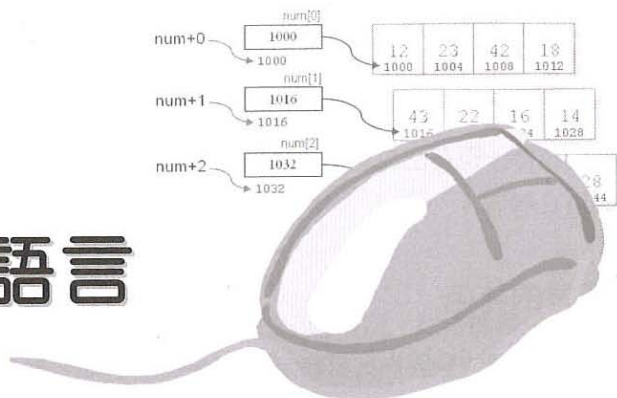


chapter

01

認識 C 語言



C 語言有如經過塑身後的一種電腦程式語言。它嬌小、勻稱，用途廣且效率高，因而廣受程式設計師及資訊界人士的喜愛。目前更有許多的應用程式，諸如影像處理、資料結構、人機介面控制及數值分析等等均以 C 語言來完成，由此可見 C 語言廣受歡迎的程度。本章將帶您初探 C 語言的世界，並撰寫第一個 C 語言程式。

◎ 本章學習目標

- 認識 C 語言的歷史
- 了解程式的規劃與實作
- 撰寫第一個 C 程式
- 學習程式碼的編譯與執行



1.1 C 語言概述

C 語言是由 Dennis Ritchie 博士，於 1972 年在貝爾實驗室（Bell laboratory）所發展出來的程式語言。它的前身為 B 語言，原先是用來撰寫 DEC PDP-11 電腦的系統程式。這個系統程式與後來為人所知悉的 Unix 作業系統有密不可分的關係。原本 C 語言只能在大電腦裡執行，現在已成功的移植到個人電腦裡，其中較為人所知悉的有 Turbo C、Microsoft C 等等。

1.1.1 C 語言的特色

每一種電腦語言的發展均有其特定的目的。例如 Basic 語言，其主要的目的是要讓電腦的初學者也可以很容易的撰寫程式，故其語法近似白話英文，淺顯易懂。此外，為了因應科學計算與商業用途的需要，Fortran、Cobol 與 Pascal 等語言也相繼產生。然而這些語言常因發展背景與語言本身的限制而無法兼顧實用與效能。C 語言的誕生恰可彌補這些缺憾，成為程式設計師的最佳工具。一般而言，C 具有下列的幾項特色：

• 高效率的編譯式語言

一般來說，當原始程式碼編輯完成後，必須轉換成機器所能理解的語言（即機器碼，machine code）後，才能正確的執行。所有的程式語言中，都附有這種轉換的程式，而轉換程式可概分為兩種，即直譯器（interpreter）與編譯器（compiler）。

直譯器

直譯器在程式執行時，會先檢查所要執行那一行敘述的語法，如果沒有錯誤，便直接執行該行程式，如果碰到錯誤就會立刻中斷，直到錯誤修正之後才能繼續執行。利用這種方式完成的程式語言，最著名的就屬 Basic。由於直譯器只要將程式逐行翻譯，因此佔用的記憶體較少，僅需要存取原始程式即可。然而，每一行程式在執行前才被翻譯，導致翻譯時間會延遲執行時間，因此執行的速度會變慢，效率也較低。

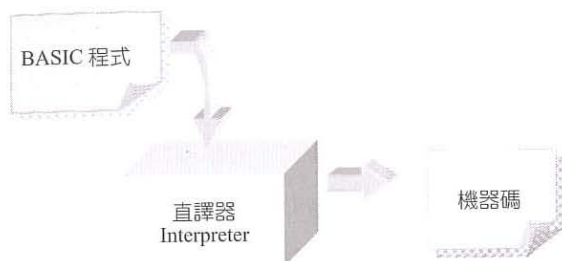


圖 1.1.1

直譯器會逐行檢查程式的語法，再直接執行該行程式，直到程式完畢

編譯器

編譯器在編譯程式時，會先檢查所有的程式碼是否合語法，然後編譯成可執行檔。當原始程式每修改一次，就必須重新編譯，才能保持其執行檔為最新的狀況。經過編譯後所產生的執行檔，在執行時不須要再翻譯，因此執行效率遠高於直譯程式。常見的編譯式程式語言有 C、Cobol、Pascal 等。而 C 的執行效率與使用的普遍性，更遠勝過其它的程式語言。

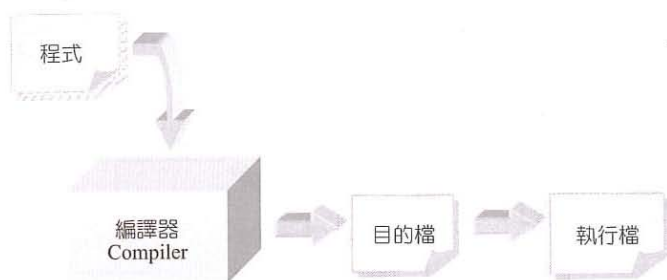


圖 1.1.2

編譯器先產生目的檔，再執行該程式

介於高階與低階之間的語言

程式語言依其特性可概分為二大類，即「低階語言」與「高階語言」。

低階語言（如組合語言）於電腦裡的執行效率相當的高，且對於硬體（如滑鼠、鍵盤等等）控制的程度相當的好，但對於人類而言它卻是生澀難懂，編寫、閱讀與維護均屬不易。

高階語言為敘述性的語言，它與人類所慣用的語法較為接近，故容易撰寫、除錯，但是相對的，它對硬體的控制能力也較差，執行效率也遠不及低階語言。常見的高階語言有 Basic、Fortran、Pascal 與 Cobol 等。

C 語言不但具有低階語言的優點（對硬體的控制能力佳），同時兼顧了高階語言的特色（易於除錯、撰寫），故有人稱之為「中階語言」。此外，C 語言還可以很容易的與組合語言連結，利用低階語言的特點來提高程式碼的執行效率。

• 靈活的程式控制流程

C 語言為一效率甚高，語法相當清晰的語言。它融合了電腦語言裡流程控制的特色，使得程式設計師可以很容易的設計出具有結構化及模組化的程式語言。

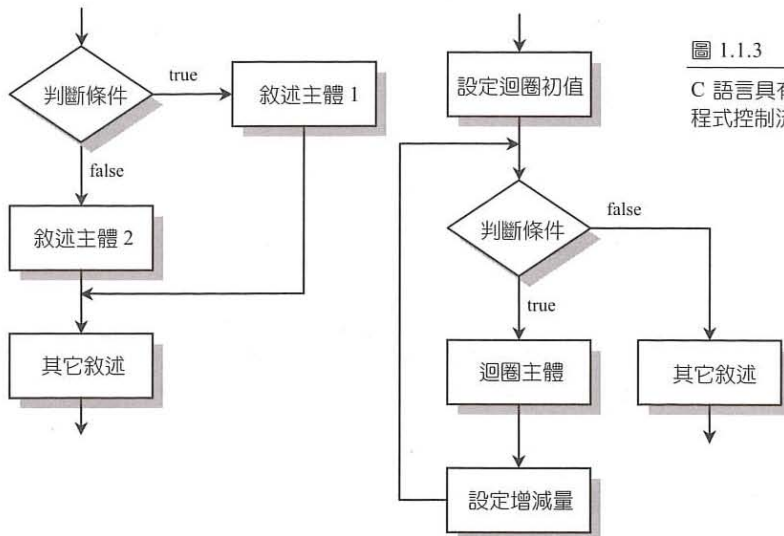


圖 1.1.3

C 語言具有靈活的
程式控制流程



由於 C 的高效率與靈活性，許多作業系統與驅動程式均由 C 寫成。此外，許多高階語言的編譯器（compiler）或解譯器（interpreter）亦是 C 的傑作。所以當您手握滑鼠，揮灑於 Windows 的天地時，想想這個精巧人機介面的背後，可是數十萬行 C 程式語言的結晶哩！

• 可攜性佳

程式語言的可攜性（portability）就像是硬體的相容性（compatibility）一樣。舉例來說，如果您買了一張音效卡，在各家廠牌的主機板上都能順利安裝，或者是只須要調整一下設定即可安裝，那麼我們就說這塊音效卡的相容性高。如果僅可在特定的主機板上使用，那麼這張音效卡的相容性就差。

同樣的，程式語言的可攜性佳意味著於某一系統所撰寫的語言，可以在少量修改或完全不修改的情況下即可在另一個作業系統裡執行。C 語言可以說是一個可攜性極佳的語言，當您想跨越平台來執行 C 語言時（如在 Unix 裡的 C 程式碼拿到 Windows 的環境裡執行）通常只要修改極少部分的程式碼，再重新編譯即可執行。此外，提供 C 編譯器的系統近 50 種，從早期的 Apple II 到超級電腦 Cray 均可找到 C 編譯器的芳蹤。

• 為程式設計師所設計的語言

C 語言可以說是專為程式設計師所設計的語言。它可以直接依記憶體的字址來存取變數，以提高程式執行的效率。此外 C 也提供了豐富的運算子（operator），使得 C 的語法更為簡潔有力。更方便的是，於大多數的 C 語言環境裡都提供了已撰寫好的程式庫（library），內含了許多 C 語言函數，以供程式設計師使用而無須重新撰寫程式碼。



• C 的另一面

通常一個優點的背後往往隱含著它的缺點，就如同一台功能超強的筆記型電腦，往往背負著體積過大與超重的罪名一樣。C 的語法嚴謹簡潔，相對的使用者就必須花更多的心思在學習 C 的語法上，尤其是指標（pointer）的應用，常常讓初學者摸不著邊際。但我們相信這只是個短暫的過程，一但熟悉了 C 的語法，您便可以享受到 C 所帶來的便利性與超速的快感。

1.1.2 C 語言與您

在 1980 年代初期，C 語言早已成為 Unix 作業系統的主要程式語言，現在 C 語言更強力介入大型電腦與個人電腦等領域。目前市面上有許許多多的程式遊戲、文書處理、繪圖及數學運算等軟體均是 C 的傑作。程式設計師之所以選擇 C 來做為軟體的開發工具，其中不僅是因為 C 較其它語言簡潔，好撰寫，效率高，更重要的是，C 易於修改，使得它可以在其它架構的電腦裡執行，因而較為程式設計師所樂於採用。

對於您個人而言，學習一個程式語言亦是一項重要的投資。學習 C 語言就好比您學英文一樣，走到世界各地都 "講" 得通，且有關於 C 的函數庫之取得也較其它電腦語言來得容易。以較現實的眼光來看，C 語言程式設計師的工作機會頗多，待遇也相對優渥，熟悉 C 語言確實較容易被業界所採用。

再者，C++ 或者是 Java 等語言均是以 C 為根基，再加上物件導向的功能，使得 C++ 與 Java 更加活躍於視窗與網路的程式設計（如 Visual C++ 與 C++ Builder 等）。即使是新一代的程式語言－Flash 的 ActionScript，其語法的撰寫也源自 C 語言。因此，C 語言的投資報酬率相當的高，值得您花時間去投資。當然，也別讓它在您的履歷表裡缺席。要學好 C 語言，就從現在開始！

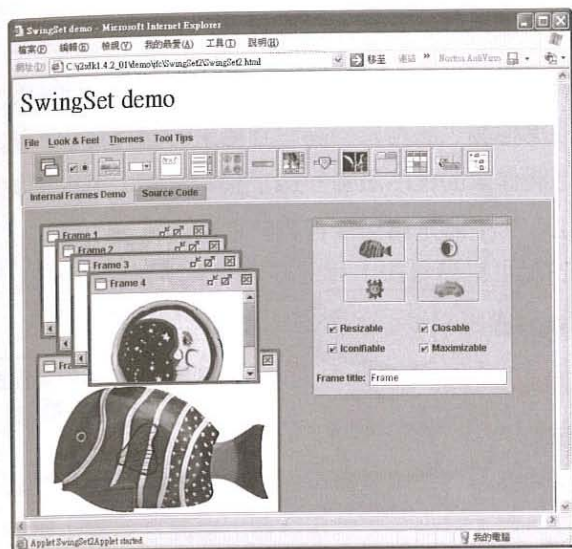


圖 1.1.4

這是由 Java 的 Swing 所設計出來的介面，它可直接在瀏覽器裡執行。Java 的源自 C++，其語法與 C 極為類似

1.2 程式的規劃與實作

在撰寫程式前的規劃是相當重要的。如果程式的內容很簡單，當然可以馬上把程式寫出來，但是當程式愈大或是愈複雜時，規劃的工作就很重要了！它可以讓程式設計有個明確的方向，有了事前的規劃流程，就可以根據這個流程來一步一步設計出理想的程式。

除了這個好處之外，習慣規劃程式後，可以發現程式會簡潔許多哦！這也意味著程式執行的速度將會更快、更有效率！程式撰寫需要經歷六個步驟，分別為規劃程式、撰寫程式碼與註解、編譯程式碼、執行程式、除錯與測試，以及程式碼的修飾與儲存等。接下來我們來看看這些步驟的內容：



(1) 規劃程式

首先，您必須確定撰寫這個程式的目的為何？使用者是誰？需求在哪兒？您可以於紙張上先繪製出簡單的流程圖，將程式的起始到結束的過程寫出，這麼做有幾個好處，一方面是讓您將作業的程序思考一遍；另一方面，可以根據這個流程圖來進行撰寫程式的工作。下圖是繪製流程圖時常用的符號：

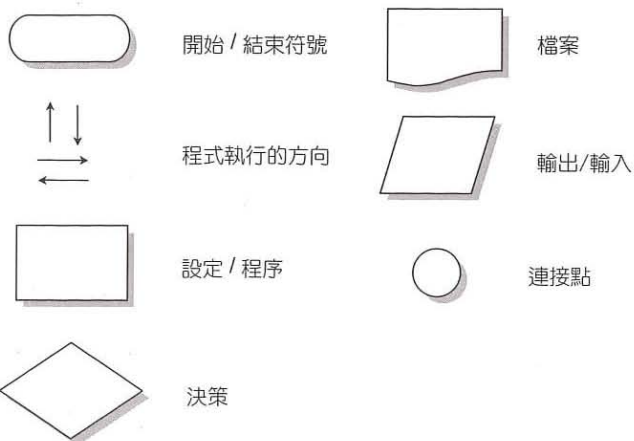


圖 1.2.1

常用的流程圖
符號介紹

我們以一個日常生活的例子「出門時如果下雨就帶傘，否則戴太陽眼鏡」，簡單的說明如何繪製流程圖：

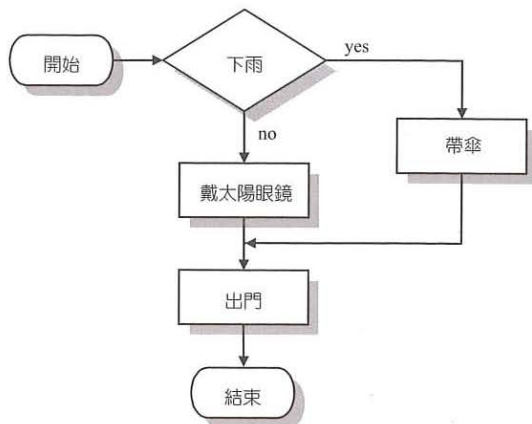


圖 1.2.2

流程圖的繪製示範

上面的流程圖裡，我們在決策方塊中填入「下雨」，如果「下雨」這件事成立，即執行「帶傘」的動作，否則執行「戴太陽眼鏡」的動作，因此在程序方塊裡分別填入「帶傘」及「戴太陽眼鏡」，不管執行哪一個動作，都必須「出門」，最後再結束整個事件。

您可以發現不管是程式設計，還是日常生活的程序，其實都可以用流程圖來表示，因此學習繪製流程圖，可是件相當有趣的事呢！

(2) 撰寫程式碼及註解

程式經過先前的規劃之後，便可以根據所繪製的流程圖來撰寫程式內容。您會發現，這種方式會比邊寫邊想下一步該怎麼做要快得多。如果事前沒有先規劃程式，往往會寫了又改，改了又寫，甚至一直都不滿意，而浪費了許多時間。

此外，建議您在撰寫程式時別忘了把註解加上。也許您會覺得，這些程式都是自己的作品，怎麼可能看不懂？但是如果很久沒有修改這個程式，或是別人必須維護它，若是於程式中加上了註解，可以增加這個程式的可讀性，相對的也會增加程式維護的容易度。舉手之勞常可節省日後大量程式維護的時間。

```

01  /* prog1_1, 第一個 C 程式碼 */
02  #include <stdio.h>
03  #include <stdlib.h>
04  int main(void)
05  {
06      printf("Hello C!\n");
07      printf("Hello World!\n");
08
09      system("pause");
10      return 0;
11  }
```

圖 1.2.3

程式加上註解，可增進程式碼的可讀性

/* 印出 Hello C! 字串 */
/* 印出 Hello World! 字串 */

↓

加上註解可增加程式的可讀性



(3) 編譯與連結程式

程式撰寫完畢，接下來必須先利用編譯器將程式碼轉換成電腦看得懂的東西。經由編譯程式的轉換後，若是沒有錯誤，再透過連結器與其它函數模組連結，原始程式才會變成可執行的程式。若是編譯器在轉換的過程中，碰到不認識的語法、未先宣告的變數…等等，此時必須先訂正這些錯誤，再重新編譯，如果沒有錯誤後，才可以正確的執行程式。

(4) 執行程式

通常編譯完程式，沒有錯誤後，編譯程式會製作一個可執行檔，於 DOS 或 Unix 的環境下，只要鍵入這個執行檔的檔名即可執行程式。如果是在 Turbo C、Visual C++ 或 Dev C++ 的環境裡，通常只要按下某些快速鍵或是選擇某個選單即可執行程式。

(5) 除錯與測試

如果所撰寫的程式能一次就順利的達成目標，真是件值得高興的事。但是有的時候，您會發現明明程式可以執行，但執行後卻不是期望的結果，此時可能犯了「語意錯誤」(semantic error)，也就是說，程式本身的語法沒有問題，但在邏輯上可能有些瑕疵，所以會造成非預期性的結果。所以您必須逐一確定每一行程式的邏輯是否有誤，再將錯誤改正。

若程式的錯誤是一般的「語法錯誤」(syntax error)，就顯得簡單許多，您只要把編譯程式所指出的錯誤訂正後，再重新編譯即可將原始程式變成可執行的程式。除了除錯之外，您也必須給予這個程式不同的資料，以測試它是否正確，這也可以幫助您找出程式規劃是否夠周詳等問題。

以 Dev C++ 為例，Dev C++ 提供了視覺化的除錯功能，可以追蹤程式的執行流程，並可查看變數的值，使用起來相當的方便。

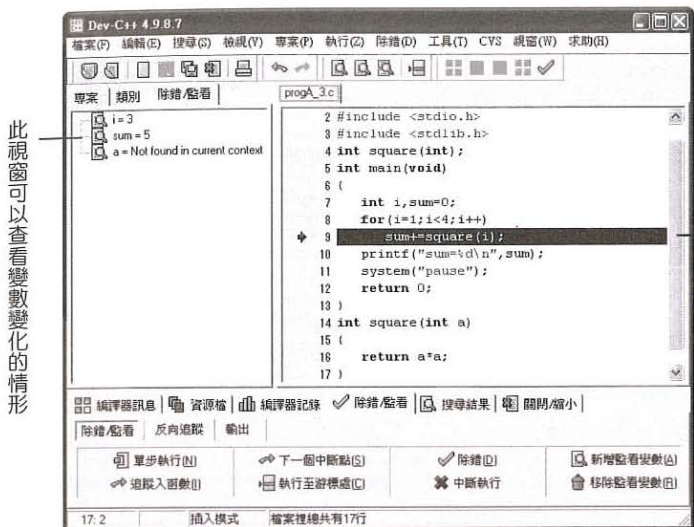


圖 1.2.4

Dev C++提供了完整的除錯功能（請參閱附錄 A）

有趣的是，程式的除錯稱為 debug，de 是去除的意思，bug 就是小蟲囉！debug 也就是去除小蟲之意。把程式碼裡的錯誤稱為小蟲是有個歷史典故的，這個典故要從 1947 年研究 MARK II 大型電腦的實驗室內的一隻飛蛾談起。

MARK II 電腦的研究人員有一天發現電腦當機了，經過一番搜查後發現原來是有一隻蛾卡在繼電器（relay）上，導致電路短路，因而造成電腦無法正常運作。有一個幽默的作業員在維修記錄本上寫下這麼一段話：

Relay #70 Panel F (moth) in relay

First actual case of bug being found.

大意是說，編號 #70 的繼電器上出了問題，這是在電腦上發現的第一隻真正的臭蟲。至今您還可以在網路上找到這份維修記錄本：

Photo # NH 96566-KN First Computer "Bug". 1945

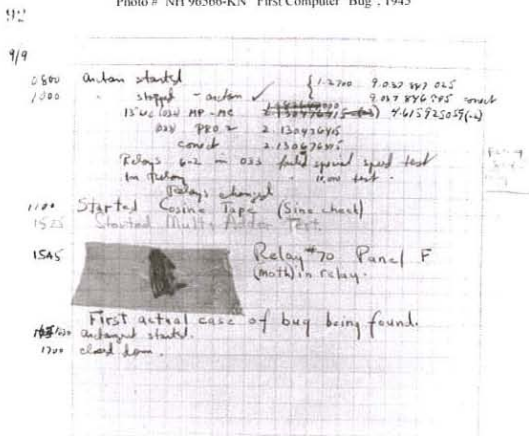


圖 1.2.5

記載電腦 bug 的維修
記錄本

本圖片轉載自 <http://www.computersciencelab.com>

從這位幽默作業員的記錄可知，bug 一詞早已被用來當成是電腦裡程式的錯誤，只是他發現的是一隻真正的 "電腦臭蟲"。事實上，當時在哈佛大學的電腦專家早已使用 bug 來形容導致電腦運作上的小差錯，以及用 debug 來形容解決問題的過程，現在那隻倒楣的飛蛾只是給了這個比喻一個真正存在的解釋。

(6) 程式碼的修飾與儲存

當程式的執行結果都沒有問題時，您可以再把原始程式做一番修飾，例如將它修改得更容易閱讀（例如將變數命名為有意義的名稱等）、或者是把程式核心部份的邏輯重新簡化…等等，以能夠做到簡單、易讀的原則所設計出來的程式，就是一個很棒的程式哦！此外，要記得把原始程式儲存下來，若是您有過一次痛苦的經驗，就會瞭解儲存的重要性。

於下圖中，我們把程式設計的六個步驟繪製成流程圖的方式，您可以參考上述的步驟，來查看程式設計的過程：

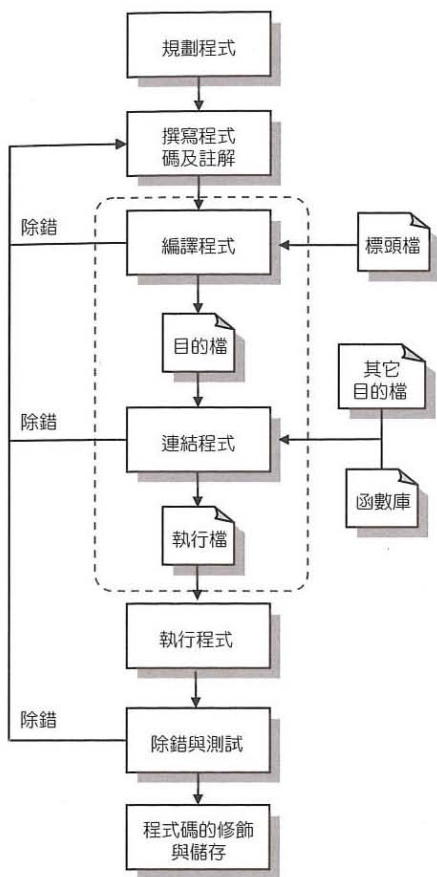


圖 1.2.6

程式設計的基本流程

1.3 撰寫第一個 C 程式

看了這麼多，讓我們快些進入 C 的世界吧！您可以使用任何喜愛的文字編輯器來撰寫程式（如 Windows 裡的記事本），撰寫完畢後，再拿到 C 的編譯器中加以編譯執行。不過一般來說，由於 C 都有提供編輯器供使用者編輯程式，因此大部分的使用者都會選擇在 C 的編輯器裡撰寫程式。



1.3.1 程式的編輯與撰寫

接下來我們以 Dev C++ 的環境為例，來撰寫第一個 C 語言。如果您不熟悉 Dev C++ 的操作，可以參考本書的附錄 A，把 Dev C++ 設定成適合最佳的操作環境。

在此要提醒您，C 語言有大小寫之分，請您在鍵入程式碼時，務必區分大小寫。另外，程式碼前面的行號只是為了方便解說，它們並不是程式碼的一部份，所以請不要將它們敲進去。下面是本範例的完整程式碼，請將它們鍵入 Dev C++ 的編輯視窗中：

```
01  /* progl_1, 第一個 C 程式碼 */
02  #include <stdio.h>
03  #include <stdlib.h>
04  int main(void)
05  {
06      printf("Hello C!\n");          /* 印出 Hello C! 字串 */
07      printf("Hello World!\n");      /* 印出 Hello World! 字串 */
08
09      system("pause");
10      return 0;
11  }
```

下面的視窗為鍵入程式碼之後的情形：

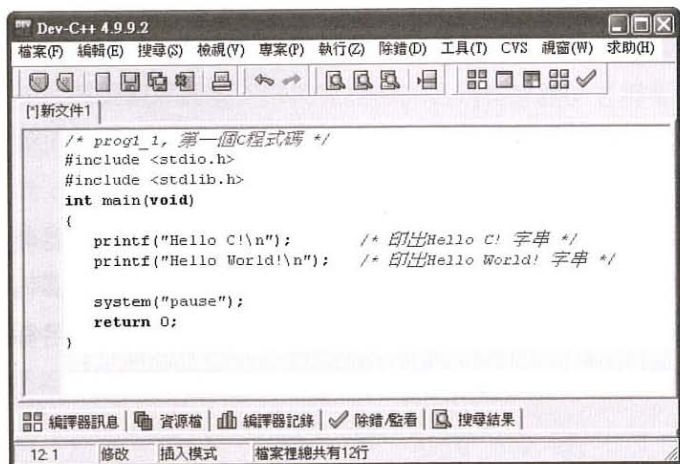


圖 1.3.1

程式碼鍵入 Dev C++ 的情形

您可以注意到 Dev C++ 使用不同的顏色來代表程式碼裡各種不同的功用，例如字串是以紅色顯示，而程式的註解以深藍色顯示。這個設計更有利於程式碼的編輯、修改以及除錯，這也是筆者選擇 Dev C++ 做為 C 的教學軟體的主要因素之一。

鍵入好之後，請先將程式碼儲存起來，以方便稍後的編譯。只要選擇「檔案」功能表裡的「儲存」，或者是在工具列裡按下「儲存」按鈕即可儲存檔案。建議您將檔名存成 `prog1_1.c`，以方便後續的操作，如下圖所示：

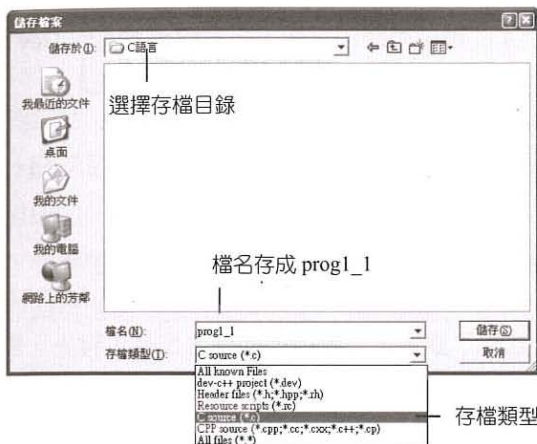


圖 1.3.2

將程式碼存檔

存檔類型請存成 C source (*.c)

1.3.2 程式碼的編譯、執行

程式寫完後，接下來就是要將原始程式變成可執行的程式。您可以使用編譯程式的快速鍵，或是功能表的選項來完成編譯與執行的動作。以 Dev C++ 為例，您可以利用下列幾種方式編譯及執行程式：

1. 選擇「執行」功能表中的「編譯」來編譯程式，然後選擇「執行」功能表裡的「執行」來執行它。
2. 選擇「執行」功能表裡的「編譯並執行」，此時 Dev C++ 在編譯完程式之後，便會自動執行程式。



prog1_1.c 的程式碼經過編譯與執行後，會出現如下圖的執行結果：

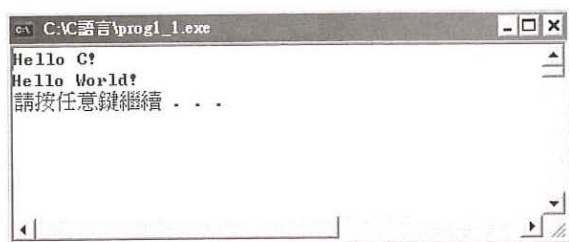


圖 1.3.3

程式 prog1_1 執行的
結果

您可以看到程式執行的結果會在 DOS 視窗中顯示，按下任意鍵即會回到 Dev C++ 的編輯環境裡。

1.4 編譯與執行的過程

C 語言編譯的過程中會產生一個「目的檔」(object file)，到底什麼是「目的檔」呢？當編譯程式進行編譯時，編譯程式除了要檢查原始程式的語法是否正確外，還要將標頭檔(header file)讀進來，根據這個標頭檔內所記載的函數原型(prototype)，檢查程式中所使用到的函數用法是否合乎規則。當這些檢查都沒有錯誤時，編譯程式就會產生一個目的檔(Dev C++ 在預設的情況是使用完目的檔便將它刪除，所以您看不到這個檔案，不過可以更改 Dev C++ 的設定選項來留住它)，所以「目的檔」即代表一個已經編譯過且沒有錯誤的程式。

目的檔產生後，就該連結程式(linker)忙碌了，連結程式會將其它的目的檔及函數庫(library)連結在一起後，成為一個「.exe」可以執行的檔案。當 C 程式變成可執行檔後，它就是一個獨立的個體，不需要 Dev C++ 的環境即可執行，因為連結程式已經將所有需要的函數庫及目的檔連結在一起了。



如果想看看 Dev C++ 把原始程式與執行檔放在什麼地方，您可以到程式所存放的資料夾（下面的範例是將檔案存放在「C 語言」資料夾）中找到這些檔案的蹤影：

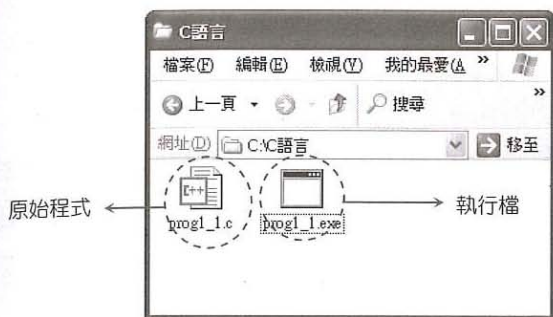


圖 1.4.1

查看原始檔與編譯連結後的執行檔

那麼，什麼又是「函數庫」呢？函數為 C 語言的基本單位，也就是說，C 語言是由函數所組成的。C 語言已經將許多常用的函數寫好，並將這些函數分門別類（如數學函數、標準輸出輸入函數等），當您想要使用這些函數時，只要在程式中載入它所屬的標頭檔就可以使用它們，這也是 C 語言迷人的地方哦！這些不同的函數集合在一起，就把它們統稱為「函數庫」。下圖為原始程式編譯及連結的過程：

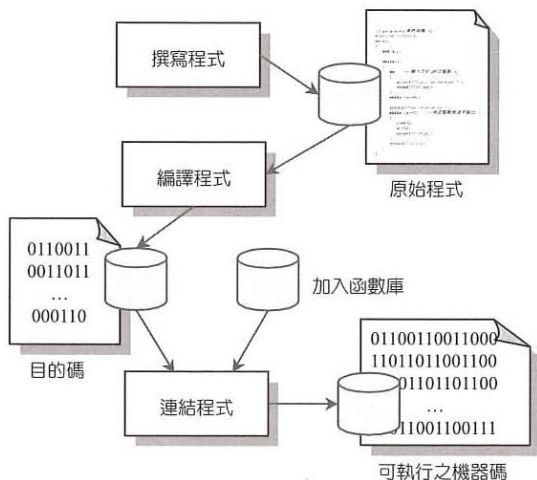


圖 1.4.2

原始程式編譯及連結的過程



1.5 本書的編排與慣例

本書的編排是以方便讀者的學習為導向，下面列出了本書的編排方式與字型的使用慣例，以方便您的閱讀：

• 程式碼與程式的輸出

本書的程式碼均以「Courier New」的字型來印出，並把程式的輸出部分列於程式的後面；於程式執行時，需要使用者輸入的部分以粗的斜體字來表示。此外，重要的程式碼會加上底色或粗體字來凸顯。以一個簡單的程式碼為例，您可以看到本書中所使用的程式碼及輸出的慣用法：

程式的行號，它們不屬於程式碼的一部份，只是為方便閱讀及解說，

程式碼，字型為 Courier New

```

01  /* prog1_2, 程式格式的說明 */
02  #include <stdio.h>    /* 將 stdio.h 這個檔案含括進來 */
03  #include <stdlib.h>   /* 將 stdlib.h 這個檔案含括進來 */
04  int main(void)
05  {
06      char ch;
07      printf("Input a character:");
08      scanf("%c",&ch);    /* 由鍵盤輸入字元並指定給變數 ch */
09      printf("ch=%c, The ASCII code is %d\n",ch,ch);
10      system("pause");
11      return 0;
12  }
```

重要程式碼會加上底色

/* prog1_2 OUTPUT

由使用者所輸入的部分，以粗斜體表示

Input a character:***R***
ch=R, The ASCII code is 82

程式的輸出部分

-----*/



- 本書所使用的作業系統與編譯程式

C 的某些特性會隨著作業系統與編譯程式而異。例如，較早期的編譯程式如 Turbo C，其整數型態的變數佔了兩個位元組，但 Dev C++、Visual C++ 等則佔了四個位元組。本書所使用的作業系統為 Windows XP，編譯程式則使用 Dev C++ 5.0 與 Visual C++ 6.0。

現在您對 C 語言應有初步的概念了。雖然本書的編譯程式是使用 Dev C++ 與 Visual C++，不過如果您是使用其它的編譯程式（如 Turbo C 或 Borland C++ builder 等），本書的程式碼一樣可以正常的編譯與執行喔！



5. 接續習題 4，PASCAL 語言發展的時間點較 C 語言來的長或短？
6. 接續習題 4，早期在商業上常用的程式語言 COBOL，它於何時開始發展？它的發展期間點比 C 語言來得早還是晚？
7. 著名的繪圖軟體 AUTO CAD，它內建有 AUTO LISP 語言，可用來在 AUTO CAD 裡撰寫程式。AUTO LISP 的語法源自 LISP，試問 LISP 語言於哪一年開始發展？
8. 於習題 4 顯示的程式語言發展圖中，每一種語言的功能特性都會以不同的符號和顏色作區隔。例如 C 語言是 Procedural，也就是程序式的語言，而 C++則是物件導向的語言（Object oriented）。試指出還有哪些語言是物件導向的語言？

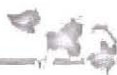
1.2 程式的規劃與實作

9. 撰寫一個好的程式，必須經歷哪六大步驟？
10. 請以流程圖繪出如下的敘述：“如果天氣好，就去爬山，否則上健身房，不管爬山或上健身房，最後都要回家吃晚飯”。
- ✚ 11. 試說明 bug 和 debug 在程式設計裡各代表的意義。
- ✚ 12. 試說明語意錯誤與語法錯誤的意義與不同處。

1.3 撰寫第一個 C 程式

- ✚ 13. 試修改 prog1_1，使得它可以印出 "我愛 C 語言" 一行中文字。
14. 試修改 prog1_1，使得它可以印出 "我愛 C 語言" 及 "這是我的第一個 C 語言程式" 兩行中文字。
15. 試撰寫一程式，利用 printf() 函數印出下面的圖案（不需使用迴圈，每一列星號請用一個 printf() 函數來列印）：

```
*  
**  
***  
****  
*****
```



16. 試以 `printf()` 函數印出下面的圖案（不需使用迴圈，每一列星號請用一個 `printf()` 函數來列印）：

```
  *  
 ***  
*****  
*****  
*****
```

17. 試撰寫一程式，利用 `printf()` 函數以星號和空白字元印出下面的圖案：

```
*****  
*****  
**      **  
**      **  
*****  
*****
```

1.4 編譯與執行的過程

18. 試說明原始檔案、目的檔案與執行檔的差別。



19. 連結程式可以為我們做哪些事？

20. 試簡單畫出程式編譯與連結的過程。