CHAPTER 09 程式語言

9-1 程式語言發展史

- FORTRAN
- LISP
- COBOL
- BASIC
- PASCAL
- C
- PROLOG
- ADA
- C++
- JAVA
- ASP.NET

9-1 程式語言發展史

- 電腦只能接受0與1組成的**機器語言**(machine language)。
- 這些機器語言所代表的意義，通常是做些簡單的加減運算，或是將特定的值指定給**暫存器**(register)。
- **組合語言**(assembly language)把一個以0、1組成的字串用較容易理解的符號表示，譬如相加之指令以機器語言表示為01011010，而在組合語言則以ADD來表示。

9-1 程式語言發展史

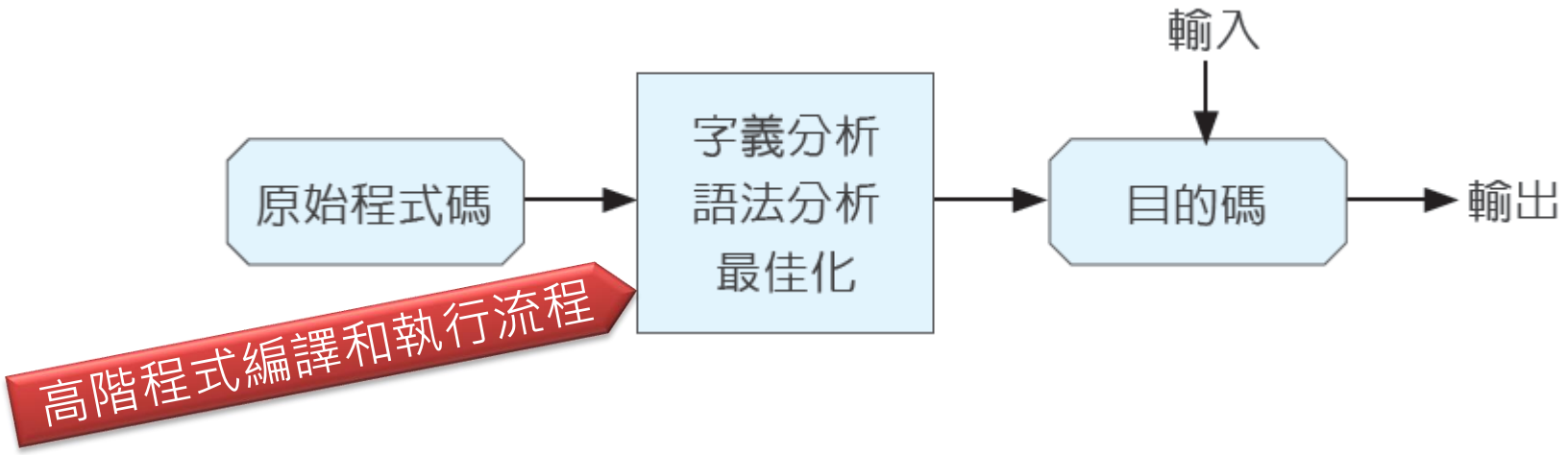
- 組合語言撰寫出來的程式，須透過組合器(assembler)，轉換成機器語言，才能為中央處理器接受。
- 組合語言缺點：
  - 由於組合語言是直接反應機器語言的指令，必須根據每個中央處理器的特性來設計，所以不同規格的電腦就各自有自己的組合語言，如此造成程式設計師學習上的困難，且寫出來的程式也只能在特定電腦上執行。
  - 組合語言只具備有簡單的指令，所以寫出來的程式通常不具結構性，程式冗長且難以閱讀，也就是我們雖然能夠理解各個指令的意義，但是整個程式所欲達到的功能卻不易理解。

9-1 程式語言發展史

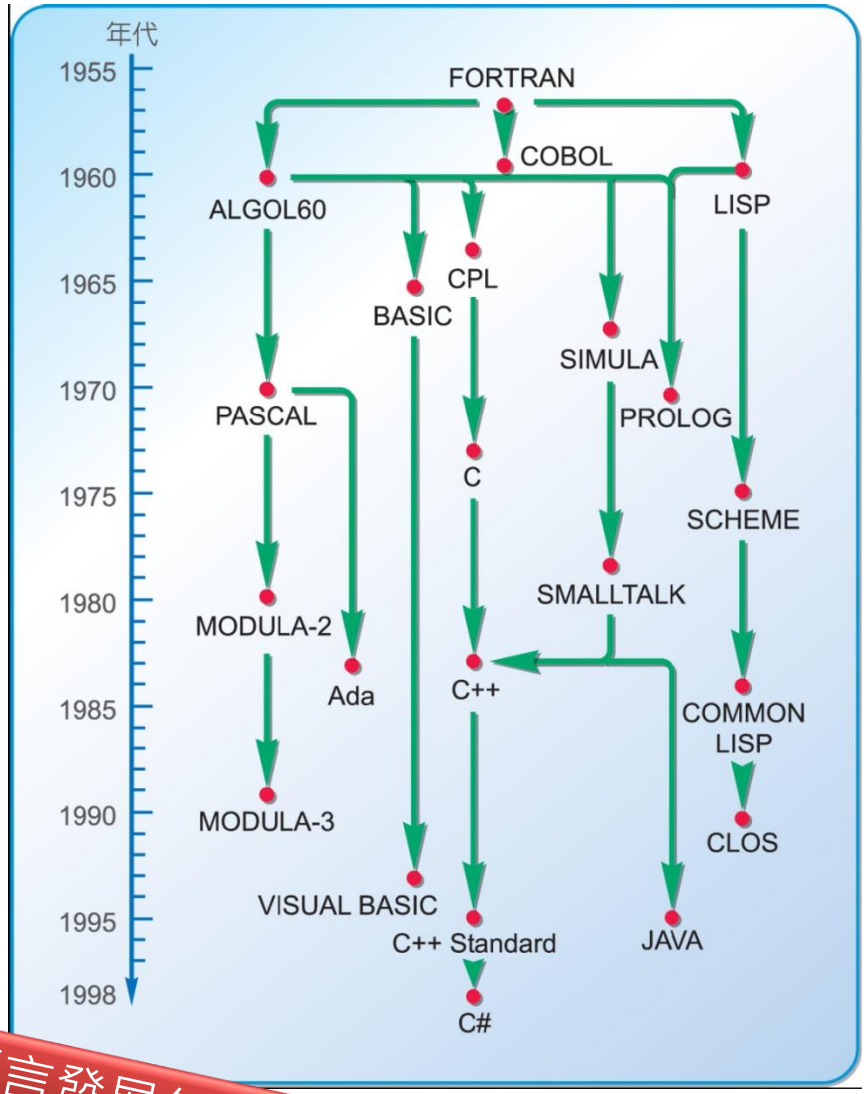
- 組合語言稱作低階語言(low level language)，表示組合語言寫出來的程式可讀性(readability)很低，同時這也是高階語言(high level language)被發展設計出來的原因。
- 高階語言如C語言，寫出來的程式，比起組合語言寫出來的程式，更容易為一般人所理解。
- 高階語言和機器的特性並沒有很密切的對應，所以較具有可攜性(portability)。

9-1 程式語言發展史

- 高階語言寫出來的程式還要經過**編譯**(compile)的步驟才能執行。
- 整個編譯的過程如下圖所示：



該圖的左邊是年代，而箭頭代表不同時間推出的程式語言的前後影響性，可以看到第一個推出的高階程式語言是FORTRAN，而在之後的40幾年仍然有相當多基於不同設計理念的程式語言推出，最近一個最具有影響力的是JAVA。



程式語言發展年表

FORTRAN

- 第一個高階語言是IBM公司於1957年右推出的FORTRAN(FORmula TRANslation language)，中文翻譯成「福傳語言」。
- 該語言當初是針對工程方面所需要的複雜科學計算所設計的，因此其程式敘述類似數學的式子。
- 不少工程數學或數值分析的程式及套裝軟體是利用FORTRAN所書寫的，尤其是需要大量計算的物理、氣象領域。

FORTRAN

- 目前市面上較新的FORTRAN套裝軟體，為Intel推出的Intel Visual Fortran 8.0專業版。

```
      DO 7, LOOP = 1, 5
            READ *, X, Y
            AVG = (X + Y) / 2.0
            PRINT *, X, Y, AVG
7           CONTINUE
      END
```

FORTRAN程式片段：
可以讓使用者輸入5對數字，然後把該數字和平均值印出來，其中第一行的數字“7”對應到第五行的數字“7”，用以表示迴圈的範圍。

LISP

- LISP(LISt Processing)是美國學術重鎮麻省理工學院(MIT)的教授John McCarthy於1958年所推出的。
- LISP並不強調數值運算的效率，反而提供很具彈性的符號表示與運算表示式，所以適合做**符號運算**(symbolic computation)，因此在人工智慧的應用上特別重要。

LISP

- COMMON LISP是目前最通用的版本，之後也擴充了CLOS(Common Lisp Object System)，提供物件導向的程式結構。

```
(defun length (x)
       (cond ((null x) 0)
                (t    (+ 1 (length (cdr x))))))
(length  '(I love computers))
       3
```

LISP程式片段：
前3行首先定義一個函數叫作“length”，該函數計算一個串列(list)內包含幾個元素。接著在第4行呼叫該函數，並且輸入串列「(I love computers)」，則會回傳“3”。

COBOL

- COBOL(Common Business Oriented Language)是專為商業資料處理而設計的語言，當時是由美國國防部推動成立的資料系統語言組織 CODASYL(COnference of DAta SYstem Language)編定，而於1959年發表。
- COBOL提供便利的檔案描述與處理，整個程式的結構，也特別重視資料的定義，適於描述不同類型的商業資料。目前仍然有一些早期開發的商業系統，繼續使用COBOL，特別是銀行界。

```
01   EMPLOYEE-RECORD
     05  EMPLOYEE-NUMBER      PIC 9(5)
     05  EMPLOYEE-NAME             PIC X(30)
     05  BIRTH-DATE
         10  BIRTH-MONTH           PIC 99
         10  FILLER                PIC X
         10  BIRTH-DAY             PIC 99
     05  DATE-HIRED
         10  MONTH-HIRED           PIC 99
         10  FILLER                PIC X
         10  DAY-HIRED             PIC 99
```

COBOL範例：
以階層式的方式定義員工的相關資料。其中“EMPLOYEE”稱作集體項，包含階層號碼和資料名稱；其餘的為基本項，除了階層號碼和資料名稱，還包含資料格式定義，譬如“X”符號代表文數字資料型態。至於FILLER主要是用來填補不用或不會參考到的位置，在程式中不會用到。

BASIC

- 1965 年 推 出 BASIC(Beginner's All purpose Symbolic Instruction Code)。
- 早期個人電腦還在使用DOS作業系統的時候，裡面就附有QBASIC的開發環境，所以當時很多人第一個接觸的程式語言就是BASIC。
- 微軟以該語言為基礎，於1992年推出VISUAL BASIC(簡稱VB)，為BASIC語言提供了視覺化的簡易開發環境。

BASIC

- 目前以BASIC為主的商用程式語言版本只剩VB。
- BASIC的好處是簡單易學，缺點則是不夠嚴謹。

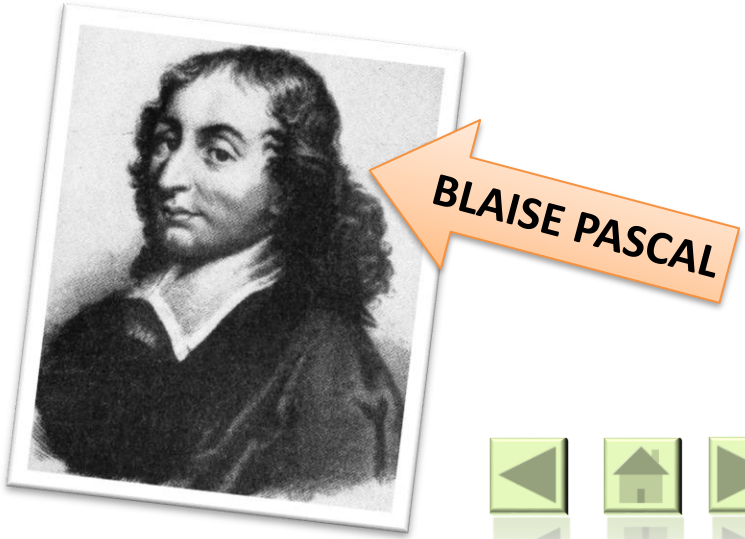
```
Dim i, sum

sum = 0
For i = 1 To 10
        sum = sum + i
Next i
```

BASIC範例：
計算從1加到10的和，其中“Dim”表示後面要宣告變數，但是並不需要明確指出變數“i”和“sum”的資料型態。

PASCAL

- 1971年推出的PASCAL，該語言的名稱是紀念17世紀重要的法國數學家BLAISE PASCAL。
- PASCAL具有完備的資料型態，和結構化的控制結構，所以語言更有效率也更易於使用。
- 由於其程式可讀性高，所以常為教科書教導初學者所用。



PASCAL

- 目前較為人知的是物件化的PASCAL語言，由Borland公司的Delphi產品所支援。

```
function      gcd (m, n : integer) : integer;
      var
            remainder : integer;
      begin
            while n <> 0 do
            begin
                  remainder := m mod n;
                  m := n;
                  n := remainder;
            end;
            gcd := m
      end;
```

PASCAL範例：
定義了一個PASCAL的函數叫作“gcd”，該函數會根據兩個參數“m”和“n”計算他們的最大公因數，然後將該值回傳給呼叫此函數的式子。

C

- 美國AT&T貝爾實驗室，為了設計UNIX系統，而於1972年研發出C語言。
- C語言和PASCAL類似，同樣具有高階的結構化敘述，但是為了因應作業系統控制硬體的需求，也具備了類似低階語言的控制硬體能力。
- 由於其強大的功能，成為目前最常見且為大多數人使用的高階語言。

PROLOG

- PROLOG(PROgramming LOGic)是邏輯化程式設計(logic programming)的代表。
- 1972年於法國所推出，當時的目的是為了自然語言處理的需求所發展出來。
- 常用於設計邏輯推論、專家系統等。
- 和LISP同樣是人工智慧領域重要的程式設計工具。

PROLOG

- 利用下列的範例，解釋邏輯化程式設計的概念。

```
Facts
        mother (mary, tom).
        father (john, tom).
Rules
        parent (X, Y) :- mother (X, Y).
        parent (X, Y) :- father (X, Y).
Queries
        ?- parent (mary, tom).
                yes
        ?- parent (john, X).
                X = tom
```

PROLOG範例：
首先，我們先給定兩個事實(facts)，說明“tom”的父親和母親是誰。接著，我們再定義只要是父親(father)或母親(mother)，就是父母(parent)。然後我們可以利用這些事實和法則來詢問系統。第一個問題是想要確認“mary”是不是“tom”的父母，答案是肯定的；第二個問題則詢問“john”是誰的父母，而得到的回覆是“tom”。

ADA

- ADA是由美國國防部於1980年代主導所設計出來的程式語言，此語言的名稱是紀念世界上第一位程式設計員Ada Byron。
- 當初此語言的目的是希望結合所有語言的特性，成為一個具有最強大功能的程式語言，但是也由於其語言過於複雜，造成推廣上的困難，目前所知的應用不多。

C++

- 第一個具有代表性的物件導向程式語言(object - oriented programming language) 其實是1980年左右推出的SMAL LTALK，其語言的特性強調物件的設計、訊息(message)的傳送，比傳統的結構化語言更具模組化的觀念，所以也更易於維護。
- C++則是將物件導向的概念融入C語言而成，換句話說，C語言可以看作是C++的子集合。

C++

- 由於C語言的廣被使用，C++也成為最重要的物件導向程式語言，甚至比C語言更受歡迎。

```
class stack {
        private:
                int top;
                char components[50];

        public:
                stack( )        {top = 0};
                char pop( )     {
                        top = top - 1;
                        return components[top+1];
                }
                void push (char c)   {
                        top = top + 1;
                        components[top] = c;
                }
};
```

我們定義了一個類別（class）叫作“stack”。在類別中，我們除了可以定義資料（data member）外，還可以定義此類別的行為（function member）。以類別“stack”為例，變數“top”和“components”是用以記載此“stack”的相關資料，而函數“stack”、“pop”、“push”則會根據定義好的程式碼執行特定動作。這種把資料和行為一起定義的特性，稱作「封裝」（encapsulation）。

在類別中比較特殊的是，可以指定某個資料或函數的可使用範圍（accessibility）。若是定義為公開的（public），則類別外部的程式碼可使用該資料或函數，如“stack”、“pop”和“push”若是定義為私自的（private），則只有定義在類別內部的程式碼可使用該資料或函數，如“top”和“components”。如此控管對資料的安全性和完整性更有保障。

JAVA

- JAVA是近幾年來最重要的程式語言之一，它是美國Sun公司於1995年正式發表，與C++一樣具備有物件導向的特性。
- 比C++更容易學習，更重要的是提供了跨平台的功能，也就是一個相同的程式可以在不同的環境下執行，所以在網路上的應用具有相當大的前瞻性。

JAVA

```
public class stack
{
        private int top;
        private char[] components = new char[50];
        public stack() { top = 0;}
        public char pop( ) {
                top = top - 1;
                return components[top+1];
        }
        public void push(char c) {
                top = top + 1;
                components[top] = c;
        }
}
```

JAVA範例：
定義了一個類別叫作“stack”。

ASP.NET

- 隨著全球資訊網的盛行，不論是個人或公司，都紛紛將眾多資料以網頁的方式放在網站上供人瀏覽。但是，HTML基本上只能將固定的資料做適當的排版和呈現，而無法即時地從資料庫中抓取資料來動態地形成網頁。
- 為了達到此需求，許多程式語言被提出來，其中微軟所提供的ASP語言（Active Server Page），由於簡單易學，受到相當多人的歡迎。但是，其程式混雜HTML語法和Script語言，並不容易維護與除錯。

ASP.NET

- ASP.NET大幅度地改善了ASP的缺點，除了將程式分成HTML和Script不同的區塊，便於撰寫和除錯，也具有物件導向語言的特性。
- 為了提高撰寫程式的彈性，針對Script的部分，ASP.NET還支援多種不同的程式語言，特別值得一提的是微軟於1998年新設計的C#語言。該語言是基於C語言所發展出來的，所以很受到一般受過C程式語言訓練的工程師的歡迎。

程式語言依照特性分類

種類	程式語言	特性
命令式(Imperative)	FORTRAN、COBOL、BASIC、PASCAL、C 、ADA	程式由一連串有順序性的指令組成
物件導向式(Object-Oriented)	C++、JAVA、ASP.NET	具有封裝特性的物件為程式的核心
函數式(Functional)	LISP	程式視為由運算式組成的函數
邏輯式(Logical)	PROLOG	提供邏輯判斷的寫法

9-2 資料型態

- 陣列
- 結構
- 指標

9-2 資料型態

- 當我們要利用某個程式語言撰寫一個應用系統的時候，我們必須要將處理的對象，以該程式語言提供的資料型態，適當的定義在程式中。
- 譬如說，要表示月和日組合起來的日期，如2月1日，可以使用字串表示成「0201」，或是利用整數「32」，來表示是1年的第32天，有的語言甚至直接提供日期型態。

9-2 資料型態

- 一般來講，高階程式語言都會提供以數字和字串為基礎的資料型態。
- 數字而言，多分為整數(int)、長整數(long int)、浮點數(float)、雙精準數(double)等，這些型態的差別在於可表示數值資料的大小範圍。
- 文字方面，有的只能定義一個字元(char)，有的則直接可定義較長的字串(string)。

9-2 資料型態

- 當我們為一個變數宣告好其資料型態之後，系統就知道應該為該變數保留多少記憶體的空間，而空間的大小會決定該型態可表示的數值範圍。
- 下表顯示C所支援的資料型態，所需的空間和資料範圍會因為機器的規格而有所不同，此表是以64位元的電腦為例，C語言的long int至少是32bits，也可能是64bits。

C 的資料型態

資料型態	所需空間	資料範圍
char	8 bits	ASCII
int	32 bits	-2147483648 ~ 2147483647
short int	16 bits	-32768 ~ 32767
long int	32 bits	-2147483648 ~ 2147483647
float	32 bits	3.4E-38 ~ 3.4E+38
double	64 bits	1.7E-308 ~ 1.7E+308

9-2 資料型態

- 為一個變數宣告好資料型態後，編譯器就會檢查該變數在程式任何地方出現的時候，是不是使用恰當。假設我們宣告「x」是一個字元的資料型態，將符號「a」指定給x就是恰當的，但是將x乘以100就是沒有意義的。
- 基於這些好處，很多高階語言如PASCAL和C語言，都要求在使用一個變數前，必須先宣告它的資料型態。

陣列

- 當有一系列相同型態的資料想要處理，如全班50個同學的數學成績，就可以使用陣列(array)的資料型態。
- 以下宣告一個包含50個整數的陣列：

```
int score[50];
```

陣列

- 陣列的名稱為「score」，陣列裡的每個資料為整數(int)型態，而陣列第一個位置為score[0]，第二個位置為score[1]，依序一直到score[49]，這是因為C語言預設以註標0來表示陣列的第一個元素。
- 定義了陣列之後，就很容易從這個序列中取出一個特定的資料。

陣列

- 假設這個陣列是以學生的學號依序建立的，那當我們要取出學號5的同學的成績，我們就可以寫score[4]，而學號20的同學的成績，則可以利用score[19]取出。

結構

- 當有一些相關資料，想要聚集成一個單元一起處理，可以使用結構(structure)的資料型態。譬如說，針對一個同學，我們想要表示他的姓名、系別、年級等3種資料，可以宣告如下：

```
struct student {
        char(6) name;
        char(10) major;
        int year;
};
```

結構

- 結構的名稱為student，其中欄位name的資料型態為6個字元(char)，欄位major的資料型態為10個字元，欄位year的資料型態為整數。
- 假設我們之後再宣告變數x的資料型態為student結構，如下所示：

```
struct student x;
```

結構

- 則以後我們可以利用小數點加上欄位名稱，來指出變數x其中的某一個成分，如x.name，x.major，和x.year。
- 這種表示式可以代表該成分在記憶體的位置，也可回傳該成分目前的值。

指標

- 指標(pointer)是一種很特殊的資料型態，它記錄的是某個資料在記憶體的位置，也就是它提供了非直接存取(indirect accessing)的功能。
- 那麼為什麼我們不直接處理該資料，而要透過指標呢？通常有以下兩個理由：
  - 為了效率性的考量。
  - 我們不能確定資料的大小。

指標

為了效率性的考量

- 指標記錄一個記憶體的位置，所以其所需的空間是固定的，通常就是一個字元的大小。
- 假設每一個顧客資料，都是用複雜的結構表示，而每個結構大小為100位元，若是希望對所有的顧客資料做處理，像是依照購買金額排序，則在記憶體內我們必須搬動很多個100位元大小的顧客結構。
- 另一方面，若使用指標為代理人，則在記憶體內我們只須搬動1個字元大小的指標，則程式執行的效率會有顯著的改善。

指標

我們不能確定資料的大小

- 假設要記錄所有顧客的資料，其中一個方法是使用陣列，但是宣告陣列時必須很明確的告知陣列內元素的個數，如50或100，以便系統在記憶體裡預留空間。
- 假設宣告陣列大小為100，但是只來了10個顧客，則有90個元素的空間被浪費了；但是若宣告為50，但是卻來了60個顧客，則事先預留的空間則不夠，造成很大的問題。

指標

- 一般的作法，是將每筆資料用一個節點(node)表示，然後利用指標將節點串連起來，稱作鏈結串列(linked list)。
- 假設現在要處理的資料是整數型態，則節點的定義如下所示：

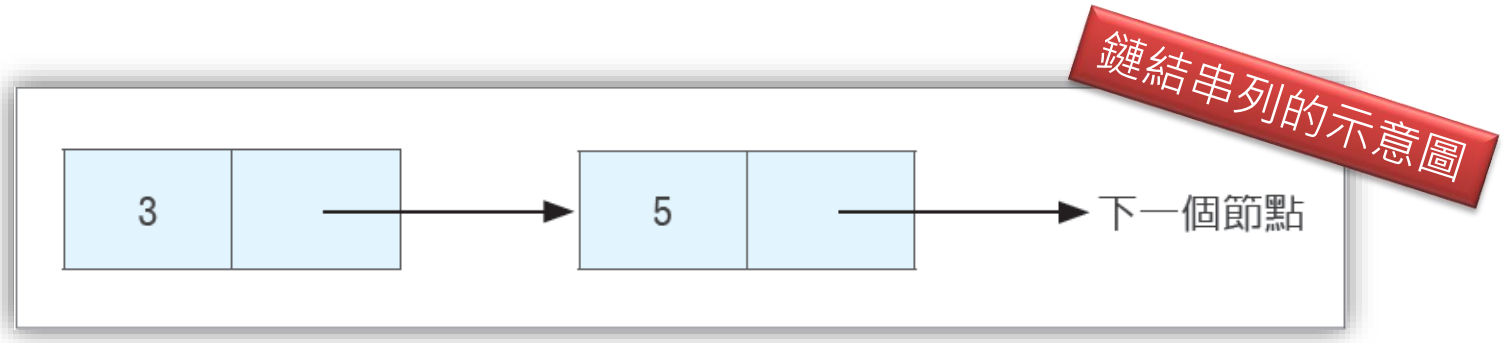
```
struct node
{
        int data;
        struct node *next;
};
```

指標

- 符號「*」表示後面接的變數字串記錄了位址，也就是說，next代表了記憶體中的一塊空間，而該空間存放的資料型態是node。
- 第一個節點裡的資料是整數3，它指到下一個節點，其資料是整數5，依此類推。

指標

- 如果要再新增資料，只需要建立一個新的節點，然後接到這個鏈結串列即可。
- 若是原先的資料不需要了，也可以將該節點移除，然後把指標重新指定，並不需要做太大的改變。



9-3 程式指令

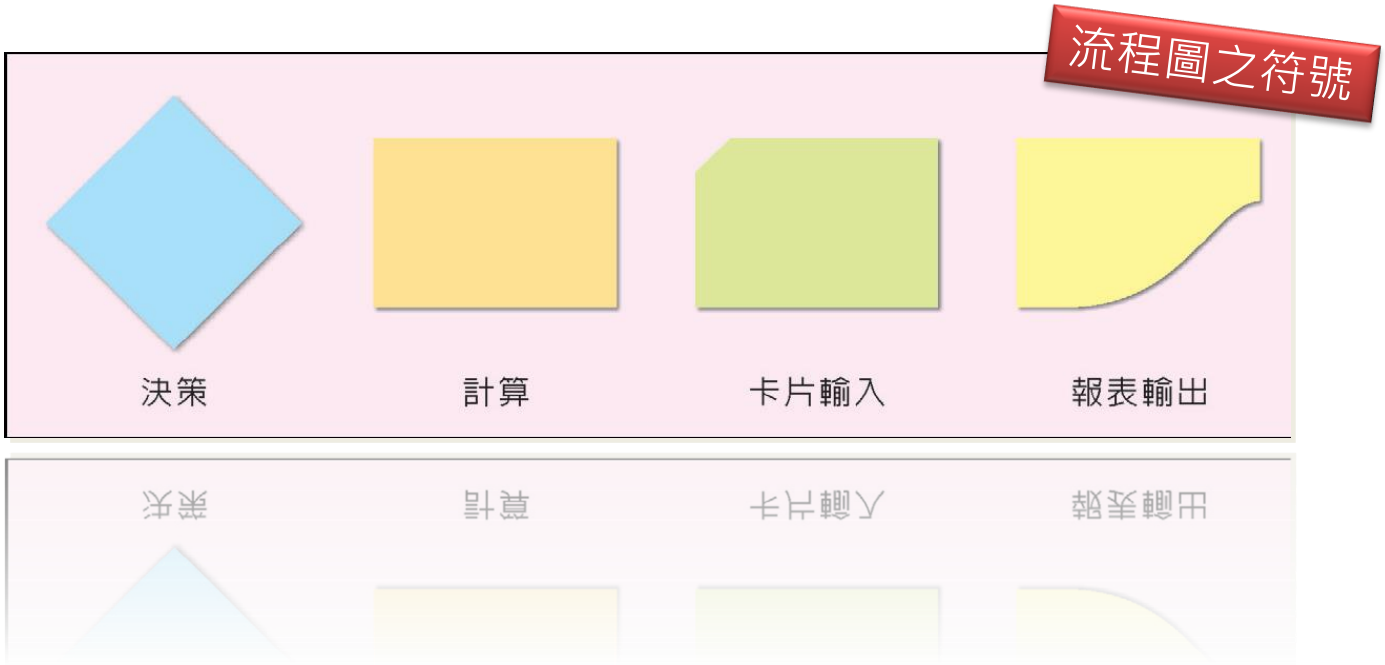
- 比較：if
- 固定次數的迴圈：for
- 不固定次數的迴圈：while和repeat
- 不固定次數的迴圈：for

9-3 程式指令

- 為了清楚的表示邏輯結構和步驟間的關聯，我們常常會使用流程圖(flow chart)來輔助說明。
- 流程圖裡有幾個不同的符號，分別有其意義：
  - 決策(decision)的運算式是用菱形框表示。
  - 計算(computation)的敘述式是用長方框表示。
  - 輸入(input)和輸出(output)有時會以特定機件(device)有關的形狀來表示。

9-3 程式指令

- 相關的符號如下圖所示：



比較：if

- if指令提供了邏輯判斷式。
- 如果if後面接的運算式被判斷為真，則程式會繼續執行then後面的運算式。
- 如果if後面接的運算式被判斷為不真，且程式設計師提供了其他運算式在else之後，則程式會改而執行該運算式，否則就不會有任何動作。

比較：if

- 下面這個範例，在變數i的值大於0時，變數x的值設定為10，否則變數y的值設定為5。

```
PASCAL                          C
      if  (i > 0)  then         if (i > 0)
            x := 10                     x = 10;
      else                      else
            y := 5                      y = 5;
```

比較：if

- PASCAL和C的寫法
  - PASCAL使用較多的關鍵字，在這裡我們看到PASCAL使用了then這個關鍵字，但是在C裡面將它省略掉。
  - 在PASCAL程式裡，只要可以清楚地分辨出每個敘述(譬如利用關鍵字)，就不用在最後加上分號;，但是在C裡面，敘述必須以分號作為結尾。

比較：if

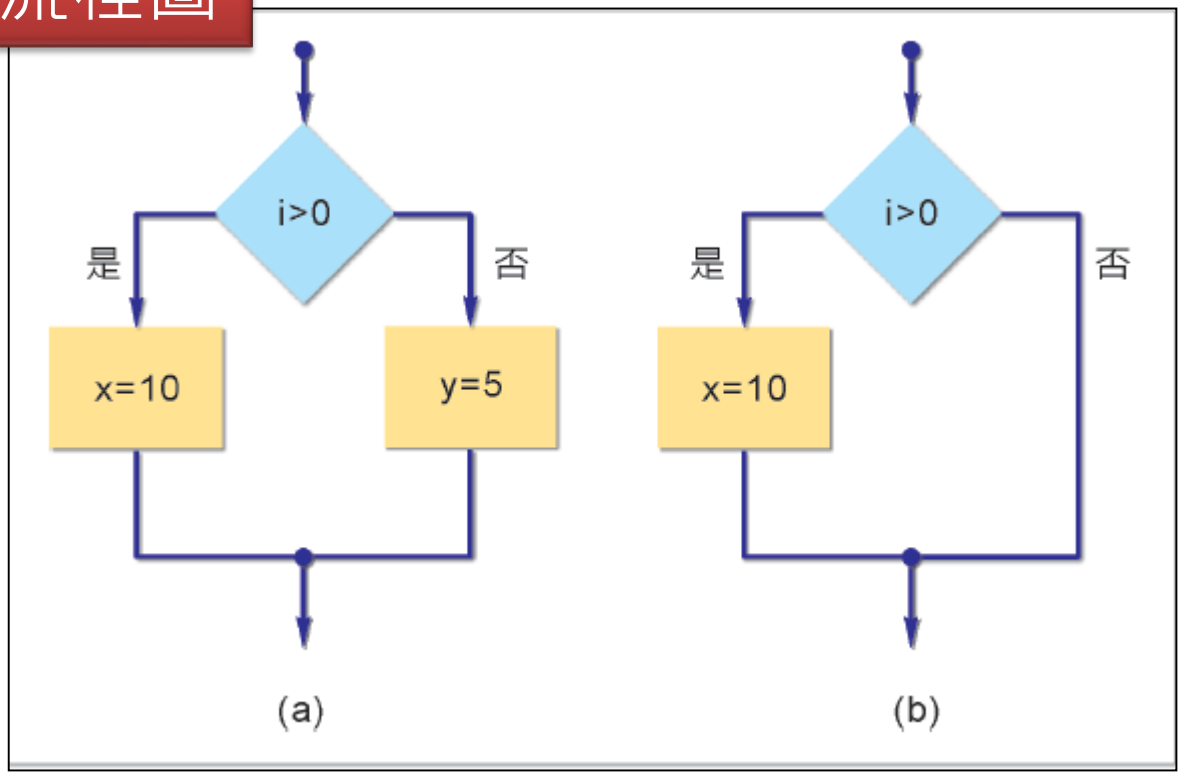
- 下面這個C語言的範例， 與上例的差別， 是在於變數i的值小於或等於0時，並不會再進一步執行任何命令，因為我們並沒有提供else子句。

```
C
        if (i > 0)
                x = 10;
```

比較：if

- 寫在if之後的邏輯判斷式，會表示在菱形符號中，然後利用標示為「是」和「否」兩條線，分別指到不同的運算。
- 為了清楚的表示整個結構，分別利用兩個小圓圈，作為一個虛擬的開始和虛擬的結束。
- 在圖(a)中，判斷式「i >0」不論是否符合，都會有一個對應的運算；但是在圖(b)中，一旦判斷式不符合，則沒有任何的運算，整個結構直接結束，進入下一個命令。

if結構的流程圖



比較：if

- 下例顯示了巢狀if(nested if)的寫法，也就是我們可以在then或else的部分，再放入另一個if敘述。
- 以此例而言，當變數i的值被判斷為正之後，我們需要再確定變數a的值大於變數b的值，才會指定變數x為10。

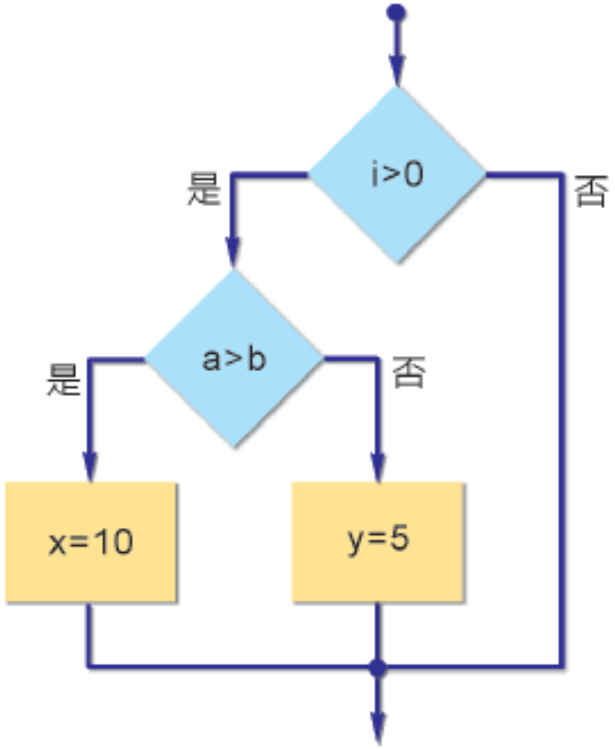
比較：if

- 值得注意的是，變數y的值會被指定為「5」，是在當變數i的值為「正」，且變數a的值「不大於」變數b的值的情況下。

```
C
        if (i > 0)
                if (a > b)
                        x = 10;
                else
                        y = 5;
```

比較：if

- 在這裡可以清楚地看出來，一旦判斷式「i ＞ 0」不符合，則整個結構沒有任何其他運算，直接結束；但是若判斷式「i ＞ 0」為真，則還要再做另一個判斷，亦即是否「a ＞ b」，才會決定相對應的動作。



巢狀if結構的流程圖

固定次數的迴圈：for

- 利用for指令，我們可以事先指定好迴圈的執行次數。
- 下面這個PASCAL範例，透過變數i的值將迴圈的執行次數控制為5次，同時變數x的值在迴圈結束後，會等於整數1加到整數5的和。

```
PASCAL
        x := 0;
        for i := 1 to 5 do
                x := x + i;
```

不固定次數的迴圈：while和repeat

- 所謂的不固定次數，就是迴圈的執行次數，並沒有很明確的在程式裡指定好，至於迴圈要執行幾次，則是利用一個特定的邏輯判斷式。
- 在PASCAL的語法中，while後面是接一個邏輯判斷式，也就是i<6，若是這個邏輯判斷式為真，則程式會進入此迴圈， 執行do後面的指令，在此例中是更改變數x和變數i的值。

不固定次數的迴圈：while和repeat

- 等到這兩個指令執行完後，程式會回到邏輯判斷式，再一次判斷變數i的值是否小於6，如此不斷重複，直到變數i的值大於6或等於6的時候，才會跳出迴圈。
- 由於一開始設定變數i的值為1，且每跑一次就把變數i的值加1，所以此迴圈總共會執行5次；同時，變數x的值，會是整數1加到整數5的和。

不固定次數的迴圈：while和repeat

```
PASCAL                          C
       i := 1; x := 0;          i = 1; x = 0;
       while (i < 6) Do         while ( i < 6)
       begin                    {
              x := x + i;               x = x + i;
              i := i + 1;               i = i + 1;
       end;                     }
```

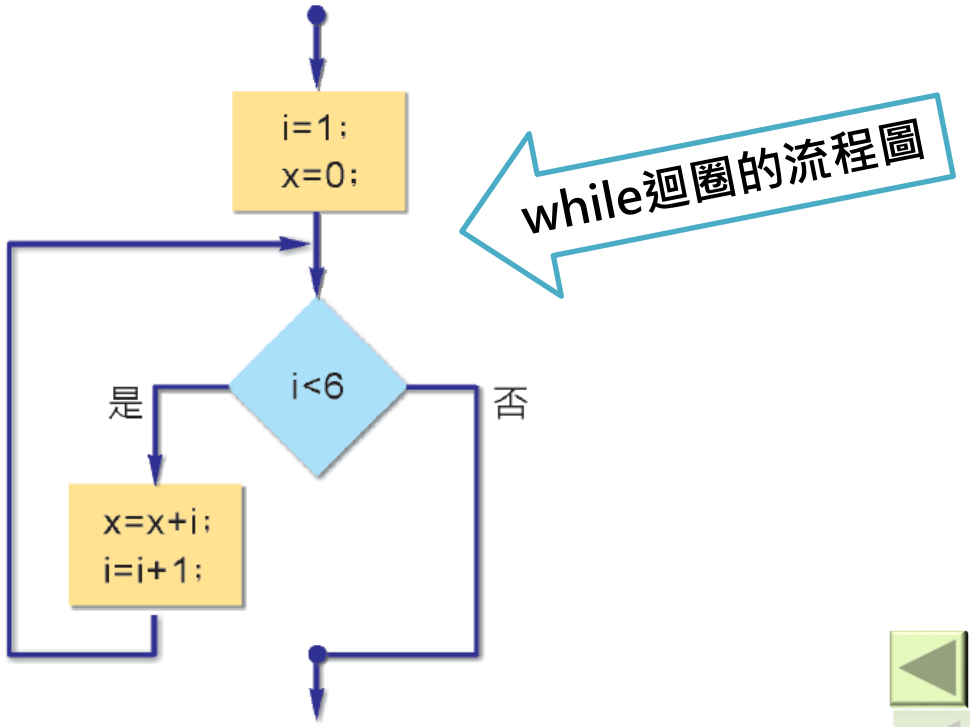
- 進入迴圈之後，要先後執行兩個命令，來依序更改變數x和變數i的值，所以這兩個命令被關鍵字begin和end包起來，被包起來的命令稱作複合命令(compound statement)，它可以被視作是一個擁有很多「小」指令的一個「大」指令。

不固定次數的迴圈：while和repeat

- 若是在迴圈內只要執行一個命令的話，則不需要關鍵字begin和end。
- 比較PASCAL和C的寫法不同，可以看到在PASCAL裡是使用到關鍵字do，但是在C裡面被省略掉；而PASCAL裡的關鍵字begin和end，在C裡面則被左大括弧「{」和右大括弧「}」所取代。

不固定次數的迴圈：while和repeat

- 為了清楚地表示此迴圈代表的邏輯結構和執行順序，我們也將對應的流程圖表示在下圖中。



不固定次數的迴圈：while和repeat

- 首先，先指定好變數“i”和變數“x”的值。接著，我們進入邏輯判斷式，若是判斷式不成立，則程式會直接跳出此結構；若是判斷式成立，則會再回到之前邏輯判斷式的位置，根據最新的變數值再重複進行判斷。
- 若是沒有適當的改變變數值，使得邏輯判斷式的真假值改變，則會再度進入迴圈，甚至造成無窮迴圈的情況，這是撰寫程式時需要注意的地方。

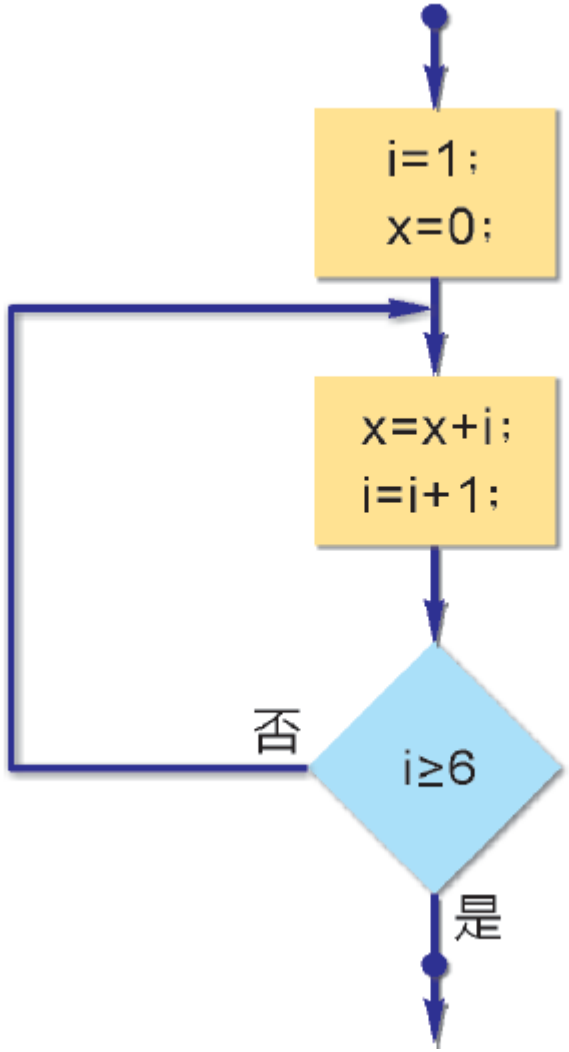
不固定次數的迴圈：while和repeat

- 另一種迴圈的寫法，則是不先做判斷，而是直接先執行命令，等到執行完再做邏輯式的判斷。
- 在PASCAL裡，定義的語法是利用關鍵字repeat和until，在C裡，則是利用關鍵字do和while。
- 不過，雖然這兩種寫法，都是先執行命令，再進行邏輯式的判斷，但是，當判斷式為真的時候，do-while 的寫法會繼續留在迴圈裡，而repeatuntil 的寫法則會離開迴圈。

不固定次數的迴圈：while和repeat

- 所以在下例中，同樣是執行迴圈5次，左右兩邊的邏輯判斷式正好相反。

```
PASCAL                          C
      i := 1; x := 0;           i = 1; x = 0
      repeat                    do {
            x := x + i;               x = x + i;
            i := i + 1                i = i + 1;
      until i >= 6;             } while ( i < 6);
```



repeat迴圈的流程圖

不固定次數的迴圈：for

- for指令後面接著的式子分三部分：
  - 第一是在執行迴圈之前，所需要先給定的初始值設定。
  - 第二是進入或留在迴圈的條件，有如while指令後面接著的判斷式。
  - 第三是在每當要執行下一次迴圈之前，所需要執行的式子。

不固定次數的迴圈：for

- 下面列出對應於之前while寫法的for的寫法：

```
while                                   for
        i = 1; x = 0;                   x = 0;
        while ( i < 6)                  for (i=1; i<6; i=i+1)
        {                               {
                x = x + i;                      x = x + i;
                i = i + 1;              }
        }
```

不固定次數的迴圈：for

- 由於控制迴圈執行次數的是變數i，所以可以將該變數的初始值、留在迴圈的條件、和每次迴圈更改的方式，都直接列在for指令的後面，如此可以更清楚分辨出迴圈內執行的內容，和迴圈執行的次數。

9-4 程序定義和使用

- 全域變數vs.局部變數
- 以值傳遞vs.以位址傳遞

9-4 程序定義和使用

- 在一個程式(program)中，可能會寫出冗長而難以理解的命令，所以大部分的程式語言都提供了程序(procedure)或函數(function)的定義。
- 一個程序對應到一段程式碼，稱作程序本體(body)，然後也指定一個對應的名稱，稱作程序名稱(name)。
- 等到定義完程序之後，只要利用該名稱呼叫該程序(procedure call)，對應的程式碼就會執行。

9-4 程序定義和使用

- 程序在定義時，必須提供下列資訊：
  - 程序名稱
  - 程序本體，含變數宣告和命令敘述
  - 正式參數(formal parameter)宣告
  - 程序回傳的資料型態

9-4 程序定義和使用

- 在下例中，定義一個程序叫作square，該程序定義了一個整數參數x，還有一個局部變數y，參數x的平方值會被計算出來然後回傳給呼叫者。

```
int square (int x)
{
        int y;

        y = x * x;
        return (y);
}
```

9-4 程序定義和使用

- 在下例中，將定義一個沒有回傳值的程序。在第9-2節中，曾經定義了結構node，用以建構出一個鏈結串列。我們把該結構再一次列在下面：

```
struct node
{
        int data;
        struct node *next;
};
```

9-4 程序定義和使用

- 假設有兩個鏈結串列p和q，希望將p串列的第一個node，變成q串列的第一個node，則對應的程式定義如下：

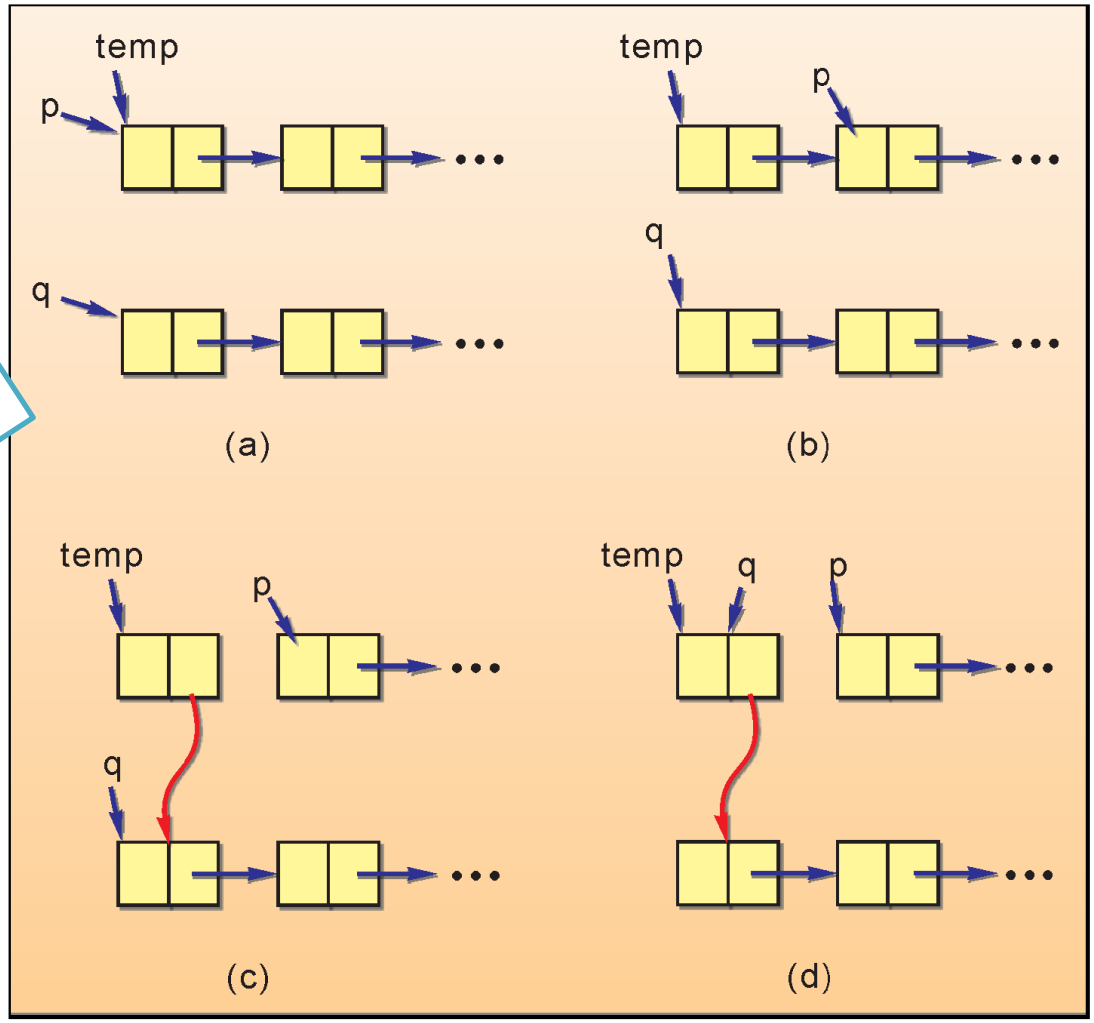
```
void changehead (struct node *p, struct node *q)
{
        struct node *temp;

        temp = p;
        p = p ->next;
        temp->next = q;
        q = temp;
}
```

9-4 程序定義和使用

程序名稱	• changehead
正式參數	• 兩個資料型態為指到結構node的指標參數，分別叫作p和q
局部變數	• 一個資料型態為指到結構node的指標變數，叫作temp
程序本體	• 將p串列的第一個節點移除，然後加入到 q串列的第一個節點前
回傳值	• 並無回傳值，在C語言裡是以void表示之

程序 changehead的執行步驟示意圖



9-4 程序定義和使用

- 程序square和程序changehead的最大差別，在於前者有回傳值，而後者沒有。
- 在 PASCAL 裡，有回傳值的程序稱作**函數**(function)，為了方便起見，我們也一律稱有回傳值的C程序為函數。
- 值得注意的是，通常我們是在一個運算式裡呼叫一個函數。

9-4 程序定義和使用

- 譬如在下面的程式碼中，先呼叫函數square以計算5的平方，然後將函數回傳的值乘以10之後，再將其值指定給變數 x。

```
x = square(5) * 10;
```

9-4 程序定義和使用

- 至於一般沒有回傳值的程序，就如同一般命令的被呼叫，如同下例所示。

```
p->data = 3;
q->data = 5;
changehead(p,q);
```

全域變數VS.局部變數

- 在撰寫一個程式時，我們必須定義變數用來記錄不同的資料。但是根據變數可被使用的範圍，我們可以將變數分為兩類：
  - 全域變數(global variable)：能被全部的程式碼使用到。
  - 局部變數(local variable)：只能被一部分程式碼使用到，通常定義在程序中。

全域變數VS.局部變數

- 以下面這個C程式的範例來說明：

```
int a;
void proc(int b)
{
        a = 3;
        b = 5;
}
main( )
{
        int c;

        a = 7;
        c = 9;
        proc(11);
}
```

全域變數VS.局部變數

- 在C程式裡，定義在每個程序裡的變數，稱作局部變數(local variable)，只有該程序可以使用該變數。
- 譬如，變數c為程序main的局部變數，若是程序proc使用了變數c，則為不合法的使用。
- 至於定義在整個程式碼的最前端，就沒有隸屬於哪一個程序，所以任何程序都可以使用它，這樣的變數稱作全域變數(global variabe)。

全域變數VS.局部變數

- 在本範例中，變數a即為全域變數，所以程序main和程序proc都可以使用它。
- 首先程序main先將它的值定義為7，接著呼叫程序proc，將其值重新定義為3，所以最後變數a的值會是3。

以值傳遞 VS. 以位址傳遞

- 定義程序時，必須定義**正式參數** (formal parameter)，同時宣告該參數的資料型態。
- 定義完之後，我們在呼叫該程序時，所提供的符合正式參數資料型態的參數，就稱作**真實參數** (actual parameter)。

以值傳遞 VS. 以位址傳遞

- 該函數定義了一個正式參數x，其型態為整數，如下所列：

```
int square (int x)
{
        int y;

        y = x * x;
        return (y);
}
```

以值傳遞 VS. 以位址傳遞

- 在下列的運算式裡呼叫該函數時，所提供的真實參數為5：

```
z = square(5) * 10;
```

- 在這裡的問題，就是我們如何把真實參數5，傳給正式參數x，以便進行運算？

以值傳遞 VS. 以位址傳遞

- 在C程式裡的作法，就是「以值傳遞」(passed by value)。
- 我們會把真實參數的「值」算出來，然後再傳給正式參數。所以，我們也可以提供一個運算式，作為真實參數。

以值傳遞 VS. 以位址傳遞

- 在下例中，我們會先算出5+3的值之後，再將其傳給正式參數x：

```
z = square(5+3) * 10;
```

- 以值傳遞是一個最方便也最常見的方式，但是它仍然有它的限制，就是沒有辦法改變真實參數的值。

以值傳遞 VS. 以位址傳遞

- 假設我們希望寫一個程序，把兩個整數值對調，我們寫出來的程序可能如下所示：

```
void donothing(int x, int y)
{
        int temp;

        temp = x;
        x = y;
        y = temp;
}
```

以值傳遞 VS. 以位址傳遞

- 然後我們在主程式裡，呼叫程序donothing幫我們交換變數a和b的值，如下所示：

```
main ( )
{
        int a, b;

        a = 3;
        b = 5;
        donothing(a, b);
}
```

以值傳遞 VS. 以位址傳遞

- 則執行的狀況如下：

```
1. x = 3
2. y = 5
3. temp = 3
4. x = 5
5. y = 3
```

以值傳遞 VS. 以位址傳遞

- 在程序裡面，參數x和y的值的確被調換了，但是對真實參數a和b卻產生不了任何影響。
- 正確的寫法，應該是利用「以位址傳遞」(passed by reference)的觀念，也就是把真實參數在記憶體的位址傳給正式參數，讓程序裡的運算直接作用在真實參數上。

以值傳遞 VS. 以位址傳遞

- 下面列出C語言的寫法：

```
void swap (int *x, int *y)
{
        int temp;

        temp = *x;
        *x = *y;
        *y = temp;
}
```

以值傳遞 VS. 以位址傳遞

- 在呼叫的時候，則必須明確地把位址傳過去：

```
main ( )
{
        int a, b;

        a = 3;
        b = 5;
        swap(&a, &b);
}
```

以值傳遞 VS. 以位址傳遞

- 則執行的狀況如下：

```
1. x = &a
2. y = &b
3. temp = *x(a) = 3
4. *x(a) = *y(b) = 5
5. *y(b) = temp = 3
```

以值傳遞 VS. 以位址傳遞

- 注意到在第4步裡，雖然在程序裡表面上是作用在正式參數x，但因為正式參數x和真實參數a，其實是指到在記憶體裡的同一塊空間，所以等於是作用在真實參數a上面。
- 第5步也是同樣的效果。如此一來，就達到了改變真實參數的目的。