第1章 導論 (INTRODUCTION)

資料結構

鍾宜玲

為甚麼要學資料結構？

計算機科學家 **維爾特** (Niklaus Wirth, 1934/2/15~)

於1975年寫了一本書，書名為

"Algorithms + Data Structures = Programs"

(演算法 + 資料結構 = 程式)

程式設計師必須有資料結構的基礎，才能撰寫出優良的程式。

資料結構 (DATA STRUCTURE)的定義

- 計算機系統中**資料**的「組織方式」及「存取運算方法」
- 探討如何將資料有系統地安排組織配合適當的演算法以達到最佳化的處理結果。
- 設計資料結構時主要考量：
  1. 如何**產製** (create) 儲存結構
  2. 如何**儲存** (store)資料
  3. 如何**取出** (retrieve) 資料
  4. 有哪些**運算** (operator)

資料結構

10	50	60

60

演算法(ALGORITHM)的定義

- **在有限步驟內解決問題的方法或程序。**
- Donald E. Knuth(高德納) 提出五個基本要素：
  1. **輸入指令 (input)**：零個或一個以上的輸入資料。
  2. **輸出指令 (output)**：一個或一個以上的結果輸出。
  3. **明確性 (definiteness)**：指令必須明確不混淆。
  4. **有限性 (finiteness)**：在有限步驟內執行結束。
  5. **有效性 (effectiveness)**：又稱可行性。

描述演算法的方法

- 使用自然語言 (中文、英文......等)
- 數學式
- 程式語言
- 流程圖 (flowchart)
- 虛擬碼 (pseudo-code)

演算法範例
-- 以自然語言描述

求兩實數相減之絕對值

解：

(1) 輸入兩個實數 a 與 b 的值。

(2) 設定實數 c 值為 a – b。

(3) 如果 (c < 0) 成立，
則實數 c 設定為 – c。

(4) 輸出實數 c 的值。

演算法範例
-- 以流程圖描述

求兩實數相減之絕對值

解：

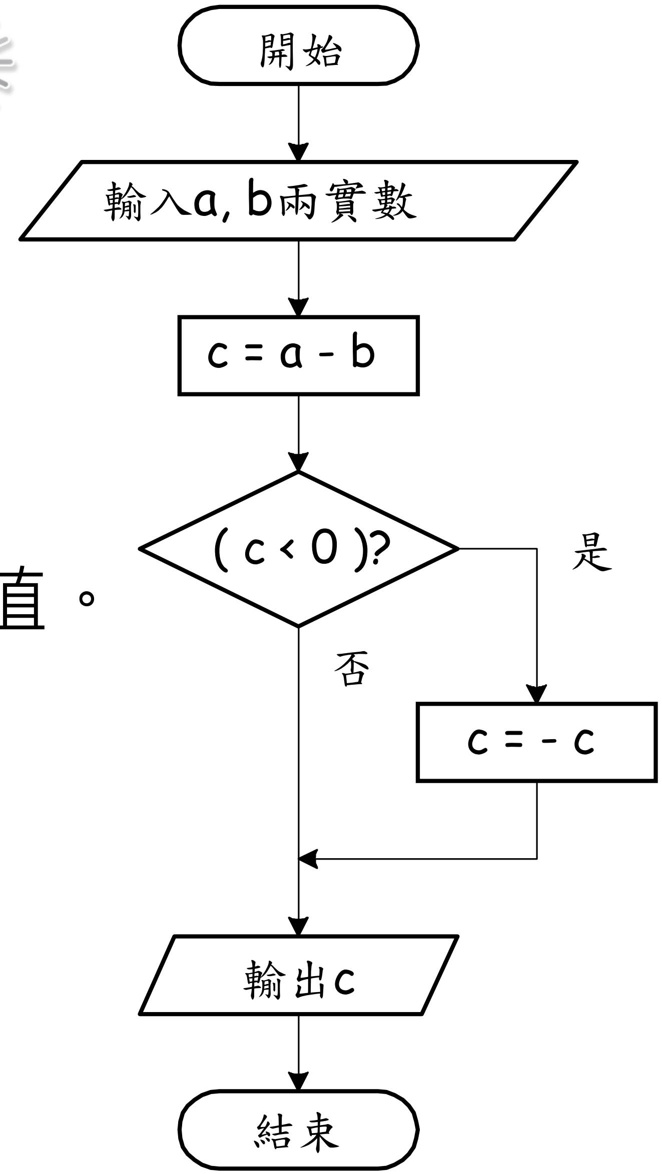
(1) 輸入兩個實數 a 與 b 的值。

(2) 設定實數 c 值為 a – b。

(3) 如果 (c < 0) 成立，

則實數 c 設定為 – c。

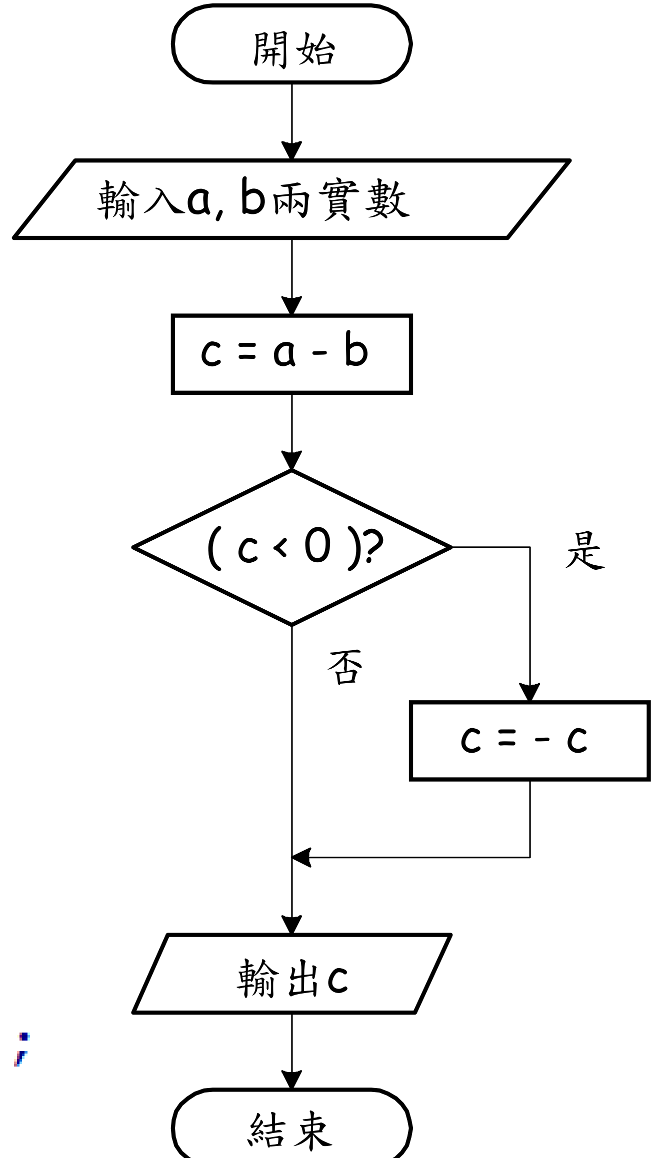
(4) 輸出實數 c 的值。



演算法範例
--以 C 語言描述

求兩實數相減之絕對值

```c
void main( )
{
    double  a, b, c;
    printf("輸入兩實數：");
    scanf("%lf%lf", &a, &b);
    c = a - b;
    if( c < 0 )
        c = -c;
    printf("絕對值為%f\n",c);
    return;
}
```



程式的效能分析

好程式的條件

- 執行結果正確
- 可維護性高
- 執行效率高

電腦執行結果要對！

程式碼容易看懂！

執行時間要短、
記憶體用得少！

1) 執行時間(run time)的長短：簡略以程式執行的敘述多寡(頻率計數)來測量
2) 儲存變數資料所需的記憶體空間大小

頻率計數(FREQUENCY COUNT)

- 可執行的敘述才會影響程式的執行時間
- **頻率計數**
  1. 計算程式敘述被執行的總次數
  2. 用來評估程式的執行時間，以判斷演算法的優劣。

範例

計算 N 位學生的總平均分數

程　　式	執行次數
`float avg(float score[], int n)`	
`{`	
`    int i;`	
`    float sum, average;`	
`    if( n <= 0)`	1
`        average = 0;`	1
`    else{`	
`        sum = 0;`	1
`        for(i=0; i<n; i++)`	n+1
`            sum += score[i];`	n
`        average = sum / n;`	1
`    }`	
`    return average;`	1
`}`	
執行總次數	2n+6

BIG-O 函數

- 程式的效率以頻率計數函數的級數 (order)分級描述，稱為**時間複雜度 (time complexity)**。
- 時間複雜度分為

  Ω ：最佳時間複雜度　不可靠！

  Θ ：平均時間複雜度　不易計算！

  O ：最壞時間複雜度　容易計算，品質保證！

- 一般皆以 O 符號來表示時間複雜度，讀作 “Big-oh” 。

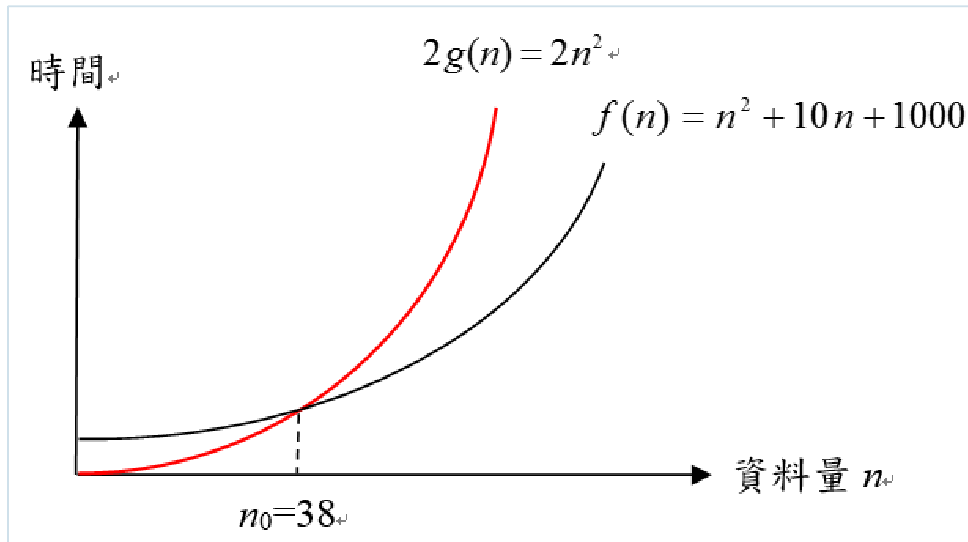
範例

考慮以下兩個頻率計數函數為：

$$f(n) = n^2 + 10n + 1000$$

$$g(n) = n^2$$

- 當資料量 $n > n_0 = 38$ 時，$f(n)$ 的執行時間永遠比 $2g(n)$ 好。
- $2g(n)$ 是 $f(n)$ 的上界。
- $f(n)$ 的最壞時間複雜度為 $O(g(n)) = O(n^2)$



頻率計數與時間複雜度

- 時間複雜度的計算
  1. 只取頻率計數函數的最高次項。
  2. 不計係數。
- 範例：若程式敘述之執行次數為 $3n+5$
  1. 則以$O(n)$ 表示，讀作 “Big oh of n ”。
  2. 程式執行時間之成長速率與資料量 n 成正比。
  3. 當資料量變成10倍時，執行時間大約也是10倍的時間。

時間複雜度(TIME COMPLEXITY)

頻率計數	時間複雜度
120	$O(1)$
$2\log n+5$	$O(\log n)$
$3n+100$	$O(n)$
$2n\log n+3n$	$O(n\log n)$
$3n^2+5n+8$	$O(n^2)$
$n^2 *(n-1)/2$	$O(n^3)$
$3*2^n + 5n^3 +7$	$O(2^n)$
$n! + 2n^5 + 7$	$O(n!)$

優劣順序

優

劣

時間複雜度排序

常見的複雜度其大小排序為

$$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$