電腦、網際網路與全球資訊網簡介

1

學習目標

在本章中，你將學到：

- 電腦基本概念。
- 不同類型的程式語言。
- C 程式語言的演進歷史。
- C 標準函式庫的功能。
- 典型的 C 程式開發環境元件。
- C 語言如何爲以後學習程式語言提供必備的基礎，特別像是 C++、Java 以及 C#等程式語言。
- 網際網路與全球資訊網的沿革。

本章綱要

1.1 簡介
1.2 電腦硬體與軟體
1.3 電腦的架構
1.4 個人式、分散式及用戶端／伺服器的計算環境
1.5 網際網路與全球資訊網
1.6 機器語言、組合語言以及高階語言
1.7 C 的沿革
1.8 C 標準函式庫
1.9 C++
1.10 Java
1.11 Fortran、COBOL、Pascal 以及 Ada 程式語言
1.12 BASIC、Visual Basic、Visual C++、C# 以及 .NET
1.13 關鍵的軟體工程趨勢：物件技術
1.14 典型的 C 開發環境
1.15 硬體發展的趨勢
1.16 關於 C 語言以及閱讀本書要注意的一般事項
1.17 資源網站

1.1 簡介

- 本書的特點是藉由結構化程式設計技巧 (structured programming)，來提昇程式的清晰性 (clarity)。
- 讓你能在一開始就以正確的方式學習程式設計。
- 本書含有數以百計的可執行程式，並且列出這些程式在電腦上執行的結果。
- 我們稱之為「實況程式碼 (live-code)」教學。所有的範例程式都可以從我們的網站 www.deitel.com/books/chtp6/ 上下載。
- 電腦 (通常稱作硬體，hardware) 由軟體 (software) 所控制 [軟體就是人撰寫的指令，可讓電腦執行動作 (action) 並做判斷 (decision)]。

1.1 簡介

- 本書將介紹如何使用C語言來撰寫程式，C語言是在1989年由美國國家標準局 (American National Standards Institute，ANSI) 訂定的標準 (稱為ANSI X3.159-1989)。接著由國際標準組織 (International Standards Organization，ISO) 推廣到全世界。
- 我們稱之為標準C。
- 我們也會在本書中介紹C99－C標準的最新版本。
- 新的C標準目前正在建立中，它被非正式的命名為C1X，可能會在2012年左右發表。
- 選讀的附錄E將討論 Allegro 遊戲開發的C函式庫。

1.1 簡介

- 此附錄將介紹如何使用 Allegro 來建構一個簡單的遊戲。
- 我們將展現如何顯示圖形及動態物件，並解釋其它的功能如聲音、鍵盤輸入以及文字輸出。
- 此附錄列出一些網頁連結與資源，包含了超過 1000個 Allegro 遊戲以及進階 Allegro 技巧的教學。
- 由於軟硬體的技術快速發展，電腦的價格急速地下降。

1.1 簡介

- 數十年前推出的龐大電腦稱為「大型主機(mainframes)」，發展至今，已廣泛運用於商業、政府機構以及工業中。
- 矽晶片技術已使得電腦十分經濟實惠，近十億部一般用途的電腦已廣泛地使用在商業、工業、政府部門及日常生活中。
- 還有十億台特殊用途的電腦用在智慧型電子產品上，像是汽車導航系統、節能裝置和遊戲控制器。

1.1 簡介

- 以C語言為基礎的物件導向程式語言——C++，現在受到更多的人喜好，我們將會在後面的章節詳細介紹C++以及物件導向程式設計。
- 您可以在下列網站訂閱免費的Deitel Buzz Online電子報，我們會將Deitel & Associates相關的C和C++最新開發訊息寄給您：
  www.deitel.com/newsletter/subscribe.html
- 以下網頁包含了C及相關資源中心的列表：
  - www.deitel.com/ResourceCenters.html

1.1 簡介

- 當你閱讀本書與C相關的章節時，以下的資源中心會對你有幫助：C、Code Search Engines and Code Sites、Computer Game Programming 以及 Programming Projects。
- 本書的勘誤和更新訊息會發表在以下網頁：
  - www.deitel.com/books/chtp6/

1.2 電腦硬體與軟體

- 電腦是一種裝置，能夠以人類數十億倍的速度，執行計算和邏輯判斷。
- 現今許多個人電腦可以在一秒鐘內執行數十億次加法。
- 現今最快的超級電腦 (supercomputers) 一秒可執行數千兆 (quadrillions) 個指令！
- 也就是說，這樣的電腦可以替地球上的每一個人每秒執行100,000次計算！

1.2 電腦硬體與軟體

- 電腦受到稱為電腦程式 (computer programs) 的一組指令控制來處理資料 (data)。
- 這些電腦程式指揮電腦去執行一連串有次序的動作，這些動作都是由電腦程式設計者 (computer programmers) 預先指定的。
- 電腦是由各種裝置 (如鍵盤、螢幕、滑鼠、硬碟、記憶體、DVD以及處理器) 所組成的，我們稱它們為硬體 (hardware)。
- 在電腦上執行的程式則稱為軟體 (software)。
- 近幾年硬體的費用已大幅降低，重要的是個人電腦已經成為一種普遍的商品。

1.2 電腦硬體與軟體

- 你可從本書中學到建構軟體的方法，大幅降低開發軟體所需的成本，這些方法就是：結構化程式設計(在C相關章節中) 以及物件導向程式設計 (在C++章節中)。

1.3 電腦的架構

- 每部電腦實際上都可分成六個邏輯單元 (logical units)：
  - 輸入單元 (input unit)：這個用來「接收資訊」的區域，會從輸入裝置 (input devices) 取得資訊 (資料與電腦程式)，並將此資訊放在其它裝置上以進行處理。人類通常透過鍵盤和滑鼠裝置將資訊輸入電腦。資訊也可用許多其它方式輸入，如語音輸入、掃瞄影像、讀取條碼、從輔助儲存單元讀入 (例如硬碟、CD與DVD光碟機、以及USB裝置 — 也稱作姆指碟)，也可透過網際網路接收資訊 (像是從YouTubeTM下載視訊或是從Amazon下載電子書)。

1.3 電腦的架構

- 輸出單元 (output unit)：這個用來「輸出資訊」的區域會接受經電腦處理過的資訊，並且將資訊送到不同的輸出裝置 (output devices)，讓這些資訊能夠在電腦之外使用。目前大部分電腦輸出的資訊會顯示在螢幕上、印在紙上、由音樂播放器播放出來 (像是Apple大受歡迎的iPods)，或用來控制其它裝置。電腦也可將它們的資訊輸出到網路上，例如網際網路。

1.3 電腦的架構

- 記憶單元 (memory unit)：這是電腦中一個存取快速、但容量相對較低的「倉庫」區域。它將輸入單元接收到的資訊保存起來，需要時，可以馬上運用這些資訊。記憶體單元也將電腦處理過的資訊儲存起來，直到該資訊可透過輸出單元送到輸出裝置為止。記憶體中的資訊是「揮發性的」(volatile)－當電腦的電源關掉時，裡面的資訊就不見了。記憶體單元又稱為記憶體 (memory) 或主記憶體 (primary memory)。

1.3 電腦的架構

- 算術和邏輯單元 (arithmetic and logic unit，ALU)：這是「生產製造」的區域，它負責執行如加、減、乘、除等計算。它所包含的判斷機制能讓電腦做運算，例如比較記憶單元中的兩個項目，判斷它們是否相等。在今日的系統中，ALU通常是下一個邏輯單元，也就是CPU的一部分。

1.3 電腦的架構

- 中央處理單元 (central processing unit，CPU)：它是電腦「執行管理」的區域，負責協調監督其它區域的作業。需要將資訊讀入記憶單元時，CPU會通知輸入單元；需要將記憶單元的資訊進行計算處理時，CPU就會通知ALU加以處理；需要將記憶單元的資訊傳送到某個輸出裝置時，CPU就會通知輸出單元。今天許多電腦都具備多個CPU，因此能同時執行許多操作，這種電腦稱為多重處理器系統 (multiprocessors) 。多核心處理器 (multi-core processor) 將多個處理器放在一個積體電路晶片中－例如雙核心處理器有兩個CPU，四核心處理器則有四個CPU。

1.3 電腦的架構

- 輔助儲存單元 (secondary storage unit)：這是電腦長期、高容量的「倉庫」區域。　　目前沒有被其它單元使用的程式或資料，通常都放在輔助儲存單元 (例如你的硬碟)，直到再度需要使用它們時才會加以讀取，這可能相隔幾個小時、幾天、幾個月甚至幾年之久。因此，儲存在輔助儲存裝置中的資料為「永續性」(persistent) 的，當電腦的電源關閉時，這些資料還是會留存。輔助儲存單元的存取速度比主要儲存單元慢，但每單元的儲存成本少很多。輔助儲存裝置包含了CD、DVD和隨身碟 (或記憶卡)，它們可儲存上億或上百億個字元。

1.4 個人式、分散式及用戶端／伺服器的計算環境

- 1977年，蘋果電腦公司 (Apple Computer) 推廣所謂的個人電腦概念 (personal computing) 。
- 1981年，世界最大的電腦供應商IBM推出了IBM個人電腦。
- 此舉將個人電腦快速推廣至商業、工業與政府機構的組織，這些組織過去曾經大量使用IBM大型主機。
- 但這些電腦仍是獨立的個體，人們彼此交換磁碟片來分享資訊 [通常稱為「私密網路」 (sneakernet)]。
- 這些機器能彼此連結起來形成電腦網路，有時透過電話線，有時透過企業組織內部的區域網路 (local area networks,LAN) 進行連結。

1.4 個人式、分散式及用戶端／伺服器的計算環境

- 由此產生了分散式計算的概念。
- 資訊可輕易在電腦網路上共享，伺服器電腦 (servers，例如檔案伺服器、資料庫伺服器、網站伺服器等等) 提供某些功能，可供分散在網路四處的用戶端 (client) 電腦使用，因此又稱為用戶端/伺服端 (client/server) 計算環境。
- C已成為撰寫作業系統、電腦網路和分散式主從式應用程式所廣泛使用的程式語言。

1.5 網際網路與全球資訊網

- 自從全球資訊網 (World Wide Web) 推出後，使用者便能夠透過網際網路搜尋到幾乎任何主題的多媒體文件，而網際網路也因此成為世界上最重要的通訊工具。
- 如今應用程式已可與全世界的電腦通訊。

1.6 機器語言、組合語言以及高階語言

- 程式設計者可用各種程式語言撰寫指令，有些程式語言可由電腦直接讀取，有些則須經過中間轉譯 (translation) 步驟。
- 電腦語言可分成三大類：
  - 機器語言
  - 組合語言
  - 高階語言
- 任何電腦都只能直接讀懂它自己的機器語言 (machine language)。
- 機器語言是電腦的「母語」，此語言由電腦的硬體設計所定義。

1.6 機器語言、組合語言以及高階語言

- 機器語言通常稱為「目的碼」(object code)。
- 機器語言通常由一連串數字組成 (最後可分解成1和0)，它們可用來指揮電腦一次執行一個電腦最基本的操作。
- 機器語言與機器有關 (machine dependent)，也就是說某種特殊的機器語言只能用在某種電腦上。

1.6 機器語言、組合語言以及高階語言

- 機器語言對人類來說既麻煩又難懂，例如以下這段機器語言，可將加班費加入基本薪資，然後將結果存入總薪資：
  - +1300042774
    +1400593419
    +1200274027
- 因此，程式設計者開始用類似英文的縮寫字代表基本操作，而不使用電腦能直接了解的數字字串。
- 這些縮寫字構成組合語言 (assembly languages) 的基礎。

1.6 機器語言、組合語言以及高階語言

- 名為「組譯器」(assemblers) 的轉譯程式 (translator program) 可利用電腦的快速將組合語言轉成機器語言。
- 以下這段組合語言程式也可加班費加入基本薪資，然後將結果存入總薪資：
  - ```
    load    basepay
    add     overpay
    store   grosspay
    ```
- 雖然對人類而言這種程式碼較清楚易懂，但除非轉譯成機器語言，電腦仍無法理解。
    ```

1.6 機器語言、組合語言以及高階語言

- 因為組合語言的出現，使電腦的使用率快速增加，但程式設計者仍要對電腦下許多指令，即便是最簡單的工作也一樣。
- 為了加快程式設計的過程，就發展了高階語言 (high-level languages)，它只需單一敘述 (statement) 就能完成不少工作。
- 名為「編譯器」(compiler) 的轉譯程式 (translator program) 可將高階語言程式轉成機器語言。
- 高階語言可讓程式設計者以近似於日常英文的用語，和一些常用的數學符號來撰寫指令。

1.6 機器語言、組合語言以及高階語言

- 使用高階語言撰寫的薪資計算程式，可能有以下敘述：
  - `grossPay = basePay + overTimePay;`
- C、C++、微軟的.NET語言（諸如Visual Basic、Visual C++、Visual C#）和Java正是功能最強大，使用最廣泛的高階語言之一。
- 直譯器 (Interpreter) 可直接執行高階語言程式（可省去編譯所花的時間），但其執行速度比編譯好的程式要來得慢。

1.7 C 的沿革

- C是由B和BCPL這兩種語言所發展出來的。BCPL語言是Martin Richards在1967年時為了撰寫作業系統和編譯器時所發明的。
- Ken Thompson將BCPL上的許多功能加以模式化後，移到他的程式語言B中使用，並於1970年在貝爾實驗室 (Bell Laboratories) 用B建立了早期的UNIX版本。
- BCPL和B都是「弱型別」的語言－即每項資料都佔有記憶體中的一個字組“word”。而要將某項資料看成是整數或其他種資料型別，則必須由程式設計師一肩挑起這項負擔。

1.7 C 的沿革

- C語言是在1972年由貝爾實驗室的Dennis Ritchie由B語言所發展出來的，原先是用在DEC PDP-11電腦上。
- C語言最初是以開發 UNIX 作業系統而聞名。
- 如今，幾乎所有新的主要作業系統都是以C和C++撰寫而成的。
- C程式可以適用於大多數的電腦。
- C是一種與電腦硬體無關的語言。
- 只要精心設計，就有可能寫出具可攜性 (portable)的C程式，在大部分電腦均可執行。

1.7 C 的沿革

- 到了1970年代晚期，C演變為現今的「傳統C語言」。1978年Kernighan和Ritchie出版了"The C Programming Language"，引起對C語言廣大的注意。
- C語言在各種電腦 (有時稱為硬體平台，hardware platform) 上快速發展，並且產生了許多的版本。這些版本相當類似，但是卻不相容。
- 「標準C」(Standard C)在1989年通過審查，而且在1999年則進行了修訂。
- C99是一種修訂後的C程式語言標準，它精鍊並擴充了標準C的功能。

1.7 C 的沿革

- 並不是所有常見的 C 編譯器都支援 C99。
- 在有提供C99支援的編譯器中，大都也只支援新功能的一部份。
- 本書第1章到第14章的基礎是已被廣泛採用的國際標準(ANSI/ISO)C。
- 附錄G則會介紹C99，並提供常用的C99編譯器以及IDE的連結。該附錄會將C99功能與前面的章節做一個連結，方便讀者學習C99。

可攜性的小技巧 1.1

因爲 C 是與硬體無關且被廣泛使用的程式語言，所以用 C 寫的應用程式只需稍加修改，甚至不需修改，就可在許多不同的電腦系統上執行。

1.8 C 標準函式庫

- 你將會在第五章中學到，C程式是由許多稱為函式 (functions) 的模組或是片段所構成的。
- 您可以自己設計所有要用到的函式，不過大多數的C程式設計師都會利用一套現成的函式，稱為C標準函式庫 (C Standard Library)。
- 以下網站有完整的C標準函式庫文件，包含C99的功能：
  - www.dinkumware.com/manuals/default.aspx#Standard%20C%20Library
- 我們鼓勵讀者使用建構區塊 (building-block approach) 的方式來撰寫程式。

1.8 C 標準函式庫

- 避免重新設計軟體。
- 儘量地利用現有的區塊，這就是軟體再利用（software reusability），它是物件導向程式設計的主要關鍵之一，讀者將會在學習C++時發現這點。
- 使用C++開發程式時，您通常會使用以下的程式區塊：
  - C標準函式庫所提供的函式。
  - 自己撰寫的函式。
  - 其他人撰寫並提供給你使用的函式。

1.8 C 標準函式庫

- 假如你利用現成的函式，可以避免重新設計軟體。
- 以標準C函式為例，你知道它們都經過謹慎的設計，也知道你所使用的這些函式在幾乎所有的標準C實作中都能夠使用，因此你的程式具有較高的可攜性及較少的錯誤。

增進效能的小技巧 1.1

儘量利用標準 C 函式庫所提供的函式，而不要自己撰寫。因爲標準函式都是經過專家精心設計的，效率會比較好。

增進效能的小技巧 1.2

儘量利用標準 C 函式庫所提供的函式，而不要自己撰寫。因爲標準函式適用於所有標準 C 的實作，所以可攜性較高。

1.9 C++

- C++是由貝爾實驗室的Bjarne Stroustrup所發展出來的。
- C++根源於C語言，而且增加了許多功能使得C語言變得更好。
- 最重要的是它提供了物件導向程式設計 (object-oriented programming) 的功能。
- 物件是模擬真實世界項目的可重複使用軟體元件。
- 模組化、物件導向的設計與實作方式，比傳統的程式設計技術更具生產力。

1.9 C++

- 在後面的章節中，我們將簡要地介紹C++，其內容選自另一本書<<C++程式設計藝術第七版>>。
- 在學習C++時，也可以參考我們的線上C++資源中心，其網址為 www.deitel.com/cplusplus/.

1.10 Java

- 微處理器對智慧型消費性電子商品已造成了很大的影響。
- 基於這項認知，Sun Microsystems建立了一種以C++為基礎的語言，最後稱為 Java。
- 1993年全球資訊網開始蓬膨發展，而Sun看見了一項重大的商機，即利用Java來撰寫具有動態內容(dynamic content，像是互動性和動畫等等) 的網頁。
- 由於全球資訊網的緣故，Java卻在商業界引起立即且高度的興趣。

1.10 Java

- Java目前被用來開發大型的商用軟體，增強網站伺服器（提供我們由瀏覽器上看到之內容的電腦）的功能，提供消費性電子商品的應用程式 (如行動電話，呼叫器和個人數位助理) 等等用途上。

1.11 Fortran、COBOL、Pascal 以及 Ada 程式語言

- 現在已經有數以百計的高階語言，但只有少數幾種有廣泛的接受度。
- FORTRAN (FORmula TRANslator) 語言是在1950年代由IBM發展，主要是運用在需要用到複雜的數學運算的科學以及工程應用方面。
- 在工程的應用程式上，FORTRAN現在仍然被廣泛使用。
- COBOL (COmmon Business Oriented Language) 是在1950年代晚期由電腦製造商，政府部門和工業電腦的使用者共同發展出來。

1.11 Fortran、COBOL、Pascal 以及 Ada 程式語言

- COBOL主要用來當作商業用途，也就是對大量的資料需要精確且有效率的應用領域。
- 許多商業軟體依舊是使用COBOL來撰寫的。
- 在1960年代期間，許多大型軟體的發展都碰到嚴酷的考驗。
- 人們開始了解發展軟體遠比他們想像的還要複雜。
- 在1960年代的研究成果產生革命性的概念－結構化程式設計 (structured programming)，一種井然有序的撰寫程式方法，比先前技術所寫出來的大型程式更為清晰，且更易於測試、除錯及修改。

1.11 Fortran、COBOL、Pascal 以及 Ada 程式語言

- 這種研發風潮的一個更具體的結果，就是促成Pascal程式語言的發展。Pascal程式語言是在1971年由Niklaus Wirth教授發展。
- Pascal是以第十七世紀的數學家和哲學家Blaise Pascal的名字來命名，而且也是為了結構化程式設計的教學而設計，並且快速成為大多數學院偏好的程式語言。
- 不幸地，Pascal語言缺乏許多用在商業、工業和政府使用的應用程式所需的功能，因此它沒有廣泛地在學術界之外被接受。

1.11 Fortran、COBOL、Pascal 以及 Ada 程式語言

- Ada程式語言在1970年代以及1980年代早期由美國的國防部(DOD)贊助之下發展。
- 數以百計的語言是用來生產DOD的龐大命令及控制軟體系統。
- DOD想要以一個單一語言來滿足大部份的程式需求。

1.11 Fortran、COBOL、Pascal 以及 Ada 程式語言

- Ada的名稱取自詩人拜倫的女兒 Lady Ada Lovelace的名字。
- Lady Lovelace一般公認在1800年代初期，寫出世界上第一個電腦程式 (是為了配合Charles Babbage所設計的機械式計算裝置 Analytical Engine所撰寫) 。
- 一個重要的Ada的功能叫做多工 (multitasking)，允許程式設計師同時執行許多程序。

1.12 BASIC、Visual Basic、Visual C++、Visual C# 以及 .NET

- BASIC (Beginner's All-Purpose Symbolic Instruction Code)程式語言在1960年代由Dartmouth學院的John Kemeny和Thomas Kurtz教授發展的，用來撰寫簡單的程式。
- BASIC基本的目的是使新手熟悉程式設計的技術。
- 微軟在1990年代初期推出Visual Basic語言，使微軟視窗應用程式的開發更為容易，現在已經成為全世界最受歡迎的程式語言之一。

1.12 BASIC、Visual Basic、Visual C++、Visual C# 以及 .NET

- 微軟最新的開發工具配合其公司整體策略，將網際網路和網頁整合到電腦的應用程式中。
- 微軟的 .NET(.NET platform) 平台實作了這個策略，使程式設計師開發出的程式，能夠在分散於網際網路上的電腦中執行。
- 微軟的三個主要程式語言分別是 Visual Basic (以原來的BASIC為基礎)、Visual C++ (以C++為基礎) 以及 Visual C# (以C++和Java為基礎的新語言，專為.NET平台開發)。
- Visual C++ 也可以用來編譯和執行C的程式。

1.13 關鍵的軟體工程趨勢：物件技術

- 1980年代初期在AT&T，Bjarne Stroustrup根據兩種語言，C語言以及Simula 67(一種在1967年公開，由挪威計算中心所設計的模擬程式語言)，發展出了C++程式語言。
- C++吸收了C語言的功能，並且加入了Simula語言中有關物件產生及操作的能力。
- 物件技術是一種包裝的方法，它能夠幫助我們產生一些有意義的軟體元件。
- 它們是日期物件、時間物件、薪資物件、收據物件、聲音物件、影像物件、檔案物件、記錄物件等等。

1.13 關鍵的軟體工程趨勢：物件技術

- 事實上，任何名詞都可以表示成一個物件。
- 我們生活在一個到處都是物件的世界裡。
- 汽車、飛機、人、動物、建築物、交通號誌、電梯等等。
- 在物件導向式程式語言出現之前，所有的程式語言(如FORTRAN，Passcal，Basic和C) 都將注意力集中在動作 (動詞) 上，而非事物或物件 (名詞) 。
- 雖然程式設計師生活在物件的世界，卻使用動詞來寫程式。
- 這種撰寫程式的方式不太順暢。

1.13 關鍵的軟體工程趨勢：物件技術

- 現在，由於物件導向式程式語言的廣泛流行 (如C++、Java和C#)，程式設計師可以用一種與現實生活相同的思考模式來撰寫程式。
- 這是一種較為自然的流程，因此可以獲得比程序式程式設計更高的生產力。
- 程序式程式設計的一項關鍵性問題在於程式設計師所開發的程式元件，不容易有效地對應到真實世界的相關事物。因此這些元件不常被拿來重複使用。

1.13 關鍵的軟體工程趨勢：物件技術

- 程式設計師常常會在每一個新的計劃開始時，要重新撰寫一些類似的程式，因而浪費了時間與金錢，「重新從基礎開始設計」。而在物件技術裡，軟體元件 (稱為類別) 如果設計正確的話，在他們產生之後，即可被接下來的所有軟體計劃重複使用。
- 我們還能使用一些可重複使用的函式庫，來大幅地減輕開發軟體系統的工作量。

軟體工程的觀點 1.1

網際網路上可以找到大量的類別函式庫，包含許多可再利用的軟體元件。其中許多是可以自由使用的。

1.13 關鍵的軟體工程趨勢：物件技術

- 有些機構的報告指出，事實上軟體再利用並不是他們從物件導向式程式設計中所獲取的最主要的好處。物件導向式程式設計能夠生產出更易於了解，易於維護、修改及偵錯的軟體。
- 這是十分有意義的，因為約有80%的軟體成本並非用於原來的開發計劃，而是用來持續不斷地改版及維護。
- 由於物件導向式程式設計具有這麼多關鍵性的優點，因此毫無疑問地，它將會是未來數十年最主要的程式設計方法之一。

1.14 典型的 C 開發環境

- C系統通常由幾個部分組成：程式開發環境、程式語言及C標準函式庫
- C程式在真正執行前，通常必須經過6個階段 (圖1.1)。
- 它們是：編輯 (edit)、前置處理 (preprocess)、編譯 (compile)，連結 (link) 、載入 (load) 和執行 (execute)。
- 雖然<<C程式設計藝術>>是一本通用的C教科書(與任何作業系統沒有關聯性)，不過在本節裡我們將會以傳統的以LINUX為基礎的C系統來進行介紹。

1.14 典型的 C 開發環境

- [注意：本書中的程式只需小幅修改或無需修改便能夠在現今大多數的C系統上執行，包括了Microsoft的Windows系統。]如果你目前使用的不是LINUX系統，那麼請參考系統的使用手冊，或請教老師，讓你能在你的環境中完成你的工作。
- 參考我們的C資源中心網站，網址是 www.deitel.com/C，可找到「getting started」教學中，常用的C編譯器以及開發環境。
- 第一個階段是編輯檔案。
- 這可以利用文書編輯器(editor program)來完成。

1.14 典型的 C 開發環境

- LINUX系統中最常用的兩種編輯器就是vi和emacs。
- 而C/C++整合開發環境的套裝軟體如Eclipse和Microsoft Visual Studio，都附有內建的編輯器，而且都將編輯器整合在程式發展的環境中。
- 程式設計師用編輯器鍵入或修改 (如果需要的話) C 程式，然後將程式存放在輔助記憶體如磁碟內。
- C程式的副檔名應為.c。

1.14 典型的 C 開發環境

- 在第二階段，程式設計者會下指令以編譯 (compile) 程式。
- 編譯器會將C程式碼轉譯成機器碼 (也稱為目的碼，object code)。
- 在C系統中，前置處理 (preprocessor) 程式會在編譯器轉譯階段之前執行。
- C前置處理器 (C preprocessor) 會按照一種叫做「前置處理指令」(preprocessor directive) 的特殊指令進行動作，該指令表示編譯前要對程式執行某些操作。

1.14 典型的 C 開發環境

- 這些操作通常會把其它要編譯的文字檔含括進來，並進行各種文字取代動作。
- 我們會在前幾章討論最常用的前置處理指令，第13章會詳談前置處理程式的功能。
- 在第三階段，編譯器將C程式碼轉譯為機器碼。

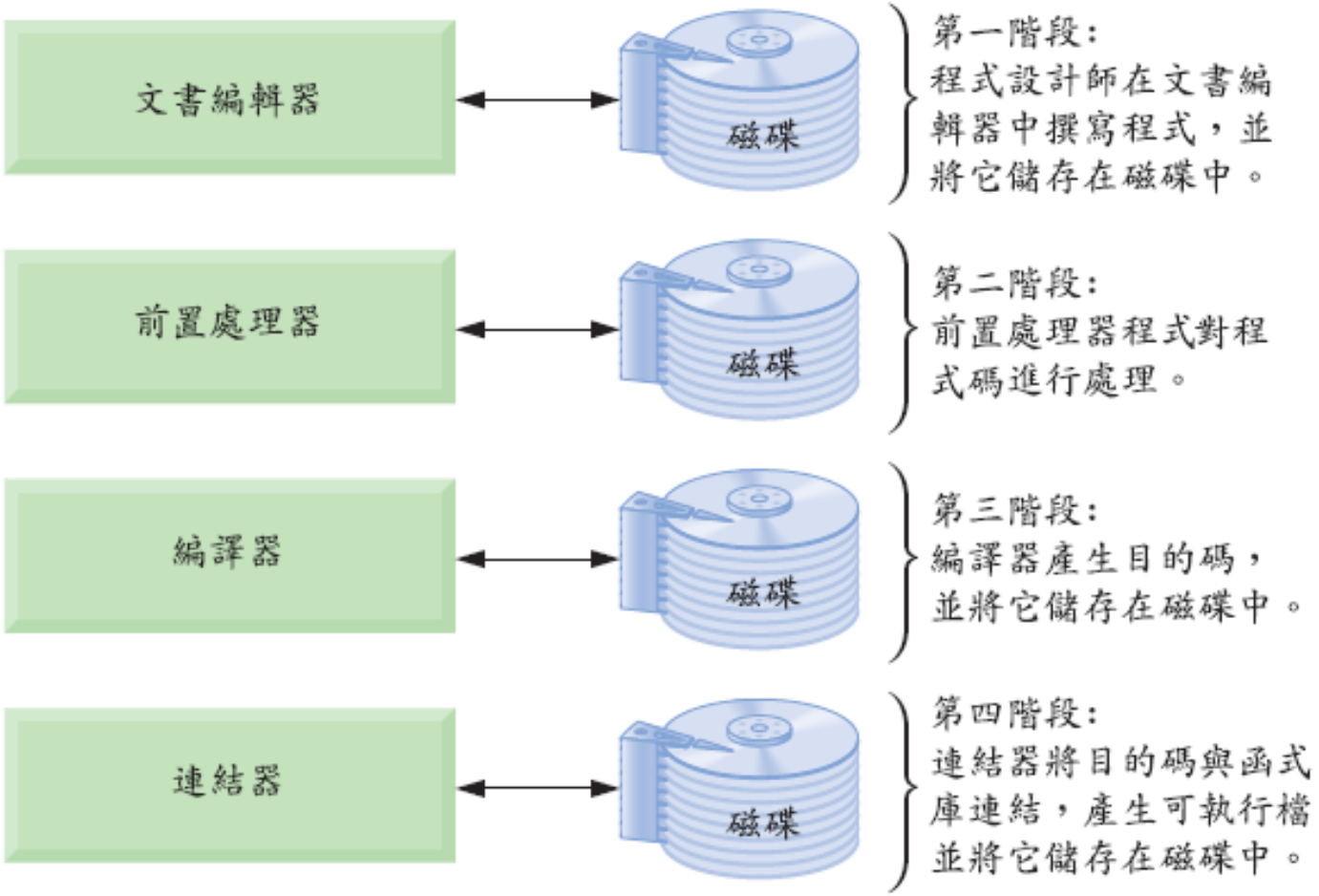

圖 1.1　典型的 C 開發環境

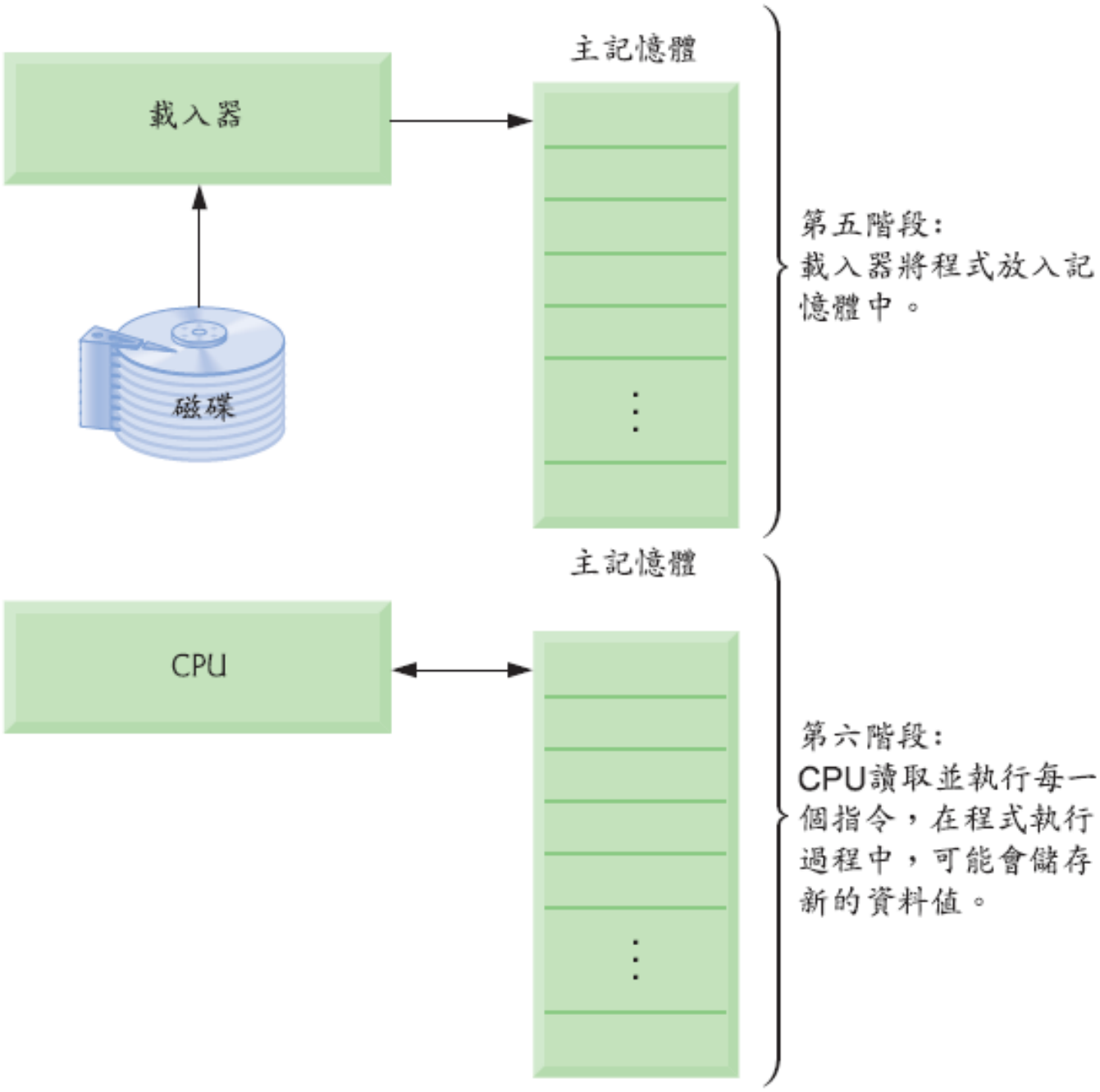

圖 1.1　典型的 C 開發環境

1.14 典型的 C 開發環境

- 第四階段叫做連結 (linking)。
- C程式中，常有些參照 (reference) 會指到在別處定義的函式與資料，如標準函式庫或特定專案成員私有函式庫中的函式與資料
- C編譯器所產生的目的碼通常包含這些尚未加入的程式碼所形成的「洞」。
- 連結器 (linker) 會將目的碼與這些尚未加入的函式連接起來，以產生可執行的影像檔 (executable image)，就沒有遺漏的部分了。
- 在一般的LINUX系統上，編譯及連結程式的命令是 cc（或 gcc）。

1.14 典型的 C 開發環境

- 舉例來說，如果要編譯及連結一個稱為welcome.c的程式，那麼我們必須在LINUX的提示字元後鍵入
  - cc welcome.c
- 然後鍵入Enter (或Return)。
- [注意：Linux 命令是有大小寫區分的，請確認你鍵入的命令是小寫，且檔案名稱的大小寫是正確的。]
- 若程式正確的編譯和連結，就會產生一個檔名為 a.out 的檔案。
- 這是我們的welcome.c程式的可執行影像檔。

1.14 典型的 C 開發環境

- 下一個階段叫做載入 (loading)。
- 程式執行前，必須先被放入記憶體中。
- 載入器 (loader) 負責這項工作，它能把可執行的影像檔從磁碟搬到記憶體中。
- 程式所用到的共享函式庫中其它元件，也須一併載入。
- 最後，電腦會在CPU的控制下，以每次執行一個指令的方式開始執行 (executes) 程式。

1.14 典型的 C 開發環境

- 在LINUX系統中，載入和執行只需在LINUX提示字元後鍵入./a.out，然後按下return鍵即可。
- 程式在第一次測試時不一定會成功。
- 前面幾個階段都可能因各種錯誤而失敗，這些錯誤本書都會討論。
- 例如：一個執行中程式可能會除以零 (如同算術，在電腦中這是一個非法的運算) 。
- 這可能會使電腦顯示一段錯誤訊息。

1.14 典型的 C 開發環境

- 若發生這樣的情形，就要回到編輯階段做些必要修正，再繼續後面的幾個階段，看看改對了沒。
- C中大部份程式都會輸入和/或輸出資料。
- 大部分的C函式都從stdin (標準輸入串流，standard input stream) 輸入資料。stdin通常是鍵盤，不過他們也可以連結到其他的裝置。
- 大部分的C函式都從stdout (標準輸出串流，standard output stream) 輸出資料。stdout通常是螢幕，不過他們也可以連結到其他的裝置。
- 當我們說「程式印出結果」時，通常是指在螢幕上顯示結果。

1.14 典型的 C 開發環境

- 資料也可輸出到其它裝置，如磁碟和印表機。
- 電腦也有標準錯誤串流 (standard error stream)，稱為stderr。
- stderr 串流 (一般會連到螢幕) 用來顯示錯誤訊息。
- 使用者通常會將輸出資料stdout指定到螢幕以外的裝置，而stderr仍指定到螢幕，因此當錯誤發生時，使用者會馬上知道。

常見的程式設計錯誤 1.1

當程式執行發生除零錯誤時，這種錯誤稱爲執行時期錯誤 (run-time error 或 execution-time error)。除以零通常是致命錯誤，也就是使程式立即終止無法繼續執行的錯誤。非致命的執行時期錯誤 (Nonfatal runtime errors) 可以允許程式繼續執行完畢，但通常會產生錯誤的結果。

1.15 硬體發展的趨勢

- 每一至兩年，電腦的價格沒有增加的同時，速度卻已倍數的成長。
- 這種現象稱為「摩爾定律」(Moore's Law)，這是以首次發現並解釋這個趨勢的人，Gordon Moore所命名的。Gordon Moore是Intel的共同創立者之一，Intel則是生產現今大多數個人電腦處理器的公司。

1.15 硬體發展的趨勢

- 摩爾定律和類似的趨勢在某些事物上特別地明顯：像是電腦的記憶體數量 (用來提供程式所需)、輔助儲存裝置的容量 (如磁碟機，可用來長時間儲存資料)、以及處理器速度 (與電腦執行程式速度有關)。
- 同樣的成長現象也發生在通訊領域，硬體的價格直線下降，特別是最近幾年通訊頻寬的大量需求與競爭之下。

1.16 關於 C 語言以及閱讀本書要注意的一般事項

- 本書適合程式設計新手，因此我們強調程式的清晰度 (program clarity)。
- 以下是第一個「良好程式設計習慣」。

良好的程式設計習慣 1.1

以簡單、直覺的方式撰寫 C 程式。有時稱之爲 KIS 原則 (保持單純化，keep it simple)。千萬別寫怪語法「扭曲」了程式。

1.16 關於 C 語言以及閱讀本書要注意的一般事項

- 你應該已經聽過，C是可攜性的語言，以C撰寫的程式可在許多不同的電腦上執行。
- 但可攜性是個難以捉摸的目標。

可攜性的小技巧 1.2

雖然寫出具可攜性的程式是有可能的，但不同的 C 編譯器，和不同的電腦之間仍有許多問題，因此可攜性難以達成。光用 C 撰寫程式不一定具有可攜性。你通常必須自己處理複雜的電腦相異處。

1.16 關於 C 語言以及閱讀本書要注意的一般事項

- C實在是一個內容十分豐富的語言，有些精妙之處是本書所沒有介紹的。
- 如果讀者需要其他C的技術細節，建議您查閱標準C文件，或查閱Kernighan和Ritchie所著的參考手冊(*The C Programming Language, Second Edition*)。

軟體工程的觀點 1.2

請閱讀您目前使用的 C 版本手冊。經常參考這些手冊，確定自己瞭解 C 豐富的功能，並正確使用它們。

軟體工程的觀點 1.3

您的電腦和編譯器都是良師。假如你不確定 C 的某一個功能如何使用，寫一個具有那個功能的範例程式，編譯並執行這個程式，看看結果如何。