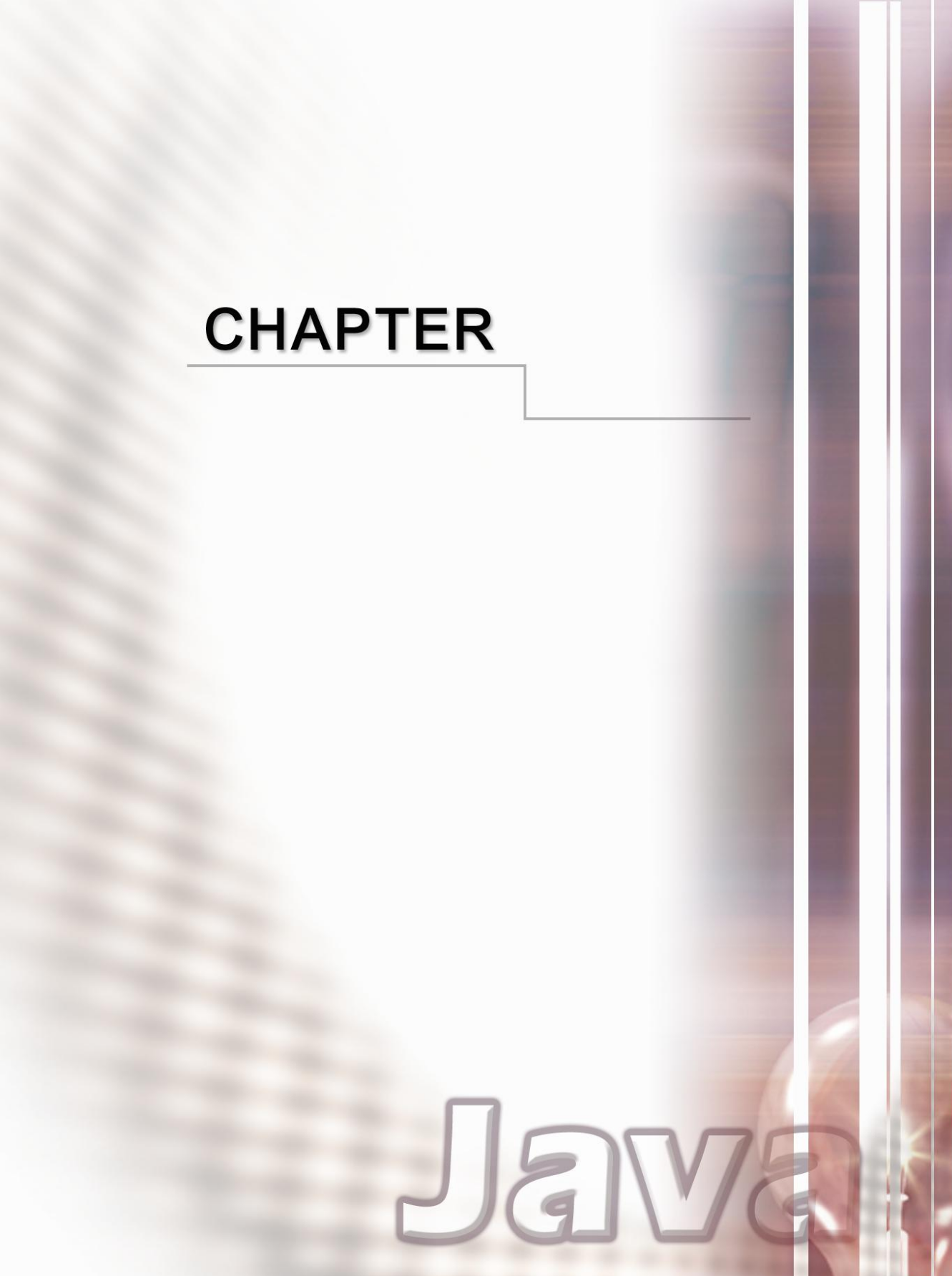
CHAPTER



撰寫一個好的程式是許多程式設計人員追求的目標，根據著名的「資料結構+演算法=程式」論點[註解]，我們可以說，要撰寫好的程式，就要先學習使用好的資料結構(data structure)及設計好的演算法(algorithm)。資料結構與演算法具有密不可分的關聯，當我們研究某一個資料結構時，必須一併研究存取此種資料結構的相關演算法，如此才可以深入了解資料結構中的資料是如何被存取的，並進而靈活應用資料結構。

本章的重點在於介紹資料結構與演算法的基本概念，包括資料結構與演算法的定義、演算法的表示與分析等。另外，由於物件導向程式設計(object oriented programming, oop)觀念於當今蔚為潮流，而資料結構又非常適合以物件導向(object oriented)的觀念來看待，因此，我們在本章中也介紹物件導向觀念與資料結構的關聯，以作為本書使用物件導向觀念介紹資料結構的起點。

1-1 資料結構的定義

資料(data)指的是可以輸入計算機中處理的數值、字元、字串甚至於是影像、視訊、音訊等訊號。在資料處理的範疇中，資料被定義為未經處理的事實紀錄，它們需要經過適當的處理，以便能夠轉換為可以提供人類決策參考的資訊(information)。當我們需要處理大量資料時，為加快資料處理速度，節省資料儲存空間，資料必須加以結構化。所謂資料的結構化就是以有系統的方式儲存與擷取資料，而許多以有系統的方式儲存與擷取的資料所構成的集合就稱為資料結構(data structure)，因此，我們對資料結構給定以下的定義：

> 資料結構(data structure)：
> 以有系統的方式儲存與擷取的資料所構成的集合

例如，以後進先出(Last In First Out, LIFO)的方式儲存與擷取的資料集合構成一個稱為堆疊(stack)的資料結構；而以先進先出(First In First Out, FIFO)的方式儲存與擷取的資料集合構成一個稱為佇列(queue)的資料結構等。

註解　「資料結構+演算法=程式」的論點源自一本名為「Algorithms+Data Structures=Programs」的經典書籍，此書由 Pascal 語言及 Modula-2 語言的創造人 Niklaus Wirth 所著，於 1975 年由 Prentice Hall 公司出版。

選擇適當的資料結構可以幫助我們解決特定的問題，例如，文書編輯器可以藉由堆疊來完成編輯復原(undo)動作。其作法為使用堆疊以後進先出的方式儲存與擷取文書編輯器的編輯動作(action)，則每次我們執行[復原(undo)]功能時，一定都是取消最後一個我們執行過的編輯動作。因此，只要我們有足夠的空間儲存堆疊的資料，理論上我們可以復原任何一個我們執行過的編輯動作。圖 1-1 所示是 Microsoft Word 軟體的「復原」按鈕提示，圖 1-2 是相對的堆疊內容變化過程。

註解　Word 軟體會將連續的輸入動作(如連續輸入 ABC 三個英文字母)視為一個單一動作，因此在復原時會一併取消，以加快復原速度。但若是游標曾經移動，則將輸入視為不連續的單一動作。

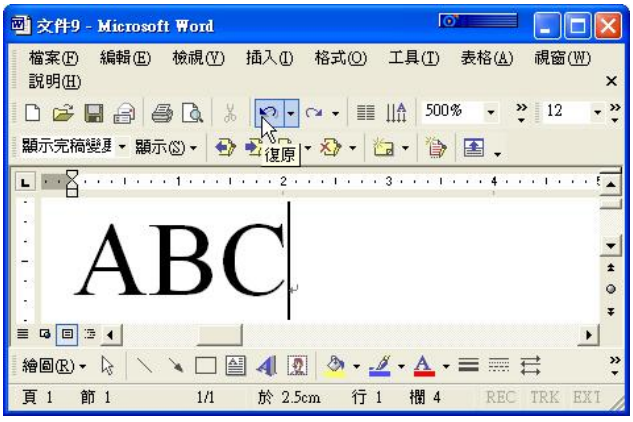


圖 1-1. 文書編輯器 Microsoft Word 的「復原」動作

次序	1	2	3	4	5	6
動作	輸入 A	輸入 B	輸入 C	復原 (undo)	復原 (undo)	輸入 Z
文件內容	A	AB	ABC	AB	A	AZ

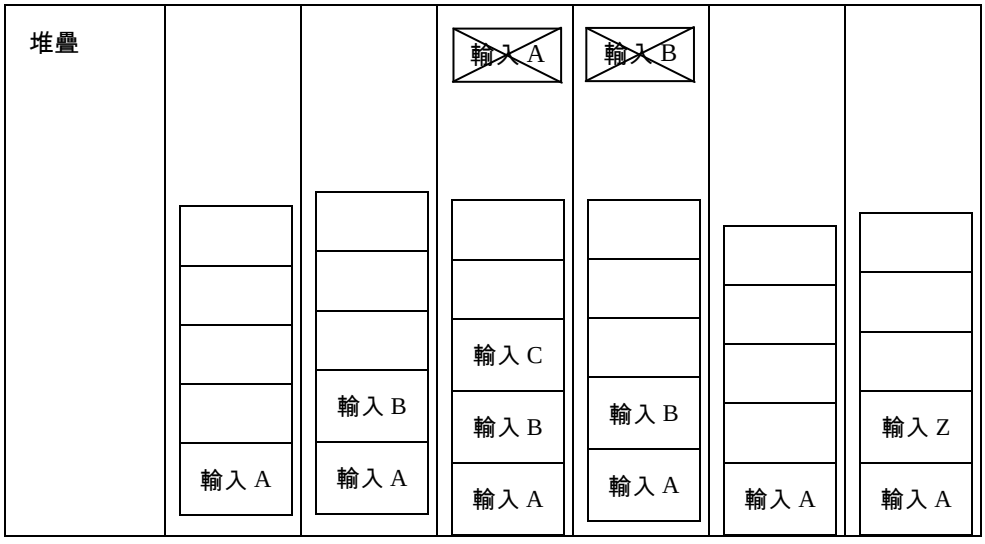


圖 1-2. 文書編輯器以堆疊(stack)後進先出(Last In First Out, LIFO)的方式儲存與擷取編輯動作

1-2 演算法的定義

一般而言，我們使用演算法(algorithm)[註解]以抽象的方式描述資料結構中儲存與擷取資料的方式。以下，我們即介紹演算法的定義：

> 演算法(algorithm)：
> 由有限的執行步驟所構成，可以用於解決某一個特定的問題。

演算法必須滿足以下五個條件：

1. 輸入(input)[提示]：可以由外界輸入 0 個、1 個或多個以上的資料。
2. 輸出(output)[提示]：至少要有 1 個以上的輸出。

3. 明確性(definiteness)：每個執行步驟都必須是明確而不含糊的。
4. 有限性(finiteness)：必須在有限執行步驟內結束。
5. 有效性(effectiveness)：每一個執行步驟必須是基本的(可行的)，也就是說，即使我們僅僅具有紙和筆也可以演算每一個執行步驟。

註解　演算法(algorithm)名稱來自於"al-Khwarizmi"，這是一個約在西元 780 年出生於伊拉克(Iraq)巴格達(Baghdad)城的阿拉伯數學家阿爾· 瓦里茲米(Abu Jafar Muhammad ibn Musa al-Khwarizmi)名字的最後一部份，此數學家將印度所發明的十進位數字記號傳入阿拉伯地區(稍後傳入歐洲並成為現今我們使用的數字記號)，並且著有一本名為"ilm al jabr w'al-muqabala"的書籍，此書有系統地討論一元二次方程的解法，啟發了代數學的發展。此書在 12 世紀被翻譯為拉丁文，名為 Algebra et Almucabala，(Algebra 源自阿拉伯書名中之 al jabr)，而成為代數學(algebra)一字的由來。

提示　請注意，當我們在本書中使用演算法描述資料結構中儲存與擷取資料的方法時，有時會省略演算法的輸入及輸出部份不寫，因為這些演算法的輸入很明顯的就是資料結構本身，而其輸出也經常是直接呈現於資料結構中。

1-3 演算法的表示

一般我們使用自然語言(中文或英文等語言)、流程圖(flow chart)、虛擬碼(pseudo code)或高階程式語言(high level programming language)來表示演算法。例如，以下的例子使用自然語言(中文與英文)描述一個古老的演算法 — 歐幾里德(Euclid)GCD 演算法。此演算法大約在西元前 300 年由希臘數學家歐幾里德提出，可用於求出二個整數的最大公因

數(GCD, Greatest Common Divisor)。

Euclid GCD 演算法:
問題：已知二個正整數 m 及 n，找出此二數的最大公因數(也就是能同時整除 m 及 n 的最大正整數)
解法：
E1.[找出餘數]　求出 m 除以 n 的餘數，並記錄於 r。
E2.[餘數為 0 嗎？]　如果 r=0 則停止，輸出 n 為 GCD。
E3.[互換]　設定 m=n,n=r，並跳至步驟 E1。

在本書中，我們採用虛擬碼及 Java 程式語言來表示演算法。虛擬碼以一種混雜著自然語言與高階程式語言結構的方式來描述演算法，試圖達到簡潔易讀、容易分析，而且也容易轉換為高階程式語言的目的。本書採用 Goodrich 及 Tamassia 在「Data Structures and Algorithms in JAVA」一書中所提出的虛擬碼格式來表達演算法，在本書後面的章節所稱的虛擬碼指的都是這種虛擬碼。

Goodrich 及 Tamassia 所提虛擬碼的撰寫規則如下：

- 方法宣告：以 Algorithm 方法名稱(參數 1,參數 2,…):來宣告一個演算法並指明其參數。例如，Algorithm Euclid-GCD(m, n):表示我們要定義一個具有兩個參數 m 及 n 而名稱為 Euclid-GCD 的演算法。
- 設定動作：以←表示，用以將一個算式之值存入某一個變數中。例如，m←3+8 表示要將 3+8 的值存入變數 m 中。
- 相等比較：以=表示，可以比較二個算式是否等值。例如，m=3+8 表示要比較 3+8 的值是否與變數 m 的值相等。
- 決策結構：以 if 條件 then 條件為真的動作 else 條件為偽的動作來

表示，並以縮排來表示所有包含在條件為真的動作及條件為偽的動作中的所有指令。例如：

```
if m=3+8 then
  b←3+8
else
  c←3+8
```

表示當變數 m 的值與 3+8 相等時，會執行將 3+8 的值存入變數 b 的動作，否則即執行將 3+8 的值存入變數 c 的動作。

- while 迴圈：以 while 條件 do 迴圈指令來表示，並以縮排來表示所有包含在迴圈中的所有指令。當條件成立時，會持續執行所有包含在迴圈中的指令。例如：

```
while m=3+8 do
  b←3+8
  c←3+8
```

表示當變數 m 的值與 3+8 相等時，會持續執行將 3+8 的值存入變數 b 與將 3+8 的值存入變數 c 的動作。

- repeat 迴圈：以 repeat 迴圈指令 until 條件來表示，並以縮排來表示所有包含在迴圈中的所有指令。此虛擬碼會持續執行所有包含在迴圈值中的指令，直到條件成立為止。例如，

```
repeat
  b←3+8
  c←3+8
until m=3+8
```

表示會持續執行將 3+8 的值存入變數 b 與將 3+8 的值存入變數 c 的動作，直到變數 m 的值與 3+8 相等為止。

- for 迴圈：以 for 迴圈範圍及變動 do 迴圈指令來表示，並以縮排

來表示所有包含在迴圈中的所有指令。例如，

```
for i←3 to 8 do
  b←3+8
  c←3+8
```

表示在變數 i 的值由 3 到 8(即 i 為 3、4、5、6、7、8)的情況下，會持續執行將 3+8 的值存入變數 b 與將 3+8 的值存入變數 c 的動作。

- 陣列元素索引：以 **A[i]**代表陣列 A 中註標(index)為 i 的元素，一個有 n 個元素的陣列，其元素註標為 0,1,…,n-1。例如，**A[5]** 表示陣列 A 中註標為 5 的元素。
- 方法呼叫：以物件.方法(參數…)來代表。注意，當物件是明確的時候，物件可以省略不寫。例如，Math.random()表示要呼叫 Math 物件無參數的 random()方法。
- 方法返回：以 return 方法返回值來代表。例如，return 3+8 表示要傳回 3+8 之值。

例如，以下就是使用虛擬碼來描述歐幾里得演算法的例子：

```
Algorithm EuclidGCD(m, n):
Input:二個正整數 m 及 n
Output:m 及 n 的最大公因數(GCD)
r← m%n
while r≠0 do
  m← n
  n← r
  r← m%n
return n
```

使用虛擬碼表示演算法可以方便演算法的分析，增加演算法的易讀

性，並且可以方便的將演算法轉變成可以直接在電腦上執行的程式語言程式碼。例如，以下為將以虛擬碼表示的歐幾里德演算法轉換為 Java 語言程式碼的例子：

```
int EuclidGCD(int m, int n) {
  int r=m%n;
  while (r!=0) {
    m=n;
    n=r;
    r=m%n;
  }
  return n;
}
```

上例中的 Java 語言程式碼以方法(method)的結構表示了 EuclidGCD 演算法。然而，要使上列的演算法能正確的執行，必須將上述的程式碼再進一步撰寫成 Java 語言中的類別(class)，並加上一些輸入與輸出敘述之後才可以進行編譯及執行動作。以下為將 EuclidGCD 演算法表示成為 Java 語言可以在瀏覽器上執行的小程式(applet)型式[提示]。

提示　實際上，除了轉換為 Java 程式語言以外，演算法也可以轉換為任何一種程式語言，這種轉換是為了讓演算法能夠在電腦上執行。

範例程式(檔名：EuclidGCDApplet.java)

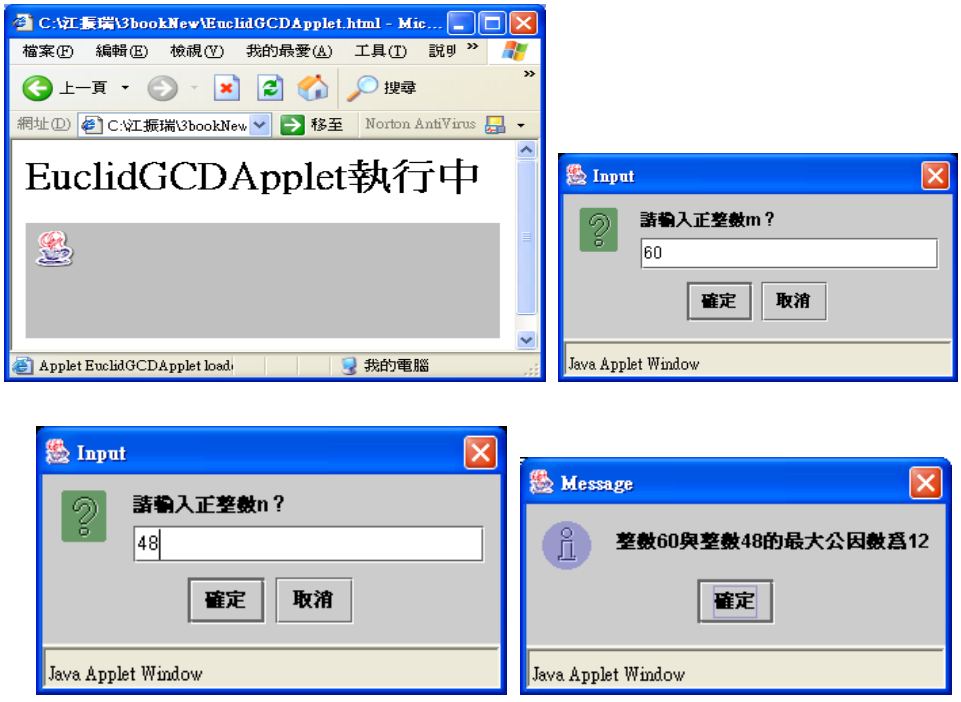
```
1：  //檔名:EuclidGCDApplet.java
2：  //說明:此程式可輸入二個整數，並以歐幾里得 GCD 演算法求其最大公因數(GCD)
3：  import javax.swing.*;
4：  public class EuclidGCDApplet extends JApplet{
5：    public void init( ){
6：      int 整數m,整數n,最大公因數;
7：      String 輸入字串1,輸入字串2,顯示字串;
```

```
8：      輸入字串1=JOptionPane.showInputDialog("請輸入正整數m？");
9：      輸入字串2=JOptionPane.showInputDialog("請輸入正整數n？");
10：     整數m=Integer.parseInt(輸入字串1);
11：     整數n=Integer.parseInt(輸入字串2);
12：     最大公因數=EuclidGCD(整數m, 整數n);
13：     顯示字串="整數"+整數m+"與整數"+整數n+"的最大公因數為"+最大公因數;
14：     JOptionPane.showMessageDialog(null,顯示字串);
15：   } //方法:init() 定義區塊結束
16：   static int EuclidGCD(int m, int n) {
17：     int r=m%n;
18：     while (r!=0) {
19：       m=n;
20：       n=r;
21：       r=m%n;
22：     }
23：     return n;
24：   } //方法:EuclidGCD() 定義區塊結束
25： } //類別:EuclidGCDApplet 定義區塊結束
```

網頁檔案(檔名：EuclidGCDApplet.html)

```
1：  <html>
2：  <h1>EuclidGCDApplet 執行中<h1>
3：  <applet code="EuclidGCDApplet.class" width=350 height=100>
4：  </applet>
5：  </html>
```

執行結果(以瀏覽器載入檔案：EuclidGCDApplet.html)



範例程式 1-1.歐幾里德 GCD 演算法於瀏覽器中執行情形[提示]。

提示　在本書後面的內容中，除非有特別的需要，我們將省略 Java 小程式(applet)在瀏覽器上執行的網頁畫面，而只列出彈出式輸入及輸出對話框。

如何撰寫、編譯並執行 Java Applet 小程式並非本書要探討的主題，讀者可以參考附錄 A 中的說明，或參考作者的另一本著作：「中文 Java 程式設計」以獲得足夠的程式設計知識。另外，讀者可以參考附錄 B，以得知如何安裝 Java 語言執行環境(JRE)與 Java 語言開發工具(JDK)。讀者也可以利用本書所附的「中文 Java 程式設計編輯器 2」－ Jeep2 軟體來執行 Java Applet 小程式，Jeep2 軟體可以直接模擬瀏覽器執行 Java Applet 小程式的效果，而不需另外編輯網頁檔案，這可大大縮短 Java Applet 小程式的

開發時間。Jeep2 軟體的使用方式在附錄 C 中有詳細的說明，讀者可以自行參考。

1-4 演算法的分析

設計一個演算法，首先要考慮的就是正確性(correctness)，也就是演算法是否能產生正確的結果，在達到正確性的要求之後，我們就要特別考慮演算法的有效性(effectiveness)了，也就是要考慮演算法是否能夠有效率的的執行。一般而言，若是數個演算法均能夠產生相同且正確的結果，則我們大都以有效性來衡量其優劣。而一個演算法的有效性，一般又以演算法的執行速度與佔用的記憶體空間大小來衡量，在學理上，我們使用時間複雜度(time complexity)及空間複雜度(space complexity)來評估演算法的執行速度與佔用記憶體空間。而由於記憶體的價格日趨便宜，因此，空間複雜度也漸漸的較不受到重視，在本書中，我們也大都只是針對時間複雜度提出評估，而少有空間複雜度的評估。

演算法的時間複雜度分析分為以下三種：

- 最佳狀況(best case)時間複雜度：考慮演算法執行時所需要的最少執行步驟數。
- 最差狀況(worst case)時間複雜度:考慮演算法執行時所需要的最多執行步驟數。

- 平均狀況(average case)時間複雜度：考慮所有可能狀況下演算法的平均執行步驟數。

一般而言，演算法的時間複雜度分析中，平均狀況時間複雜度最難求得，因而在某些狀況下，我們只考慮求出最佳狀況時間複雜度及最差狀況時間複雜度。另外，在三種狀況的時間複雜度中又以最差狀況時間複雜度的用處最大，因為這項複雜度可以讓我們了解演算法在最壞的情況下需要執行多少時間。

為了講解時間複雜度的分析，我們舉以下一個檢查整數是否為質數的演算法(Prime1)為例子來作說明。針對一個大於 1 的正整數而言，所謂質數(prime)是指除了 1 和本身之外沒有其他因數的整數，例如，2、3、5、7 等整數為質數，而 4、6、8、9 等整數不是質數(因為它們具有不是 1 與本身的因數)[註解]。以下是演算法 Prime1 的虛擬碼：

```
Algorithm Prime1(n):
Input:一個大於 2 的正整數 n
Output:true 或 false(表示 n 是質數或不是質數)
for i←2 to n-1 do
  if (n%i)=0 then return false
return true
```

註解　1 既不是「質數」也不是「非質數」，而 2 是最小的質數也是唯一一個為偶數的質數。

我們可以看出，輸入大於 2 的任意正整數 n，若 n 是質數，則演算法 Prime1 需要執行整數除法求餘數(n%i)動作與整數比較((n%i)=0)動作 n-2 次之後，才可以知道 n 是質數。另外，若 n 不是質值，則演算法 Prime1 只要執行整數除法求餘數與整數比較動作 1 次，就可以知道 n 不是質數了。因此，我們很容易看出來，在最壞狀況下，演算法 Prime1 的執行

步驟次數與輸入的正整數 n 的大小成正比；而在最佳狀況下，演算法 Prime1 的執行步驟次數為與輸入的正整數 n 無關的某個常數。也就是說，演算法 Prime1 具有呈現線性的最差狀況時間複雜度(正比關係即為線性關係，我們稍後會進一步說明)與呈現常數的最佳狀況時間複雜度。

一般而言，我們使用所謂的趨近記號(asymptotic notation)來分析演算法的複雜度，趨近記號考慮的是演算法在處理資料範圍趨近於無窮大時的狀況。在演算法的資料處理範圍(或處理資料量)較小時，不管是有效率的(執行步驟數較少)或是沒有效率的(執行步驟數較多)演算法通常都可以很快的執行完畢，在這個狀況下，演算法的時間複雜度比較不是那麼重要。而當演算法處理資料範圍(或處理資料量)相當大時，有效率的演算法通常還是可以很快的結束；而一些極度沒有效率的演算法則很可能需要幾日、幾月甚至於幾年才能執行完畢。因此，我們僅關心演算法在處理資料範圍(或處理資料量)非常大的狀況，這是為什麼分析演算法時間複雜度要採取趨近記號的原因。

在演算法處理資料範圍趨近於無窮大的狀況下，演算法的時間複雜度會趨近於一個量級(order)。一般而言，演算法的時間複雜度是一個多項式，當演算法處理資料範圍趨近於無窮大時，時間複雜度的多項式中除了最高次方的項目外，其他的部分都可以被忽略；而同時，最高次方項目的係數也同時可以被忽略。例如，若一個演算法的時間複雜度為 n-2，則當 n 相當大(趨近於無窮大)時，此演算法的時間複雜度趨近於 n，屬於一次方或稱線性(或正比)量級(稱為線性的原因是因為一次方多項式於平面座標上可描繪出直線圖形)；若一個演算法的時間複雜度為 $35n^2+12n+11$，則當 n 相當大(趨近於無窮大)時，此演算法的時間複雜度趨近於 n^2，屬於平方量級；而若一個演算法的時間複雜度為 $28n^3+1245n^2+162n+321$，則當 n 相當大時(趨近於無窮大時)，此演算法

的時間複雜度趨近於 n^3(屬於立方量級)。我們通常使用大 O 記號(Big-O notation)的來表示這種趨近情形，大 O 記號可以說是一個用來表示演算法時間複雜度量級(order)的記號，我們將時間複雜度與大 O 記號的關係整理如下：

> 一般而言，若是一個演算法的時間複雜度表示為一個多項式，則我們取這個多項式的最高次方為其時間複雜度的量級(order)，並且將此量級以大 O 記號表示(O 代表 order 之意)。

以下我們正式定義大 O 記號：

[定義] 大 O 記號 (Big-O notation)

令 $f(n)$與 $g(n)$是由非負整數對應至實數的函數，若存在正實數常數 $c>0$ 和正整數常數 n_0 使得對所有的 $n\geq n_0$ 而言，$f(n)\leq cg(n)$成立，則我們說 $f(n)=O(g(n))$。(唸作「$f(n)$是屬於 Big-O of $g(n)$」，比較正式的英文唸法為「f of n is of Big-O of g of n」)。

例如，針對 $35n^2+12n+11$ 而言，存在 $c=58$ 和 $n_0=1$(58 由 35+12+11 求得)，使得當 $n\geq n_0=1$ 時，$35n^2+12n+11\leq cn^2=(58n^2)$成立，因此，我們說 $35n^2+12n+11=O(n^2)$。

時間複雜度	以大 O 記號表示	量級
162	O(1)	一次方常數(constant)量級
63*log n*+4	O(*log n*)	次線性(對數)(sub-linear,

		logarithmic)量級
$37\sqrt{n}+52$	$O(\sqrt{n})$	平方根(square root)量級
n-2	$O(n)$	線性(linear)量級
$156n+81$	$O(n)$	線性(linear)量級
$35n^2+12n+11$	$O(n^2)$	平方(quadratic)量級
$28n^3+1245n^2+162n+321$	$O(n^3)$	立方(cubic)量級
$14\cdot 2n+457n^3+248n^2-45n+81$	$O(2^n)$	指數(exponential)量級

表 1-1. 演算法時間複雜度的大 O 記號表示及其量級

$\log n$	$\sqrt{n}$	n	$n \log n$	n^2	n^3	2^n
0	1.00	1	0	1	1	2
1	1.41	2	2	4	8	4
2	2.00	4	8	16	64	16
3	2.83	8	24	64	512	256
4	4.00	16	64	256	4,096	65,536
5	5.66	32	160	1,024	32,768	4,294,967,296

表 1-2.演算法各種時間複雜度的執行步驟對照數值

表 1-1 列出一些以大 O 記號表示的時間複雜度及其量級，而表 1-2、

圖 1-3 與圖 1-4 則顯示演算法時間複雜度在各種不同量級之下的比較，我們可以發現，某些量級在演算法的處理資料範圍或處理資料量(即問題大小，problem size)還不是很大的情況之下，演算法的時間複雜度的執行步驟對照數值(即執行時間，execution time)就已經是相當大的值了，這表示演算法需要運算相多的執行步驟，當然也就是要執行相當久的時間。因此，如何設計一個時間複雜度量級較低的演算法是我們必須一直擺在心中的最重要目標。

以下是演算法時間複雜度量級的高低次序比較，比較低的量級表示執行步驟較少，也就是代表演算法的執行時間較短、速度較快。這個量級的高低次序比較，可以很容易的由圖 1-3 及圖 1-4 得到印證。

$$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$$

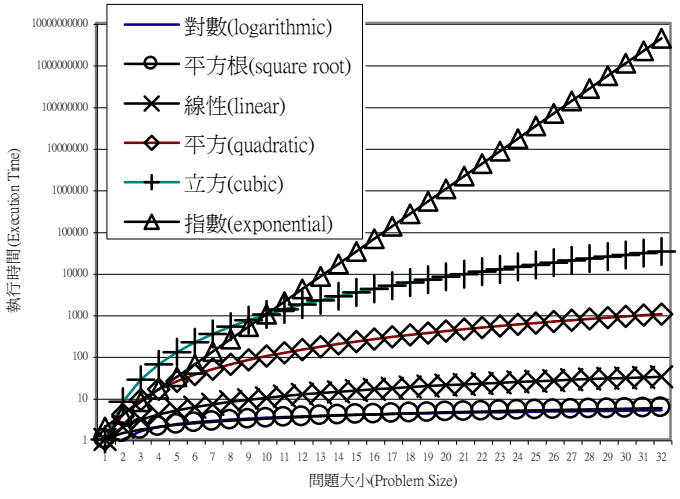


圖 1-3.演算法各種時間複雜度量級成長圖(對數圖)

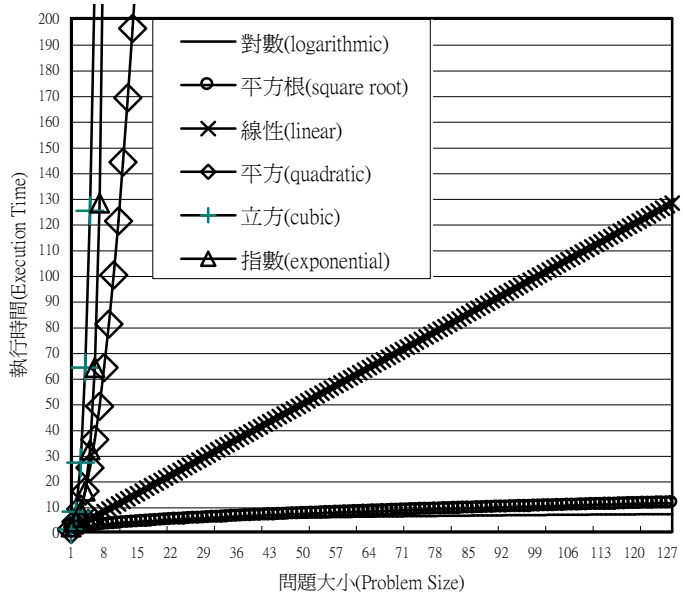


圖 1-4.演算法各種時間複雜度量級成長圖

以下，我們再舉一個檢查正整數是否為質數的演算法為例，來看看演算法的量級對執行時間的影響。我們先看以下的定理，若我們能善加利用它，則我們可以很容易的設計出一個比演算法 Prime1 量級更低的質數檢查演算法。

[定理][註解]

對於任意的大於 2 的整數 n 而言，若所有小於或等於 $\sqrt{n}$ 的整數(1 除外)都無法整除 n，則 n 是一個質數。(在此我們不討論定理的證明)

註解　每個整數 n 的因數都是成對出現的，例如，16 的因數為 1 與 16(1*16=16)、2 與 8(2*8=16)、4 與 4(4*4=16)。成對出現的因數中，一個會小於等於 n 的平方根，而另一個則是大於等於 n 的平方根，因此，我們只要檢查小於等於 n 的平方根的所有不等於 1 的整數中是否有 n 的因數就可以知道 n 是不是質數了。

我們根據上列的定理設計出一個屬於平方根量級的演算法—演算法 Prime2：

```
Algorithm Prime2(n):
Input:一個大於 2 的正整數 n
Output:true 或 false(表示 n 是質數或不是質數)
for i←2 to √n do
if (n%i)=0 then return false
return true
```

我們可以很輕易的推導出，演算法 Prime2 的最差狀況時間複雜度為 $\sqrt{n}$-1=O($\sqrt{n}$)，屬於平方根量級。而演算法 Prime2 與演算法 Prime1 一樣，其最佳狀況(best case)時間複雜度也是 1=O(1)，屬於常數量級。由圖 1-3 中我們可以看出，平方根量級的演算法 Prime2 的時間比線性量級的演算法 Prime1 的執行時間要少上許多。例如，若輸入的 n 為 2147483647，則演算法 Prime1 需要執行 2147483645 次整數除法求餘數和整數比較敘述才可以得知 2147483647 是質數，而演算法 Prime2 只要執行 $\sqrt{2147483647}$-1=46339 次整數除法求餘數和整數比較敘述就可以得知 2147483647 是質數了。二個演算法的執行時間差距約為 46340 倍，當然，當所輸入的 n 越大時，這種差距越大。我們將演算法 Prime1 和演算法 Prime2 以 Java 語言來表示，並再其中插入測量執行時間相關指令來比較二個演算法實際的執行時間。當輸入的整數 n 為 2147483647 時，演算法 Prime1 需要執行 89078 毫秒(約 1.48 分鐘)，而演算法 Prime2

則僅需要不到 1 毫秒(因此執行結果顯示 0 毫秒)。理論上，當輸入的整數 *n* 為 2147483647 時，二個演算法所需的執行時間差距超過 46340 倍。從這二個演算法的執行情形，我們可以明確的看出演算法的好壞對執行時間的影響，尤其當演算法所面對的處理資料範圍(或處理資料量或問題規模)比較大時，量級較高與量級較低的演算法之間的執行時間差距更是可觀。

範例程式(檔名：Prime1Applet.java)

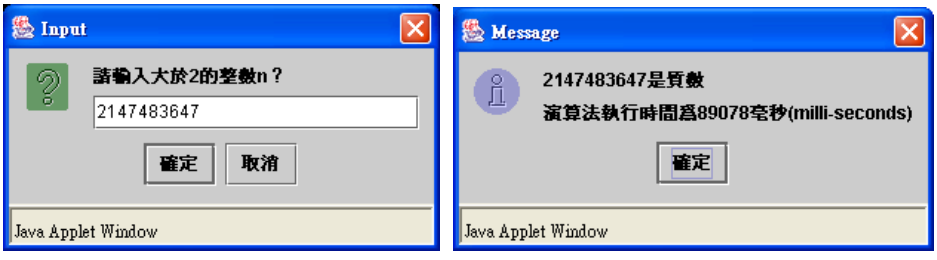
```
1：  //檔名:Prime1Applet.java
2：  //說明:此程式可輸入一個大於 2 的整數，並判斷此整數是否為質數(prime number)
3：  import javax.swing.*;
4：  public class Prime1Applet extends JApplet {
5：    public void init(){
6：      int 整數 n;
7：      String 輸入字串,顯示字串;
8：      輸入字串=JOptionPane.showInputDialog("請輸入大於 2 的整數 n？");
9：      整數 n=Integer.parseInt(輸入字串);
10：     boolean 是質數;
11：     long 演算法執行前時間=System.currentTimeMillis();
12：     是質數=Prime1(整數 n);
13：     long 演算法執行後時間=System.currentTimeMillis();
14：     long 演算法執行時間=演算法執行後時間-演算法執行前時間;
15：     if(是質數) 顯示字串=整數 n+"是質數";
16：     else 顯示字串=整數 n+"不是質數";
17：     顯示字串+=("\n 演算法執行時間為"+演算法執行時間+"毫秒(milli-seconds)");
18：     JOptionPane.showMessageDialog(null,顯示字串);
19：   } //方法:init() 定義區塊結束
20：   boolean Prime1(int n) {
21：     for (int i=2;i<=n-1;++i)
22：       if (n%i==0) return false;
23：     return true;
24：   }
```

```
25：  } //類別:Prime1Applet 定義區塊結束
```

網頁檔案(檔名：Prime1Applet.html)

```
1：   <html>
2：   <h1>Prime1Applet 執行中<h1>
3：   <applet code="Prime1Applet.class" width=350 height=100>
4：   </applet>
5：   </html>
```

執行結果(以瀏覽器載入檔案：Prime1Applet.html)



範例程式 1-2.質數檢查演算法 Prime1 於瀏覽器中執行情形

範例程式(檔名：Prime2Applet.java)

```
1：   //檔名:Prime2Applet.java
2：   //說明:此程式可輸入一個大於 2 的整數，並判斷此整數是否為質數(prime number)
3：   import javax.swing.*;
4：   public class Prime2Applet extends JApplet {
5：     public void init(){
6：       int 整數 n;
7：       String 輸入字串,顯示字串;
8：       輸入字串=JOptionPane.showInputDialog("請輸入大於 2 的整數 n？");
9：       整數 n=Integer.parseInt(輸入字串);
10：      boolean 是質數;
```

```
11：     long 演算法執行前時間=System.currentTimeMillis();
12：     是質數=Prime2(整數 n);
13：     long 演算法執行後時間=System.currentTimeMillis();
14：     long 演算法執行時間=演算法執行後時間-演算法執行前時間;
15：     if(是質數) 顯示字串=整數 n+"是質數";
16：     else 顯示字串=整數 n+"不是質數";
17：     顯示字串+=("\n 演算法執行時間為"+演算法執行時間+"毫秒(milli-seconds)");
18：     JOptionPane.showMessageDialog(null,顯示字串);
19：   } //方法:init() 定義區塊結束
20：   boolean Prime2(int n) {
21：     for (int i=2;i<=Math.sqrt(n);++i)
22：       if (n%i==0) return false;
23：     return true;
24：   }
25： } //類別:Prime2Applet 定義區塊結束
```

網頁檔案(檔名：Prime2Applet.html)

```
1：  <html>
2：  <h1>Prime2Applet 執行中<h1>
3：  <applet code="Prime2Applet.class" width=350 height=100>
4：  </applet>
5：  </html>
```

執行結果(以瀏覽器載入檔案：Prime2Applet.html)

範例程式 1-3.質數檢查演算法 Prime2 於瀏覽器中執行情形

1-5 演算法分析常用公式

演算法的分析因為需要求取執行步驟或所佔記憶體的總數，因此，經常需要用到許多與總合相關的數學公式。在本節中，我們列出幾個常用的數學記號或數學公式，例如，表 1-3 及表 1-4 中所列的分別是常用的數學記號及數學公式。

(1) $\lfloor x \rfloor$ = 比 x 小或等於 x 的最大整數
(2) $\lceil x \rceil$ = 比 x 大或等於 x 的最小整數
(3) $\lg n = \log_2 n$ = 以 2 為基底的對數值
(4) $\ln n = \log_e n$ = 以 e 為基底的對數值=自然對數 ($e \approx 2.71828$)

表 1-3. 常用的數學記號

(1) $\log_c (ab) = \log_c a + \log_c b$

(2) $\log_c \frac{a}{b} = \log_c a - \log_c b$

(3) $\log_b a = \frac{\log_c a}{\log_c b}$

(4) $\log_b a = \frac{1}{\log_a b}$

(5) $a^{\log_b n} = n^{\log_b a}$

(6) $\log_{b^r} a = \frac{1}{r} \log_b a$

(7) $\sum_{i=1}^{n} i = 1 + 2 + \Lambda + n = \frac{n(n+1)}{2}$

(8) $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \Lambda + n^2 = \frac{n(n+1)(2n+1)}{6}$

(9) $\sum_{i=1}^{n} i^3 = 1^3 + 2^3 + \Lambda + n^3 = \frac{n^2(n+1)^2}{4}$

(10) $\sum_{i=0}^{n} a^i = 1 + a + a^2 + \Lambda + a^n = \frac{a^n - 1}{a - 1}$

表 1-4. 常用的數學公式

以下我們舉二個利用表 1-4 公式進行時間複雜度分析的例子，假設有一個資料結構的「資料處理」操作的演算法如下：

```
Algorithm 資料處理(n):
Input: n 為處理的資料總數
for i← 1 to n do
j← i
for k← j+1 to n do
   進行處理;
```

則這個資料結構的資料處理演算法的「進行處理」步驟的執行總數為 $\frac{n(n-1)}{2}$=O(n^2)，因為「進行處理」步驟的執行總數=

$$\sum_{i=1}^{n}\sum_{k=i+1}^{n} 1 = \sum_{i=1}^{n}[n-(i+1)+1] = \sum_{i=1}^{n}[n-i] = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i = n^2 - \left(\frac{n(n+1)}{2}\right) = \frac{n(n-1)}{2}$$

我們再舉另外一個例子，假設有一個資料結構的「資料處理」操作的演算法如下：

```
Algorithm 資料處理(n):
Input: n 為處理的資料總數
for i←1 to n do
   for j←i to n do
      for k←j to n do
         進行處理;
```

則這個資料結構的資料擷取演算法需要執行的「進行處理」步驟的執行總數為 $\frac{n(n+1)(n+2)}{6}=O(n^3)$，因為，「進行處理」步驟的執行總數 =

$$\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j}^{n}1=\sum_{i=1}^{n}\sum_{j=i}^{n}(n-j+1)=$$

$$=\sum_{i=1}^{n}\left[\sum_{j=i}^{n}(n+1)-\sum_{j=i}^{n}j\right]$$

$$=\sum_{i=1}^{n}\left[(n+1)(n-i+1)-\sum_{j=i}^{n}j\right]$$

$$=\sum_{i=1}^{n}\left[(n+1)(n-i+1)-\frac{(n+i)(n-i+1)}{2}\right]$$

$$=\sum_{i=1}^{n}\left[\left(n+1-\frac{n+i}{2}\right)(n-i+1)\right]$$

$$=\sum_{i=1}^{n}\left[\frac{1}{2}(n+2-i)(n-i+1)\right]$$

$$=\frac{1}{2}\sum_{i=1}^{n}[(n+2-i)(n+1-i)]$$

$$=\frac{1}{2}\sum_{i=1}^{n}\left[(n+2)(n+1)-(2n+3)i+i^2\right]$$

$$=\frac{1}{2}(n+2)(n+1)\sum_{i=1}^{n}1-\frac{1}{2}(2n+3)\sum_{i=1}^{n}i+\frac{1}{2}\sum_{i=1}^{n}i^2$$

$$=\frac{n(n+2)(n+1)}{2}-\frac{n(2n+3)(n+1)}{4}+\frac{n(n+1)(2n+1)}{12}$$

$$=\frac{n(n+1)(2n+4)}{12}=\frac{n(n+1)(n+2)}{6}$$

1-6 物件導向觀念與資料結構

物件導向(object oriented) 觀念普遍用於資訊技術領域，這個觀念可以讓我們輕易的設計出容易修改的、容易維護的及容易重複使用的系統。因此，若我們也使用此一觀念來看待資料結構，則也可以讓資料結構有容易設計、容易修改、容易維護及容易重複使用的好處。

所謂物件(object)是包含屬性(attribute)與操作(operation)[提示]的東西，一般而言，物件的屬性僅能透過物件本身的操作加以改變。前述定義中所謂的東西，可以是實體存在的或僅是概念上的。例如，實體存在的汽車可以視為一個物件，它具有車長、車重、顏色、速度、方向等屬性，也具有可以改變速度與方向屬性的加速、減速、煞車、轉向等操作；而概念上的會議也可以視為物件，它具有會議日期、會議地點、會議名稱、會議參加者等屬性，也具有可以改變各屬性的更改日期、更改地點、新增參加者等操作。

提示　在本書後面的內容中，我們交替使用「操作(operation)」與「方法(method)」這兩個術語；也就是說，我們將這二個術語視為是相同的。

物件導向觀念具有以下三個特色[註解]：

- 多型 (polymorphism)
- 繼承 (inheritance)
- 封裝 (encapsulation)

就是這三個特色讓我們可以輕易的設計出容易修改的、容易維護的及容易重複使用的系統。其中，多型特性指的是同樣名稱的操作在不同的物件中(甚至於同一個物件中)可以有不同的做法，繼承特性指的是物件可自其他物件延續使用其屬性及操作，而封裝特性指的是我們僅能透過物件本身的操作來變更物件的屬性。在專門探討物件導向觀念的書籍中，針對這三個特性均有深入的探討，讀者可以自行參考這些書籍以得到更進一步的資訊。

註解　有人以 PIE 來稱呼物件導向觀念的三大特色 — 多型 (polymorphism)、繼承 (inheritance)及封裝 (encapsulation)。

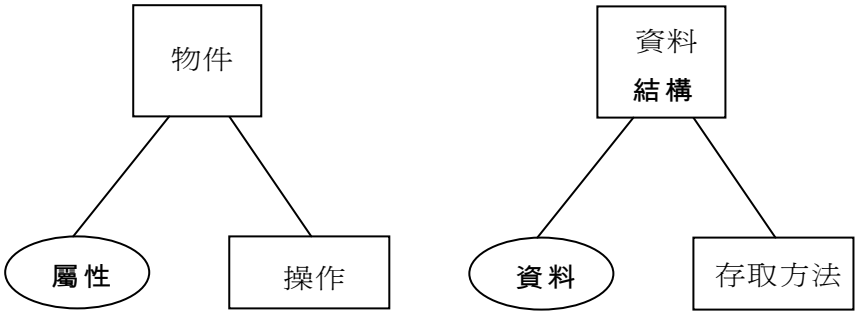


圖 1-5. 物件導向觀念與資料結構

以物件導向的觀念來看待資料結構是再恰當不過了，因為資料結構包含二大要素：一為資料結構的資料本身，而另一為資料的儲存與擷取方法(演算法)，若我們將資料結構視為物件，則資料結構的資料就是屬性，而資料的儲存與擷取方法就是操作。圖 1-5 中顯示了資料結構與物件導向觀念的關聯，在本書後面的內容中，我們將每一個資料結構都視為物件，並善用物件導向觀念來讓我們輕鬆的設計出容易修改的、容易

維護的及容易重複使用的資料結構。希望讀者研讀完本書之後，能夠同時對「資料結構」、「演算法」及「物件導向觀念」有基本的概念。

習題

1. 資料結構的定義為何？演算法的定義為何？二者之間有何關聯？
2. [88 年五等關務特考]

   為了求解某一個問題而定出一套以文字敘述之法則來說明工作處理法與程式稱為？ (A)運算法則 (B)流程圖 (C)結構圖 (D)直譯程式。
3. [83 年普考]

   下列對於演算法(Algorithm)具備之條件的敘述何者為錯誤？ (A)至少須有由外界供給一輸入 (B)至少須有一個以上的結果輸出給外界 (C)指令動作必須明確，不可似是而非 (D)必須在有限個步驟內完成結果。
4. [82 年普考]

   下列何者不是計算方法(Algorithm)的要素？ (A)每個執行步驟一定要明確 (B)每個執行步驟一定在有限時間內執行完畢 (C)一定要有輸出 (D)一定要有輸入。
5. 設計一個時間複雜度為 $O(\sqrt{n})$的演算法，求一個正整數 n 有多少個因數。以虛擬碼表示你的演算法，並說明為何其時間複雜度為 $O(\sqrt{n})$？
6. 設計一個最差時間複雜度為 $O(\sqrt{n})$的演算法，求一個正整數 n 不等

於本身的最大因數。以虛擬碼表示你的演算法，並說明為何其最差時間複雜度為 O($\sqrt{n}$)？

7. 設計一個最佳時間複雜度為 O(1)而最差時間複雜度為 O($\sqrt{n}$)的演算法，求一個正整數 n 不等於本身的最大因數。以虛擬碼表示你的演算法，並說明為何其最佳時間複雜度為 O(1)而最差時間複雜度為 O($\sqrt{n}$)？

8. 設計一個時間複雜度為 O($\sqrt{n}$)的演算法，求一個正整數 n 所有因數的總合。以虛擬碼表示你的演算法，並說明為何其時間複雜度為 O($\sqrt{n}$)？

9. 設計一個時間複雜度為 O($\sqrt{n}$)的演算法，求一個正整數 n 是否為完美數(perfect number)。以虛擬碼表示你的演算法，並說明為何其時間複雜度為 O($\sqrt{n}$)？

   註：一個整數被稱為完美數(perfect number)若它等於除了它自身之外所有因數的和。例如，1+2+3=6，因此 6 為 perfect number；而 1+2+4≠8，因此 8 不是 perfect number。

10. 設計一個演算法，求小於正整數 n 的所有完美數(perfect number)。以虛擬碼表示你的演算法，並求其時間複雜度？

    註：一個整數被稱為完美數(perfect number)若它等於除了它自身之外所有因數的和。例如，1+2+3=6，因此 6 為 perfect number；

而 1+2+4≠8，因此 8 不是 perfect number。

11. 設計一個演算法，求一個正整數 m 與一個正整數 n 是否互質。以虛擬碼表示你的演算法，並求其時間複雜度？

12. 請以大 O 記號表示下列函數？

$$(1)\sum_{i=1}^{n}1 \qquad (2)\sum_{i=1}^{n}i \qquad \sum_{i=1}^{n}i^2$$

13. [87 年資料處理高考]

試決定下列片斷程式中的指令 x← x+1 的執行次數？

```
(1)for i← 1 to n do
     for j← 1 to n do
          x← x+1
     end
    end
```

```
(2)for i← 1 to n do
     for j← i to n do
          for k← j to n do
               x← x+1
          end
     end
    end
```

```
(3)for i← 1 to n do
     j← i
     for k← j+1 to n do
          x← x+1
     end
    end
```

```
(4)k← 100000
    while k≠5 do
     k← k div 10
```

```
    x← x+1
  end
```

14. [88 年普考]
若一個程式執行的時間是 $1000n^2+n\ log\ n^2$，則相當於：(A)O(n^2) (B)O($n\ log\ n$) (C)O(n^3) (D)O($n^2\times n\ log\ n^2$)。

15. 試證明 $\sum_{i=1}^{n}(i+32)$ =O(n^2)。

16. 請列出物件導向觀念的三大特色，並大略說明之？

17. 請繪圖說明資料結構與物件導向觀念的關聯？