

附Aztec C65系統使用手冊
(Apple II適用)



語言程式 製作

PROGRAMMING GUIDE

陳原森 譯



序

這本書是爲了教導您如何寫作 C 語言的程式而寫的，此地假設您已經了解到 C 語言比其它語言來得優異的地方；因爲 C 本身有很多種運算與資料的型態，使得 C 成爲一個多采多姿的語言，可以讓您寫作從作業系統到會計系統等等的程式，當您往後閱讀時，就可以更清楚地了解這些與其它的優點了。

這本書在寫作時有兩個基本的假設：(1)學習程式寫作的唯一法門就是寫程式；(2)如果您能夠看到程式的話，學習的工作會簡單些。本書用使您容易閱讀的方法來告訴您如何使用 C 以及爲什麼要用 C；最後還有幾個附錄爲有興趣的讀者進一步探討幾個論題。

每一章都有一些例子來介紹 C 語言的內容。每個例子我們都儘可能地在能夠討論到目前內容的狀況下，寫得簡單一些，我們鼓勵您每一個例題都做一次；有時候，我們也會問一些只有做過了例題之後才能回答的問題，就此而言，本書也可以作爲一本自修的指南。如果是拿來當教本，這些問題倒可以拿來上機，作爲加強了解本書內容的手段。

爲了要讓您與已往的經驗有所比較，書中有時會拿 Basic 語言與 C 語言一起比較，如果您會 Basic，那也許會有助於把經驗轉移到 C 上頭；當然，縱使是根本不曾用過其它語言，也

是可以學C的。有時候我們也會用簡單的圖來說明，讓您可以“看”到某些指令，這對於沒學過其它語言的人來說應該是特別有用的。

我們也嚐試減輕您執行本書中例題的負擔；本書中前五章所有的例題大部份都可以用低於 50 美圓的 C 編譯程式來實習，事實上第五章之後很多例題也是如此。等您學完本書之後，相信您就能夠了解到C的內涵，也許就會買個更有效力的C編譯程式了。

這本書是衆人合力的成果，很多人都對它有所貢獻。特別地，我要感謝 Steve Browning 與 Chris DeVoney，謝謝他們的建議批評；Tim Lesile 永遠能夠挑出小問題；還有 Que 公司的出版與編輯部，他們早就忘了一個星期只工作 40 個鐘頭了，最後，要謝謝 Ron Cain 與他的 C 編譯程式在推廣 C 上面所作的貢獻。

前言

本書的程式與函數都已經在好幾個 CP/M 下的 C 上頭測試過，大多數程式與例題都可以在與 Unix 第 7 版相容的 C 下作業；當然簡單一些的 C 也是可以用的。從第四到第五章的例子用到了浮點 (floating) 變數，在簡單一點的 C 中不一定可以使用。

因為不同出版商之間對 C 的標準程式庫的差異，您應該仔細地看看您的手冊，找出叫用函數時的差異，或者是應該在程式中加上幾列。其中的一個例子就是在程式中加入 `stdio.h` 的問題，有些編譯程式必須要加入這個檔，但有些却只有在用到某些動作，如磁碟檔的讀或寫時才需要它。`stdio.h` 在第四、第八章中討論。

爲了容易閱讀本書的程式、範例、C 的關鍵字與變數名稱都是用 OCR-B 這種字體來打的，字型如下：

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

另外，下面是一個格子尺，以便您算出程式列中的容格數目*：

0	1	1	2	2	3	3	4	4	5	5	6
12345678901234567890123456789012345678901234567890123456789012											

*因為原書程式是用打字完成的，所以有很多錯誤，而且格子也不一定對齊，因此格子尺往往沒有作用；為了方便讀者，每一個程式都已經用 Aztec C65 做過了，譯本中用的是列表機的報表，因此不會出錯。本書後還附有 Aztec C65 的使用手冊。

目 錄

第一章 緒論.....	1
爲何使用 C ?	1
關於讀者的某些假設.....	3
C 程式的基本特性.....	4
C 的函式.....	4
標準 C 函式庫.....	7
C 中的分號.....	9
函式的引數.....	10
綜合說明.....	11
程式規劃風格.....	12
小寫及大寫字母.....	12
大括號的位置及對齊.....	13
程式變數.....	15
變數名稱.....	17
關鍵字.....	17
變數及 printf () 的簡單用法	18

風格摘要	21
第二章 運算子、變數、及迴路	23
運算子	23
算術 / 關係運算子	23
邏輯運算子	27
變數：在 C 中增加 / 減少	29
迴路	32
while 迴路	32
do-while 迴路	35
附錄2 八進位及十六進位數系	43
二進位數系	43
八進位數系	45
十六進位數系	47
第三章 設計自己的函式	51
C 函式的格式	51
型態宣告辭	53
函式名稱	54
引數列	55
引數的宣告	56
函數本體	57
C 中的 return 敘述	58
從函式歸還一個數值	59

變數的範圍、儲存種類、及壽命	60
編譯器及連結器	61
外部變數	63
自動變數	67
值和有範圍的變數	69
靜態變數	70
暫存器變數	71
隱私權及函式	72
設計 C 程式	76
 第四章 指 標	 81
宣告及指標	83
設定指標初值	85
左值及右值	85
左值、右值及指標	86
指標的重要性	89
一個使用指標的程式範例	90
另一個範例	95
增加及減少指標值	101
 第五章 輸入及輸出	 103
使用標準函式庫中的 I / O 函式	103
一個簡單的程式	105
# include	106

# define	107
Null 結束字元	109
累積函式	110
進一步說明字串陣列	112
讀取數值資料	114
程式的輸出：printf ()的選擇	117
scanf ()，通用輸入函式	119
含小寫字母的輸入函式	122
其他狀況	125
switch ()敘述	130
第六章 其他的資料型態	133
基本資料型態	133
基本資料型態的擴充	135
short	135
unsigned	135
long	136
混合資料型態	136
char 及 int	137
轉換及設定	137
二元運算子及混合的資料型態	138
浮點算術運算及倍準確度	139
函式的引數及提升	139
提升及被提升的資料型態	140
使用其他的資料型態	141

轉換 ASCII 字元成浮點數目	142
計算一數的平方根	145
使用不帶符號的資料型態	148
使用 long 資料型態	150
右——左規則	151
資料型態的縮寫	152
多維陣列	153
第七章 結構及組合	157
設定結構的初值	159
使用結構	162
結構和函式並用	163
結構與隱私權	165
在呼叫的函式中改變結構元素的內容	167
傳送整個結構給函式	168
結構及陣列	170
組合	178
運算子的層次及簡寫	181
第八章 磁碟檔案作業	185
低層次對高層次的磁碟 I/O	186
啓開檔案	186
讀取檔案	193
命令列引數：argc 及 argv	194
簡易資料繪圖	197

低層次檔案 I/O	203
open ()	203
read ()	205
write ()	206
close ()	207
使用低層次檔案 I/O	208
其他選擇	211
 附錄八 進一步探討檔案 I/O	 215
FILE 結構	215
和檔案交談	219
stdin , stdout , 及 stderr	219
 第九章 常犯的錯誤及除錯	 221
常見的錯誤	221
遺漏或錯置分號	221
遺漏大括號	223
設定與關係測試	226
程式註解	227
傳給函式的引數是拷貝	228
在函式呼叫中忘了宣告引數	230
在 main () 中忘了宣告函式	231
函式呼叫歸還整數給 main ()	232
指標在被設定初值之前含的是垃圾	233
徹底的愚蠢	234

除錯	235
錯誤的種類	236
語法錯誤	236
程式錯誤	236
潛伏性的臭蟲	236
錯誤的偵測及分離	238
結論	242
附錄A ASCII 碼	243
附錄B 供應中的 C編譯器	247
附錄 C 語法摘要	251

第一章

緒 論

(An Introduction to C)

本書主要目的是要在最短時間裡，教你使用 C 語言來撰寫程式。書中使用許多簡單例子說明 C 語言的每一觀念，這些例子均可借價格便宜的 C 編譯器（Compiler）來執行。因此，當您在研究 C 時，不必投資大量的金錢。

當你研讀此書時，手邊若有 C 編譯器，效果將會更佳。附錄 B 中，列有各種價錢合理的 C 編譯器，即使使用其中最便宜（低於 \$ 20.00）的，亦可執行前五章中的所有程式，後面幾章中的程式，則需要功能較齊全的 C 編譯器才能執行；到那時候，你就會發現 C 是值得投資的。

為何使用 C？（Why C？）

C 具有許多優於其他程式語言的特點。此健全的語言具有種類齊全的運算子（operator）及命令（commands）可供撰寫任何軟體程式，上至作業系統（operating system）下至會計套裝軟體（package）。事實上，市面上許多 C 編譯

2 C 語言程式製作

器即是以 C 撰寫的。

C 是一種可携性的語言 (portable language)。在某一電腦上執行的 C 程式，只要經過少許的修改即可在另一部擁有 C 編譯器的電腦上執行。“只寫一次”的理想是可惜 C 來實現的。

另一優點是 C 的執行速度。若你是慣用諸如 BASIC 的解釋性語言 (interpreted language)，而未使用過 C，那你就會覺得很驚訝。比如說，對於第二章中一個只是單純的使變數由 0 增至 30,000 的程式，非正式的測試結果，以 BASIC 執行要花 96 秒，但以 C 執行只需 2 秒。

可携性及速度二種效益的結合會產生另一副效益。對執行時間要求極嚴的情況，如排序 (sorting)，利用組合語言 (assembly language) 來撰寫程式是一種解決之道。但問題就出在當更換新的中央處理單元 (CPU) 時，程式設計師不得不再重新學習新 CPU 的指令。更新訓練所需的經費是相當可觀的，因此許多軟體公司已採用 C 程式語言作為發展系統的程式語言。

此種邏輯也適用於你，雖然要花點時間學習 C，但在新的 CPU 上市後，就不必重新學習新的語言。好的 C 編譯器使得以 C 和組合語言撰寫的程式，在執行上除了對速度要求極嚴格的情況外，其差異是可被忽視的。

另一優點是，C 本身隱含結構化程式規劃技巧 (structured programming technique)，強迫你顧及功能模組 (modules) 及區段 (block)。每一區段具有特殊的目的或功能 (function)。C 程式即是一些工作模組的排列，模組化

的發展使得程式的偵錯及維護更為容易。

最後，C 也是一種極有趣的語言——不但易於學習，且彈性極大。藉著建立自己的功能模組，您可用 C 做任何事情。若你有心的話，更可用 C 設計自己的語言！不論如何，讓我們先完成手邊的工作——學習 C。

關於讀者的某些假設

(Some Assumptions about the Reader)

本書對你做了某些假設——第一，你會使用電腦、本文編輯器 (text editor) (將程式輸入電腦，並把它存入磁碟)，及 C 編譯器。那一類型的編譯器並不重要，只要能用來測試書中的範例即可。因為你不能只“讀”語言，要真正動手去做才行。

第二個假設是，你熟悉某些程式規劃的技巧，最好是熟悉 BASIC 語言。但此假設並不是必要的，不要因此而裹足不前；甚至於你不會寫程式，也可開始學習 C。文中是一步一步的介紹以 C 撰寫的程式或副程式，並和以 BASIC 撰寫的程式互相對應比較。BASIC 是用來建立設計程式的一些觀念。

最後一個假設是，你不必在明天晚上前就熟悉 C 語言。某些章節是要你花一點時間仔細去考慮一番。採用這種方式是因為 C 宛如一座金字塔，需要穩固的基礎，在繼續閱讀下一章前，一定得熟悉前面的所有的章節，實際去演練每一個範例是學習的捷徑，了解文中的例子和設計自己的程式是兩回事。

C 程式的基本特性

(Fundamental Characteristics of C Programs)

C 的函式 (Functions in C)

整個 C 程式可視為一群稱為函式 (function) 的區段所堆成的。函式是一群為了完成某項工作而設計在一起的一個或數個敘述 (statement)。請看圖 1-1 中的程式：

圖 1-1

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

main()
{
    printf("This is my first program.\n");
}

10 REM This BASIC program does the same thing
20 PRINT "This is my first program."
30 END
```

程式先以 C 撰寫，再以 BASIC 寫出。二者均會印出：

This is my first program。在 C 中，程式的註解 (comment) 或說明 (remark) 是以左斜線及星號的組合 (/*) 開頭，而以二者的相反組合 (/*) 作為結束，編譯器會忽略介於此二者之間的任何文字。此註解就像 BASIC 中的 REM (亦即，REMark) 敘述。均為不執行的敘述。

提示：因為編譯器忽略介於 `/*` 及 `*/` 間的任何文字，這些註解文字可能是極有用的偵錯工具。若你欲從測試中的程式，移去某一行，則只需在此行的兩端加上註解符號即可。亦可使用此法。移去整段程式，以免日後再重新打入一次，註解符號被除去後，整段程式又會復原。

main () 函式 (The main () Function)

特殊函式 `main()` 指出 C 程式開始執行的地方。每一程式均要有一個 `main()` 函式以告知編譯器程式從那裡開始。同時在程式中亦只能出現一次，若使用了數個 `main()` 函式，編譯器無法辨別那一個才是說明程式開頭的 `main()` 函式。

文中提及函式時，均在其後加上一對左、右括號，以指明它是一個函式名稱。例如：括號幫助你區別名為 `main()` 的函式及名為 `main` 的單字。

大括號 (Braces)

在圖 1-1 的 `main()` 中，`m` 下方的左大括號指出函式本體 (function body) 的起點。函式本體為完成某一特定工作的一個或數個敘述。

先不考慮 `printf()`。在程式下方有一右大括號 (`)`)，指出函式本體的終點。因此左、右大括號則會括住形成函式本體的敘述。圖 1-2 圖示這些符號的用法。點表示本體中的敘述。