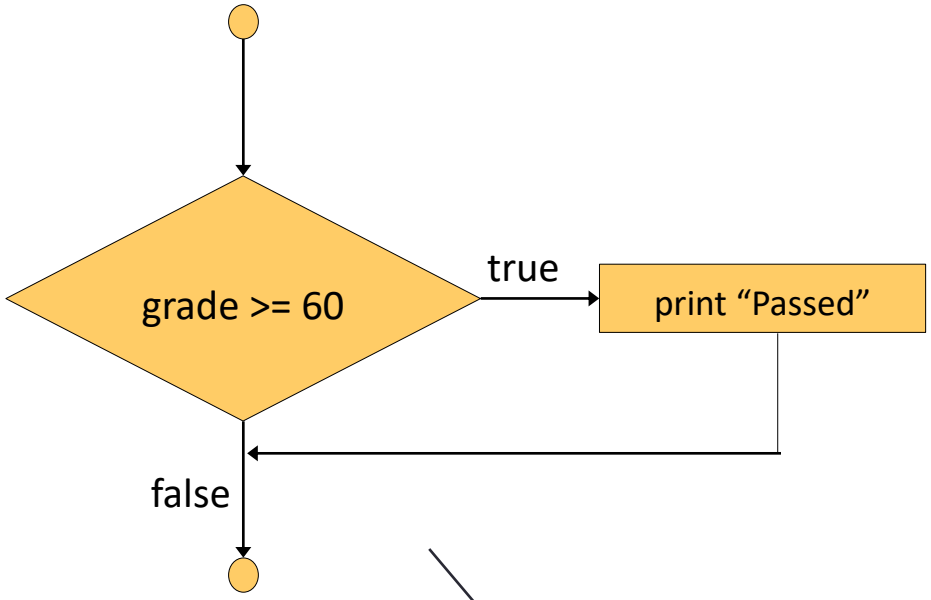
- 表示控制結構的出/入點
  - Connect exit point of one control structure to entry point of the next (control-structure stacking)
  - Makes programs easy to build



- Diamond symbol (decision symbol)
  - 表示決策(選擇)
    - Contains an expression that can be **`true`** or **`false`**
    - Test the condition, follow appropriate path
- 以箭號(稱flowline)連結特殊符號




```
total = total+grade;
counter=counter+1;
```



A decision can be made on any expression.

zero - **`false`**

nonzero - **`true`**

Example:

`3 - 4` is **`true`**

in C

```
if (grade>=60) printf(“Passed”);
```

選擇結構(Selection structures):

- C語言裡有四種表達選擇結構的語法:
  - **`if`**,
  - **`if/else`**,
  - ? :,
  - **`switch`**

The `if` Selection Structure

- Pseudocode:

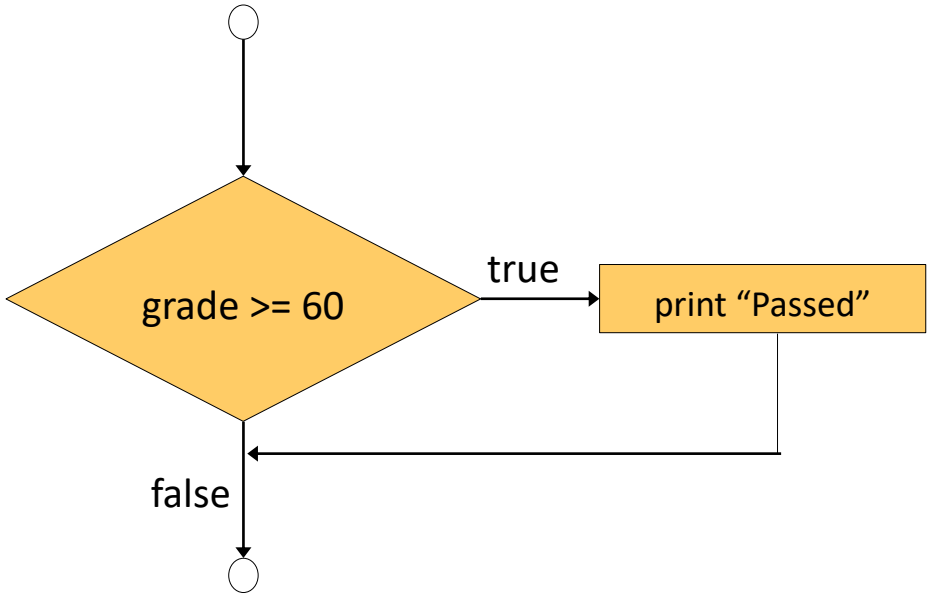
  If student's grade is greater than or equal to 60
  Print "Passed"

- Pseudocode statement in C:

```
if ( grade >= 60 )
   printf( "Passed\n" );
```

  - 若是If 的條件成立
    - printf的指令會被執行然後繼續進行下一個 statement
  - 若是If 的條件不成立
    - printf的指令會被忽略然後繼續進行下一個 statement

The `if` Selection Structure



A decision can be made on any expression.

zero - **`false`**

nonzero - **`true`**

Example:

`3 - 4` is **`true`**

```
if (grade>=60) printf(“Passed”);
```

The `if/else` Selection Structure

- **`if`**
  - 只有當if條件式成立(true)時做某些動作，條件式不成立(false)則沒動作
- **`if/else`**
  - 只有當if條件式成立(true)時做某些動作，條件式不成立(false)則做另一動作
- Psuedocode:

  If student's grade is greater than or equal to 60
  Print "Passed"
  else
  Print "Failed"

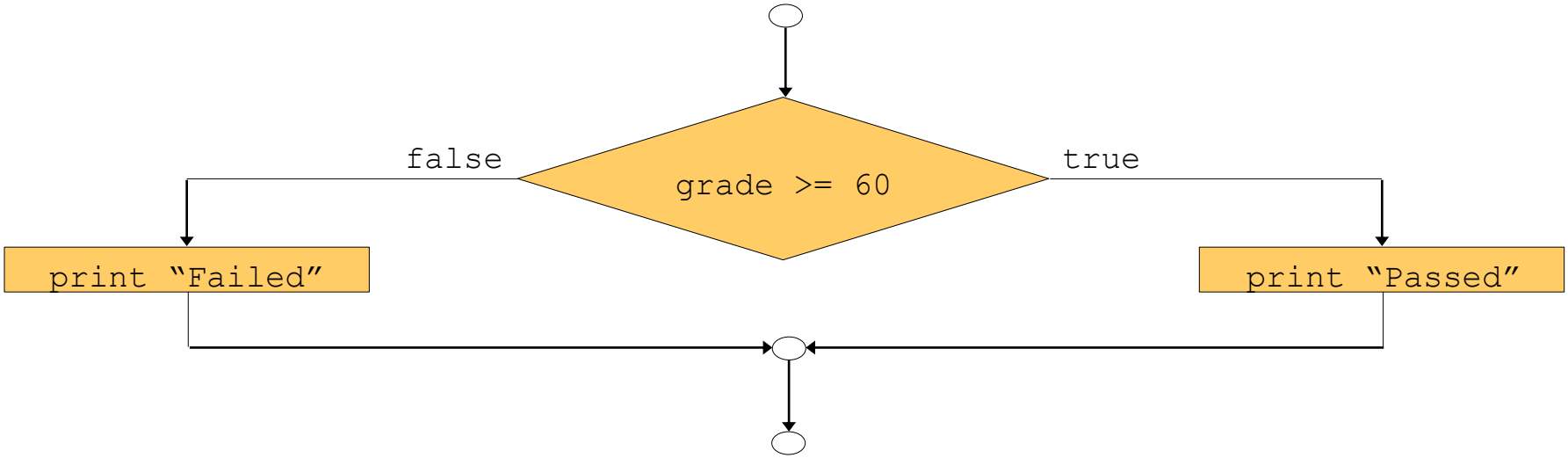
  - C code:

```
if ( grade >= 60 )
   printf( "Passed\n");
else
   printf( "Failed\n");
```

建議寫成這樣子

```
if ( grade >= 60 ) {
   printf( "Passed\n");
} else {
   printf( "Failed\n");
}
```

The `if/else` Selection Structure




```
if ( grade >= 60 )
   printf( "Passed\n");
else
   printf( "Failed\n");
```

The `if/else` Selection Structure

- 巢狀if/else結構
  - 只要某一條件符合，其他敘述就會被省略
- Pseudocode for a nested **`if/else`** structure

```
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
            Print "B"
        else
            If student's grade is greater than or equal to 70
                Print "C"
            else
                If student's grade is greater than or equal to 60
                    Print "D"
                else
                    Print "F"
```

```
if （grade>=90)
   printf("A");
else if (grade >= 80)
  printf("B");
else if(grade >= 70)
  printf("C");
else if(grade >= 60)
  printf("D");
else
  printf("E");
```

The `if/else` Selection Structure

- 複合敘述(Compound statements):
  - 在{ 與 }內的敘述。
- 區塊(Block):

```
//複合敘述與宣告
{
  int x,y;  /* 這裡可以宣告變數喔 */
  x = x+1;
}
```

- Example:

```
if ( grade >= 60 )
   printf( "Passed.\n" );
else {
   printf( "Failed.\n" );
   printf( "You must take this course again.\n" );
  }
```

超過一行Statement用{}標示此區塊

```
if ( grade >= 60 )
   printf( "Passed.\n" );
else
   printf( "Failed.\n" );
   printf( "You must take this course  again.\n" );
```

有什麼差別?

The ?: Selection Structure

- **Ternary conditional operator (?:)**
  - 語法
    - condition ? value if **true** : value if **false**
  - 範例

```
printf( "%s\n", grade >= 60 ? "Passed" : "Failed" );
```

或

```
grade >= 60 ? printf( "Passed\n" ) : printf( "Failed\n" );
```

效果一樣，但是看得出差異嗎？

反覆結構(Repetition structures)

- 反覆結構(Repetition structure)
  - 當某個條件為真(true)時重複做某些指令
  - 要訣：
    - **必須知道迴圈終止的條件**
      - 作幾次
      - 持續迴圈的條件
    - **迴圈每做完一回合，哪些變數必須改變，及如何改變**

The `while` Repetition Structure

- **`while`** loop 當條件式成立時會一直重複執行

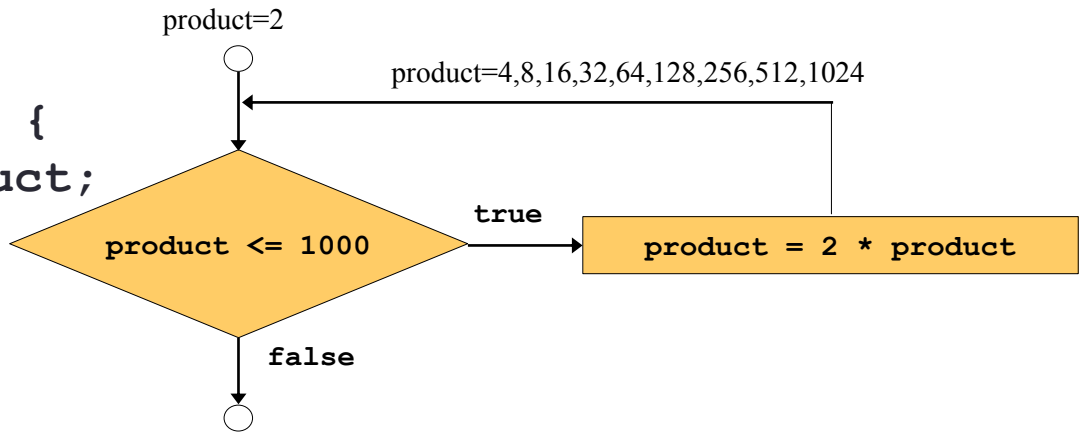
語法:

```
while (條件式) {

}
```

Example:

```
int product = 2;
while ( product <= 1000 ) {
       product = 2 * product;
}
```



重新思考下面的問題

- 問題二：輸入三十個整數，然後輸出其總和。

```
#include<stdio.h>
void main()
{
        int x = 0, y;
        scanf(“%d”,&y); x = x + y;
        ….
        scanf(“%d”,&y); x = x + y;
        printf(“sum: %d\n”,x);
}
```

  - 應用while-loop可以寫成

```
#include<stdio.h>
void main()
{
        int x = 0, c = 0, y;
        while (c < 30) {
            scanf(“%d”,&y);
            x = x + y;
            c = c + 1;
        }
        printf(“sum: %d\n”,x);
}
```

Formulating Algorithms (Counter-Controlled Repetition)

- 計數器控制的反覆結構(Counter-controlled repetition)
  - 迴圈一直重複到計數器等於某個值
  - 若反覆次數是明確的，那麼我們可以使用下面例子的方式

Example:

A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz.

Pseudocode:

- 需要兩個變數來計數反覆次數與總和
  - total: 總和，
  - counter: 紀錄已輸入人數
- 設定變數初始值

```
total=0; counter = 0;
```

- 當 counter < 10
  輸入成績;
  加入總和
  已輸入人數加I
- 計算平均

```c
/* Fig. 3.6: fig03_06.c
   Class average program with
   counter-controlled repetition */
#include <stdio.h>

int main()
{
   int counter, grade, total, average;

   /* initialization phase */
   total = 0;
   counter = 0;

   /* processing phase */
   while ( counter < 10 ) {
      printf( "Enter grade: " );
      scanf( "%d", &grade );
      total = total + grade;
      counter = counter + 1;
   }

   /* termination phase */
   average = total / 10;
   printf( "Class average is %d\n", average );

   return 0;   /* indicate program ended successfully */
}
```

1. Initialize Variables
2. Execute Loop
3. Output results

設定變數初始值

total=0; counter = 0;

當 counter < 10

輸入成績;

加入總和

已輸入人數加I

計算平均

```
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
```

Formulating Algorithms with Top-Down, Stepwise Refinement (Sentinel-Controlled Repetition)

- Problem becomes:

  Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.

  - 不知道有多少學生
  - 程式如何知道資料結束?
- 可使用一個標兵值(sentinel value)
  - 當輸入資料等於sentinel value時就是資料結束(end of data)
  - Sentinel value 必須與一般資料能有所區隔 (such as `-1` in this case)
    - 又稱為 signal value, dummy value, or flag value

Formulating Algorithms with Top-Down, Stepwise Refinement

- 許多程式可分為三階段:
  - 初始(Initialization):初始化變數。
  - 處理(Processing): 輸入與處理資料。
  - 結束(Termination): 計算與輸出最後結果。

Determine the class average for the quiz

Initialize variables
Input, sum and count the quiz grades
Calculate and print the class average

由上到下逐步設計清楚

(Top-down, stepwise refinement)

```
設定變數初始值
        total=0; counter = 0;

輸入第一筆成績成績;
當 輸入成績 != -1
                加入總和
                已輸入人數加I
                輸入下一筆成績

計算平均: total/counter
```

```c
/* Fig. 3.8: fig03_08.c
   Class average program with
   sentinel-controlled repetition */
#include <stdio.h>

int main()
{
   float average;              /* new data type */
   int counter, grade, total;

   /* initialization phase */
   total = 0;
   counter = 0;

   /* processing phase */
   printf( "Enter grade, -1 to end: " );
   scanf( "%d", &grade );

   while ( grade != -1 ) {
      total = total + grade;
      counter = counter + 1;
      printf( "Enter grade, -1 to end: " );
      scanf( "%d", &grade );
   }
```

```
25
26    /* termination phase */
27    if ( counter != 0 ) {
28       average = ( float ) total / counter;
29       printf( "Class average is %.2f", average );
30    }
31    else
32       printf( "No grades were entered\n" );
33
34    return 0;   /* indicate program ended successfully */
35 }
```

```
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
```

Program: 判斷奇數與偶數(1/2)

```
#include<stdio.h>
int main()
{
    int n; /* the number of test cases */
    int u; /* a counter */
    int x; /* input integer */
    scanf("%d",&n); /* input n, the number of test cases */
    u = 0;
    while(u < n) {
        scanf("%d", &x); /* read an integer x */
        if (x % 2!=0) { /* x % 2 yields the remainder of the division of x by 2 */
            printf("%d is odd.\n",x);
        } else {
            printf("%d is even.\n",x);
        }
        u = u + 1; /* increase u by 1 */
    }
    return 0;
}
```

Program: 判斷奇數與偶數(2/2)

```
#include<stdio.h>
int main()
{
    int x; /* input integer */
    while(scanf("%d", &x)!=EOF) {
        if (x % 2!=0) { /* x % 2 yields the remainder of the division of x by 2 */
            printf("%d is odd.\n",x);
        } else {
            printf("%d is even.\n",x);
        }
    }
    return 0;
}
```

輸入時按
Windows: <Ctrl>-z
Linux: <Ctrl>-d
產生EOF

Nested control structures

- Problem
  - A college has a list of test results (1 = pass, 2 = fail) for 10 students. Write a program that analyzes the results
    - If more than 8 students pass, print "Raise Tuition"
- 分析問題
  - 需要輸入10筆資料
    - Counter-controlled loop will be used
  - 需要變數紀錄通過考試的人數
  - 輸入資料1表示通過，2表示不通過
    - If the number is not a 1, we assume that it is a 2

Nested control structures

- 初步設計
  1. 初始化變數;
  2. 輸入10筆成績計數及格與不及格數;
  3. 輸出結論;

- 詳述第一步驟: 初始化變數

  Initialize passes to zero

  Initialize failures to zero

  Initialize student counter to one

Nested control structures

- 詳述第二步驟:輸入10筆成績與計數及格與不及格人數;

  While student counter is less than or equal to ten
  Input the next exam result
  If the student passed
  Add one to passes
  else
  Add one to failures
  Add one to student counter

- 詳述第三步驟:輸出結論;

  輸出及格人數
  輸出不及格人數
  如果超過8人及格輸出 “Raise tuition”

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

int main()
{
   /* initializing variables in declarations */
   int passes = 0, failures = 0, student = 1, result;

   /* process 10 students; counter-controlled loop */
   while ( student <= 10 ) {
      printf( "Enter result ( 1=pass,2=fail ): " );
      scanf( "%d", &result );

      if ( result == 1 )        /* if/else nested in while */
         passes = passes + 1;
      else
         failures = failures + 1;

      student = student + 1;
   }

   printf( "Passed %d\n", passes );
   printf( "Failed %d\n", failures );

   if ( passes > 8 )
      printf( "Raise tuition\n" );

   return 0;   /* successful termination */
}
```

1. Initialize variables

2. Input data and count passes/failures

3. Print results

```
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
```

Program Output

Assignment Operators

- `c = c + 3;` 可以被縮寫成`c += 3;`
- 下面形式的運算式

  variable = *variable* *operator* *expression*;

  可被改寫成

  variable ***operator=*** *expression*;
- 例子

```
d += 4      (d = d + 4)
d -= 4      (d = d - 4)
e *= 5      (e = e * 5)
f /= 3      (f = f / 3)
g %= 9      (g = g % 9)
```

- 不過x=x+v; 與x+=v; 並不一定相等。
  - x=x+v; 總共算x兩次。x+=v;算x一次。
  - 例如x[i++]+=v; x[i++] = x[i++]+v;就不會一樣。

Increment and Decrement Operators

- **Increment operator (++)**
  - 可以取代 `c+=1`
- **Decrement operator (--)**
  - 可以取代`c-=1`
- **Preincrement**
  - `++c` or `--c`
  - 變數c會先改變內容， c的值才被應用。
- **Postincrement**
  - `c++` or `c--`
  - c的值先被應用，然後變數c才改變內容

Increment and Decrement Operators

- if (c == 5) printf("%d", ++c);
  - Prints 6
- if (c == 5) printf("%d", c++);
  - Prints 5
  - 最後c的值都是 6
- 當變數不在運算式內Preincrementing與postincrementing效果相同

```
i = 0;
printf("%d\n", i++); // 印出0。
printf("%d\n", i);   // 印出1。
i = 0;
printf("%d\n", ++i);// 印出1。
printf("%d\n", i);   // 印出1。
```

千萬不要寫出讓人不好讀的敘述，例如

```
i = 1;
j = 2;
k = ++i + j++; // i+=1; k=i+j; j+=1;
```

Expression evaluation

優先權	運算子	結合律
1	++(postfix) --(postfix)	from left to right
2	++(prefix) --(prefix) + (unary plus) - (unary minus)	from right to left
3	* / %	from left to right
4	+ -	from left to right
5	= *= /= %= += -=	from right to left

```
a=b+=c++ - d + --e / -f;
```

根據左表，對上式刮上當對映的刮號

```
(a= (b+= (( c++ - d) + ((--e) / (-f)))));
```

c-d c+=1

用適當的刮號可以讓運算式子好讀一點，但是太多的刮號也會造成閱讀困擾。

Order of subexpression evaluation

- 每個子運算式的運算順序也會影響整個運算式的結果。
  - 例一：(a+b)*(c-d)，到底是a+b先算，然後再算c-d。還是反過來？結果一定一樣嗎？

    實例: a=3; b=1; (a++ + b++) * (a++ - b++)

    若先算a++ + b++ //結果為4, a為4,b為2

    再算a++ - b++ //結果為2, a為5,b為3

    (a++ + b++) * (a++ - b++) //結果為8

    若先算a++ - b++ //結果為2, a為4,b為2

    再算a++ + b++ //結果為6, a為5,b為3

    (a++ + b++) * (a++ - b++) //結果為12
  - 例三：a=5; c=(b=a+2) – (a=1); // c??
- C並沒有明確定義子運算式計算順序。因此要避免會造成運算式結果不明確的寫法。
  - 換另一個編譯器可能算出來的結果就不一樣。

Expression statement

- C語言裡運算式可以當做一個敘述。

  例：

  ```
  int i;
  i++;


  i+2;
  ```

  這個敘述將i+2計算出來後，便將結果捨棄，沒有改變任何變數。因此除了花了計算時間外，沒有任何效果。

避免溢位的寫法

- 計算時要避免溢位(overflow)
- 例子:
  - int sum, a, b; ….
  - sum = a + b;
  - 使用INT_MAX; INT_MIN //include<limits.h>
  - 參考www.securecoding.cert.org
  - 避免溢位的寫法

```
signed int sum, a, b;
if (((b>0) && (a > (INT_MAX-b)))
 || ((b<0) && (a < (INT_MIN-b)))) {
   /* Handle error condition */
} else {
  sum = a + b;
}
```

一個有趣的問題:判斷9的倍數

- 輸入一個正整數，請判斷它是否為9的倍數。

```
#include<stdio.h>
int main()
{
    unsigned x;
    scanf("%u",&x);
    if (x%9==0) {
        printf("a mutiple of 9\n");
    } else {
        printf("not a mutiple of 9\n");
    }
    return 0;
}
```

萬一x很大呢??

```
#include<stdio.h>
int main()
{
    long long unsigned x;
    scanf("%llu",&x);
    if (x%9==0) {
        printf("a mutiple of 9\n");
    } else {
        printf("not a mutiple of 9\n");
    }
    return 0;
}
```

萬一x非常非常大呢??

- 一個不管輸入數字有多大都可以應付的算法。

```
#include<stdio.h>
int main()
{
    int x,s=0;
    while(scanf("%1d",&x)!=EOF) {
        s = (s + x) % 9;
     }
    if (s==0) {
        printf("a mutiple of 9\n");
    } else {
        printf("not a mutiple of 9\n");
    }
    return 0;
}
```

原理:

$x\%9 = (a_n 10^n + \cdots + a_1 10 + a_0)\%9$

$= (a_n(999\ldots9 + 1) + \cdots + a_1(9 + 1) + a_0)\%9$

$= (a_n + \cdots + a_1 + a_0)\%9$

$= ((\ldots(((a_n)\%9 + a_{n-1})\%9 + a_{n-2})\%9 + \cdots + a_1)\%9 + a_0)\%9$

類似的問題可以用類似的方法分析出來類似的計算方式。試試11的倍數。

程式流程控制機制

1. **循序(Sequencing)**: 敘述照一定的順序(通常是在程式裡的順序)依序執行。
2. **選擇(Selection)**: 依指定變數或運算式的結果，決定後續執行的程式。如if; if else, switch等。
3. **疊代(迴圈)(Iteration)**: 迴圈是指一段程式碼重複執行指定次數或重複執行至繼續執行條件不成立為止。如for,while等。
4. **程序(Procedural abstraction)**: 程序是完成一項特定工作的程式碼。其他程式片段可以將流程移轉到程序中，執行特定工作後再回到原來的程式。若程式中有許多部份都需要執行一特定工作，利用程序可以減少程式碼的長度。
5. **遞迴(Recursion)**: 程序直接或間接呼叫到自己。需要使用堆疊來記錄遞迴計算過程。
6. **並行(Concurrency)**:兩個或多個程式片段同時計算。它可以在不同的處理器下執行，或在同一顆處理得到相同效果條件下交錯執行。
7. **異常處理(Exception handling and speculation)**:程式例外發生時會跳到例外處理者處理。
8. **不確定性(Nondeterminacy)**: 敘述或運算的順序故意讓它不確定，使得可以得到多種可能的正確計算結果。