

程式設計

Ch04. Iteration Control

Chuan-Chi Lai 賴傳淇

Department of Communications Engineering
National Chung Cheng University

Spring Semester, 2024

Outline

- 1 while 迴圈敘述式 (while Loop Statement)
- 2 遞增與遞減運算子 (Increment and Decrement Operators)
- 3 迭代的基本概念 (Essentials of Iteration)
- 4 for 迴圈敘述式 (for Loop Statement)
- 5 do while 迴圈敘述式 (do while Loop Statement)
- 6 break 與 continue 敘述式 (break and continue Statements)
- 7 應用：歐幾里得演算法 (Euclidean Algorithm)
- 8 函式 scanf 的回傳值 (Return Values of scanf)

while 迴圈敘述式 while Loop Statement

迴圈與迭代

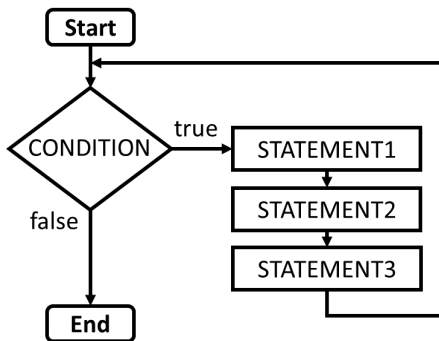
- 當我們想要讓電腦重複運行同一個動作時，可使用迴圈敘述式。迴圈敘述式會帶有一個判斷式 (condition)，以及一個區塊 (block)。
- 計算機會將迴圈敘述式中的區塊重複運行，直到判斷式為 false 為止。
- 在迴圈 (loop) 中，每一次對運行動作的重複被稱為一次「迭代 (iteration)」。

while 迴圈敘述式 (while Loop Statement)

while 敘述式

- while 敘述式的語法如下：

```
1 While (CONDITION)
2 {
3     STATEMENT1;
4     STATEMENT2;
5     STATEMENT3;
6 }
```



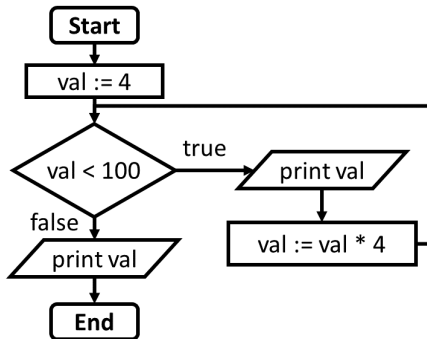
while 迴圈敘述式 (while Loop Statement)

- while 迴圈程式範例：

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int val = 4;
7     while (val < 100)
8     {
9         printf("val = %d\n", val);
10        val *=4;
11    }
12    printf("Exit loop\nval = %d\n", val);
13 }
```

C:\Projects\test.exe

```
val = 4
val = 16
val = 64
Exit loop
val = 256
```



遞增與遞減運算子

Increment and Decrement Operators

遞增與遞減運算子 (Increment and Decrement Operators)

- 在寫程式時，特別是在迴圈內，我們很常會用到遞增或遞減，可以使用「+=1」或「-=1」來實現，也可以使用運算子「++」與「--」來更便利的代表加 1 與減 1。
- 遞增運算子 (++) 與遞減運算子 (--) 個別分為前置與後置。

運算式	說明
++a、--a	先將 a 加/減 1，再以 a 的新值進行運算
a++、a--	以 a 目前的值進行運算，再將 a 加/減 1

遞增與遞減運算子 (Increment and Decrement Operators)

- 比較前置與後置：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int val = 100;
7      int bar = ++val;
8      printf("%d\n", ++bar);
9      printf("%d\n", bar);
10 }
```

C:\Projects\test.exe

102
102

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      int val = 100;
7      int bar = val++;
8      printf("%d\n", bar++);
9      printf("%d\n", bar);
10 }
```

C:\Projects\test.exe

100
101

遞增與遞減運算子 (Increment and Decrement Operators)

- 運算優先度由高而低排序，以分隔線表示不同優先度：

運算子	關聯性	形式
++ (後置) -- (後置)	由左至右	後置
sizeof() ++ (前置) -- (前置) ! (邏輯 NOT) - (負號) + (正號) & (取址) (type) (轉型)	由右至左	單元性
* (乘法) / (除法) % (模數)	由左至右	乘法
+ (加法) - (減法)	由左至右	加法
< <= > >=	由左至右	關係
== !=	由左至右	相等
&&	由左至右	邏輯 AND
	由左至右	邏輯 OR
?:	由右至左	條件
= += -= *= /= %=	由右至左	指派

while 迴圈敘述式 (while Loop Statement)

練習思考

- 下方程式碼，第 8 行與第 10 行印出變數 a 與 b 的值分別為何？

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int a = 5, b = 10;
7     b += a++;
8     printf("a = %d, b = %d\n", a, b);
9     a += ++b;
10    printf("a = %d, b = %d\n", a, b);
11 }
```

迭代的基本概念 Essentials of Iteration

迭代的基本概念 (Essentials of Iteration)

- 迴圈敘述中，當迴圈繼續判斷式為 `true` 時，電腦會將迴圈敘述的區塊進行迭代 (重複)。
- 通常，我們會使用以下兩種方式設計迴圈：
 - 計數器控制
 - 警示值控制

計數器控制

- 計數器控制的迴圈需要：

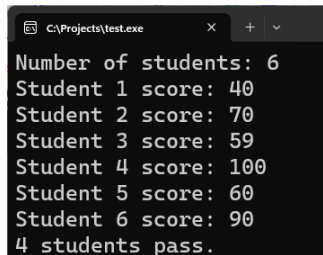
- ① 計數器變數
- ② 計數器的初始值
- ③ 每次迭代的遞增/遞減量
- ④ 計數器的終止值

```
1 int counter = INIT_VAL;  
2 While (counter <= FINAL_VAL)  
3 {  
4     /* DO SOMETHING */  
5     counter += INCREMENT;  
6 }
```

迭代的基本概念 (Essentials of Iteration)

- 計數器控制的範例：

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int counter = 0, nums, pass = 0;
7     printf("Number of students: ");
8     scanf("%d", &nums);
9     while (counter < nums)
10    {
11        int score;
12        printf("Student %d score: ", counter + 1);
13        scanf("%d", &score);
14        if (score >= 60) pass++;
15        counter++;
16    }
17    printf("%d students pass.\n", pass);
18 }
```



```
C:\Projects\test.exe
Number of students: 6
Student 1 score: 40
Student 2 score: 70
Student 3 score: 59
Student 4 score: 100
Student 5 score: 60
Student 6 score: 90
4 students pass.
```

迭代的基本概念 (Essentials of Iteration)

警示值控制

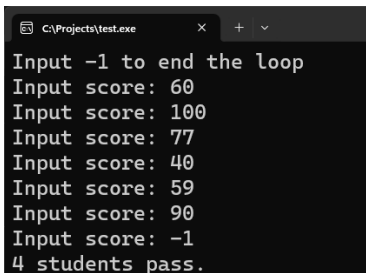
- 無法事先知道迭代的次數的情況時，必須使用警示值控制。
- 當輸入特定的值或是當特定變數變為警示值 (sentinel value 或稱旗標值 flag value) 時結束迴圈。

```
1 int flag = INIT_VAL;  
2 While (flag != FLAG_VAL)  
3 {  
4     /* DO SOMETHING */  
5     if (/* SOME REASON */)   
6         flag = FLAG_VAL;  
7 }
```


迭代的基本概念 (Essentials of Iteration)

- 警示值控制的範例：

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void main()
5 {
6     int isEnd = 0, pass = 0;
7     printf("Input -1 to end the loop\n");
8     while (!isEnd)
9     {
10         int score;
11         printf("Input score: ");
12         scanf("%d", &score);
13         if (score < 0) isEnd = 1;
14         else if (score >= 60) pass++;
15     }
16     printf("%d students pass.\n", pass);
17 }
```



```
C:\Projects\test.exe
Input -1 to end the loop
Input score: 60
Input score: 100
Input score: 77
Input score: 40
Input score: 59
Input score: 90
Input score: -1
4 students pass.
```

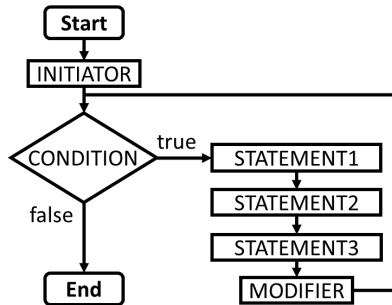
for 迴圈敘述式 for Loop Statement

for 迴圈敘述式 (for Loop Statement)

for 敘述式

- for 迴圈的語法中包含了計數器控制所需的細節，因此我們通常選擇使用 for 迴圈來控制計數器的迭代。
- for 敘述式的語法如下：

```
1 for (INITIATOR; CONDITION; MODIFIER)
2 {
3     STATEMENT1;
4     STATEMENT2;
5     STATEMENT3;
6 }
```



for 迴圈敘述式 (for Loop Statement)

- for 迴圈括號中的 3 個敘述句稱為標頭運算式。
- 使用 for 迴圈可以非常明確的表示計數器控制的初始值、中止值與遞增或遞減量。

```
1 # include <stdio.h>
2 # include <stdlib.h>
3
4 int main()
5 {
6     int i = 0;
7     for (i = 0; i < 10; i++)
8         printf("%d ", i);
9     printf("\n");
10 }
```

C:\Projects\for count inc.exe

0 1 2 3 4 5 6 7 8 9

```
1 ~# include <stdio.h>
2 # include <stdlib.h>
3
4 ~int main()
5 {
6     int i;
7     for (i = 19; i >= 0; i-=2)
8         printf("%d ", i);
9     printf("\n");
10 }
```

C:\Projects\for count dec.exe

19 17 15 13 11 9 7 5 3 1

for 迴圈敘述式 (for Loop Statement)

- 如果要讓迴圈執行 N 次，for 迴圈的標頭可寫成以下兩種：

```
1 for (i = 0; i < N; i++) printf("A");  
2  
3 for (i = 1; i <= N; i++) printf("A");
```

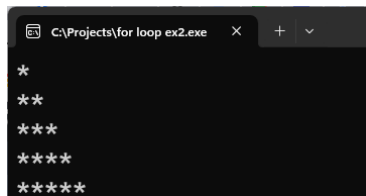
- 如果判斷式寫錯的話，可能會造成誤差為 1 的錯誤 (off-by-one error)：

```
1 for (i = 0; i <= N; i++) printf("A"); // N + 1 times  
2  
3 for (i = 1; i < N; i++) printf("A"); // N - 1 times
```

for 迴圈敘述式 (for Loop Statement)

- 於版本 C99 以後，for 敘述式的初始處可以宣告迴圈內的區域變數。

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      for (int i = 1; i <= 5; i++)
7      {
8          for (int j = 0; j < i; j++)
9          {
10             printf("*");
11          }
12          // now variable j is unable
13          printf("\n");
14      }
15      // now variable i is unable
16  }
```



```
C:\Projects\for loop ex2.exe
*
**
***
****
*****
```

for 迴圈敘述式 (for Loop Statement)

- 多層的迴圈 (例如上一頁的程式碼) 稱為巢狀迴圈，巢狀 for 迴圈的計數器習慣命名：
 - 第一層：i
 - 第一層：j
 - 第一層：k
 - 第一層：m (盡量不要用到第四層以上)
- 若能使用有意義的英文單字來取名會更好。

for 迴圈敘述式 (for Loop Statement)

- for 迴圈的標頭運算式是可有可無的，
 - 若計數器變數已經在之前就已經初始化，則可以將初始敘述式省略。
 - 若不需要遞增或遞減，或已經在迴圈本體中完成，則可以省略循環敘述式。
 - 若省略條件式，編譯器會認為控制條件永遠為 true，而形成無窮迴圈。
 - 雖然標頭運算式都可以省略，但是分隔運算式用的分號 (;) 是不能省略的。

```
1 for (;;)
2 {
3     /* infinite loop */
4 }
```

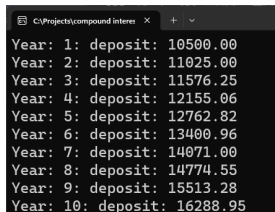
```
1 While (1) //while的條件式不可省略
2 {
3     /* infinite loop */
4 }
```


for 迴圈敘述式 (for Loop Statement)

應用：列出每年複利

- 若某人將 10000 元存入年利率 5% 的帳戶裡，請列出 10 年內每年結算時帳戶的錢。

```
1  #include <stdio.h>
2
3  int main()
4  {
5      double deposit = 10000, interestRate = 0.05;
6
7      for (int year = 1; year <= 10; year++)
8      {
9          deposit *= 1 + interestRate;
10         printf("Year: %d: deposit: %.2f\n", year, deposit);
11     }
12 }
```



```
C:\Projects\compound interest x + v
Year: 1: deposit: 10500.00
Year: 2: deposit: 11025.00
Year: 3: deposit: 11576.25
Year: 4: deposit: 12155.06
Year: 5: deposit: 12762.82
Year: 6: deposit: 13400.96
Year: 7: deposit: 14071.00
Year: 8: deposit: 14774.55
Year: 9: deposit: 15513.28
Year: 10: deposit: 16288.95
```

do while 迴圈敘述式 do while Loop Statement

警示值控制的迭代 (1/2)

- 警示值 (或稱旗標值 flag value) 是常用來做判斷是否離開迴圈的手段。
- 常見的警示設計值方式：
 - ① 在一般的變數 (例如儲存成績的變數或儲存搜尋結果的變數等) 設定一個功能上不可能出現的值作為旗標值，例如 -1 或 999 等。
 - ② 宣告一個旗標變數作為目前狀態的標誌，旗標變數通常是 bool 或 int 型態。

警示值控制的迭代 (2/2)

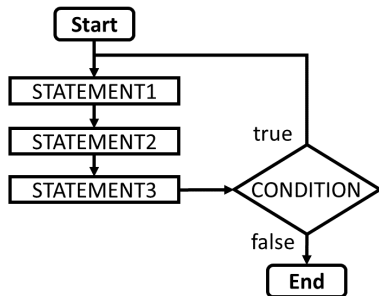
- 欲判斷警示值的變數在進入 while 迴圈前必須先指派數值，否則該變數的數值會是保留在該記憶體位置中的「垃圾值 (garbage value)」，導致結果能不如預期。
- 若保證會迭代至少一次且不想在迴圈之前指派數值，則需選擇使用 do while 迴圈。

do while 迴圈敘述式 (do while Loop Statement)

do while 敘述式

- do while 迴圈的判斷式是在每次迭代之後。可以將其看作是必定會先執行一次迭代的 while 迴圈。
- do while 敘述式的語法如下：

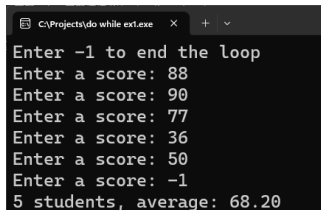
```
1 do
2 {
3     STATEMENT1;
4     STATEMENT2;
5     STATEMENT3;
6 } while (CONDITION);
```



do while 迴圈敘述式 (do while Loop Statement)

- do while 範例 1：連續輸入成績，警示值-1 結束輸入，計算並最後輸出平均成績。

```
1 #include <stdio.h>
2
3 int main() {
4
5     int score, sum = 0, count = 0;
6     printf("Enter -1 to end the loop\n");
7     do {
8         printf("Enter a score: ");
9         scanf("%d", &score);
10        if (score >= 0) {
11            sum += score;
12            count++;
13        }
14    } while (score != -1);
15
16    if (count > 0) {
17        printf("%d students, average: %.2f\n",
18              count, (double)sum / count);
19    } else {
20        printf("No scores entered.\n");
21    }
22 }
```

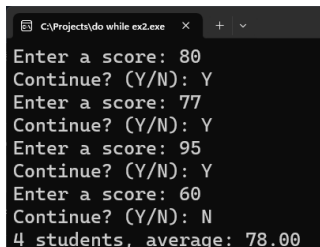


```
C:\Projects\do while ex1.exe X + v
Enter -1 to end the loop
Enter a score: 88
Enter a score: 90
Enter a score: 77
Enter a score: 36
Enter a score: 50
Enter a score: -1
5 students, average: 68.20
```

do while 迴圈敘述式 (do while Loop Statement)

- do while 範例 2：連續輸入成績，輸入 Y 或 N 選擇是否繼續，計算並最後輸出平均成績。

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5
6     int score, sum = 0, count = 0;
7     char conti;
8     do {
9         printf("Enter a score: ");
10        scanf("%d", &score);
11        if (score >= 0) {
12            sum += score;
13            count++;
14        }
15        printf("Continue? (Y/N): ");
16        scanf("%c%c", &conti); // %c for skipping '\n'
17    } while (conti != 'N' && conti != 'n');
18
19    if (count > 0) {
20        printf("%d students, average: %.2f\n",
21              count, (double)sum / count);
22    } else {
23        printf("No scores entered.\n");
24    }
25 }
```



```
C:\Projects\do while ex2.exe
Enter a score: 80
Continue? (Y/N): Y
Enter a score: 77
Continue? (Y/N): Y
Enter a score: 95
Continue? (Y/N): Y
Enter a score: 60
Continue? (Y/N): N
4 students, average: 78.00
```

break 與 continue 敘述式

break and continue Statements

break 與 continue 敘述式

- break 與 continue 敘述式可以改變程式的控制流程：
 - break：離開當前的迴圈敘述式或 switch 敘述式。
 - continue：跳過本次迭代中尚未執行的部分。

break 與 continue 敘述式

- break：離開當前的 while、for、do while 或 switch 敘述式。

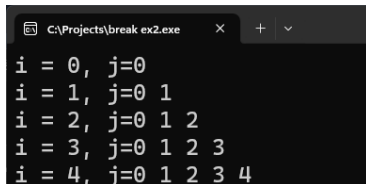
```
1  # include <stdio.h>
2  # include <stdlib.h>
3
4  int main()
5  {
6      int i;
7      for (i = 0; i < 10; i++)
8      {
9          if (i == 5)
10             break;
11             printf("%d ", i);
12     }
13     printf("\nleaved loop at i = %d\n", i);
14 }
```

```
C:\Projects\break ex1.exe  x  +  v
0 1 2 3 4
leaved loop at i = 5
```

break 與 continue 敘述式

- 若 break 敘述式在兩層以上的迴圈，則只會離開當前的那一層。

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int i, j;
7      for (i = 0; i < 5; i++)
8      {
9          printf("i = %d, j=", i);
10         for (j = 0; j < 5; j++)
11         {
12             printf("%d ", j);
13             if (i == j) break;
14         }
15         printf("\n");
16     }
17 }
```



```
i = 0, j=0
i = 1, j=0 1
i = 2, j=0 1 2
i = 3, j=0 1 2 3
i = 4, j=0 1 2 3 4
```

break 與 continue 敘述式

- continue：於 while、for、do while 敘述式內，跳過本次迭代中尚未執行的部分。

```
1  # include <stdio.h>
2  # include <stdlib.h>
3
4  int main()
5  {
6      int i;
7      for (i = 0; i <= 10; i++)
8      {
9          if (i % 3 == 0) // if i is divisible by 3
10             continue;
11             printf("%d ", i);
12     }
13     printf("\n");
14 }
```

C:\Projects\continue ex.exe

1 2 4 5 7 8 10

應用：歐幾里得演算法 Euclidean Algorithm

應用：歐幾里得演算法 (Euclidean Algorithm)

- 歐幾里得演算法又稱輾轉相除法，是用來計算兩整數最大公因數 (GCD) 的演算法，可以使用迴圈控制來達成。
- 演算法如下：
 - ❶ 將兩整數代入到 $r = a \% b$ 的 a 與 b 中，求出 r 。
 - ❷ 將上一步的 b 與 r 的值重新代入 $r = a \% b$ 的 a 與 b 中，求出新的 r 。
 - ❸ 重複步驟 2，直到 r 等於 0。
 - ❹ 最後的 b 即是兩整數的最大公因數。

應用：歐幾里得演算法 (Euclidean Algorithm)

- 以下是使用歐幾里得演算法計算 $\text{GCD}(98, 35)$ 的過程：

$$r_0 = 98 \% 35, r_0 = 28$$

$$r_1 = 35 \% 28, r_1 = 7$$

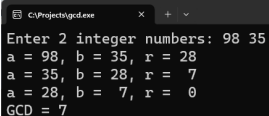
$$r_2 = 28 \% 7, r_2 = 0$$

$$\Rightarrow \text{GCD}(98, 35) = 7$$

應用：歐幾里得演算法 (Euclidean Algorithm)

- 歐幾里得演算法的程式實作：

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int a, b, r;
6     printf("Enter 2 integer numbers: ");
7     scanf("%d %d", &b, &r);
8     do {
9         a = b;
10        b = r;
11        r = a % b;
12        printf("a = %2d, b = %2d, r = %2d\n", a, b, r);
13    } while (r != 0);
14    printf("GCD = %d\n", b);
15 }
```



```
C:\Projects\gcd.exe
Enter 2 integer numbers: 98 35
a = 98, b = 35, r = 28
a = 35, b = 28, r = 7
a = 28, b = 7, r = 0
GCD = 7
```


函式 scanf 的回傳值 (Return Values of scanf)

函式 scanf 的回傳值 Return Values of scanf

函式 scanf 的回傳值 (Return Values of scanf)

- scanf 與 printf 在執行完畢之後，都會回傳一個整數：
 - scanf 會回傳成功讀入的數值個數
 - printf 會回傳印出的字元數

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int foo, bar, n;
7      n = scanf("%d %d", &foo, &bar);
8      printf("scanf() returned %d\n", n);
9      n = printf("foo = %d, bar = %d\n", foo, bar);
10     printf("printf() returned %d\n", n);
11 }
```

```
C:\Projects\scanf ex1.exe
1234 48763
scanf() returned 2
foo = 1234, bar = 48763
printf() returned 24
```

函式 scanf 的回傳值 (Return Values of scanf)

EOF (End of File)

- 當標準輸入流 (stdin) 來源為檔案時 (例如於線上測驗平台或於自己電腦執行時指定)，當 scanf 讀到檔案結尾時，會回傳一個特殊常數 EOF(數值為-1)。
- 鍵盤輸入檔案結尾：

Linux/Unix	ctrl + d
Windows	ctrl + z

函式 scanf 的回傳值 (Return Values of scanf)

- 使用 CMD 執行程式，指定輸入檔案，重複輸入直到 EOF：

The image shows a C program and its execution. The program, 'input until EOF.c', reads integers from a file 'my input.txt' until it reaches the end of the file (EOF). It calculates the sum of these integers. The execution is shown in a Windows command prompt window.

```
C:\input until EOF.c> ...
1  #include <stdio.h>
2
3  int main()
4  {
5      int value, sum = 0;
6
7      while (scanf("%d", &value) != EOF)
8      {
9          sum += value;
10     }
11     printf("Sum = %d\n", sum);
12 }
```

my input.txt

```
1 10
2 25
3 77
4 80
5 66
6 88
7 90
8
```

Microsoft Windows [版本 10.0.22621.3155]
(c) Microsoft Corporation. 著作權所有，並保留一切權利。

C:\Users\> cd C:\Projects

C:\Projects> gcc "input until EOF.c" -o "input until EOF.exe"

C:\Projects> "input until EOF.exe" < "my input.txt"

Sum = 436

Q & A