前言

C 程式語言是在 20 世紀 70 年代早期，由 Dennis Ritchie 於 AT＆T 貝爾實驗室所開創的。直到 20 世紀 70 年末，此程式語言才開始廣泛的普及。這是因為在那個時候之前，C 編譯器不能在貝爾實驗室以外做商業使用。最初，C 的人氣增長也受到了 Unix 作業系統普及的影響。這個作業系統，也是在貝爾實驗室開發的，以 C 作為其 "標準" 程式語言。事實上，超過 90％的作業系統是用 C 語言編寫的！

IBM PC 的成功很快地使 MS-DOS 成為 C 語言最受歡迎的環境。隨著 C 在不同作業系統中越來越流行，越來越多的供應商響應了從眾效應，開始銷售自己的 C 編譯器。在大多數情況下，他們的 C 語言版本是基於第一本 C 程式設計的書 — The C Programming Language — 由 Brian Kernighan 和 Dennis Ritchie 所撰寫的，請參閱附錄。

在 1980 年代初期，人們認為需要對 C 語言的定義進行標準化。美國國家標準學會（ANSI）是處理這種事情的組織，所以在 1983 年成立了一個 ANSI C 委員會（稱為 X3J11）來標準化 C。1989 年，委員會批准此項任務，1990 年，公佈了第一個官方 ANSI 的 C 語言標準定義。

因為 C 使用於世界各地，國際標準組織（ISO）很快就參與了。他們採用了一個標準，被稱為 ISO/IEC 9899:1990。從那時起，對 C 語言進行了額外的修改。最近的標準於 2011 年被採用，此稱為 ANSI C11 或 ISO/IEC 9899:2011。它是本書所採用的版本語言。

C 是一種 "高階語言"，但它提供了使用者能夠觸及硬體與處理電腦底層的的能力。儘管 C 是通用結構化程式語言，但它最初的設計是系統程式的應用，因此提供使用者大量的強大與彈性的功能。

這本書的目的是教您如何使用 C 語言編寫程式。它假設您沒有接觸過於程式設計，並為新手和有經驗的程式設計師而設計的。如果您有程式設計經驗，您會發現 C 有一個獨特的方式來處理事情，有異於您所使用的其它程式語言。

C 語言的每個特性都會在本書中介紹。當呈現每個新特性時，通常會提供小小的完整程式範例來說明。這實現了寫這本書時所使用的哲學概念：以範例教學（To teach by example）。 正如一張圖勝過千言萬語，所以程式範例是最佳的學習方法。如果您使用支援 C 程式語言的電腦，強烈建議您下載並執行本書中所介紹的每一個程式，並將您在系統上所執行的結果與內文中顯示的結果進行比較。利用這樣的做法，您不僅可以學習語言及其語法，還可熟悉編寫、編譯和執行 C 程式的過程。

您會發現本書中程式所強調的可讀性。這是因為我堅信程式應該要寫得很容易讓撰寫者或其他人閱讀。根據經驗和常識，發現這樣的程式可以更容易撰寫、除錯和修改。此外，開發可讀性的程式是真正遵守結構化原則的自然結果。

由於這本書是以要做為教本來編寫的，所以每個章節涵蓋的內容是基於之前提出的教材。因此，有順序的閱讀，將會獲得最大的利益，所以請您打消以 "跳章節" 的方式來閱讀。您還應該練習每章末所列出的習題，之後再繼續下一個章節。

第 1 章 "一些基本概念" 涵蓋了關於高階程式語言和編譯程式的一些基本術語，以確保您理解本書的其餘部分所使用的術語。從第 2 章 "編譯與執行第一個程式" 開始，您將慢慢了解 C 語言。到第 15 章 "C 語言的輸入與輸出"，所有 C 語言的基本特性大概都已介紹過。第 15 章更深入地介紹了 C 語言的 I/O 運作。第 16 章 "其它論題及進階功能" 涵蓋更進階或深奧的語言特性。

第 17 章 "除錯程式" 展示如何使用 C 前置處理器幫助除錯程式。還會為您介紹交談式的除錯。我們選擇目前流行的除錯器 gdb 來說明這種除錯技術。

在過去十年中，程式設計的風潮已經朝向物件導向程式設計（Object-Oriented Programming, OOP）的概念。C 不是 OOP 語言；然而，基於 C 的其它幾種程式語言是 OOP 語言。第 18 章 "物件導向程式設計" 簡略地介紹 OOP 及其一些術語。它還簡要概述了基於 C 的三種 OOP 語言，其分別為 C++、C # 和 Objective-C。

附錄 A "C 語言摘要" 提供了 C 語言的完整摘要，以供使用者參考。

附錄 B："C 標準函式庫" 提供了許多標準函式庫程式的摘要，您將可在支援 C 的所有系統上找到它們。

附錄 C："使用 gcc 編譯程式" 摘錄使用 GNU 的 C 編譯器 — gcc，來編譯程式時的許多常用選項。

在附錄 D："常見的程式設計錯誤"，您將找到一系列常見的程式錯誤。

最後，附錄 E："其它有用資源" 提供了一系列的資源，您可以查閱有關 C 語言更多的資訊和進一步的研究。

本書沒有預設實作 C 語言時的特定電腦系統或作業系統。內文只簡要地提到，如何使用受歡迎的 GNU C 編譯器 — gcc，來編譯和執行程式。

1

一些基本概念

本章將先介紹在學習如何使用 C 語言程式之前，必須要了解的一些基本術語。同時也提供在高階語言的程式設計的一般概述，以及編譯程式的過程。

程式設計

電腦是一部非常笨拙的機器，因為它們只做被告知要做的事項。大多數電腦系統在原始的層級上執行它們的運作。例如，大多數電腦知道如何將一個數字加 1，或如何測試一個數字是否等於 0。這些基本運算的複雜度通常沒有那麼大。電腦系統的基本運作形成所謂的電腦指令集（instruction set）。

為了使用電腦來解決問題，您必須以電腦所能了解的指令，來表達問題的解決方案。電腦程式（program）只是解決特定問題所需的指令的集合。用於解決問題的方法被稱為演算法（algorithm）。例如，如果你想開發一個程式來測試一個數字是奇數還是偶數，解決問題的指令集就成為程式。用於測試數字是偶數還是奇數的方法是演算法。通常，為了開發一個程式來解決一個特定的問題，你首先用演算法表達問題的解決方案，之後開發一個實現該演算法的程式。因此，用於求解偶數/奇數問題的演算法可以表示如下：首先，將數字除以 2。如果除法的餘數為 0，則該數字為偶數；否則，其為奇數。利用手中的演算法，便可以隨後撰寫在特定電腦系統上實現此演算法所需的指令。這些指令將以特定電腦語言的敘述中表示，例如 Java、C++、Objective-C 或 C。

高階語言

一開始開發電腦時，它們可以被編寫的唯一方式是，根據直接對應於特定機器指令和電腦記憶體中的位置的二進制數字。下一個軟體技術的進步發生在組合語言

（assembly language）的開發中，這使得程式設計師能夠更高階地與機器一起工作。作為取代以指定二進制數字序列執行特定任務，組合語言允許程式設計師使用符號名稱，來執行各種操作並引用特定的記憶體位置。被稱為組譯器（assembler）的特殊程式，將組合語言程式從符號格式轉換成電腦系統的特定機器指令。

由於在每個組合語言指令和特定的機器指令之間，仍然存在一對一的對應關係，所以組合語言被認為是低階語言。程式設計師必須學習特定電腦系統的指令集以用組合語言編寫程式，並且所得到的程式是沒有可攜性的（portable）；也就是說，程式不會在不被重寫的情況下，在不同的處理器類型上運作。這是因為不同的處理器類型具有不同的指令集，並且因為組合語言程式是根據這些指令集來編寫的，所以它們是依賴於機器的。

然而，所謂的高階語言，FORTRAN（FORmula TRANslation）語言是第一個。在 FORTRAN 中開發程式的程式設計師，不再需要關注特定電腦的體系結構，在 FORTRAN 中執行的操作是更複雜、更高階的，遠遠相差特定機器的指令集。一個 FORTRAN 指令或敘述可執行許多不同的機器指令，這與在組合語言敘述和機器指令之間發生的一一對應不同。

高階語言的語法的標準化，意味著程式可以用與機器無關的語言編寫。也就是說，只需要很少的更改或不必更改，程式可以在支援該語言的任何機器上運行。

為了支援高階語言，必須開發一個特殊的電腦程式，將在高階語言中開發的程式的敘述，翻譯成電腦可以理解的形式 — 換句話說，轉換成電腦指令。這樣的程式被稱為編譯器（compiler）。

作業系統

在繼續講編譯器之前，需要了解一個稱為作業系統（operating system）的電腦程式所扮演的角色。

作業系統是控制電腦系統整個運作的程式。在電腦系統上執行的所有輸入和輸出（即 I/O）運作都經由作業系統進行引導。作業系統還必須管理電腦系統的資源，並且必須處理程式的執行。

當今最流行的作業系統之一是由 Bell 實驗室開發的 Unix 作業系統。Unix 是一個相當獨特的作業系統，因為它可以在許多不同類型的電腦系統上並且在不同的 "風格" 下找到，例如 Linux 或 Mac OS X。歷史上，作業系統通常僅與一種類型的電腦系統

相關聯。但是因為 Unix 主要是用 C 語言編寫的，並且對電腦的架構做了很少的假設，所以它已經可以成功地移植到許多不同的電腦系統中，並且花費相對少的代價。

Microsoft Windows 是流行作業系統的另一個範例。該系統主要在 Intel（或 Intel 相容）處理器上運行。

最近較佳的是開發在行動裝置（如智慧型手機和平板電腦）上運行的作業系統。蘋果的 iOS 和谷歌的 Android 作業系統是兩個最受歡迎的例子。

編譯器

編譯器是一個軟體程式，原則上與您在本書中看到的沒有什麼不同，雖然它肯定來得復雜得多。編譯器分析以特定電腦語言開發的程式，並將其轉換為適合在特定電腦系統上執行的形式。

圖 1.1 顯示了進入、編譯和執行以 C 程式語言開發的電腦程式，所涉及的步驟和從命令列輸入的典型 Unix 指令。

要編譯的程式首先被輸入到電腦系統上的檔案中。電腦安裝具有用於命名檔案的各種協定，但一般來說，名稱的選擇取決於您。C 程式通常可以指定任何名稱，前提是最後兩個字元是 ".c"（這不是一個要求，因為它是一個協定）。因此，名稱 prog1.c 是系統上 C 程式的有效檔案名稱。

文字編輯器通常用於將 C 程式輸入到檔案中。例如，vim 是在 Unix 系統上使用的流行文字編輯器。輸入到檔案中的程式稱為原始程式（source program），因為它表示用 C 語言表示的程式的原始形式。將原始程式輸入到檔案中之後，隨後就可以編譯它了。

編譯過程通過在系統上輸入特殊指令來啟動。輸入此命令時，還必須指定包含原始程式的檔案的名稱。例如，在 Unix 下，啟動程式編譯的指令為 cc。如果您使用流行的 GNU C 編譯器，您使用的指令是 gcc。輸入以下指令：

```
gcc prog1.c
```

將啟動與 prog1.c 中包含的原始程式的編譯過程。

在編譯過程的第一步，編譯器檢查原始程式中包含的每個程式敘述，並檢查它以確保它符合語言[1]的語法和語義。如果編譯器在此階段發現任何錯誤，它將報告給使用

[1] 從技術上講，C編譯器通常會做一個程式的預備，用以尋找特殊的狀態。這個預處理階段在第12章 "前置處理器" 中將有詳細描述。

者，編譯過程就會在那裡結束。然後必須在原始程式中（使用編輯器）更正錯誤，並且必須再重新啟動編譯過程。在編譯階段的報告的典型錯誤，可能是因為是不平衡括號（語法錯誤）的運算式，或者因為使用了未被 "定義"（語義錯誤）的變數。

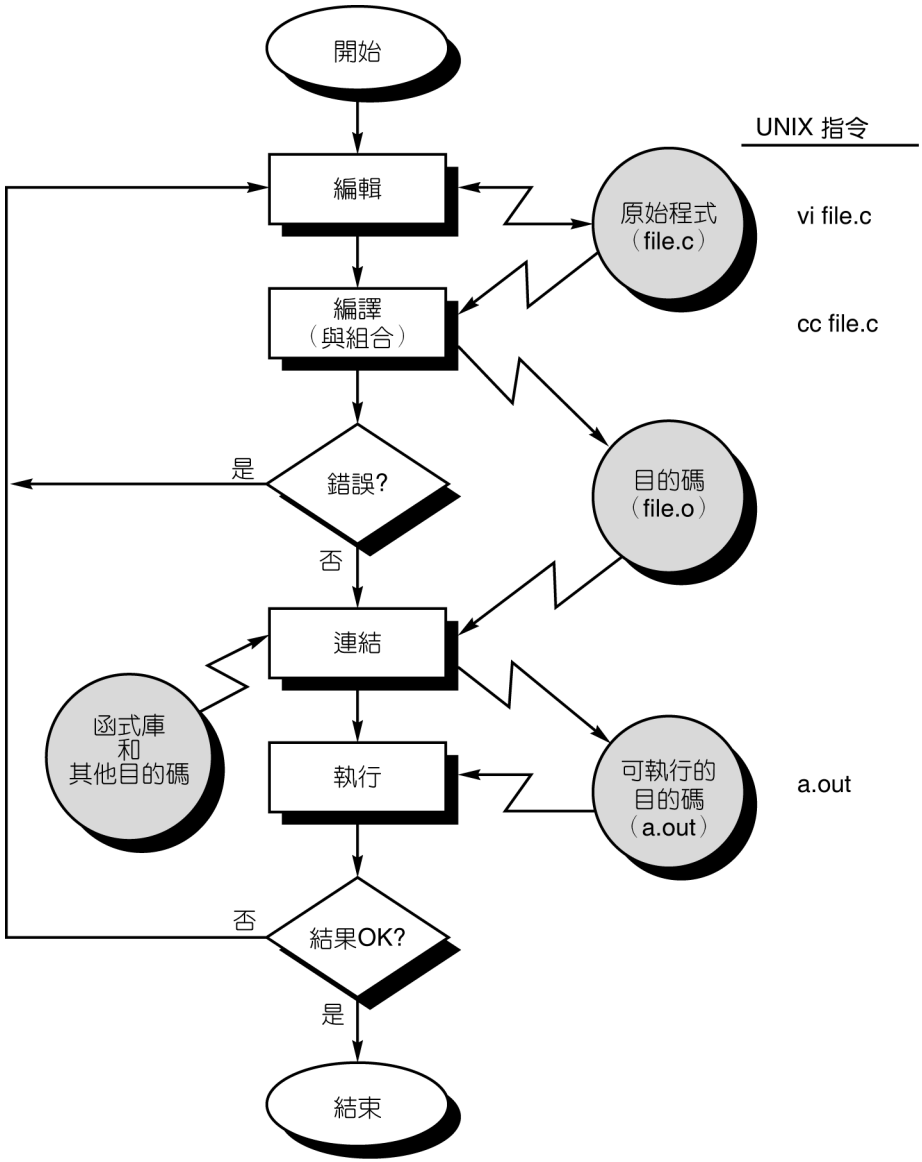

圖 1.1 從命令列進入、編譯和執行 C 程式的一般流程

當所有的語法錯誤和語義錯誤都從程式中加以更正時，編譯器隨後繼續採取程式的每個敘述，並將其翻譯成 "低階" 的形式。在大多數系統上，這意味著每個敘述，將由編譯器轉換為相對的敘述或組合語言的敘述以完成某項任務。

在程式被翻譯成相對的組合語言程式之後，編譯過程的下一步是，將組合語言敘述翻譯成實際的機器指令。這個步驟可能涉及或不涉及組譯器的單獨程式的執行。在大多數系統上，組譯器的自動執行作為編譯過程的一部分。

組譯器接受每個組合語言敘述，並將其轉換為稱為目的碼的二進制格式，然後將其寫入系統上的另一個檔案。此檔案通常與 Unix 下的原始檔具有相同的名稱，最後一個字母為 "o"（object）而不是 "c" 。在 Windows 下，通常以 "obj" 取代檔名中的 "c"。

在程式被翻譯成目的碼之後，它就可以被連結（linked）。每當 cc 或 gcc 指令在 Unix 下發出時，此過程再次自動執行。連結階段的目的是使程式成為在電腦上執行的最終形式。如果程式使用先前由編譯器處理的其它程式，則在該階段將程式連結在一起。在這個階段，還進行搜索所使用的系統函式庫（library）的程式，並與目的碼一起連結。

編譯和連結程式的整個過程通常被稱為構建（building）。

最終的連結檔案是可執行的目的碼（executable object）格式，儲存在系統上的另一個檔案中，準備運行或執行（executed）。在 Unix 下，預設情況下，此檔案稱為 a.out。在 Windows 下，可執行檔通常與原始檔具有相同的名稱，副檔名 c 替換為副檔名 exe。

要執行程式，您所要做的就是輸入可執行目的檔的名稱。所以，下面指令：

```
a.out
```

將名為 a.out 的程式載入（loading）到電腦的記憶體中，並啟動其執行。

當執行程式時，依次順序執行程式的每個敘述。如果程式要求來自使用者的資料，稱為輸入（input），此時程式將暫停其執行以便輸入。又或者，程式等待如滑鼠被點擊之類的事件（event）發生。程式的結果此稱為輸出（output）。將結果顯示在視窗或稱控制台（console）。又或者，輸出可能直接寫入系統上的檔案。

如果一切順利（並且可能不是第一次執行程式），程式執行其預期的功能。如果程式沒有產生所需的結果，則必須返回並重新分析程式的邏輯。這被稱為除錯階段（debugging phase），在此階段嘗試從程式中刪除所有已知的問題或錯誤（bugs）。為此，很可能需要對原始程式進行更改。在這種情況下，必須重複編譯、連結和執行程式的整個過程，直到獲得所期望的結果。

整合開發環境

前面概述了開發 C 程式時所涉及的各個步驟，展示了為每個步驟輸入的基本指令。編輯、編譯、執行和除錯程式的這個過程，通常由被稱為整合開發環境（integrated Development Environment，IDE）的單一的整合應用程式來管理。IDE 是一個基於 Windows 的程式，它允許您輕鬆管理大型軟體程式，在 Windows 中編輯檔案，以及編譯、連結、執行和除錯程式。

在 Mac OS X 上，Xcode 是由 Apple 支援的 IDE，被許多程式設計師使用。在 Windows 下，Microsoft Visual Studio 是流行 IDE 的一個很好的例子。所有 IDE 應用程式，大大簡化了程式開發中涉及的整個過程，因此值得您學習如何使用它。大多數 IDE 也支援除了 C 之外的幾種不同的程式語言的程式開發，例如 Objective-C、Java、C # 和 C++。

有關 IDE 的更多訊息，請參閱附錄 E "資源"。

直譯器

在離開編譯過程的討論之前，請注意，還有另一種方法用於分析和執行以高階語言開發的程式。使用這種方法，程式不會被編譯，而是被直譯（intepreted）。直譯器（intepreter）同時分析和執行程式的敘述。這種方法更容易除錯程式。另一方面，直譯語言通常比它們的編譯副本慢，因為程式敘述在它們的執行之前不會被轉換成低階形式。

BASIC 和 JavaScript 是兩種通常被直譯而不是被編譯的程式語言。其它例子包括 Unix 系統的 shell 和 Python。一些供應商還提供 C 程式語言的直譯器。

2

編譯與執行第一個程式

本章將為您介紹 C 語言，讓您可以看到 C 程式的概況。還有什麼更好的方式比看用 C 撰寫的實際程式，來獲得此語言的正面評價呢？

這一章很簡短，但是您會驚訝地發現，可以在一個簡短的章節中討論多少主題，包括：

- 編寫第一個程式
- 修改它以改變其輸出
- 理解 main() 函式
- 使用 printf() 函式輸出訊息
- 使用註解提高程式的可讀性

首先，選擇一個相當簡單的例子，在您視窗中顯示 "Programming is fun" 的程式。範例程式 2.1 呈現了一個完成此項工作的 C 程式。

範例程式 2.1 撰寫您的第一個程式

```
#include <stdio.h>

int main (void)
{
     printf ("Programming is fun.\n");

     return 0;
}
```

在 C 程式語言中，小寫字母和大寫字母是不同的。此外，在 C 中，從一行的哪個位置開始輸入是無關緊要 — 亦即您可以在該行的任何位置開始輸入敘述。這成為在開發更容易閱讀的程式之優勢。程式設計師經常使用 Tab 鍵作內縮的方便方法。

編譯程式

回到第一個 C 程式，首先需要將它輸入到一個檔案。任何文字編輯器都可以用來達成此目的。Unix 用戶經常使用如 vi 或 emacs 編輯器。

C 編譯器會識別以 "." 和 "c" 兩個字元結尾的檔名作為 C 程式。因此，假設將範例程式 2.1 輸入一個名為 prog1.c 的檔案。接下來，您需要編譯程式。

使用 GNU C 編譯器，只需要在終端機上發出 gcc 指令，後跟檔案名稱，就這麼簡單：

```
$ gcc prog1.c
$
```

如果您使用標準的 Unix C 編譯器，其指令是 cc 而不是 gcc。在這裡所輸入的文字是粗體的。如果從命令列編譯您的 C 程式，美元符號是命令提示符。實際命令提示符有可能是美元符號之外的一些字元。

如果在程式中有任何錯誤，編譯器將在輸入 gcc 指令後列出它們，通常會標識程式中有錯誤的行號。反之，出現另一個命令提示符，如前面的範例所示，表示在您的程式中沒有找到錯誤。

當編譯器編譯和連結程式時，它會建立程式的可執行版本。使用 GNU 或標準 C 編譯器，預設情況下，此程式為名 a.out。在 Windows 下，它通常名為 a.exe。

運行您的程式

現在可以在命令列上輸入其名稱來執行可執行檔：[1]

```
$ a.out
Programming is fun.
$
```

您還可以在編譯程式時，為可執行檔指定不同的名稱。這是利用 -o（即字母 O）選項完成的，後面跟著可執行檔的名稱。例如：

```
$ gcc prog1.c -o prog1
```

[1] 若得到這樣的錯誤：a.out: No such file or directory，這可能意味著當前目錄不在您的PATH。您可以將其添加到PATH或在命令提示符後面輸入以下指令：./a.out。

將編譯程式 prog1.c，把可執行檔放在檔案 prog1 中，然後指定其名稱來執行：

```
$ prog1
Programming is fun.
$
```

了解第一個程式

仔細看看您的第一個程式。程式的第一行:

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

應被載入於您寫的每個程式的開頭。它告訴編譯器關在稍後的程式中使用 printf()輸出函式的訊息。第 12 章 "前置處理器" 詳細討論這一行的作用。

下面程式行：

```
int main (void)
```

通知系統該程式的名稱是 main()，並回傳一個整數值，縮寫為 "int"。main() 是一個特殊的名稱，表示程式開始執行的位置。緊隨 main() 之後的一組括號，指定 main() 是函式（function）的名稱。在括號中的關鍵字 void 表示函式 main() 不使用參數（也就是說，它沒有參數）。這些概念在第 7 章 "函式" 有詳細的解釋。

> **注意**
>
> 如果您使用 IDE，可能會發現它為您生成一個模板 main()。在這種情況下，可能發現 main() 的第一行看起來像是這樣：
>
> `int main(int argc, char * argv [])`
>
> 這不會影響程式的運作，所以請忽略現在的差異。

現在您已經確認將 main() 加到系統，準備指定這個程式要執行什麼。這是利用將程式的所有程式敘述，封裝在一對大括號內來完成的。包含在大括號之間的所有程式敘述，都被系統作為 main() 程式的一部分。在範例程式 2.1 中，只有兩個這樣的敘述。第一個敘述指定要呼叫名為 printf() 的函式。傳遞給 printf() 函式的參數是字串：

```
"Programming is fun.\n"
```

printf() 函式是 C 函式庫中的一個函式，它簡單地在螢幕上印出或顯示其參數（或多個參數，稍後將會看到）。字串中的最後兩個字元，即反斜線（\）和字母 n，統稱為換行（newline）字元。換行字元告訴系統其字面上的意思 — 換句話說，到一

個新的行。換行字元之後，要印出的任何字元，將顯示於下一行。事實上，換行字元在概念上類似於打字機上的返回鍵。（記得那些嗎？）

C 中的所有程式敘述必須以分號（;）結束。這是緊跟在 printf() 呼叫的右括號後出現分號的原因。

main() 中的最後一個敘述：

```
return 0;
```

表示完成 main() 的執行，並回傳狀態值 0 到系統。您可以在這裡使用任何整數。零是按照慣例使用，表示程式成功完成，即沒有遇到任何錯誤。不同的數字可用於指示發生的不同類型的錯誤條件（例如未找到檔案）。這個退出狀態可以通過其它程式（例如 Unix shell）來測試，以查看程式是否成功執行。

現在您已經完成了第一個程式的分析，您可以修改它來顯示 "And programming in C is even more fun." 這可以簡單地添加另一個 printf() 函式呼叫，如範例程式 2.2 所示。記住每個 C 程式敘述必須以分號結束。

範例程式 2.2

```
#include <stdio.h>

int main (void)
{
     printf ("Programming is fun.\n");
     printf ("And programming in C is even more fun.\n");

     return 0;
}
```

如果鍵入範例程式 2.2，接著編譯並執行它，您可以期望在程式的輸出視窗（有時稱為 "控制台"）中輸出以下的結果：

範例程式 2.2 輸出結果

```
Programming is fun.
And programming in C is even more fun.
```

從下一個範例程式中可以看出，沒有必要對每一行輸出單獨呼叫 printf() 函式。研究範例程式 2.3 中呈現的程式，並在檢查輸出之前，嘗試預測其結果。（請不要作弊！）

範例程式 2.3 顯示數行輸出

```
#include <stdio.h>

int main (void)
{
     printf ("Testing...\n..1\n...2\n....3\n");

     return 0;
}
```

範例程式 2.3 輸出結果

```
Testing...
..1
...2
....3
```

顯示變數的值

printf() 函式是本書中最常用的函式。因為它提供了一種簡單又方便顯示程式結果的方法。不僅可以顯示簡單的訊息，而且還可以顯示變數（variable）的值及計算的結果。實際上，範例程式 2.4 使用 printf() 函式來顯示把兩個數字（即 50 和 25）相加的結果。

範例程式 2.4 顯示變數

```
#include <stdio.h>

int main (void)
{
     int sum;

     sum = 50 + 25;
     printf ("The sum of 50 and 25 is %i\n", sum);

     return 0;
}
```

範例程式 2.4 輸出結果

```
The sum of 50 and 25 is 75
```

在範例程式 2.4 中，第一個 C 程式敘述將變數 sum 宣告為整數型態。C 要求在程式中使用所有程式變數之前進行宣告。變數的宣告指定了 C 編譯器如何使用特定的變數。編譯器需要此訊息來生成正確的指令，以將值儲存和讀取到變數。宣告為型態

int 的變數只能用於儲存整數值；即不帶小數位的值。整數值的一些範例如 3、5、-20 和 0。具有小數位的數字，例如 3.14、2.45 和 27.0，被稱為浮點數。

整數變數 sum 用於儲存兩個整數 50 和 25 相加的結果。在該變數的宣告之後，有意地留下一個空白行，以便從程式敘述中可視覺化分離程式的變數宣告；這是嚴格的風格問題。有時，在程式中添加空白行可以幫助使程式提高其可讀性。

下面程式敘述：

```
sum = 50 + 25;
```

就像大多數其它程式語言一樣：將數字 50 加到（如加號所示）數字 25，並將結果儲存於變數 sum 中（由等號 — 指定運算子 — 表示）。

範例程式 2.4 中的 printf() 函式呼叫，有兩個參數括在括號中。這些參數以逗號分隔。printf() 函式的第一個參數永遠是要顯示的字串。但是，隨著字串的顯示，通常也可能希望顯示某些程式變數的值。在這種情況下，您希望變數 sum 的值顯示在下列字串後：

```
The sum of 50 and 25 is
```

第一個參數內的百分比字元，是由 printf() 函式所識別的特殊字元。緊跟在百分號後面的字元，指定要在該處顯示值的型態。在前面的範例程式中，字母 i 被 printf() 函式認定為要顯示整數值。[2]

每當 printf() 函式在字串中找到%i 字元時，它會自動向 printf() 函式顯示下一個參數的值。由於 sum 是 printf() 的下一個參數，所以它的值在顯示字串 "The sum of 50 and 25 is " 之後自動顯示。

現在請試著預測範例程式 2.5 的輸出。

[2] 請注意，printf 還允許您指定%d格式字元來顯示整數。本書在剩下的章節中會一直使用%i。

範例程式 2.5 顯示數個值

```
#include <stdio.h>

int main (void)
{
     int  value1, value2, sum;

     value1 = 50;
     value2 = 25;
     sum = value1 + value2;
     printf ("The sum of %i and %i is %i\n", value1, value2, sum);

     return 0;
}
```

範例程式 2.5 輸出結果

```
The sum of 50 and 25 is 75
```

第一個程式敘述宣告了三個變數，名為 value1、value2 和 sum，全部為 int 型態。該宣告也可以使用三個單獨的宣告，如下所示：

```
int value1;
int value2;
int sum;
```

在宣告三個變數後，程式將值 50 指定給變數 value1，接著將 25 指定給 value2。然後計算這兩個變數的和，並將結果指定給變數 sum。

對 printf() 函式的呼叫中包含四個參數。第一個參數統稱為格式字串，它向系統描述其它參數的顯示方式。value1 的值將緊跟在 "The sum of " 字串的顯示之後顯示。類似地，value2 和 sum 的值將在適當之處印出，即格式字元串中後兩次出現 %i 字元的地方。

註解

本章最後的程式（範例程式 2.6）介紹了註解（comment）的概念。在程式中使用註解敘述來說明程式，以增加其可讀性。正如您將從下面的範例程式看到的，註解用來告訴閱讀程式的人 — 程式設計師或者負責維護程式的其他人 — 程式設計師在撰寫一個特定的程式，或者特定的一系列敘述的時要注意什麼。

範例程式 2.6 在程式中使用註解

```
/* This program adds two integer values
   and displays the results             */

#include <stdio.h>

int main (void)
{
    // Declare variables
    int  value1, value2, sum;

    // Assign values and calculate their sum
    value1 = 50;
    value2 = 25;
    sum = value1 + value2;

    // Display the result
    printf ("The sum of %i and %i is %i\n", value1, value2, sum);

    return 0;
}
```

範例程式 2.6 輸出結果

```
The sum of 50 and 25 is 75
```

有兩種方法可以將註解插入到 C 程式中。註解可以由兩個字元 / 和 * 初始。這標記著註解的開始。這類型的註解必須有終止之處。要結束註解，使用字元 * 和 /，不帶任何嵌入空格。開始 /* 和結束 */ 之間的所有字元，都被視為註解敘述的一部分，並被 C 編譯器忽略。當註解跨越程式中的幾行時，通常使用此註解形式。在程式中加入註解的第二種方法，是使用兩個連續的斜線字元 //。任何跟隨在這些斜線後面的字元，直到行尾，都將被編譯器忽略。

在範例程式 2.6 中，使用了四個獨立的註解說明。該程式的其它部分與範例程式 2.5 相同。顯然，這是一個太勉強的範例，因為只有程式開始的第一個註解是有用的。（是的，可以在程式中插入這麼多的註解，但程式的可讀性實際上是退化的，而不是改善的！）

在程式中不應過分使用註解敘述。很多時候，一個程式設計師回到一個程式，他或許在六個月前才編寫過，但他會很沮喪，因為他已記不住某特定的函式或某特定敘述的目的。在程式中的特定點加入簡單的註解敘述，往後可以節省很多的時間，否則會浪費要重新思考函式或敘述串列邏輯的時間。

在編寫程式時，加入註解敘述到程式中是一種好習慣。有些更好的理由要您這樣做，首先，當特定的程式邏輯在您的心中仍然清晰時，比程式完成後重新思考其邏輯更容易記錄程式的功能與用意。其次，將註解插入到程式中就像遊戲的前期階段，在除錯階段，當程式邏輯錯誤被隔離和除錯時，您可以獲得註解的好處。註解不僅可以幫助您閱讀程式，而且還可以幫助找到邏輯錯誤的來源。最後，我還沒發現到真正喜歡撰寫程式文件的程式設計師。事實上，在您完成除錯程式後，您可能不會喜歡回到程式插入註解。在開發程式時插入註解，會使這個乏味的任務較容易完成。

在 C 語言中開發程式的這個入門章節到此結束。現在，您應該對在 C 語言中編寫程式所涉及的內容有了良好的感覺，您應該能夠自己開發一個小小的程式。在下一章中，將開始學習這種奇妙強大和靈活的程式語言一些更複雜的特性。但首先，試著實作下面的習題，以確保您理解本章提出的概念。

習題

1. 輸入並執行本章介紹的六個程式。將每個程式產生的輸出結果與本書中每一個程式的輸出結果進行比較。

2. 請撰寫一程式顯示下列文字：

    1. In C, lowercase letters are significant.
    2. main() is where program execution begins.
    3. Opening and closing braces enclose program statements in a routine.
    4. All program statements must be terminated by a semicolon.

3. 您期望從以下程式得到什麼輸出結果？

```
#include <stdio.h>

int main (void)
{
    printf ("Testing...");
    printf ("....1");
    printf ("...2");
    printf ("..3");
    printf ("\n");

    return 0;
}
```

4. 請撰寫一程式，以 87 減掉 15，並以適當的訊息顯示結果。

5. 請找出以下程式中的語法錯誤。接著輸入並執行更正後的程式，以確保您已正確更正所有的錯誤。

```
#include <stdio.h>

int main (Void)
(
        INT  sum;
        /* COMPUTE RESULT
        sum = 25 + 37 - 19
        /* DISPLAY RESULTS //
        printf ("The answer is %i\n" sum);
        return 0;
}
```

6. 您期望從以下程式得到什麼輸出結果？

```
#include <stdio.h>

int main (void)
{
      int answer, result;

      answer = 100;
      result = answer - 10;
      printf ("The result is %i\n", result + 5);

      return 0;
}
```