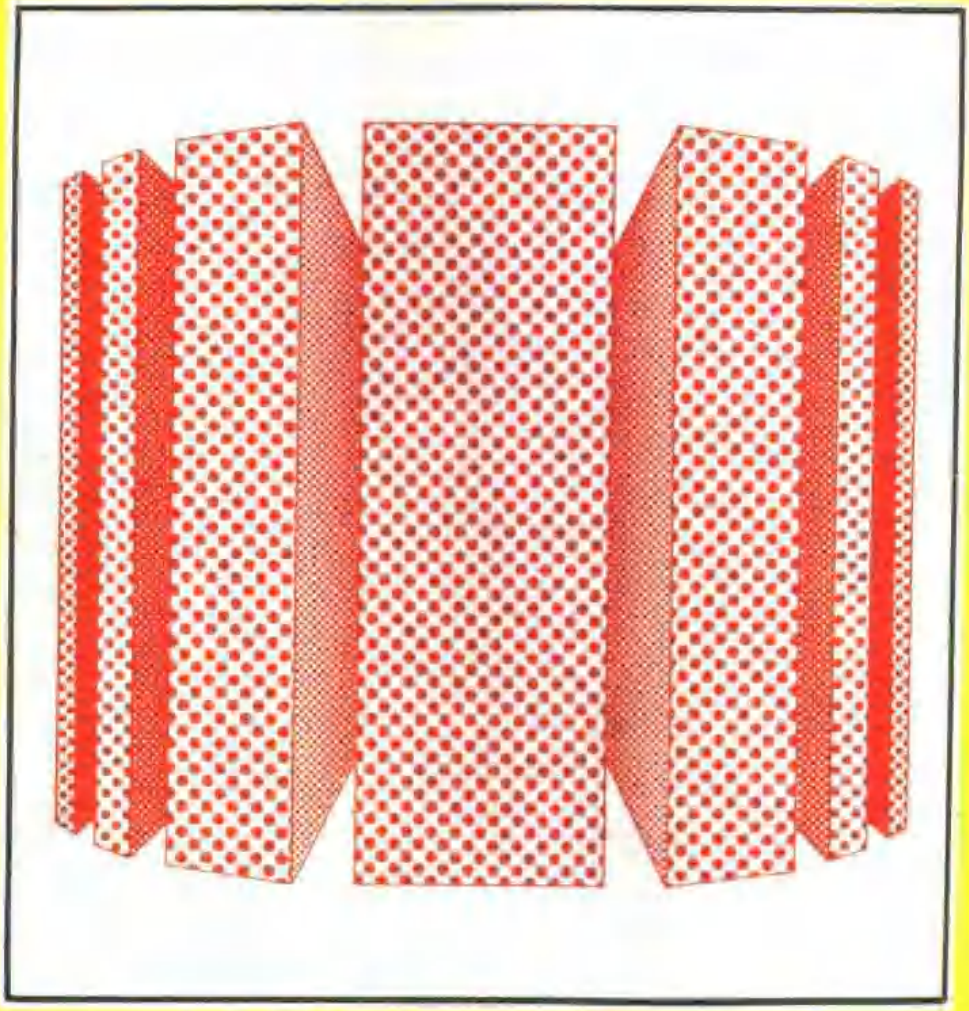
系統程式計劃概論

王學智 ■譯編



力新出版社

系統程式計劃概論

編譯・王　學　智

力新出版社

系統程式
計劃概論

編譯者：王　學　習
出版者：
發行者：力新出版社
九龍新山道四三八號
承印者：力行印刷公司
香港柴灣保興工業大廈8樓A座

定價：港幣

前言

筆者執筆寫本書的主要動機，是要讓讀者站在本身自修的立場，以有系統的方法構成演算法，並予以定式化的技巧或技藝來介紹程式計畫的。廣義來說，所謂演算法就是一種對資料處理與控制過程之群集的「步驟書」。我們必須把演算法當做屬於邏輯性的、可靠性高的，很巧妙地設計的構造堅固的建築物來了解它。

希望讀者勤勉學習，實地演練在程式計劃上屬於比較典型的，但可不是要依靠應用領域的問題與技巧，藉以有所進展，進而能設計有系統的演算法。基於這個理由，目標不放在特定的應用領域。對於習題或例題之選擇，是使其能當作一般通用的問題與解決法的實例。以同樣的精神，對於表記法，亦即程式計劃語言，則不作深刻的說明。語言雖然是一種工具，但其本身卻不是目標。我們不能把程式計劃教育，以教授有關特定語言的性質或特徵相關的知識或使用方法爲主要目的。毋寧說，語言必須對演算法的比較基本，却是最重要的觀念，予以明確、自然而且容易了解那樣反應才對。此外，對於語言，同時也必須考慮到數位電子計算機本質上的性質與限制。

關於程式的可靠性問題，筆者是致力於說明程式「證實」的基本構想與技巧。總而言之，程式計劃的目的，是在於要把計算的程序全體當作演算法來定式化的。設計者唯有能夠證明他的作品在任何狀況之下，都能按照規格書那樣發揮功能，良心上才能安心。可是普通所使用的程式測試（除錯）的方法，只是要檢查各計算而已，並非"精密地"檢查程式所執行的整個計算。爲了要提出和程式本身相

關的一般“主張”，解析方式的「證實技巧」是不可或缺的。

即使這是初步的，只要能處理證實的方法，則讀者只需比過去的程式計劃教育稍微高一點的抽象能力就可派上用場。因此，要把這話題編排於比較初步的教科書的這一見識，過去是被保留下來的。可是，筆者認爲本書所採用的基本概念—雖然限定於“（不變的）主張”的相關構想—在本質上有其重要性，因而堅持在本教科書一開始就詳細說明。同時，我也反對應把這話題說給最偏重理論的人了解的這個意見。程式的可靠性之所以重要，就是在臨場，而不是理論。程式的證實這一個觀念，是爲更加深了解演算法所需的基石，缺少這個基石，程式設計師勢必僅把不充足的自己的直感當作工具不可。

本書是爲了接受演算法的系統性結構相關的授業爲基本的數學性訓練者爲目的而編輯的，而不是爲能暫時對問題寫碼，立即使用計算機獲得答案的人士所寫的。

程式計畫語言ALGOL 60 雖然已透過本書成爲表記法的基本，但不是將ALGOL 60按照字面原本本採用的。這是因爲今天的計算機—其程式計劃也隨之—比十二年前更廣泛地運用，而程式計劃的入門，不應該僅限定於單一應用領域的緣故。ALGOL 60主要是爲能適合數值計算而設計的。因此，即使要藉它來撰寫其他領域的程式，則更應根據別的構成原理才恰當，因而使得更與此不同的概念成爲中心，往往會誤用語言。於是，凡是不符合目的的工具，既絕對不能使用在教育，而且也不能推薦把它模仿作爲一種良好的例子。

筆者之殷切盼望使用可明確而具有系統地表現計算過程式資料結構的表記法的教育，是因爲筆者看到差不多的人對於剛開始學的語言是一成不變地永久抱往它不放的緣故。已經說過好幾次，這種傾向不僅是來自人傾意識性的慣性，卻也是基於“開始的”語言，能提供抽象性的思考，以便於取得較具體的形態的這種框架的事實。藉這一次

的語言，我們除了學會詞彙與文法規則之外，也即開了新思考領域的大門。因此，我們必須小心翼翼愼重地選擇此語言。不幸的是，目前最廣泛使用的計算機語言，太不符合乎邏輯性，及系統性所需要的條件。

在本書所必須預備的知識，就是屬於較初步的數學。尤其是讀者最好能親近命題邏輯的基本與數學歸納法。解析學幾乎不需要但仍會出現於若干例題與習題中，不過把它跳過去不唸也無妨。

讀者最重要的就是要積極地做習題。程式計劃在本質上是要進行結構與統合的學習，若耽於瞑想，則無法竟成。筆者認爲不必連細節都嚴密的規定，能明確地定式化的一易解的問題才恰當。目的或解答的形式，即使不採用複雜的數學之形式論，也應該是明快才對。讀者也許可從習題進而應用到所說明的概念或技巧。習題最好不要在獲得解法以前花費太多時間，必須花上許多時間或經驗的比較屬於技巧的難題上。每章末的習題既完成了身爲實例的任務，或許也能夠以各種方法加以變化。

程式計畫教育之是否成功，有賴於計算機中心的組織。如果不能符合某最低的條件，其教育可能立即以失望與挫折而終。首先，必須使用能以短時間退回的工作那種計算機。如果僅僅把處理裝置佔有數秒鐘時間而輸出數 10 行輸出的任務，要花上 15 分鐘以上，就沒有用處了。第二，編譯程式無論在任何情況時，必須輸出容易懂的響應。尤其是一個初學者對這些響應絕對不會得到所盼望的計算結果，而大多數是錯誤的資訊。編譯程式必須輸出藉自然語言或在使用中的程式的表記法所寫的訊息。絕對不應該輸出不明確而意圖不明的操作系統的訊息或一最壞的—8 進或 16 進的傾印。同時，操作系統的命令也必須抑制爲所需要的最低限度。

本書是根據 Stanford 大學與 Zürich 的 Federal Institute of Technology（ETH）所用的講義上筆記本編輯的。因此不可能把

對本書有貢獻的人與事或有益的構想全部都一一致謝。話雖然這麼說，特別衷心感謝我的伙伴E. W. Dijkstra (Eindhoven), C.A.R. Hoare (Belfast),及P.Naur (Copenhagen)。他們不但對本書有所貢獻，同時程式計劃的一切話題也對我有很大的影響。筆者也謹致謝意，希望回想起和H. Rutishauser所作的議論。他是程式計畫語言觀念的創始者，是ALGOL 60的著者之一，對本書的影響可以說最大。A. Forsythe不厭其詳唸本書的草稿，在這裡敬致謝忱。最後，對於程式計劃語言PASCAL的編譯程式製作不遺餘力携手相助的共同研究者，U. Ammann, E. Marmier, R. Schild也在這裡敬致十二萬分的謝忱。由於他們的努力，本書所採用的表記法不但得以表現了抽象性的演算法，同時，亦實際證實了適合於用在計算機上的，效率良好而可靠性高的程式。

Niklaus Wirth

系統程式計劃概論目錄

第 1 章

緒論

這十年來，計算機在商業、工業、科學研究方面，已經成爲完成原是絕不可能做到的艱難工作所不可或缺的工具。計算機雖是一種規規矩矩按照規則進行計算處理的自動機器，但凡是“能夠了解，”能夠遵從的基本指令，通常非常有限。可是，這些指令却令人驚訝地迅速而且精確地執行。計算機之強而有力，廣大應用範圍的本質，是在於它能夠無窮盡地組合較基本的演算，及具備了能執行很長的指令之列的能力。把這種指令的列歸納起來，製作用以表現計算過程的某級的“步驟書”的行爲，叫做**程式計劃**（ programming ）。可是，要設計程式時，即使不使用計算機本身，對於基本構想，既能夠說明，也能夠了解。

程式計劃是“才智的磨練”，可供各種應用。其一，既擺脫了許多複雜的問題所帶來的數學上的解析，有系統性的方法之路，尤其是其中就是一種知性的“戰爭”。程式計劃幾乎未曾當作組織性的技巧

解析，其理由如後所述。程式計劃雖然和有趣的應用或挑戰性的問題有聯帶關係，但是，卻必須藉很可靠的處於理論的基礎與有系統歸納起來的方法，而且在程式變成（有數千或數百萬指令的這種）相當複雜與長度時，才會產生其必需性的緣故。計算機尚未出現之前，並不存在將這種長的指令之列，在絕對性服從的情況下，確實地執行的，或者能夠執行的“奴隸”。因此，要寫這種程式的動機也就不存在了。由於有了現在的數位電子計算機，才使程式計劃除具挑戰性之外，也具有面臨問題的意義。

第 2 章

基本概念

本書將引進有關程式計劃的若干重要概念。這些概念都是屬於基本概念，所以，無法使用其他概念作形式上的定義。不過，筆者仍然要舉出若干例子來說明。

最重要的概念，就是**作用**（action）的概念。在這裏所謂一個作用，會帶來在有限的時間內正正確確地被定義的**效果**（effect）。各作用要求該作用所發生功能的對象的存在，與認識該狀態變化的效果。同時，各作用必須是在某**語言**（language），換句話說，在某邏輯體系中被記述。該記述稱爲**敍述**（statement）。

作用能分解成若干部分時，叫做**處理**（process）或**計算**（computation）。如果這些部分是在時間上井然有序，而不致於有二個部分被執行，則該處理就叫做是**順序性的**（sequential）。因此，用來記述處理的敍述也能分割成若干部分。這時候，此敍述叫做**程式**（program）。程式雖然由敍述群集而成，但是，敍述之被寫的順序

，在時間上看來，未必和所對應的作用之順序一致。

按照所定的敍述，實際執行作用的推動力稱爲**處理機**（ processor ）。這句有點模稜兩可的名詞，可不是特別地指定用以控制此執行的是人或者是自助機械。事實上，只要對程式加以定義的語言不拖泥帶水，則即使不以特定的處理機爲前提，程式仍然有其意義。程式設計師通常對於要 —— 指定某處理機這件事不感興趣 。程式設計師所要的就是能夠保證處理機，能了解自己所寫的程式的語言即可。這是因爲，他們認爲程式是指定了處理機動作之規則的緣故。因此，程式設計師必須了解自己所能使用的處理機，而且也必須知道能執行的敍述種類，因此，必須設法使自己的語言適合於這些處理機。

無論任何作用，都因應處理機而有一定的工作量。此工作量能夠當作處理機執行作用的時間來表現。於是，此時間通常直接用來估價費用。有經驗的程式設計師也許會經常考慮自己所能使用的處理機的性能，選擇可以使費用降低爲最低限度的解決法。

本書主要是和由自動性的處理機（亦即計算機）所執行的程式設計相關的，所以，在本章，以下讓我們概觀一切數位電子計算機共同之基本特徵。在還沒說明計算機的鳥瞰圖之前，茲舉兩個簡單的例來說明上面所定義的概念。

例：乘法

乘二個自然數 x， y，試以 z 表示其積。

假設有某處理機能了解此敍述。亦即，假設了解“自然數”或“乘以”的意思。於是，就不必再做些什麼事。

爲了要繼續議論起見，對於能使用的處理機，讓我們作下面的假設。

(a) 假設不能了解自然語言的敍述，而只能接納某種的式。

(b) 不能進行乘法，而只能加法。

首先，第一個要注意的就是計算的對象是自然數。可是，程式卻無法直接把這些數本身規定下來。亦即，會一般地規定，對「自然數的任意之對進行乘法的」處理之動作的圖形。這裏，我們不使用數，而是使用名叫做**變數**（variable）的用以表示會變化的對象之一般性的名字。各處理之開始時，特定之值非指定於這些變數不可。此（assignment）在藉計算機所執行的計算之處理是最基本的作用。

變數好比黑板。其值若有必要，既可以看（“讀”）好幾次，也可以抹掉，也可以寫。但是，如果要重新寫的話，失去前面之值。要把值w指定於變數v，往後是寫成

$$v := w \tag{2.1}$$

記號：=叫做**指定演算子**（assignment operator）。

若將前面的敍述予以定式化來寫，則可以寫成

$$z := x * y \tag{2.2}$$

此敍述如果能分解成以時間性按一個一個地排列的加法之列，那麼乘法之作用就變成順序處理，而敍述（2.2）就取程式的體裁。以眼前來說，此程式也許能定式化如下面所述。

步驟1： $z := 0$

$u := x$

步驟2：試將下面的2個敍述反覆執行之 (2·3)

$z := z + y$

$u := u - 1$

至u = 0爲止但是反覆

x，y有各值時，因應計算之進行記錄被指定於變數u，z之值，使能夠看到藉此程式所引起的處理。假設$x = 5$，$y = 13$，則可得表（2.4）。

如果變成$u = 0$，則此處理藉步驟2終止。在這個時點，$u = 0$變成最終性的結果$65 = 5 * 13$。這種表叫做**軌跡**（trace）。這些

值雖然按順序登記於表。但是，我們必須注意，這些值並不是全部都繼續地保持。亦即，各變數在一時點，正好僅保持一個值而已。這是根據指定是在變數的以前之值上面有“重新再寫”的事實有以致之。

步驟	變數之值	
	z	w
1	0	5
2	13	4
2	26	3
2	39	2
2	52	1
2	65	0

（2.4）

現在計算的對象是數。爲了要對特定的數實徹執行演算，則非以某表記法表現這些數不可。要選擇表記法，是在計算之執行時不可避免的。（可是，程式本身通常不賴以對象的特定表記法或表現而有效。）此外——即使如數這種抽象性的對象也是——要區別對象與其表現這一點是重要的。譬如說，數通常是在計算機內當作磁性的記憶元件的狀態表現。可是，要藉計算機計算數的處理而不依賴磁性的狀態也能夠定式化。這是很理想的。現在以這種觀念之例，讓我們看一下不同的表現法究竟如何記述一樣的計算處理。下面的表（2.5）是將表（2.4）之值以羅馬數字調換過來的。

步驟	變數之值	
	z	w
1	0	V
2	XIII	IV
2	XXVI	III
2	XXXIX	II
2	LII	I
2	LXV	0

（2.5）