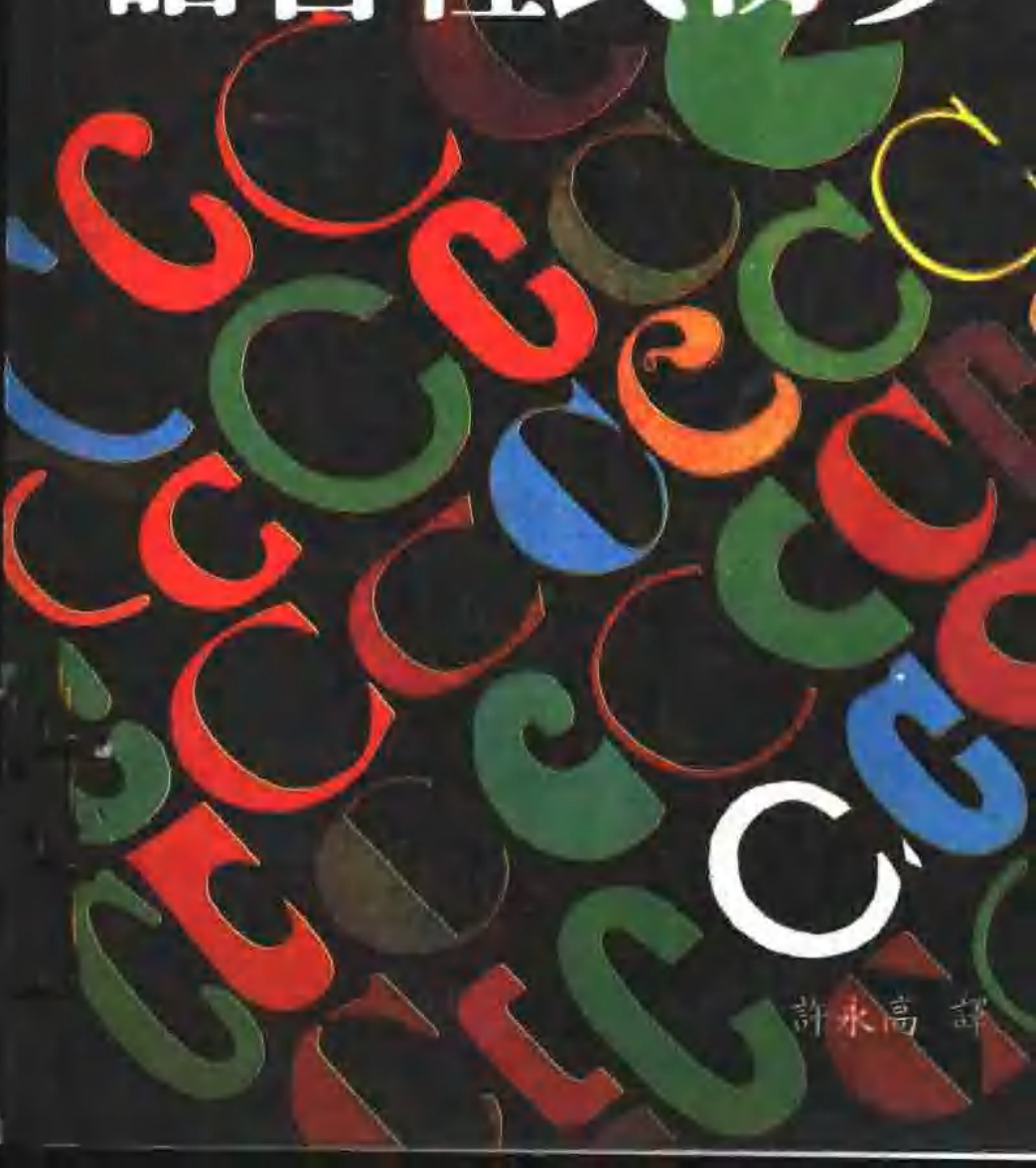
C
語言程式初步



許永高 譯

語言程式初步

許永高 譯

電腦科技社

概　論

(Introduction)

本書是屬於入門性質的書，是爲一些不瞭解C程式語言的程式設計者而撰寫的。讀者只需要具備基本的電腦程式設計觀念，就能看懂本書。我們假設讀者已經知道程式的編校（edit）、編譯（compile）及執行這些觀念。

我們承認，這本書並沒有將C語言作完整地描述；在附錄中，列舉本書對C語言的省略部份。本書亦沒有深入討論有關於系統軟體在C語言裏所扮演的角色；雖然此點恰好是C語言當初設計時最著重的部份。我們完全以讀者爲著眼點，而略去C語言裏比較艱深的部份。我們很驕傲的說，本書是用一種清楚、簡單而且又能把握重點的方式，將C語言介紹給初學的讀者。

本書所做的最重大省略，就是有關於檔案的處理（file handling）。大部份的人都知道，C語言大都使用在UNIX這個作業系統上。UNIX具有相當完備的檔案處理功能。由於本書的撰寫，是採用語言和作業系統獨立的方式，所以我們才將其省略。但無論如何，當你讀完本書後，你可以查尋C語言的使用手册（manual），進而容易的知道檔案處理的方法。

雖然C語言是一個清楚而且直接了當的電腦語言，但是初學者在使用時常會產生一些困擾。尤其，如果你已經學過了其他的電腦語言，有二項問題必定會令你覺得相當迷惑。第一項困擾就是，C語言對於資料型態的要求並不十分嚴格。在PASCAL裏，所有的變數都必須經過宣稱後，才能使用。但是C卻沒有這個要求，使用者可以使用不經宣告的變數。在這種情況下，C的編譯器（compiler）在處理時，就會依據程式的結構而自動假定（system default）變數的型態。當然，這個假定的結果並不一定會符合使用者的意願。所以，爲了避免混淆起見，初學者最好還是不要冒險，先宣稱變數後再去使用。

另一個容易使初學者困惑的地方就是，C語言裏有關於指標（pointer）的使用。由於指標的做法是間接性的，所以一般的初學者並不容易弄清楚。如果讀者對於組合語言（assembly language）很熟悉，這個困擾就不存在了。要解決指標所帶來的困擾，最好的方法就是不去用它。不使用指標的話，C語言就和其他結構化語言的使用沒有什麼太大不同。因此，我們希望讀者能儘量習慣於指標的使用，並發掘出它的優點。

在計劃撰寫本書之前，我們曾經考慮到要給讀者一些練習的作業（exercise）。現在於書中，並沒有提供練習的習題。因爲，我們發現已經有人專門爲C語言撰寫了一些非常好的習題，並出版成書了。這本書的名字叫做“The C Puzzle Book”，是由Alan R. Feuer所著。我們希望讀者在看完本書後，能儘量找到此書以做爲練習。當然二本書並沒有完全的配合，所以讀者在The C Puzzle Book中可能會發覺一些本書所未描述的問題。但是我們相信，讀者應該能很輕易的解決。

當讀者看完本書（和Feuer 的書）後，要如何增進自己對於C 語言的認識呢？我們的建議是請讀者去看：The C Programming Language 這本書。此書是由Kernighan 及Richie 兩人所著（後者爲C 語言的設計人）。本書的主要目的，其實就是將初學者提昇到能看懂The C Programming Language 這本書的水平。我們希望讀者能先經由本書較淺顯的說明後，再去看Richie 所著的書，如此學習起來才不會太過吃力。此外，我們也要警告讀者，本書在閱讀時最好能夠按照編排章節，依次閱讀。如果讀者對於系統程式（system programming）的觀念不很清楚，看完本書後，還是可能會覺得The C Programming Language 這本書並不容易懂。

如果讀者對於程式設計的“藝術”很有興趣的話，在此我們也推薦一本叫“Software Tool”的書，它是由Kernighan 和Plauge 所撰寫的。這本書所使用的電腦語言是Ratfor；Ratfor和C 語言相當的類似。

當然，讀者應該馬上練習撰寫C 語言的程式。這是眞正了解這個語言的唯一途徑。

目　錄

第 1 章

C是什麼
(What C Is)?

C是一個電腦程式語言（programming language）。它是在 1972年左右，由貝爾實驗室（Bell Laboratories）所發展出來的。整個語言的設計及撰寫都是由Dennis Ritchie 一個人所完成的；他當時和Ken Thompson兩人一起負責UNIX這個聞名的作業系統的發展。

UNIX這個作業系統是專門爲軟體工程師所設計的系統。C語言則是這個系統下，一個最基本而重要的工具。其實，所有UNIX所提供的軟體，甚至於作業系統本身，都是用C語言所設計的。

大約於70 年代中期，UNIX由貝爾實驗室裏被推廣了出來；不久它就被全世界所廣泛的接受。C語言也因此征服了愈來愈多的軟體設計從業人員。目前，C語言在貝爾實驗室內已經成爲一項“標準”語言，甚至成爲了一個“商標”。但是，值得強調的，就是C語言的受到歡迎，在事前並沒有經過任何有計劃的推銷工作，而完全是由於使用者的樂於接受的關係。到了1980 年代，在許多不是UNIX的作業系統上，也開始提供C語言。因此，C語言的使用已經是軟體發展的必然趨勢了。

C語言的規則相當簡單、清楚而易於使用，所以它是一種可靠性極高的語言。這一類的語言，通常被大家稱做“結構化語言”（structured language）。由於1970年代，結構化程式設計（structured programming）成爲時勢所趨，因此有許多屬於這一類的程式語言被人設計並推廣出來。其中，PASCAL 成爲了衆所注目的焦點。但是，由目前的情勢來看，已經有許多的PASCAL 的愛好者，在不知不覺中，開始以C語言做爲他們程式發展的工具。因此，C語言可以說是“後來居上”，成爲結構化語言的新寵兒了。

C語言的發展歷程如下所示：

ALGOL 60；於1960年提出

↓

CPL；於1963年提出

↓

BCPL；於1967年提出

↓

B；於1970年，由Ken Thompson 於貝爾實驗室所發展出

↓

C；於1972年，由Dennis Ritchie 所發展出

雖然ALGOL 僅比FORTRAN 的出現晚了幾年，但是由於它的結構非常嚴密，因此對於後來的程式語言均有非常重大的影響其設計者非常注重語法、模組結構和其他我們認爲是“現代”的特徵。但是很不幸的，ALGOL 本身卻未被人廣泛的

接受。CPL（Combined Programming Language）是仿照ALGOL 而設計出來的另一個語言，但受到本身過於龐大且複雜的影響，很難學習和製作。隨後的BCPL（Basic Combined Programming Language）和B 語言，則是截取前二者的一些精華所設計出來的特殊用途語言，並不能解決一般常見的許多問題。一直到Ritchie 把一些缺少的功能加入並做規劃整理後，C 語言才算是眞正的誕生了。

C 語言和LISP、PASCAL 及APL ，通常被冠以“單人語言”（one-man language ，即少數人所設計出來的語言）之名。這些語言的共同特點就是：簡單、清楚；至於相反的語言則有：PL/I、Ada 及ALGOL 68 等。

C 語言有一項極特殊的性質：它能由小而大，由簡單而趨複雜的堆積起來。因此，一個非常複雜的程式可以經由一些簡單的功能相互結合起來。舉例子來比較的話，就好像：將電子及中子組合成各種原子，再將各種原子任意組合成各種物質的步驟一樣。利用此種能力，程式的設計就顯得方便且有彈性了。此種能力，我們可以用一句英文來描述：a lot from a little。

通常，由一個人所設計的語言，對於處理他本人所涉及的問題會有比較優越的能力。Dennis Ritchie 本人比較著重於系統軟體（system software）的發展；包括了電腦語言、作業系統和文件資料的處理。所以，C 語言是設計這一方面軟體最具成效的一種。雖然，它具有處理一般問題的能力，但是我們特別強調一下：C 語言並不是適合所有應用問題的語言。當然，你可以用C 語言來撰寫有關會計作業的軟體，或是類似的

應用；但是，我們建議你最好不要這麼做。你如果採用其他語言來處理會計作業可能會比C語言來得便利一些。

因此，我們這麼說：C語言是一個著重於系統軟體的程式語言。爲什麼我們會這麼說呢？原因有二：首先，它具有低階語言（low-level language），如組合語言的性質。因此用它來設計的程式可具有相當高的效率。其次，它具有一些高階語言的性質，因此對於程式設計的效率亦可提高。讀者必定會奇怪，何以C語言能同時具有二種不同層次語言的特性呢？首先，我們必須先看下面這個層次圖表：

人工智慧"語言"

⋮

命令語言（作業系統所提供）

以問題爲出發點的程式語言

以機器爲出發點的程式語言

組合語言

⋮

硬體

根據這個層次表，我們可以發現：愈往上層，語言就和機器的距離愈遠。主要由於，較上層的語言是以解決問題爲著眼（problem-oriented），因此和直接與機器相關的組合語言，自然有極大的距離。舉例來說，如果把FORTRAN 中的一個敍述：a=b+c，換成以8080組合語言來設計的話就是：

```
LHLD .c
PUSH H
POP B
LHLD .b
DAD B
SHLD .a
```

對於設計程式的人來說，前者並不需要有機器硬體的知識就可以設計出來；但是，後者則正好相反。

但是，因爲FORTRAN、ALGOL 這一類以問題爲出發點的語言，由於層次過高，因此不適合設計和機器關係極大的系統軟體（system software）。因此，早期的系統軟體設計師還是必須以組合語言來設計程式。這種情形經過多年以後，系統工程師將高階語言的層次稍微的降低一點，而發展出以機器爲出發點（machine-oriented）的語言。其中，B 和 BCPL即是二個屬於這一層次的語言。這種語言，由於和機器的關係過於接近，所以除去使用於系統軟體的設計外，並沒有其他的用處。C 語言則是介於機器爲出發點和以問題爲出發點之間的語言，因此，它不僅具有低階語言的特性，也有高階語言的能力。

C 不是什麼？（What C Isn't?）

C 語言其實並不能算是一種“語言”；它是由一些符號的組合所構成的。由於一般人都稱之爲語言，因此，我們只得隨著大家這樣子稱呼它了。

C 語言並不是數學（mathematics） 的一支。但是，它看起來卻容易讓人發出這種感覺。由於C 語言和機器的關係還是相當的接近，因此它所提供的數學運算能力並不很強。它僅能使用如 a＝（b＋1）／c 這種簡單的運算功能。

C 語言並沒有嚴格的規定。C 語言摒棄了這種做法，因爲絕大多數使用C 來設計程式的人，都是屬於較專業的人員。他們通常不會把任何程式語言視爲一項萬能的工具。

C 並非是一個完美的語言。因爲C 語言爲了易於使用及執行速度的效率等因素，將一些其他程式語言所提供的特點予以犧牲。對於熟悉C 語言的人來說，這樣的設計可能更會適當些。由於讀者對於C 語言尚不熟悉，因此對於這個部份，我們將留待後面再說明。

C 語言程式的編譯（Compiling C Programs）

電腦無法直接執行由 C 語言所設計的程式。因此，爲了要執行一個C 語言的程式，首先必須透過另一個軟體程式將它轉換成屬於機器層次的語言。這個轉換的軟體程式，我們稱之編譯器（ compiler）。

以C 語言所撰寫的程式，被稱爲原始碼（ source code）。編譯器的主要工作即是將原始碼翻譯成機器能夠了解並執行的指令（ instruction ）。由於電腦之間的結構均有所不同，因此在不同的機器上，隨著機器語言的不同，就需要有不同的C 語言編譯器。原始碼經過翻譯後所產生的結果，我們稱爲執行碼

（executable code）。由上述，我們可以知道，相同的原始碼，隨著機器的不同，會產生不同的執行碼。

由原始碼一直到執行碼的產生，中間需要經過多次的處理步驟。首先，程式必須經由電腦系統所提供的編輯器（editor）予以鍵入。整個原始碼必須先存在一個檔案後，再由用戶鍵入要求翻譯的命令（在UNIX中，此命令是CC）去進行翻譯程序。下面就是整個過程的圖表說明：

程式的撰寫：　編輯程式

↓

C的原始碼

↓

程式的編譯：　C的前置處理器

↓

廣義的C原始碼

↓

C編譯器

↓

組合語言碼

↓

組合語言編譯器（assembler）

↓

程式目的碼（object code）

↓

連接器（linker）

↓

執行碼

↓

程式的執行： 載入器（loader）

首先，原始碼必須經過C的前置處理器（C preprocessor）將其中的縮寫型式敍述轉換成完整的C程式原始碼（expanded C source code）。此點，我們將於第八章中做完整的說明。其後，再經過C語言編譯器的處理，原始碼即被轉換成電腦的組合語言程式碼（assembly program）。這個組合語言碼會再經過組合語言編譯器（assembler）的處理而產生一個可做位置調整的目的碼（relocatable object code）。這個目的碼並不可直接執行，而必須再經由連接器（linker）的幫助，將和該程式相關的其他程式或是庫存程式（run time library）連接起來；後者，我們將於第十三章做說明。經過連接器處理過的結果即是執行碼。執行碼的執行必須再經過系統所提供的載入器（loader），才能放入記憶體內開始執行。

由上述的步驟來看，程式由原始碼開始，一直到能執行爲止，似乎是一條極漫長的路程。幸好，上面的絕大步驟都不需要用戶去操心，UNIX系統會自動處理這些煩雜的程序。通常，用戶只要鍵入CC這個命令，經過約數秒的時間後，整個翻譯的程序就都可以完成了。下面，我們將用一個例子來做說明。

首先，我們把本書所假設的電腦使用環境做一個說明。本書中所有的例子，均是以在PDP-11/45這個機器上，UNIX/6作業系統下做爲依據。如果，讀者所使用的不是UNIX系統的話，則請自行查閱使用者手册，以便了解其中的差異。