

目錄

| | |
|-------------------------------------|-----------|
| Chapter 01. | 1 |
| 1-1 程式設計與電腦解題的關係..... | 1 |
| 1-2 程式設計的用途與重要性..... | 1 |
| 1-3 程式設計在各領域的應用..... | 2 |
| 1-4 資料結構及演算法與程式設計的關係..... | 2 |
| 1-5 程式設計步驟..... | 5 |
| 1-6 程式設計工具..... | 6 |
| Chapter 02. | 7 |
| 2-1 變數與常數..... | 7 |
| 2-2 基本輸入與輸出..... | 10 |
| 2-3 運算式與敘述..... | 13 |
| Chapter 03. | 16 |
| 3-1 選擇性結構..... | 16 |
| 3-2 重複敘述功能..... | 21 |
| Chapter 04. | 27 |
| Chapter 05. | 30 |
| 5-1 模組(Module)和副程式(subprogram)..... | 30 |
| 5-2 程式庫..... | 32 |
| Chapter 06. | 36 |
| 6-1 陣列(Array)..... | 36 |
| 6-2 資料錄..... | 40 |
| 6-3 指標..... | 44 |
| Chapter 07. | 49 |
| 7-1 佇列(Queue)..... | 49 |
| 7-2 堆疊(Stack)..... | 54 |

| | | |
|--------------------|----------------------------------|-----------|
| 7-3 | 串列(List)與鏈結串列(Linked list) | 58 |
| Chapter 08. | | 64 |
| 8-1 | 排序演算法(Sorting Algorithm) | 64 |
| 8-2 | 搜索演算法(Searching Algorithm) | 70 |

Chapter 01

概論

何謂 C 語言? C 語言是電腦的高階程式語言，其語法具有簡潔的特性，除此之外，優秀的執行效率，以及跨平台的特性，都是 C 語言成為熱門程式語言的優點之一。從最上層的軟體設計到底層的硬體控制，都可以看到 C 語言的影子，也就是說，C 語言兼具了軟硬體的應用，範圍廣泛。源起：1972 年，發展於貝爾實驗室，以 B 語言為基礎，開發出 C 語言。

特色：

| 可攜性 | 具高低階語言能力 | 控制結構 | 撰寫格式 |
|---------------------------------------|--|----------------|---------------|
| - 高可攜性 -使用 c 語言撰寫的程式，很容易可以移植至不同的平台 | -C 語言具可讀性且具結構化(高階語言特性) -程式的效率高且對硬體存取能力佳(低階語言特性) | 程式的控制執行上具高度靈活性 | C 語言的格式相當靈活自由 |

我們使用 C 語言用來做程式設計，目的是要利用電腦來替人類解決問題進而找出答案。當然問題根據其應用可能十分複雜，因此本書將會介紹如何利用程式的撰寫，來符合使用者的需求。而學習程式語言除了要了解基本概要之外，必須要依照一些設計步驟，加以分析與設計，讓我們能夠應付所有複雜的問題。

1-1 程式設計與電腦解題的關係

程式設計是針對電腦給出解決特定問題程式的過程，是軟體構造活動中的重要組成部分。解決問題是一個過程，這個過程會對問題進行分析，找出解決的方案。

由於資訊科技的進步，電腦的運算速度日漸增快，複雜的運算對於時間的需求愈來愈小；記憶體容量也隨著硬體技術的進步愈來愈大，處理大量的資料變的更加容易；在相同的條件下，反覆執行，對於電腦來說，不會有如人為一般出錯的可能，大大提高運算的正確性，並且可以重複不停的反覆作業，加上網路的蓬勃發展，隨時能夠與不同所在的電腦進行溝通。綜合以上的優點，電腦在日常生活中成為我們用來解決問題的重要工具。








1-2 程式設計的用途與重要性

早期電腦發展硬體資源昂貴，程式執行時間與空間代價往往是設計的主要考量，隨著硬體的技術的發展，軟體規模日益增大，硬體軟體之間的設計，與以往也大大的不同。故在各種約束條件和軟硬體相互矛盾的需求之間尋求一種平衡。程式設計整合硬體、軟體、網路及資訊整合等 4 個層面發展，並且考量到程式的結構、可維護性、復用性、可延伸性等等因素。透過程式設計，讓未來的世界更多采多姿。

現今的硬體運算速度越來越快、體積越來越小、功能越來越強大，讓軟體得以應用於大量資料處理之外，未來隨著人工智慧技術，將使電腦越來越「智慧」。另外值得一提的是，網路發展傳輸速度越來越快之下，使得無線傳輸技術應用更為廣泛，整合各個領域的科技，機器人、虛擬實境...，都已經是我們可以看到的實體應用，而這些都透過程式語言的設計來完成。

1-3 程式設計在各領域的應用

近年來軟體發展日益增大，發展及應用的領域更顯寬廣，如同上述小節所提及一般，以下列舉幾個相關的領域應用，將程式設計運用在不同的領域上，能發揮令人驚豔的結果。舉凡：

-  科學研究：太空探測、行星軌道計算、...
-  軍事發展：彈道計算、飛彈導航、...
-  醫學實驗：基因研究、遠距醫療、...
-  氣象預測：颱風預測、地震預測、...
-  商業用途：金融稅務、投資分析、...
-  影音休閒：電腦遊戲、電腦繪圖、...
-  生活應用：網路訂票、網路購物、資料搜尋、...等等，不盡其數。

1-4 資料結構及演算法與程式設計的關係

曾經有一位瑞士的計算科學家- Niklaus Emil Wirth 說過：Algorithms + Data Structures = Programs (演算法+資料結構=程式) 是計算機科學的名句。故要提升程式設計的能力，除了會用程式語言之外，資料結構及演算法更是必修重點。

資料結構跟演算法皆可以獨立開成一門課，在此我們只提簡單的概要，建議有興趣的同學可以再另習其他教材。

演算法

演算法是有限數目的一群指令，依序執行後可完成某一特定的工作，用文字或虛擬指令(虛擬碼)，將解決問題的先後順序和步驟表達出來。任何機械性或者重覆性的計算程序稱之為演算法(又稱法則或計算方法)。

演算法不僅可用來解決數學或科學上的問題，也可以應用於日常生活中，來解決特定問題或完成特定工作。

演算法的表示法

- ✎ 敘述式表示法(虛擬碼):利用語言敘述的方式來表達演算法的處理步驟。
- ✎ 流程圖表示法:是透過簡明且標準的圖示符號來表達問題解決步驟的示意圖,更清楚地呈現解決問題的步驟。

演算法必須滿足下列五個條件

- ✎ 輸入(input):可以有零個或更多的輸入資料
- ✎ 輸出(output):至少會產生一個輸出。
- ✎ 明確性(definiteness):每個指令必須是簡潔明確的,而且不是含糊的。
- ✎ 有限性(finiteness):在有限步驟後必須結束,不能產生無窮迴路。
- ✎ 有效性(effectiveness):可僅藉由紙、筆計算,即可求出答案。

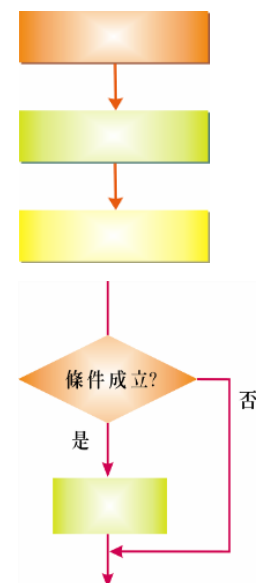
結構化程式設計的特徵

結構化的程式設計要使用三種基本控制結構:循序、選擇、重覆等等而每個結構都具有單入口以及單出口的特性。利用三種結構,將程式設計模組化,意思為將整個程式區分為許多單元(模組),每個單元(模組)完成某個定義明確的工作,整體結構採用由上而下的設計。先定義最高階的模組,再往下定義低層的模組。

結構化流程控制結構

基本控制結構有循序結構、選擇結構、重覆結構,在此將作概念性的介紹,在後面的章節將有更詳細的說明以及操作方式。

- ✎ 循序結構
逐步的撰寫敘述,由上而下,依序逐一執行。
- ✎ 選擇結構
依某些條件做邏輯判斷。
 - if
 - if...else
 - switch...case...
- ✎ 重覆結構
依某些條件決定是否重複執行某些敘述。
 - for
 - while
 - do...while



流程圖的優點

上圖用來表示結構的圖稱為流程圖，利用標準圖形及流程圖用紙，並可藉由流程圖規增加美觀。在撰寫程式，開發軟體的時候，用流程圖來表示軟體個規劃，可使工作流程更易掌握，除錯(debug)較為容易，將一作以標準化的符號表達，有助於工作人員的彼此溝通協調，加速工作的進行。故預先繪製流程圖，可使程式的編寫更為容易。

1-5 程式設計步驟

從面對問題，到程式設計完成，通常程式設計過程會經過下列六個階段

(1.) 分析問題——需求認識

分析問題的階段包括了探討以電腦解決問題的可行性、找出輸入輸出的資料項目等。

(2.) 找出演算法——設計規劃

所謂演算法(Algorithm)，就是解決問題的處理步驟。以電腦解決一個問題，可能有許多種不同的方法。每一種方法都是一個演算法。此階段的工作，就是要想出一種較好的解決問題方法。所謂好的方法，主要是指電腦執行此演算法的速度較快者。



(3.) 繪製流程圖或列出演算法步驟

| 類別 | 簡介 | 範例 |
|------|--------------------------|--|
| 低階語言 | 不須翻譯機器即能執行, 但程式的可讀性較低 | 機器語言, 組合語言 |
| 高階語言 | 使用人較能接受的文字及符號, 因此程式的可讀性高 | FORTRAN, COBOL, BASIC, PASCAL, LISP, Visual Basic, NET, C, C++, JAVA |

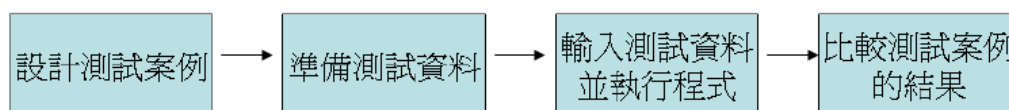
依所採用的演算法，繪製程式的流程圖(Flow Chart)。流程圖勢將處理問題的步驟，或一連串工作程序，用標準化的圖形和線條表現出來，也可以使用文字敘述的方式列出演算法步驟，來表達程式設計的思考邏輯，或是程式的流程。

(4.) 撰寫程式

根據流程圖或演算法步驟撰寫程式。

(5.) 測試程式

最後必需確認程式的輸出是否符合需求，程式撰寫後，必須反覆以多組輸入資料測試，以去除語法錯誤(Syntax Error)和邏輯錯誤(Logic Error)。



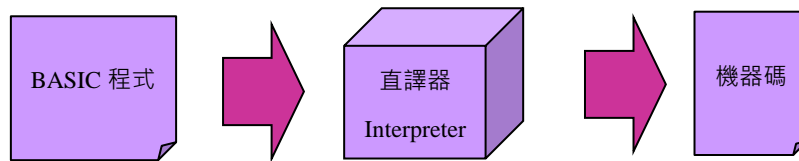
(6.) 編寫文件

一般在程式中加註解(Remark)，作為程式說明即可。但大型的系統程式應該有詳細的說明文件，包括系統說明、操作手冊等，以利於操作使用與日後維護。

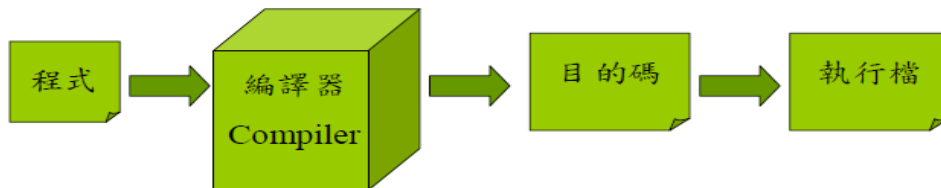
1-6 程式設計工具

電腦只能執行機器語言指令，所以使用的程式語言，必須經過轉換程式之後，電腦才能執行。而此轉換程式可概分為兩種：直譯程式(interpreter)與編譯程式(compiler)。

✎ **直譯程式：**直譯器是將程式語言所撰寫的原始程式碼(source code)逐行翻譯成機器語言指令，但使用此方式執行程式時都必須重新翻譯，程式才能執行，因此執行速度較慢。常見的直譯式程式語言有 BASIC。如下圖所示。



✎ **編譯程式：**編譯是將程式語言所撰寫的整個程式翻譯成機器語言指令，但不會立即執行的一種程式翻譯方式。當原始程式每修改一次，就必須重新編譯，才能保持執行檔為最新狀況。經過編譯後所產生的執行檔，在執行時不須再翻譯，因此執行速率遠高於直譯程式。常見的編譯式程式語言有 C、Cobol、Pascal 等。



Chapter 02

基礎觀念

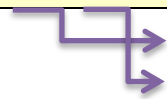
我們可以說寫程式的目的是為了完成一件特定的工作，因此一個基本的程式執行流程為「輸入資料」、「資料運算」、「輸出結果」。不過在輸入資料之前，我們必須先告訴電腦有哪些資料將被輸入，以及如何將這些資料儲存起來，這樣才能對於這些資料進行運算。舉例來說，假設我們要寫一個計算學期成績的程式，那麼我們會輸入的資料可能包含文字資料（例如：學生姓名和科目名稱）與數字資料（例如：小考成績、期中考成績和期末考成績）。這些資訊都要在資料輸入之前讓電腦先知道，而要讓電腦知道如何儲存輸入的資料則是透過變數的宣告來完成。

2-1 變數與常數

變數(Variable)

在程式中，若要使用變數，必須事先宣告，這樣電腦就會在記憶體裡配置一塊足以容納此變數大小的記憶體空間，之後不管變數的值如何改變，他永遠占用相同的記憶體空間。一個變數的宣告，是在變數前給定一個指定的資料型態，型態視需求而定，這個概念就好像我們在運送貨物時，如果是耐撞的貨物，我們可以用一般的卡車運送，如果是運送需要冷藏的食材，就必須要選擇配有低溫裝置的貨車運送。所以在宣告變數的時候，記得仔細想想要用這個變數儲存甚麼類型的資料。變數的宣告是撰寫程式的基礎，在後面的章節中我們將會不斷看到有關於變數的宣告，以下則是一個簡單的變數宣告範例：

```
int num;
```



整數資料型態
變數名稱為 num

在這個例子中，num 就是我們宣告變數的名稱，它的資料型態是 int。資料型態是 int 代表程式在執行的過程中，儲存在 num 這個變數的內容將會是整數，像是 -7, 0, 1, 5566, … 等等。此外，在為變數命名時，可以使用英文字母、數字和底線（_）的組合（例如：student1、name、year_2000），但是不能用數字開頭，也不可以和 C 語言已經使用的關鍵字作為命名（如 if, while, for, … 等等，有關 C 語言已經使用的關鍵字我們在後面的章節將會一一介紹）。C 語言中對於變數的名稱會將英文大小寫視為不同的字母。也就是說，num1 和 NUM1 是兩個合法且相異的變數名稱。在命名變數

的時候，最好採取有意義的命名方式，以免程式愈大，變數的名稱也會愈多，屆時需要做維護的時候造成不易分辨的困擾。

程式中也允許同時宣告多個變數，只要在各變數之間用逗點(，)隔開。例如以下的例子一次宣告了三個整數變數 num1、num2 和 num3。

```
int num1, num2, num3;
```

定義變數的時候，也可以同時指定該變數的初始值。

```
int num1 = 200;
```

常數(Constant)

常數(Constant)的值是固定的，不會因為程式的執行而改變，也就是說 CPU 記錄數值資料的地方，CPU 在整個程式執行過程中，只會去讀取而不會對此進行修改。當我們在做命名的時候，最好採取有意義的命名方式，以免程式愈大，相對上常數的名稱也會愈多，屆時需要做維護的時候造成不易分辨的困擾其功能。

```
const int maxCount = 100 ;
```

maxCount 被定義成是一個不可改變的整數變數，程式裡不可以去修改這個變數的內容，例如(maxCount = 24 ;)，因此，在整個程式中，maxCount 就是整數 100 的代表。

基本資料型態

變數的基本型態有整數(int)、浮點數(float)、字元(char)…等，在宣告變數的時候，必須給定資料型態，讓編譯程式(compiler)能配置適當的記憶體空間給該變數。其中，表示方法中的型態修飾詞 unsigned 表示此數沒有負號，就是只有正數。如下表所示：

| 型別 | 符號 位元 | 位元 長度 | 表示 方法 | 數值 範圍 |
|----|----------|----------|----------------|---|
| 整數 | 有 | 32 | int | -2147483648 ~ 2147483647 |
| | | 16 | short int | -32768 ~ 32767 |
| | | 32 | long int | -2147483648 ~ 2147483647 |
| | | 64 | long long | -9223372036854775808 ~ 9223372036854775807 |
| | 無 | 32 | unsigned int | 0 ~ 4294967295 |
| | | 16 | unsigned short | 0 ~ 65535 |

| | | | | |
|-----|---|----|--------------------|---------------------------------------|
| | | 32 | unsigned long | 0 ~ 4294967295 |
| | | 64 | unsigned long long | 0 ~ 18446744073709551615 |
| 浮點數 | 有 | 32 | float | 10 ⁻³⁸ ~10 ³⁸ |
| | | 64 | double | 10 ⁻³⁰⁸ ~10 ³⁰⁸ |
| 字元 | 有 | 8 | char | -128 ~ 127 |

常用跳脫字元大多由反斜線(\)所組成，通常加在字串內一起使用，下表示幾個常用的控制字元。如下表所示：

| 跳脫字元 | 所代表的意義 | 跳脫字元 | 所代表的意義 |
|------|-----------------|------|----------------|
| \a | 警告音(Alert) | \t | 跳格(Tab) |
| \b | 倒退一格(Backspace) | \\ | 斜線 |
| \n | 換行 | \\ | 反斜線 |
| \r | 歸位 | \' | 單引號 |
| \0 | 字串結束字元 | \" | 雙引號 |
| \d | ASCII 碼(8 進位) | \x | ASCII 碼(16 進位) |

資料型態轉換

在我們寫程式的過程中，雖然都會進行變數的宣告在進行變數的運用，但其實有些時候，在撰寫程式的過程中，我們會遇到需要做型態轉換的時候。換句話說，當我們需要某個變數的資料，但是變數之間的型態不一樣的時候，就需要透過型態的強制轉型語法來完成。

欲轉換變數的型態，以下提供 C 語言的強制轉型語法

(欲轉換的資料型態) 變數名稱

《範例》

```
int x ;
float y = 13.5;
x = (int) y;
printf("%d", x);
```

《執行結果》

13

我們可以看到原本 x 變數的型態是 int 整數型態，y 變數是 float 浮點數型態並且含有值 13.5，當 y 的值要傳給 x 的時候，透過第三行的強制轉型語法，我們可以將 y 的值傳遞給 x 變數。

但這裡必須要注意的是，根據上例，作強制轉型的時候可能會遺失某些值的準確度，13.5 經過強制轉型之後會變成 13，做無條件捨去。

2-2 基本輸入與輸出

懂了變數的宣告之後，接著我們來看如何來做輸入與輸出，而學習一個新的程式語言，這往往是最重要的開始，能夠讓使用者輸入資料進程式，能夠將值輸出於螢幕中。在下面兩個小節中，我們將分別介紹 printf 與 scanf 函數。

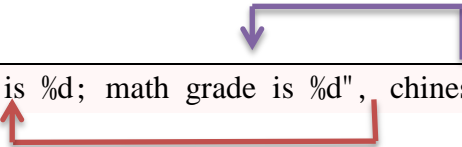
基本輸出敘述

輸出函數為 printf，將程式的執行結果或者是內容輸出至螢幕，其格式字串必須用雙引號包圍；項目 1、2 可以是常數、變數、或者運算式。

```
printf("格式字串", 項目 1, 項目 2)
```

《範例》

```
printf("My Chinese grade is %d; math grade is %d", chinese, math);
```



此範例會將把 chinese 的值填入前方%d，將 math 的值填入後方的%d。就上述範例中，%d 視為列印格式，為印出整數變數的內容。除了%d 之外還包括其他不同的特定字元格式，用於不同的資料型態列印輸出使用。下表列出 printf() 函數常用的格式碼：

| 種類 | 表示法 | 功能敘述 |
|-----------------|----------|--|
| 整數 (integer) | d | 以十進位方式印出。 |
| | o | 以八進位方式印出。 |
| | x | 以十六進位方式印出。 |
| | u | 以不帶符號的十進位方式印出。 |
| | l (小寫 L) | 以長整數(long)方式印出。 |
| 浮點數(float) | f | 以 xxx.xxxxxxx 方式印出。 |
| | e | 以指數的方式印出。 |
| 字元(char) | c | 以字元方式印出。 |
| | s | 以字串方式印出。 |
| 其他(other) | - | 向左邊靠齊印出。例如：% -3d |
| | + | 將正負號顯示出來。例如：% +5d |
| | *, * | 指定浮點數之精確度。 %6.3f：印出浮點數時，包含小數點共有 6 個位數，小數點前佔有 2 個長度，小數點後 |

| | | |
|--|--|--------------|
| | | 只要顯示 3 位數即可。 |
|--|--|--------------|

除此之外，在做列印輸出的同時，我們也可以指定要輸出的寬度(width)，可以看成要印出幾個欄位。如%3d，表示成 3 個欄位。

| 資料內容 | 格式 | 執行結果 | | | | | | | | | |
|---------|---------|------|---|---|----|----|----|---|---|---|-----|
| 12345 | %10d | | | | | | | 1 | 2 | 3 | 4 5 |
| 12345 | %+d | + | 1 | 2 | 3 | 4 | 5 | | | | |
| 12345 | %-10d | 1 | 2 | 3 | 4 | 5 | | | | | |
| 12345 | %□d | | 1 | 2 | 3 | 4 | 5 | | | | |
| 12345 | %010d | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 5 |
| 123.456 | %7.2f | | 1 | 2 | 3. | 4 | 6 | | | | |
| 123.456 | %010.3f | 0 | 0 | 0 | 1 | 2 | 3. | 4 | 5 | 6 | |
| 123.456 | %+10.4f | | + | 1 | 2 | 3. | 4 | 5 | 6 | 0 | |

基本輸入敘述

透過鍵盤輸入，讓我們可以輸入資料給程式。scanf 函數可以讓我們做到這件事，將鍵盤所輸入的資料存於變數之後，告訴編譯程式來指定記憶體位址(address)。在 stdio.h 標頭檔中定義的輸入函數，函數的格式如下：

```
scanf("格式字串", &變數 1, &變數 2, ...)
```

資料輸入後，再放到接收資料的變數位址中，記得要在變數前上加一個位址運算子&(例如:&變數)。

《範例》

```
#include <stdio.h>
int main()
{
    char name[10];
    printf("what's your name :");
    scanf("%s", name);
    printf("Hi, %s, How are you ? \n", name);
    return 0;
}
```

《執行結果》

```
what's your name : Eric
Hi, Eric, How are you ?
```

設定字串給變數之前，必須要先宣告字元陣列，給字串變數，char 字串變數[字串長度]，在使用 scanf 時指定輸入格式為%s。可以看到當我們使用 scanf 的時候，我們會將使用者所輸入的名字(Eric)存到變數 name 中，當使用 printf 輸出函式的時候，就可以將 name 裡的值印出來。

字元的輸入與輸出

getchar 函數可以讓我們從鍵盤上輸入一個字元，你所輸入的字元會立即顯示出來。並且當你按下 Enter 鍵後，這個字元才就會被變數所接收。

putchar 函數會把字元變數、常數等當成參數，傳遞到函數後再列印出來。

《範例》

```
#include <stdio.h>
int main()
{
    char a;
    printf("請輸入一個字元：");
    a = getchar();
    putchar(a);
    return 0 ;
}
```

《執行結果》

```
請輸入一個字元：t
t
請輸入一個字元：test
t
```

當我們輸入值的時候，會將輸入值儲存至變數 a 中，呼叫 putchar 函數的時候，就會將第一個字元輸出至螢幕。如果輸入的值超過一個字元，也只會輸出第一個字元，如上述兩個例子所示。

2-3 運算式與敘述

大部分的敘述都是由運算式(Expression) 所構成，而運算式則是由一組一組的運算子(Operator) 與運算元(Operand) 所組成而成。如：

```
a = a + 10
```

其中 a 和 10 都是運算元，而=和+則為運算子，其中=等號我們可以看是「設定」的意思，亦即將 a+10 的值設定回給 a 變數中。另外，在運算式之後加上分號，就成為敘述(Statement)，如：

```
a = a + 10;
```

接著我們將介紹幾種不一樣的運算子：一元運算子、算數運算子、關係運算子、遞增與遞減運算子、邏輯運算子。

一元運算子

一元運算子指運作在一個運算元上。好比「+」單一運算員表示的就是鄭號的意思。

算數運算子

算數運算子代表的是數學運算，好比「+」、「-」、「*」、「/」、「%」，分別作加法、減法、乘法、除法、取餘數的運算。

關係運算子

關係運算子用來做運算元之間的比較，比較的結果不是 true 就是 false。其中等於運算子(==)，常常被誤寫成指定運算子(=)，這兩個運算子的意義是截然不同的，也是初學者常常犯的錯誤，要格外小心。

遞增與遞減運算子

運算乃將運算式的值+1 或者-1。

邏輯運算子

通常利用邏輯運算子將兩個真假值加以處理。AND(且)為兩者皆是，OR(或)為兩者取其一，其結果回傳 true 或 false。

| | 運算子 | 意義 |
|---------------------------|-----|---------|
| 一元運算子 (unary Operator) | + | 正號 |
| | - | 負號 |
| | ! | Not，否 |
| | ~ | 取 1 的補數 |
| 算數運算子 | + | 加法 |

| | | |
|-------------------------|----|-----------------------------|
| (Mathematical Operator) | - | 減法 |
| | * | 乘法 |
| | / | 除法 |
| | % | 取餘數 (適用於整數) 浮點數取餘數(fmod) |
| 關係運算子 | > | 大於 |
| | < | 小於 |
| | >= | 大於等於 |
| | <= | 小於等於 |
| | == | 等於 |
| | != | 不等於 |
| 遞增與遞減運算子 | ++ | 遞增，變數值+1(加 2 不能使用) |
| | -- | 遞減，變數值-1(減 2 不能使用) |
| 邏輯運算子 | && | AND，且 |
| | | OR，或 |

運算子至此，以介紹了八九成，現在我們將按照他們運算的優先順序加以歸類，好比數學運算中，加減乘除是有優先順序的，而在程式中也不例外。原則是：【遞增與遞減運算子】>【算術運算子】>【關係運算子】>【邏輯運算子】。

在 C 語言中，運算優先順序與其結合性，在同一格內的優先順序一樣，但最上面的優先權高於下面的所有運算順序。好比順序 1：（）、[]、->，優先順序皆高於下面九種，屬於最高優先權。

| 優先順序 | 運算子 |
|------|---------------------|
| 1 | ()、[]、-> |
| 2 | !、+(正號)、-(負號)、++、-- |
| 3 | *、/、% |
| 4 | +、- |
| 5 | <<、>> |
| 6 | >、>=、<、<= |
| 7 | ==、!= |
| 8 | && |
| 9 | |
| 10 | = |

在第二章的尾聲，我們將告訴讀者一些寫秘訣，有很多時候我們常常用到一些運算式，其實是有較為簡潔的寫法，而這些簡潔的運算式也會隨著寫程式的經驗與能力的提升，自然而然的記憶在我們的腦海中，就讓我們來看看有那些簡潔的運算式：

| 運算子 | 範例 | 意義 |
|------------|---------------|-----------------------------|
| += | a+=b | a=a+b |
| -- | a-=b | a=a-b |
| *= | a*=b | a=a*b |
| /= | a/=b | a=a/b |
| %= | a%=b | a=a%b |
| b++ | a*=b++ | a=a*b; b++ |
| ++b | a*=++b | b++; a=a*b |

舉例來說：

```
a = a + 10;
```

a = a + 10 ;我們就會寫成 **a +=10;**

```
b = b - 5;
```

b = b - 5; 我們就會寫成 **b -=5;**

諸如此類…。等等。

Chapter 03

流程控制

學習程式語言的過程，在了解基礎的變數宣告以及資料型態之後，我們要知道程式中的流程如何控制與敘述，加以運用宣告的變數來做到任何我們想要達到的程式目的。

在第一章介紹當中，我們有提到結構化程式設計。接著，我們將會介紹選擇、重複結構的運用介紹。透過以上兩種結構，就能設計出具有結構化的程式。

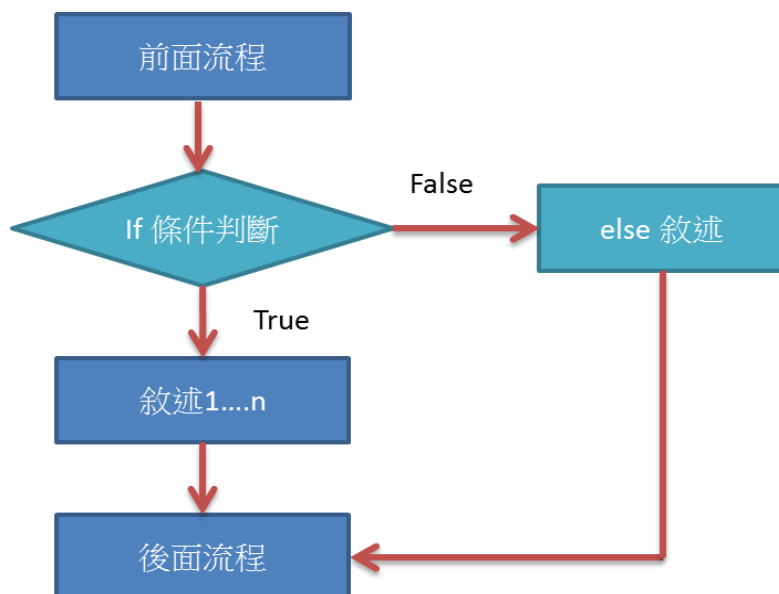
3-1 選擇性結構

選擇性結構 if

寫程式時，有時候會需要判斷某些條件或者狀況，給予不同回應，則可使用 if 選擇結構，亦即擁有條件性的敘述，必須滿足某設定條件，才能允許執行至指定的程式碼。

if...else 翻成白話為「如果...則...否則」。舉例來說，「如果今天的天氣是晴天，我們去陽明山賞花；否則，我們去看電影。」寫成程式的方式如下：

```
if (今天的天氣是晴天)
    陽明山賞花；
else
    看電影；
```



《程式語法》—if 敘述:依據判斷的結果執行不同的敘述

```
if (判斷條件)
{
    敘述 1;    //條件成立
    ...
    敘述 n;
}
else
    敘述 2;    //條件不成立
```

使用選擇性結構 if 時，在判斷條件中，如果要判斷兩個值是否相等要使用「==」，而不是「=」，這點常常造成初學者在撰寫程式上常發生的錯誤。另外，你的敘述式只有 1 行時，可以不用加「{」及「}」，可省略，意即，兩行以上就一定要有「{」及「}」將敘述式包覆。

《程式語法》—多重 if 敘述

```
if (判斷條件 1)
    敘述 1;
else if (判斷條件 2)
    敘述 2;
else if (判斷條件 3)
    敘述 3;
else
    敘述 4;
```

接著，我們來看一下實際的例子。輸入兩個整數，利用 if 敘述，判斷當 $a > b$ 時，印出 $a-b$ 即 a/b 的值；判斷當 $a < b$ 時，印出 $a+b$ 及 $a*b$ 的值。

《範例》

```
#include <stdio.h>
int main(void)
{
    int a, b;
    printf("first number:");
    scanf("%d", &a);
    printf("second number:");
    scanf("%d", &b);
    if (a>b)
    {
```

```
        printf("a-b=%d\n", a-b);  
        printf("a/b=%d\n", a/b);  
    }  
    else  
    {  
        printf("a+b=%d\n", a+b);  
        printf("a*b=%d\n", a*b);  
    }  
    return 0;  
}
```

《執行結果 1》

```
first number:2  
second number:1  
a-b=1  
a/b=2
```

當我們輸入兩個數字，第一個數字會傳遞到變數 a，第二個數字會傳遞到變數 b，第一個 if 條件是成立的同時，也就是當 $a > b$ ，就會執行括號內的敘述，印出 a-b 即 a/b 的結果。

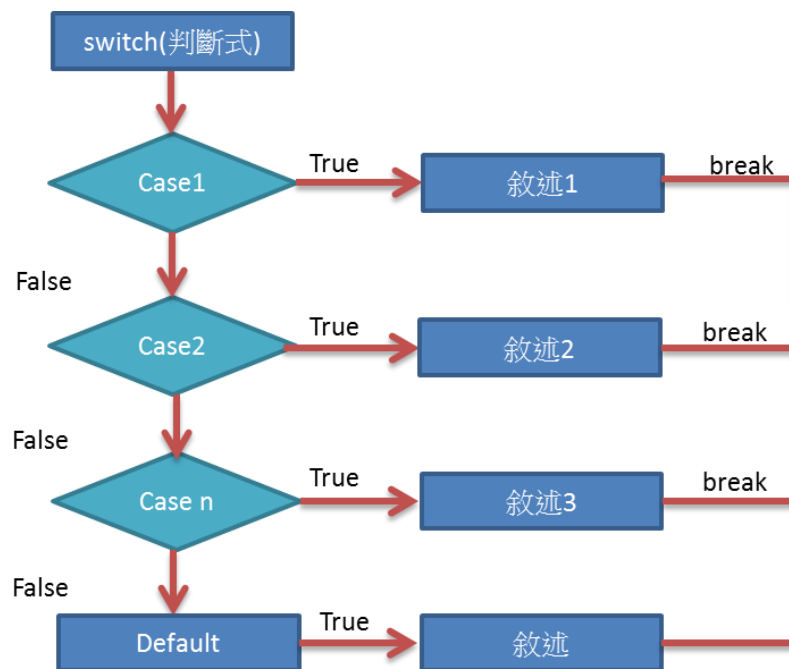
《執行結果 2》

```
first number:1  
second number:5  
a+b=6  
a*b=5
```

多重選擇性結構 switch

寫程式時，如果有許多選擇時(兩個以上), 可使用 switch 敘述, 它會依據某一種條件來判斷該進入何種程式區塊。當然，你也可以使用上一節所提到的多重 if 敘述來完成，但是過多的 if 敘述會使程式複雜度提高，使閱讀不易，增加維護難度，有效的使用 switch 敘述式，能簡化程式的設計。

一樣，讓我們來看看，多重選擇性結構 switch 的【流程圖】：



使用多重選擇性結構 switch 要在每個 case 的結尾都加上 break，讓編譯器知道，執行完這個 case 就跳出此敘述式，而 switch 跟其他控制流程一樣，由 {} 所組成。其中 default 是設定一個預設的程式敘述給這個結構，意即一進入該結構但找不到 case 中的條件，就會視為 default 中的程式敘述中作執行。

《程式語法》— 多重選擇性結構 switch

```
switch (變數名稱或運算式) {  
    case 符合數字或字元:  
        敘述一;  
        break;  
    case 符合數字或字元:  
        敘述二;  
        break;  
    default:  
        陳述三;  
}
```

接著讓我們來看一個範例，快速的了解 switch 的使用方法吧！

《範例》

```
#include <stdio.h>
int main(void)
{
    int grade, inputgrade;
    printf(" 請輸入分數:");
    scanf("%d", &inputgrade);
    grade= inputgrade/10;
    switch(grade) {
        case 9:
            printf(" 成績為 A");
            break;
        case 8 :
            printf(" 成績為 B");
            break;
        case 7 :
            printf(" 成績為 B");
            break;
        case 6 :
            printf(" 成績為 B");
            break;
        default:
            printf(" 成績為不及格");
    }
    return 0;
}
```

《執行結果》

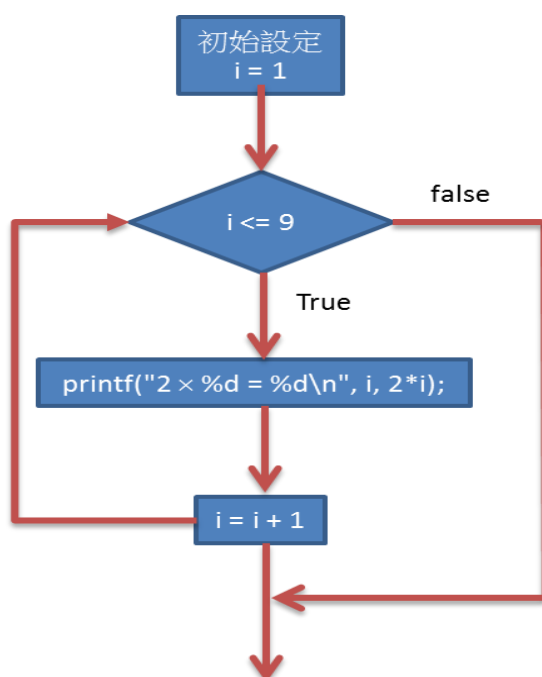
```
請輸入分數:90
A
請輸入分數:75
B
請輸入分數:55
成績為不及格
```

從結果我們可以看到，當分數大於 90 分以上會得到輸出值 A，介於 90~60 之間會得到 B，低於 60 分以下會看到” 成績為不及格”。

3-2 重複敘述功能

當某段程式敘述會因為在符合某種條件下需要重複執行多次(只有一些變數的值會作改變)，我們可以利用執行重複敘述的方法來求得答案。重複敘述指令有：for 迴圈、while 迴圈、do while 迴圈。

假設我們要計算 2×1 ， 2×2 ， 2×3 ， 2×4 ， 2×5 ， 2×6 ， 2×7 ， 2×8 ， 2×9 ，只有乘數(\times 右邊)的數字在改變，每次都增加 1。則我們可以設計一個運算式， $2 \times i$ 每次計算完這個運算式， i 都會加 1。(i 的起始值是 1，最後結束時的值是 9。)



重複性結構-for

當程式需要來回重複執行某一段程式碼時，就可以使用 for 迴圈來完成，就好比我們常常利用程式來做一些運算，但程式執行的優點在於，使用重複性結構可以利用短短的幾行語法，做出重複的程式。

《程式語法》－重複性結構 for 迴圈

```
for (初值運算式; 測試運算式; 增減量運算式)
{
    程式敘述;
}
```

初值運算式:設定該 for 迴圈初始的設定。

測試運算式:判斷迴圈結束的條件。如果傳回 true 的值，敘述的指令就會被執行。反之傳回 false 的值就結束此迴圈的執行。

增減量運算式:遞增或遞減變數的值。

簡單的原則來說，測試運算式為真(true)執行迴圈內指令，測試運算假(false)就跳出結束迴圈。

以上述例子來說，如果要創造一個 2x(1~9)的式子利用 for 迴圈的寫法，初值運算式為(i=1)；測試運算式(i<=9)；增減量運算式(i=i+1)或者(i++)。意即此迴圈的初始條件為 i=1 的時候，每次執行完 i 會加 1(增量運算)，直至 i<=9 就會結束這個迴圈。

重點小提醒：迴圈內由分號(;)來分隔運算式，而非逗號(,)。

《範例》

```
#include <stdio.h>
int main(void)
{
    int i ;
    for(i=0; i<=9 ; i++)
    {
        printf("2*%d = %d\n", i, 2*i);
    }
    return 0;
}
```

《執行結果》

```
2*0  = 0
2*1  = 2
2*3  = 6
2*4  = 8
2*5  = 10
2*6  = 12
2*7  = 14
2*8  = 16
2*9  = 18
```

透過 for 迴圈，我們可以做出一個 2*0 到 2*9 的結果輸出制螢幕上。For 的迴圈初始值為 i=0 的時候，每次執行一次 printf()，i 就會作累加的動作，直到 i<=9 會終止該迴圈。

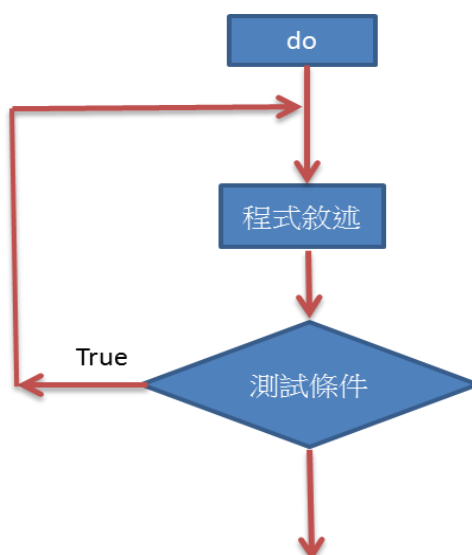
前測式重複性結構-while

如果迴圈的重複執行次數是無法預先確定的話，可以使用 while 迴圈來取代 for 迴圈(重複執行次數是可以預先確定的)，如果不知道迴圈何時結束，但可以設定某條件(測試條件)是否成立的情況下，也可以使用 while 迴圈。

《程式語法》—前測式重複性結構 while

```
while (測試條件)
{
    程式敘述;
}
```

先判斷測試條件之真假，若為真則執行迴圈內敘述後，再次判斷測試條件之真假，若為真則再度執行迴圈內敘述，直到判斷測試條件為假，方能離開迴圈。



《範例》累加 1 到 100 的所有數(1+2+3+...+100)

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i=1, sum=0;
    while(i<=100)
    {
        sum += i;
        i++;
    }
}
```

```

    }
    printf("1+2+3+...+100 = %d\n", sum);
    system("pause"); //使程式暫停在執行畫面讓我們看到結果
    return 0;
}

```

《執行結果》

```
1+2+3+...+100 =5050
```

While 迴圈亦如同介紹一般，只要滿足 while 內的條件，就會不斷的執行，根據此例看來，只要 i 的值 ≤ 100 ，就會不斷執行 while 迴圈內的敘述，也就是進行累加的動作，並且將值存入 sum 變數當中，最後當 $i > 100$ 就會跳出 While 迴圈，並且將結果輸出螢幕。

後測式重複結構 do while

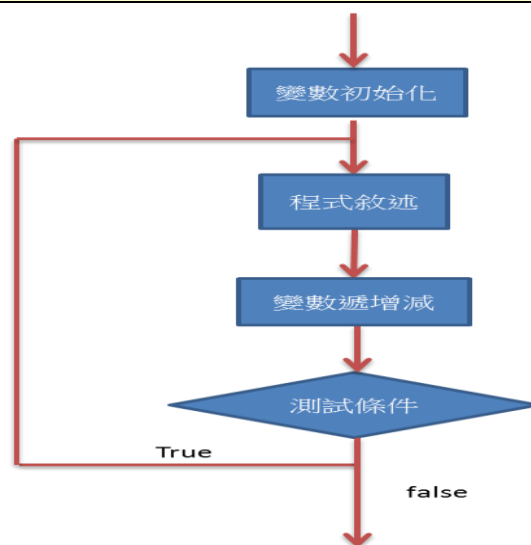
同樣的，迴圈的重複執行次數無法預先確定，可以使用 do while 迴圈來取代 for 迴圈(重複執行次數是可以預先確定的)。但 do while 迴圈與 while 迴圈的不同在於：do while 迴圈會先執行一次迴圈內的程式敘述，再去看條件(測試條件)是否成立。

《程式語法》－後測式重複結構 do while

```

do
{
    程式敘述;
    變數遞增減;
}while (測試條件);

```



《範例 1》:印出 1 到 10 的所有數

```
#include <stdio.h>
int main()
{
    int i=1;
    do{
        printf("%d\n", i);
        i++;
    }while(i<=10);
    return 0;
}
```

《執行結果 1》

```
1
2
3
4
5
6
7
8
9
10
```

透過後測式迴圈 do while 的做法，程式會先執行在檢查是否在符合的測式/條件下，由此例我們可以看出，一進入 do while 迴圈，立即就會先做 print 的動作，而每做一次 i 就會做一次累加，接著執行完程式敘述之後，會檢查 $i \leq 10$ 是否成立，當 $i=11$ 的時候即跳出此迴圈。

《範例 2》:累加 1 到 1000 的所有數($1+2+3+\cdots+1000$)

```
#include <stdio.h>
int main()
{
    int i=1, sum=0;
    do{
        sum += i;
        i++;
    }while(i<=1000);
}
```

```
    printf("1+2+3+...+1000 = %d\n", sum);  
    return 0;  
}
```

《執行結果 2》

1+2+3+...+1000 = 500500

Chapter 04

陣列

陣列是一種結構性的資料儲存空間，由相同資料型態的變數所組成的集合，元素與元素之間的記憶位置是相鄰在一起，通常我們利用一個變數來代表整體的資料，也就是這個陣列。

假設我們要計算一個月當中所開銷的伙食費，不使用陣列來做運算，d1 表示第 1 天的開銷花費，d2 表示第 2 天的開銷花費，以一個月 30 天做計算，依此類推，第 30 日則以 d30 表示，這時候計算加總無法使用重複性結構，需要寫成如下格式：

```
cost=d1+d2+d3+d4+d5+d6+d7+d8+d9+d10+d11+d12+d13+d14+d15+d16+d17+d18+d19+d20+d21+d22+d23+d24+d25+d26+d27+d28+d29+d30;
```

當數量少的時候可能不覺得不妥，倘若我們要計算一年度的伙食花費，就要算 12 次，如果要再多算幾年分的花費，就會顯得麻煩，並且撰寫不便。

陣列因此而誕生，我們先來看看陣列的用法。

陣列的宣告

C 語言中，陣列的表示方法用[]來表示，陣列可以宣告一個以「索引」(index)作為識別的資料結構，宣告陣列的方式如下：

資料型態 名稱[大小];

```
int cost[10];
```

cost 為此陣列的名稱，表示 cost 是一維陣列，此陣列共有 10 個元素，我們先假設每個月都有 10 筆花費。其中陣列內每個元素皆為整數型態的變數，分別為 cost [0], cost [1], cost [2], ..., cost [9]。注意陣列的起始索引是從 0 開始，而非 1，故最後一個元素為 9(10 減 1)，所以我們可以知道，陣列的第 4 個元素是 cost [3]，第 7 個元素則是 cost [6]。

| | | | | |
|---------|---------|---------|-------|---------|
| cost[0] | cost[1] | cost[2] | | cost[9] |
|---------|---------|---------|-------|---------|

```
int cost [10] = {100, 150, 200, 230, 500, 100, 150, 130, 200 , 225};
```

另外，在陣列的宣告當中，我們亦可以直接指定陣列內的值。好比上述的例子中，我們宣告一個陣列名為 `cost`，而此陣列的大小為 10，其中陣列內的值，我們可以連同宣告的同時一同指定。所以，`cost [0] = 100`，`cost [3] = 230`，`cost [9]=255`。

| | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 100 | 150 | 200 | 230 | 500 | 100 | 150 | 130 | 200 | 225 |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|

`cost[0] cost[1] cost[2] cost[3] cost[4] cost[5] cost[6] cost[7] cost[8] cost[9]`

有了陣列來記錄我們一個月的花費之後，現在我們要來計算這一個月的總開銷花費金額為多少，也就是累加每天的花費，這時我們只需要這樣做：

```
int sum =0;
for(int i=0; i<30; i++)
{
    sum += cost[i];
}
printf( “總金額: %d” ,sum);
```

結果：

總金額： 1985

接著我們再來看幾個相關範例吧！

《範例》

```
#include <stdio.h>
int main()
{
    int arr[5] = { 10, 20, 30, 40, 50};
    int total = 0;
    printf( “arr 陣列中包含的元素有:\n” );
    for(int k=0; k<5; k++)
    {
        printf( “%-5d” , arr[k]);
        total += arr[k];
    }
    printf( “arr 元素總和為: %d\n” , total);
    return 0;
}
```

《執行結果》

arr 陣列中包含的元素有：

10 20 30 40 50

arr 元素總和為:150

透過 for 迴圈，我們除了將 arr 陣列中的值讀取出來之外，進而累加至 total 變數當中，最後進行加總後輸出。這是一個典型的陣列用法，在宣告陣列的同時，就給予初值，接著對陣列中的內容取出加以應用。

《範例》輸入 2 個實數，並求其平均值)

```
#include<stdio.h>
int main() {
    float data[3], sum=0;
    int count =2;
    for (int i = 0; i < count; i++) {
        printf("請輸入數字資料[%d]：", i);
        scanf("%f", & data [i]);
        sum = sum + data [i];
    }
    printf("您輸入的數字平均數為： %f\n", sum/ count);
    return 0;
}
```

《執行結果》

請輸入數字資料[0]： 10

請輸入數字資料[1]： 50

您輸入的數字平均數為:30

我們宣告了一個 data 的陣列，其型態為 Float 浮點數，透過 for 迴圈讓使用者輸入兩次數字資料，接著每次都將資料累加到 sum 變數當中，最後在輸出的同時，取平均除法運算。

Chapter 05

模組化程式設計

5-1 模組(Module)和副程式(subprogram)

在設計一個龐大的程式時，我們通常會將程式分成許多個部分來分開處理，這如同我們日常生活在處理一個龐大的問題時，會將問題分解成許多個小問題分開處理，最終再將每個小問題處理後的結果合併處理來得到原本問題的解答，所謂的程式的模組化就是這樣的一個概念。在程式設計中，程式通常由數個**模組**(Module)所組成，而每一個模組可能由一個以上的**副程式**(subprogram)所組成。

副程式又可稱為**函式**(function)，副程式是一個獨立被設計出來完成特定工作的程式碼。因為某些功能的程式碼常常不同的地方被用到，所以會導致許多地方都有重複相同的程式碼片段，這樣會使得程式碼變得十分冗長且難以維護。而將這些程式碼設計成一個副程式便可增加程式的可讀性，並且以後再進行維護時不需要修改許多地方，只需將獨立出來的副程式內容做修改即可。

建立並使用副程式

我們可以由下段程式碼來說明如何建立一個副程式。

《範例》

```
#include<stdio.h>
int add(int, int);

int main()
{
    int sum, a = 5, b = 3;
    sum = add(a, b);
    printf("%d", sum);
    return 0;
}

int add(int num1, int num2)
{
    int ans;
    ans = num1 + num2;
```



```
return ans;  
}
```

```
int add(int, int);
```

上述的程式碼稱為**宣告**，這段程式碼的意義是宣告一個名為 add 的副程式，回傳資料型態是 int，而呼叫(Call)副程式時，需將兩個資料型態為 int 的資料傳入副程式中。在這邊我們須了解兩個副程式的基本概念。

1) **回傳資料型態**副程式可依回傳資料型態分成兩類，會回傳值和不回傳值。若我們不希望副程式回傳值，則在宣告回傳資料型態時以 void 宣告，相對而言，若希望副程式回傳值，則依據我們希望副程式回傳的資料型態來做宣告。如同上例來說就是 int。

2) **傳入資料型態**副程式在撰寫時，有時會因為撰寫需求而必須傳入變數到副程式，而決定要傳入什麼型態的資料進入副程式也是在宣告時就需決定好。對於上例而言，我們宣告了副程式 add 在被呼叫時需要傳兩個 int 的資料。

《範例》

```
int add(int num1, int num2)  
{  
    int ans;  
    ans = num1 + num2;  
    return ans;  
}
```

上面這段程式碼為副程式的**主體**，這邊是用來讓我們撰寫副程式碼內容的區塊。程式碼的第一行即是我們宣告該副程式時的定義，但是需更進一步的定義傳入資料型態的變數名稱，對範例程式碼而言，我們將副程式傳入的變數名稱宣告成 num1 和 num2。如果有宣告回傳資料型態，在主體的最後需要做回傳值的動作，即為 return 動作。

```
sum = add(a, b);
```

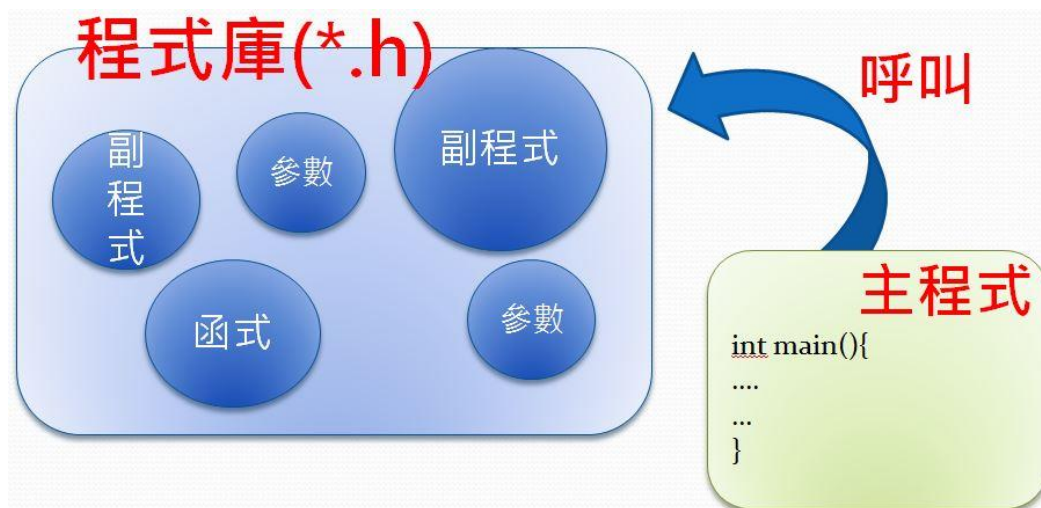
如果副程式有回傳值，我們可以直接把回傳的值指定給一個變數。在 C 語言中，並非一定要將副程式的回傳值指定給一個變數。

5-2 程式庫

程式庫和標頭檔

是將許多常用的副程式與參數存放的檔案(副檔名為.h), 需要使用的時候, 只要於主程式呼叫即可, 程式庫的來源可分成兩種: C 語言提供和自定義程式庫。

副檔名為.h 的檔案稱為標頭檔, 開發大型程式時, 通常會把介面與實作分開。這是說, 常數、結構的定義與函數原型放在標頭檔中, 也就是副檔名為 .h 的檔案, 此即為介面, 然後把函數的實際定義放在另一個 .c 的原始程式碼檔案裡, 這就是實作。實作檔須留意要用 #include 的前置處理器指令將標頭檔包含進來, 至於實際程式的測試與執行, 就另寫一個含有函數 main() 的 .c 的原始程式碼檔案, 然後將實作檔與含有 main() 的原始檔一起編譯, 便可產生可執行檔。標頭檔可從 C:\Dev-Cpp\include 資料夾中找到。



常見的程式庫

工具函數<stdlib.h>

標準輸出輸入函數<stdio.h>

字串函數<string.h>

數學函數<math.h>

字元檢查函數<ctype.h>(ex. isdigit, isalpha 等)

日期/時間函數<time.h>

《範例》

```
int main(){
    float answer, x;
    inti, n, temp;
```

```
temp = 1;
for(i=1; i<n; i++){
    temp = temp*2;
}
answer = x*temp;
}
```

《範例》

```
#include<math.h>
int main(){
    float answer, x;
    int n;
    answer = ldexp(x, n);
}
```

程式庫的使用方法

利用#include（前置處理器）含括程式庫，使用格式有兩種：

#include <標頭檔>：編譯器會去/include 的資料夾尋找程式庫的檔案。

#include "標頭檔"：編譯器會在和主程式同一個位置的資料夾尋找程式庫的檔案。

自定義標頭檔的使用方法

1. 開新檔案&撰寫程式

利用 Dev-C++或是記事本編輯程式，於檔案中定義下列程式碼。

《範例》

```
#define ADD(a, b) (a+b)
#define SUB(a, b) (a-b)
#define MUL(a, b) (a*b)
#define DIV(a, b) (a/b)
```

2. 另存新檔，副檔名為.h

新增一個資料夾於桌面，依題目要求將檔名命名為 operation.h，將檔案另存於桌面的資料夾內。

3. 到主程式內利用#include 引入並呼叫欲使用之程式

在主程式利用#include" operation.h" 引入並於主程式撰寫下列程式碼並執

行

《範例》

```
#include <stdio.h>
#include <stdlib.h>
#include "operation.h"

int main()
{
    int a, b;
    printf("請輸入兩個整數a, b:\n");
    scanf("%d %d", &a, &b);
    printf("a+b = %d\n", ADD(a, b));
    printf("a-b = %d\n", SUB(a, b));
    printf("a*b = %d\n", MUL(a, b));
    printf("a/b = %d\n", DIV((float)a, (float)b));
    system("pause");
    return 0;
}
```

《範例》

請撰寫一個 `abssqu.h` 的自訂標頭檔，裡面定義計算絕對值的方式和計算數字的平方。在自定義的標頭檔內，除了利用 `#define` 外，也可撰寫程式。

```
#include<stdio.h>
#include<stdlib.h>
int ABS(int a)
{
    if(a<0)
        a = -a;
    return a;
}
int SQUARE(int a)
{
    int ans;
    ans = a*a;
    return ans;
}
```



可在自定義標頭檔內引入
C語言原有的程式庫



可在自定義標頭檔內撰寫
需要的副程式

主程式的部分，只需引入標頭檔，即可使用標頭檔內的資料。

《範例》

```
#include <stdio.h>
#include <stdlib.h>
#include "abssqu.h" //使用自訂標頭檔abssqu.h

int main()
{
    int a;
    printf("請輸入任一整數:\n");
    scanf("%d", &a);
    printf("輸入的整數之絕對值為: %d\n", ABS(a));
    printf("輸入的整數之平方為: %d\n", SQUARE(a));
    system("pause");
    return 0;
}
```

Chapter 06

進階資料型態

6-1 陣列(Array)

陣列是一種結構性的資料儲存空間，其同一陣列裡的資料性質相同，元素與元素之間的記憶體位置是相鄰的，通常我們利用一個變數來代表整體的資料。根據陣列的結構而言，可以把陣列分為一維陣列、二維陣列、多維陣列。宣告方法如下：

一維陣列： 資料型態 陣列名稱[陣列大小]；
二維陣列： 資料型態 陣列名稱[個數][個數]；
多維陣列： 資料型態 陣列名稱[個數][個數] [個數].....；

一個「m*n 陣列」的 int 陣列，會有 m 列 n 行，m*n 個元素。宣告方法為

```
int score[m][n];
```

設定初始值

我們可以想像「m*n 陣列」是由 m 個一維陣列(內含 n 個元素)所組成，所以可以透過下列方式設定初始值。

```
int score[2][4] = { {95, 85, 76, 84} , {62, 84, 99, 71} }
```

2 列 x 4 行的陣列 一維陣列，有 4 個元素 一維陣列，有 4 個元素

2 列 x 4 行的陣列即由 2 個一維陣列(內有 4 個元素)所組成

此程式碼在數學矩陣上的表示為：

| | 第 0 行 | 第 1 行 | 第 2 行 | 第 3 行 |
|-------|-------------------|-------------------|-------------------|-------------------|
| 第 1 列 | Score[0][0] 95 | Score[0][1] 85 | Score[0][2] 76 | Score[0][3] 84 |
| 第 2 列 | Score[1][0] 62 | Score[1][1] 84 | Score[1][2] 99 | Score[1][3] 71 |

《範例》第一個索引值不填的表示法

```
int score[][4] = { {97 , 87 , 67 , 28} ,  
                  {85 , 84 , 59 , 48} ,
```

```
{33 , 84 , 32 , 17}};
```

二維與二維以上的陣列，設定初始值時可以省略第一個索引值，但是其他的大小仍須填寫。

二維陣列的使用方式

二維與二維以上的陣列，在輸入值的時候可使用下列方法來處理。

《範例》程式內部寫入值

```
int score[0][0]=80;  
int score[1][5]=100;
```

或可由鍵盤輸入的數值至設定於陣列索引值[1][3]的元素後，再輸出內容

《範例》鍵盤輸入值

```
scanf( "%d" , &score[1][3]);  
printf( "%d" , score[1][3]);
```

或可使用迴圈處理

《範例》使用迴圈輸入值

```
int i, j, k, score[2][4];  
for(i=0; i<2; i++)  
{  
    for(j=0; j<4; j++)  
    {  
        scanf("%d", &score[i][j]);  
    }  
}
```

《範例》使用迴圈輸出值

```
int i, j, k, score[2][4];  
for(i=0; i<2; i++)  
{  
    for(j=0; j<4; j++)  
    {  
        printf("%d\n", score[i][j]);  
    }  
}
```

多維陣列的使用方式

《範例》2 X 3 X 4 的三維陣列

```
int score[2][3][4] = { { {97, 87, 67, 28},  
                        {85, 84, 59, 48},  
                        {33, 84, 32, 17}},  
                      {12, 78, 76, 82},  
                      {28, 59, 17, 84},  
                      {65, 48, 23, 59}} };
```

《範例》使用迴圈輸入值

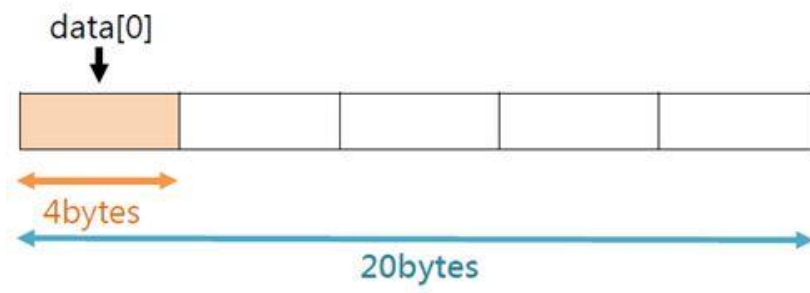
```
int i, j, k, score[2][4][6];  
for(i=0; i<2; i++)  
{  
    for(j=0; j<4; j++)  
    {  
        for(k=0; k<6; k++)  
            scanf("%d", &score[i][j][k]);  
    }  
}
```

《範例》使用迴圈輸出值

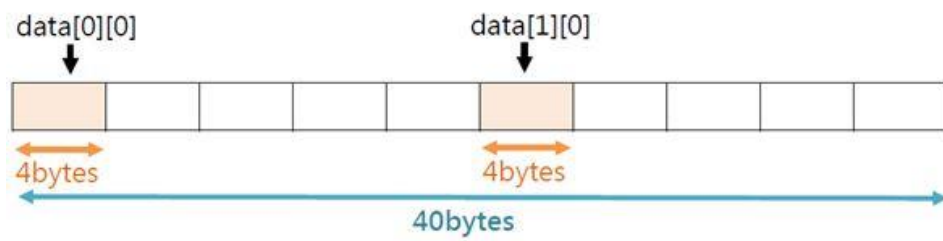
```
int i, j, k, score[2][4][6];  
for(i=0; i<2; i++)  
{  
    for(j=0; j<4; j++)  
    {  
        for(k=0; k<6; k++)  
            printf("%d\n", score[i][j][k]);  
    }  
}
```

陣列的記憶體配置方式

整數資料型態占用的位元組為 4bytes，宣告一維整數陣列 `int data[5]`。則



整數資料型態占用的位元組為 4bytes，宣告二維整數陣列 `int data[2][5]`。則



6-2 資料錄

資料錄

一種資料型態。在 C 語言中，我們稱之為結構(structure)。結構可以將許多不同資料形態的變數、陣列組合在一起，形成新的資料型態。定義一個結構，等同於定義一個新的資料型態。簡單來說其實就是自訂的資料型態，因此宣告方法與一般的資料型態並無不同。而要如何存取結構的成員變數呢？在 C 語言中是採用運算子「.」來做存取的動作。

定義方式：

```
struct 結構名稱
{
    資料型態 結構成員名稱 1;
    資料型態 結構成員名稱 2;
    ...
    資料型態 結構成員名稱 n;
};
```

《範例》

```
struct member
{
    char name[10];
    char gender;
    int age;
}
```

宣告方式：

```
struct 結構名稱 變數 1, 變數 2, ..., 變數 n;
```

```
struct member a, b, c;
```

也可在定義時做宣告

《範例》合併定義與宣告

```
struct 結構名稱
{
    資料型態 結構成員名稱 1; 資料型態 結構成員名稱 2;
    ...
    資料型態 結構成員名稱 n;
} 結構變數1, 結構變數2, ..., 結構變數n; //定義完直接宣告
```

運算子「.」的使用方式：

```
struct member a, b, c;  
a.name;  
c.gender;
```

宣告的 member 結構變數 a, b, c，進行下列動作：

1. 利用 gets() 函式讀取 a 和 b 的名字
2. 輸入 b 的性別與年紀
3. 設定 c 的性別為 F，年紀為 16。

《範例》

```
struct 結構名稱  
printf("請輸入 a 和 b 的名字");  
gets(a.name);    //利用 gets() 讀取 a 的姓名  
gets(b.name);    //利用 gets() 讀取 b 的姓名  
printf("請輸入 b 的性別(F/M)與年紀");  
scanf("%c%d", &b.gender, &b.age);    //讀取 b 的性別與年紀  
c.gender = 'F';    //設定 c 的性別為 F  
c.age = 16;        //設定 c 的年紀為 16
```

typedef 指令

typedef 無法定義新的型態，但可以將現有的型態定義新的名稱。可節省撰寫程式的時間和使程式更加結構化、更容易閱讀。下例是利用 typedef 將現有資料型態 struct member 定義為新的型態名稱 INFO 的一個範例

《範例》

```
struct member  
{  
    char name[10];  
    char gender;  
    int age;  
};  
typedef struct member INFO;
```

結構陣列

當想要使用許多個結構變數的時候，可以宣告結構陣列，利用索引值指出正確的陣列元素，即可存取該結構陣列元素內的成員。

《範例》

```
struct member a[10];    //宣告結構陣列 a
a[0].gender = 'F';      //設定陣列 a[0]的 gender 為 F
a[1].age = 13;          //設定陣列 a[1]的 age 為 13
gets(a[3].name);        //利用 gets()讀取 a[3]的姓名
```

《範例》

```
int i;
struct member a[10];
for(i=0; i<10; i++)
{
    printf( "請輸入姓名：\n" );
    gets(a[i].name);
    printf( "請輸入性別、年紀：\n" );
    scanf( "%c %d" , &a[i].gender, &a[i].age);
}
```

巢狀結構

```
struct 結構 a
{
    //結構 a 的成員
}
struct 結構 b
{
    //結構 b 的成員
    struct 結構 a 變數名稱;
}
```

結構 b 為巢狀結構，結構 b 包含了結構 a，所以結構 a 必須寫在結構 b 的前面。存取結構 b 中結構 a 的成員，仍以「.」進行存取

《範例》

```
struct date          //定義結構 date
{
    int month;
    int day;
};
struct info          //定義結構 info
{
    char name[10];
```

```
    struct date birth;  
} mary;           //宣告結構 info 變數 mary  
mary.birth.month = 2;    //設定 mary 生日的月份  
mary.birth.day = 18;     //設定 mary 生日的日期
```

6-3 指標

指標(Pointer)

指標就是內含記憶體位址(address)的變數。它是由 C 語言所提供的一種資料型態，用來存放變數在記憶體中的位址。我們透過指標(pointer)來存取記憶體存放的變數和程式這些資料，藉由傳遞指標，可以將程式碼簡化，也可以增加程式的執行效率。當指標存放變數 a 的位址時，我們稱「指標 Ptr 指向變數 a」。但是確定指標變數所指向的資料型態後，便不能再更改指向的資料型態。

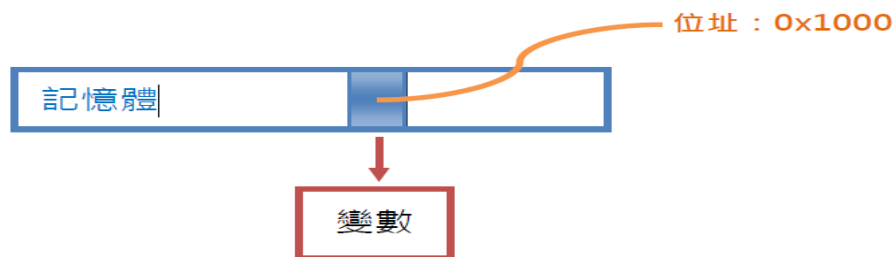
優點：

函式在傳遞陣列或字串時更有效率。

將複雜的資料結構鏈結在一起，如：鏈結串列(linked list) 藉由指標傳達記憶體訊息，如：記憶體配置函數 malloc()會回傳位址，此時，我們就需要利用指標來儲存這個位址。

位址(address)

當宣告一個變數時，編譯器就會配置一塊記憶體空間給這個變數。每一個記憶體空間都有它自己獨一無二的編號，我們稱這些編號為記憶體的位址(address)。當知道記憶體的位址時，程式就可以利用它來存取變數的資料。



使用方法

位址運算子 → &：

取出變數的位址，然後儲存於指標變數之中。下例中示範了在變數前方加上「&」，便可取得該變數的位址的方法。

《範例》

```
int a = 10;
printf(“a 的記憶體位址為：%p”, &a);
```

《執行結果》

a 的記憶體位址為：0022FF44

依址取值運算子 → *

取出指標變數所指向之變數的值。

《範例》

```
int a = 10;
int *Ptr;
Ptr = &a;    /*指標 Ptr 儲存 a 的位址 */
printf("a 的位址為：%p\n", &a);
printf("Ptr 為：%p\n", Ptr);
printf("*Ptr 為：%d\n", *Ptr);
```

《執行結果》

```
a 的位址為：0022FF44
Ptr 為：0022FF44
*Ptr 為：10
```

將指標傳遞至函式

宣告函式(將指標傳入) - 不必填上指標變數名稱

《範例》

```
void print_address(int*);    //宣告函式

int main()
{
    int num, *ptr;
    num = 10;
    //設定指標變數ptr儲存num的位址
    ptr = &num;

    printf("傳入ptr的結果:\n");
    print_address(ptr); //呼叫函式並傳入指標變數
    printf("傳入&num的結果: \n");
    print_address(ptr); //呼叫函式並傳入位址
    system("pause");
    return 0;
}

void print_address(int *a)    //撰寫函式
{
```

```
printf("位址: %p, 內容: %d\n", a, *a);
}
```

指標與陣列

當宣告一陣列時，陣列名稱可以當作指標使用。當陣列名稱當作指標使用時，指標會指向陣列的第一個元素，但不是整個陣列。而且不可任意對陣列的名稱直接進行運算。

《範例》

```
int *ptr, a[5]={1, 2, 3, 4, 5};
ptr = a+3;
printf("%d %d", *a, *ptr);
```

《執行結果》

```
1 4
```

陣列 a 的第一個元素為 1，所以指標指向第一個元素。

指標 ptr 指向陣列 a+3，即第 4 個元素，印出結果：*a=1，ptr*=4。

指標與字串

除了利用陣列儲存字串外，也可利用指向字元型態的指標變數儲存字串。

| 宣告方式 | 舉例 | 差別 |
|-----------------------|----------------------|---------------------------|
| char 陣列名稱[]="字串內容" ; | char a[]="hello!" ; | 將陣列名稱當作指標，不可以任意對陣列的名稱進行運算 |
| char *ptr="字串內容" ; | char *ptr="hello!" ; | 可以直接對指標進行運算 |

《範例》

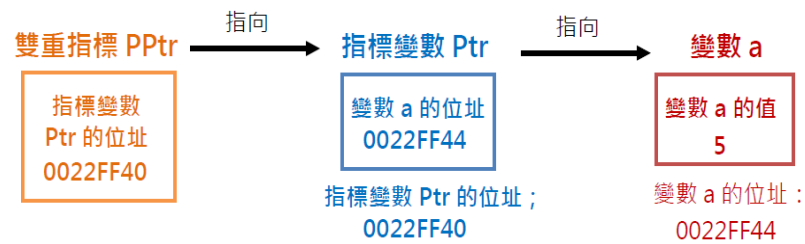
```
char *ptr = "hello! world!";
ptr = ptr + 3;          //可直接對指標進行運算
printf("%s", ptr);
```

《執行結果》

```
hello! world!
```


雙重指標

每個變數都有自己的位址，指標變數也有自己的位址，而雙重指標就是用來儲存指標變數的位址。



《範例》

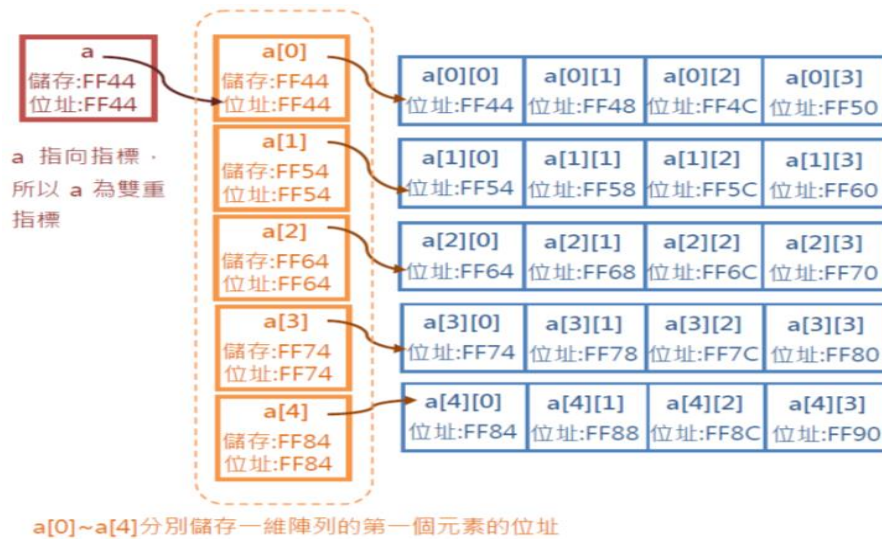
```
int a = 5;
int *ptr, **pptr;
ptr = &a;           //設定ptr為a的位址
pptr = &ptr;        //設定pptr為ptr的位址

printf("ptr= %p\n", ptr);
printf("ptr= %d\n", *ptr);
printf("pptr= %p\n", pptr);
printf("pptr= %d\n", *pptr);
```

《執行結果》

```
ptr = 0022FF44
ptr = 5
pptr = 0022FF40
pptr = 5
```

雙重指標與二維陣列的關係



用指標來表示陣列元素 $a[i][j]$ ，我們可以將其表達為： $*(*(a+i)+j)$;

《範例》

```
int a[2][3] = {{1, 3, 5},
               {2, 4, 6}};

int i, j;
for(i=0; i<2; i++)
{
    for(j=0; j<3; j++)
        printf("a[%d][%d]=%d", i, j, *(*(a+i)+j));
    printf("\n");
}
```

《執行結果》

```
a[0][0] = 1 a[0][1] = 3 a[0][2] = 5
a[1][0] = 2 a[1][1] = 4 a[1][2] = 6
```

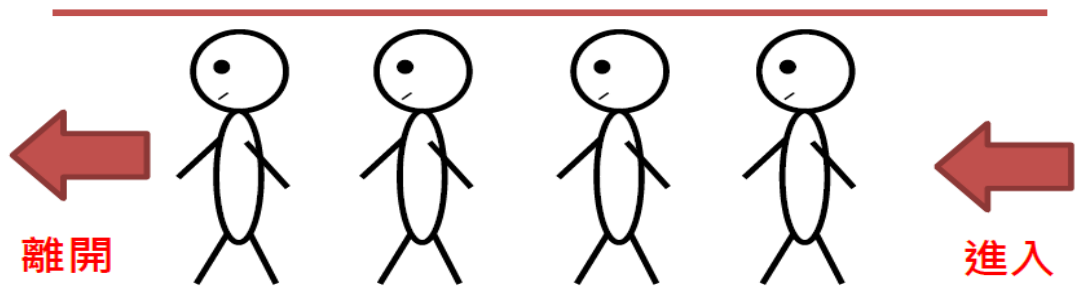
Chapter 07

資料結構

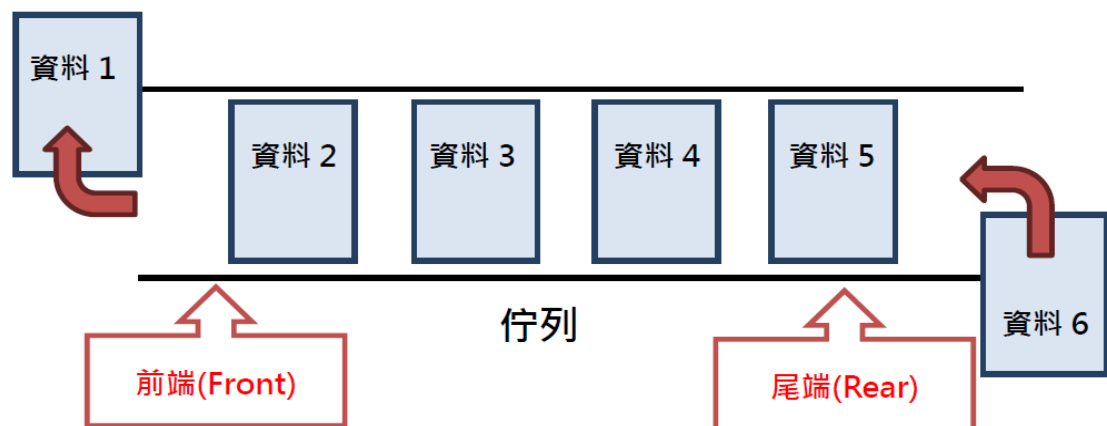
資料結構主要是一門如何把原始「資料」(data)，組織、安排、存放到電腦中的一門學問。程式中的資料結構如果設計良好，可以節省儲存空間、增加資料的安全性與增加程式的執行速度。基本的資料結構包含陣列(Array)、串列(List)、堆疊(Stack)、佇列(Queue)、樹狀(Tree)及圖形(Graph)等，這幾種基本的資料結構乍看之下雖然非常抽象，其實在日常生活中經常可以看到。

7-1 佇列(Queue)

平時我們排隊買票、排隊結帳、排隊辦事…先進入隊伍排隊的人，可以先買到票、先結完帳、先辦好事，這種依照順序先進入隊伍可以先進行工作的資料結構，稱為佇列(Queue)。而其他現實中佇列的應用有印表機列印報表、記憶體緩衝區和接龍遊戲中的循環翻牌。



我們可將上圖中人想成是資料，而整個進出的過程就是佇列的存取方式。



如上圖，我們將每個人當作一筆資料。資料加入從佇列的尾端 (Rear) 加入，從前端 (Front) 取出。

先進先出(FIFO)

佇列有一個入口及一個出口。新的資料由尾端(Rear)加入，而由前端(Front)取出資料。這種資料存取的順序在資料結構中稱為先進先出(First In First Out , **FIFO**)。

佇列的指令和使用方式

佇列的基本動作可分為下面四種

```
//對佇列進行輸入與輸出的動作
enqueue()    //將資料加入佇列尾端。
dequeue()    //將資料由佇列的前端取出

//判斷佇列是否已經填滿或是空佇列
isFull()     //檢查佇列是否已滿，以便判斷是否還可以存入資料。
isEmpty()    //檢查佇列是否是空的，以便判斷是否還有資料可以取出。
```

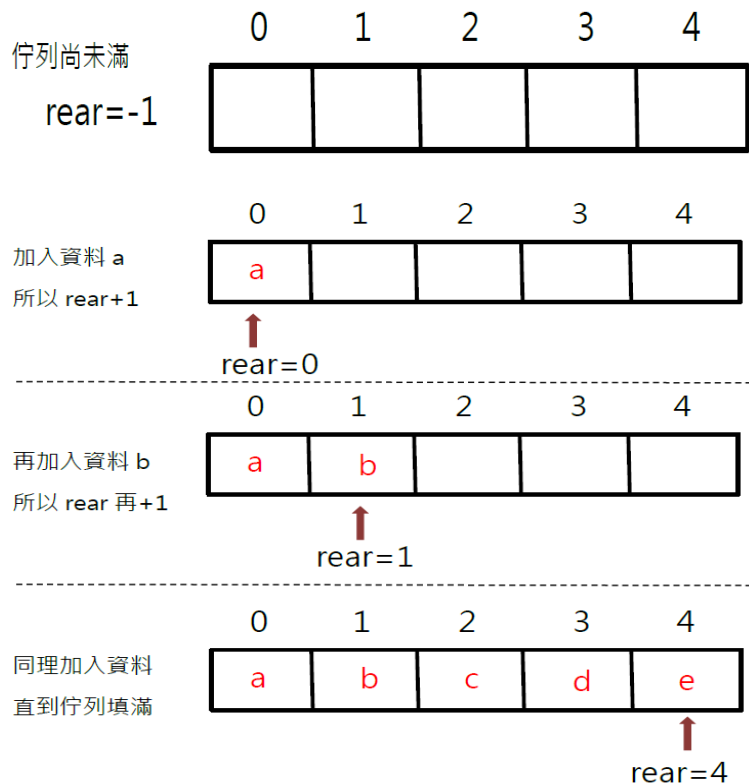
enqueue()

假設 rear 初始值為-1

步驟 1：判斷佇列是否已滿

步驟 2：將尾端指標 rear 往後移動，即 rear+1

步驟 3：將值存入 rear 所在位置的陣列元素。



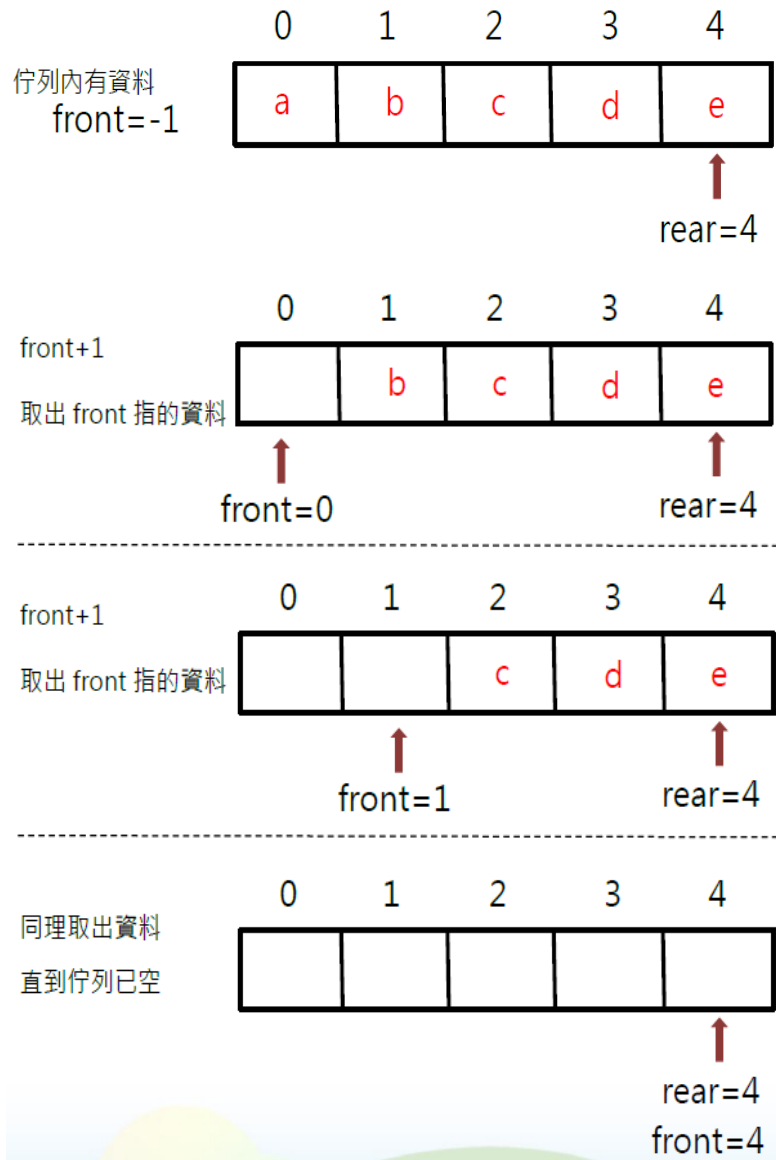
dequeue()

假設 front 初始值為-1

步驟 1：判斷佇列是否是空的(空佇列無法取出資料)

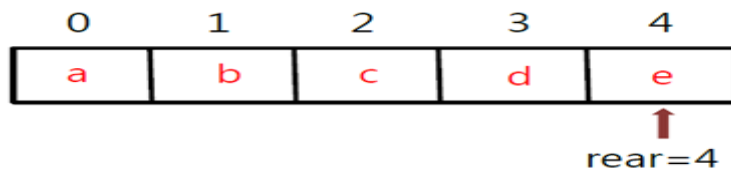
步驟 2：將前端指標 front 往後移動，即 $\text{front}+1$

步驟 3：取出前端指標 front 所指的陣列元素



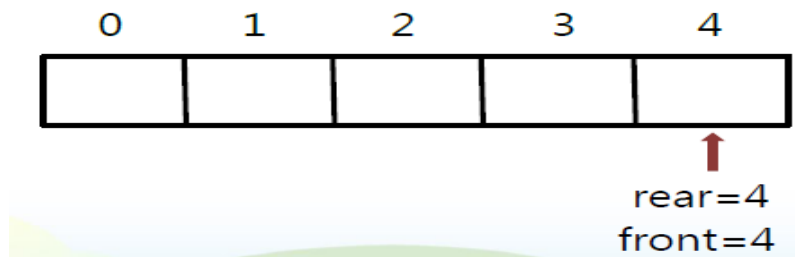
isFull ()

每當想加入一筆新資料的時候，要先判斷佇列是否為滿。判斷的方法是判斷 rear 再往後一個位置是否已超過佇列大小。若 $(\text{rear}+1) > \text{佇列大小}$ ，則佇列已滿。



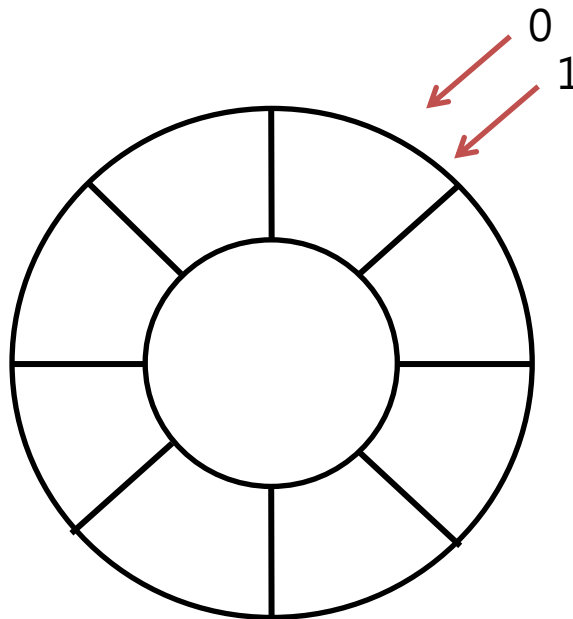
isEmpty ()

用來判斷是否還有資料可以取出。判斷的方法是當 `front` 和 `rear` 相等時，佇列為空。



環狀佇列

想像將佇列的最末端和最前端連接，成為一個環狀。將佇列以環狀呈現，我們稱之為「環狀佇列」(Circular Queue)。



環狀佇列的指令和使用方式

環狀佇列的基本動作和佇列相同，但是實作時要注意邊界問題

```
//對佇列進行輸入與輸出的動作
enqueue() //將資料加入佇列尾端。
dequeue() //將資料由佇列的前端取出
```

```
//判斷佇列是否已經填滿或是空佇列  
isFull()    //檢查佇列是否已滿，以便判斷是否還可以存入資料。  
isEmpty()   //檢查佇列是否是空的，以便判斷是否還有資料可以取出。
```

enqueue()

步驟 1：判斷佇列是否已滿

步驟 2： $\text{rear} = (\text{rear} + 1) \bmod n$

步驟 3：將值存入尾端指標 rear 所指的陣列元素

dequeue()

步驟 1：判斷佇列是否為空

步驟 2： $\text{front} = (\text{front} + 1) \bmod n$

步驟 3：取出前端指標 front 所指的陣列元素

isFull ()

步驟 1：判斷 $\text{rear} \bmod n$ 和 $(\text{front} + 1) \bmod n$ 是否相等

步驟 2：若相等，表示佇列已滿

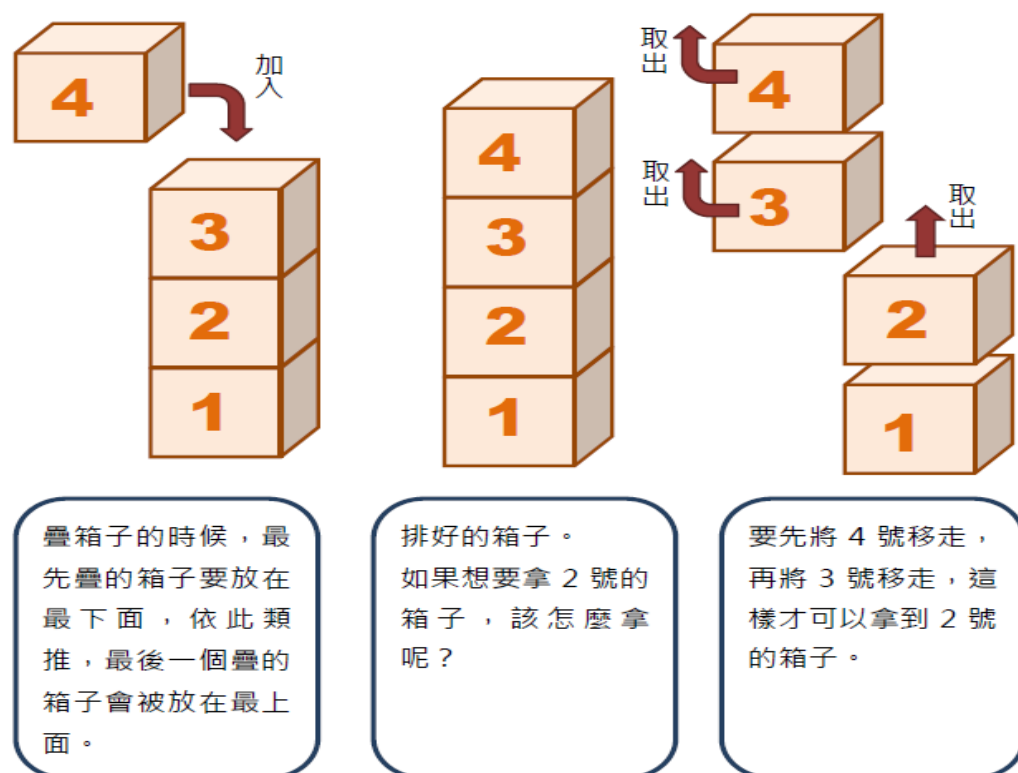
isEmpty ()

步驟 1：判斷 $\text{rear} \bmod n$ 和 $\text{front} \bmod n$ 是否相等

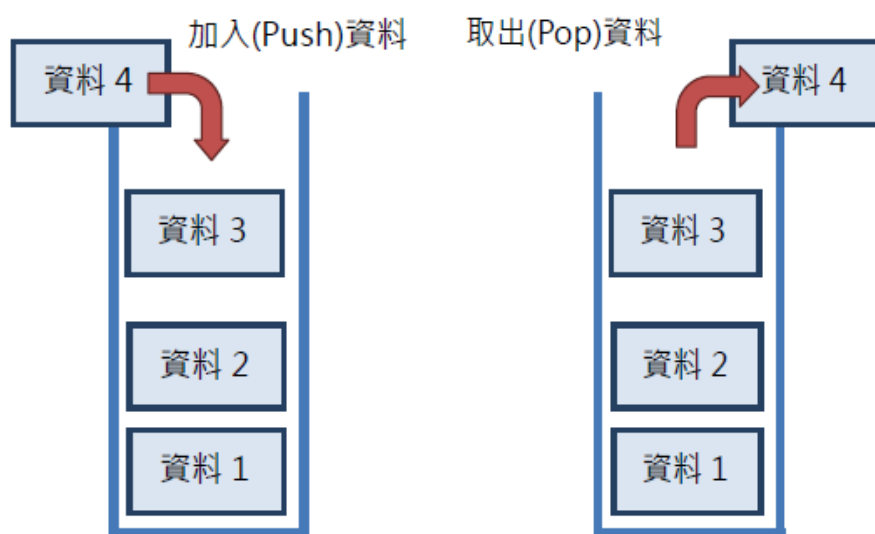
步驟 2：若相等，表示佇列為空

7-2 堆疊(Stack)

一種資料結構。日常我們常把書本裝箱一箱一箱疊起來時，如果想要拿下面某一個箱子的書，必須把上面的箱子一個個搬走以後，才能拿到我們想要的箱子。這種依照順序，**最先進入隊伍的，卻是最後一個離開的資料結構**，稱為**堆疊(Stack)**。而這種資料存取的順序，我們在資料結構稱為**先進後出 (First In Last Out , FILO)**。



由上圖可知，堆疊的概念就是從上方加入(Push)資料，從上方取出(Pop)資料，且 Push 和 Pop 的動作皆發生在同一端。

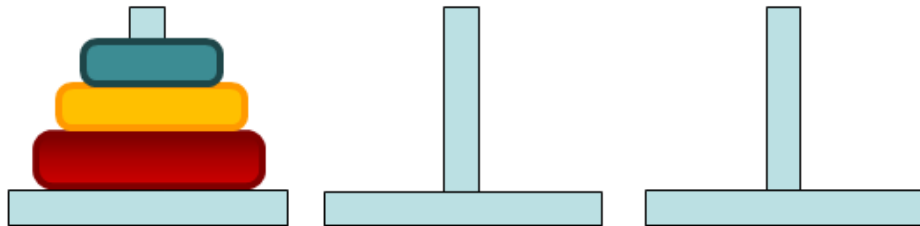


堆疊的特性

我們可以從上面堆疊的基本原理發現，堆疊只有一個開口，其底端為封閉的。而且堆疊的空間有限，如果堆疊滿了，則資料將無法再加入。放入或取出資料，只能從最上方的位置開始進行。

堆疊的應用

河內塔，是堆疊最有名的應用之一。其規則為一次僅可搬移一個盤子，小盤子一定要放在大盤子的上面，搬移的盤子須放至另外兩個柱子之一。



另一個常見的應用則是用在**程序與執行緒**上，支援多執行序的作業系統中，每一個執行中的程式可以有一個以上的執行緒(thread)，每一個執行緒中 CPU 都獨立執行程式，不互相干擾，每一個執行緒擁有自己的堆疊來輔助其正確地執行函式。

堆疊的指令和使用方式

堆疊的基本動作可分為下面四種(和佇列類似)

```
//對堆疊進行輸入與輸出的動作
push()    //將資料加入堆疊。
pop()     //將資料由堆疊的頂端(top)取出。

//判斷堆疊是否已經填滿或是空佇列
isFull()   //檢查堆疊是否已滿，以便判斷是否還可以存入資料。
isEmpty()  //檢查堆疊是否是空的，以便判斷是否還有資料可以取出。
```

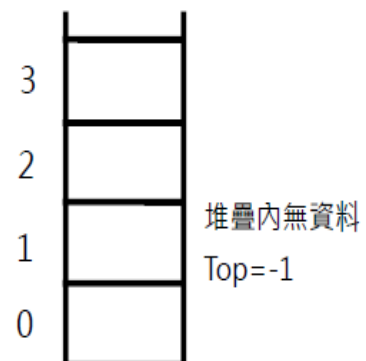
push()

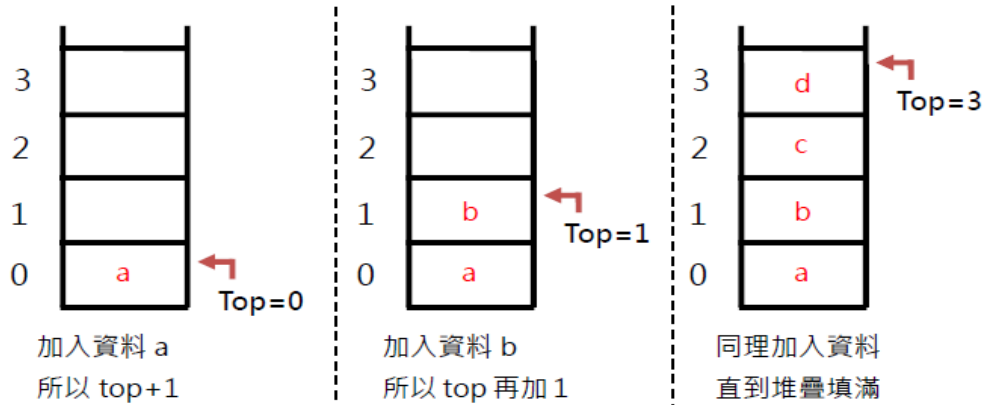
假設頂端(top)初始值為-1

步驟 1：判斷堆疊是否已滿

步驟 2：將 top 往上移動，即將 top 加 1。

步驟 3：將值存入頂端指標 top 所指的陣列元素。





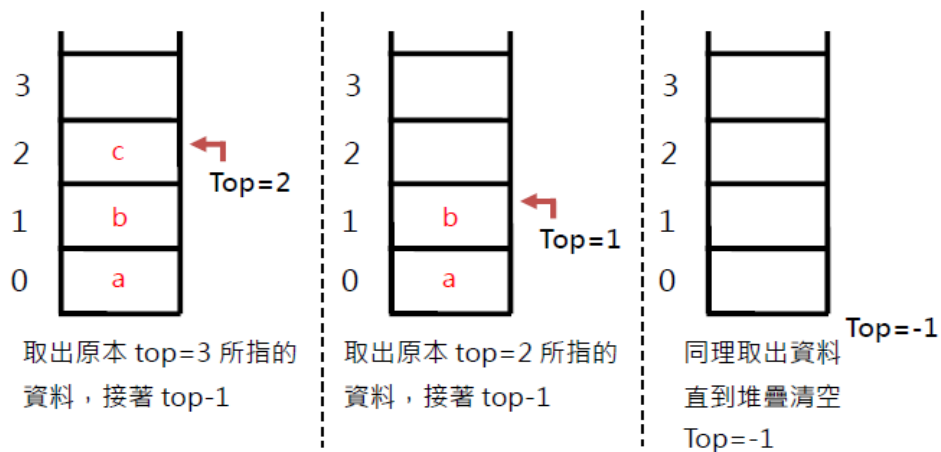
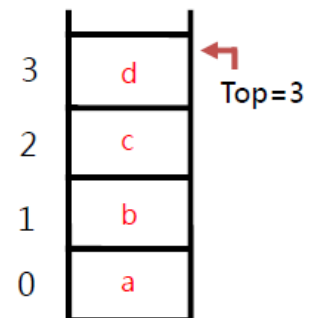
pop()

假設堆疊大小為 4，頂端 top 的值為 3

步驟 1：判斷堆疊是否是空的(空堆疊無法取出資料)。

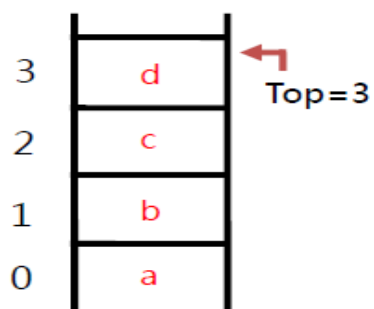
步驟 2：取出頂端指標 top 所指的陣列元素。

步驟 3：將頂端指標 top 往下移動，即將 top-1。



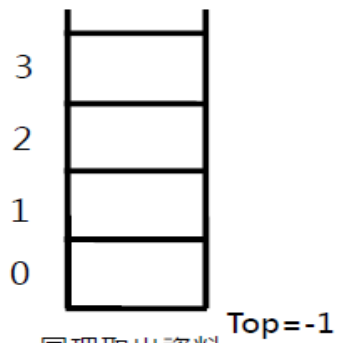
isFull ()

每當想加入一筆新資料的時候，要先判斷堆疊是否為滿。是否為滿，判斷 top 是否在堆疊的頂端(即 top=陣列容量-1)若 top=陣列容量-1，則堆疊已滿。



isEmpty ()

判斷是否還有資料可以取出。top=-1 時，堆疊為空。



同理取出資料

直到堆疊清空

Top=-1

7-3 串列(List)與鏈結串列(Linked list)

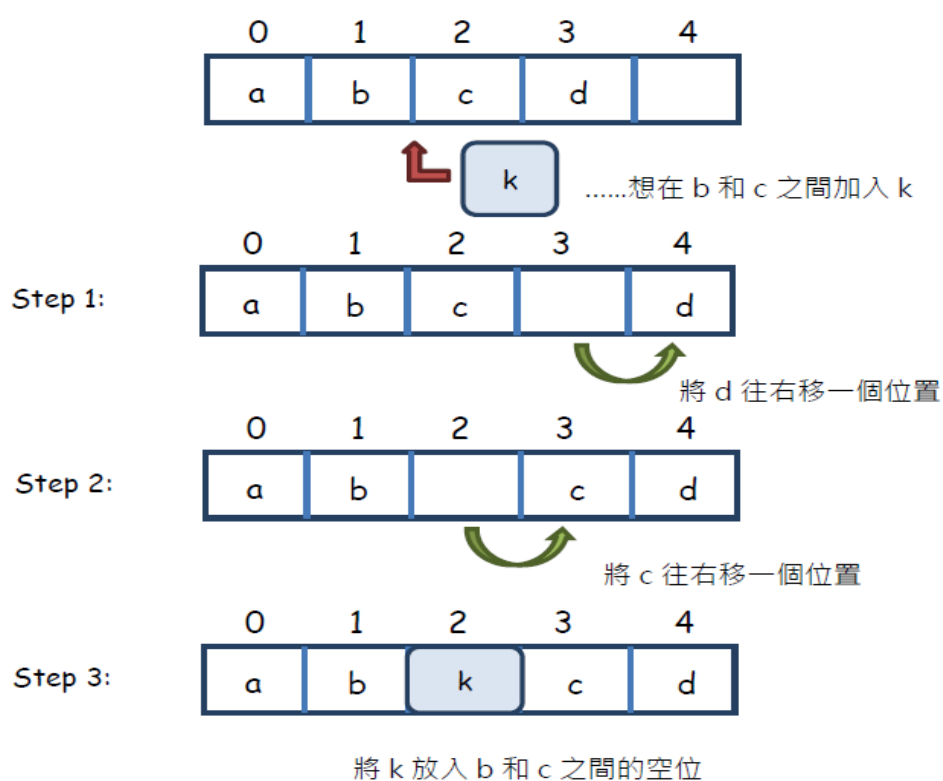
串列是一種有順序性的資料結構，它的元素以某種順序排列，該順序具有一定的意義且不可交換。依照串列存放於記憶體中的方式，分為兩種。

循序串列(sequential list)：由一塊連續的記憶體存放串列元素，如：陣列。

鏈結串列(linked list)：利用指標將不連續的記憶體鏈結起來，存放串列元素。

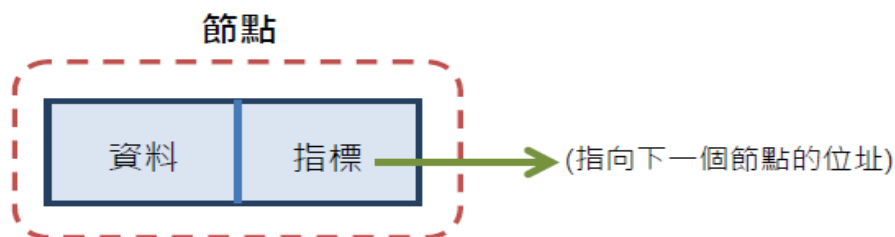
循序串列

知道索引值，很容易存取或修改串列內元素。加入或删除某筆資料時，會造成資料頻繁的移動，導致程式執行效率降低。

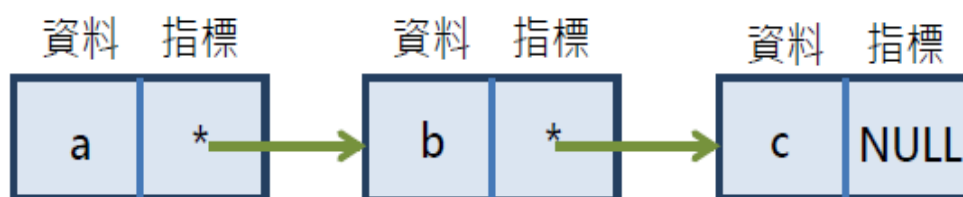


鏈結串列

鏈結串列是由節點(node)串接而成。記憶體空間由動態配置的方式取得。一個節點包含兩個成員，該節點所儲存的資料和指向下一個節點的指標。鏈結串列的節點在記憶體中的位置並不連續，因此當要存取或修改節點時，必須從第一個節點開始，依照指標向後尋找，因此在操作上比較費時。如果想在鏈結串列內加入或删除某筆資料時，只需更改指標的指向，不必造成資料頻繁的移動。



鏈結串列的每個節點的指標都會指向下一個節點。第一個節點稱之為首節點，最後一個節點稱為終端節點，其指標將設為 NULL，表示其後面已無其他節點。



鏈結串列的表示方法

剛剛有提到鏈結串列的每個節點有兩個成員：該節點所儲存的資料和指向下一個節點的指標。這表示有兩種不同的資料型態，所以須利用結構來設計節點。

```
struct node
{
    資料型態 data;
    struct node *next;
}
```

data 為節點所儲存的資料，依不同要求宣告不同的資料型態。next 為指向下一個節點的指標，因此其資料型態為「指向節點的指標」。

回憶一下——結構和 typedef

結構可以將許多不同資料形態的變數、陣列組合在一起，形成新的資料型態。所以定義一個結構，等於定義一個新的資料型態。typedef 無法定義新的型態，但是可以將現有的型態定義新的名稱。節省撰寫程式的時間，也可以使程式更加結構化、更容易閱讀。利用 typedef 將 struct node 定義成新的名稱 NODE，在使用時會比較容易。

動態記憶體配置和釋放

在 C 語言中，我們利用 malloc() 函式來進行動態記憶體的配置，而釋放動態記憶體則是使用 free() 函式來進行。

malloc 的用法如下：

1. malloc 為 memory(記憶體)和 allocation(分配)的縮寫。
2. malloc 定義於標頭檔 stdlib.h 中，因此當要使用此函式時，需要 include 此標頭檔。
3. malloc 會傳回被配置的記憶體位址，因此需要一個指標變數來接收。指標變數的型態有許多種，因此我們需將 malloc 傳回的位址進行型態轉換。
4. 若記憶體配置失敗，malloc 會傳回 NULL。
5. 使用完配置的記憶體要記得釋放，若未能釋放已經不再使用的記憶體，會造成記憶體洩漏(memory leak)。

不同的編譯程式會用不同的位元組存放不同的資料型態，因此可以利用 sizeof() 函數來得到所需資料型態所佔的位元組。

free 的用法如下：

1. free 定義於標頭檔 stdlib.h 中，因此當要使用此函式時，需要 include 此標頭檔。
2. 透過 malloc 配置的記憶體位址，需要透過 free 才能歸還給系統，否則會造成記憶體洩漏 (memory leak)。
1. 若使用已歸還的記憶體空間，則會發生記憶體空間分割失敗 (segmentation fault) 的錯誤。

《範例》

```
//宣告一整數指標 ptr，利用 sizeof 得知整數所需的記憶體空間，利用 malloc  
//配置一塊記憶體空間給 ptr。  
  
int *Ptr;  
Ptr = (int*)malloc(sizeof(int));  
  
//釋放記憶體  
  
free(Ptr);
```

存取鏈結串列的資料

《範例》

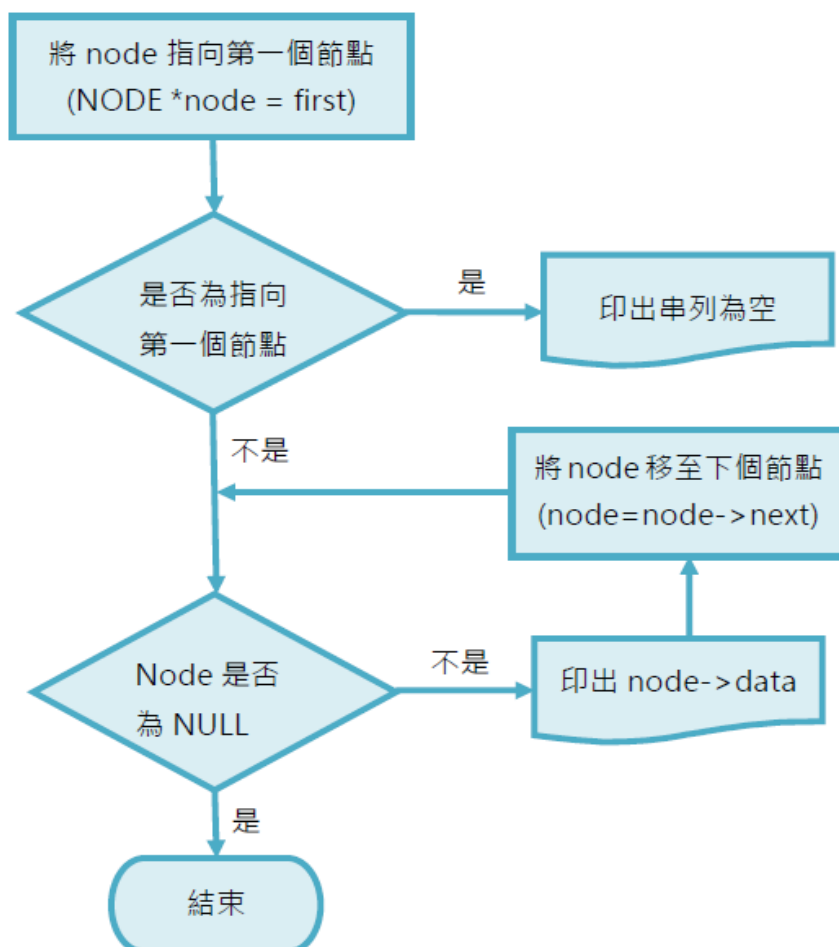
```
//建立一個鏈結串列資料型態
typedef struct node
{
    int data;
    struct node *next;
}

//宣告 NODE 型態的變數，利用「.」存取資料
node a;
a.data = 12;

//宣告指向NODE型態的指標，利用「->」存取資料。
node *ptr = &a;
ptr->data = 12;
```

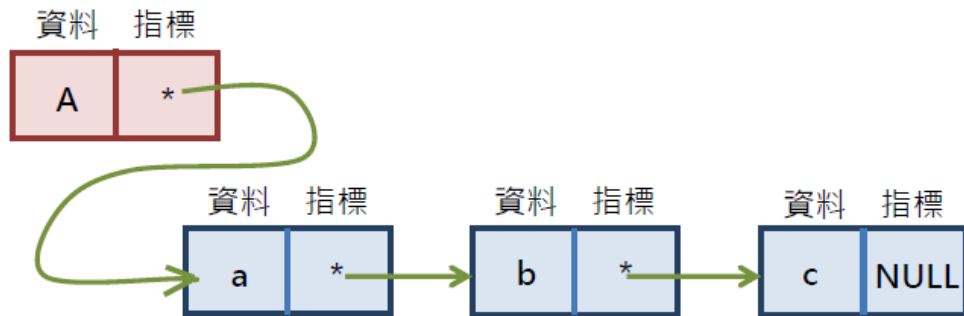
列印鏈結串列

欲將鏈結串列的每個節點印出，表示需要藉由鏈結串列的方向，經過一個個節點並將經過的節點印出，即列印鏈結串列。

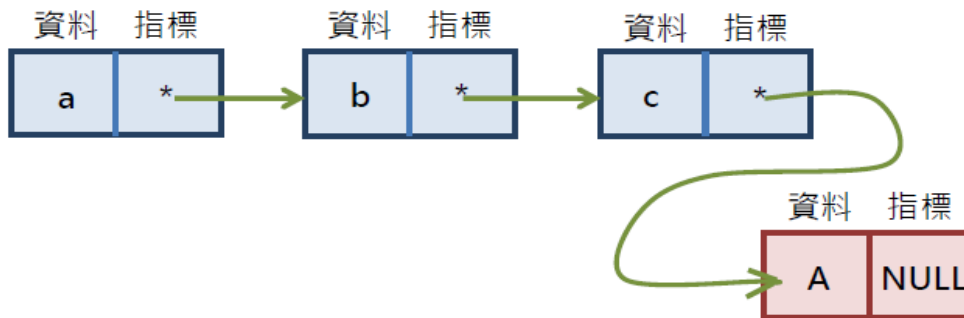


鏈結串列新增和刪除節點

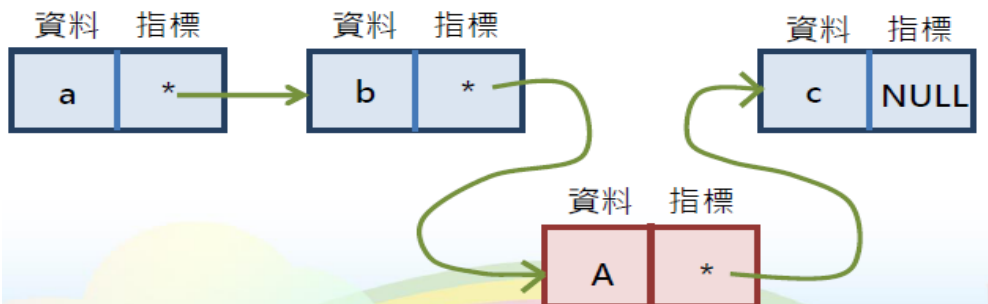
於串列最前端加入新節點的作法是將新節點的指標指向原本第一個節點的位址。



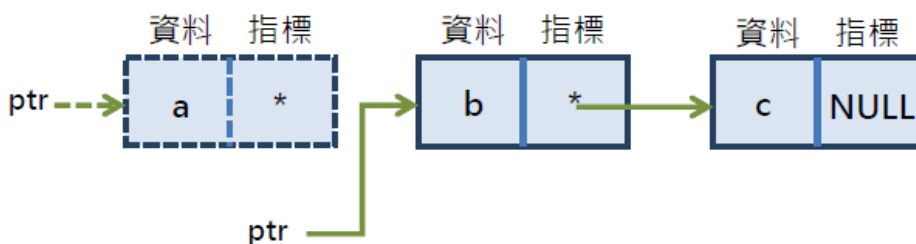
於串列最尾端加入新節點的作法是將原本最尾端的節點指標指向新節點的位址。最後將新節點的指標設為 NULL。



於串列中間加入新節點的方法是設置新節點的指標指向舊節點的指標指向節點。再設置舊節點的指標指向新節點的位址。



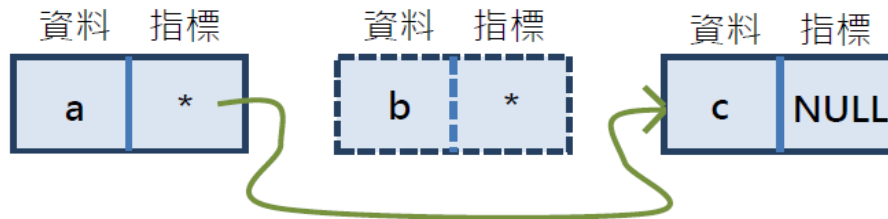
刪除串列最前端的節點的方法是把指向第一個節點的指標指向下一個節點後，再刪除第一個節點。



刪除串列最尾端的節點的方法是將倒數第二個節點的指標設為 NULL，再刪除最後一個節點。



刪除串列中間的節點的方法是設置欲刪除節點的前一個節點，將其指標指向欲刪除節點的下一個節點位址，再刪除欲刪除的節點。



鏈結串列與陣列的比較

| | 鏈結串列 | 陣列 |
|----|--|---|
| 優點 | <ol style="list-style-type: none"> 1. 可使用動態記憶體配置，減少記憶體的浪費。 2. 加入或刪除元素不會造成資料頻繁的移動，導致程式執行效率降低。 | <ol style="list-style-type: none"> 1. 製作容易。 2. 具索引值，存取容易。 |
| 缺點 | <ol style="list-style-type: none"> 1. 製作不易，需使用指標完成。 2. 無索引值，存取不易，需經過其他節點才能存取目標元素。 | <ol style="list-style-type: none"> 1. 宣告時已決定陣列空間，無法增加或減少。空間被限制而未使用到的空間又浪費。 2. 加入或刪除元素所造成的資料頻繁移動，導致程式執行效率降低。 |

Chapter 08

演算法

8-1 排序演算法(Sorting Algorithm)

在日常生活中，我們常常需要從一大堆雜亂無章的資料中尋找一筆資料時，我們首先會將資料做整理，按照一定的規則和順序排好。而這樣的動作在電腦科學裡，我們就稱為排序，排序的方法就稱為排序演算法。

排序演算法是一種能將一串資料依照特定排序方式的一種演算法。最常用到的排序方式是數值順序以及字典順序。有效的排序演算法在一些演算法（例如搜尋演算法與合併演算法）中是重要的，如此這些演算法才能得到正確解答。排序演算法也用在處理文字資料以及產生人類可讀的輸出結果。

基本上，排序演算法的輸出必須遵守兩個原則。

1. 輸出結果為遞增序列（遞增是針對所需的排序順序而言）。
2. 輸出結果是原輸入的一種排列、或是重組。

內部排序(Internal sort)和外部排序(External sort)

內部排序又稱「陣列排序」，其排序的工作，主要是在主記憶體完成，適用於資料量較小的情況。

外部排序又稱「檔案排序」，其排序的工作，主要是在輔助記憶體完成，適用於資料量較大的情況。

排序的分類方式 - 穩定和不穩定排序

排序的資料中，有值相同之資料，在排序後相對位置與排序前仍相同時，稱穩定排序。

8、9、42、58、8、5、6、7

排序後得：

5、6、7、8、8、9、42、58

紅色 8 仍排在藍色 8 之前方，則稱之為穩定排序

排序的資料中，有值相同之資料，在排序後相對位置與排序前不相同時，稱不穩定排序。

8、9、42、58、8、5、6、7

排序後得：

5、6、7、8、8、9、42、58

藍色 8 排在紅色 8 之前方，相對位置不同，我們稱之不穩定排序。

時間複雜度(Time Complexity)和空間複雜度(Space Complexity)

時間複雜度(Time Complexity)是指計算執行程式所花費的時間。依照不同的程度，可以分為：

最佳情況(Best Case)

資料已完成排序。如：原本數列已經完成由小至大的排序，對此數列再進行由小而大的排序時，所使用的時間複雜度就是最佳情況，以 $BIG\ \Omega$ 來表示。

最壞情況(Worst Case)

需要對數列的每一個元素進行重新排列。如：原本數列已經完成由小至大的排序，但卻對此數列進行由大到小的排序，因此需要進行重新排列，其所使用的時間複雜度就是最壞情況，以 $BIG\ O$ 來表示。

平均情況(Average Case)

以 $BIG\ \theta$ 表示，其時間介於 $BIG\ O$ 和 $BIG\ \Omega$ 之間。

空間複雜度(Space Complexity)

一個程式執行時所需要額外的記憶體空間。固定的空間需求：如程式本身的常數、靜態變數及結構等。變動空間需求：如動態記憶體配置空間。以排序法來說，所使用到額外的記憶體空間越少，空間複雜度就越佳。

泡沫排序演算法

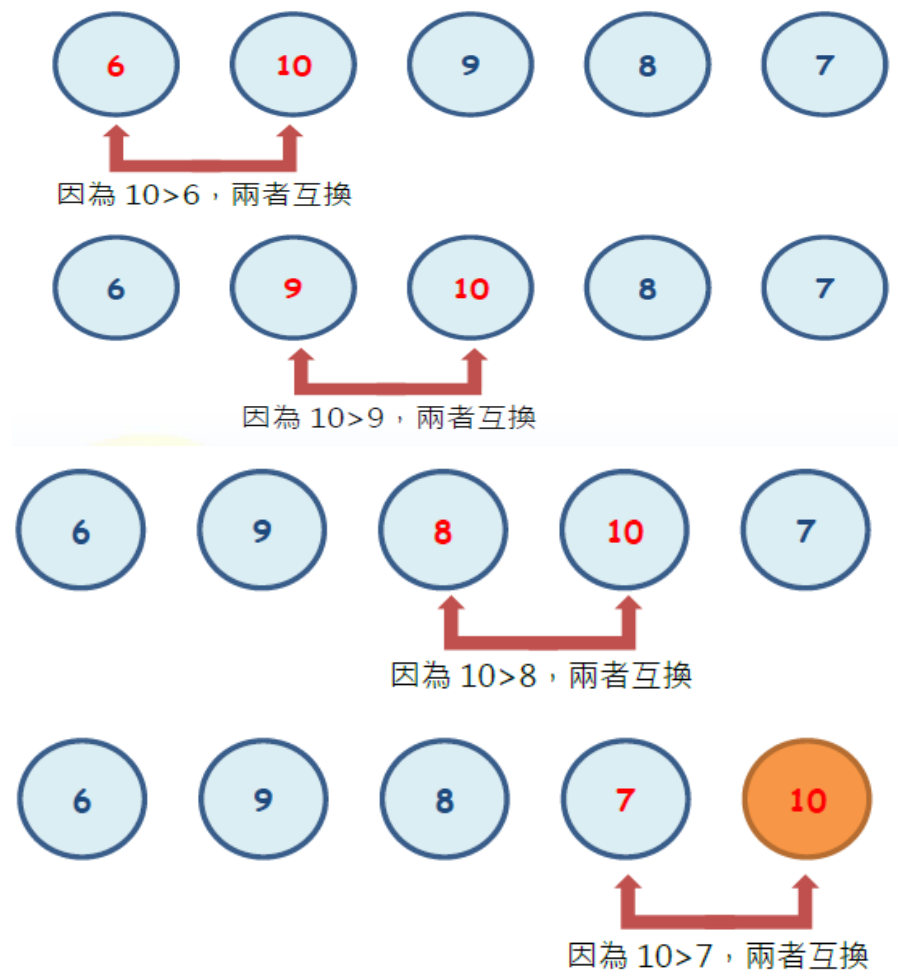
為資料結構中，最簡單的排序法。

運作原理：依序將相鄰的兩個資料互相比較，依排序的條件(由小至大或由大至小)交換位置，直到資料排好為止，而其結果如氣泡般，依次由下往上浮起，因此稱之氣泡排序法。

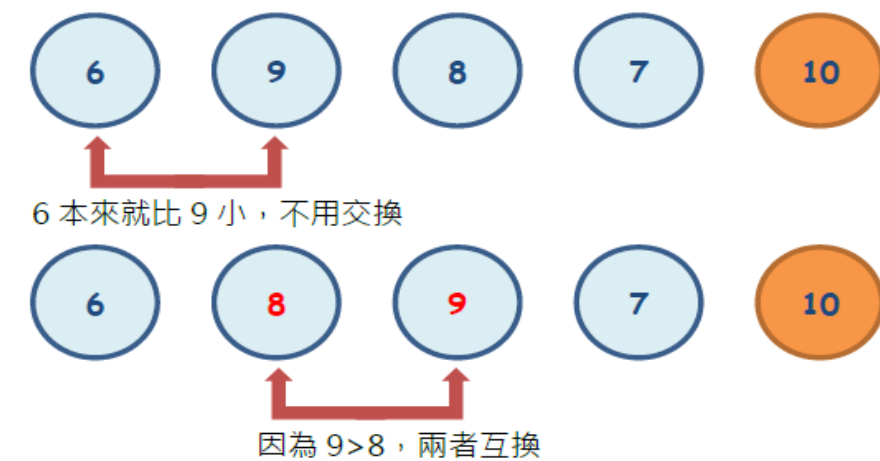
EX: 有一數列 10、6、9、8、7，請利用泡沫排序法將其由小而大依序排列。



第一次循環



第二次循環





第二次循環結束後，第二大的數字：9，此次循環一共比較 3 次。



同理，第三次循環結束後，得到第三大的數字：8，此次循環一共比較 3 次。此時此數列已排序完畢。雖然數列已在第三次循環時排序完畢，但是在程式中，此排序法仍會繼續，直到第四次循環結束，程式才結束。

我們跑過一次泡沫排序的流程後，我們可以歸納出一些泡沫排序的特性。泡沫排序是穩定排序的一種，適合用於資料量小的時候。每一次循環次數 = 資料個數 - 1。每次循環之後，至少有一個資料可以排列到正確位置。因此進行下個循環的排列時，便可以減少資料的比較。

從數學的分析角度來看。

N 個元素，利用泡沫排序法：

最壞的比較次數為：

- $(N-1) + (N-2) + \dots + 2 + 1 = \frac{N(N+1)}{2}$ 次
- 時間複雜度為 $O(N^2)$ 。

平均的比較次數為：

- $(N-1) + (N-2) + \dots + 2 + 1 = \frac{N(N+1)}{2}$ 次
- 時間複雜度為 $O(N^2)$ 。

最佳的比較次數為：

- $(N-1)$ 次
- 時間複雜度為 $O(N)$ 。

選擇排序演算法

運作原理：先在未排序序列中找到最小（大）元素，然後將其存放至序列的第一個位置，接著再從剩餘未排序元素中繼續尋找最小（大）元素，將其依序放置於第一個位置之後。依此類推，直到所有元素均排序完畢。

EX: 有一數列 10、6、9、8、7，請利用泡沫排序法將其由小而大依序排列。



第一回合：找出最小的數字：6，並和第一個位置交換



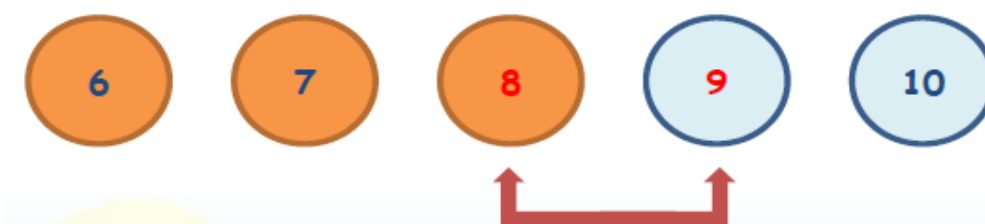
數字必須和第一個數字交換位置

第二回合：找出第二個小的數字：7，並和第二個位置交換



第二個小的數字必須和第二個數字交換位置

第三回合：找出第三個小的數字：8，並和第三個位置交換



第三個小的數字必須和第三個數字交換位置

第四回合：找出第四個小的數字：9，並和第四個位置交換



因為 9 已經在第四個位置了，所以不交換



由於前面四個數字已經排序好了，所以不需要對第五個位置進行判斷。
而且從範例我們得知，五個數字只需要比較四個回合。

我們跑過一次選擇排序的流程後，我們可以歸納出一些選擇排序的特性。

選擇排序是穩定排序的一種，也是適合用於資料量小的時候。比較的回合數 = 資料個數 - 1。需要一個額外的空間存放元素(交換時使用)，空間複雜度 $O(1)$ 。當資料量小時，排序效果優於氣泡排序演算法。

從數學的分析角度來看。

N 個元素，利用選擇排序法：

最壞的比較次數為：

- $(N-1) + (N-2) + \dots + 2 + 1 = \frac{N(N+1)}{2}$ 次
- 時間複雜度為 $O(N^2)$ 。

平均的比較次數為：

- $(N-1) + (N-2) + \dots + 2 + 1 = \frac{N(N+1)}{2}$ 次
- 時間複雜度為 $O(N^2)$ 。

最佳的比較次數為：

- $(N-1)$ 次
- 時間複雜度為 $O(N^2)$ 。

8-2 搜索演算法(Searching Algorithm)

日常生活我們常常需要在許多排好的資料中找尋資料，例如像是圖書館，而如何在特定的範圍內快速的找到我們需要的資料呢？就是靠搜索，而搜索的核心動作就是透過不停的比較資料來判斷甚麼樣的資料是我們需要的。然而重複這樣的動作是非常浪費時間的，因此越是龐大的資料越是需要有效的搜索方法和步驟來節省時間的浪費。

搜尋的分類方式

依資料量大小分成內部搜尋(Internal searching)和外部搜尋(External searching)。

內部搜尋為搜尋的資料量小，可以完全在主記憶體內進行搜尋，即從主記憶儲存資料檔中進行尋找所要的資料之操作。

外部搜尋則是搜尋的資料量大，無法直接在主記憶體內進行搜尋，必須使用輔助記憶體（如硬碟、磁帶）。

依搜尋時資料表格是否異動分成靜態搜尋(Static searching)和動態搜尋(Dynamic searching) 靜態搜尋，資料在搜尋過程中，被搜尋的資料表格不會有任何的異動如：增加、刪除、或更新等，像是查詢電話簿的資料或搜尋紙本字典，即靜態搜尋。

動態搜尋，搜尋過程中，資料表格會經常異動，如：增加、刪除、或更新等。

循序搜尋演算法

又稱線性搜尋法，是資料結構中，最簡單的搜尋法。

運作原理：從資料表中的第一個資料開始取出，依序與欲查詢的資料(稱作「鍵值」)互相比較，直到找尋到想要的元素或是所有資料均尋找完畢為止。

EX: 有一數列 73、93、13、22、67、58、49 儲存於陣列，請問資料 22 儲存於陣列的哪位置呢？利用循序搜尋演算法搜尋。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|----|----|----|----|----|----|----|
| 73 | 93 | 13 | 22 | 67 | 58 | 49 |

位置 0

| | | | | | | |
|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 73 | 93 | 13 | 22 | 67 | 58 | 49 |

\neq 22

位置 1

| | | | | | | |
|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 73 | 93 | 13 | 22 | 67 | 58 | 49 |

\neq 22

位置 2

| | | | | | | |
|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 73 | 93 | 13 | 22 | 67 | 58 | 49 |

\neq 22

位置 3

| | | | | | | |
|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 73 | 93 | 13 | 22 | 67 | 58 | 49 |

$=$ 22

由於找到鍵值 22，因此程式可以記錄其所在的陣列位置(位置 3)，並結束搜尋。但若找尋不到鍵值，則程式會繼續往位置 4、位置 5、…一直搜尋，直到搜尋全部資料仍找不到目標資料時，則顯示找不到鍵值。

循序搜尋演算法的特性

是一種外部和靜態搜尋。適合用於資料量小的搜尋。優點是程式容易撰寫和不須對資料先進行排序。缺點則是搜尋速度慢而且不管資料有無排序，都要從頭開始尋找。

從數學的分析角度來看。

N 個元素，利用選擇排序法：

如果資料沒有重覆，找到資料就可終止，否則要找到資料結束。

最壞情況：需做N次的比較，時間複雜度為 $O(N)$ 。

平均情況：假設資料出現與分佈之機率相等，需做 $\frac{N+1}{2}$ 次比較，時間複雜度為 $O(N)$ 。

最佳情況：欲查詢的資料就是第一個，只需比較1次，時間複雜度為 $O(1)$ 。

二分搜尋演算法

必須針對**已排序過**的資料進行搜尋。

運作原理：將資料從中間分成兩部份，再將鍵值與中間值比較，若鍵值相等則找到，若鍵值比中間值小，則再將前半段資料重複上述步驟，同理，若鍵值比中間值大，則再將後半段資料重複上述步驟，直到找尋到鍵值或是無資料可以搜尋為止。

EX: 有一數列 13、22、49、58、67、73、93 儲存於陣列，請問資料 22 儲存於陣列的哪位置呢？利用二分搜尋演算法搜尋。

| | | | | | | |
|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 13 | 22 | 49 | 58 | 67 | 73 | 93 |

步驟 1：將資料從中間分成兩部份，中間值為 58。

| | | | | | | |
|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 13 | 22 | 49 | 58 | 67 | 73 | 93 |

22 < 58
鍵值比中間值小，因此針對前半部重複動作

步驟 2：將前半部從中間分成兩部份，中間值為 22。

| | | | | | | |
|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 13 | 22 | 49 | 58 | 67 | 73 | 93 |

22
鍵值和中間值相同，找到鍵值的位置：1

由於找到鍵值，因此程式可以記錄其所在的陣列位置(1)，並結束搜尋。但若找尋不到鍵值，則程式會繼續往最接近鍵值的位置搜尋，直到資料無法再被切分為二分時，則顯示找不到鍵值。

二分搜尋法的特性

是一種內部和靜態搜尋的演算法。優點是搜尋速度快。但是缺點是資料需要先經過排序而且檔案資料必需使是可直接存取或隨機檔。

從數學的分析角度來看。

N 個元素，利用選擇排序法：

如果資料沒有重覆，找到資料就可終止，否則要找到資料結束。

最壞情況：比較 $\log_2 N + 1$ 或 $\log_2(N + 1)$ 次，時間複雜度為 $O(\log_2 N)$ 。

平均情況：時間複雜度為 $O(\log_2 N)$ 。

最佳情況：鍵值就是第一個中間值，只需比較1次，時間複雜度為 $O(1)$ 。

總計畫【高中資訊科學創新學習計畫】

主持人 國立臺灣師範大學資訊工程系(所) 李忠謀 教授

子計畫二【雲端合作學習輔助程式設計教學之課程研發計畫】

主持人 臺北市立內湖高級中學 吳正東 校長

計畫參與人員

國立臺灣師範大學資訊工程系 簡志峰 博士

朱德清 博士

臺北市立內湖高級中學 林朝興 主任

羅玕貞 老師

侯莉莉 老師

林瓊甄 老師

馮毓琪 老師

施竣印 老師

李官恩 老師

網站建置

國立臺灣師範大學資訊工程系 楊承嘉

顏羽君

陳紹中