C++程式設計工程師手冊

序言

這本書是寫給首次接觸電腦程式設計的初學者，以及有志於把程式設計作為發展未來應用專業技能的一項重要工具的程式人員。

七年來，筆者在大一的「計算機概論」課程中教授 C/C++程式設計。本書即匯整歷年的上課講義，經過編修和去蕪存菁而成，內容適合大專院校相關課程作為教科書或參考用書。本書分成八個主要子題，相對於一般程式語言的教科書，其安排較為精簡緊湊，因此也適用在 C/C++程式設計的短期或密集訓練課程中使用。

市面上已經有很多 C/C++程式設計的用書，不論是中文或英文版本，其中不乏入門的、或教學導向的或技術實作導向的 C/C++語言教科書(有興趣的讀者可參考本書最後的參考書目)。那麼，為什麼還要寫這本書呢？

為什麼寫這本書？

寫這本書的動機有三個。第一是針對非資訊專業的理工背景學生或工程師，培養其良好的程式設計應用專業能力；第二、從解決問題的面向─而非從語言語法的面向來學習程式設計；第三、以簡易實用的、而非羅列式(exhaust)主題的方式來介紹程式設計。

從工程與科學計算的角度出發

對於資訊專業的人員來說，程式設計通常與計算機結構、演算法、資料結構，或是軟體工程聯繫在一起，或是作為後者一個重要的基礎技能。另一方面，對於許許多多非資訊專業的理工背景學生或工程師而言，程式設計是一個非常重要的解題工具，它可以協助工程師計算、分析、設計、模擬、控制、測試、實驗各式各樣的工程和科學問題和系統。但與資訊專業人員不同，工程師們最關心的問題通常是：怎樣把領域知識轉成計算機解題的形式？C/C++語言提供了必要的工具和方法，得以幫助工程師實現這個目標。

可惜的是，只有少數的 C/C++程式設計用書，從這個角度切入來引導初學者進入程式設計的領域。譬如說，多數的書都會花很大的篇幅和實例來解說指標(pointer)的原理和使用技巧，這是一個令很多 C/C++初學者感到頭痛的主題，也是造成程式除錯的困難和程式執行時不穩定狀況發生的最主要來源之一，但很少有工程師在學習時候就知道，在計算的用途上，我們可以把指標的用途限

定在矩陣的動態記憶體配置，而在其他場合，盡量避免使用指標，而仍可達到相同的程式效能。重要的是，我們的程式因此可以更方便除錯、程式碼更容易閱讀、系統更容易維護和擴充。

就如同上述的例子，在這本書中，我們將強調 C/C++與這類解題實務的聯繫，而不僅就一個電腦語言的技術內容作介紹。對於多數沒有機會進一步學習計算機結構、演算法、資料結構，或是軟體工程等高階電腦課程的理工科系學生來說，這樣的安排無疑是較為妥適且實用的。

程式設計的目標在於解題(Problem Solving)

如何評價一個程式設計人員？不同技術領域和時空背景可能會有不同的觀點和標準，但有一個指標幾乎所有人都同意的，那就是：一個好的程式設計人員必須具有應用電腦程式來解決實務問題的能力。

解決實務問題能力的培養不全然靠經驗。事實上，它更應該是建立在一個結合了理論、方法、工具的一個有條理的流程，它可以指引工程師用一種嚴謹的、系統化的方式去解決類似的、甚至一個從未被解決過的問題。

本書在介紹每一個主題或每一個語法時，都會伴隨一個範例程式，來展示一個解題的類型(pattern)，接著每一節結束時都有即時演練的題目，得以讓學習者藉由前面的類型範例，參考其解題程序—比如宣告變數、給定初值、應用適當的流程控制語法、結果輸出等，來完成相似的解題程式的設計。

極簡主義的設計哲學

設計一個工業系統或計算系統，有很多的品質評量方式，例如性能、效率、可靠度、程式碼大小、記憶體使用量等等，很少有程式能夠同時滿足所有的目標，因此就有各種程式設計的技巧。使用這些技巧確實對程式設計者達成特殊需求的系統設計有很大的幫助。但是另一方面，過於強調這些技巧反而經常造成初學者的學習上的障礙。因為它們多數不容易結構化、程式碼不容易閱讀、程式除錯困難、以及日後程式的修改和擴充困難。

在這個背景下，簡單(simplicity)反而突顯出重大的優點。簡單的程式碼風格讓程式更符合程式設計者和應用系統開發者的直覺，因此容易閱讀和溝通，也得以設計出更可靠的、更容易維護擴充的程式模組，此外，整體程式開發的時間更因而縮短。

如何設計簡單的 C/C++程式？一個參考原則是：不要過於倚賴特殊指令和技法，而用直覺而且清晰的方式來撰寫程式。例如避免使用 goto 指令；少用遞

迴(recursive)結構；可以使用 for 迴圈的地方就不要使用 while 迴圈；儘量少用指標(pointer)；使用結構化的資料結構–陣列(array)來取代其他如 List 之類複雜的資料結構等等。

簡單的程式風格並不總是意味犧牲程式的效率，相反的，由於結構清晰，很容易依據新的任務需求去調整程式碼。一般而言，我們的程式都是在個人電腦上撰寫、編譯和執行。一旦需要將程式移植到嵌入式系統(embedded system)平台，執行速度、程式碼大小和記憶體使用量可能會變成關鍵性的考量，此時，一個簡單、清晰的原始碼將使我們針對最佳化的目標進行修改和測試變得更容易。

本書並非提倡學習簡單的程式設計，相反的，我們的理念是以儘可能簡單的方式來撰寫最複雜的解題程式。簡單也不是理所當然，這種能力必須經由學習得來。本書以很大的努力來協助讀者進行這樣的學習。

為什麼學習 C/C++?

C 語言在程式設計和電腦應用系統開發的領域，有著無可比擬的重要地位。實際上，幾乎所有當代電腦程式語言例如 Perl、Java、C++、C #，幾乎都建立在 C 語言的基礎上。小從智慧型手機或 PDA 裏執行的作業系統，到個人電腦、Unix 工作站，到超級電腦內部，絕大部分所採用的作業系統都是以 C 語言開發出來的。

80 年代以後，軟體開發的複雜度大幅提高，傳統 C 語言不再能夠滿足大型軟體系統設計的需求，因此誕生了物件導向的 C++語言，它在 C 語言的語法裏添加了許多物件導向程式設計的元素。由於 C++語言向上相容 C 語言，因此多數的編繹器(compiler)都是同時支援 C 語言程式、C++語言程式和 C/C++混合語言程式的編繹工作。

在內容的安排上，本書融合了 C 語言與 C++語言的語法。由於 C++語言是 C 語言的後繼者，所以 C++語言擁有許多比 C 語言優越的特徵，譬如物件導向程式語法的支援、串流型態的輸入輸出，但另一方面，C++語言同時也形成了更複雜龐大的語言結構，對初學者或者和工業界從事嵌入式系統設計的工程師來說，C++程式似乎是一個不合時宜的龐然怪獸，他們多數寧可選擇 C 語言，來撰寫小巧、但是實用的程式。

為了精簡的目的，本書並不疊加 C 語言和 C++語言的內容，而是基於設計實務的需求，以 C 語言為基礎架構，再加上選擇性的 C++語言的部分語法特徵加以延伸，整合成本書的骨幹。書中，我們多數採用 C++的串流型態(streaming)的輸入輸出來取代 C 語言的格式化(formatting)輸入輸出，因為它讓程式更

精簡、更容易撰寫和使用。另外，我們也採用 C++的動態記憶體配置方法，也是因為它比傳統 C 語言的記憶體管理方法更容易使用，同時具有更大的彈性。本書的第 8 章介紹 C++的物件導向的程式設計，對於 C 語言設計者來說，這是一個全新的主題。我們考量物件導向方法可以為一個大型、複雜的系統設計帶來模組化的好處，但基於作為入門教科書的簡易性考量，我們也省略了一些過於精細的物件導向技巧，例如多重繼承等。這類主題即使不是沒有機會使用，我們也不建議在一個講求精簡、效率和可靠性的工程計算導向問題中使用這些高階但相對複雜的物件導向技巧。

長久以來，工業界絕大多數的工程系統都是以 C/C++語言進行開發，因此相對於其他電腦語言，C/C++相關的參考資源最為豐沛，很多的解題類型和演算法都很容易找到 C/C++的參考程式碼，加上最為廣泛的平台支援，包括 8051 微控制器、ARM 系統單晶片、MAC/PC 個人電腦、工作站等等，無一不支援 C/C++程式設計。

1995 年以來，由於網路應用程式開發和跨平台運算的潮流，帶動了一個新的程式語言—Java 的興起。但是 C/C++語言的需求並沒有因為消退，反而由於所謂的後 PC 時代，系統設計的平台從 PC 移轉到各類資訊家電的嵌入式系統，甚至移轉到 IC 設計領域的 SoC，其中軟體和韌體(firmware)的設計，C/C++語言都扮演著愈形重要的角色。

如何使用這本書？

針對所有初學者，我們建議從第 1 章到第 8 章循序漸進的學習，學習過程中，一面參考範例程式，瞭解指令和相關語法的應用，並藉以熟悉解題程序；接著會有相同類型的練習題，只要將範例程式稍加改寫，應該很容易解決這些練習，所以本書並未附上這些練習題的解答程式碼。本書每一章結束都有一些綜合程式練習，讀者可以自我測驗是否對於整章的內容有充分的掌握。至於第 4、7 章之後，我們提供了許多精選的程式實務演練程式，讀者可以先自行撰寫解題程式，再比對書中提供的解題程式。任何的題目都不會有唯一的解答，儘量以不同的思考邏輯和嘗試不同的語法去解題，因為越多元的解題策略和思考方向越能夠厚實未來解題的實力。

對於有程式設計經驗的工程師，本書提供了許多設計參考方案，讀者可以選擇感興趣的主題，學習範例程式的解題方法；或者可將本書作為工作時的隨身參考手冊。

至於未來想從事嵌入式系統設計或系統單晶片(System-On-A-Chip)設計的工程師們，除了上述的學習目標外，我們希望他們還可以從本書學習到精簡、可靠的程式撰寫風格。這對他們未來在快速產品上市(Time-To-Market)的壓力

下從事系統設計會有很大幫助。