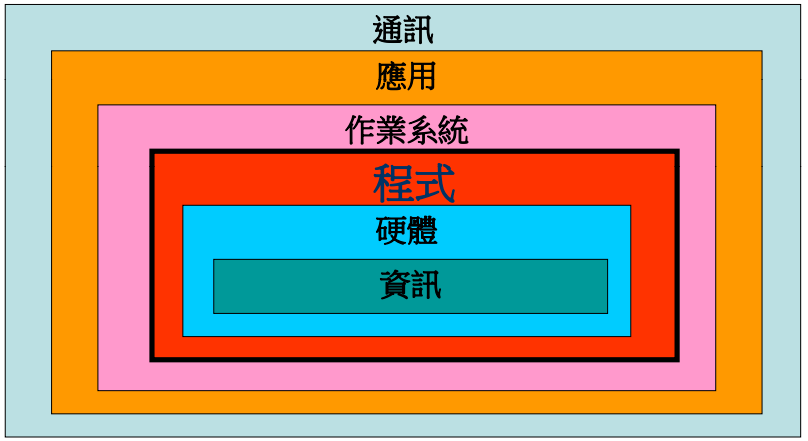
Chapter 6 問題解決與演算法設計

- 目標----研讀完本章後，你應該可以：
  - 判斷某個問題是否適合用電腦來解。
  - 描述電腦問題解決過程以及將它與波亞 (Polya) 的「如何解決它表單」(How to Solve It List) 之間建立關聯性。
  - 區分遵循(follow)演算法以及開發(develop)演算法。
  - 描述用來表示演算法的虛擬碼結構。
  - 使用虛擬碼來表示演算法。
  - 應用由上而下設計方法來開發演算法以便用來解決問題。
  - 定義物件導向設計的關鍵名詞。
  - 應用物件導向設計方法來開發一組互動物件以便用來解決問題。
  - 討論下列有關問題解決的串連思路 (thread)：資訊隱藏(information hiding)、抽象概念(abstraction)、為事物及事件命名(naming things and events)以及測試(testing)。

電腦系統的階層 (複習)



6.1 問題解決

- 問題是一種錯綜複雜且尚未解決的難題。
- 而解決則被定義成為某件事 (比如問題) 尋找解答。
- 問題解決 (problem solving) 是一種尋找面對使人困惑、煩惱、厭煩或尚未解決等問題解答的動作。
- 電腦是不具備智慧的，它無法分析問題以及提供解答。
- 人類(**程式規劃師**) 必須分析問題，開發解決問題的指令 (**程式**)，然後讓電腦完成這些指令。

如何解決問題

- Polya 於 1945 年寫了一本名為《如何解決它：數學方法的新觀念》的書，提出以下 “如何解決問題”表單：
  1. 了解問題：已知資料有哪些、未知數是什麼、相關條件為何
  2. 規劃計劃：找出資料與未知數 [即問題] 間的關聯性
  3. 完成解題計畫：核對每一步驟
  4. 檢查你所獲得的解答：能夠驗證結果嗎？

- 我們可以用**問題**（*problem*）來取代**未知數**（*unknown*），用**資訊**（*information*）來取代**資料**（*data*），用**解答**（*solution*）來取代**理論**(*theorem*)，這樣便可將 Polya 表單應用於任何類型的問題上。

如何解決問題

- Polya 表單所建議的幾種策略
  - 提出問題(注意：要會區別 problem 與 question)，直到你很清楚你要做的工作內容為止，通常會問 5 個 W、1 個 H (When, Why, Where, Who, What, How)
    - ✓ 關於這個問題我知道什麼？
    - ✓ 為了找到其解答，我必須處理的資訊是什麼？
    - ✓ 解答應該長什麼樣子？
    - ✓ 有那些特殊情況？
    - ✓ 我要如何判斷已找到解答？

如何解決問題

- Polya 表單所建議的幾種策略
  - 尋找熟悉的事物：找到類似問題的解決方法，加以套用
    - ✓ 你應該不要再重新設計一台腳踏車，如果解答存在，直接使用它
    - ✓ 某些問題會以不同的面貌一再呈現
    - ✓ 一位好的程式設計師看一個工作（或是工作的一部份，即子工作）時，先看它之前是否已經有解答，如果是，則直接插入該解答即可
  - 各個擊破 (divide and conquer)：將大問題細分成可以個別解決的小問題
    - ✓ 應用了抽象的概念
    - ✓ 各個擊破的方法可以重覆被應用(於分出之較小問題單位)，一直到每個子工作都可以被處理為止

- "如何解決問題" 表單可應用範圍很廣泛
  - 原來是用來解決數學問題
  - 此表單變成可應用於所有類型的問題
  - 提供逐步程序來解決特殊問題並非是一直重複將問題各個擊破的動作。它通常是必須經過許多嘗試與細分的嘗試錯誤過程（trial-and-error process）。

6.2 演算法

- **演算法** (*algorithm*)
  - 用在計算領域的解決問題計畫
  - 是以有限時間用有限資料量來解決問題或子問題的一組明確、不含糊的指令。
- 在計算領域中，電腦上的問題解決過程有四個階段：**分析與定義階段** (*analysis and specification phase*)、**演算法開發階段** (*algorithm development phase*)、**實作階段** (*implementation phase*) 以及**維護階段** (*maintenance phase*)。

電腦上的問題解決過程

分析與定義階段	
分析	了解 (定義) 問題
定義	指定程式要去解決的問題
演算法開發階段	
提出演算法	開發一種可以用來解決問題的邏輯步驟順序
測試演算法	依循所勾勒的大綱來看看這個解答是否真的可以解決問題
實作階段	
寫出程式碼	將演算法 (一般解) 翻譯成一種程式語言
測試程式碼	用電腦依循此指令執行，核對其結果並做修正直到答案正確為止
維護階段	
使用程式	使用此程式
維護程式	修改此程式以符合一系列需求或修正任何錯誤

圖 6.2 電腦的問題解決過程

電腦上的問題解決過程

- 圖6.3顯示這些階段間如何互相影響的關聯。
  - 黑色線顯示通過各階段的一般流程。
  - 綠色線代表如果有問題發生，回溯到前一個階段的路徑。

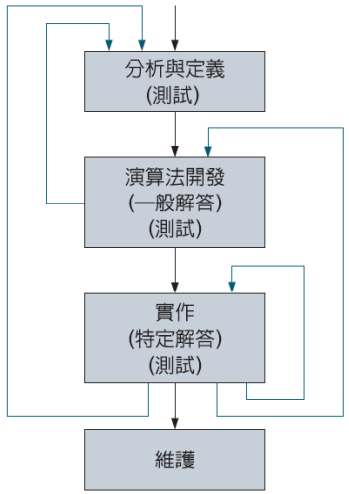


圖 6.3 四個問題解決階段之間的互動情形

電腦上的問題解決過程

- 下述演算法的表示形式稱為**虛擬碼** (*pseudocode*)，亦即利用英文 (或其他語文)與程式語言格式的混合體來產生有關問題解答的明確步驟。

範例：

```
While (商不為 0)
    十進制數字除以新基底
    讓餘數成為答案的左邊下一個數元
    以商取代原來的十進制數字
```


遵循演算法

永不失敗的混合荷蘭酸味醬

1 杯奶油	1/4 匙塔巴斯哥辣椒
4 個蛋黃	1/4 匙乾芥末
1/4 匙鹽	2 匙檸檬汁
1/4 匙糖	

把奶油加熱至冒泡後，將所有原料放入攪拌器內，在攪拌器電源打開的同時，以緩慢的速度將奶油倒入攪拌器 (含有蛋黃等原料) 內，一直到所有的奶油倒完為止。將攪拌器電源關閉，然後將上述完成的配料放置在冰箱妥善保存幾天。重新加熱時，讓它過熱但不要沸騰，之後加入雙倍的水到熱鍋內將混合物稀釋。這樣便可製成大約 1-1/4 杯的醬汁。

圖 6.4 荷蘭酸味醬 (Hollandaise sauce) 的食譜

```
如果擔心膽固醇問題
    放入奶油替代品在鍋裡
否則
    將奶油放進鍋裡
打開爐火開關
將鍋子放在爐火上
While (沒有冒泡)
    把鍋子留在爐火上
將其他原料放進攪拌器材內
打開攪拌器開關
While (還有原料)
    緩慢將原料倒進攪拌器內
關閉攪拌器開關
```

遵循演算法

- 當你遵循演算法求解時，就像是遵循食譜做菜、遵循指南玩遊戲、遵循指南組合玩具或依處方服藥一樣。
- **Repeat** (重複) 與**While** (當) 這兩個名詞在程式本文中都有其涵義，而Bubbling (冒泡) 則是廚師們所認可的一個名詞。
- **基本要素** (*primitive*) 是讓人或設備了解如何去執行演算法的~~一種項目~~ 最基礎指令，這些最基礎指令就是基本要素。

開發演算法

- 目前有兩種方法可供使用：
  - **由上而下設計** [*top-down design*，也稱為**功能分解** (*functional decomposition*)] 。
  - **物件導向設計** (*object-oriented design*, *OOD*)。

6.3 虛擬碼：演練一個虛擬演算法

While (商數不為 0)
　將十進制數除以新基底
　將餘數放在答案左邊的下一個位數
　以商數取代原有的十進制數

基底轉換演算法

(a) 初始值

十進制數	新基底	商數	餘數	答案
93	8	?	?	?

(b) 經過第一次迴圈後 (93/8)

十進制數	新基底	商數	餘數	答案
11	8	11	5	5

(c) 經過第二次迴圈後 (11/8)

十進制數	新基底	商數	餘數	答案
1	8	1	3	35

(d) 經過第三次迴圈後 (1/8)

十進制數	新基底	商數	餘數	答案
0	8	0	1	135

圖 6.5 基底轉換演算法的演練

演練一個虛擬演算法

- 在電腦的演算法中，我們稱圖 6.5 的方塊為**變數** (*variable*)，可以儲存數值也可以從中取出數值。
- 圖 6.5(a)中的最左邊方塊，十進制數，原來是放入這個問題的初始值，也就是要被轉換的數，其值在迴圈中會變改變。我們必須提供指令好讓使用者由鍵盤輸入這個數值
- 圖 6.5(a)中的左邊算起第 2 個方塊，新基底，其值不變，但也須提供指令好讓使用者由鍵盤輸入這個數值
- 圖 6.5(a)中的左邊算起第 3 個方塊，商數，要讓這個演算法正常運作，其值不可為 0，一開始必須將它設定為某個不為 0 的值

將基底轉換演算法改寫成電腦可執行的演算法

```
寫出「輸入新基底」
讀入新基底
寫出「輸入要被轉換的數字」
讀入十進制數
設定商數為 1
While (商數不為 0)
    設定商數為十進制數 DIV 新基底
    設定餘數為十進制數 REM 新基底
    將餘數放在答案左邊的下一個位數
    設定十進制數為商數
寫出「答案是」
寫出答案
```


虛擬碼功能

- 變數
  - 在虛擬碼演算法中出現的名稱，對應於儲存數值的位置 (方塊)。這個名稱必須要能反映出其內容在演算法中的角色。本書以綠色字型表示。
- 指定 (設定)
  - 如果我們有了變數，那我們必須要有一個方法來將數值放入其中。

```
設定商數為十進制數 DIV 新基底
或是
商數 ← 十進制數 DIV 新基底
```


虛擬碼功能

- 輸入 / 輸出
  - 從外部世界中輸入資料值與輸出結果到螢幕上。我們是用「寫出」(Write) 這個詞代表輸出而用「讀入」(Read) 這個詞代表輸入。

```
寫出「輸入新基底」
讀入新基底
```

  - 使用顯示 (Display) 或印出 (Print) 是等同於寫出 (Write) 的；而使用取得 (Get) 或是輸入 (Input) 與讀入 (Read) 意思是一樣的。

```
寫出「答案是」
寫出答案
```


虛擬碼功能

- 重複 (repeat)：也被稱作 迴圈(loop)、循環(iterations)
  - 重複的結構是要讓指令可以被重複使用。
  - 內層需縮排

```
設定重複次數為 0
While (重複次數 < 5)
    讀入數字
    印出數字
    設定重複次數等於重複次數 + 1
```

虛擬碼功能

- 選擇 (判斷)
  - 選擇結構可以讓你選擇要執行一個動作或是跳過不執行。

```
讀入數字
If (數字 > 0)
    印出數字
// 接著是其他敘述
```

if-then 版本

```
讀入數字
If (數字 > 0)
    印出數字
    印出「是正數」
Else
    印出數字
    印出「是負數或是 0」
```

if-then-else 版本

虛擬碼功能

表 6.1 虛擬碼敘述

結構	它的意義	所使用的字句或範例
變數	代表命名的位置可以存入值在其中，或是從中取出值	名稱是代表該值在演算法中描述問題時所代表的角色
指定	將一個值儲存到一個變數	設定數字為 1 數字 ← 1
輸入 / 輸出	輸入：讀入一個值，或許是從鍵盤 輸出：顯示一個變數或一個字串的內容，或許是從螢幕顯示	讀入數字 取得數字 寫出數字 顯示數字 寫出 "Have a good day"
重複 (循環，迴圈)	只要條件成立時就重複一個或多個敘述	While (條件) // 執行內縮編排的敘述

虛擬碼功能

結構	它的意義	所使用的字句或範例

選擇
if-then
	如果條件成立，執行內縮的敘述；如果條件不成立，跳過內縮的敘述	If (新基底 = 10)
		寫出「你正在轉換成」
		寫出「相同的基底」
		// 其他的程式碼

選擇
if-then-else
	如果條件成立，執行內縮的敘述；如果條件不成立，執行 Else (否則) 之後內縮的敘述	If (新基底 = 10)
		寫出「你正在轉換成」
		寫出「相同的基底」
		Else
		寫出「這個基底不是」
		寫出「相同的」
		// 其他的程式碼

虛擬碼範例

需求：讀入成對的正數，並依其大小次序印出每對數字

```
While (還有其他對)
    寫出「輸入兩個值以空白分開；按下 return」
    讀入數字 1
    讀入數字 2
    依次序印出它們
```

第一次嘗試

```
寫出「有多少對的數值要被輸入？」
讀入 numberOfPairs
設定 numberRead 為 0
While (numberRead < numberOfPairs)
    寫出「輸入兩個值以空白分開；按下 return」
    讀入數字 1
    讀入數字 2
    依次序印出它們
    numberRead ← numberRead +1
```

第二次嘗試

虛擬碼功能

```
寫出「有多少對的數值要被輸入？」
讀入 numberOfPairs
設定 numberRead 為 0
While (numberRead < numberOfPairs)
    寫出「輸入兩個值以空白分開；按下 return」
    讀入數字 1
    讀入數字 2
    If (數字 1 < 數字 2)
        印出 數字 1 + " " + 數字 2
    Else
        印出 數字 2 + " " + 數字 1
    遞增 numberRead
```

執行介面：

```
3
10 20
20 10
10 10
```

輸入

10 20
10 20
10 10

輸出

第三次嘗試

虛擬碼功能

(a) 一開始時

numberOfPairs	numberRead	數字 1	數字 2
3	0	?	?

(b) 第一次循環結束時

numberOfPairs	numberRead	數字 1	數字 2
3	1	10	10

(c) 第二次循環結束時

numberOfPairs	numberRead	數字 1	數字 2
3	2	20	10

(d) 第三次循環結束時

numberOfPairs	numberRead	數字 1	數字 2
3	3	10	10

圖 6.6 演練成對數字演算法

6.4 由上而下的設計方法

- 由上而下設計 (top-down design) 過程是由將問題分割成一種子問題集合開始，之後個別的子問題再分割成更多的子問題。上述過程會連續的進行，直到每個子問題足夠被定義成往後不需要再分解的形式為止。
- 我們建立了一種被稱為模組 (module) 的問題與子問題階層式結構 (hierarchical structure)，這種結構也稱為樹狀結構 (tree structure)。

由上而下的設計方法

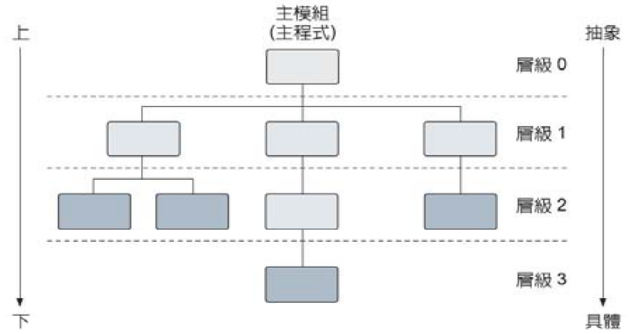


圖 6.7 由上而下設計的範例

- 這個過程持續的次數會像它用來將每個工作擴展至最小的細節部分一樣多。
- 需要擴展的步驟稱為抽象步驟 (abstract step)，而不需要擴展的步驟稱為具體步驟(concrete step)。

一個計畫大型派對的一般範例

- 計畫大型派對
  - 一種比較好的方法大概是將想要邀請的人列出來做成表單，然後檢查這個表單，看看我們遺忘了哪些最要好的朋友。
  - 一個一個地填上電話號碼，開始撥電話，並記下留言以及答覆內容。
  - 當我們連絡的人數到達某個數目時，便可以開始準備食物了。

計畫大型派對的細分範例

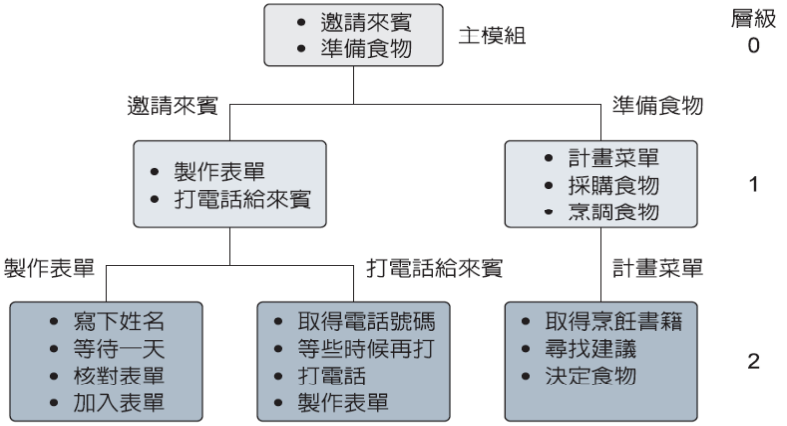


圖 6.8 細分派對計畫

計畫大型派對的變形範例

如果不確定“寫下姓名”怎麼做，可以再往下增加一個如下階層：

```
你有紙嗎？
    沒有的話，去拿紙來。
你有筆嗎？
    沒有的話，去拿筆來。
拿起筆來。
把筆放在紙上寫。
……等。
```

如果不打算自己烹飪， 主模組可以改成：

```
邀請來賓
叫熟食店
```


一個列印通訊錄的電腦範例

需求：建立包含姓名、地址、電話號碼以及電子郵件信箱的地址表單 ，這個表單應該於稍後列印出來，而即將要加入表單的名稱是根據便條紙以及名片而來

```
主模組                                  層級 0
將名字與號碼輸入表單
將表單依照字母的順序排列
列印表單
```

```
將名字與號碼輸入表單                    層級 1
While (還有名字)
    輸入名字
    輸入電話號碼
    輸入電子郵件位址
    將資訊插入表單
```

一個電腦範例

將名字與號碼輸入表單 (改寫) 層級 1

```
設定還有名字為真
While (還有名字)
    提示字句並輸入名字
    提示字句並輸入電話號碼
    提示字句並輸入電子郵件位址
    將資訊插入表單
    寫出「輸入 1 來繼續或是輸入 0 來結束」
    讀入回應值
    If (回應值 = 0)
        設定還有名字為偽
```

提示字句並輸入名字 層級 2

```
寫出「輸入名字；按下 return」
讀入名字
寫出「輸入姓氏，按下 return」
讀入姓氏
```


一個電腦範例

提示字句並輸入電話號碼 層級 2

```
寫出「輸入區域號碼及七位數字號碼；按下 return」
輸入電話號碼
```

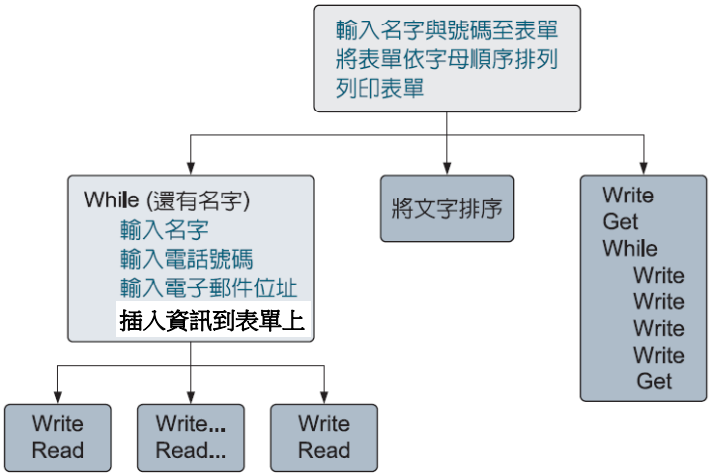
提示字句並輸入電子郵件位址 層級 2

```
寫出「輸入 e-mail 位址；按下 return」
輸入電子郵件位址
```

列印表單

```
Write「含有姓名、電話號碼以及電子郵件位址的表單如下：」
從表單取得一個項目
While (還有項目)
    Write 項目的名字 + " " + 姓氏
    Write 項目的電話號碼
    Write 項目的電子郵件位址
    Write 一行空白列
    從表單取得一個名稱
```


一個電腦範例



方法的總結

1. 分析問題
2. 寫出主模組
   - 不要太粗糙，但也不可太繁瑣
3. 寫出剩餘部分的模組
   - 將每一個模組持續加以細分，直到每個陳述句都是具體步驟為止
4. 如果有必要，重新排好順序並做修改
   - 演算法如有改變，不要害怕，重新再來一遍

測試演算法

- 桌面核對 (desk checking)
  - 在設計過程中我們訂定的演算法就必須在實作它們之前要被測試的過程
- 演練法 (walk-through)
  - 是一種由團隊成員來共同規劃設計的手工模擬方法
  - 以取樣所得的資料來針對設計實施模擬
- 檢驗法 (inspection)
  - 程式設計是事先分派工作給團隊中的成員，而當其他人指出有錯誤時，某人 (不是該程式原設計者) 須逐行檢視這個程式設計。

6.5 物件導向的設計方法

- 物件導向設計是一種問題解決的方法，它是以一種稱為**物件** (*object*) 的自含實體 (self-contained entity，意思是由該物件所提供的介面清楚明白，足夠讓外界知道該物件要完成的工作) 來產生問題的解答。
- 將資料與操作資料的演算法綁在一起即為物件導向的基本觀點。它讓每個物件對自己的操作方式 (行為) 負責。物件導向設計的內在是類別 (class) 與物件 (object) 的概念。
  - 物件 (object) 是一個在問題本文內能顯現意義的事物或實體。

物件導向

- 一群相似的物件是以物件類別 (object class) 或簡稱類別 (class) 來描述它。
  - **類別** (*class*) 與將物件分類成相關群組並描述它們共同特性的想法有關。
  - 類別描述了存在於類別中該物件的性質與行為。任何特殊物件都是類別的實例 (具體例子)。
  - 類別含有許多表示類別性質與行為的欄位 (field)，類別也可以包含資料數值與 (或) 方法 (子程式)。
  - 方法 (method) 則是物件中管理資料、數值的已命名演算法。

第 6.5 節 由p.176 設計方法 到 p. 184 跳過，以後會教

6.6 重要的串連思路

- 資訊隱藏(information hiding)
  - 意即在設計較高層級期間，讓較低層級的細節無法被存取。
  - 延遲細節
    - 為每一工作取名，先不需要擔憂工作後來是如何加以實作
    - **抽象概念**與資訊隱藏是一體兩面，表示相同概念 (如同硬幣的兩面)

重要的串連思路

- 抽象概念(abstraction)
  - 與資訊隱藏是事物的一體兩面。資訊隱藏是隱藏細節的實踐，而抽象概念則具有細節隱藏的效果。
    - 資料抽象概念 (data abstraction) 與資料的觀點有關；它將資料的邏輯觀點與其實體加以區分。
    - 程序抽象概念 (procedural abstraction) 與動作的觀點有關；它將動作的邏輯觀點與其實體加以區分。
    - 控制抽象概念 (control abstraction)。控制抽象概念與控制結構的觀點有關；它是將控制結構的邏輯觀點與實作方法加以區分。
    - 控制結構 (control structure) 可以讓我們控制改變一個演算法的循序流程，While與 If 都是控制結構。

重要的串連思路

圖 6.9 相同概念的不同觀點

重要的串連思路

- **為事物及事件命名(naming things and events)**
  - 使用速記法來代表工作與我們所處理的資訊
  - 我們會給資料及過程名稱，這些名稱稱為識別符號 (identifier)
  - 在將演算法轉換成為電腦可以執行的程式語言時，因為每個程式語言都有關於可作為識別符號的特有規則，因此可能需要修改識別符號
    - 例如有些程式語言要求識別符號 的第一個字母必須是英文，有些保留字不能作為識別符號

重要的串連思路

- 程式~~規劃~~語言(programming language)
  - 是一種人工語言，它由符號、特殊單字以及一組規則所組成，且被用來建構一個程式 (program) ——也就是說，用來呈現有意義的指令順序給電腦。
  - 語法 (syntax)是說明語言的指令如何組合在一起。
  - 語義 (semantic)則是說明這些指令代表什麼意義。
- 測試
  - 類似演算法的測試，實作階段的測試需要嘗試在不同輸入資料的排列組合下，程式是否正確執行。通常不是看測試的資料的量，而是需要測試在各種臨界值時，程式是否仍然正確執行