常用C函數集

管中徽　譯

常用C函數集

管中徽　譯

儒林圖書公司　印行

常用C函數集

譯　者：管　中　徽
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
中華民國七十五年十一月初版
定價新台幣　180　元正

序　言

一本書可以許多不同的方式開始，不過大多數的書，都是以一篇序言作爲開場白，概略地說明編寫的目的；讀者需具有那些基礎方能獲益最多；並對完成此書貢獻良多之士致上深忱的謝意，本書自不例外。

本書爲儒林圖書公司C語言系列的一員，讀者如自己擁有一部電腦，一個C編譯程式，並對程式設計有些許經驗的話，必可自本書獲得最大的收獲，作者於撰寫之初，雖未假設讀者對C全無經驗，但也並未針對某一層次的讀者而寫，任何對C好奇的人，都有資格成爲本書的讀者。

C語言自發展以來即受專家們喜愛，它的普及可由各種機器上C編譯程式的建立；各種使用者團體的成立；衆多有關C的書籍、討論；以及無數以C編寫的軟體間也得到證明。C語言本身可能並非十全十美，而且將來也可能會有功能更強，使用更方便，學習更簡單的語言出現，不過就目前而言，C不僅在衆多語言之間保持領先的地位，並正逐漸拉開其間的距離。

當C語言出現，也正是作者全心投入Z80和8088組合語言的同時，作者正在設計CP/M-80、CP/M-86，和MS-DOS的系統程式，這種工作以組合語言來作是最恰當不過，而且作者也想不出有那種語言更適合這種著眼於效益的應用。不過，有時作者也需設計檔案I/O，使用者I/O，和非整數運算的程式，此時，組合語言即顯得極爲笨拙。

然後，Leor Zolman的BDS C出現了，此個人電腦上C編譯程式的先

驅，立刻吸引了組合語言的使用者（如作者本人），以及其他正想嘗試——結構化語言的BASIC程式設計者的注意力，而且，與其他編譯程式語言比較起來，BDS C的$150價格也多爲一般人所能負担，所以幾乎是一夜之間，許多使用者團體所出版的C原始程式如雨後春筍般地出現。

此後不久，有關C的著述文章開始出現在非專業性雜誌等刊物上。這些文章一致推崇C爲一多用途的語言，即使是初學者也應該熟悉，可是爲時不久，市面上介紹C的書籍（主要爲Kernighan和Ritchie的The C Programming Language）逐漸無法提供初學者與程式設計者與日俱增的需求。因此，敏感的出版商開始反映此一需求，而本書就在這種環境下誕生了。

作者在學習組合語言的時候，曾借助於許多熱心作者所提供的原始程式，在研讀這些原始程式的過程中，作者同時學到了一些正確與不正確的習慣，以及一些有趣但卻容易遺忘的技巧，不過重要的是，作者了解其正確與否。作者以爲，這種一方面研讀一些正確的原始程式，另一方面對照一本參考書籍，以了解程式作者係以何種方式與電腦溝通，來達成其目的的作法，實爲學習一種語言最有效的方法，因此，作者也如法炮製來學習C。

C之所以有別於PASCAL而較近似組合語言的一項特性，即是具有可個別加以編譯的程式庫（library）。所謂程式庫是您存放已經測試無誤的函數的地方。在Chris DeVoney鼓勵作者撰寫本書後，作者注意到市售的電腦書籍裏，充斥著許多有關BASIC副程式的書籍，作者不禁自問“爲何C却渺無踪影呢？”，其實C更適用於設計多用途的函數，因爲這些函數可合併於程式庫內，而在需要的時候，像是“工具”（tools）一樣重新叫用，而不需重新鍵入、思考或編譯。這些程式庫通常被稱爲軟體工具箱（software tool chests）。

因此，這本綜論C語言書籍的編寫，自始至終均秉持此一概念：藉著研讀原始程式以及一些常用的函數以協助讀者學習。作者希望這本書能帶領您建立自己的C工具箱。

任何程式設計者都能從其周遭一些深具耐心且程度頗佳的朋友處獲益良多，多年來作者本人即受益於許多朋友無倦地協助及諒解。Escort C編譯程

式的主要作者 Tim Leslie 在其專業的領域內有極卓越的成就，作者必需感謝他對本書提供了許多寶貴的構思。Dr. Jack Purdum、Allen Stegemoller 和 Bill Burton 給作者的許多建議及批評，亦使本書增色不少。

至於促成本書付梓致使作者得享著述的樂趣，還需感謝 Chris DeVoney 和 Ginny Noble 兩位所提供的機會。

此外，Steve Browning 及 Dr. Mitch Bodanowicz 不吝地刪改本書的手稿，以及 Ottawa Carleton 大學的 Dave Thomas 所提供有關結構型態的觀念，Ottawa 的 DY-4 系統提供了作者一個學習 C 語言的機會，在此均一併致謝。

爲使本書的函數、程式、範例、關鍵字及檔案名稱更爲清晰，本書採用如下之 Digital 字型：

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789
!@#$%^&*()_+~|
-=\`[]:;'<>",./?{}
```

本書以下列刻度尺（ ruler line ）協助讀者計算一程式列內的空白：

```
          0    1    1    2    2    3    3    4    4    5    5    6
1234567890123456789012345678901234567890123456789012345678901234567890
```

在程式列裏，方括號（ [ ] ）所包含的項目是可有可無的（選擇項），而垂直的短直綫（ | ）是用來分隔這些可選擇使用的選擇項。

目　錄

序　言 …… I

第一部份　這是一種學習語言的好方法嗎？…… 1

第 0 章　C語言的大概 …… 3

檔案與函數…… 3
變數和常數…… 8
宣　告…… 8
辨識名稱…… 9
範　圍…… 10
記憶體類型…… 12
資料型態…… 14
函　數…… 17
指　標…… 18
集合型態…… 22
尺　度…… 39
初設值…… 41

第 1 章　C的其他特性 …… 45

屬性串列…… 45
轉型運算…… 50
Typedef…… 50

常數和結果資料型態 …… 52
型態轉換 …… 54
型態的提昇 …… 55
型態的平衡 …… 55
設定運算的轉換 …… 55

第二部份　C程式看起來是什麼樣子 …… 59

第 2 章　C程式內部大概的構造 …… 61

前端處理程式指引命令 …… 61
#include …… 62
#define , #undef …… 63
#if , #ifdef , and #ifndef …… 65
#line …… 67
宣告看起來是什麼樣子 …… 68
函數看起來是什麼樣子 …… 69

第 3 章　運算式 …… 75

敍　述 …… 77
運算的優先順序 …… 78
邏輯運算符號 …… 82
Boolean 運算符號 …… 84
數值運算符號 …… 86
設定運算符號 …… 90

第 4 章　流程控制 …… 93

條件執行敍述 …… 94
重複執行敍述 …… 97

中斷執行敍述……101
結　論……105
第三部份　C通用函數之介紹……107
第一組函數：基本的函數……108
saygetmv.c……109
atoi.c……112
calc.c……117
keepme.c……120
listme.c……123
macros.c……126
macros.h……127
第二組函數：常用的函數……129
jul.c……129
revjul.c……133
twodates.c……136
zeller.c……139
numbers.c……142
第三組函數：使用者輸入／輸出函數……147
window.c……148
menu.c……154
report.c……162
scan.c……172
scan.h……175
main.c……177
庫存函數……194
adv.c……196
atsay.c……197

blank.c …… 199
bump.c …… 201
charstr.c …… 203
getnlin.c …… 205
gettime.c …… 206
inchar.c …… 208
incharux.c …… 210
place.c …… 212
setcurs.c …… 214
setmem.c …… 215
shift.c …… 216
sindex.c …… 218
第四組函數：可執行的程式 …… 219
kidedit.c …… 219
amort.c …… 226
pipe.c …… 233
list.c …… 235
附錄 A　名詞解釋 …… 241
附錄 B　C的關鍵字 …… 245
附錄 C　函數概要 …… 247
中英名詞對照 …… 267

第一部份

這是一種學習語言的好方法嗎？

身爲「知難行易」的信奉者，我深信這本書裏所用的「由操作中學習」（learning by doing）方式是學習一種程式語言的最佳方法。這並非老王賣瓜式的自吹自擂，而是可從作者以及許多自學成功朋友的經驗裏得到印證的。我們當初學習的方式是，先儘量的搜集並研究所有用C寫成的程式（很幸運地，大多數的程式都是公開的），然後再藉著不斷地練習，才能掌握C語言裏的訣竅，以及進一步的瞭解如何去結合資料與演算法來達成程式設計者的目的。

因此，如果您是C（或其他語言）的初學者，最好是有人交給您許多用該語言寫好的程式並要求您加以修改、修正或是說明這些程式的用途及用法。當然，這是一個痛苦的開始，不過您將來的收穫絕非一般傳統的學習方法所能比擬的。在本書第三部份所提供的函數裏，您將有機會接觸到許多C語言的程式設計技巧。

本書的第一和第二部份則是在幫助您了解，第三部份裏的函數到底是在作些什麼，在讀完第二部份後，您將會了解C語言到底能作些什麼、怎樣作才是正確的，以及如何獨立地去設計一個程式。當您接著研讀第三部份時，您會發現所獲得的報酬是驚人的。當然，除了本書所提供的函數外，您也可以自行搜集其他複雜的如C編譯器或簡單的如#字遊戲等程式來豐富您的經驗。不過並不是所有的C程式都是值得研讀的範例，確實某些程式只適合那

些對流程控制、函數、變數、編譯、連結以及 I/O 有基礎認識的程式設計者（一些您必需熟悉的字彙均包含在附錄A裏）。

C語言的基本觀念是很簡單的，它們可以很輕易地加以說明或了解，不過只有藉著研讀使用這些觀念的範例，才能眞正掌握到這些觀念在應用上的奧妙。如果您能好好學習各個章節裏的內容，並將所學應用到第三部份的函數上，C語言的美就會自然地呈現出來，而您，各位讀者，也將無可抗拒地受到感動。

所以，就讓我們開始吧……。

第0章
C語言的大概

檔案與函數

C語言有一個常讓初學者不習慣而實際上是一大進步的特點，那就是C能將一個程式分割爲好幾個部份，這些部份我們稱之爲檔案（ file ）。一個程式所包含的某些檔案常常不是程式設計者所能接觸到的，這樣說可能太籠統了一些，您不妨這樣想—如果您學過 FORTRAN語言的話，您一定曉得除了基本的 FORTRAN指令外，FORTRAN 還提供了許多庫存函數。比如說計算平方根的SQRT或絕對值的ABS ，這些函數都不是您寫的，可是如果您在程式裏用到這些函數的話，它們自然成爲您程式的一部份，而您在設計程式時卻無法感覺到它們的存在。實際上任何程式都具有這種特性，比如說每個程式都需要透過終端機、磁帶機或磁碟機與外界進行資料的交換。這些資料的交換在程式裏可能只不過是一條 INPUT,READ,WRITE 或 PRINT 指令，而實際上這些資料怎麼從輸入設備上讀入，而又怎麼存放到週邊設備上，全交給了作業系統如CP/M、MS-DOS 或UNIX 來處理。這些動作其實都是您程式的一部份，只是您渾然不覺而已。

高階語言功能強大的原因就在這裏，由於許多工作上的細節都交給了該語言的系統來處理，您和您所設計的程式可以全神貫注在手邊的工作上。而

C語言更將這種觀念充份地加以延伸，一個複雜的程式可以不必辛苦地從頭設計，而可以先分割成數個較小、較簡單的部份來分別處理，最後再將這些部份合併起來組成一個功能完整的程式，所得的結果雖然相同，但時間與腦力的節省卻是可觀的。

C程式最大也最明顯的部份就是檔案與函數（C語言的基礎），因此分析一個程式的檔案和函數，可以幫助您了解不同部份之間的關係，以及當初程式設計者爲何及如何將程式作如此的分割。檔案與程式之間的關係是非常有組織而且劃分得很清楚的，一個檔案或函數所執行的工作不僅在設計之初，而且在與其他檔案或函數組合起來後，是與程式的其他部份獨立而不相干的。一個較大的C程式可以看成是由數個模組（module）所組成，而模組本身也是由個別的函數所構成的。每個函數則被設計去執行一項小而且重複的工作，而許多函數合併成一個程式後，就可以完成一項大而且複雜的工作。因此一個大程式的設計變成了一種組合小程式的動作。圖 0.1 是一個中等大小的C程式可能的結構：

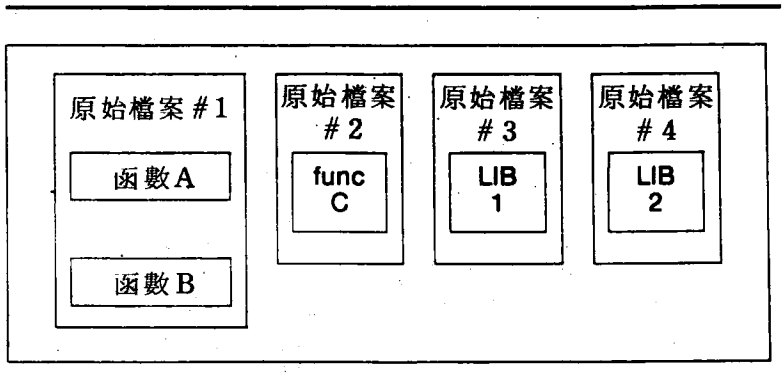

圖 0.1　一個C程式

#1 原始檔案裏的A函數和B函數構成了程式主體的模組，C函數單獨在#2 原始檔案裏，它可能是執行許多程式都常需要的工作。這三個函數都必需由程式設計者來撰寫，而它們也代表了整個程式設計者所能接觸到的部

份。在測試及偵錯的過程裏，需要加以修改及重新編譯的對象都不出這兩個原始檔案。

所有的C程式都是從一個名爲main的函數開始執行，然後在這個函數裏，程式的執行可能會跳到其他的函數去（C的函數也具有遞歸的能力），不過通常在程式結束前，程式的執行都會回到main函數。

＃3和＃4檔案裏的程式庫分別代表使用者程式庫（user library）和標準程式庫（standard library）[1]。程式庫裏包含的都是一些經常使用到的函數，這些函數都已先經C編譯程式編譯成可重定位目的碼（relocatable object code）。它們的存在對C而言是非常重要的，這一部份是因爲C不像其他語言有PRINT、WRITE或READ等I/O的指令，而這種輸出、入的工作C都是交由程式庫裏的函數來執行的。程式庫裏的函數都已經經過測試是可以正確動作的，所以您可以想像的到，當在檢查某個程式的錯誤時，由於您不需再修改及重新編譯所用到的程式庫，許多寶貴的時間可以節省下來。

程式的模組之間是以變數作爲資料交換的媒介，爲了促進程式的模組化，我們可以將不同的變數設定爲不同的使用範圍。所謂變數，對程式設計者而言是一個名稱，對C編譯程式而言則是一個記憶空間。因此在設定使用範圍後，某個記憶空間可以只能爲A函數所使用，縱然程式設計者在B函數內也用了有相同名稱的變數，其實兩者所代表的記憶空間是不同的，某個記憶空間也可以爲A、B函數所共享卻不能被C函數所使用。此外，和大多數的語言一樣，C程式裏的某些變數（記憶空間）可以爲程式裏的所有函數所使用。

歸納起來，所有相關函數所組成的原始檔案都稱之爲模組。而從程式設

---

1. 一般程式所需要的I／O常式（routine），在大多數C編譯程式裏都被包含在標準程式庫裏，不過各位讀者不要被「標準」二字所迷惑，標準程式庫所包含的I/O和其他公用函數（utility function）通常只適用於某種作業系統及硬體環境。因此不同廠家所提供的或用於不同機器的C編譯程式，其中的程式庫通常是不能互通的。

計者組織模組的方式可以看出，同一模組內的函數如何相輔相成，以完成整個程式的一部份動作。另外我們從模組也可以看出，那些變數的用途是暫時用來存放一些資料，那些變數的使用範圍只是局部性的，還是整個程式的所有函數都可以使用。

前面已經提過，函數是所有C程式的基礎，這是因爲函數裏所包含的都是爲達成該程式目的的“動作敍述”（ action statement ）。這些敍述所處理的變數可以是由其他函數傳遞過來的，或是和同一模組內其他函數所共用的變數，也可以是整個程式內所有函數都可以使用的變數。此外，函數也可以自行建立一些新的而且只能爲自己所使用的變數。雖然函數和FORTRAN或BASIC 語言裏的副程式很類似，不過它們之間還是有下列這些不同點：

1. 我們不是“呼叫”（ call ）一個函數，而是“啓動”（ invoke ）它。
2. 交由一個函數處理的變數只是這些變數的拷貝，因此即使這些變數的值在該函數內有所改變，原來的變數仍然不受影響。
3. 一個函數在啓動後，會將某個數值傳回原先啓動它的程式，它也可能只是進行某些動作而不作任何資料的傳遞[2]。
4. 一個函數所建立的變數只能在該函數內使用。
5. 一個函數所建立的變數可以是暫態的或永久的。所謂永久的變數，是指該變數的值在啓動函數之初和前一次離開該函數時是一樣的，而暫態變數則需在每次啓動後重新賦予新的值。

一個函數可以啓動其他函數的執行，這些被啓動的“子”函數實際負責了啓動它們的“父”函數工作的一部份。藉著這種二分方式，父函數可將注意力專注在工作的重點，而將工作的細節留給子函數來處理，這種觀念可再加以推廣：我們可以設計一些函數專門來負責許多程式都共同需要的一些細節部份，這些函數可以分別地加以發展、測試及偵錯，而在最後一起保存在程式

---

2. 目前先不必擔心這種限制，因爲不久你就會發現，函數雖然只能傳回少量的資料，不過這些資料可以用來“指向”其他較大量資料的位置。

庫裏成爲一種“軟體工具”（ software tool ），以後程式裏的細節部份就可以藉著啓動程式庫裏的這些軟體工具來完成，而不必再特意去重新寫過。

父函數傳遞給子函數的變數實際上是這些變數的拷貝，因此父函數裏的這些變數是不會受到子函數的任何影響。父、子函數之間資料的交換限於下列這些函數通訊的方式：

1. 變數的拷貝是以參數（ parameter ）的型式傳給子函數。
2. 子函數可將某個數值或資料傳回父函數。
3. 除了父函數所傳送的變數，子函數還可以使用那些同一模組裏或同一程式裏所有函數所共用的變數。

父函數在啓動子函數時可以不用參數來傳遞任何資料，也可以同時使用數種不同型態（ type)的變數（型態的定義將在後面加以說明），至於子函數所傳回來的“東西”，則可以是某種資料或狀態（ status ），也可以什麼東西都不傳回父函數。不論交換的資料是什麼，很重要的一點是，子函數必需正確地掌握父函數所傳給它的參數的意義，同樣父函數也必需正確地了解子函數傳回來的是那些資料。如果父子函數之間沒有足夠的“溝通”，通常都會造成“代溝”的形成——亦即程式的失敗。

除了父函數所傳送的參數，子函數所使用的變數的範圍，可以只侷限於子函數自己或某一模組內（也就是只能爲同一原始檔案內某些函數所使用）。每次啓動子函數時，這些變數所代表的記憶空間可能都不一樣，也可能是同一個記憶空間。因此某些變數在程式的執行進入或離開子函數時所包含的內容，是一些沒有絲毫意義的“垃圾”（ garbage ），而有些變數因爲代表的是相同的記憶空間，所以它的值在子函數啓動之初是和上一次離開子函數時是一樣的，而在程式的執行回到父函數時，這些變數的值都會被保留。

您可能還無法完全了解這些似乎非常晦澀的觀念（原始檔案、模組、函數及變數的範圍）應用的方式，不過只要您耐心的往下讀，您對C的認識會日益加深的。

到目前爲止，我們說明了什麼是原始檔案、模組和函數，我們也以抽象