香港計算機科技聯盟

教學課程

合訂本

第二版

主編：Zinedine Zhou

封面設計：Bonnie Ho

Casio 認可計數機部份

第一章：fx-3900Pv 程式設計之直線程式

1.1 概論及基本指令

A. 概論

在筆者開首起筆之時，想起多年來的學習成果，真的不知從何說起。忽然間腦子裡閃出一段某電子用品的廣告，第一課就是要準確讀出該牌子名稱。現在就先說說你們學的東西怎樣稱呼。

在英文裡，很簡單也很肯定地是 Calculator。但是在中文裡卻不然。在香港，我們通常說成「計數機」，或者俗稱「Cal 機」或者「Cal-ter」，文字寫成「計算機」。然而，在中國內地，「計算機」一詞指香港人所說的「電腦」，內地人說 Calculator 為「電算機」，可是香港人說的「電算機」通常指大型電子用品裡的附件。為了統一起見，筆者會在往後寫成「計數機」。筆者是香港人，可苦了內地的同胞們了！

首先，大家看看以下的式子：

1 + 2 =

筆者不是在侮辱大家的智慧，當然知道每位讀者也能在半秒來答出正確的「３」來。可是，如果你要以計數機來計算以上的算式，你會怎麼做？

很明顯，按下「１」、「＋」、「２」、「＝」這四個鍵，不消一秒，便算出「３」來。但其實，以上四個鍵經已是一個有意義的程式了！為甚麼？因為你可以叫計數機進行這四個指令。**用某種特定的方法，使計數機自動執行一個或以上的指令來達到某些目的，便是程式。**

你可能會問，「『１＋２＝』已是程式？不是１＋２＝３嗎？」沒錯，１＋２真的等於３。可是，在計數機程式裡，**每一個指令也是動作，而非數學的表達。**就如上例，「１」是一個動作，這個動作是使螢幕上出現「１」這個數字，「＋」也是動作，作用是把「１」這個數字和往後的數作「＋」這個運算的準備，「２」則指重新輸入數值，使之與之前的數值（１）作「＋」的連繫，最後是「＝」，**等號的意思是「執行運算」而非「兩邊相等」**，這個概念非常重要。

B. 模式 (Modes)

可是，如果計數機是在「三進制」的模式 (Mode) 之下（雖然 *fx-3900Pv* 沒有這個模式，筆者只是打個譬喻），答案顯然不同，可見「模式」的重要性。**計數機裡的各種模式，就像是一間企業的不同部門。**

例如一間企業有人事部、門市部、對外關係部、會計部等等。各個部門在需要的時間互相溝通，互相協助。有些時候在不同的部門做同樣的事情，會有不同的後果。例如老闆說：「我想找一個漂亮的女孩當廣告主角」，人事部會在公司內部找尋合適人選，門市部會在接見客人的同時多加留意對方的外表，對外關係部會聯絡經理人公司，但會計部的員工卻會不滿，責怪老闆給他們無謂的工作。

計數機裡也有不同的「部門」，負責不同的職責。 “Mode” 鍵是在右上角 “OFF” 之下，按下之時螢幕左上角會有一 M 字出現。*fx-3900Pv* 的各個模式包括：

１）三角函數模式：

Mode 4 (DEG)： 螢幕上方會出現 “D” 字，表示三角函數 *sin, cos, tan, sin^{-1}, cos^{-1}, tan^{-1}* 會以「一圓為三百六十度」計算；

Mode 5 (RAD)： 螢幕上方會出現 “R” 字，三角函數以「弧度制」計算；

Mode 6 (GRA)： 螢幕上方會出現 “G” 字，三角函數以「百度制」計算；

以上三種模式必定有其中一種是被選擇的，而且不能同時選擇兩種三角函數模式但可以和其它模式同時使用。它們只影響三角函數的結果，對於其它運算無異，而且它們也可以在程式編寫時使用。

２）四捨五入模式：

Mode 7 (FIX)： 螢幕上方會出現 “FIX” 字樣，表示螢幕上的數字正以「準確至小數點後某個位」的方法顯示出來。可是，如果你單按「Mode 7」，計數機是沒有反應的，因為你並未給定其準確至的位值。你必須給定一個數字，如「４」。換句話說，若你想「準確至小數點後４個位」，則按「Mode 7 4」，記作 “FIX4”。

Mode 8 (SCI)： 螢幕上方會出現 “SCI” 字樣，表示螢幕上的數字正以「準確至有效數字某個位」的方法顯示出來。和 FIX 一樣，你必須給定有效數字的位值數量。你可知 SCI5 的意義嗎？

以上兩個模式並不能同時使用，也不一定要使用，它們可以在編寫程式時使用。值得注意的是，若單單使用上述模式，只是在**顯示上**準確至某個位，在實質運算上卻沒有捨去數值。**如果需要在實質運算的數值上捨去尾數，則按 RND, (RouND)** 即 SHIFT 0。

Mode 9 (NRM)： 這個模式可以在編寫程式時使用，它的作用是取消 FIX 及 SCI 的。若果並不是在四捨五入模式之時，卻是「科學記數法及普通寫法轉換」，按「0.00000005 NRM NRM」便見 NRM 在這方面的功用。

例子１．１．１：*在 FIX0 的模式下按「0.2 + 0.2 + 0.2 =」及「0.2 RND + 0.2 RND + 0.2RND =」顯示答案是甚麼？*

答案：

在第一個情況，FIX0 使 0.2 顯示成 “0”，當加上 “0.2” 之時，螢幕顯示依然是 “0”，但實際上計數機已把兩數加起，亦即是 0.4，再加上 0.2 時，答案變成 0.6，因為是 FIX0，所以「準確至小數點後０個位」，亦即是「準確至整數」，得「１」。

在第二個情況，FIX0 使 0.2 顯示成 “0”，當加上 RND 之時，它的實質數值不再是 0.2 而變成 0 。同樣地其餘兩個 0.2 也變成 0，故此整個運算成為「0+0+0」，得「０」。

例子１．１．２：*承上例，比較「0.2 + 0.2 + 0.2 = NRM」及「0.2 + 0.2 + 0.2 = RND NRM」的結果。*

答案：

如上例所述，在第一個情況之中得「１」的原因是 FIX0 準確至整數，然而在實質運算上結果依然是 0.6。所以，NRM 之後答案會還原為「0.6」。第二個情況之中，得出 0.6 之後 RND，於是在實質運算中捨去尾數，變成「１」。這個１已經是實質的數據，再與 0.6 無任何關係。NRM 之後答案保持「１」。

比較結果，兩者不同。前者無論是否在 FIX0 模式下運算，結果也是 0.6，但後者發揮到 FIX0 在運算上的作用。

由此可見：

> 在四捨五入的模式裡插入 RND，則顯示數值必和實質數值一致。否則，以運算結果而言，四捨五入模式並沒有發揮作用。

除三角函數模式及四捨五入模式之外，還有其它的模式：

３）積分模式 (Integration)

∫dx (Mode 1)： 這個模式運用「森普松法則」執行積分運算。執行這個模式的特別功能，必須編寫程式。但由於編寫的程式是固定的，故不在此述，讀者如有需要請閱讀說明書。

４）線性分析迴歸分析模式 (Linear Regression)

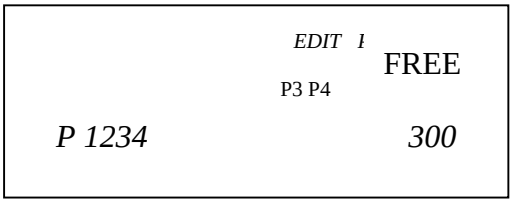
LR (Mode 2)： 這個模式是統計學功能，用以分析數據。由於在這個模式之中不能執行程式，故不在此述。

５）程式編寫模式

a) 程式編輯模式：(Mode 0, Edit)

這個模式是 Casio 認可系列中僅 *fx-3900Pv* 有的功能，亦因為這個功能的方便使我們注意在這部機之上。

在任何情況下按 Mode 0，螢幕會出現多組文字，這是**程式目錄**。



從左面開始，”P” 字是固定的，右面四個數字代表四個**程式區** P1, P2, P3, P4 都是空的。若果有最少一個指令步，則該程式區的代表數字會變成 “_”，如 “P12_4” 代表 P3 裡有指令。上面有 EDIT 字樣，表示現在是在 EDIT 模式之下，其餘 P1 P2 P3 P4 則是固定的。“FREE” 所在的位置稱為「點矩陣顯示」(Dot Matrix Display, DMD)，又由於在那個區域之中，有５個字元，每個字元的長和闊也是５點，故 *fx-3900Pv* 有「5ｘ5ｘ5 點矩陣顯示」。FREE 下面的數字代表該計數機的程式空間。完全沒有程式儲存的計數機有３００步。

現在看看計數機鍵盤上排第一橫行，從左至右是「SHIFT」、「P1」、「P2」、「↑」、「↓」、「OFF」。在程式目錄中，選定一個程式區，如 P1，則左面出現 P1 字樣。隨意按入一些數字，該數字便會出現在點矩陣顯示區內，這就是程式的內容，中間的數字，如 003，代表顯示的程式步是該程式區的第３步。

若翻查上一個或者下一個程式步，可按「↑」或「↓」。在按程式區之後按「↑」會到達程式的最後一步。

要刪除某一步，則在顯示要刪除的程式步時按「SHIFT」「↑」（即 CLR）。若需刪除整個程式，便按「SHIFT」「MODE」（即 PCL），返回程式目錄。若在程式目錄中按 PCL，則會刪除所有程式區的所有程式。

要跳出普通運算模式，可以關機重開，或者按 「Mode. 」（小數點）。

由於編輯模式可以翻查過往輸入的指令，筆者建議每一次編輯程式之時也使用編輯模式。

b) 學習模式 (LeaRN)

事實上，學習模式可以做到的事，編輯模式也可以做到，故此筆者不會在 *fx-3900Pv* 的部份解說。但由於其它型號如 *fx-50F, fx-3600Pv* 只有 LRN 模式，故此筆者會在較後的部份之中再詳述。

C. 常數記憶體 (Constant Memory)

甚麼是記憶體呢？記憶體就是把數字儲存，留待日後之用的地方。它們不會在關機之後被清除。在 *fx-3900Pv* 之中，可以記存常數的地方主要有七個，它們是：K1, K2, K3, K4, K5, K6, 及 M。前六個被統稱為「常數記憶體」 (Constant Memory 或統稱為「K-memory」)， “M” 稱為「獨立寄存器」。它們的作用都是一樣：記存一個數字。

情況就好像是一個個箱子一樣。一個箱子只可以盛載一個汽球，你可以把這個汽球加大、縮小、倍大，甚至「斬件」，隨你喜歡。只是，那個 “M” 的箱子寫著一些較特別的字句，用以和其餘六個箱子分開而已。

怎樣才能進行運算呢？在初級的課程裡，筆者打算只介紹當中的兩個：Kin 及 Kout。如果筆者想知得更多，可以參考中級課程中的「程式簡化」。Kin 讀做 “K-in”, Kout 讀做 “K-out”。這個不是甚麼國際音標，只是，人們把 Kout 說成「溝」，很是礙耳。

如果你想把 “4” 這個數字輸入 K1 裡，你就按：*4 Kin1*。那麼，K1 裡就儲存著 “4” 這一個數字。如果想把 K1 裡的數字就按 *Kout1*，那麼螢幕上就出現 “4” 這個數字。K1 裡的數字並沒有被改變，依然保持 “4”，直至下一個數字出

現並執行 *Kin1*，則新數值會**覆寫** (Overwrite) 舊的數。

從以上的文字可以看到，K1, K2 是名詞，Kin1 Kout1 是動詞。而且，如果你想把 4 儲存在 K2 之中，那麼你可以按 "*Kin2*"，如此類推。在手寫的時候，Kin1 可以略作 Ki1，Kout1 略作 Ko1，等等。

至於獨立寄存器，若要把數字儲入就按 *Min* (讀作 "M-in")，若要把數字讀取，則按 "MR"。

筆者在這裡再介紹多二十四個鍵：***K+1, K-1, K✗1, K ÷1, K+2, K-2, …K ✗6, K ÷6***

這二十四個鍵的功能是大同小異的。一理通，自然百理明，讀者不用擔心。這些按鍵的按法是「Kin」「+/-/✗/÷」「1/2/3/4/5/6」。例如：

K+1 即是 Kin + 1；
K-3 即是 Kin – 3，等等。寫時略作 K+1、K-3 等等。

這些鍵有甚麼用處呢？

設 **P** 為 "K" 字後的運算符號，*Q* 為運算符號後的字：
"K**P***Q*" 的意思就是「記憶體 K*Q* 的數字 **P** 螢幕數字，並儲存在 K*Q* 之中而螢幕數字不變」。(「螢幕數字」即是在運算的過程之中正在執行運算的數字)

例如：

4 Kin3 5 K+3

當執行 K+3 的時候，K3 = 4，螢幕數字是 "5"，「記憶體 K*3* 的數字**加**螢幕數字，並儲存在 K*3* 之中而螢幕數字不變」故此答案是：K3 = 9，螢幕數字維持 "5"。又如：

4 Kin3 5 K-3

當執行 K-3 的時候，K3 = 4，螢幕數字是 "5"，「記憶體 K*3* 的數字**減**螢幕數字，並儲存在 K*3* 之中而螢幕數字不變」故此答案是：K3 = -1，螢幕數字維持 "5"。只要大家小心運算時螢幕數字和記憶體數字的先後次序，那就可以了。

雖然常數記憶體和獨立儲存器的基本用處是一樣, 但是, 嚴格來說是不同的:

*X←→K：*這個功能可以應用在常數記憶體之中，但不能應用在獨立記存器之內，X←→K 的功用在「中級課程」中「程式簡化」一節中介紹。

*管理問題：*我們可以在六個常數記憶體之中, 以 K1, K4 為一組, K2, K5 為一組，K3, K6 為一組，可以有對應的值，在編寫程式上可以更方便更清楚。

*美觀問題：*如果獨立記存器內存有數字，螢幕上會長期有一「M」字出現, 很多用者都不愛看見這個 M 字。

D. 程式分類

好了，現在開始正式談談「程式」。筆者在「概論」中提到，程式就是使計算機自動執行那些指令。而在計算機的程式之中，可以分為兩大類：**特建程式**及**通用程式**。

特建程式就是程式內的指令是完全內置，外間無法更改其答案。亦即是：

程式→輸出

例如：求 $\frac{1}{\sin 1^\circ + \sin 2^\circ} - \frac{1}{\sin 3^\circ + \sin 4^\circ} - \ldots - \frac{1}{\sin 39^\circ + \sin 40^\circ}$ 之值。

讀者可以以數學的方法求出上述一式之值。但是，也可以編寫一程式來計算。這一個程式不需要任何輸入，(因為當中沒有任何變數)。只要設定開首為「1」，一直到「40」就可以了。這一個程式只會給出一個答案，而且它除了用來計算這條題目之外，沒有第二個用處，所以特建程式只是用來解決一個特定的問題。問題解決了之後，程式就變得沒有用處。

通用程式是一個可以根據用者輸入的資料，以特定的運算模式給出答案。

輸入→程式→輸出

例如我們常常說的「解一元二次方程」：

給定 $ax^2+bx+c=0$ 的 $a, b, c,$ 則 $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$

由於這條公式之中 x 的值是取決於 a, b, c，故此不能編寫一個特建程式解決問題。（並不是每一條二次方程的根都是一樣！）我們要編寫一個**要求輸入**的程式，從而給出**對應答案**，那就是一個通用程式了。在絕大部份的時間下，我們都是在編寫通用程式的。

「要求輸入」這個動作，就是用 ENT 和 HLT 這兩個鍵了。

當編寫程式時，按 RUN 鍵就是 ENT, SHIFT RUN 就是 HLT。在本質上它們是一樣的，但當計算機執行 ENT 的時候，螢幕的右面（指數的下面）會出現「ENT」字樣，於是，程式編寫員就會以 ENT 這個鍵來表示「要求輸入」，而 HLT 則用來顯示答案了。

程式除了可以因應其用途來區分成兩大類之外，也可以以它的編寫手法來區分成四大類：（括弧內為非必要的步驟）

收集程式

輸入→記存→輸入→記存……記存→運算→輸出

特點：先要求輸入所有變數，儲入記憶體，然後經過運算，輸出答案
優點：易於管理，而且編寫方式簡單，甚至可以很容易「心算出程式來」，
　　也不容易受用者輸入的答案所影響。（可以在輸入答案時執行運算）
缺點：程式會較長，對於一些大型程式可能不足夠記憶體使用

即時程式

輸入→（記存）→運算→輸入→（記存）→運算……→輸出

特點：在運算途中要求輸入變數
優點：程式一般會較短
缺點：容易受用者的輸入所影響（不可以在輸入答案時執行運算）

遞歸程式

運算→條件指令轉移→（運算）→輸出→輸入→記存及運算→回歸

特點：輸入數據之後記存及運算，然後回到程式開首作回歸運算，直至某一個數值符合要求，才跳出回歸部份，如有需要則再經過整理之後才輸出答案
優點／缺點：這種手法是專為某些需要回歸的程式而設的，他可以避免重覆的指令分寫成冗長的程式。但有一個缺點，便是在寫成之後較難找出錯處，也較難替其分析、改良等。

指令程式

運算→輸出

「甚麼？只有兩個步驟？筆者一定是打漏字了……」
我沒有打漏字。指令程式只有兩該步驟，就是運算和輸出。它內容上並不包含輸入部份，但本身是需要輸入數值的。執行方法是先按數字後執行程式，**模擬計數機上面三排功能鍵的操作方法，**所以指令程式依然是通用程式的一種。而且程式當中**一定不可以包括 + - ×÷= 這些運算符**，如果要執行運算則必須使用「K+1」「K-2」這些步驟。它的用處是模擬計算機上排指令的運算方法，例如：假定 P2 為 「x^4」，則在 P2 編寫「x^2 x^2」兩步，按「5 P2」時得出答案「625」。此兩步程式就是指令程式了。

程式的手法時常交替使用，尤其指令程式經常嵌入其它類型的程式之中。讀者能把以上四種手法運用純熟的話，就能寫得一手好程式。

E. 編寫程式的過程

第一步：認清問題
寫程式時一定要認清楚問題究竟是怎麼樣子。你必須和用者溝通清楚，究竟用者想要一些甚麼。很多時計算機不能滿足用者的某些要求，作為編寫員必須如實把它說出，並應提出相應的建議，務求盡量把用者的要求達到。

第二步：決定編寫及操作方式
也即是決定應該編寫「程式分類」之中哪一種編寫方法。考慮各種方法的優劣，因應使用者要求的操作方式而編寫程式。編寫的方式沒有一定，但通常只是「收集程式」和「即時程式」的交換。

第三步：數學上解決問題
這是一個十分重要的步驟，因為數學的概念直接影響程式的結構。而且如果編寫員不能把問題在數學上解決，根本不可以把它寫成程式！

第四步：編寫程式
編寫程式就是把數學上的成果轉成真正可以運作的程式，這一步當然也是我們不斷求上進的地方。上了很多很多的課，也是為了這一步。

第五步：程式測試及改良
把程式寫好之後，當然要看看程式本身有沒有問題，執行程式，看看能否達到用者的要求。很多時程式的表現和我們所想的有所出入，理論就是理論，實際就是實際，實際和理論不符，怎麼說也要改理論……你不能改變實況！

第六步：為程式撰寫文件及備份
當你肯定程式經已寫成的話，你應該把程式的**內容、用法、誤差、記憶體使用、數學分析、程式編寫分析，甚至推理過程**寫下。如果程式是給別人用的話，那麼你也要寫一份簡單的**操作方法**，令用者了解這個程式怎麼用。

第七步：把你用過的紙張儲起
做這一步的人不多，但這一步卻十分重要。因為你將來可能把它拿出來，對比一下當時你寫程式時的技巧如何，而且如果這條程式是給用者的話，用者遇上問題時你可以很快地回答他。還有在程式編寫的過程之中，你可能運用了一些嶄新的技術，很想把它留下，便應把它們寫下並儲好，才能流傳後世，造福人群。

1.2 空運行表與程式簡化 (Dry run table and simplification)

A. 直線程式的空運行表

當大家編寫大型的程式時, 會不會覺得混亂呢？記憶體呼喚錯誤, 又或是記憶體代表了兩種數字, 結果弄得一團糟？

計數機不同於電腦。電腦的高層次語言 (High level language) 可以根據程式編寫員的意願去設定記憶體的名稱 (User-defined identity), 但計數機不能。計數機裡只有預設的七個常數記憶體 (K1~K6 和 M)。名稱上它們是統一的, 因此旁人不能從常數記憶的名字裡得知它的用處。於是, 編寫程式的時候很容易產生混亂。**空運行表的作用就是記錄常數記憶體對於該程式的數學或非數學意義。**而我們所說的**分析程式**, 便是指**從計數機程式推導出記憶體輸入及輸出, 並解釋其作用。**

需要分析一條程式的情況有好幾個。可能是程式有一些問題, 編寫員想找出來；可能是程式本身的用途被人忘記了, 想找回程式本身的用處；可能是程式的使用方法被人忘記了, 不知道應該先輸入甚麼數據；可能是程式寫成了, 想知道別人是怎樣寫出來的, 用甚麼方程寫出來, 運用甚麼技巧。但是, 姑勿論甚麼目的, 我們寫空運行表的方法都是一樣的。如：

分析以下程式：
ENT KIN1 ENT KIN2 KIN3 x^2 – 2 K✕1 ✕KOUT1 ✕ENT = √ K+2 K-3
KOUT1 +/- K÷2 K÷3 KOUT2 HLT KOUT3 RTN

首先, 我們看見 K1/K2/K3 這三個記憶體進行運算, 我們畫表如下：

K1	K2	K3	運算中的數 ｜螢幕數字

我們看看程式的頭五個指令：*ENT KIN1 ENT KIN2 KIN3*。很明顯, 這是一個收集程式。縱觀整個程式, 共有三個輸入, 兩個輸出。其中這五個指令是要求首兩個輸入。我們設第一個輸入為 *X*, 第二個輸入為 *Y*。寫法如下：

K1	K2	K3	*Y* / *Y*
X	*Y*	*Y*	

再看看接著的運算：$x^2 - 2$ K↗1

K1	K2	K3	$Y^2 - 2$ | 2
X 2X	Y	Y	

（要注意不要把以前的記錄刪去）

之後是：↗Kout1↗ENT = √

此處要求第三個輸入，我們設它為 Z。

K1	K2	K3	$\sqrt{Y^2 - 4XZ}$ | $\sqrt{Y^2 - 4XZ}$
X 2X	Y	Y	

然後的程式是：K+2 K-3 Kout1 +/-

K1	K2	K3	-2X | -2X
X 2X	Y $Y + \sqrt{Y^2 - 4XZ}$	Y $Y - \sqrt{Y^2 - 4XZ}$	

K÷2 K÷3, 小心螢幕數字和記憶體數字相除的次序！

K1	K2	K3	-2X | -2X
X 2X	Y $Y + \sqrt{Y^2 - 4XZ}$ $\frac{Y + \sqrt{Y^2 - 4XZ}}{-2X}$	Y $Y - \sqrt{Y^2 - 4XZ}$ $\frac{Y - \sqrt{Y^2 - 4XZ}}{-2X}$	

最後是 Kout2 HLT Kout3 RTN，亦即是輸出答案。

大家看看 K2 和 K3 最後的兩條算式：很明顯，這是一條解一元二次方程。而且是經過數學改良的。編寫員將原式的分子和分母正負倒轉，亦即是說，程式應先輸出負根，然後才輸出正根。

從以上的例子可以看到，空運行表的另一個作用是用人手模擬計算機裡的運作，從而得出這個程式的作用、毛病等等。

無論是編寫程式時用到的空運行表，還是分析程式時寫的空運行表，當中的符號是**不用理會其普遍性的，也就是說，你可以以意創造，你喜歡怎樣就怎樣。**而事實上，在你分析程式之前，你根本不可能知道這是一個甚麼程式（或者你根本不知道程式裡有甚麼問題），故此你可以隨意地寫個符號上去來代表。

可是，我們從空運行表去思考程式編寫的同時，也得要從程式編寫去思考空運行表的結果。有一些如「取整」、「測試是否整數」等的程式部件，未必可以很容易就能把它們找出來，遇上這樣的情況，就要**憑經驗了**。在本課程及高級課程之時，更會用到邏輯運算。如果單單使用空運行表的話，差不多不可能找出程式的用處來。這樣，你就應該在空運行表的結果之上，**代入實數嘗試運作**，才能了解程式是用甚麼原理寫成了。

空運行表除了可以分析別人的程式之外，在自己編寫程式之時，也應該有一個寫空運行表的習慣。無論你寫多麼簡單的程式，又或是極其複雜的，都應該寫。最主要的原因是，**空運行表可以在數學表達和程式編寫之間的轉換中減少錯誤**，從而加快編寫的速度。此外，空運行表亦是**程式的具體記錄。**它記錄著程式的輸入及輸出，對於編寫之後的除錯工作有很大的幫助。

B. 程式簡化

為甚麼我們要把程式簡化呢？原因十分簡單：提升運算速度來增加效率及減少步數。總而言之，這是一個改良的步驟。不過，有很多時程式是否簡化直接影響程式的運作方式。例如，用「收集法」寫成的程式，第一步應為「ENT」。如果你把這一步刪去的話，理論上程式依然運作正常。可是，用者卻要在執行程式之前先輸入第一個數據。所以，作為一個程式設計員，應當以用者的志願為最優先考慮。其實筆者對於上述的例子，也不能有一個完全正確的答案，因為可能程式的使用者喜歡先輸入數據，後執行程式。如果對於用者沒有指明的話，那麼收集法的程式應是先執行程式，然後才要求數據。又或者，在程式簡化之後可能會得出某些問題，那麼這些程式簡化都不值得鼓勵。總括而言，程式簡化不應影響（或者「只可以從正面影響」）其操作方法和準確性。只能越來越好，不能越來越差。

不過，程式的簡化程度並沒有一個特定標準，又或者數學公式求得程式的最短可能性。最經典的例子莫過於「解一元二次方程」了。從平時我們手按的三十多步，到數學書上的二十五步，再到現在的二十三步甚至更短（詳情請參看程式總集的「中三程式」），沒有人知道會不會有一條十五步的程式出現，最重要的還是切合用者的需求。這個章節就為大家探討一下程式簡化的方法及理念。

從數學方面簡化程式

說到這裡，我猜讀者們也很悶了吧！雖然筆者的文筆欠佳，性格悶蛋，可是筆者也想和大家說一個小故事，為大家解解悶吧！

筆者執筆之時正值情人節，住在元朗的我約了女朋友到中環歡渡佳節。我從家裡出發，向北走了幾分鐘，走到巴士站去。等了十幾分鐘巴士才到，我便乘巴士到中環。真要命呀，幹麼在元朗不行，偏偏要到中環老遠去。不過算吧，她喜歡，我也沒有所謂。到了中環，她依舊是那麼迷人。我先和她到就近的餐廳吃午餐。一吃就花了我八十元！我很清楚，是 $80.1，真不明白為甚麼會有一個「一毫子」！我給了侍應一張一百元鈔票和一個一毫硬幣，他就給了我一張二十元鈔票，好像完全不在乎我的「貼士」。之後我和她看戲逛街……

故事說到這裡就告一段落吧！其實，筆者和女朋友的事又怎會放在這裡呢？這個故事也只是我自己作出來而已，大家不用猜了！而我說這個故事的原因，無非都是想向大家介紹，如何用數學的方法簡化程式。在這個故事之中，筆者就體現了兩次「數學簡化」了！

大家都知道，中環是在元朗的南面。筆者從元朗出發到中環，卻不向南走，偏偏要向北走，到了巴士站，才乘巴士向南駛去。為甚麼呢？為甚麼我不向南走？在吃飯結賬的時候，要八十元一角，為甚麼我要那麼麻煩，給侍應一百元和一個一角硬幣呢？要是直接給他一百元鈔票一張，不是更方便嗎？

大家也明白，要是我從元朗向南走到中環，我猜我在日落之前也不能趕到中環。如果我結賬時只給侍應一張一百元鈔票，沒錯我的確比「一百元一角」更方便，但後果將會是：我會整個錢包都滿是硬幣！

這就是所謂的「行個方便」了。我們在日常生活中所用到的算式，未必能完全適合計算機裡的運算方式。有時把算式複雜化，可能使程式更簡單，更快捷。至於怎樣才算是最好的算式、怎樣才能到達最好的算式，是沒有一定的標準的。也就是說，大家未必可以找到一個更好的算式。又或者，走了反方向的路，把程式極度複雜化也不為奇。通常來說，我們會把它簡化為一些相同的（或較相似的）大項。最好的例子就是「解一元二次方程」。

大家知道：當 $ax^2+bx+c=0$ 時，$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$。這是大家在數學書中看到的。可是，這個式子甚為麻煩，基本上每一項都不甚相同。於是，把這條式

改動一點點，可以很容易就得出：$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \equiv -\frac{b}{2a} \pm \sqrt{\left(-\frac{b}{2a}\right)^2 - \frac{c}{a}}$。

多麼一目了然的式子！從四項不甚相同的東西，一變就變成兩項差不多樣子，相信以讀者的頭腦，一定可以很容易就把它寫成程式了！理論上，後面的式子，筆者可以在十九步之內完成這個程式，可是由於計不了重根，也就沒有把它放上去程式總集了。各位讀者不妨嘗試一下編寫這個程式吧！

從計算機編程簡化程式

A) K+1, K-1, K×1, K÷1, K+2, K-2, …K×6, K÷6（共 24 個鍵）

這二十四個鍵在初級課程中已經學過，筆者不在此贅述了。

B) M+, M-

這兩個鍵的功能和 K+1 K-1 的功能差不多，只是對象是「M」而已。可是，這兩個鍵還有一個特別的功能，就是有一個隱藏的「等號」放在這兩個鍵之前，亦即是說，如果之前是連接著一堆運算的話，你不用加上「=」，直接「M+」或者「M-」就可以了。例如：

2 Min 4 + 7 M+

M 數為 14，螢幕數字為 12。雖然如此，但是由於 M+、M- 的隱藏等號會停止四則運算的持續，故此並不適用於指令程式。

C) 10^x, EXP

這個兩個鍵要看情況而定，在某些情況要使用 10^x，某些情況使用 EXP，大家只要小心觀察，就能看得出使用哪一個比較好。

D) 括號、等號及運算子編排

大家看看例子：

(3 – Kout1) +/- ÷Kout3 = 1/x Kin2

首先，大家可以看到，前面減號部份「(3 – Kout1) +/-」可以簡化成「(Kout1 – 3)」，因為：- (a – b) = b – a 。於是得出：

(Kout1 – 3)÷Kout3 ＝ 1/x Kin2

兩數相除，其商之倒數就即是兩數相除的次序的相反。亦即是說 $\frac{1}{\frac{a}{b}}=\frac{b}{a}$。

由此得出：

Kout3÷(Kout1 – 3)= Kin2

最後，**關括號和等號重疊，所以關括號可以省略不要。**最後答案為：

Kout3÷(Kout1 – 3 = Kin2

E) X←→K1, X←→K2…X←→K6

這六個鍵的意思是把 K 裡的數字和螢幕數字調換。例如：

3 Kin4 7 X←→K4 Kin5

大家可以猜到 K5 裡的數字會是甚麼？那就是 3 了。這個鍵非常有用，請讀者必須多加留意。

F) "FIX" RND "NRM"

"FIX0 RND NRM" 是一個十分常用的程式部件，其作用是「準確至個位數」，大家也明白。可是，當大家使用了 "FIX" (或者 "SCI" 情況一樣) 之後，如果還有其它數字需要作四寫五入的運算，只須要 "RND" 即可。直到輸出之時，才執行 NRM 解除。

G) + + XXX =

這種寫法對於大量重覆的四則運算有顯著的簡化作用。例如：

Kout1 + Kout3 = Kin 5 Kout1 + Kout4 = K×2 Kout1 + Kout2 = K-4

在這裡，由於所有的運算都包含 Kout1 的指令，所以我們可以將之簡化為：

Kout1 + + Kout3 = Kin5 Kout4 = K×2 Kout2 = K-4

總括來說，簡化程式的技巧便是上述的各種。簡化技巧十分重要，因為在 *fx-3600Pv, fx-50F* 的機種裡，編寫程式的位置不多（分別是三十八步和二十九步），如果程式稍有一點長，便不能在此兩種機種裡使用。此外，程式的長度直接影響其運作時間和輸入時間，這樣便會減低用者使用該程式的意欲。

1.3 計數機邏輯（插值法）(Calculator logic / Interpolation)

A. 基本知識

所謂「邏輯」(logic)，簡單來說，就是「對」(**T**rue) 和「錯」(**F**alse)。一些能斷言是對是錯的句子，我們稱之為「命題」(statement)，它的對錯稱為「值」(value)。我們可以表示兩個命題的邏輯關係，這種關係我們稱之為「運算符」(operator)，運算符只是一種關係，它是沒有值的，而且它所表示的關係是定義出來，並沒有任何根據，為表示該運算符的關係，我們會列出一個表，內容是所有運算符前後的各種邏輯值的組合和結果，這樣的一個表稱為「真值表」(truth table)。如果一個代數，其值是邏輯值，則該代數被稱為「布琳代數」(Boolean variable)，「對」以「１」表示，錯以「０」表示。

例如「在日出時，人站在地球上望看天，太陽從西邊升起」是一個命題，因為你可以確定它是對還是錯。在這句之中，它是錯的，故此它是一個錯的命題。

又如「樹木是動物**或**中環廣場比中銀大廈高」之中有兩個命題，前者是「錯」，後者是「對」。兩個命題之間以「或」(or) 這個運算符連接起來。根據「或」的定義，除非前後兩者都是錯，否則整個命題（前面的加運算符加後面的）都是「對」。根據定義，整句句子的值便是「對」。

此外，此部份某些公式以函數形式表示，如 *f(x), m(x)* 等，意思並不是 f 乘是 x，而是「有一條等式 *f*，它的未知數是 *x*，若給定 *x* 的值，則只會給出一個對應的答案」，我們稱之為「函數」(function)。

在編寫程式，尤其較複雜或變數較多的程式中，我們經常會遇到不同情況做不同的事，例如：

若 K2 > 7，則 Ko3 + Ko2 = Ki3，否則 K3 之值不變

由於計數機的程式編寫之中並沒有這種「若…則…」邏輯的運算，故此我們要在實數的領域之中尋求邏輯的運算條件，又因為計數機沒有「跳步」的功能，故此不可以以子程式 (Subroutine) 的方法去做。

在計數機程式編寫的領域之中，「邏輯」被定義為：

構作函數而滿足給定的條件

我們稱之為「計數機邏輯」，也等同於數學上的「插值法」。它未必存在「對」「錯」的成份，但我們可以用「如果…那就…」的命題來表達。請注意，作出來的程式必然是**直線性的**，我們可以很容易就寫出回歸性的程式，但這種並不是「邏輯」。

邏輯運算可以涉及很多很高深的數學概念，由於這本書是寫給初中學生們，所以筆者不打算介紹難以掌握的課題。

邏輯可以根據輸入條件的變數的多寡，分成：

1) ***一元邏輯***
2) ***多元邏輯***

也可以根據整個條件的類型，分成：

1) ***點邏輯***
2) ***集邏輯***

所有邏輯都屬於上面三組邏輯裡兩種的其中一種，換句話說邏輯的類型總共有八種。而我們也順序地為各種邏輯仔細地命名，如「一元點邏輯」、「多元集邏輯」等等**（注意：沒有「二進集邏輯」）**。筆者將會由淺入深，詳細介紹各種邏輯的運算方法及其應用。

讀者初時可能覺得難以掌握，不要緊，要了解邏輯需要同時學懂很多數學及計算機的概念，一時間難以掌握並非奇事，讀者應持之以恆學習各種要訣而不應急於熟習所有邏輯模式。

B. 一元點邏輯

何謂一元？一元的意思就是輸入的條件之中只有一個變數。
何謂點邏輯？點邏輯的意思就是只考慮給定的有限個實數的條件。

筆者舉個例子說明：（一元二進）

若輸入 1 則輸出 3，若輸入 0 則輸出 4

我們怎樣做才達到效果呢？如何構作一函數來滿足以上的條件？**在計算機邏輯之中，要找到答案，就必須要找「座標幾何」幫手。**步驟如下：

1）設定輸入為 x，輸出為 y
2）在座標幾何之中給出兩點，(1, 3) 及 (0, 4)

３）用**線**把它們連起來（這兒只需要用到直線）
４）把直線求出，那就是我們要找的函數了！

怎樣求出直線？你可以使用兩點式 (two-point form):

設直角座標上有兩點 (x_1, y_1) 及 (x_2, y_2)，則通過該兩點的直線的等式是：

$$\frac{y - y_1}{x - x_1} = \frac{y_1 - y_2}{x_1 - x_2}$$

在這個例子之中，等式為：

$$\frac{y-3}{x-1} = \frac{3-4}{1-0}$$

然後化簡，用 x 表達 y。(Express y in terms of x)，得

$$y = 4 - x$$

我們驗算一下：
當 $x=1$ 時，$y = 4 - 1 = 3$
當 $x=0$ 時，$y = 4 - 0 = 4$…*同時符合所有條件*

我們根據以上的函數寫成計算機程式：

ENT – 4 = +/-

就是這麼簡單。由於筆者相信各讀者經已熟習程式編寫，故此筆者往後只會解釋數學運算方法而不會把程式寫出來。

除以上例子之外，我們也可以運用其 **1, 0 的特性**來構作函數，特性就是１乘以任何數，其值不變，０乘以任何數，其值都是０，而且０加上任何數，其值亦不變。這種特性十分重要，讀者務必緊記。

但是，如果有好幾個條件的話，我們可以怎樣做呢？例如：

輸入值 (x)	輸出值 (y)
0	4
2	0
4	3
5	0
6	7

筆者在這裡介紹三個方法：

一、聯立多元方程

首先，我們知道：

> 如果直角座標上有 x 點且其中沒有任意兩點在同一 x 座標上，則最多需要一個 $x-1$ 次方程便可貫通所有點。

在上例之中給定五個條件（五點）。所以，設我們求的方程為：

$$y = ax^4+bx^3+cx^2+dx+e$$

將 x 及 y 各個數值代入上式，得：

$$4 = a(0)^4+b(0)^3+c(0)^2+d(0)+e$$
$$0 = a(2)^4+b(2)^3+c(2)^2+d(2)+e$$
……等等……

從這幾條式之中得出 a, b, c, d 四個數的關係，運用代入法 (substitution) 求出它們的值。但這個方法需要很多紙上工作，而且很容易出錯。另一方面，我們可以利用矩陣 (Matrix / Matrices) 來找出答案，但這種方法是在高等數學之中，筆者不打算介紹它了。

二、拉格朗日插值多項式 (Lagrange interpolating polynomial)

在數學上，拉格朗日插值多項式是這樣的：

> 設直角座標上有多個點，它們是 $(x_0, y_0), (x_1, y_1) \ldots (x_n, y_n)$，
> 則通過各點的函數為：
>
> $$y = \sum_{i=0}^{n} \left(\left(\prod_{j=0, j \neq i}^{n} \frac{x - x_j}{x_i - x_j} \right) y_i \right)$$

沒錯，的確是很駭人的式子。看不明白？不要緊，筆者寫出來就即是代表初中學生能應付得來。筆者把以上的數學算式寫成文字於下，好讓讀者們熟習：

1）寫 $y=$
2）畫一條份線，在最後加一個括號，括號後畫加號 (+)
3）我們現在做第一組數據（第一點）：把它的**輸出值**寫在括號裡
4）份線上的份子，是 x 這個**英文字母**減去其它 x **值**，乘在一起
5）份線上的份母，是 x 在這組數據的**值**減去其它 x **值**，乘在一起
6）在加號後面，用第二組數據重覆（2）至（5）步驟直至最後一組

說回剛才的例子，運用這種方法做的步驟是：

1）寫 $y=$

$y=$

2）畫一條份線，在最後加一個括號，括號後畫加號

$y=\frac{\quad}{\quad}(\ )+$

3）用第一組數據的輸出值 (4)，放在括號裡

$y=\frac{\quad}{\quad}(4)+$

4）x 這個**英文字母**減去其它的 x **值**，乘在一起，作為份子。

$y=\frac{(x-2)(x-4)(x-5)(x-6)}{\quad}(4)+$

5）x 這個數的**值**減去其它的 x **值**，乘在一起，作為份母。

$$y=\frac{(x-2)(x-4)(x-5)(x-6)}{(0-2)(0-4)(0-5)(0-6)}(4)+$$

６）重覆以上的動作，做第二組數據。

$$y=\frac{(x-2)(x-4)(x-5)(x-6)}{(0-2)(0-4)(0-5)(0-6)}(4)+\frac{(x-0)(x-4)(x-5)(x-6)}{(2-0)(2-4)(2-5)(2-6)}(0)+$$

大家看見最後乘「０」便知道，這一項是可以不做的。故此所有輸出項是「０」的數據也可以省卻不要，如上例之中，輸入值為２及５也可以不做，只要做三項就行了。很容易，你會得到：

$$y=\frac{(x-2)(x-4)(x-5)(x-6)}{(0-2)(0-4)(0-5)(0-6)}(4)+\frac{(x-0)(x-2)(x-5)(x-6)}{(4-0)(4-2)(4-5)(4-6)}(3)$$
$$+\frac{(x-0)(x-2)(x-4)(x-5)}{(6-0)(6-2)(6-4)(6-5)}(7)$$

剩下的就是多項式簡化的工作，便完成了函數的構作了。

這條式子為甚麼可以做到通過各點的效果呢？大家看看那個份數（這種份數，正統來說被稱為 Kronecker delta，記作δ。）：

假設 x_m 為某一個條件之中的輸入值。如果不是那組數據衍生出來的一項，那麼份子必定有其中一個乘數是 $(x_m - x_m) = 0$，則整項的值為０；如果那組數據是衍生出來的一項，你會發現份子和份母是完全一樣的，故份數的值是１，１乘以該對應的輸出值，得出對應答案來，再加上若干個０，便成為最後答案。（這正是１和０的特性！）

不知道讀者是否已掌握以上的計法呢？如果可以的話，恭喜你！你已經學會了高考「應用數學」的其中一個課題了！

三、函數斬件：邏輯控制

如果你認為拉格朗日插值多項式仍然是過於複雜的話，你可以把各條件**按輸入值的大小排列並分組（不要分得太多，建議兩組），並以特定的邏輯運算去控制。**可是這個方法需要用到集邏輯公式，讀者可先看下一部份「集邏輯」。

設 *m(x)* 為得出邏輯的函數，*f(x)* 為通過某特定點的函數，則整條函數是：

$$y = m_1(x)f_1(x) + m_2(x)f_2(x) + \ldots + m_n(x)f_n$$

如上例，筆者把頭兩組數據為一組，其餘為第二組。分組的方法並不是固定的。設通過第一組的所有點的函數為 $f_1(x)$，第二組的函數為 $f_2(x)$。

運用任何方法求出該組的函數，得：

$$f_1(x) = 4 - 2x$$
$$f_2(x) = 5x^2 - 48x + 115$$

當 $x \leqq 2$ 時，$m_1(x) = 1$ 而 $m_2(x) = 0$；當 $x \geqq 4$ 時，$m_1(x) = 0$ 而 $m_2(x) = 1$

我們把前提改一改：

當 $x < 3$ 時，$m_1(x) = 1$ 而 $m_2(x) = 0$；否則，$m_1(x) = 0$ 而 $m_2(x) = 1$

運用「集邏輯公式」及「兩點式」，我們很容易知道：

$$m_1(x) = \frac{1 - \frac{x-3}{|x-3|}}{2}$$

由於第二個條件和第一個條件是「１、０互轉」的關係：

$$m_2(x) = 1 - m_1(x)$$
或
$$m_2(x) = \cos^{-1}(\sin(m_1(x)))$$

最後把整條方程寫成 $y = m_1(x)f_1(x) + m_2(x)f_2(x)$ 即可。

C. 集邏輯

何謂集邏輯？集邏輯是指考慮條件之中包含某一特性，而且它所表達的是無限個實數。

舉例：

若 x 為正數則給出 1，若 x 為負數則給出 0

首先，我們能直接看得出，**x 的值的大小與給出的答案無直接關係，關係只在乎 x>0 還是 x<0**。我們可以運用「集邏輯公式」：

$$\frac{x}{|x|}$$

|x| 的意思是取其整數，在計算機裡就是「x^2 √」這兩步。運用集邏輯公式，我們得出正數為 1，負數為 –1 的值。再根據條件和兩點式定理（見下），即：

1 換成 1，－1 換成 0

得 $y=\frac{x+1}{2}$。（此處 x 是集邏輯公式的結果而非當初的輸入）

集邏輯公式能分辨正負數，將其加一然後除二，就能得出相應的二進邏輯。所以，如果要分辨某兩種特性，就應把它符合某條件的為正數，其餘的為負數，然後運用集邏輯公式把它們歸納成「１」或「－１」，利用兩點式轉成所需要的輸出結果便成。

可是，集邏輯公式有一個壞處，那就是集邏輯的邊緣是不可以運算的。也就是說，如上面「正負數」的例子之中，若 x= 0 的話，運用集邏輯公式便會得出不可能運算的答案來，在計數機裡面遇到這樣的情況便會 –E– 終止運算，就算使用０〆的方法也行不通。故此，我們除了要懂得運用集邏輯公式計算集邏輯之外，我們還須要按情況使用其它方法來進行運算，我們可以用**點邏輯捨位法**來達到。

例如，給定以下條件：

若 x < 0 或 x>0，則 y = 0
若 x = 0，則 y = 1
任何情況下 – 10 ≦ x ≦ 10

首先，我們要熟識函數數線的特性。對於一條二次方程而言，代表它的線有一個頂點，而整條線是沿那個頂點的垂直線為對稱軸，左右對稱。我們也知道，若果某數 a≧0.5 四捨五入時便會進位，其餘則不會進位。要注意的是「大於或**等於**」的特性。運用這個「等於」我們可以設一條方程使得：

當 x = 0, y = 0.5
當 x = 9 或 x = -9, y = 0

我們可以得出一條兩邊向下彎, 尖頂向上的曲線, 其最高點為 *x = 0* 且 *x* 在其它的值之時也是在 0.5 及 0 之間。得出函數之後，把答案四捨五入 (FIX0 RND) 便可以了！而這一個方法也可以在混合點邏輯和集邏輯的條件中使用到。

如果我們需要計算好幾個不同的間距, 我們該怎樣做呢？這時便需要用到**集間公式（又稱「卑斯公式」，Base formula）：**

設 x_1 及 x_2 分別是兩個給定點。
$$y = (x - x_1)(x - x_2)$$
當 y < 0 時, x 是介乎 x_1 及 x_2 之間
當 y >0 時, x 不是介乎 x_1 及 x_2 之間
當 y = 0 時, $x = x_1$ 或 $x = x_2$ 其中之一

這條式來自二次方程的曲線圖像，證明從略。

D. 多元邏輯

多元的意思是輸入的變數有多個，注意**多條件**和**多元**是有分別的。多元的意思是變數的多寡，而且這些變數的**關係**是從各**條件**去表達。

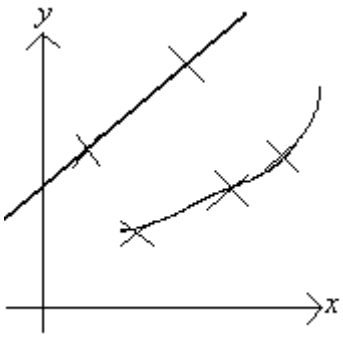
筆者用以下的條件來舉一個多元邏輯的例子：

輸　入		輸出
a	*x*	y
1	1	4
1	5	6
2	3	2
2	7	3
2	9	4

從上表可見，*a* 和 *x* 的值是互不干涉的，可是它們的值也**同時**影響到最後輸出值 *y*。面對這個情況，我們會使用**系數控制**的方法。

系數控制其實就是之前各個方法的「變數版」。一直在討論的一元邏輯只有一個變數 *x*，構作函數 *f(x)* 使之在直角坐標上通過某些特定點（條件）。現在我們要說的是在**已有的函數的實數裡**再加入更多的變數（我們在這個例子中只談多加一個變數）。系數控制法也需要用到「分組」這個步驟，和之前點邏輯「邏輯控制」部份一樣，分組的方法有很多個，以下僅為其一。

由於 *a* 在首兩個條件的值是一樣，接著的三個也是一樣，故此筆者以此把五個條件分成兩組，即下圖：



（線段的特性隨著 *a* 的值而改變！）

首先，**暫時不理會 *a* 的值**，把它當作是僅有 *x* 的情況。我們很容易知道：

直線方程是 $y=\frac{1}{2}x+\frac{7}{2}$

曲線方程是 $y=\frac{1}{24}x^2-\frac{1}{6}x+\frac{17}{8}$

a 的值是影響著兩條函數的系數，當中：

f(1)=0, f(2)=1/24

g(1)=1/2, g(2)=-1/6

h(1)=7/2, g(2)=17/8

到了最後我們需要的式就是：

$$y = f(a)x^2+g(a)x+h(a)$$

運用兩點式定理，得出：

$$f(a)=\frac{a-1}{24} \qquad g(a)=\frac{7-4a}{6} \qquad h(a)=\frac{39-11a}{8}$$

最後答案：

$$k(x)=\frac{a-1}{24}x^2+\frac{7-4a}{6}x+\frac{39-11a}{8}$$

筆者已介紹了在計數機裡常用的邏輯方法。其實邏輯學是一們很高深的學問，無論是在哲學層面，還是電學、數學，甚至是在計數機裡的邏輯用法，也有不少是很深奧的（如：多元二進的邏輯等價轉換等等），由於上述方法已足夠應付絕大部份的複雜條件，故此筆者不在此多述了，各讀者如對此有興趣可以參考有關「數論」、「數值方法」、「邏輯學」等的書籍。

1.4 有關計算與記憶體的補充

MT 析取法／數位析取法 (MT extraction)

此方法由筆者的網友 MT 所建議，她是第一個引用這個方法於計數機之中，故此在計數機裡此方法以她來命名。

如果大家在運算途中有大量整數（或小數點後一兩位的數）需要儲存，把它們直接放進記憶體裡，令一個記憶體儲存著一個數字，不是很浪費嗎？一個記憶體儲存的數字至少多達十個數位，如果只用其極小部份來獨立儲存，實在沒有必要。我們可以盡量運用這些空餘的數位來儲存，那麼記憶體便能更有效地運用了。

數位析取法的原理，是把一整個數字視為一個「**字串**」，把各種數字以**特定的間距**存入該字串之中。到取出之時，便以實數的運算方法把需要的數字**抽出**。其實在電腦學中這個方法老早就給人用了，但筆者暫時未見到別人在計數機裡使用這種方式。

筆者把儲存著好幾個數字的大數稱為「字串」(string)，把每個組成該字串的各數稱為「字元」(character)，故字串由字元所組成。值得留意的是，在電腦之中，字元有時僅指單一個數字（０至９），但是在這裡，字元是指需要記憶的一個數，可能是個位數，也可能是十位、百位等等。為了方便起見，筆者只以字元為個位數的字串，其原理和非個位的字元的類同。

假設我們需要儲存的數字順序是 5, 4, 5, 7, 6, 8, 9, 2, 1, 0。當接到一個數的時候，便要決定它儲存的位置。假定字元是在字串裡**從右至左**排列，即該字串最後應是 “129867545”。

把字元放進字串的步驟不難，只要在收到數字之後乘 10^{n-1}（該字元是在整個字串之中排第 n），然後加進字串裡即成。

至於把字元取出就不太容易了。

１）決定要取出第幾個字元（從右向左數），假設是第 n 個

２）把字串除 10^n，取整，又乘 10^n，和字串相減

３）得出之數再除 10^{n-1}，取整，答案便是該字元了

數位析取法的好處便是在計數機中引入「數據庫」(database) 和「一次元陣列」(1D array) 的概念，令記憶體更全面地運用之餘，亦令計數機能記憶多個數字。

暫存器(Register)記憶：括號儲存法 (bracket memory) 及 X←→Y

無論是甚麼計算工具, 總有一個位置供數據暫時儲存, 我們稱這些暫時的儲存位置為「暫存器」(Register), 通常來說這些暫存器是在運算的過程之中儲存數據, 當運算完成之後便會被清除。在計數機之中也有暫存器, 其中我們較常用的是「括號」及「X←→Y」兩個。

當我們開啟了一個括弧組, 我們便可以完全重新地建立新的一組運算, 不受任何「先乘除後加減」或者其它的運算的先後次序所影響。故此, 在開啟括弧之前的運算是必須要給儲存的, 我們便運算這（幾）個儲存的位置來增加我們記憶常數的地方。

假設我們剛完成一輪運算, 最後一步驟為 "=", 其答案需要給儲存, 運用「括號儲存法」如下：

….…..… = + [(- - -

在括弧裡可以開展一運算, 以**繼續更改常數記憶的值**, 到更改完畢之時：

⁄0 - - -)]

關括號之前由於是乘零, 故此在括弧裡的最後值為「０」, 所以關括弧後答案是０, 但此時如果需要括號前的數字繼續運算, 便可繼續使用四則運算或其它符號, 如果僅 Kin, 或 K+1 之類, 則需在關括號後加一等號。

使用括號儲存法時要注意：

１）留意括號會否超過計數機的上限：

２）括號外要記的數不能直接參與括號內的運算, 同樣括號內的運算結果不能和括號外的數進行運算, 因此宜把括號內的運算視為「更改記憶體的值」

在括號儲存法的括號之中的運算, 如果相等於我們在普通運算時的「=」, 我們當然不可以在括號之中加上等號（為甚麼？）, 此時我們可以在括號裡再增建一組括號, 以關括號代替平時使用的等號。

至於 X←→Y (SHIFT Kout, 中文唸成「X 去 Y」) 的概念, 我們便要從最根本的四則運算觀念說起了。

舉個例子：

5 – 3 =

按這四個簡單不過的鍵，相信任何人也不會覺得困難。但當中的運作情況，大家就不太了解。當用者按「+」時，計算機就會把「5」放進暫存器中，並以另一記憶體儲存「+」這個運算子。用者輸入「3」並按「=」執行運算，計算機便會把暫存器裡的「5」和之後輸入的「3」以「-」(減) 的方式依次序進行運算，然後刪除暫存器裡的數據。因為 5 – 3 = 2，故此計算機得出「2」這個答案。

X←→Y 的作用就是把螢幕顯示中的數字和暫存器的數互相交換。引用上例，若加插此指令如下：

5 – 3 X←→Y =

計算機就會給出「-2」的答案。用者輸入「3」時，暫存器內儲存著「5」，當執行「X←→Y」時，計算機就把暫存器和螢幕上的數字互調，即是暫存器變成「3」而螢幕顯示「5」，當按「=」時，就會執行暫存器中的「3」「-」「5」，故此得出「-2」。用圖表解釋如下：

按鍵	5	-	3	X←→Y	=
螢幕顯示	5	5	3	5	-2
暫存器	0	5	5	3	0

需要注意的是，X←→Y 只能在運算過程中轉換**即時的**被加數、被減數、被乘數、被除數，對於先乘除後加減的之前的數是在其它地方儲存著，由此可知，如果按了四則運算符之後，暫存器的數字必定和螢幕數字相等，且每次按了之後，其值亦會被更新，且在等號之後其值必等於０。

除了在運算之中會用到 X←→Y 的轉換之外，某些其它功能也會用到，例如 R→P 之中，顯示答案（Modulus）之後按 X←→Y 便會顯示另一相關答案（Argument），等等。

可是，在程式編寫領域之中，X←→Y 的用處並不是運算的暫存器轉換（雖然這是它的最基本意義）。這個指令在程式之中的角色，主要是因為它容易存取和刪除，對於重覆運算的遞歸程式（第二章）比使用 K-memory 更方便。

截距誤差 Truncation error

無論是簡單如一部計數機, 還是精確如一部電腦, 儲存數字的記憶體總會有限度。因為世上有太多的無理數（不能寫成兩整數相除的數), 如絕大部份的三角函數、對數、開平方、開立方等等, 在計數機裡只能儲存**準確至某一位**的**一個實數**。結果在多重無理數的運算之後, 其答案會因為尾數的不同而與筆算的準確值有所偏差, 如果這種偏差是因為捨去尾數（不是四捨五入！), 我們稱之為「截距誤差」。一般來說這種偏差, 對人的影響不大, 因為人可以用腦袋分辨出其真正的值, 但是計數機卻不然, 而且甚至會帶來災難性的後果。

在 *fx-3900Pv* 的機身上, OFF 鍵的上面或是「０」鍵的下面會看見 "10+2 DIGITS" 字樣, 這個就說明計數機的儲存數字的多少了。「１０」的意思是直接顯示的數位數量, １０即是說這部機可以同時顯示十個位值的數。後面的「+2」意思是它的內部還儲存著「2」個位, 所以, 運算上它做到 10+2=12 個位。

例如：

$$2 \div 3 - 0.666\,666\,666$$

大家也知道２÷３的答案是 0.666… 且有無限個６字, 在計數機之中, 它只能儲存著１２個而顯示１０個, 我們把這個數減去所有可以顯示的數, 結果是「6.6×10^{-10}」, 可見計數機的確儲存著十二個位的數。（包括０）

這樣很好嗎？我看不是。試試：

$$2 \div 3 = x^2 - 4 \div 9$$

出人意表地, 你會看見「-1×10^{-11}」！$(2/3)^2$ 不是等於 4/9 的嗎？為甚麼會這樣呢？明顯地, 這是截距誤差惹的禍。數學上, ４÷９和２÷３都有無限個小數位, 在計數機裡儲存著的分別是「0.444 444 444 44」和「0.666 666 666 66」, 把後者平方, 數學上便是：

$$0.666\,666\,666\,66^2$$
$$= 0.444\,444\,444\,435\,555\,555\,555\,56$$

計數機取答案（包括開首的０）十二位, 便會發覺它的第十二個位是「３」, 這就是做成誤差的原因了。筆者還沒有治本的避免方法, 現時唯一的方法只是在適當的地方加上「FIX9 RND」（或 FIX8 RND）而已。

標準計數機按鍵代碼

(Standard Identity of Calculator Keystrokes, SICK)

SICK 是筆者的網友 TM 於 2002 年 10 月發明。它由很多套代碼所組成，每一套由 00 開始，以十六進制排列到 FF，每一個代碼代表該系列計數機的某一按鍵方法。每一個**系列**的計數機的 SICK 都不同，但如果兩種**型號**的計數機的按鍵方法相近的話，便會使用同一套的代碼。例如 Casio 的 *fx-50F*、*fx-3600Pv*、*fx-3900Pv* 等份屬不同型號，但它們的按鍵及其表示方法也很相近，所以這些機種使用相同的一套代碼。但如果是 Casio 和 Sharp 的計數機，那麼它們的 SICK 必定是不同的了。

要注意的是，**SICK 所代表的不一定是程式碼，而是包括一切功能**，這些功能以「指令」為單位，例如「X←→K2」雖然要按 SHIFT Kout 2 三個鍵，但是我們只需要一個代碼就行了。此外，**SICK 還照顧到一些按情況決定其內容的程式**，例如「二分法」之中所輸入的函數，是需要用者按情況去更改程式的內容，SICK 可以以「*f(K1)*」來表達，也有「…」的代碼代表一切按情況變更的複雜程式段。

為甚麼我們不直接使用計數機程式碼或者按鍵次序而要有 SICK 呢？首先，如果我們只寫真正的按鍵，往往耗費大量篇幅，顯示也不明確，例如「x^2」、「X←→K1」等的按鍵需要使用特別的文書處理效果才能表達，故此在沒有這些功能的系統便不能有效地傳播這些指令，甚至引起誤會。發明者已在網上供應計數機指令及 SICK 互相轉換的軟件，程式編寫員無需誦記各編碼。 Casio *fx-3900Pv* 適用之 SICK 在本書最末「附錄」中列出。

SICK 的原理是用者透過 SICK 編碼器 (encoder) 把輸入的計數機碼轉成 SICK，並將之儲存成為一個特定的檔案 (.sic)。檔案包含以下幾項資料：

１）該編碼器的版本

２）編碼使用的計數機系列

３）編碼內容

轉成檔案之後，便可以更便捷地互相傳播，當使用者收到 sic 檔案時，使用 SICK 解碼器 (decoder) 將之正確地翻譯成計數機程式碼，翻譯成的程式碼可以以不同的格式輸出和儲存，理論上它可以儲存成為普通文字檔 (txt)、圖片檔 (jpeg) 等等，但網上現時提供的只有 html 以表格形式輸出。

而 SICK 最大的優點是：它只包含英文字母和數目字，用者甚至可以以「複製貼上」的功能傳播程式，令用者可以在沒有文書處理功能的系統裡傳遞程式。

第二章：fx-3900Pv 程式設計之遞歸程式

2.1 轉移 (Jump)

讀者在第一章裡已經學過「RTN」這個指令，它的用途就是**無論如何**也**回到程式的第一步重新執行**。「無論如何」我們稱之為「非條件」(Unconditional),「回到程式的第一步重新執行」的指令我們稱為「指令轉移」(Jump)，所以 RTN 也被稱為「非條件指令轉移」(Unconditional Jump)。

有「非條件」的，當然也有「條件」(Conditional) 的,「條件指令轉移」(Conditional Jump，簡稱 CJ) 是計數機裡非常重要的指令。在 *fx-3900Pv* 裡有兩種 CJ，一個是「X>0」，另一個是「X≦M」。兩者的「X」也是代表「螢幕數字」。這兩個 CJ 的特性也是一樣：當條件是「對」時，則當作 RTN 的用途，當條件是「錯」時，則繼續執行 CJ 之後的步驟。

舉個簡單的例子：Kout2 X>0，那麼如果 K2 不是大於零（即「等於０」或「小於０」），便執行之後的步驟，但若果是大於０的話，計數機便會走到最開首之處重新執行。

CJ 的作用就好像是一道鎖上了的閘門。當計算機運行至此之時，閘門就會決定計算機的某些數值是否符合條件，從而決定應該回歸執行重覆的部份，還是通過閘門執行其它部份。至於決定計算機應否通過這一個閘門的數字，我們稱為**「終點標記」(Sentinel)**，是開啟閘門的鑰匙。這一個標記可以是一個記憶體裡的特定數字，也可以是（加工過的）運算結果。

可是，計算機始終是計算機，功能有限，不能要求它可以有電腦的效果。（如果可以的話用者就要付上電腦的價錢！）指令轉移使計算機回歸執行，但回歸的地方一定是計算機的最開首部份，用者不可以設定它回歸到某一個特定的位置，於是，如果用者想設定一個遞歸程式，他必須要把重覆的部份放在程式的開頭。也正因如此，很多程式也要求用者先按「KAC」才執行程式，就是因為程式可能不通過開首的重覆部份；也有一些程式要求用者先輸入數據（「ｘ」Kin1 「ｙ」Kin2 Px）是同樣的道理。當然，最理想的程式應該只需要「執行程式」。但是，無論是怎樣好的回歸程式，在第一次執行的時候，也要求用者「按若干次 RUN 才輸入數據」，這個現象我們在下一節詳細探討。

2.2 RRR、流程道 (flow path)

正如剛才所說，計數機如果要回歸執行之時，必定要回歸到程式的最開首部份。那麼，如果程式需要兩個**獨立**部份作回歸運算的話，其中一個部份必需無意義地重覆執行著，這個現象，在計數機的領域裡，我們稱為「回歸重覆定律」(Rule of Repetitive Recursion, RRR)。

> RRR：
> 如果程式中有某些重複的部份，必須將之放在程式開首。
> 如果程式中有多個重複部份，則除了第一個遞歸 (looping) 之外，
> 執行其他的遞歸時必須無意義地重複之前的所有遞歸。

RRR 也闡明了以下的不可能：

執行程式→輸入數據→執行運算（包含回歸）→輸出→輸入數據→……

在第一次執行程式的時候，是輸入數據為先，然後運算。(1)
但根據 RRR，回歸運算部份必須放在開頭，而上述流程則要求在回歸之後輸出答案，然後才要求輸入。(2)

如果 (1) 是可行的話，在回歸之後，便直接要求輸入數據，和 (2) 不符；如果 (2) 是可行的話，則在第一次執行時，經過了「回歸部份」，便是「輸出答案」然後才「要求輸入」，和 (1) 不符。故此 (1) 和 (2) 是不能同時滿足的，也就是說，不可能達到以上的要求。

而流程道是筆者多年前自我歸納出的一種思考方法，早在第一版教學文件的初稿中經已提及，至今仍然覺得很有用。

流程道是一個程式的大意。首先，畫一條長長的直線，代表整個程式。

線的左手面代表程式的開首部份，一直到右手面，而計數機也好像是從左手面的步驟執行至右手面一般。在流程道之中使用符號來代表程式的結構：「｜」代表 CJ,「↓」代表「要求輸入」,「↑」代表輸出,「‖」代表 RTN。符號之間的線段代表要執行的程式步，我們在線段上以文字來寫出該段程式的用途。由此可見，流程道展示了程式的各部份的用途。

2.3 單邏輯與暫存器的使用

運用流程道和 RRR 可以很容易知道，對於一個需要回歸運算的程式來說，其回歸部份必定放在程式開首的位置，然後是輸出／輸入數據，之後是直線運作的程式，然後才回到程式的一開首作回歸運算。由此可見，在我們**第一次執行程式**的時候，我們**必先經過回歸部份**，才執行輸入數據的程式。但是，在回歸的部份之中會有「提取記憶體」的步驟，而這些步驟對於程式的執行來說是沒有作用的，反而會因此而造成錯誤，例如在回歸之中包含「除號」而該記憶體正好是「0」，又或者記憶體不符合某些條件而導致無限回歸，不能走出回歸部份之外。但這些錯誤只會出現在第一次執行時，而在往後的輸入之中不會出現。所以，在第一次執行程式之前，必先使記憶體的數據有良好的特性，令計數機能「逃出生天」，這個動作，我們稱為「初始化」(Initialization)。

較舊的程式要求用者先把數儲存在特定某幾個記憶體，然後才執行程式。但這種方法對使用者來說十分不方便，因為需要按的鍵太多，而且如果用者忘記了初始化的鍵，便會使程式「-E-」或者無限遞歸而不能正常運作。所以這種方法必須予以取締。幸運地，單邏輯和暫存器的應用可以近乎完全地解決了這個問題。

還記得二進單邏輯有甚麼特性嗎？0乘任何數，其值必是0，0加任何數，其值不變，1乘任何數，其值也不變，而且0和1之間可以以「$cos^{-1}(sin(x))$」來互相轉換。暫存器的作用，大家也記得吧。在這裡讀者需要多一個概念：如果暫存器裡面是儲存著數字的話，即使計數機執行了 CJ 或 RTN，儲存著的數字也不會被刪除。

那麼，暫存器和單邏輯如何應用呢？其實十分簡單。在程式的開首編寫：

X←→Y K1 K2 K3 K4 …

在 CJ 之前：

(前接「終點標記」) X←→Y 1 X←→Y [CJ]

當第一次執行的時候，暫存器內沒有數據，於是清除 K1、K2、K3。到後來運算了一系列步驟，並將「1」放進暫存器中。然後執行 CJ，若回頭便執行第二次運算。

當第二次執行的時候，暫存器內儲存著「1」，於是沒有改變 K1、K2、K3 的數值，繼續運算。通常 X←→Y 的用途就是這個。

2.4 最大公因數與最小公倍數

數學背景

最大公因數 (Highest Common Factor, HCF or Greatest Common Divident, GCD) 是最大的可以同時除盡兩個或以上既定的正整數的正整數, 而**最小公倍數 (Least Common Multiplication)** 是最小的可同時被兩個或以上既定的正整數除盡的正整數。求的方法有很多, 常用的有：

1) 撞數： 從 1 開始逐個數去除, 直至達到最小的那個數為止。這個不算是數學的方法。
2) 質因數連乘： 把各數化成**質因數連乘式**, 運用擷取的方法抽答案。這是小學生們的做法, 筆者不在此贅述。
3) 輾轉相除法： 這是本課程的重點, 於 1.2 節詳細講述。

對於單單兩個數而言：

對於任何兩個整數, 其積等於最大公因數和最小公倍數之積

證明從略。

例子 (2.4.1)： 18 和 24
LCM：72
HCF：6
兩數之積：18 ×24 = 432 = 72 ×6 = LCM ×HCF

我們可以根據以上的定理, 只需要考慮怎樣找出它們的 HCF, 便能取得其 LCM。在這個前提下, **筆者將會在此課僅談論找出 HCF 的方法。**然而, 以上的定理只適用於僅有兩個數的時候, 故此對待超過兩個數的情況時, 如果使用以上的定理, 可能過於轉折而變得過份繁複。

此外, 找尋 HCF 或 LCM 的方法必須是**遞歸性**的, 並沒有直線性的方法。

數學裡的輾轉相除法

輾轉相除法 (Euclidean algorithm) 是一種機械式的除法，和「撞數」或「質因數連乘」的方法不同，它含任何撞數的性質。要化一個數成為質因數連乘式需要撞數。例如化 35 為質因數連乘式，你必需從 “2” 這個最小的質數試起。2 不能整除 35，於是試 3，如此類推，直至試出 5 為止。由此可見，質因數連乘也需要「撞數」的元素，而且由於 RRR (請參考「基礎課程」)，運用質因數連乘的方法求 HCF 並不化算。我們可以使用輾轉相除法。不幸的是，它不能同時使用於三個數之中，又因 RRR 會使之更為繁複。

輾轉相除法運作原理如下：

例子(2.4.2)：用輾轉相除法求 1894 及 3622 之最大公因數

首先，畫三條直線，把該兩數放在線與線之間：

1894	3622

兩數相除。3622÷1894 = 1 餘 1728，寫法如下：

1894	3622	1
	1894	
	1728	

當中 3622 下的 1894 是 1894ˣ1 之積。之後將餘數和 1894 相除，1894÷1728 = 1 餘 166，寫法如下：

1	1894	3622	1
	1728	1894	
	166	1728	

再用 166 和 1728 繼續相除：

1	1894	3622	1
	1728	1894	
2	166	1728	10
	136	1660	
	30	68	

繼續下去，直至除盡為止：

1	1894	3622	1
	1728	1894	
2	166	1728	10
	136	1660	
3	30	68	2
	24	60	
3	6	8	1
	6	6	
		2	

最後的一個餘數為「2」，故此 1894 及 3622 的最大公因數為 2。值得留意的是，在輾轉相除法之中，相除時的商（左右兩排外圍數字）是沒有意義的。我們再舉一個例子，看看讀者可否自行算出答案：

例子(2.4.3)：用輾轉相除法求 11137 及 20167 的最大公因數

答案如下：

1	20167	11137	1
	11137	9030	
4	9030	2107	3
	8428	1806	
2	602	301	
	602		

由此可見，20167 及 11137 之最大公因數為 301。

而事實上：

11137 = 7×37 ×43

20167 = 7 ×43×67

最大公因數為 7×43 = 301，答案和使用輾轉相除法得出的一致。

為甚麼輾轉相除法可以成立？這樣便需要從數學的角度去證明了。首先

> 對於任意兩個相異正整數 a 及 b，若 c 同時整除 a 及 b，則 c 亦整除 $a - b$.……*(1)*

證明：

設 $a = cp, b = cq$ (p 及 q 為某一個正整數)

則 $a - b = cp - cq = c(p - q)$

同理,

> 若 c 整除 b，則對任意整數 w，c 定必整除 bw.……*(2)*

證明從略。

現在證明輾轉相除法的數學意義。

設 h 為給定整數 a 及 b 的最大公因數，那麼 h 可以整除 a 及 b。如果我們把 a 及 b 相除，得商 s 及餘數 t，即：

$$a \div b = s + t$$
$$a = b(s + t)$$
$$a = bs + r$$
$$r = a - bs$$
$$(r = bt)$$

當中 s 為「商」，r 為「餘數」。從 (1) 及 (2) 可知，h 定必同時整除 r，即是 h 同時是 a, b, r 三數的最大公因數。又由於 $r < a$ 及 $r < b$，於是必定在某一個時候 $r=0$，即：

$$0 = a_n - b_n s$$
$$a_n = b_n s$$

最後餘數為 b_n，故此 b_n 就是最大公因數了。

2.5 隨機數 (RAN#) 及遊戲設計

隨機數？

在計數機的領域之中，隨機數的按法就是「RAN#」，計數機便會製造一個由０至１之間，小數點後三個位的隨機數。例如「0.234」、「0.555」等等。實驗結果顯示，隨機數最大只曾是「0.999」，但最小可以是「0」，以「0.000」表示，但實際上的數卻是「0」。

但究竟甚麼是隨機數呢？隨機數在外行的人來說，就是一些沒有規律的數字，而且當抽取的結果數量越大，便會有越平均的分佈性。例如你擲銀幣，擲出「人頭」和擲出「字」的機會率，理論上是相等的。如果你在過程之中提出不合理的條件使之其機率不平均，那麼擲出來的結果便不算是「隨機」了。

正如剛才所說，以上的解釋是外行人說的。真正的隨機數是怎樣形成的呢？是不是有一顆骰子在計數機裡面呢？當然不是，事實上，計數機（和電腦）的隨機數是十分科學化地計算出來！只要這種計算「具有良好的分佈性」和「結果不可以輕易推測得到」便可以了。所以我們稱這種隨機數為「偽隨機數」(pseudo-random)。

至於隨機數在程式編寫的步驟 之中起了甚麼作用，說實話，筆者除了在編寫遊戲的過程之中使用過隨機數外，可真的未曾編寫過實用的程式是包括隨機數在內的。但另一方面，在我編寫的遊戲當中，絕大部份也包含了隨機數在內。這是因為遊戲的特點在於其隨機數的神秘性，或者，說得明白一點，在計數機裡玩遊戲的人，很大部份只是在玩隨機數而已。

遊戲概論

甚麼是遊戲？遊戲之中充滿無常 (uncertainty)，但另一方面也體現了某一方面的宿命論。

例如大家在玩大富翁的時候，「不確定」就是擲骰子和策略，抽到不同的機會／大眾寶藏，便會有不同的際遇，在大家也不知道下一步會發生甚麼事的情況下，一起在不確定裡滿足宿命：勝出的人必定家財萬貫，做世紀大地王。勝出的人不是做皇帝，也不會是做英雄。他們在遊戲開始的時候也知道自己的結果，要麼是贏（做大地王），要麼是輸（負債纍纍），但仍是要玩，玩的就是當中的隨

機成份，既享受隨機的過程中的迷惘和不確定，也享受最終成為大富翁／負資產的宿命。

但有時我們並不需要真正無人知曉的「隨機數」，就如「神璣妙算」(Master Mind) 之中，出題者在不同的顏色的「棋子」之中以某種組合為答案，玩者便要在有限的步數裡推斷出題者的組合。出題者在每一次玩者的估計中回答有多少個「正確顏色正確位置」和「正確顏色錯誤位置」的棋子的數量。隨機成份便是出題者的顏色配搭和次序，而這些都是出題者的意願（當然，出題者會盡量把配搭做到隨機）。所以，「隨機成份」並不一定是「天曉得」的。

至於在遊戲中可以微妙地維持著「不確定」和「宿命」之間的趣味，便有賴遊戲隱含的核心：遊戲規則。

何謂「隱含的核心」？簡單一點來說便是「隨機數的應用」。隨機數本身是沒有意義的，加上了遊戲規則的闡釋，便有了意義。你無聊地擲骰子，出來的答案有意思嗎？沒有。但是在大富翁裡，擲骰子便是代表你前進的步數。在飛行棋裡，一開局時擲骰子的數字本身也無意義，但是那個數字的單雙便有意義了：雙數「起機」，而且「三次六，返大陸」。由此可見，在不同遊戲規則之下，同樣的一個數字便有了不同的意義。

所以，在設計遊戲的時候，必須掌握「隨機成份」、「遊戲規則」，和兩者之間的「不確定」和「宿命」。這四個元素加起來，便組成了「遊戲」。能同時滿足這四方面的，便稱得上是一個好遊戲。

至於創製遊戲的步驟，其實和編寫程式的差不多，不過，創製遊戲要把編寫程式的**「原始動機」也要自我激發出來**。

在絕大部份的時候，我們寫遊戲是從「遊戲規則」開始的。腦子裡想著，究竟這個遊戲是讓人興奮地拍打著計數機，還是二人對戰鬥扭六壬？心裡邊先想著遊戲規則和玩法，想通之後便運用流程道編排各輸入及輸出的位置，要是踫著不可能的矛盾情況，便要使用邏輯、其它改良，或者改變遊戲的規則，最後當然是寫成程式和測試了。

對於遊戲的編寫，通常是自我創作的，而且別人聽到可以在計數機裡玩刺激（真真正正使人投入的刺激）的遊戲，會提起興趣。計數機遊戲的編寫是推廣計數機的一個重要步驟，讀者宜加熟練。

第三章：fx-3900Pv 之硬件

3.1 外部硬件

好了！你們已經有了足夠的程式編寫知識，要繼續深造，不得不向外擴展：硬件。

為甚麼要學習硬件呢？因為，作為一個學習計數機的人，既然把計數機的編程學得如斯徹底，那麼，外行的人們必定會認為你是「計數機學者」，甚麼關於計數機的問題也會尋求你的意見。其中一種很常見的問題，便是關於硬件了。筆者曾經遇上大大小小有關硬件的問題，如果掌握了硬件的知識，不但可以協助他們解決各類問題，也使自己成為一個更為全面的「計數機學習者」。

要學習計數機的硬件，當然從外觀入手。

我們先協定：從正面望著計數機，「上面」是指接近螢幕的部份，「下面」指接近「小數點」、「RUN」按鍵。如果是指鍵盤和底部的黑膠，以「前面」和「後面」表示。

由上面開始，你可以看見一個液晶體顯示的螢幕。螢幕上有一塊透明膠片覆蓋著，這塊膠片還覆蓋螢幕之上 “CASIO *fx-3900Pv*” 及各種 MODE 的按法。膠片下面刻著 “SCIENTIFIC CALCULATOR” 字樣。這些字是刻在一塊黑硬膠之上。事實上，這塊黑硬膠有一整部計數機那麼大。在刻字的上面，黑膠和透明膠在同一平面；在刻字的下面，黑膠是埋藏在灰色金屬片之下。換句話說，這塊黑膠的大小和計數機背面的那塊黑膠是相同大小的，大家只要從側面看它的厚度就會明白。為了識別，我稱正面的那塊黑膠為「前殼」，背面的那塊為「後殼」。

在刻字的下面有一條波浪線，那就是灰色金屬片的邊緣。這塊金屬片有四十四個洞，上面寫著各個鍵的 “SHIFT” 功能。正如我剛才所說，金屬片是附在前殼之上的，而且只是利用膠水黏合而成，故此年代久遠的計數機，金屬片都很可能因膠水失效而脫落。這樣就會導致灰色金屬片表面的「H.K.E.A. APPROVED」字樣也會離開計數機。

有一個特徵是十分有趣的：”10+2 DIGITS” 字樣。**所有在日本製造的，字樣也是印在「OFF」之上**，至於馬來西亞製造的，早期出廠的是在 OFF 之上，但很快便把它移至「０」之下。為甚麼要這樣做呢？筆者*估計*，是關於「H.K.E.A. APPROVED」這個考試局印章。

所有“10+2 DIGITS”印在 OFF 之上的，考試局印章都是印在透明膠片之上的（INTEGRATION 上面）；而印在「0」之下的，考試局印章便改印在灰色金屬片的表面。但是，如果金屬片脫落的話，便會連考試局印章也消失了。筆者並不肯定「10+2 DIGITS」和考試局印章位置不同，哪個是因，哪個是果，但可以肯定，兩者互相影響著。

把計數機反轉，就會看到一塊大黑膠，上厚下薄。上半部份刻著“CASIO”、電池電壓、輸出功率、電池型號、出廠地點及電池正負。在左手面有一個“RESET”按鍵。如果用尖的東西按它，計數機就會終止運算，螢幕上出現“RESET”字樣，此時如果你按“RUN”的話，它就會把一切常數記憶及程式全部刪除；如果你按 AC 或者其它鍵的話，常數記憶和程式就不會被刪除，計數機回復普通運算模式。如果硬要這樣說的話，RESET 可以算是「不按 AC 的開機方法」（原因會在下一節裡解釋）。黑膠上面還有一條銀色封條，上面寫著這部機算機的出廠號碼。如非必要，不要把它撕去，因為它在保養時是必要的。

以上便是有關外部硬件的補充了，有沒有想過單是外觀也需要用到這樣的篇幅來解釋呢？下一節筆者帶領大家深入了解計數機的內部硬件結構！

3.2 內部硬件

首先，筆者必須聲明：讀者依照以下指示作硬件裝配純粹自願，筆者並不對任何有關硬件損壞負責，讀者亦必須知道，因人為而造成的損壞並不包括在「保養」之內。

揭開外殼

要了解計數機的內部，必先揭開外殼。在揭開之前要留意好幾點。

心理上必須**保持平靜**，切忌輕率，不要貪快，也不要為了逞英雄而拆機，這樣很容易導致不必要的錯誤；**不要勉強**，遇到不確定的事，應先尋求有經驗的人協助，切忌拿著僥倖的心，「生命冇 Take 2」，計數機也一樣，壞了 CPU 便不可能修復；**要處變不驚**，遇到突發的事件不要驚慌徬徨，做任何動作之前拿最壞的打算，並計劃好應該怎樣做；除非你很有經驗，否則**要有充足的時間**，因為時間的限制可能使你著急起來，結果把機弄壞。

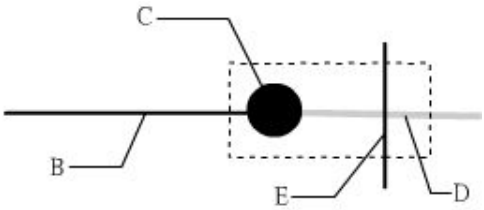
實質上應**找一個整潔、平穩的平面**，整潔固然重要，但有時桌子著地點不穩，可能會左搖右擺，十分影響拆機的過程；**找齊工具**，當然要有一個十字型螺絲批（螺絲刀），可以的話找一個細小的容器，例如盛載油彩的碟子，或者水瓶的蓋子，用來盛載螺絲；**要有一對靈活和清潔的手**，最重要是**乾手**，因為水很容易弄壞靈件；**四週盡量不要有人**，因為他們的一些無心之失會令你大失所措，就算有人在你的附近，也要確定他們不會影響到你的工作。

好了！現在拿起你的螺絲批，把螺絲都拆走！它們的次序可以隨意，也不用記著對應的位置，螺絲是可以互相調轉的。把螺絲安放好以後，**後殼向前推，前殼向後拉**，不用使勁，力度很輕即可，**切忌渾勁！**這是最容易弄壞計數機的過程！

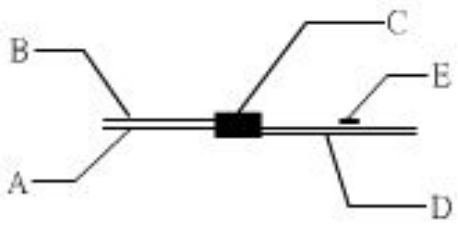
要是你推開了後殼，便可以拿走它了……精美的線路呈現眼前！

線路

整部機的內部是沒有硬的底版，只有一塊軟軟的膠片，膠片上有無數的線路。膠片的右上角是電池，中間閃著發光的就是計數機的腦袋：中央處理器 (Central Processing Unit, CPU)，CPU 上面有一排線路連接螢幕，下面也有一排線路連接各按鍵，那麼，線路是如何接駁呢？



圖一（俯視圖）

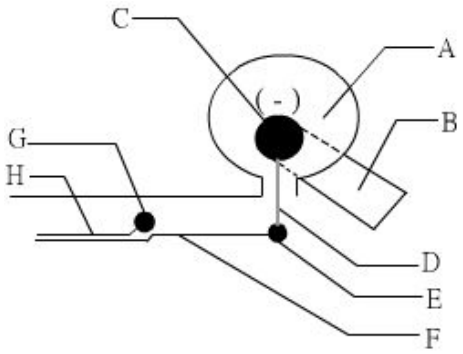


圖二（切面圖）

圖二是圖一虛線部份的切面圖。計數機的透明膠片 (A) 是絕緣體，故此膠片的底面互不通電。線路當然是導電體吧，但是，像計數機那麼複雜的線路，不免有很多相交的地方，然而如果好像我們十字路口一般「平交道」的話，電流很可能會沿著錯誤線路，故此我們運用「膠片底面不通電」的特性來做合適的交叉路。假設電流從左流向右，那麼，當到達「黑點」(C) 時，電流被引導向膠片底層 (D) 繼續前進，而上層則有其它的線路 (E) 相交著。

實際上，如果線路是在膠片的上層（B、C、E）的話，它們之上還有一層導電層，能幫助導電和按鍵。

電池

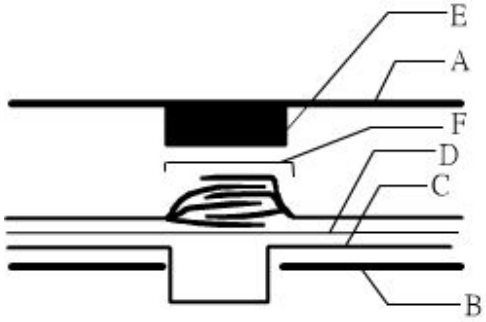


圖三（俯視圖）

電池放在特定的位置之上 (A)，當前殼和後殼是合上的時候，有一塊附在前殼的鐵片 (B) 和在後殼的另一塊鐵片（不在圖）把電池固定著。我們可以從電池的正極開始追綜電流的路線。當前後殼是合著時，後殼鐵片一面和電池正極接觸，另一面和膠片上的一個黑點 (G) 接觸，然後經上層線路 (H) 連接至 CPU。負極又如何？電流從 CPU 出發，在上層經線路 (F) 和黑點 (E) 轉到下層 (D)，然後再經過大黑點 (C) 回到電池的負極去。

由此可見，電池的兩極也是直接連接 CPU 的，這樣可使 CPU 控制開關，也能令 CPU 某些地方長期通電，可以儲存數據。

按鍵



在計數機的前殼 (B) 和後殼 (A) 是合著的時候，橡膠 (C) 和膠片 (D) 是非常緊貼的。每一個按鍵也有一個「蛇形空隙」(F)，在不是按鍵的時候，空隙的左右互不相通。然而，在後殼於極接近空隙的地方有一「導電小圓片」(E)。

當用者按下鍵時，橡膠被壓下，連同膠片也遭壓下，蛇形空隙的兩邊也會接觸到小圓片，使得空隙的左右相通，空隙的左右配搭是固定的，在 CPU 裡收到特定的路線傳來脈沖 (pulse)（或者電流），便可以解碼成內部運算的訊息了。這樣的按鍵原理解釋到好幾個現象：膠片必須是柔軟的；線路不可以穩藏，必須在膠片上導電；不可以胡亂把膠片取出來（因為膠片是在機內以幾個定點，運用膠的塑性來固定，如果把它取出來，會失去其固定性，使某些蛇形空隙長期通電），等等。

蛇形空隙的左右兩邊線路，朝著 CPU 前去。在 CPU 下面會形成一排上下的線路。筆者以最接近 RESET 的為「１」最接近電池為「１２」，排出各按鍵的線路組合：

＼	1	2	3	4	5	6	7	8	9	10	11
12		ENG	+/-	[(---	7	4	1	0			
11		√	1/x	---)]	8	5	2	.			
10	P2	Log	°’ “	Kin	9	6	3	EXP			
9	↑	ln	Sin	C	X	+	=				
8	↓	x^y	kout	÷	-	RUN					
7	OFF	Cos									
6	MODE	MR									
5	Tan										
4	M+										
3											
2											

（注：AC, P1, SHIFT 並不在上表）

第四章：fx-3900Pv 之實驗與表達

4.1 程式表達 (Presentation)

作為一個資深的程式編寫員，儘管寫得好程式，如果沒有加上清楚的使用說明，自己又沒有清楚的記錄的話，一樣是徒勞無功。你遇到編寫程式的問題，數學上解決了，把它寫成程式。但是，程式只是一堆步驟，本身充其量是答案的主幹。所謂「人機溝通」，創造了一部世上沒有人懂得使用的東西，等於沒有。同樣地，我們編寫了程式之後，必須清楚地加上所有操作方法，**使人看了便可以操作到程式**，編寫員應該預備所有有可能的錯誤。例如，如果程式執行期間出現「-E-」便代表某些數據錯誤，使用前應先按些甚麼，等等。這些文字上的附加功夫，我們稱為程式的「表達」。

以上所談及的，是用者所要求的表達，程式的外在使用說明。懂得寫程式的人總會把程式留個副本，記錄數學上和計數機上用到的算式和技巧，也可以記錄各記憶體所代表的數學意義。

以下是筆者的其中一個記錄：

羅盤方位角轉真方位角

(Convert compass bearing into true bearing)

A) 程式

1) ENT	9) 6	17) KOUT 2	25) -	33) KOUT 3
2) KIN 1	10) KIN ÷1	18) ☓	26) 1	34) =
3) ENT	11) 5	19) KOUT 3	27) 8	35) KIN + 6
4) KIN 2	12) KIN – 3	20) =	28) 0	36) KOUT 6
5) ENT	13) 1	21) KIN 6	29) =	
6) KIN 3	14) 8	22) 2	30) ☓	
7) 2	15) 0	23) ☓	31) KOUT 1	
8) KIN – 1	16) -	24) KOUT 2	32) ☓	

B) 注意事項：沒有

C) 用法

執行程式，以「8」代表「北」，「6」代表「東」，「2」代表「南」，「4」代表「西」，順序輸入羅盤方位角的兩個方向和角度，輸出真方位角的角度。

D) 舉例

問題　：N 26°E = ?
步驟　：8 RUN 26 RUN 6 RUN
結果　：26
結論　：N 26°E　= 026°（小心百位「0」）

E) 數學解說

羅盤方位角的角度，是指以正北或正南作為起始點，向正西或正東偏離的一個角度。而真方位角是指以正北為起始點，順時針計算該角度。當中羅盤方位角的角度一定小於 90°，而真方位角的角度就小於 360°。而兩者之間角度的關係可以以下表表達：(設羅盤方位角角度為 x)

情況	北／南	東／西	真方位角角度
1	北	東	x
2	北	西	360 - x
3	南	東	180 – x
4	南	西	180 + x

F) 程式解說

之前的程式是簡化版，以下程式為原始指令：

ENT KIN1 ENT KIN2 ENT KIN3 2 K-1 6 K÷1 4 K-3 2 K÷3 180 + KOUT2 – 2 ╳KOUT2 ╳KOUT3 = KIN6 180 – 2 ╳KOUT2 = ╳KOUT1 ╳ (1 – 2 ╳KOUT3 = K+6 KOUT 6

ENT KIN1 ENT KIN2 ENT KIN3

這幾步把收集得來的數據放入 K1, K2, K3 之中。當中 K1 是「北、南」，K2 是角度，K3 是「東、西」。

2 K-1 6 K÷1 4 K-3 2 K÷3

這幾步把「東南西北」轉成 1 和 0 ，以供之後運算所用。當中「北」及「東」為 1 而「西」及「南」為 0。

180 + KOUT2 – 2 ╳KOUT2 ╳KOUT3 = KIN6

根據「數學解說」的附表，這一步首先符合了情況 3 及情況 4。

180 – 2 ╳KOUT2 = ╳KOUT1 ╳(1 – 2 ╳KOUT3 = K+6 KOUT 6

如果是在情況 1 和情況 2 的時候，這段程式就會起作用，如果是在情況 3

及情況 4 時，則不會有任何效果。這是因為在 “3” 及 “4” 時，K1 是 0，和所有數據相乘後必定得到 “0”，加進 K6 之中，故此沒有更改 K6 之值。由於情況 1 和情況 3 、情況 2 和情況 4 的「東西」一樣（情況 1 和情況 3 都是「東」，情況 2 和情況 4 都是「西」），故此筆者視之為兩組。而且：

情況 3 + (2x – 180) = 情況 1
情況 4 + (180 – 2x) = 情況 2

所以，先取 180-2x 的值，然後以「南北」決定應否更改答案，以「東西」決定應否將 180-2x 乘以 –1。
把最後結果加進原先計到的數，就是答案了。

E) 運作時間

少於一秒

F) 誤差及限制

誤差少於有效數字第十個位之

4.2 實驗

科學，離不開實驗。大膽假設，小心求證，才是做學問的最佳方法。做實驗就是求證的方法，歸納求證得出的結果，再進行新的猜想，便是「進步」了。做實驗，和做科學實驗一樣，要寫下報告，才能有系統地找出答案，日後再翻查之時，也能對理論、理念、猜想提出有力的證明或反對。

筆者認為，實驗報告應包含以下的幾項：

１）基本假設及基準

每一個實驗都有一些基本假設，例如：假設按 “1” 字後，CPU 就接收到 “1” 這個信息，這個基本假設很合理，於是你按甚麼鍵，CPU 就接收到那個鍵的信息。但如果，日後發現原來事實並不是這樣，你按 “1” 字，原來 CPU 並沒有接收到 “1” 字的，（我只是說個例子而已）那麼，整個實驗也是徒勞了。

２）清楚列明步驟，以便日後翻查

３）寫下結果

４）對結果進行分析，得出結論

５）為結論作出新的猜想

６）寫出實驗的缺點

實驗的方法，不外乎編寫程式來做多次重覆實驗，但是如果你認為在程式裡重覆運行得出的結果，和手按的結果有所不同的話，便不可以編寫程式了。至於要做些甚麼實驗，比較普遍的有以下兩個：

速度實驗：即使是同一型號，同時期出廠的計數機，其速度也有很大的差別。筆者試過為學校集體訂購 *fx-3900Pv*，並為所有計數機作速度測試，但最慢的可以比最快的慢超過三成！而且速度測驗還有一個很歷史性的使命：創製「通用計時程式」。正是由於每一部計數機的速度也不同，我們需要一個通用的方法來寫出計時程式，無奈至今，筆者仍未找到解決辦法。

隨機數實驗：筆者在上一章已討論過隨機數的原理，如果你計算極多數量的隨機數，其平均值也有明顯的偏差：程式測試一萬個隨機數的平均值，從 0.495 到 0.508 也有。可是，如果做「隨機數分佈實驗」的話，便會發現它的分佈是極之不平均的。好奇的你也許會覺得室溫可能影響到隨機數的分佈和數值，你也要做實驗來證明你的猜想是否正確。

以下是一個有關速度的實驗作為例子 (*斜體字*是補充語句)：

測試基準： 1) 人手測試，以程式遞歸的方法，執行三十秒。停止其執行並統計其回歸次數。重覆以上步驟一次，並取其平均值即為該次測試的最後數據。單位為「次／秒」(Hertz, Hz)

2) 假設一部計算機對於「基準測試」的速度，和所有其它功能鍵的速度是成正比例的。*(這其實是需要證明的)*

測試一：基準測試

編寫程式：

1 K+1 RTN

記錄其測試速度。

第一次：100.77 Hz *(其實只做兩次取平均值不太準確，宜做四次)*

第二次：100.20 Hz *(第一次和第二次的結果必須相近)*

平均值：100.485 Hz *(如相差太大，結果作廢*

∴標準運算頻率 (Standard Running Time Frequency, SRTF) 為 100.485。

(SRTF 其實是「基準測試」的實質運算速度，以「次每秒」為單位)

測試二：求 RTN 的標準運算時間系數。(Standard Running Time Coefficient, SRTC)

(SRTC 是一個公認的數，每一個指令只有一個 SRTC。筆者假設每一個指令的 SRTC 和其 SRTF 成正比，從而輕易計出該指令的實際執行時間。而 SRTC 的值是假設在 SRTF = 100 的計數機上執行的時間。)

以同一部計算機測試，編寫程式

1 K+1 1 RTN

第一次：72.43 Hz

第二次：72.40 Hz

平均值：72.415 Hz

將此數與「測試一」之結果比較，得出執行 “1” 所需時間為：

$$\frac{1}{\frac{1}{72.415}-\frac{1}{100.485}} = \textbf{259.23 Hz}$$

以同一部計算機測試，編寫程式

1 K+1 K+1 RTN

第一次：68.23 Hz

第二次：68.60 Hz

平均值：**68.415 Hz**

將此數與「測試一」之結果比較，得出執行 “K+1” 所需時間為：

$$\frac{1}{\frac{1}{68.415}-\frac{1}{100.485}} = \textbf{214.36 Hz}$$

將三項測試拼合起來，得出 RTN 的執行時間為：

$$\frac{1}{\frac{1}{100.485}-\frac{1}{214.36}-\frac{1}{257.98}} = \textbf{709.01 Hz}$$

最後，RTN 的 SRTC 為：

$$\frac{1}{\frac{1}{709.01}\times\frac{100.485}{100}} = \textbf{705.59 Hz}$$

同樣地，各個功能的 SRTC 都可以被計算出來。如 “1” 的 SRTC 為：

$$\frac{1}{\frac{1}{259.23}\times\frac{100.485}{100}} = \textbf{257.98 Hz}$$

問題：若一計算機，其 SRTF 為 111.95，則此機執行 RTN 需時多久？

SRTC of RTN = 705.59

SRTF = 111.95

執行 RTN 之實質頻率為：

$$\frac{1}{\frac{1}{705.59}\div\frac{111.95}{100}} = 789.91 \text{ Hz}$$

∴在此計算機執行 RTN 需時$\left(\frac{1}{789.91}\right)$秒。

總結：求出數個 SRTC：

“1”=257.979

“K+1” =213.325 (註：連運算時間在內)

“RTN” =705.59

總結

四年了。

四年前，我接觸計數機，四年後，我寫成了這篇修訂本。許多人和事，我也不多說了。要多謝的人太多，支持過我的人太多。哥哥借給我第一部計數機、「家姐」給我沿途的鼓勵、Desmond 發明了「全方位倉頡」使我把近三萬字輸入電腦也不覺費力、無數網友給我的支持和啟發……

這一次的修改，我猜是最後的一次了。因為，要加進去的內容真的不多。而且，計數機種類繁多，這一本教材是為專門深究 *fx-3900Pv* 的人而寫。作為一個學習者，應該向多方面發展。

若然讀者學成之後能活學活用，筆者很滿足；學成之後能協助他人解決問題，筆者很高興；學成之後能教導他人，使他們也學會計數機知識，筆者感到無上的光榮。

筆者希望讀者透過「計數機」這項工具，拉近人與人之間的距離，使人更外向更活潑。在解決問題的同時，必須緊貼用者的要求，當中的溝通過程便是計數機知識以外所得到的經驗。而筆者的最大心願，是讀者透過計數機的思維方法，用在數學、電腦、科學等等其它領域之上，更能發揮計數機的力量。然而，只有這樣，才能把所學到的完全展露出來。

願與各位共勉之！

初版：二零零零年十一月

再版：二零零二年三月

第一次修改：二零零二年四月

第二次修改：二零零二年五月

第三次修改：二零零二年八月

再版修訂本：二零零二年聖誕節

〈全文完〉