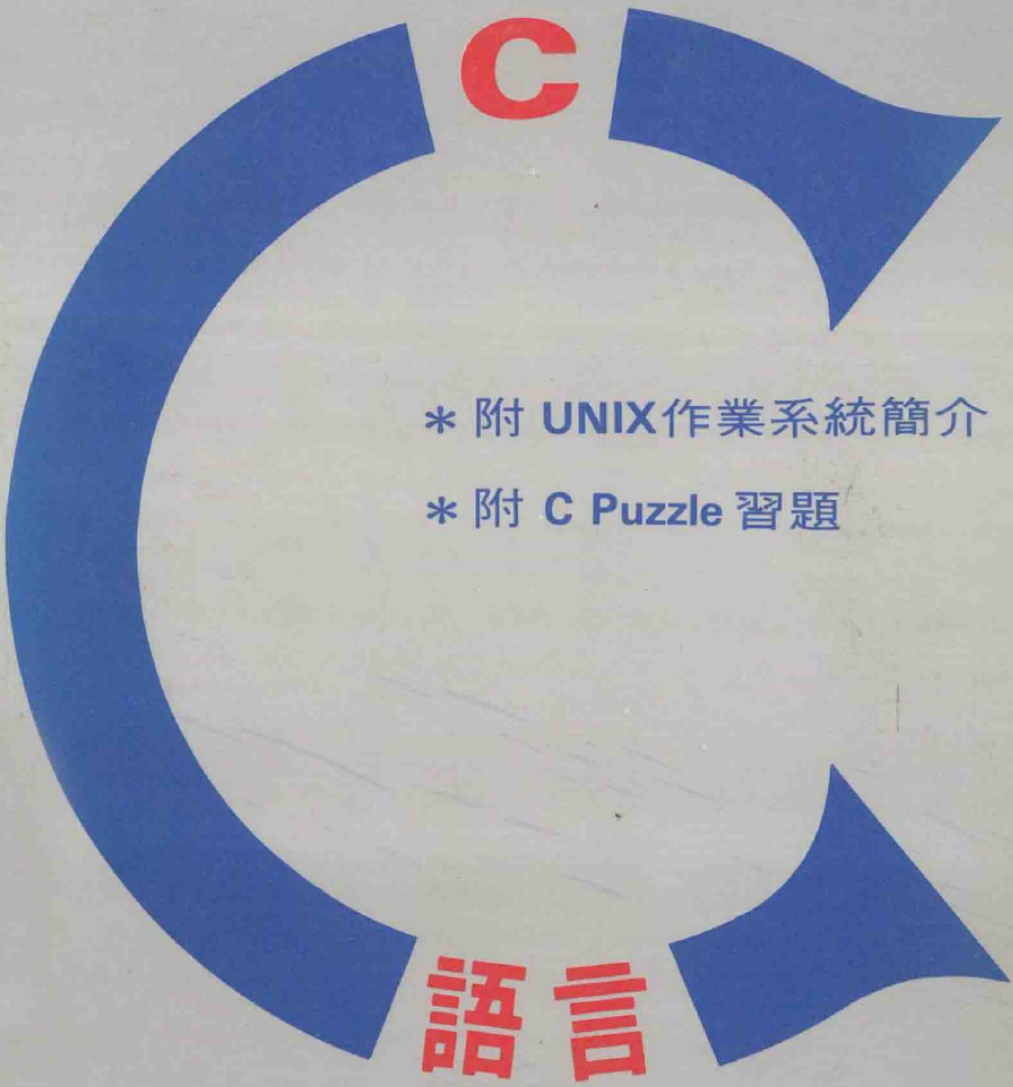
C 程式語言



＊附 UNIX作業系統簡介

＊附 C Puzzle 習題

涂永超 編著

C 程式語言

＊附 UNIX作業系統簡介

＊附 C Puzzle 習題

涂永超 編著

儒林圖書公司 印行

C程式語言

編譯者：涂　永　超
發行人：楊　鏡　秋
出版者：儒　林　圖　書　有　限　公　司
地　址：台北市重慶南路一段111號
電　話：3812302　3110883　3140111
郵政劃撥：0106792-1號
吉豐印刷廠有限公司承印
板橋市三民路二段正隆巷46弄7號
行政院新聞局局版台業字第1492號

1984年11月10日
定價新台幣200元正

序

C語言之所以迅速地被廣泛採用，主要是因爲它不但具備了高階語言方便、易學的特性，而且還具備了組合語言的優點－執行速度快且節省記憶體空間。此外，要精通UNIX作業系統，更必須由C語言入門。因爲，幾乎所有的UNIX系統軟體均是利用C語言所撰寫而成的。

本書共分成十章，第一章至第八章是介紹有關C語言本身的特性，讀者若要用C語言來撰寫程式，解決一般的問題，那麼讀完一至八章已經綽綽有餘了。可是若要進一步認識UNIX系統本身，則第九、第十兩章可說是入門之鑰，非讀不可。

本書的特色是在各章後面均附有精簡習題及解答。讀者務必費心精研這些習題，以便探得C語言的特性。此外，在附錄A亦附有UNIX系統的操作方法，讀者可依照附錄A中的例題，逐一打進電腦練習。

編者利用課餘之暇，參考有關著作，編著此書。雖然力求完美，唯時間倉促且編者才疏學淺，漏誤之處在所難免，尙祈讀者及諸專家不吝賜正。

涂永超

民國73年6月於

美國，維基尼亞州

目　　錄

第一章

C程式語言簡介

C程式語言是由貝爾實驗室（Bell Laboratories）在1972年所發展出來的。整個語言的設計及撰寫都是由Dennis Ritchie個人獨自完成。他當時正和Ken Thompson共同發展UNIX 作業系統（Operation System）。

1.1 C的演進

UNIX作業系統是專門爲軟體工程而設計的系統。C語言可說是該系統下最基本的工具，UNIX系統本身和其相關軟體均是以C語言撰寫而成。由於C語言可用來設計作業系統，所以又稱之爲系統程式語言。其實C語言並非僅適用於某特定作業系統或特定機器，它同樣適用於數值運算、文字處理和資料庫（data base）等方面的程式設計。

大約在1970年代，UNIX 系統由貝爾實驗推廣出來，當時普遍使用在美國各大學中，但是不久它就被全世界廣泛的接受了。目前，C語言在貝爾實驗室內已經成爲一項“標準”語言，甚至成了它的註册商標。但是值得強調的是C語言之所以受到熱烈的歡迎，在事前並沒有經過任何促銷廣告，而完全是由於使用者樂於接受的關係。到了1980年代，軟體設計公司相繼推出多

種不同的C編譯器（ compiler ）。因此，在許多非UNIX作業系統上，也開始提供C語言。如今，C語言的使用已經是軟體發展的必然趨勢了。

C語言是一種結構化程式語言（ structured language ）。由於結構化程式語言的使用規則相當簡單、清楚，所以類似這種語言，在1970年代相繼被設計且推廣開來。其中PASCAL語言可說是衆所注目的焦點。但是由近幾年的發展情勢來看，已經有甚多PASCAL的喜好者，逐漸以C語言來替代PASCAL作爲軟體發展的工具。因此，C語言可說是後來居上，成爲結構化程式語言的新寵兒了。

C語言的逐代演變如下：

ALGOL 60 （1960年問世）
↓
CPL （1963年問世）
↓
BCPL （1967年問世）
↓
B （1970年，Ken Thompson，貝爾實驗室）
↓
C （1972年，Dennis Ritchie，貝爾實驗室）

雖然ALGOL只比FORTRAN 晚出道幾年，但是由於其嚴密的結構，對於後來的結構化程式設計，影響非常重大。但是不幸地，ALGOL本身卻未被大家廣泛的接受。CPL（ Combined Programming Language ）是延襲ALGOL而設計出來的另一種語言，但是由於該語言本身過於龐大且複雜，對於程式使用者來說，很難學習和撰寫。隨後的BCPL（ Basic Combined Programming Language ）和B語言，則是擷取前二者的部份精華所設計

出來的特殊用途語言。然而，這二種語言仍然無法解決一般常見的許多問題。後來經由 Dennis Ritchie增添了一些特殊功能並經規劃整理後，**C**語言才算是眞正的問世了。

C語言的撰寫有一項特性：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。舉例來說，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。利用此種功能，程式的設計就顯得方便且有彈性多了。

Dennis Ritchie本人比較著重於系統軟體的發展；諸如電腦語言、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，但是得特別強調的是，**C**語言並不是適合所有應用問題的語言。當然，使用者可以用**C**語言撰寫有關會計作業的軟體或是類似這方面的應用。但是，若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。

C語言是一種著重於系統軟體的程式語言。其主要有二項優點。首先，它具有低階語言，如組合語言（ Assembly Language ）的性質。因此用它來設計的程式能夠節省記憶體空間及執行時間。其次，它具有一些高階語言的性質，使用者可以不須了解計算機硬體結構，即可著手撰寫程式，對程式的撰寫較具彈性且有效率。

舉例來說，若將 **FORTRAN** 中的一條敍述（ statement ）：

```
a = b + c
```

轉換成 8080 組合語言，表示如下：

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

對於程式使用者來說，前者並不需要具備計算機硬體的知識就可以設計出來，但是後者則恰好相反。

由於 **FORTRAN**、**ALGOL** 等高階語言不適合用來設計和計算機硬體息息相關的系統軟體。因此，早期的系統軟體設計師還是必須使用組合語言來撰寫程式。但是，若使用 C 語言來設計系統軟體程式，不僅具有低階語言的功效，同時亦有高階語言的同等彈性。

1.2 編譯C程式

計算機無法直接執行由 C 語言所撰寫的程式。因此，爲了執行 C 語言程式，首先必須藉著另一個軟體程式，將 C 程式轉換成計算機能夠認識的語言。這個軟體程式稱爲編譯器。

以 C 語言所撰寫的程式稱爲原始碼（ source code ）。編譯器的主要功能即是將原始碼翻譯成計算機能夠了解及執行的指令。由於各種計算機的結構均有所不同，因此在不同的機器上，隨著機器語言的不同，就需要不同的 C 語言編譯器。原始碼經過編譯後所產生的結果稱爲執行碼（ executable code ）。由上述可以知道，相同的原始碼，隨著機器的不同將會產生不同的執行碼。

由原始碼到執行碼的產生，中間需要經過多次的處理步驟。首先，使用者撰寫好的程式必須經由系統的文件編輯器（ text

editor）予以輸入。然後將該程式存檔起來，稱爲原始碼檔案（source code file）。再打入編譯的指令（**UNIX** 系統中，編譯指令是ＣＣ）來進行編譯該原始碼。

下面是整個步驟的流程說明：

撰寫程式：　文件編輯器（text editor）
↓
C原始碼（C source code）
↓
編譯程式：　C前置處理器（C preprocessor）
↓
擴充原始碼（expanded C source code）
↓
C編譯器（C compiler）
↓
組合語言碼（Assembly Language code）
↓
組合語言編譯器（Assembler）
↓
程式目的碼（object code）
↓
連接器　（Linker）
↓
執行碼　（executable code）
↓
執行程式：　載入器　（Loader）

C 前置處理器將原始碼中的縮寫型式敍述轉換成完整的**C**程式原始碼，再經過**C**編譯器的編譯後，原始碼即被轉換成計算機的組合語言程式碼，組合語言程式碼再經過組合語言編譯器的編

譯後就產生了目的碼，目的碼並不可以直接執行，必須再經過連接器的幫助，將和該程式相關的其他程式或是庫存函式（library function）連接起來。經過連接器處理過的即是執行碼。執行碼的執行必須再經過系統的載入器才能載入記憶體內執行。

由上述的步驟來看，程式由原始碼到能執行爲止，其過程似乎是極其漫長。幸好，上面的絕大步驟都不需要使用者操心，**UNIX** 系統會自動處理這些煩雜的程序。通常使用者只要打入 CC 指令，編譯的程序即可完成。下面舉例說明。

首先必須聲明的是，本書的例子均是以 PDP-11 計算機及 **UNIX** 作業系統作爲依據。若使用者採用的不是 **UNIX** 作業系統，請參考該系統的操作手册並比較其中的差異。

第一個簡單的程式是要印出下列文字：

```
hello, world
```

首先利用文件編輯器將下面的程式輸入：

```
/*                          C                          */
/*  This C program prints a message on the screen */
main (    )
 {
      printf ( "hello, world\n" ) ;
 }
```

假設該程式存檔在 hello.c 檔案中。接著要求系統予以編譯及執行。整個過程如下所示：

```
$ cc hello.c
$ a.out
hello, world
```

$ 是UNIX 系統所顯示的符號稱爲謝耳提示（ shell prompt）。使用者必須待終端機螢幕上出現此符號後，始能打入任何指令。因此，在 $ 符號之後打入 CC 編譯指令：

```
$ cc  hello.c
```

此處必須強調的是，原始碼的檔案名稱一定要以 .c 做爲結尾。例如 hello.c。

假設整個編譯過程進行得很順利，$ 符號會再度顯示。此時打入 a.out 指令要求執行該程式：

```
$ a.out
```

系統即會立刻執行該程式，並將結果顯示在終端機上：

```
hello, world
```

若使用者熟悉UNIX 系統（請參考附錄A），此時檢查自己的檔案目錄（ file directory ），將會發現其中多了一個a.out 檔案名稱。因爲C編譯器每次編譯後均會將編譯結果存檔到a.out 檔案名稱中。

上面的例子非常簡單。事實上一般C語言的程式要比該程式複雜得多，而且可能分別存檔在數個不同的檔案中。這些程式必須經過個別編譯，直到使用者將他們組合成一個完整而可執行的程式。例如若將三個函式存檔在三個檔案內，其檔案名稱分別是 file1.c 、file2.c 和file3.c ，使用指令：

```
$ cc file1.c  file2.c  file3.c
```

編譯這三個檔案，並把編譯後產生的目的碼存檔在 file 1.o 、file2.o 和file3.o 中，最後將這些目的碼載入 a.out的可執行檔案。

若在編譯過程發生錯誤，例如 file1.c 有錯誤，該檔案可重

新編譯，並將結果和先前沒有錯誤的目的檔案一起載入。使用下面的指令：

```
$ cc file1.c  file2.o  file3.o
```

cc 指令會根據檔案命名方式，以“ .c ”和“ .o ”來區分原始檔案和目的檔案。

利用上述的編譯程序就可以將一個複雜的程式分割成數個較小的程式，分別予以撰寫、編譯和測試，直到最後再組合起來。

現在僅對該程式作個簡單說明。前面二行：

```
/*                    C                    */
/* This C program prints a message on the screen */
```

即是所謂的註解（ comment），註解可以任意自由使用，使得程式令人容易了解。任何空格（ blank ）和跳行字元（ newline ）可以出現的地方，都可以加入註解。註解部份必須置於/*和*/二組註解符號之間。

不管程式的大小，C程式至少可包含一個或數個函式（ function ），這些函式都具有特定的功能。C語言中的函式和**FORTRAN** 中的副常式（ subroutine ）或**PLI** 、**PASCAL** 中的程序（ procedure ）類似。在上例中main 即是一個函式。一般而言，函式名稱可以任意命名。然而，main 是其中比較特殊的一個函式，程式被執行時將由 main 函式的開端執行起。換句話說，每一個C程式一定要有一個名稱爲main的函式。通常main將呼叫（ call）其他的函式以執行其工作。函式呼叫時可以呼叫在同一程式中的其他函式，也可以呼叫系統中的庫存函式。

函式間資料轉換的方法之一是利用引數（ argument ）方式來傳遞引數串列（ argument list ）。引數必須放在函式名稱後