Chapter 8 高階程式語言

- 目標--- 研讀完本章後，你應該可以：
  - 描述翻譯的過程並能區別組譯(assembly)、編譯(compilation)、直譯(interpretation)及執行(execution)的不同。
  - 命名四種不同程式應用範例並描述每一種語言的特性。
  - 描述以下的結構：輸入及輸出資料流、選擇、迴圈及副程式。
  - 建立布林表示式並描述如何使用它們來改變演算法控制流程。
  - 定義資料型態及強制型態(strong typing, 強行鍵入?)的概念。
  - 解釋參數的概念並能區別數值及參考參數的不同。(下學期)
  - 描述兩種複合型 (composite，組成型)資料結構的機制。(下學期)
  - 說出(name, 命名?)、描述及舉例說明一個物件導向程式組成的三個要素。(下學期)

8.1 翻譯過程

- 編譯過程：編譯器
  - 高階語言是提供一個較豐富的指令集來使程式設計師的工作更容易一些。
  - 專門用來翻譯高階語言寫成的程式至機器碼語言程式的這種程式叫作編譯器 (compiler)。

圖 8.1 編譯過程

直譯器

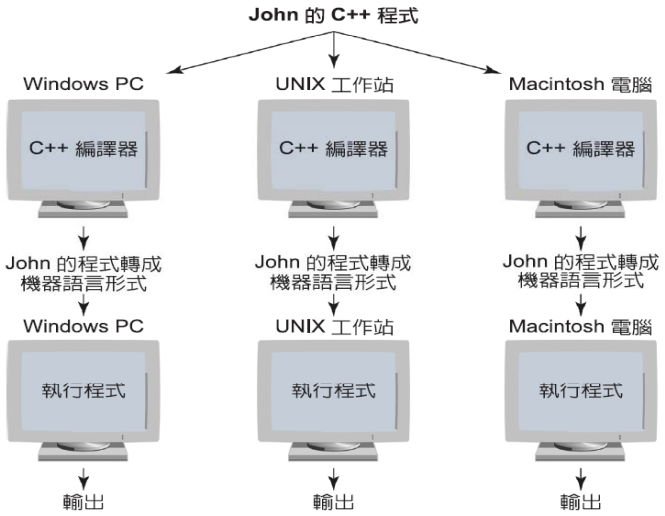
- 一個直譯器 (interpreter) 是一個翻譯程式，它依序地翻譯並執行程式中的每一行敘述。
  - 不同於組譯器及編譯器會先產生機器碼輸出然後再於另一個步驟中執行，直譯器是翻譯完一行敘述就立即執行這一行敘述，因此執行速度較慢。
  - 直譯器可以被看作是程式所寫成的語言之**模擬器** (*simulator or virtual machine*)。

直譯器

- Java程式語言是在1996年推出，然後就像一場颶風般橫掃整個電腦界。
  - 在Java的設計上，可攜性 (portability) 是至為重要的。
  - Java將程式編譯成一個標準的機器語言叫作位元組碼 (Bytecode)，此位元組碼不是針對特定硬體 CPU 的機器語言。
  - 各硬體機器只需要一個軟體直譯器叫做 JVM（Java Virtual Machine, Java虛擬機器）來載入位元組碼並執行它，因此位元組碼具有可攜性

直譯器

(a) 一個 C++ 程式在不同系統上編譯與執行



直譯器

(b) Java 程式編譯成位元組碼並在不同系統上執行

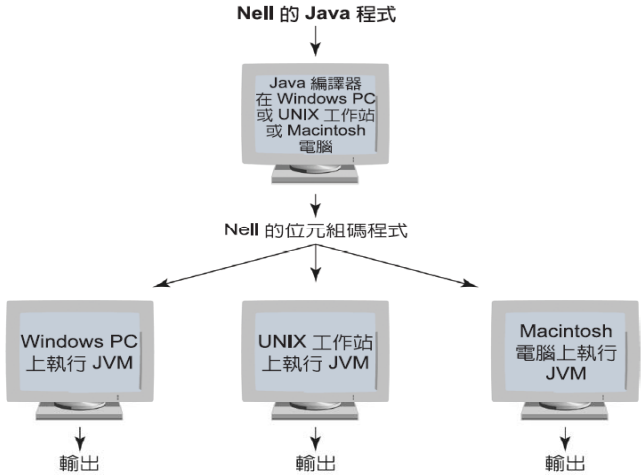


圖 8.2 標準化程式語言與位元組碼直譯器所提供的可攜性之不同

8.2 程式語言架構

- **架構** (*paradigm*)
  - 假設、觀念、數值與實作的集合所構成的方法，讓一些特別是心智訓練的群體能夠了解實體的狀況
- 程式語言架構有 4 種：
  1. **命令式** (*imperative*) 的或是**程序式** (*procedural*) 的模式。程式是描述解決問題所需步驟，以變數代表記憶體位置
     - 有FORTRAN、COBOL、BASIC、C、Pascal、Ada及C++ 等。這是最普遍的程式語言架構
  2. 另一個計算的模式是**函數** (*functional*) 模式。計算過程以函數演算來表達。
     - 有Lisp、Scheme (由Lisp演變而來) 及ML。
     - 例如 (+ 30 40) 表示 "+" 這個函數對其後兩個數字進行運算後，傳回 70，(+ 30 (+ 20 40))，傳回 90

程式語言架構

3. **邏輯程式** (*logic programming*) 是第三種程式架構。
   - PROLOG是一種在1970年於法國發展出來的第三代邏輯式程式語言。
   - 例如：由 "父子關係" 可定義 "祖先-子孫 關係"
     - ancestor(X, Z) :- ancestor(X, Y), parent (Y, Z)
     - ancestor(X, Z) :- parent (X, Z)
4. 第四種結構是**物件導向** (*object-oriented*) 架構。
   - SIMULA和Smalltalk是首先問世的兩個物件導向程式語言。
   - C++ 也被人認為是一種物件導向語言，但是我們將它視為一種命令式語言兼具部分物件導向特性。
   - Java語言則被視為物件導向語言兼具部分命令式語言特性。

8.3 命令式語言的功能

- 有兩種方法我們可以用來設計演算法：
  - 選擇結構 (使用條件敘述)。
  - 重複結構 (使用迴圈敘述)。
- 布林表示式是高階語言用來做選擇及控制迴圈的結構

布林表示法

需求：每次讀入一對數字，並依其大小由小至大依序列印
演算法：

```
Write "How many pairs of values are to be entered?"
Read numberOfPairs
Set numberRead to 0
while (numberRead < numberOfPairs)
    Write "Enter two values separated by a blank; press return"
    Read number1
    Read number2
    If (number1 < number2)
       Print number1 + " " + number2
    Else
       Print number2 + " " + number1
    Increment numberRead
```


布林表示法

```
(numberRead < numberOfPairs)
(number1 < number2)
```

  - 這些敘述就叫作**判斷式** (*assertion*) 或是**條件式** (*condition*)。
- 一個布林表示式 (Boolean expression) 是一串列的識別子，以相匹配的運算子分開，運算得到真(true)或偽(false)的值。一個布林表示式可以是
  - 一個布林變數(如以往定義一個整數變數 COUNT 一般，我們可以定義一個布林變數 T1 ，他只有 2 種可能值 TRUE 及 FALSE)。例如 Bool T1 = (x < y)
  - 一個算術表示式跟著一個關係運算子再跟著一個算術表示式。 如 x + y <= z^2
  - 一個布林表示式跟著一個布林運算子再跟著一個布林表示式。(如 x < y AND z < x ， 又如 T1 OR y < z)
- 一個布林變數是一個在記憶體中的位置含有真或偽的值，可以讓一個識別子參考到它。

布林表示法

符號	意義	範例	運算式
<	小於	Number1 < Number2	如果 Number1 小於 Number2 時為真；否則為偽
<=	小於或等於	Number1 <= Number2	如果 Number1 小於或等於 Number2 時為真；否則為偽
>	大於	Number1 > Number2	如果 Number1 大於 Number2 時為真；否則為偽
>=	人於或等於	Number1 >= Number2	如果 Number1 大於或等於 Number2 時為真；否則為偽
!= 或 <> 或 /=	不等於	Number1 != Number2	如果 Number1 不等於 Number2 時為真；否則為偽
= 或 ==	等於	Number1 == Number2	如果 Number1 等於 Number2 時為真；否則為偽

布林表示法

- 如果兩個算術運算式存在某種關係，那就可以在兩個算術運算式中間放入關係運算子。
  - 例如，

    xValue < yValue
  - 如果 xValue 確實小於 yValue，則這個表示式的結果為真；如果 xValue 不小於 yValue，則這個表示式的結果為 偽

強制型態

- 強制型態 (strong typing)
  - 組合語言不需考慮資料型態，但一般高階語言必須說明是那一種值(如整數、單精確度浮點數、字元、布林值等)會被儲在變數的記憶體位置
  - 只有正確型態的數值才能被儲存到一個變數中的要求就是強制型態
- 資料型態 (data type)
  - 是一種描述數值的集合與能被使用到這種型態數值的基本運算法的集合。
  - 例如：假設用一個位元組表示 2 補數整數，則整數變數的值的範圍由 -2^7 到 2^7-1，而可以使用到這種型態數值的基本運算子為 + - * / mod > < == 等

資料型態

- 大多數的高階語言都有以下四種不同的資料型態來建構程式語言：整數、實數、字元與布林值。
- 整數
  - 範圍的大小是依據有多少位元組被指定來代表一個整數值。
  - 有些高階語言提供幾種不同位元組樹目的整數型態，可以讓使用者自行決定一個適合於特定問題的資料。如 短整數(SHORT)、整數(INTEGER)、長整數(LONG)
  - 可以應用在整數的運算是標準的算術及關係運算子
- 實數
  - 這個範圍的大小是依被指定來代表實數的位元組數量而定。
  - 許多高階語言有兩種範圍的實數：如 單精確度浮點數(SINGLE 或 FLOAT)、雙精確度浮點數(DOUBLE)
  - 可以應用在實數上的運算與那些可應用在整數的運算是一樣的。

資料型態

- 字元
  - ASCII字元集裡使用1個位元組來代表字元，在Unicode字元集裡是使用2個位元組來代表字元。
  - 我們使用的英文字母是以ASCII碼表示，同時也是Unicode的一個子集。使用算術運算在字元上並沒有什麼意義。
  - 但是字元間的比較是有些意義的，所以關係運算子可以用在字元上。
  - 在字元間的「小於」與「大於」的意思代表著在字元集裡「排在前面」與「排在後面」的涵義。
    - 在 ASCII 字元集中，’ 0’ < ‘9’ < ’A’ < ‘Z’ < ‘a’ < ‘z’
    - 在 BIG-5 碼， ‘中’ < ‘淡’ < ‘難’

資料型態

- 布林值
  - 布林值資料型態只有兩個值：true (真) 與false (偽)。
  - 並不是所有高階語言都有支援布林值資料型態。
  - 如果某一種程式語言沒有支援布林值資料型態，那麼你可以自行指定布林值true代表1及布林值false代表0來模擬布林值。
  - 依照二進位制，應該只要 1 個位元 (bit) 就可以表示這 2 種值，但是因為我們需要用一個記憶體位址來代表此變數，因此一般的機器上會用一個位元組來代表***真(TRUE)*** 或 ***偽(FALSE)*** 的值
    - 當此位元組為 0000 0000 時，其值為***偽(FALSE)***
    - **其他 63 種情況時，**其值為***真(TRUE)***

資料型態

- 字串
  - 一個字串型態是一個字元的串列，通常被視為單一資料數值。例如

    “This is a string.”

    是一個含有17個字元的字串：一個大寫字母，12個小寫字母，3個空白字元，與一個句點所組成。

    一般以雙引號框住字串，以單引號框住字元
  - 對於字串的運算方式，不同程式語言間的定義差異很大。
  - 包含有字串的串接與依字典排序方式的字串比較等。
    - 例如：“abc” + “xyz” 結果為 “abcxyz”，
    - ”abc” < “xyz” 、 “xy” < “xz”、 “xy” < “xyz” 等為 TRUE

宣告

- 宣告 (declaration)
  - 組合語言的巨集 (directive，如 .BLOCK .WORD 等) 為特定記憶體位置命名、給定初始值，但沒有資料型態的觀念
  - 高階語言的一個宣告基本上是一個敘述，用以關聯一個識別子到一個變數、一個動作或一些其他在語言中的實體實體 (如一個布林表示式)，賦予它們一個名字以便程式設計者稍後能以這個名字參考到這些實體。

VB.NET

```
Dim sum As Single = 0.0F 'set up word with 0 as contents
Dim num1 As Integer      'set up a two byte block for num1
Dim num2 As Integer      'set up a two byte block for num2
Dim num3 As Integer      'set up a two byte block for num3
. . .
num1 = 1
```

C++/Java

```
float sum = 0.0; // set up word with 0 as contents
int num1;        // set up a two byte block for num1
int num2;        // set up a two byte block for num2
int num3;        // set up a two byte block for num3
. . .
num1 = 1;
```


宣告

- 保留字 (reserved word)
  - 就是在程式語言中有特殊意義的字，它不能被用作識別子。如資料型態、及之前虛擬碼提過的 IF、WHILE 等
- 大小寫有別 (case-sensitive) 的語言表示無論是大寫或小寫字母都被視為是不同的[訂正 p.245]。，如 ADA 、 Ada 、ada … 都被視為不同
  - Ada 不是大小寫有別，VB.NET、C++ 與 Java 則是大小寫有別

指定敘述

- **指定敘述（assignment statement ，也可翻譯為設定敘述）**是一種程式執行動作的敘述（不是宣告），它將指定敘述符號右手邊的數學表示式計算後的結果值儲存到左手邊變數的記憶體位置
  - 如 SUM = SUM + NUM
  - **指定敘述的**左手邊一定要是一個變數
- **命名常數（named constant）**：由這種識別子參照的記憶體位置一個不能被改變數值

命名常數範例（補充）

VB.NET	Const WORD1 As Char = “,” c Const MESSAGE As String = “Hello” Const TaxRate As Double = 8.5
C++	const char COMMA = ‘,’ ; const string MESSAGE = “Hello” ; const double TAX_RATE = 8.5 ;
Java	final char COMMA = ‘,’ ; final String MESSAGE = “Hello” ; final double TAX_RATE = 8.5 ;

輸入 / 輸出結構

- 在虛擬碼演算法中，我們使用Read及Write或是Print的表示式來表示我們與程式外部的環境互動方式。
  - Pep-7 機器語言基本指令為 字元輸入 (CHARI) 及字元輸出 (CHARO)，其組合語言增加了 10 進位數輸入 (DECI) 及 10 進位數輸出 (DECO)
- 高階語言將輸入資料視為一個分成許多列的字元串流，輸入敘述包括 3 部份：宣告輸入資料將被放置(即儲存)的變數、輸入敘述、資料串流本身

```
Read name, age, hourlyWage
Maggie 10 12.50
```

資料串流

- 處理的關鍵是資料的型態，它決定了字元要如何被轉換為位元型態 (輸入) 及位元型態要如何被轉換成字元 (輸出)。
- 我們不以例子來說明輸入 / 輸出敘述，是因為它的語法通常是相當複雜，而且在各種高階語言間的差異非常大。

控制結構

- 控制結構 (control structures)
  - 一個能決定在程式中其他指令的執行順序的指令。
  - 組合語言中如 Pep-7 的 BR, BRLT, BREQ 等
- **結構化程式設計**(*structured programming*)
  - 是指在程式中的每一個邏輯單元應該只能有一個入口及一個出口。
  - 這些結構包括有循序、選擇敘述、迴圈敘述及副程式敘述及視窗程式的非同步處理。

控制結構

- 循序
  - 依照順序，一個指令接著一個指令執行
  - 所有的敘述都是依序執行，直到碰到一個可以改變這個順序執行的指令。
- 選擇敘述
  - if敘述有兩種樣式。一種是 if-then，可以決定要執行一群敘述或是跳過它們。第二種是 if-then-else 可以決定要執行兩群敘述中的哪一群。

```
If (temperature > 75)
    Write "No jacket is necessary"
Else
    Write "A light jacket is appropriate"
```


控制結構

- 以下的表是顯示如何以 VB.NET 、C++及 Java 來實現這個演算法。

VB.NET	If (Temperature > 75) Then
	  MsgBox("No jacket is necessary")
	Else
	  MsgBox("A light jacket is appropriate")
	End If

C++	if (temperature > 75)
	  cout << "No jacket is necessary";
	else
	  cout << "A light jacket is appropriate";

Java	if (temperature > 75)
	  System.out.print ("No jacket is necessary");
	else
	  System.out.print ("A light jacket is appropriate");

輸出

輸出

輸出

注意縮排 (也叫 內縮)

控制結構

- 將一串的指令視為一個單一群組的方式叫作**複合敘述** (*compound statement*)。
  - VB.NET 使用保留字 “End If” 來結束段落，所以不需要特別群組方法，就可分辨 出複合敘述
  - C++ 及Java則是將敘述群組以大括號包圍起來，並稱之為一個**區段** (*block*)。

VB.NET	if (Temperature > 75) Then
	  MsgBox("No jacket is necessary")
	Else
	  MsgBox("A light jacket is appropriate")
	  MsgBox("but not necessary")
	End if

C++	if (temperature > 75)
	  cout << "No jacket is necessary" ;
	else
	{
	  cout << "A light jacket is appropriate" ;
	  cout << "but not necessary" ;
	}

Java	if (temperature > 75)
	  System.out.print(No jacket is necessary");
	else
	{
	  System.out.print("A light jacket is appropriate");
	  System.out.print("but not necessary");
	}

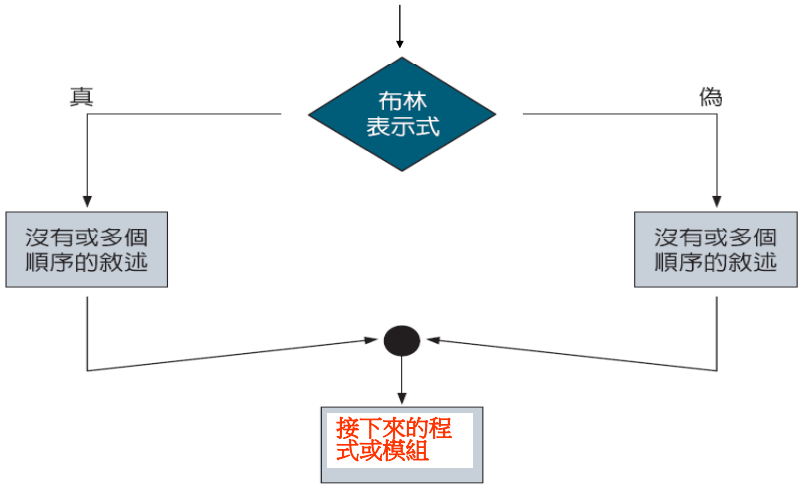


圖 8.3 *if* 敘述的控制流程圖

控制結構

一系列的選擇：

```
If (temperature > 90)
    Write "Texas weather: wear shorts"
else If (temperature > 70)
    Write "Ideal weather: short sleeves are fine"
else If (temperature > 50)
    Write  "A little chilly: wear a light jacket"
else If (temperature > 32)
    Write  "Philadelphia weather: wear a heavy coat"
else
    Write  "Stay inside"
```


重複敘述

- while 敘述
  - 如同 if 敘述，會改變一個程序的正常循序流程
  - if 是在兩個動作中選擇其中之一，而 while 則是用於重複同一動作，有時事先知道次數，有時事先不知次數

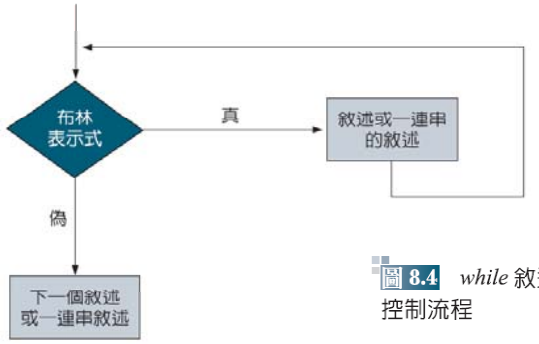


圖 8.4 *while* 敘述的控制流程

重複敘述：計數控制型迴圈

- 一個計數控制型迴圈是一種重複執行一特定數值的次數。
- 在這種迴圈型態由三個不同的部分組成，各使用一個特別的變數，叫作**迴圈控制變數** (*loop control variable*)。
  - 第一個部分叫作***起始化***：讓迴圈控制變數被起始設定成某一特定值。
  - 第二個部分叫作***測試***：檢驗迴圈控制變數是否達到一個預定的數值？
  - 第三個部分叫作***增量***：迴圈控制變數的值被增加1。

計數控制型迴圈

```
Set count to 1                  起始化 count 值為 1
While (count <= limit)          測試
    ...                         迴圈主體
    Set count to count + 1      增量
...                             迴圈之後的敘述
```

- *while*迴圈又被稱為先測型迴圈，這表示它是在迴圈執行前先測試條件是否滿足。
- 一個迴圈如果無法終止那就稱為**無窮迴圈** (*infinite loop*)。

計數控制型迴圈

- 下表顯示以 VB.NET、C++ 及Java程式語言實現這個演算法的程式片段。

VB.NET	Count = 1
	While (count <= limit)
	  …
	  count = count + 1
	End While

C++/Java	count = 1;
	while (count <= limit)
	{
	  …
	  count = count + 1;
	}

事件控制型迴圈

- 若迴圈中重複執行的次數是由發生在迴圈主體內的事件來控制的，則被稱為事件控制型迴圈。

```
Read a value          起始化事件
While (value >= 0)    測試事件
   …                  迴圈主體
   Read a value       更新事件
…                     迴圈之後的敘述
```

重複至輸入值為負範例

```
Set sum to 0                起始化 sum 為 0
Set posCount to 0           起始化事件
While (posCount <= 10)      測試事件
   Read a value
   If (value > 0)           測試看看事件是否要被更新
      Set posCount to posCount + 1   更新事件
      Set sum to sum + value         加上數值到 sum 之中
…                           迴圈之後的敘述
```

加總 10 個正數範例

迴圈

- While 迴圈為*先測型迴圈*
- *後測型迴圈*則是在迴圈內容執行後才進行測試，通常被稱為 repeat 迴圈
- 針對計數控制型迴圈的變型叫做 for 迴圈，將起始化、測試、及增量都包含在迴圈結構中
  - 例如： for (count=1; count <= 10; count++) {
    read value
    set sum to sum +value
    }

副程式敘述

- 當我們在處理演算法時常會在某一層級上給予某一項工作一個名稱，然後在一個較低的層級上才擴展這項工作的內容。
- 將一段的程式碼命名，然後在程式的另一個部分使用這個名稱於敘述上。這在第 6 章的演算法上、下層級間常會用到
- 當程式執行中遇到了這個名稱敘述時，在程式另一個部分的處理程序將暫停等待這個名稱程式碼的執行。當這個名稱的程式碼執行完畢後，緊跟著這個名稱之後的敘述會被繼續執行下去。
- 名稱程式碼出現的地方也叫作**呼叫單元** (*calling unit*) ，又稱呼叫程式或 主程式。

副程式敘述

•有 2 種副程式模式，在不同程式語言有不同名稱
  a) 單純執行一項特定工作，在呼叫單元被當作一個敘述
  b) 執行一項特定工作後，傳回一個值給呼叫單元，在呼叫單元被當作一個函數表示式

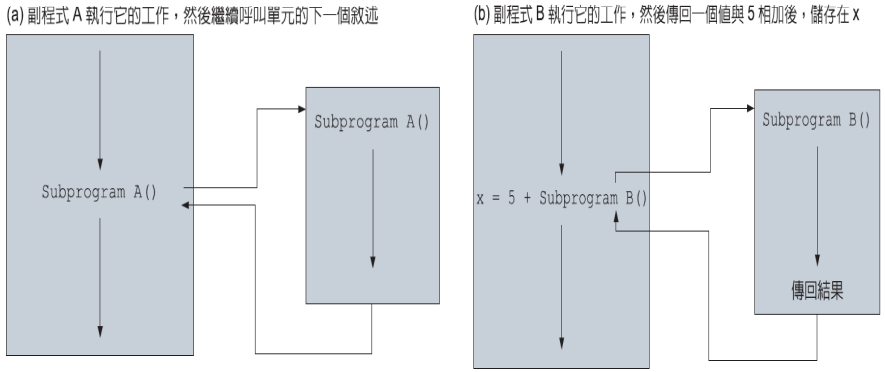


圖 8.5 副程式控制流程

參數傳遞 (下學期)

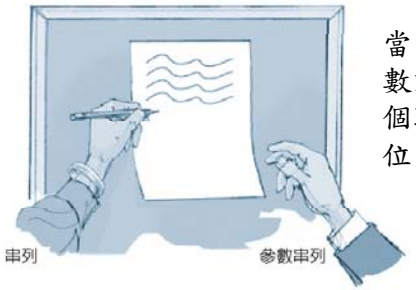
- 有的時候呼叫單元必須送出一些資訊給副程式來使用於它的處理程序中，這種溝通方式稱為參數串列（parameter list），它是一個可以讓副程式動作的識別子串列，每一個識別子通常伴隨著其資料型態，位於副程式名稱旁邊的括弧內
- 參數 (parameters)
  - 在副程式名稱旁邊以括弧圍住指定一個串列的變數名稱及相關的資料型態的識別子，如下例中的 x 與 y

```
int add(int x, int y)
{ return x+y; }
```

- 引數 (arguments)
  - 當副程式需要被呼叫時，呼叫單元列出副程式名稱並跟著在一個括弧內列出識別子串列，這些在呼叫單元的識別子。如下例中的 a 與 2

```
z = b+ add(a,2)
```


傳值及參考參數



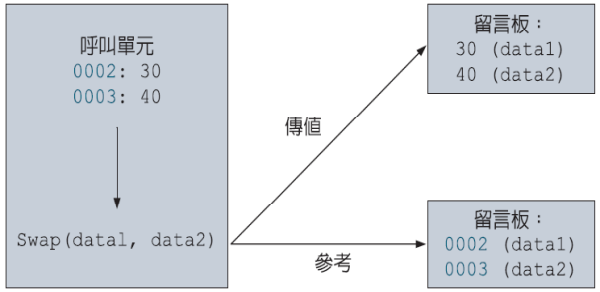
當副程式被呼叫時，呼叫的引數就一個一個取代了參數，這個取代機制有點像一個有固定位置的留言版

圖 8.6 參數傳遞

- 如果一個參數是一個傳值參數 (value parameter)，呼叫單元會將引數複製一份給副程式。
- 如果一個參數是一個參考參數 (reference parameter)，那呼叫單元會將引數的**記憶體位址**傳給副程式。

傳值及參考參數

圖 8.7 傳值參數與參考參數的不同




```
Swap (Integer item1, Integer item2)
Integer temp          宣告區域變數
Set temp to item2
Set item2 to item1
Set item1 to temp
```

```
Swap (data1, data2)
```

Swap 的參數傳遞應該是傳值或是參考參數？