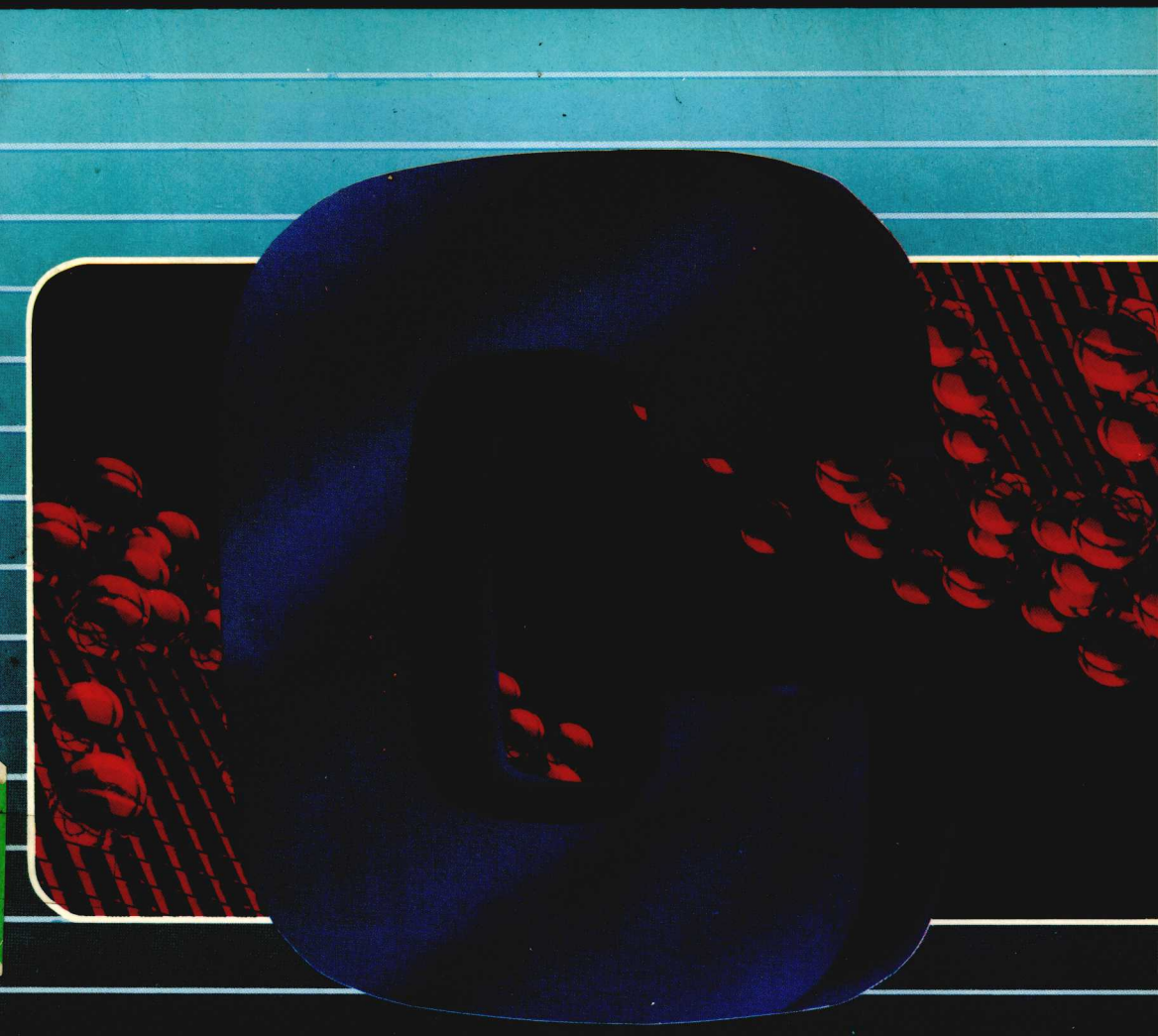
高等C語言程式設計



黃俊堯　盧振強　譯

高等C語言程式設計

黃俊堯　盧振強　譯

儒林圖書公司　印行

版權所有
翻印必究

高等C語言程式設計

譯　者：盧振強・黃俊堯
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：3118971-3　3144000
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
中華民國七十六年八月初版
定價新台幣　160　元正

前　言

以下是本書的特色：

——它是一本進階級的書；是站在已熟悉C程式語言的人的立場來解題的。

——它在課文的字裏行間討論C語言，而不只是介紹一些命令和資料結構的形式。我們視它爲動態的解題工具，可用以探討一個問題集，並將問題結構化，進而找出問題的解決之道。

——它將C語言的本質表達出來。本書把C程式設計上的樣式和它的優點擷取給讀者。在此，結構化程式設計的觀念依然適用，我們會在C語言的上下文中展現出來，不會爲強行附和理論上的概念，而扭曲了語法。

——最後，我們介紹一些新的結構化概念。這種構造很容易用C語言作成，但對其他語言而言，則屬不易。最重要的是，我們使用了範域和儲存等級的法則，以及獨立編譯的特性，而建立了低關連性的程式模組——它是結構化程式設計上的重點。

大多數的C程式設計書籍，都提供了語法的說明和一些簡單到中級的範例。這是爲了能使讀者熟悉C語言的程式設計方法，並提供對C語言的概觀。本書的目標，則與它們不同：

——以眞正程式設計上面臨的問題，來介紹先進的技巧。

——介紹C程式設計的環境，包括可用的工具，以及最適合以C語言解決的問題。

本書將幫助讀者，使C程式語言，成爲您內在的解題工具。

此为试读,需要完整PDF请访问: www.ertongbook.com

目　錄

1

導　論

本章宗旨

* 瞭解一些我們須先具有的知識，以幫助往後的閱讀。
* 瞭解本書的格式（format）。
* 瞭解編譯（compilation）時的前置處理階段（preprocessor phase）。

本章是一種概略性的介紹；包含了兩個部分。第一部分，乃本書其他章節之導言，它詳述了本書的宗旨：

——你所要有的知識。
——如何完成這些目標。

此外，還包含了一些本書所要討論的主題。

第二部分介紹C語言的一種特色——巨集前置處理程式（macro-preprocessor），它對一些組譯程式（assembler）而言，是很常見的。因爲前置處理程式，對於本書中所述之C程式的結構化很有幫助，所以我們先在此處介紹它的基本概念。前置處理程式含有一些敍述，這些敍述對編譯程式（compiler）而言，有如後設命令（meta command）。這些敍述，對C程式設計師而言，十分的重要，因爲它有助

於設計出可攜的（portable）和易於維護的程式。本章將在課文中，清楚地介紹這些命令，並使讀者瞭解，它們如何成爲結構化的軟體設計工具。

1.1 本書討論的主題

C程式語言已成爲最重要的軟體發展工具之一。它的規模、深受學術機構的重視與採用，以及它的中階語言特性——與高階語言一樣有結構化的程式形式，同時又與低階語言一樣能直接發展出硬體介面（interface）的演算法（algorithm）——使它成爲每個程式設計師心愛的寵物。C語言擴展了計算的範圍，從大型電腦，到最小的個人電腦，都可以發現它的踪跡。目前在小型到中型的商業電腦上，最受歡迎的應用（application）軟體，大多是由C語言所寫成的，它已逐漸地廣受每個人的歡迎。

市面上，有許多對C程式語言，作概要性介紹的書籍。這些書大都討論C語言的語法（syntax），其中包含C語言的一些艱難而特殊的結構。然而，一本書，如果想涵蓋一個像程式語言一樣複雜的主題，就必須避免冗長的討論。冗長的討論，實際上並非這些書籍的缺陷，而是想作全盤介紹的必然下場。本書主要在於簡明地討論一些坊間書籍所忽略的先進主題。

對程式設計而言，C語言不是一種很笨的工具，也不是一種難以學習的語言；但它也不像BASIC 語言那樣的簡單。在增加程式語言的效力，以及與使用者的親善（friendliness）上，我們較傾向於前者的協調。C語言是一種工作性的語言，不是爲教育性的目的而設計的，它是用以解決日常生活的問題；它具有自己的特質。通常在機器上的運作是不易掌握的，而C語言允許我們以簡單的程式來處理這個棘手的問題。總而言之，C語言是一種複雜的——而非簡單的——輔助工具。

我們不在負面上，作過多的討論。C語言實際上要比大多數的程式語言都好。它具有以下特性：

- 更具一致性。

此为试读，需要完整PDF请访问：www.ertongbook.com

- 在邏輯上更具結構性。
- 比多數的現代程式語言更具可攜性。

和任何複雜的工具一樣，它需要花上一點時間來學習。

本書的目的在引導已瞭解C語言語法並且寫了一些中小級程式的人，進入另外一個新的領域。我們將針對C語言的解題技巧討論，而不再贅述C語言中的基本元素。

我們的終極目的，在使C語言成爲一種內在的解題工具，讓你能透過C語言找出解決之道，而不是在問題的答案上，把C語言給應用上去。我們的目標是頗具野心的，但只要指出一個方向給你，相信這種目標，一定可以達成。

1.2 你必須具備的知識

本書並非C語言的介紹性課本。我們希望你都已瞭解C語言的語法。本書先假設你已熟悉了它的語法。在每個例子中，我們將任意使用C的敍述（statement）或結構，即使是在詳論它的用法之前。一些基本的核心敍述，只有在它們涉及一些有趣或不尋常的構造時，才會在課文中予以討論。

此外，我們還假設你已熟悉了編修程式（editor）的用法，編譯程式、電腦作業系統（operating system）與其公用程式（utility）的用法，以及電腦科學上的基本概念，如堆疊（stack）、連結串列（linked list）等等。然而，我們並不要求你是專家。你必須能把電腦當作一種工具來使用，而不是把它視爲頭痛的問題。你也不須具備電腦科學上的專業知識；只須有程式設計的基本概念就可以了。

此書針對已學會C語言的基本教育性資源，而希望對此程式語言有更深一層瞭解的人而作的。我們是指所有希望把C語言作爲內在的解題工具者而言。

另一個重要的目的是，本書希望使你能面對專業程式師所遇到的問題，也就是以前教學性的知識所無法立即洞穿的問題。我們希望能讓你了解，在結構、優美和解決問題這三方面，必須作個適度的妥協，並教導你如何作一個彼此傷害度最小的妥協。此外，本書將把一些技巧和速解法傳授給你，而我們敢大言不慚地說，在其他較理論性的教本上，這些技巧和方法可能只因非結構化或程式設計的樣式（style）不佳而遺漏。我們這種作法，並非向這些原則進行攻擊，而是希望呈現程式的眞實面貌給讀者。也許對讀者而言，最恐怖的是第一次在課文裏面對一個“眞實”世界中的問題。但如果，我們能消除讀者的恐懼，本書的目的就達到了。

1.3 本書討論的次序

我們計劃在每一章中，針對一個先進的主題進行討論：

- 第二章將探討C語言中所具有的模組式工具：如範域（scope）、分離式編譯和函數定義（function definition）。
- 第三章將探討標準程式庫中的輸入及輸出函數。
- 第四章將討論結構化變數（structured variable）：陣列（array）和結構（structure）。
- 第五章探討無所不在的指標（pointer），它是C語言最有效力的構造之一。
- 第六章探討作業系統與程式之間的直接界面。
- 第七章涵蓋了位元（bit）的處理。
- 第八章爲一個小型資料庫管理程式（data base manager）的討論、設計與實行方法。

在每一章中，我們並非將這些項目視爲獨立的數學實體，而是視爲可與其他元素結合成一個程式的個別元素。我們的興趣在於使這些

敍述、運算子（operator）、函數和資料結構，都能一起運作，互相呼應，也許有時候是以一種不整齊劃一的方式進行。在每一章中，都有具體的例子來作討論。

在第八章中，我們設計並製作了一個小型的資料庫管理系統；這一章主要是希望以一個大型的計劃，把本書中所討論過的概念完全結合起來。這些例子在探討一些C語言的元素因著彼此間的交互作用而產生的複雜度，這也正是我們在該處可以順便一提的觀念。

我們希望在此書中，不僅能使你瞭解所有C程式設計的概念，同時也能作爲C語言的一種環境。藉著瞭解C語言的成員彼此交互運作的關係，以及它們與問題元素之間的關係，我們就可以建立這種環境；也就是用C語言來解決問題的一種感覺：如何用C語言的元素，C語言的資料結構，甚至C語言的觀念，來設計一個演算法。這就是初學者所欠缺，而專業者所擁有的洞察力。

1.4 C語言的前置處理程式

在C編譯程式掃描原始檔（source file）之前，先執行一次巨集處理程式（macro processor）。這個常常被忽略而未受重視的前置處理程式階段，實際上十分有助於結構化的程式設計，和上至下（top-down）的設計。組合語言的程式設計師（assembly language programmer），要比高階語言程式設計師更爲熟悉巨集，和巨集前置處理程式等公用程式。這也可能是C編譯程式中，這個階段之所以被忽略的原因。

前置處理程式能爲我們作的第一個工作，就是允許一個外在的文字檔（text file）在編譯之前，包含到我們的原始檔中。這個前置處理命令，是每個曾寫過C語言程式的人，都使用過的：

```
#include <stdio.h>
```

這個命令，必須在程式的原始檔中，通常出現於程式的開端。它引導前置處理程式把 stdio.h 的內容，插入於原始檔的該處位置。角括號（＜，＞）表示，該檔案在標準目錄（directory）中可以找到。

```
#include "stdio.h"
```

上述命令能完成相同的工作，但它所找的目錄，是程式檔所在的目錄。

“ # ”表示前置處理程式的一個命令列（command line），它必須放在該列的第一行。請注意，該列結尾並無分號。和C程式不同的是，前置處理程式是以一整列爲處理單元，所以不需要列末尾的標示。前置處理命令可以繼續到下一列，只要在該列末尾使用一個“＼”字元作爲結束。

include 命令能使我們的程式，更趨於模組化。一些文句列，比方說，許多檔案中常見的定義（definition）和變數宣告（variable declaration）等等，最好能放在一個由它們所組成的特定檔案之中，這樣凡是需要用到它們的檔案，只要在程式開頭，以一個命令，把它們包含進來即可；因此，大大地減少了抄寫錯誤的可能。# include的使用，使得我們的原始檔更易於瞭解。它允許我們賦予名稱給

```
struct name  {
  char lname[81],
       fname[81],
       mname[81];
  };
struct address  {
  char street[30],
       city[30],
       state[2];
 };
struct  {
  struct name pname;
  struct address paddress;
 }entry;
```

(a) adr_decl 檔案的內容

```
#include "adr_decl"

main()
{
  :
  :
```

程式列表 1-1　# include 命令的使用說明

一個龐大的集合，並將它們的細節隱藏起來。程式列表 1-1 說明了這種方法。adr_decl這個檔案，避免主程式中會出現一些細瑣的宣告細節，因此能產生較清楚、易懂的程式列表。請注意，# include 只對文字檔有效。將編譯過的檔案連結起來，則是另一種處理。

前置處理程式還提供了一些工具，讓我們直接作文字的轉換。我們可先賦予某一數值一個特定的名稱（name），然後在檔案中，只要使用該名稱，就能代表該數值。前置處理程式，能很正確地把名稱代換成數值。下述之命令：

```
#define RATE 1.5
```

將使得RATE字串，代換成 1.5。習慣上，#define的名稱欄通常是大寫的，以便和變數有所區分。

在程式執行時，數值會保持一定，但 #define 命令最好擺在檔案的開端，這樣才能正確無誤地把該檔案的數值改變。（參見程式列表 1-2 ）。

```
#define RATE .065

main()
{
 double cost,price;

 printf("enter item cost:");
 scanf("%lf",&cost);

 printf("total cost is %lf\n",cost+cost*RATE);

}
```

程式列表 1-2　#define 命令的簡單用法

#define ，也能作有引數（argument）的代換。下述之命令：

```
#define MAX(a,b)   ((a)>(b) ? (b) : (a))
```

能建立一個 MAX 敍述，它需要兩個引數，而且會傳回較大的一個。在叫用（invocation ）此巨集時，a 、b這兩個虛擬（dummy）引數

，將被實際的數值所取代。因此：

```
t=MAX(x,y)
```

將以 x 代換 a ，並以 y 代換 b ，巨集定義與一個眞實函數的不同在於，當巨集被叫用時，巨集敍述將實際地放入程式之中。而函數碼，則只出現一次。如果我們呼叫MAX( )10次，則程式中將有MAX( )的10份拷貝（ copy ）。通常我們根據程式碼的多寡來決定巨集和函數的取捨。在一定的列數之下，簡單的巨集定義，要比函數呼叫的花費來得少些。請注意，在虛擬變數的兩旁，須加上括弧。

和 #define 相反的命令是 #undef ，例如：

```
#undef MAX
```

將破壞此字串原有的定義。程式列表 1-3 是這兩個命令的相對用法。和 #include 命令一樣，這些命令，可在檔案中的任何地方使用；但是和 #include 不同的是，#undef 和 #define 通常不用於程式的開端。

```
#define ORD(a,b)  ((a) > (b) ? (a) : (b) )

largest(x)
int x[];
{
 int i=0,max;
 max=x[0];
 for(i=1;i<10;i++)
   max=ORD(x[i],max);
 return(max);
}
#undef ORD
#define ORD(a,b)  ((a) < (b) ? (a) : (b) )

smallest(x)
int x[];
{
 int i=0,min;
 min=x[0];
 for(i=1;i<10;i++)
   min=ORD(x[i],min);
 return(min);
}
```

程式列表 1-3　#define 與 #undef 的相對用法

C 前置處理程式，提供了 #if 命令家族，而具有條件式（ condi-

此为试读，需要完整PDF请访问：www.ertongbook.com