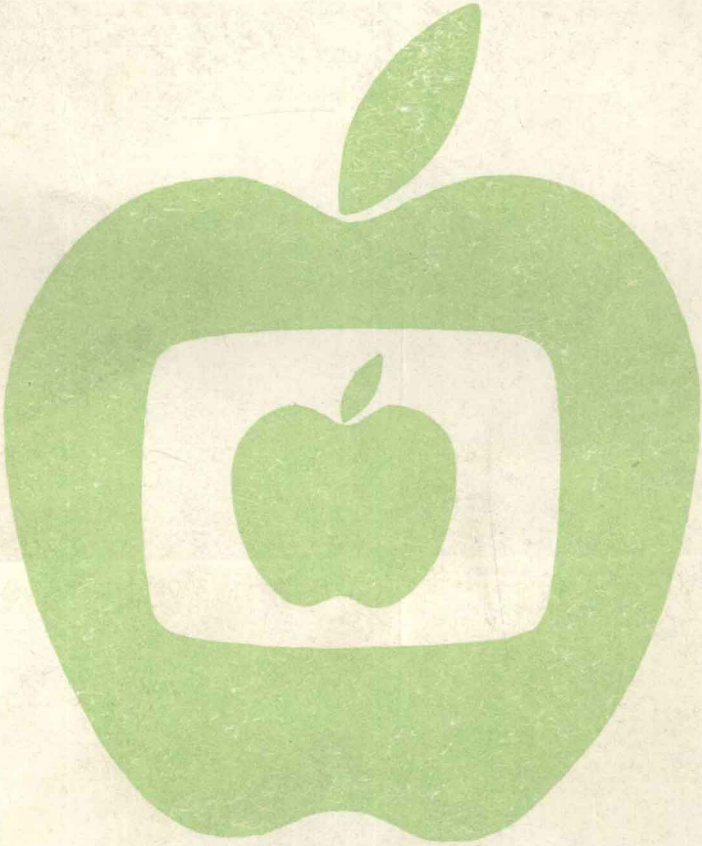
apple Ⅱ
程式语言



北方电脑公司信息资料部

C 程式語言

張宗發
張照煌 譯

原 序

C是一種通用性程式語言，特徵是簡潔的運算式、合乎時代潮流的控制流程和資料結構；以及豐富而多功能的運算子（operators）。C並不是所謂「極高階」（very high level）語言，也不是「龐大的」語言，更不是爲特定的應用而設計地。然而，對許多方面的應用而言，C的限制少，並具通用性，比許多號稱其多功能的語言，要來得方便而有效。

C語言的創始者Dennis Ritchie 最初用這個語言在DEC公司PDP-11 機器上設計UNIX作業系統，並製作其編譯程式。整個UNIX作業系統、C編譯程式、和幾乎全部的UNIX應用程式（包括編輯本書的所有軟體）均以C語言設計撰寫而成。許多其他廠牌的機器，也已經有C編譯程式產品問世，其中包括 IBM系統／370，Honeywell 6000，以及Interdata 8/32。C具有不依賴特定的硬體或系統的特性，在某一種機器上設計的C程式，大多數可以不經任何修改，就能在擁有C編譯程式的另一種機器上執行。

本書的主要目的在使讀者知道怎麼利用C撰寫程式。主要的內容包括一章指引性的介紹，使初學者可以很快地開始用C撰寫程式，此外對C的各種主要特性均闢有專章介紹，最後則是附錄中的參考手册。大部分的討論均以研讀、撰寫、並修改程式實例的方式進行，而不是只敍述其規則。大部分的程式實例均爲完整而可用的程式，而不是孤立的片段。而且所有的例

子均曾以書中出現的型式加以測試無誤，也就是說這些例子均可直接由機器讀入、編譯、並執行。除了介紹如何有效地利用C撰寫程式之外，我們也附帶說明一些有效的演算法（Algorithms）、較佳的程式體裁（style）原理、及好的設計方法。

本書並不是程式設計的入門手册；讀者必須先具有一些基本的程式設計觀念，例如變數、指定敍述、迴圈、及函式（functions）等。雖然如此，初學者還是可以經由本書的閱讀而學會這種語言。當然，請教一下有經驗的程式設計師是非常有幫助的。

在我們的使用經驗中，C是一種易於使用、具表達性、並具多功能的語言，適用於多種類的程式。C是相當易學的，當經驗與時俱增時，使用上更顯得方便。作者希望這本書可以使讀者善用這個語言。

感謝許多朋友和同仁的批評及建議，使本書更加完善。尤其是，Mike Bianchi、Jim Blue、Stu Feldman、Doug McIlroy、Bill Roome、Bob Rosin、和Larry Rosler 均曾仔細地看過本書各版。另外還要感謝Al Aho、Steve Bourne、Dan Dvorak、Chuck Haley、Debbie Haley、Marion Harris、Rick Holt、Steve Johnson、John Mashey、Bob Mitze、Ralph Muha、Peter Nelson、Elliot Pinson、Bill Plauger、Jerry Spivack、Ken Thompson、和Peter Weinberger 諸位在本書撰寫過程各階段中所做的建議，此外Mike Lesk 和 Joe Ossanna 兩位在排版方面提供了非常重要的幫助。在此一併致謝。

Brian W. Kernighan
Dennis M. Ritchie

目 錄

第零章

簡　介

C是一種通用性（general-purpose）程式語言，它和UNIX系統的關係十分密切。因爲C語言就是在該系統發展出來的，而且UNIX系統本身和其附屬軟體都是以C撰寫而成。雖然如此，這個語言並非僅適用於某特定作業系統（operating system）或特定機器；由於C可用來撰寫作業系統，有人稱之爲「系統程式語言」（system programming language），但是C也同樣適用於撰寫數值運算、文稿處理（text-processing）、和資料庫（data base）方面的程式。

和其他語言相比較，C是一種相當「低階的」（low level）語言。這個語法絕非貶損，只不過是表示C所處理的對象（比方說，字元、數值、位址）和原始機器類似而已。因此C可以直接使用原始機器的算術和邏輯運算指令來處理這些對象。

C並不提供對整個合成物件（composite object），如字元串（character string）、集合（set）、串列（list）、和陣列（array）的整體運算。例如，C並沒有如PL/I中處理整個陣列或字元串的運算。除了靜態定義（static definition）和函式中區域性變數（local variables of functions）所用的堆疊（stack）原則之外，此語言並不提供任何

儲存體（ storage ）安置功能：亦即沒有如Algol 68 中所提供的動列（ heap ）和無用位置收集法（ garbage collection ）。此外，C本身也不提供輸入輸出功能，它沒有 READ 或 WRITE 敍述，也沒有現成的檔案存取方法。所有上述的高階功能均須經由明白表示的函式呼叫完成。

同樣地，C僅提供一些直接的、單線的（ single thread ）控制流程構句（ constructions ）：測試（ tests ）、廻圈、群集（ grouping ）、及副程式等，而沒有多程式的（ multiprogramming ）、並行的（ parallel ） 運算、同步的功能（ synchronization ）、和互常式（ coroutine ）。

沒有上列特性似乎是很大的缺陷（讀者或許會問：「難道連比較兩個字元串，也須要呼叫另一函式？」）但是這也使C語言本身的複雜度變小，而產生了某種利益。由於C語法（ syntax ）簡單，其編譯程式（ compiler ）所佔儲存空間小，使用者學起來也較爲迅速，C的編譯程式簡單而密集（ compact ），並且容易撰寫；利用最新的技術，新機器的C編譯程式可以在一兩個月內完成；由於大約百分之八十的程式碼和已有的編譯程式重覆。因此這個語言有高度的可攜性（ portability ）。C的大部分資料型態和控制流程均可直接爲現存的機器所接受，所以要製作自含（ self-contained ）程式所須的執行時程式館（ run-time library ）相當小。舉例來說，在 PDP-11 上原就提供了32 位元的乘法和除法運算，以及副常式進出順序的處理。當然，各編譯程式的製作，均提供許多相容的（ compatible ）程式舘函式，藉以完成I/O、字元串處理、和儲存體分配功能；使用時可以明白的呼叫這些函式，不須要時根本就可以不管其存在。這些程式舘函式本身也可以

利用C撰寫。

正因爲C充分利用了現代機器的能力，用C撰寫而成的程式非常有效率，這使組合語言的需要性降低。UNIX作業系統就是一個最明顯的例子，UNIX中幾乎所有的程式都是以C撰寫而成。此系統核心部分總共有13000行系統程式碼，而其中僅有最低階的800行是以組合語言寫成的。此外，幾乎所有的UNIX應用軟體也是以C寫成的；大多數UNIX的使用者（包括原書作者之一在內）甚至可以不懂 PDP-11 的組合語言。

雖然C的許多特性和原始機器相似，但並不依賴任何特定的計算機結構（ architecture)使用C語言，只要稍加注意，很容易地就可以寫成「可攜性的」程式，也就是不用改變就可以在不同硬體上執行的程式。目前在作者所接觸的環境中，在UNIX上發展出來的軟體可轉移到當地的Honeywell 、IBM 、和 Interdata 系統使用。事實上，這四種機器上的C編譯程式和執行時的支援程式，較ANSI 標準版的Fortran 還要具相容性（ compatibility)。UNIX作業系統本身現在可以在 PDP-11 和 Interdata 8/32 上運作。除了一些和機器相關的程式，如編譯程式、組合程式（ assembler)、和除錯程式（ debugger ）之外，兩種機器上的C程式是完全相同的。UNIX系統，除了組合語言支援程式和I/O裝置處理器（ I/O device handler ）之外，大約有7000 行程式在兩種機器上是完全相同的（佔百分之九十五）。

對於一些熟悉其他語言的程式設計師，我們有必要介紹一下C語言歷史上的、技術上的、和哲學上的由來，以供參考、對照、和比較。

C有許多重要的觀念得自一種歷史悠久、且相當重要的語

言BCPL；BCPL是由Martin Richards 發展出來的。其間還存在一層間接的關係，1970年Ken Thompson先在第一代UNIX系統（所用機器是PDP-7）上發展出語言B，而B後來才發展成C。

雖然C和BCPL有許多共同的特性，但C絕不是BCPL的同系語言（dialect）。BCPL和B均爲「無型態的」（typeless）語言：僅有的一種資料型態，就是機器字（machine word），要存取其他型態的資料，則必須透過特殊的運算子或函式呼叫。而C却有多種基本型態：字元、各種大小的整數、和浮點數（floating point numbers）。此外，還可以利用指標（pointers）、陣列、結構（structures）、組合（unions）、和函式等建立不同層次的導出（derived）資料型態。

C也提供設計良好結構化程式所須的各種基本流程控制構句：敍述的集群、分叉的決定（if）、先決定結束與否的廻圈（while，for），和後決定的廻圈（do）、和選取幾種事例（case）之一的敍述（switch）等等。而BCPL也提供上列各種敍述，雖然語法稍有不同；顯然BCPL在數年前就預知了所謂「結構化程式設計」（structured programming）的流行。

C提供了指標和位址運算的能力。函式引數（argument）的傳遞法是抄錄引數值，因此被呼叫的函式不會改變呼叫函式中的眞實引數值。若須要達到「以址呼叫」（call by reference）的功能時，可利用傳遞指標達成，使被呼叫函式可以改變指標所指到的資料。陣列名稱被當做陣列開頭的位址傳遞，因此陣列引數事實上是採「以址呼叫」的。

各函式均可以遞廻呼叫（recursively），函式的區域性

變數（通常是「自動的」，automatic）在各次呼叫都會重新建立一份副本。函式不可以包含另一函式，不過在各區段結構（block-structure）均可以宣告變數。一個C程式裏的各函式可以分開編譯（separate-compiling）。變數可以宣告成屬於函式內部（internal），屬於函式外部但只在一檔案使用，或完全總體性（global）。內部變數又可以分成自動的和靜態的兩種。自動變數可以存放於暫存器（register）中以增加效率；不過暫存器宣告只是對編譯程式的一項暗示，並沒有眞正使用到機器中的特定暫存器。

C並不是像Pascal或Algol 68 那樣強烈型態化（strongly-typed）的語言。C允許相當程度的型態轉換，但也不像PL/I那樣可以隨便的自動轉換型態。現存的C編譯程式並未提供陣列註標和引數型態的執行時檢查。

如果因特殊需要，必須做強烈的型態檢查時，可以使用另版的編譯程式。UNIX有這麼一個符合要求的程式－lint，此名稱顯然是由於它從輸入程式挑出一些小毛病而得。lint並不產生程式碼，只專門用來檢查程式在編譯、載入時可檢查出來的小毛病。lint可偵出型態的不一致，引數用法的不配合，無用或未給初值的變數，潛在可携性的困難等等。一個程式如果可以通過lint而沒有任何警告，則應該（還是有少數例外）可以如Algol 68 程式一樣，沒有型態錯誤了。有機會時，本書還會隨時提到lint的其他功能。

無可諱言地，C也和其他的語言一樣有其缺點存在。包括某些運算子的優先權（precedence）應重定；部分語法可以修改得更好；現存的各版C語言有一些小差異。不過到目前爲止，在許多程式應用上，C仍是非常有效並富表達力的語言。

本書以後的內容如下：第一章是C核心部分的指引性介紹，目的在使讀者可以儘快的具有用C撰寫程式的能力。因爲作者認爲學會一種新語言唯一的方法就是用這個語言去寫程式。這個介紹已先假設讀者有一些基本的程式設計概念；比方說，我們不解釋何謂「計算機」，也不解釋如n = n + 1這種簡單運算式的意義。有機會的話，我們會介紹一些有用的程式撰寫技巧，不過本書並不適用於當做資料結構或運算法方面的參考書；因爲主要的內容還是集中在語言的介紹。

第二章到第六章更加詳細的討論C各方面的內容，要比第一章來得正式。不過兩者均強調怎麼撰寫完整的、有用的程式，而不是討論孤立的程式片段。第二章討論基本的資料型態、運算子、和運算式。第三章介紹控制流程：if-else、while、for 等。第四章包括函式和程式結構——外部變數、範圍規則等等。第五章討論指標和位址運算。第六章則詳細討論結構、組合的用法。

第七章描述標準的C輸入／輸出程式舘，以便使用者的程式可以做輸入、輸出及使用系統的其他功能，並可以由一個機器轉到另一機器而無須做任何改變。

第八章描述C程式和UNIX作業系統的界面（interface）；主要集中在輸入／輸出、檔案系統、和可携性的討論。誠然此章的某些內容僅對UNIX有效，不過不是使用UNIX的讀者仍可以由此章得到一些有用的知識，例如如何製作一個標準程式舘，如何撰寫可携程式碼等等。

附錄A是C的參考手册。對C的語法、語意有「正式的」描述，並補足了（除了個別編譯程式的差異之外）前面各章中描述不夠詳盡的地方。

由於在各種系統上，C語言都在不斷的成長、改革，本書的少數內容可能已和某些系統上發展出來的編譯程式不完全一致。作者已儘量避開這個問題，並對潛在可能的改變提出警告。不過在需要做取捨時，作者通常選擇PDP-11 UNIX上的做法，因爲這是大部分C程式設計師所處的環境。附錄A也描述各主要C系統的差異。