國立屏東大學 資訊工程學系 C++程式設計入門教材

2. Hello, C++!

C++語言[1]是由比雅尼．史特勞斯特魯普博士(Bjarne Stroustrup)[2]所創，目前為最主流、最重要的高階程式語言之一。本章將簡述C++語言的發展歷程，並以一個可以輸出「Hello, C++!」的程式為讀者示範C++語言的程式開發流程，並分別針對主流的Microsoft Windows、Linux與Mac OS等作業系統，介紹相關的C++語言編譯器與開發工具，包含了在Microsoft Windows系統中常被使用的Dev-C++開發工具、在Linux系統中的GNU Compiler Collection以及在Mac OS系統的Clang。讀者可以依據慣用的作業系統，參考本章的內容為後續的學習準備好所需的開發環境。

2.1 發展歷程

C++語言可以被視為是C語言的後繼者，不但承襲了C語言的高效率還更進一步支援新穎的物件導向特性，成為目前最為重要的(同時也是產業界使用最廣泛的)程式語言之一。本節將從其前身(也就是C語言)開始，為讀者簡述其發展歷程，並彙整C++語言相關的國際標準。

2.1.1 Unix系統與C語言

影響現代作業系統甚鉅的Unix作業系統，最初是由美國貝爾實驗室（Bell Laboratories）的丹尼斯．里奇（Dennis Ritchie）與肯．湯普森（Ken Thompson）在PDP-7電腦上以組合語言（assembly language）[3]進行開發，雖然執行效率很高，但卻不易開發與維護。所以後來當Unix計劃要移植到新的PDP-11電腦時，他們希望能使用高階程式語言（high-level programming language）重新改寫整個作業系統，但是卻苦無適合的程式語言可擔此重任。因此他們計畫先開發一套新的高階程式語言，再用以進行Unix系統的重新改寫。首先，在1969年，湯普森先嘗試修改英國劍橋大學的馬丁．里察德(Martin Richards)所開發的BCPL（Basic Combined Programming Language）程式語言[4]，開發出一套名為B的程式語言，可惜因缺乏對PDP-11特性的支援而以失敗告終。後續在1972年，里奇再以B語言為基礎，設計出一套新的高階程式語言 — C語言，並用來在Unix Version 2裡開發一些工具程式。由於使用C語言所寫的程式，不但比起組合語言更容易開發與維護，且其執行效率也很接近組合語言，他們兩人開始合力使用C語言將整個Unix作業系統改寫，其最終成果就是1973年所發表的Unix Version 4。後來隨著Unix作業系統被陸續地移植到其它電腦系統上，C語言也隨之拓展到了許多不同的作業平台與電腦系統之上。

由於C語言具備優異的執行效率（efficiency）與高度的可移植性（portability）等優點，很快地就成為眾多程式設計師的首選，廣泛地被採用來設計作業系統、系統程式、工具程式以及各式各樣的應用程式。C語言不但是目前產業界使用最廣的程式語言之一，其語法與觀念也成為了後續許多程式語言學習或致敬的對象，是當代絕大多數程式語言的共同基礎。從語法上來看，C語言又被稱為是結構化程式語言(Structured Programming Language)，它支援使用循序(Sequence)、選擇(Selection)及重複(repetition)等簡單且具有層次的流程架構來編寫程式。整體來說，C語言具有很多優點，茲簡述如下：

- 精簡但強大：C語言是一個非常精簡的語言，相較很多其它的程式語言，它僅有少量的指令卻能夠設計出包含作業系統在內的各式應用，可說是既精簡又強大的程式語言。C語言的語法雖然精簡，但其所支援的資料型態、算數運算子與邏輯運算子都相當地完整，讓C語言可以實現各式個樣運算的需求。同時，為了維持語言本身的精簡，C語言將許多的功能獨立設計成函式庫，包含輸入/輸出、字串處理、檔案操作、記憶體管理等相關函式庫，Programmer在開發各種程式時，只要視需要將特定的函式庫

載入即可使用。

- 高效率：由於C語言原始的用途是用以實作Unix作業系統，因此如何在記憶體有限的情況下，讓作業系統仍能夠快速的運作，就成為了C語言設計的目標之一。從結果來看，C語言的確做到了這一點，它支援包含位元運算、記憶體存取在內的低階操作，甚至允許在C語言的程式中使用assembly language的程式碼，因此其程式的執行效率相當高。
- 高度可移植性：隨著Unix系統被移植到許多不同的電腦系統上，C語言也跟著出現在各式的平台之上。後續又透過標準化的努力，C語言成為了portability（可移植性）相當高的程式語言。雖然不同平台之間，不論在硬體或是作業系統上都存在著許多差異，但標準化後的C語言，在所有平台上都可以維持相同的語法規則。使用C語言所撰寫的程式，往往僅需要小幅度的修改，有時甚至完全不需要修改，就可以在其它電腦平台上編譯並加以執行。
- 良好的彈性與延展性：雖然C語言是用以設計Unix系統的程式語言，但這並不會限制住C語言的用途。事實上，從小型的embedded system（嵌入式系統）到大型的伺服器上，都可以看到C語言的各式應用，C語言不但適合開發簡單的小型程式，由於C語言支援模組化設計，因此大型的軟體也可以使用C語言來進行設計。
- 標準化，C語言有國際標準可遵循，所以儘管在不同平台上的實作存在著一些差異，但在語法上則有共通遵循的標準。做為C語言的創作者Dennis Ritchie於1978年與Brian Kernighan合著了「The C Programming Language」一書[5]，成為了在當時人手一本的聖經。當時C語言還未展開標準化，人們就將這本書內所規範的C語言，稱之為「K&R C」，成為了無形中的一種標準。後來到了1983年，American National Standards Institute（ANSI，美國國家標準協會）開始制定C語言的標準，並於1988年完成、1989年通過成為了ANSI標準X3.159-1989，一般將這個版本又稱之為C89，後續又在1990年，通過成為International Organization for Standardization（ISO，國際標準組織）的標準ISO/IEC 9899:1990，又常被稱為C90，在1999年，ISO又通過了更新版本的C語言標準ISO/IEC 9899:1999，或被稱為C99，2011年12月8日，ISO又發佈了更新的C語言標準ISO/IEC 9899:2011，被稱之為C11，對於C語言而言，標準化是一個相當重要動作，因為這讓我們得以在不同平台上，都能使用一個統一的C語言標準。

2.1.2 C++物件導向程式語言

隨著電腦功能愈趨強大，使用者的需求也不斷提升，程式設計也愈見困難與複雜。支援結構化程式設計的程式語言似乎已不敷所需。誠如艾茲赫爾．戴克斯特拉(Edsger W. Dijkstra)[6]所說：

> As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming had become an equally gigantic problem.
>
> 翻譯吐司→(當世上還沒有電腦的時後，並不存在程式設計的問題；但當我們有了幾台功能弱小的電腦後，程式設計就開始成為了小問題；現在我們擁有了功能強大的電腦，程式設計也就隨之變成了巨大的問題。）[7]

在過去程式設計受限於電腦只具有簡單的功能，其所能完成的工作與使用者的需求都相對有限；然而隨著電腦功能的不斷提升，程式設計所能實現的功能以及使用者的需求都隨之大幅地增加 – 當然，程式設計的困難度與複雜性也伴隨著大幅提升。中大型軟體的開發與設計，不論是在功能、規模、複雜度等各方面，都比設計開發小型軟體有者極大的差異。試想，自己用手摺紙飛機的方法與經驗，完全不能套用在製造大型的民航客機上。資訊界將這種日趨複雜與困難的軟體開發問題，再加上大型軟體的開發週期長、費用不斷追加、可重用性低、品質低落等問題，稱之為「軟體危機(Software Crisis)」[8]。傳統的結構化程式語言似乎不能應付日趨嚴重的「軟體危機」，因此新一代的「物件導向程式語言(Object-Oriented Programming Language)」也就應運而生。

物件導向程式語言具有封裝(Encapsulation)、繼承(Inheritance)與多型(Polymorphism)等三大特性，使用與傳統程式語言不同的思維方式進行程式設計與軟體開發 – 使用物件來對映真實或虛擬世界中的人、事、時、地、物，再透過物件與物件、物件與使用者間的互動來完成特定的應用目的。由於物件導向的三大特性之一的繼承性(Inheritance)，有助於提升程式碼的可重用性(reusability，又譯做再利用性)，因此在過去曾被視為是解決軟體危機的不二法門，因此物件導向程式語言一直受到廣大的矚目。不過要注意的是，儘管物件導向程式語言的確有助於解決軟體危機，但絕對不是唯一的解答，要真正解決軟體危機還有很多其它問題必須考慮 – 事實上，這已經屬於「軟體工程(Software Engineering)」[9]的範疇了，本書將不予贅述。

讓我們再回到物件導向程式語言的發展歷程。最早期的物件導向程式語言可以追溯到上一世紀60年代與70年代所發表的Simula與Smalltalk語言，其中Simula是由奧利-約翰．達爾(Ole-Johan Dahl)和克利斯登．奈加特(Kristen Nygaard)所設計，是第一個提出「類別(Class)」概念的高階程式語言，據傳一開始是為了管理船隻所開發的 – 將不同種類的船隻定義為不同的類別，並為真實世界中的每一艘船隻，依其所屬種類的類別定義對映物件的屬性與行為。受到Simula語言的啟迪，在上一世紀70年代初期，全錄PARC研究所發表了Smalltalk語言，不但支援動態地建立、刪除物件，更首次引入了「繼承性(Inheritance)」的概念。做為物件導向程式語言的開拓者，Simula與Smalltalk語言無疑是成功的，它們將物件導向的程式設計概念成功地推廣開來，讓當時的程式設計師能接觸到除了結構化程式語言外的其它選擇 – 而且是更好的選擇！後續80年代接棒誕生的C++語言，由於承襲了C語言的語法基礎與新穎的物件導向觀念，旋即吸引了為數眾多C語言程式設計師投身其中，終於成功地普及了物件導向程式設計。

C++語言是於1983年，是由比雅尼．史特勞斯特魯普(Bjarne Stroustrup)博士[10]任職於貝爾實驗室(Bell Laboratories)[11]時所開發的一套通用程式語言(General-Purpose Programming Language)[12]，最初被稱為「C with Classes（具備類別的C語言）」，是以C語言做為基礎再加上對於類別的支援所設計。後來陸續加入了許多先進的程式語言特性，並改名為C++語言，以表明其做為C語言後繼者的企圖[13]。

做為C語言的後繼者，C++語言擁有和C語言相同的基礎，自1983年誕生以來就受到許多原本C語言程式設計師的喜愛；再加上C++語言的物件導向特性，使它迅速成為產業界最受歡迎的程式語言之一。與C語言發展的脈絡相似，C++語言一直致力於標準化的發展。1998年11月，國際標準化組織(International Organization for Standardization，ISO)與國際電工委員會(International Electrotechnical Commission，IEC)共同頒佈了C++語言的第一個國際標準起，後續包含了多重繼承(Multiple Inheritance)、運算字重載(Operator Overloading)、虛擬函式(Virtual Function)、例外處理(Exception)、執行時期型態資訊(Runtime Type Information，RTTI)與命名空間(Namespace)等創新概念與特性，亦陸續逐漸納入標準之中。這些標準都被規範於ISO/IEC 14882文件並以發佈年份做為區別，其中包含了ISO/IEC 14882:1998、ISO/IEC 14882:2003、ISO/IEC 14882:2011、ISO/IEC 14882:2014與ISO/IEC 14882:2017等多項標準，依發表年份區別，這些標準又被稱為是C++98、C++03、C++11、C++14與C++17。

這些國際標準規範了C++語言的語法與語意規則，所有相關產業人員在開發C++語言的編譯器時，都應該依照這些標準來進行。當然，這些標準都是考慮到C++語言在當時以及未來的需求與發展所制定，現有的C++編譯器不一定能完整支援這些標準。目前大部份的C++語言編譯器應該都已經完整支援C++98與C++03的規範，因此本書並不會特別註明相關內容是符合C++98或C++03的版本；不過由於包含C++11、C++14以及最新的C++17在內的新規範，絕大多數的編譯器都只有部份的實作，所以為了清楚說明起見，本書在使用到C++11、C++14與C++17等新版本的C++標準時，會為讀者註明其所使用的相關版本。不過，做為一本C++語言的入門書籍，本書絕大多數內容都僅針對C++98與C++03，僅有少數內容會使用到C++11、C++14與C++17。

由於C++承襲了C語言的高效率[14]，與廣大的C語言開發人員，自其誕生以來，C++語言一直都是最受歡迎與最受重視的程式語言之一，同時也是幾乎所有大專校院資訊相關系所的必修程式語言。本章後續將針對其程式設計的開發流程加以說明，並透過一個簡單的程式來示範在Windows、Linux與Mac OS系統上的詳細開發過程。

2.2 程式開發流程

使用C++語言進行程式設計的流程相當簡單，請參考figure 1，我們將其步驟說明如下：

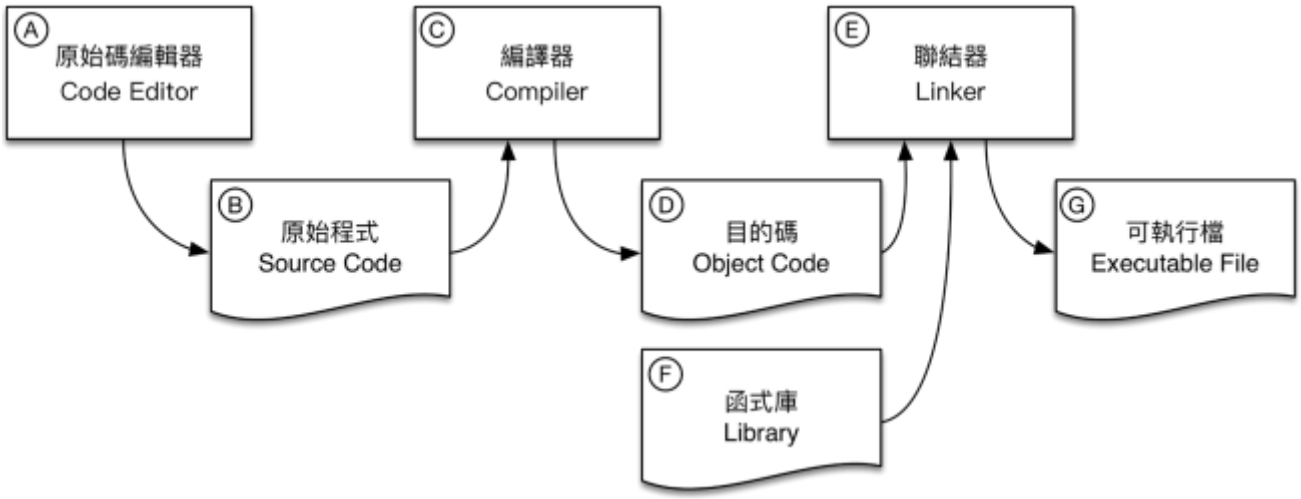

Fig. 1: C□□語言程式設計流程

首先，我們必須使用程式碼編輯器□Code Editor□在figure 1中標示為A之處）來撰寫程式，完成後會產生C□□語言的原始程式□Source Code□在figure 1中標示為B之處）檔案，為方便辨識起見，其副檔名通常會命名為□cpp□□我們通常將撰寫原始程式的過程稱為「寫程式□□□Coding□或「編程」。值得注意的是，由於C□□語言的原始程式檔案格式為純文字格式，所以你可以選擇使用任意的文字編輯器□Text Editor□[15]來進行原始程式的撰寫，不過程式碼編輯器通常提供了許多便利程式開發的功能，例如程式語言的語法突顯□Syntax Highlighting□□自動縮排□Auto Indenting□□自動完成□Auto Completion□等功能，對於程式設計師而言是更方便的選擇。

當原始程式撰寫完成後，還必須交由編譯器□Compiler□在figure 1中標示為C之處）來將其轉換為可執行檔□Executable File□的格式，才能夠在作業系統內加以執行。在這個轉換的過程中，編譯器會先確認原始程式碼是否符合C++語言的語法規則，然後才會將其轉換為目的檔□Object Code□在figure 1中標示為D之處），其中包含有可以在CPU上執行的機器碼□Machine Code□指令以及其執行所需的資料。如果在編譯時發現錯誤，則必須重新審視原始程式碼的正確性，將錯誤加以修正後再重新進行編譯，直到成功為止才會產生目的檔。我們將這種找出程式碼的錯誤並加以修正的過程，稱之為除錯□Debug□□對於程式設計師而言，除錯是非常重要的能力之一。

一旦成功地產生目的檔後□C++語言的編譯器就會自動啟動聯結程式□Linker□在figure1中標示為E之處），來將目的檔與相關的函式庫□Lbrary□在figure 1中標示為F之處）進行結合，以產生符合作業系統要求的可執行檔□Executable File□在figure 1中標示為G之處），例如Microsoft Windows平台下的EXE檔，或是Unix/Linux系統中的ELF檔；當然，如果在進行聯結時發生錯誤，一樣必須進行原始程式碼的除錯直到聯結成功為止。最後，所產生的可執行檔就可以透過作業系統加以執行。

2.3 開發你的第一個C++程式

工欲善其事，必先利其器！所有C++語言的學習者，都必須先備妥相關的工具，才能夠進行C++語言的程式開發。從前一節的說明中，我們可以得知在C++語言的程式開發流程中，最為重要的工具就是用以撰寫程式碼的程式碼編輯器（在figure 1中標示為A之處），以及用以產生可執行檔的編譯器（在figure 1中標示為C之處）。本節將針對慣用不同作業系統的讀者，推薦適用的開發工具，並且以一個最簡單的C++程式，實際示範開發過程中的每一步驟與細節。如果讀者是程式設計的初學者，建議你必須詳細地閱讀每一

個步驟，並且親自在你的電腦進行實際的操作，如此才能在開始學習C++語言前，將所需的程式開發環境準備好，以便進行後續的學習。

首先，請參考Example 1的C++程式碼，此程式是一個Console模式的程式，其執行結果會在Console輸出一個「Hello, C++!」字串。

```
C++</nowiki>程式 hello.cpp >
/* Hello, C++! */
// This is my first C++ program

#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, C++!" << endl;
    return 0;
}
```

此程式的執行結果如下所示，會在螢幕上輸出「Hello, C++!」並將游標移到下一行的開頭處：

```
Hello, C++!
```

在此我們先不解釋此程式碼的內容，待你依據本節後續的說明，完成此程式的編譯與執行後，在本章末再進行相關的說明。現在，請依照後續小節的說明，在你慣用的作業系統上完成這個hello.cpp程式的開發。本書後續所有的範例程式都可以在Windows、Linux或Mac OS等主流的作業系統上執行，不論你慣用的作業系統為何，都可以依照本節後續的說明，將本書的範例程式進行撰寫、編譯與執行等動作。具體來說，我們將在在Linux上開發C++程式節先針對使用Linux系統的讀者說明其開發流程與詳細的步驟，至於慣用Mac OS與Microsoft Windows的讀者，則請參考在MacOS上開發C++程式與在Windows上開發C++程式小節。

資訊補給站：Console模式

所謂的Console指的是一種文字界面環境，提供使用者以文字指令進行各項操作。早期在沒有視窗作業環境之前，所有電腦系統都是以此種方式操作；時至今日，雖然大部份的作業系統都已提供便節利的視窗作業環境，但仍保留Console環境，（也就是在現代作業系統中所謂的「終端機」或是「主控台」等應用程式）供我們使用，例如在Microsoft Windows系統中的「命令列提示字元」，或是在Linux/Unix/Mac OS中的「Terminal（終端機）」都是這一類型的操作環境。使用Console來下達操作指令，可以省去以滑鼠移動、點擊的動作，仍是許多程式設計師喜好的做法。目前仍有許多的應用程式（包含許多的伺服器軟體）必須在Console中執行，我們將其稱為「Console模式」的程式。

由於現代的作業系統大多數都有提供視窗作業環境，因此筆者相信許多讀者應該都不習慣Console模式的文字指令操作，也可能會覺得Console模式的程式比起現代的視窗應用程式較為單調乏味。但是開發Console模式的程式，不需要瞭解視窗應用程式背後複雜的機制與開發環境，有助於讓我們先專注於學習程式語言的基本語法，對於

程式設計的初學者來說是較為適合的。本書所使用的程式範例將以Console模式的程式為主，希望讀者們可以逐步適應並從中學習C++語言程式設計的邏輯概念，為未來的學習奠定基礎。

2.3.1 在Linux系統上開發C++程式

絕大多數的Linux版本已預先安裝了可用以撰寫程式的工具，以及C++語言的編譯器。你可以直接在Console模式中（例如使用Terminal（終端機） 軟體）使用vi（vim（emacs（pico或是joe等文字編輯器（Text Editor） 撰寫程式。現在，請你依照以下的步驟，在你的Linux系統上進行程式碼的撰寫、編譯與執行：

- **步驟1:建立（~/examples/ch1（目錄**

為便利後續的討論，首先請讀者先登入你的Linux系統，開啟並使用Terminal（終端機）軟體來輸入以下指令：

```
[user@urlinux ~]$ mkdir examples
[user@urlinux ~]$ cd examples
[user@urlinux examples]$ mkdir ch1
[user@urlinux examples]$ cd ch1
[user@urlinux ch1]$
```

上述的指令會在你的使用者家目錄內建立一個名為（examples（的目錄，用以做為配合本書進行相關的程式練習之用，並且在該目錄中再建立一個名為（ch1（的子目錄，用來存放本章相關的範例程式檔案。本書後續章節的範例程式，也請你依據此方式建立相關的目錄，以便利日後程式碼的管理，以第三章為例，相關程式碼請放置於（~/examples/ch3（目錄中。當然，你也可以依喜好與需求，自行決定目錄的名稱與其位置。

- **步驟2：撰寫hello.cpp原始程式**

接下來，請你使用文字編輯器（Text Editor（或程式碼編輯器（Code Editor（（將Example 1中的hello.cpp原始程式進行編輯，並將其檔案儲存於（~/examples/ch1（中。

首先，在文字編輯器方面，由於Linux系統有許多文字編輯器可供選擇，筆者無法一一地詳細加以說明，在此僅以Linux系統預設會安裝的vi為例，進行相關的說明（vi是隨著最早期的Unix系統就誕生的文字編輯器，目前所使用的是其後續的增強版本Vi Improved（在絕大部份的Linux系統上，如果下達（vi（或「vim（指令，就可以將其加以啟動。請在（~/examples/ch1（目錄裡，輸入以下指令來啟動vi（

```
[user@urlinux ch1]$ vi hello.cpp
```

下達此一指令後，如果你得到的是以下的訊息，就表示你所使用的Linux系統並未預先安裝有vi（

```
vi: command not found
```

如果遇到此種罕見的情況，表示你所使用的Linux系統並沒有預先安裝vi（你可以依據所使用的Linux版本，使用以下的指令自行加以安裝：

Ubuntu/Debian版本：

```
sudo apt-get install vim
```

Redhat/CentOS版本：

```
yum –y install vim-enhanced
```

安裝完成後，你將可以順利地啟動vi進行程式碼的撰寫。figure 2為在~/examples/ch1目錄中，下達vi hello.cpp指令後，順利啟動vi的執行結果。

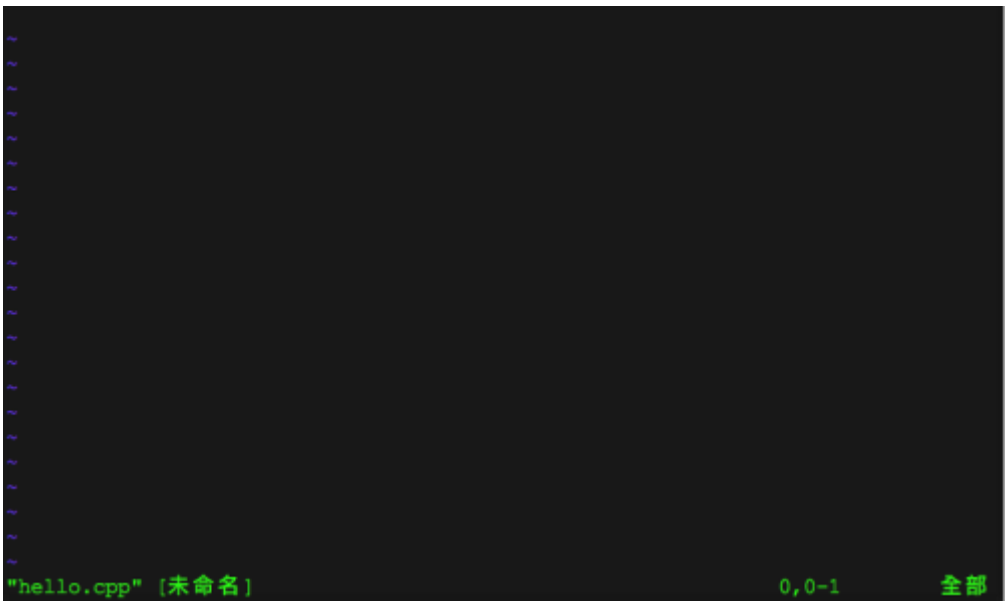

Fig. 2: vi啟動後的畫面

請輸入i進入vi的「插入Insertion模式[16]，然後將Example 1所列示的原始程式在vi中加以編輯，請參考figure 3的操作畫面。

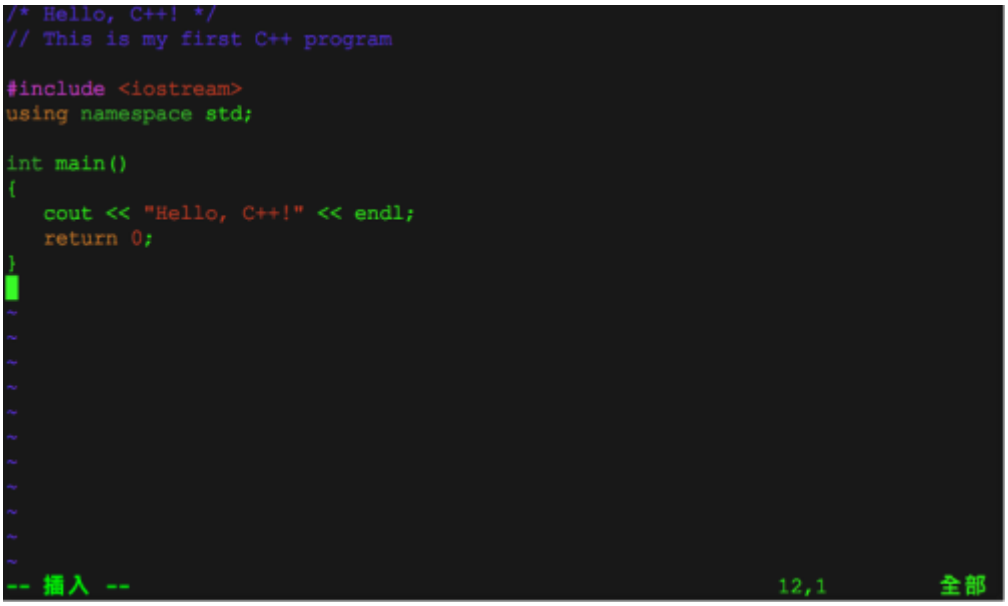

Fig. 3: 使用vi撰寫hello.cpp原始程式

完成撰寫後，請依序輸入□ESC□□□□□□□wq□□以便結束vi的操作並將檔案加以存檔。如果一切順利，你將可以在□~/examples/ch1□目錄中，看到所完成的檔案。請輸入□ls□指令，你應該可以看到以下的結果：

```
[user@urlinux ch1]$ ls
hello.cpp
[user@urlinux ch1]$
```

你也可以輸入以下的指令，確認hello.cpp檔案的內容是否正確：

```
[user@urlinux ch1]$ cat hello.cpp
hello.cpp/* Hello, C++! */
// This is my first C++ program

#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, C++!" << endl;
   return 0;
}
[user@urlinux ch1]$
```

至此，撰寫原始程式的步驟已經順利完成。當然，你也可以使用其它你所偏好的文字編輯器，來完成此一步驟，只要最後存檔於□~/examples/ch1□目錄中，並將檔名命名為□hello.cpp□即可。

- **步驟3：進行原始程式編譯**

接下來，請你使用大部份Linux系統都已預先安裝好的GNU Compiler Collection(GNU的編譯器工具集，一般簡稱為GCC)□來進行C++語言程式的編譯，其指令都常為□c++□或「g++□□並請在其後加上欲進行編譯的原始程式檔案名稱。

請在你存放□hello.cpp□原始程式的目錄中，輸入□c++ hello.cpp□指令，你應該可以看到以下的結果：

```
[user@urlinux ch1]$ c++ hello.cpp
[user@urlinux ch1]$
```

是的，你沒有看錯，如果程式碼完全正確，使用□C++□指令進行編譯時，將不會顯示任何的訊息；只有在有錯誤的情況下，編譯器才會輸出相關的錯誤訊息。你也可以使用□g++□指令來進行程式的編譯，在大部份的Linux系統中□□C++□與「g++□都是對應到相同的GCC編譯工具。你可以試著改以□g++□指令來編譯hello.cpp程式，其結果應該與使用□C++□指令一致：

```
[user@urlinux ch1]$ g++ hello.cpp
[user@urlinux ch1]$
```

資訊補給站□GNU Compiler Collection GNU Compiler Collection是由GNU所開發的編譯器工具集，其中包含有C語言以及C++語言的編譯器，一般常簡稱為GCC□GNU是一個非營利的團體，以開發一套名為GNU的開放、自由、無專屬著作權作業系統為其成立宗旨，其名稱來自於GNU's Not Unix的遞迴縮寫□GNU作業系統的開發計畫是由李察．斯托曼□Richard Mathew Stallman□於1983年所發起，後續

於1985年號召同好組成自由軟體基金會（Free Software Foundation）負責推動GNU作業系統以及其它自由軟體（亦稱為開放原始碼軟體（Open Source Software）的開發。可惜GNU作業系統的開發並不順利，至今仍未發展完成；不過自由軟體基金會倒是發表了許多自由軟體，目前被廣泛地使用在Linux作業系統中，為開放原始碼社群做出了具大貢獻。 GNU的編譯器工具集（GNU Compiler Collection）就是其中一個著名的例子，它是目前最多人使用的C語言及C++語言編譯器。

雖然大部份的Linux系統已經預先安裝有GCC編譯工具，但如果你下達「c++ hello.cpp」這個指令時，所得到的是以下的訊息，那麼就表示很不幸地你所使用的Linux系統並未預先安裝有GCC。

```
[user@urlinux ch1]$ c++ hello.cpp
c++: command not found
[user@urlinux ch1]$
```

此時，你可以依據所使用的Linux版本，使用以下的指令自行安裝GCC。

Ubuntu/Debian版本：

```
sudo apt-get install build-essential
```

Redhat/CentOS版本：

```
yum group install "Development Tools"
```

安裝完成後，你將可以順利地使用GCC中的「c++」指令來進行程式碼的編譯。建議讀者先確認你的Linux系統是否已經安裝有GCC編譯工具，以便搭配本書繼續後續的學習。

- **步驟4：執行程式**

完成程式的編譯後，你應該已經順利地得到檔名預設為「a.out」的可執行檔（Executable File）了。請先在你的工作目錄內下達「ls」指令，檢查是否已經順利得到可執行檔：

```
[user@urlinux ch1]$ ls
a.out   hello.cpp
[user@urlinux ch1]$
```

如果你沒有在目錄裡看到「a.out」檔案，那麼表示你並沒有成功地完成編譯，請退回前面的步驟，仔細檢查看看是哪裡出了問題，將問題修正後再繼續這個步驟。

請下達「./a.out」來執行此程式，你應該可以看到以下的執行結果：

```
[user@urlinux ch1]$ ./a.out
Hello, C++!
[user@urlinux ch1]$
```

至此，你的第一個C++程式的開發已經順利完成。不過要注意的是，我們是在Linux的Console環境中，使用vi文字編輯器來進行原始程式的撰寫，並使用GNU Compiler Collection來進行編譯。如果你想要使用可以在視窗環境運作的軟體來進行C++語言的程式開發，那麼你可以考慮使用包含Code::Blocks[17]、Eclipse[18]、

與NetBeans[19]等IDE開發工具，以便在單一的視窗軟體環境中進行程式的撰寫、編譯與執行或除錯等工作。另外也有一些程式碼編輯器（Code Editor），例如Sublime Text[20]、Atom[21]與Brackets[22]等，除了提供視窗使用者界面的程式碼撰寫環境外，也能夠透過設定來使用GCC或其它的C++的編譯器來進行程式碼的編譯，也是許多程式設計師偏好的選項。本書限於篇幅無法一一為讀者詳加介紹，請有興趣的讀者自行至相關網站下載安裝。

資訊補給站：IDE整合開發環境 IDE（Integrated Development Environment）整合開發環境，可以在單一軟體環境中，進行原始程式碼的撰寫、編譯與執行，甚至還可以進行程式碼的除錯。由於開發上的便利性，IDE是許多程式設計師偏好使用的開發方式。

本節後續將針對Mac OS與Microsoft Windows的使用者，介紹相關的開發流程。雖然是不同的作業系統，但其開發C++程式的流程仍有些相似之處（尤其是Mac OS與Linux系統具有很高的相似性），相關內容難免會有些重複，如果你不需要參考這些內容，請直接忽略接下來的兩個小節。我們將在程式碼說明節接續說明hello.cpp原始程式的程式碼內容。

2.3.2 在MacOS上開發C++程式

Mac OS作業系統其實是BSD的衍生系統，所以和Linux系統一樣，都是屬於類Unix的作業系統，因此前小節中所介紹的適用於Linux系統的Console模式的操作方法，也仍適用於MacOS的作業環境。Mac OS也有提供終端機（Terminal）軟體，讓我們可以進行Console模式的各項文字指令操作，包含vi（vim）、emacs、pico或是joe等文字編輯器軟體都可以在MacOS中使用，因此本節將不再重複說明，請讀者自行參考前一小節的說明完成在MacOS上開發C++程式的練習。

值得一提的是，MacOS預設的C++語言編譯器是Apple公司的Clang，而不是Gnu的g++編譯器 – 但你仍然可以使用C++或g++來進行C++程式的編譯，只不過這兩個指令將會被對應到Clang的編譯器。若是你的MacOS沒有安裝(或者你想要升級Clang的版本），請自行參考Clang安裝指引的說明加以完成。

2.3.3 在Windows上開發C++程式

Microfot Windows系統一樣可以如前兩個小節所介紹的方式，透過Console模式來進行程式的開發，但是對於大部份慣用Microsoft Windows圖形使用者界面的讀者而言，應該都不會偏好這種以指令來進行操作的方式。為了幫助讀者先專注學習C++語言的語法規則與程式設計邏輯，而不用受困於不熟悉的指令操作，筆者建議讀者可以考慮使用視窗環境的開發工具，例如Dev-C++、Code::Blocks、Eclipse、NetBeans、Sublime Text、Atom與Brackets等IDE或程式碼編輯器（Code Editor），限於篇幅，本書在此將僅就Dev-C++的安裝與使用細節提供說明，至於其它軟體請有興趣的讀者自行參考它們的官方網站。

Dev-C++是Microsoft Windows作業系統上，一套簡單易用的C++語言IDE開發工具，起初是由Bloodshed Software公司的Colin Laplace所開發，不但可以免費使用[23]，同時又具有簡單易用與高可擴充性等優點，自其推出後就受到程式設計師的矚目。後來由於Colin Laplace個人的因素，Dev-C++的開發自2005年起就陷於停頓，沒有再繼續推出新的版本。目前其後續的衍生版本是由Orwell所開發，讀者們可以至SourceForge上取得其最新的版本（截至2018年3月，Orwell所釋出的Dev-C++最新版本為5.11版）。接下來，請先依照以下的說明進行Dev-C++的安裝與設定：

- **步驟1：下載Dev-C++**

請前往網址https://sourceforge.net/projects/orwelldevcpp/files/latest/download，下載最新版本的Dev-

C++ 以5.11版為例，所下載安裝程式之檔案名稱為Dev-CPP 5.11 TDM-GCC 4.9.2 Setup.exe[24]。

- **步驟2：安裝Dev-C++**

以滑鼠雙擊所下載的安裝程式，以啟動Dev-C++的安裝程序。安裝程式一旦啟動後，首先是選擇安裝程式所使用的語言，如figure 4所示。由於此處並沒有提供中文，所以直接按下「OK」按鈕即可。

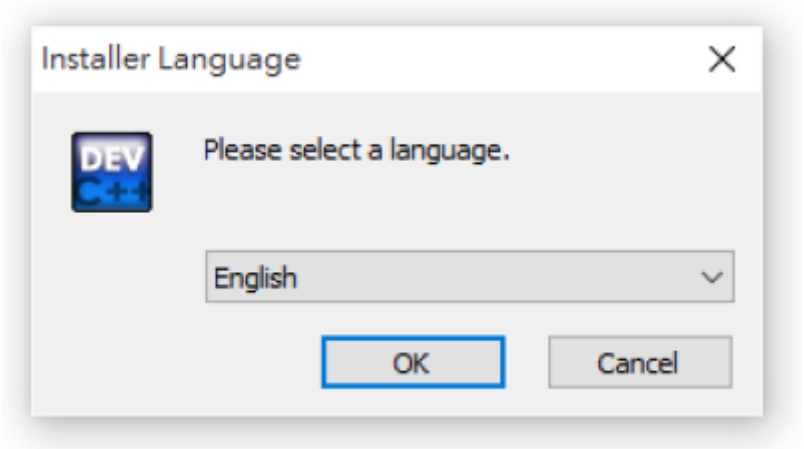

Fig. 4: 選擇Dev-<nowiki

接著出現的是授權協議， Dev-C++使用的是第二版的GNU通用公眾授權條款「GNU General Public License」Version 2」一般簡稱為GNU GPLv2」。如figure 5所示。依其授權條件，任何人都可以免費、自由地使用、共享，甚至是修改Dev-C++ 。如果你同意其授權條款，請按下「I Agree」。否則，請按下「Cancel」放棄安裝Dev-C++。

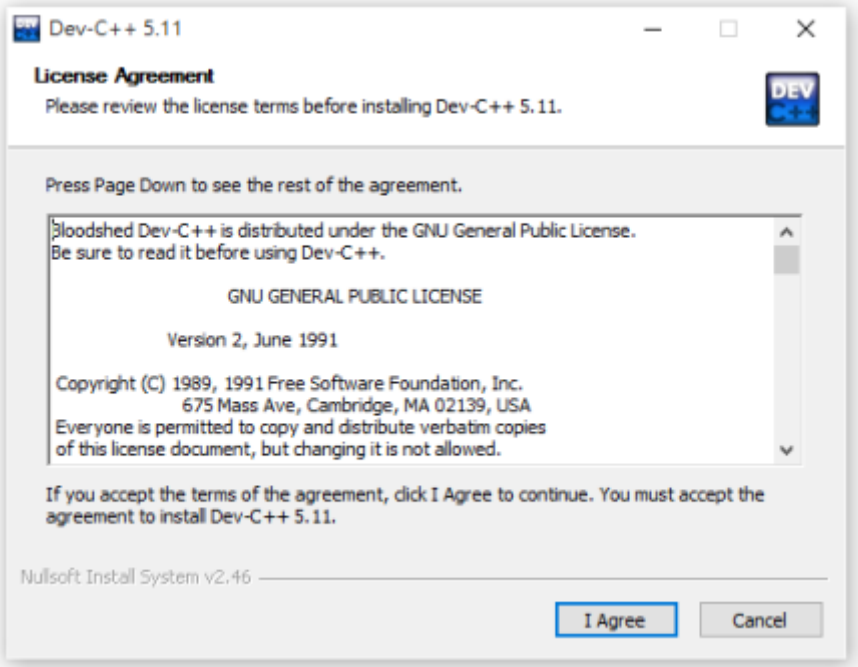

Fig. 5: Dev-<nowiki

接著是為Dev-C++的安裝，選擇需要的軟體元件，如figure 6所示。如果沒有特別的需求，直接使用預設

的選擇即可，請按下Next按鈕。

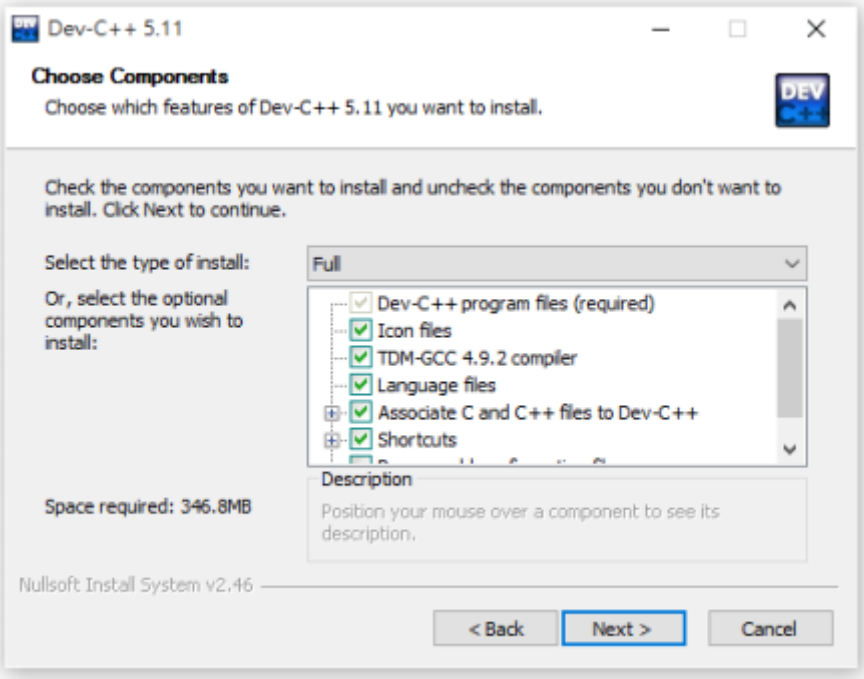

Fig. 6: 為Dev-<nowiki

在開始安裝前，你還必須指定Dev-C++的安裝目錄，如figure 7所示。請在此處選擇你偏好的安裝目錄（或是直接使用預設的安裝目錄），然後請按下Install按鈕開始進行檔案的複製與安裝設定。

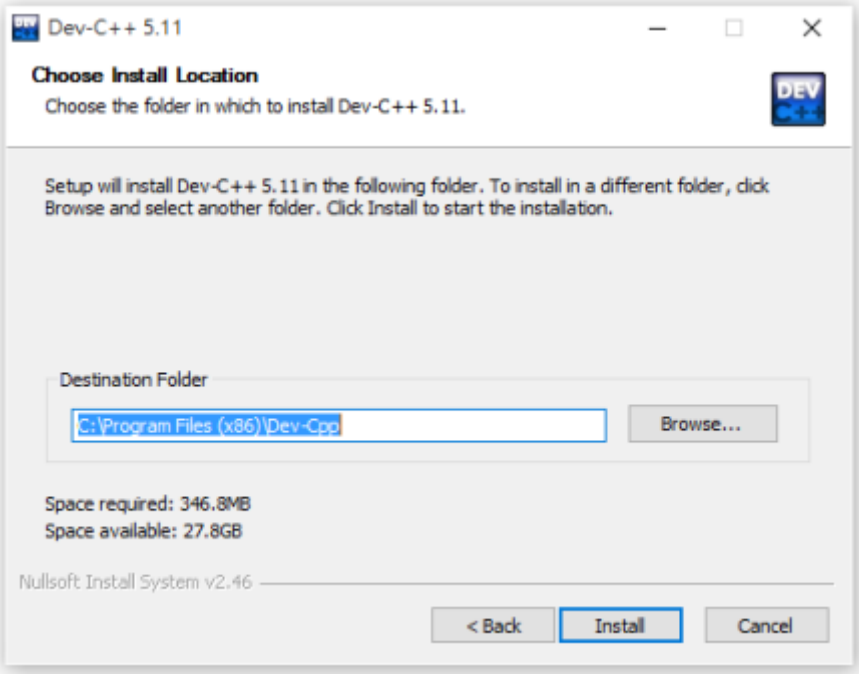

Fig. 7: 選擇Dev-<nowiki

經過一段時間的等待後，你就可以看到如figure 8的畫面。此時，請按下Finish按鈕結束安裝程式。由於預設已勾選了Run Dev-C++ 5.11選項，所以在安裝程式結束後，就會啟動Dev-C++日後，當你需要

使用Dev-C++時，只要使用滑鼠雙擊在桌面上的Dev-C++圖示（如figure 9），就可以將Dev-C++加以啟動。

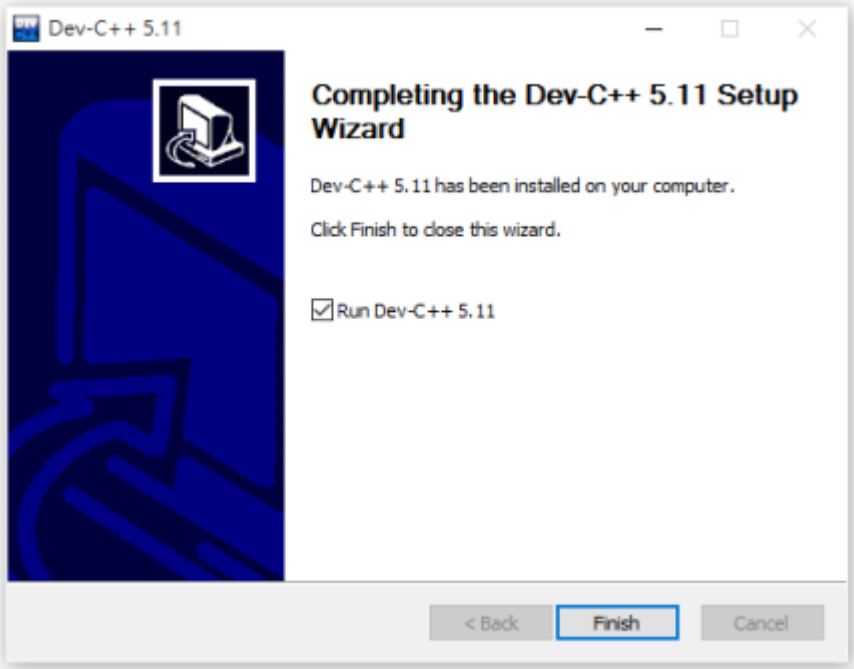

Fig. 8: Dev-<nowiki

Fig. 9: Dev-<nowiki

- **步驟3：啟動與設定Dev-C++**

首次啟動Dev-C++時，會出現如figure 10的畫面，我們可以在此處選擇Dev-C++執行時所要使用的語言，請選擇□Chinese□TW□□選項，來使用繁體中文，完成後請按下□Next□按鈕。

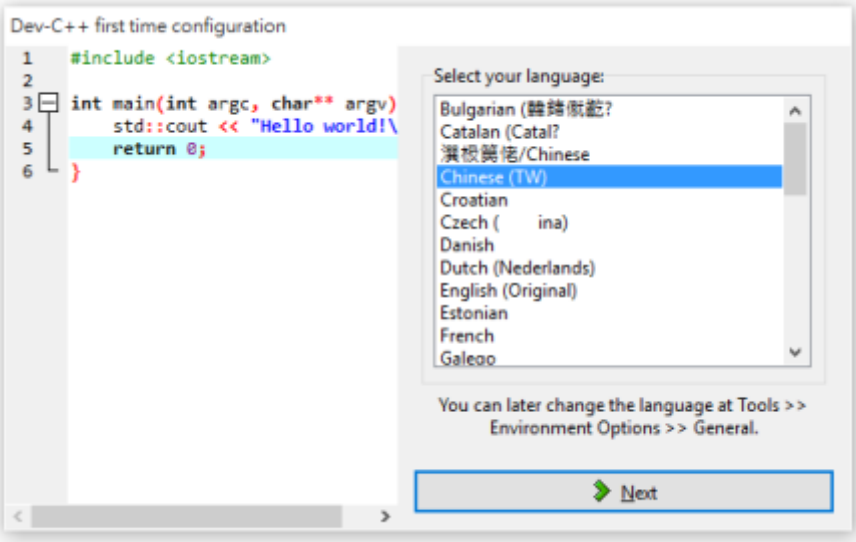

Fig. 10: 設定Dev-<nowiki

Dev C++中文亂碼問題

Dev C++在Windows 11裡存在著中文顯示及輸出的亂碼問題，具體來說，這些問題包含：

- 選擇中文語系(Chinese/Taiwan)後，選單內的中文顯示為亂碼
- 在編輯程式碼時，所輸入的中文會變成問號(??)；或是能夠正確地輸入中文，但當游標移動離開該行後，所輸入的中文變為不可視
- 程式編譯執行後，其執行結果中的中文將變為亂碼

上述問題的根源在於過去Dev C++在中文方面使用的是Big5(繁體)與GBK(簡體)的編碼方法，但在Windows11已全面改用UTF-8做為中文的編碼方法，所以才會造成上述的情況。

在Dev-C++已經沒有官方持續開發的情況下，暫時的解決方法如下：

1. 在Windows 11裡，選擇執行「設定 | 時間與語言」，再接著選擇「相關設定 | 系統管理語言設定」，如下圖：

Fig. 11: 系統管理語言設定。

2. 接著選擇在「非Unicode程式的語言」裡的「變更系統地區設定」：

Fig. 12:

變更系統地區設定。

3. 將「目前的系統地區設定」設定為「中文(繁體、台灣)」，並且不要勾選□Beta:使用Unicode UTF-8提供全球語言支援」。

Fig. 13: 變更目前的系統地區設定。

以上設定就可以解決Dev-C++的中文的亂碼問題。

接下來則可以針對字型、色彩配置及圖示，進行個人的偏好設定，如figure 14所示。在此，我們並不做任何變動（當然，你也可以視需要加以調整），直接按下「下一步」按鈕。

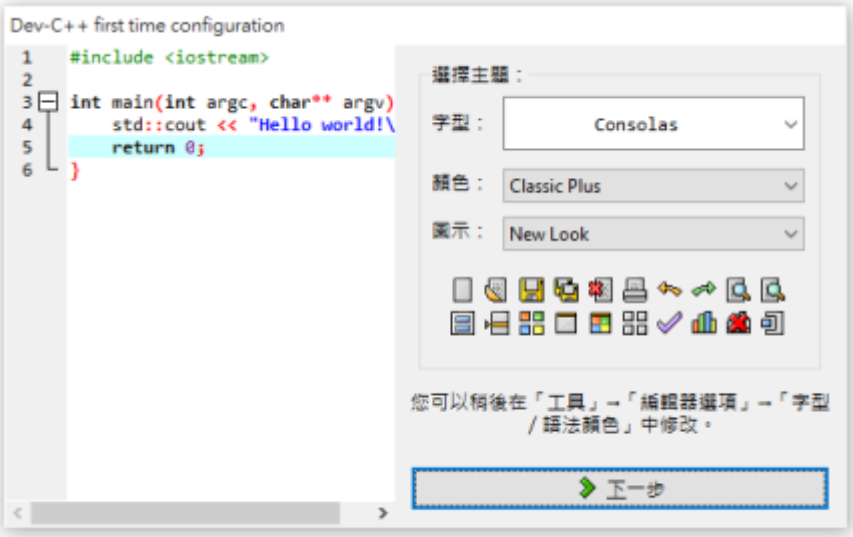

Fig. 14: 設定Dev-<nowiki

設定完成後就會看到如figure 15的畫面，直接按下□OK□按鈕，就可以開始使用Dev-C++□你將可以看到如figure 16的操作畫面。

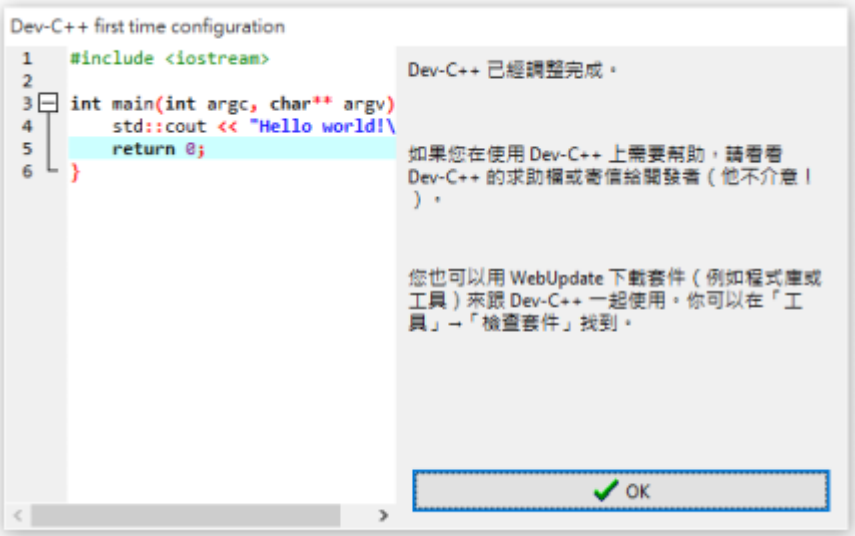

Fig. 15: 完成Dev-<nowiki

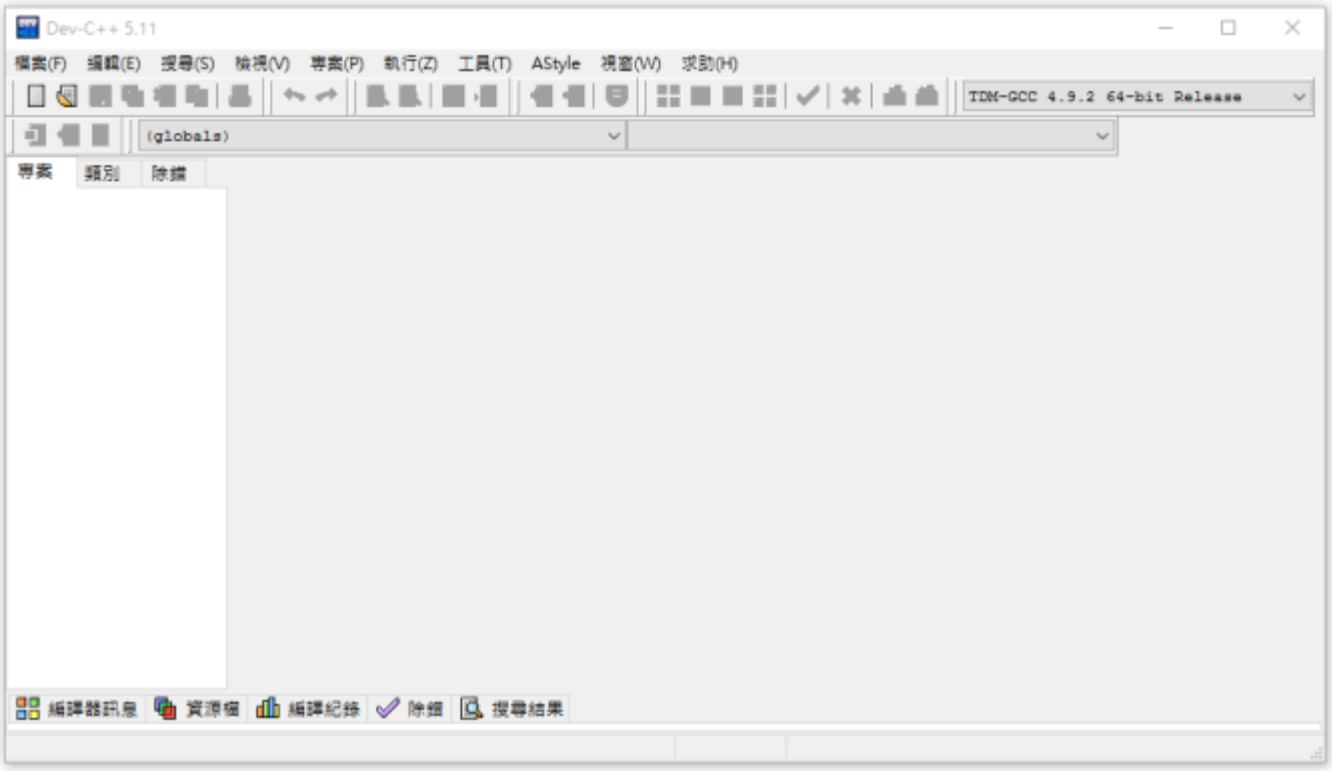

Fig. 16: Dev-<nowiki

完成Dev-C++的安裝後，接下來請依照以下的說明，使用Dev-C++來完成Example 1的程式碼撰寫、編譯與執行等動作：

- **步驟1：撰寫程式碼**

啟動Dev-C++後，請先在選單中選取「檔案>開新檔案>原始碼」，以建立一個新的原始程式，請參考figure 17。你也可以在Dev-C++的工具列上，直接以滑鼠點擊圖示，或者是使用Ctrl+N快速鍵，來建立一個新的原始程式，請參考figure 18。

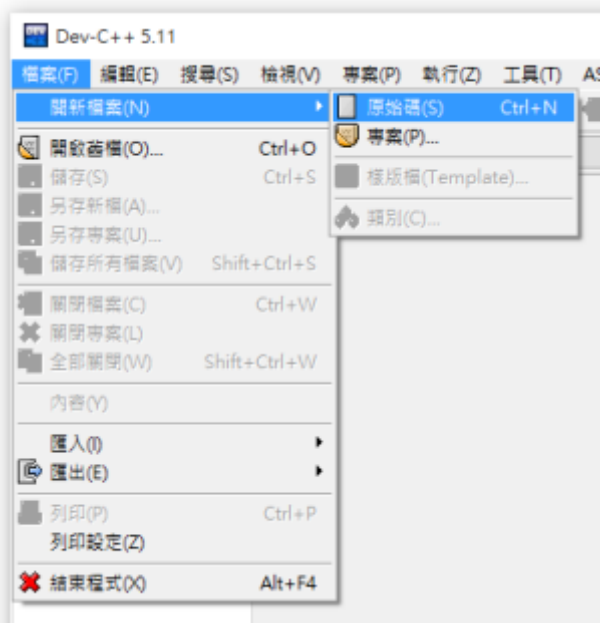

Fig. 17: 從選單中選取建立新的原始碼檔案。

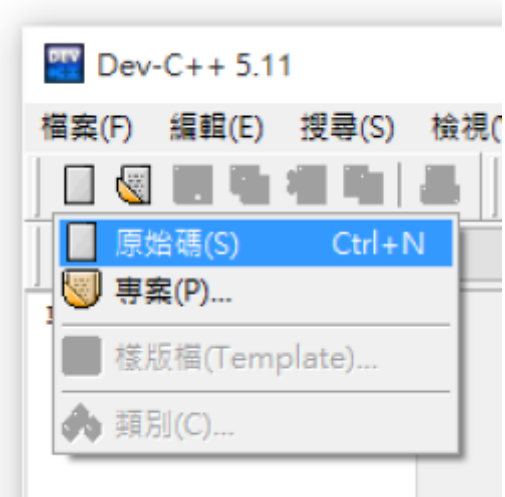

Fig. 18: 從工具列中選取建立新的原始碼檔案。

完成上述動作後，將會在畫面中央得到一個新增的「新文件」，如figure 19所示，請在此處將Example 1的程式碼加以輸入。

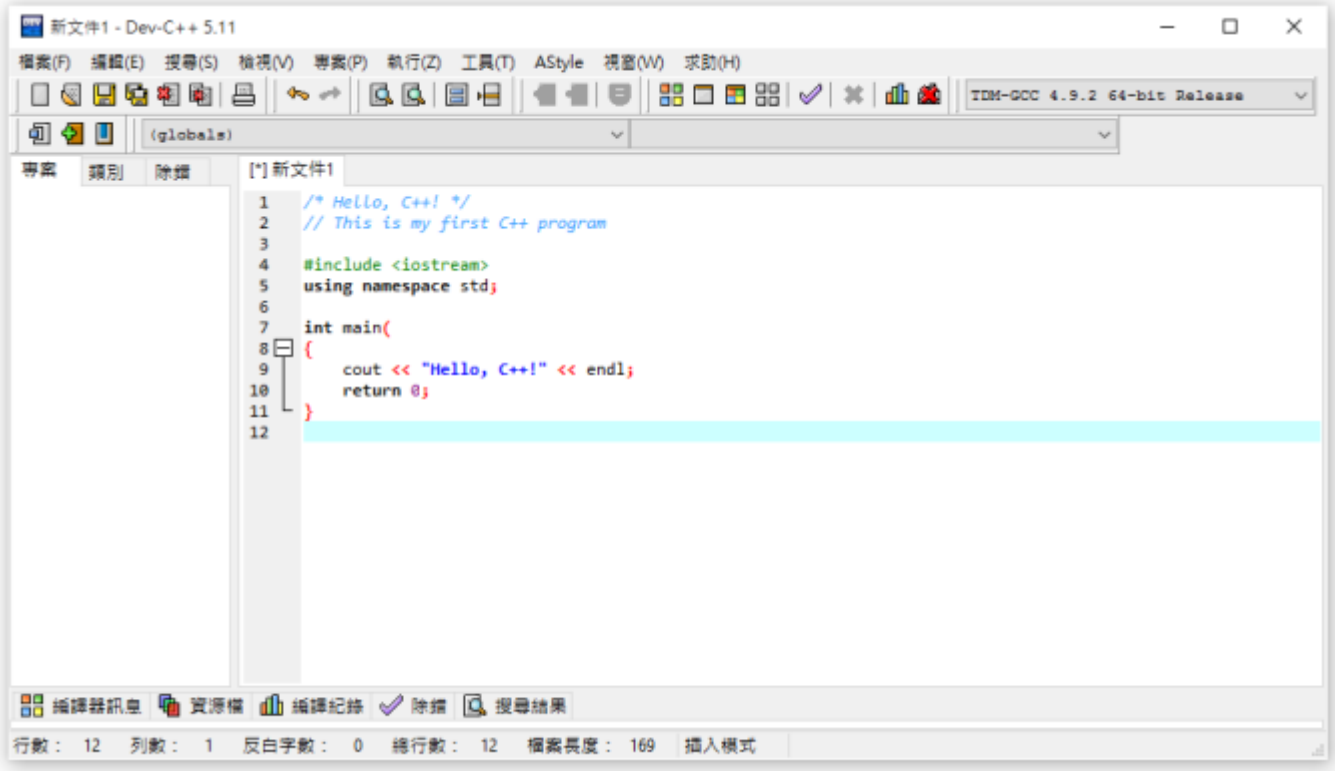

Fig. 19: 在Dev-<nowiki

- **步驟2：程式碼編譯**

當你將Example 1的原始程式碼撰寫完成後，就可以使用編譯器將其轉換為可執行檔，請在選單中選取「執行>編譯」，或是使用F9快速鍵進行編譯，如figure 20所示。

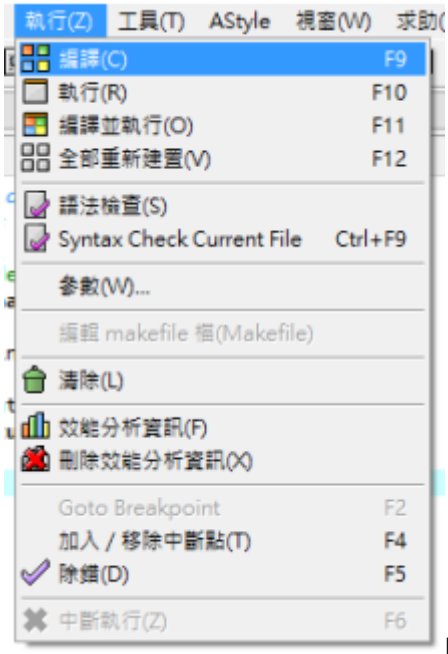

Fig. 20: 從選單中執行編譯。

在第一次進行編譯前Dev-C++將會要求你先將原始程式碼存檔，如figure 21所示。請在此處選取你所偏好的目錄及檔案名稱（以Example 1為例，我們所使用的檔名為hello.cpp其中cpp為C++原始程式預設

所使用的副檔名），完成後請按下「存檔」按鈕。後續Dev-C++就會開始進行鍵進行編譯。

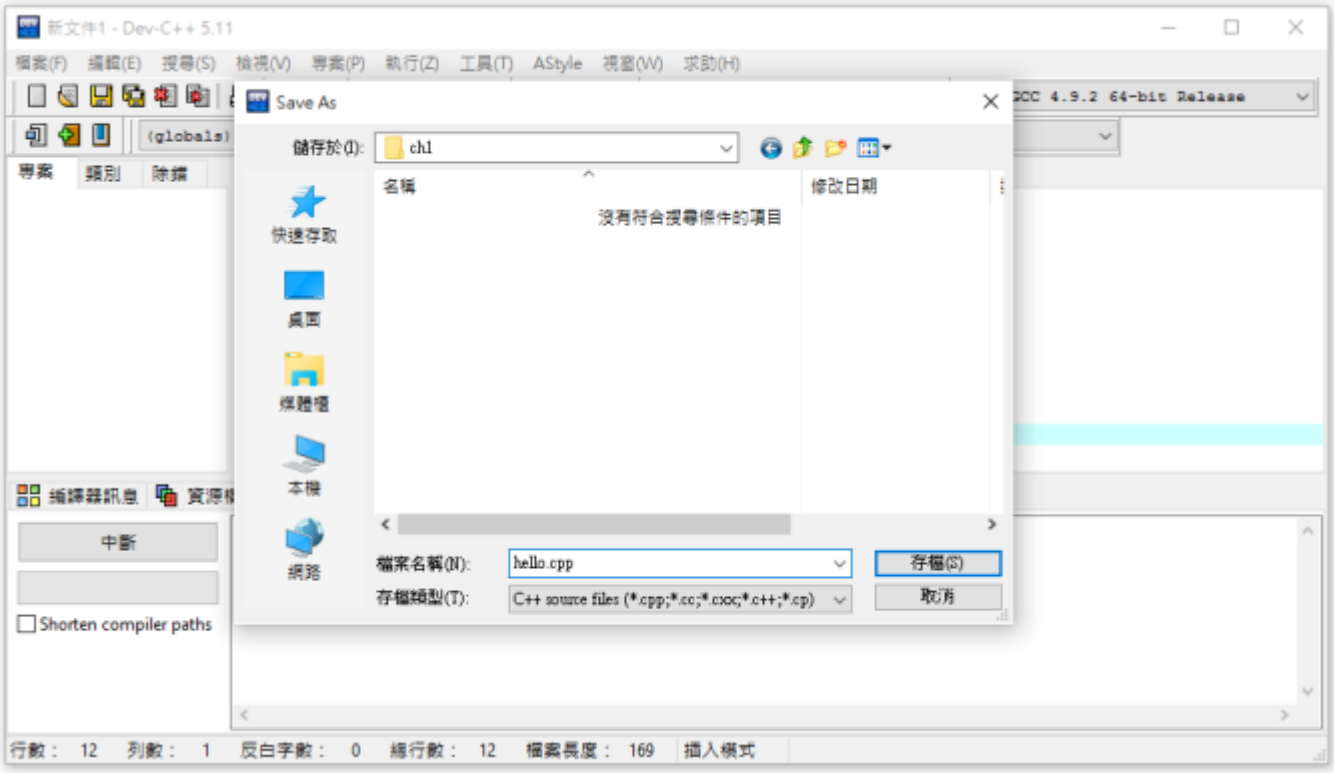

Fig. 21: 在編譯前先進行原始程式的存檔。

原始程式碼的編譯結果，將會顯示在程式碼下方的「編譯記錄」區域，如figure 22所示。

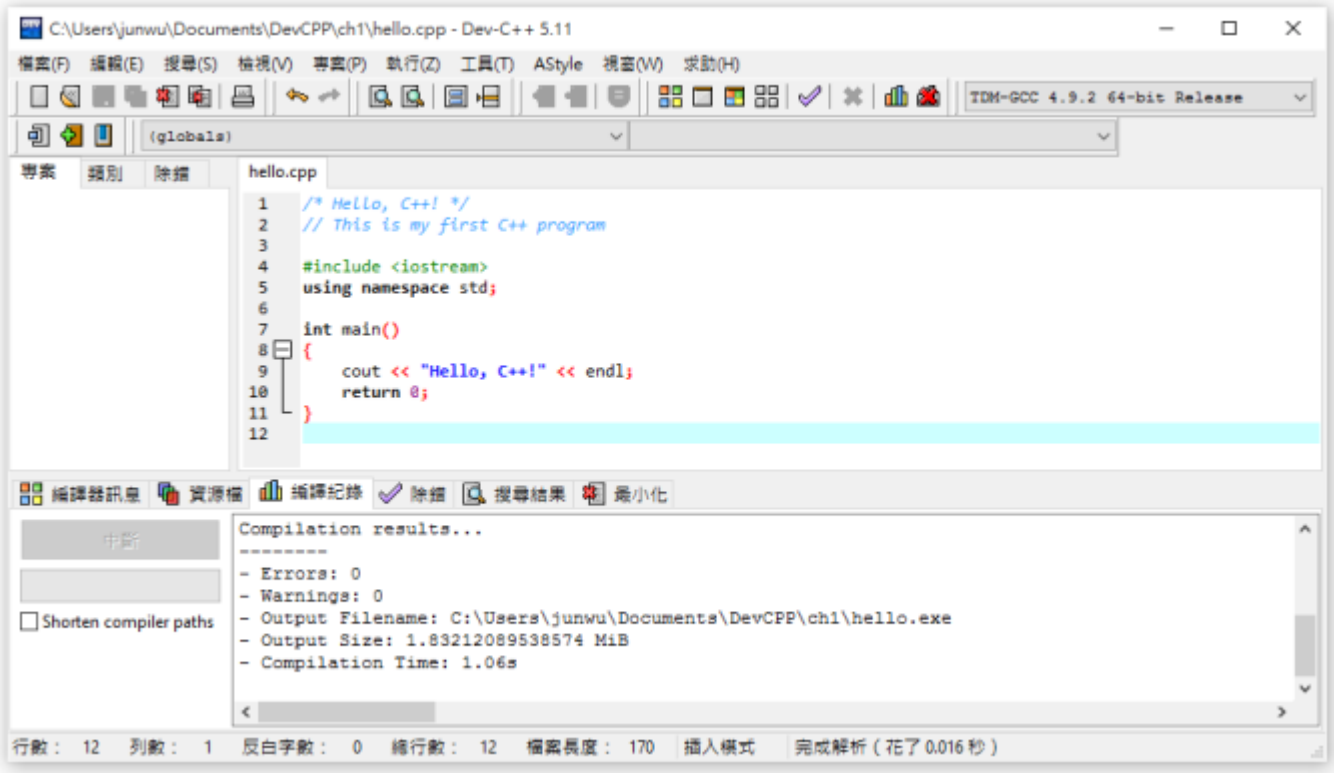

Fig. 22: 編譯結果顯示於程式碼編輯區下方的「編譯記錄」區。

每次進行程式碼的編譯時，Dev-C++都會將編譯結果是否正確，顯示在程式編輯區下方的「編譯記錄」區。以此處所編譯的hello.cpp為例，我們將其「編譯記錄」區的記錄顯示如下：

```
Compiling single file...
--------
- Filename: C:\Users\junwu\Documents\DevCPP\ch1\hello.cpp
- Compiler Name: TDM-GCC 4.9.2 64-bit Release

Processing <nowiki>C++</nowiki> source file...
--------
- <nowiki>C++</nowiki> Compiler: C:\Program Files (x86)\Dev-
Cpp\MinGW64\bin\g++.exe
- Command: g++.exe "C:\Users\junwu\Documents\DevCPP\ch1\hello.cpp" -o
"C:\Users\junwu\Documents\DevCPP\ch1\hello.exe"  -I"C:\Program Files
(x86)\Dev-Cpp\MinGW64\include" -I"C:\Program Files (x86)\Dev-
Cpp\MinGW64\x86_64-w64-mingw32\include" -I"C:\Program Files (x86)\Dev-
Cpp\MinGW64\lib\gcc\x86_64-w64-mingw32\4.9.2\include" -I"C:\Program Files
(x86)\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-
mingw32\4.9.2\include\<nowiki>C++</nowiki>" -L"C:\Program Files (x86)\Dev-
Cpp\MinGW64\lib" -L"C:\Program Files (x86)\Dev-Cpp\MinGW64\x86_64-w64-
mingw32\lib" -static-libgcc

Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\junwu\Documents\DevCPP\ch1\hello.exe
- Output Size: 1.83212089538574 MiB
- Compilation Time: 1.06s
```

這裡所顯示的資訊包含了Dev-C++此次所要編譯的檔案以及其所使用的編譯器，請參考第3行與第4行[25]；至於第9行則是此次進行編譯的指令，雖然看起來有點複雜與冗長，但是這就是使用IDE整合開發環境的好處，因為你可以完全不必理會這些既複雜又冗長指令。對於讀者而言，此時最重要的應該是第13與第14行的錯誤與警告數目。一個正確的程式，應該是不會有任何的錯誤與警告的。

- **步驟3：執行程式**

一旦確認程式碼的編譯沒有任何錯誤後，就會得到一個檔名為hello.exe的可執行檔。請在選單中選取「執行>執行」或是使用F10快速鍵，就可以執行這個編譯完成的hello.exe可執行檔，如figure 23所示。

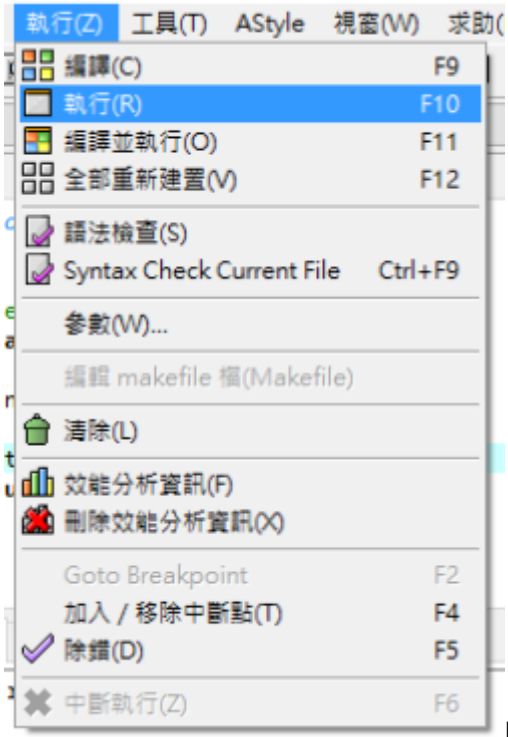

Fig. 23: 從選單選取「執行

對Dev-C++下達了執行的命令後，Dev-C++會幫我們啟動一個「命令提示字元」的應用程式（也就是Microsoft Windows系統的Console環境），並在此程式中執行我們所編譯完成的hello.exe可執行檔，其執行畫面如figure 24所示。你可以看到此程式輸出了「Hello, C++!」字串，並在按下任意鍵後結束。

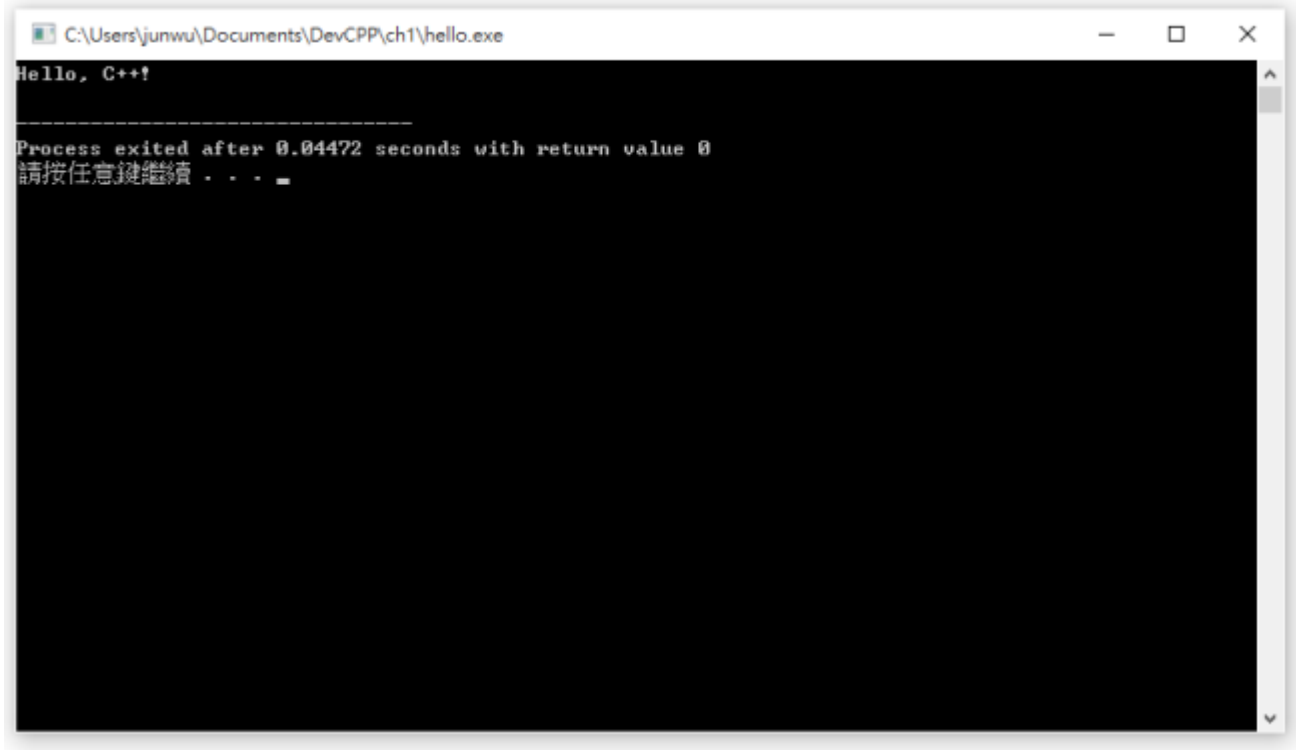

Fig. 24: Dev-<nowiki

我們已經使用Dev-C++示範了Example 1的hello.cpp程式的開發與執行過程，建議讀者依照前述的步驟實際進行一遍，以掌握Dev-C++的使用方式。本書後續的程式範例，也都可以使用Dev-C++來進行開發，你可以下載這些程式範例的原始程式，並使用Dev-C++來載入這些程式[26]，並加以測試其執行結果。最後，

我們針對Dev-C++時常會用到的「編譯」、「執行」、「編譯並執行」以及「全部重新建置」等常用的功能，特別為讀者歸納其操作方法：

功能	選單	快速鍵	工具列圖示	備註
編譯	「執行>編譯」	F9		將原始程式加以編譯以產生可執行檔。
執行	「執行>執行」	F10		執行可執行檔。
編譯並執行	「執行>編譯並執行」	F11		將原始程式加以編譯以產生可執行檔，並且加以執行。
全部重新建置	「執行>全部重新建置」	F12		不論原始程式是否修改過，強制將其加以編譯以產生可執行檔。

Tab. 1: Dev-<nowiki

2.4 程式碼說明

本節將就Example 1的hello.cpp程式碼進行說明，為便利起見hello.cpp的原始程式再次列示如下：

```
/* Hello, C++! */
// This is my first C++ program

#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, C++!" << endl;
    return 0;
}
```

2.4.1 程式基本構成元素：函式與敘述

每一個C++語言的程式，都是由一個或多個函式（Function）[27]所組成，而在每個函式中又包含有一個或多個敘述（Statement）。我們可以把函式視為是C++程式的架構，至於敘述則是用以定義程式所應執行的特定動作。請參考figure 25，一個典型的C++程式是由n個函式所組成（n≥1），其中第i個函式又包含有ni個敘述。請注意figure 25是使用一組對稱的大括號，來將函式所包含有的敘述包裹起來，這也正是C++語言關於函式的語法規定。

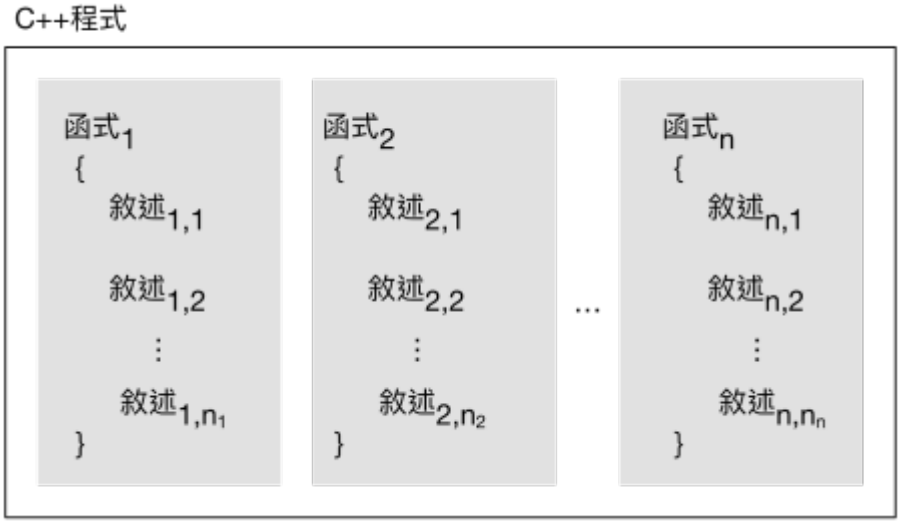

Fig. 25: <nowiki

當然在真實的程式碼中，我們並不會看到如「函式$_1$」、「函式$_2$」等名稱，figure 25只是為了向你解釋C++程式的組成元素而已。在C++語言的程式中，每個函式都必須具備一個函式名稱（Function Name），例如在Example 1的hello.cpp程式中，其第7-11行的程式碼就是一個名為「main」的函式。請注意，為了方便起見，本書後續將使用「fun()」代表一個名為「fun」的函式，而不再以「名稱為fun的函式」加以表示。

Example 1的hello.cpp是一個架構最簡單的C++程式，其中僅包含有一個具有兩個敘述的函式，也就是第7-11行的main()函式。現在，讓我們以figure 25的方法，將Example 1的hello.cpp的架構繪製於figure 26。如果你仔細對照hello.cpp的程式碼，應該會發現figure 25函式的部份，還缺少了第7行開頭處的「int」定義。其實這個「int」是main()函式執行結束時，所需要傳回的資料之型態定義。

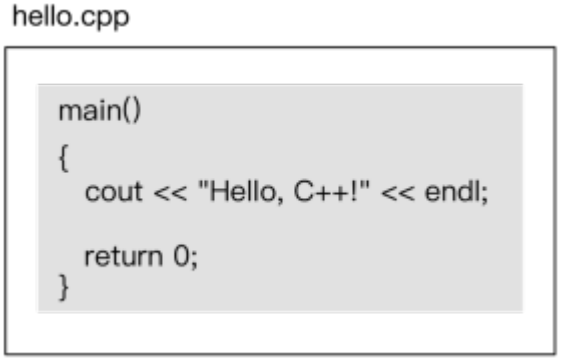

Fig. 26: hello.cpp程式架構圖

至於定義在函式內部的敘述（也就是在其大括號內的程式碼），就是我們希望程式進行的動作；當函式被執行時，這些敘述將會依序逐行加以執行。因此，C++語言的程式設計，就是將程式定義為若干個函式，並使用敘述來定義各個函式所欲完成的功能。若要成為一位專業的C++程式設計師，就必須熟悉各種敘述的使用方式，並正確地使用它們來完成特定的應用功能設計。在此先讓我們大致瞭解一下C++語言提供了哪些種類的敘述可供我們使用:

- 宣告敘述（Declaration Statement）：宣告在程式中所要使用的識別字，必須使用分號「；」做為其結尾。在絕大多數的情況下，宣告敘述是用於變數的定義。
- 運算敘述（Expression Statement）：進行算術與邏輯等各式運算的敘述，必須使用分號「；」做為其結尾。
- 條件敘述（Conditional Statement）：又稱為選擇敘述（Selection Statement）：是可以依據特定條件改變程式執行動線的敘述。
- 重複敘述（Iteration Statement）：讓特定程式碼可以反覆執行的敘述。
- 跳躍敘述（Jump Statement）：可用以改變程式執行的動線的敘述，必須使用分號「；」做為其結尾。
- 複合敘述（Compund Statement）：以一組大括號包裹的多個敘述，以程式基本構成元素：函式與敘述小節所介紹的main()函式為例，其後所接續的一組大括號就是一個包含有兩個敘述的複合敘述。由於包含條件敘述、重複敘述等多個敘述，都允許我們將原本單一的敘述，改寫為包含多個敘述的

複合敘述，因此我們將在本書後續內容中，適時地提供更多的相關說明。

本節後續將會為你詳細說明，在hello.cpp中的main()函式以及相關敘述的作用與意義。

2.4.2 程式進入點（Entry Point）

一個C++語言的程式在執行時，會由作業系統將其載入到記憶體中，並從一個特定的函式開始執行。這個特定的函式被稱為是程式的「進入點（Entry Point）」，在該函式中所包含的敘述將會從左大括號處開始逐一地依序執行，直到遇到該函式用以標記結束的右大括號為止。要注意的是，一旦結束了main()函式的執行，程式也會隨之結束。

雖然我們在前一小節已經說明過，一個C++語言的程式可以有一個或多個函式，但用以擔任進入點的函式並不是隨便指定一個就可以，而是有特定的規定 — 其函式名稱必須為「main」，以Example 1的hello.cpp為例，雖然它僅擁有一個函式，但其名稱就是「main」，因此這個從第7行開始到第11行的main()函式就會被視為是程式的進入點，在執行時會將在第8行的左大括號 { 後的程式碼逐一的加以執行，直到第11行的右大括號 } 為止，main()函式與程式的執行都會結束。

做為本書的第二章，在此我們僅需要暫時瞭解main()函式是C++語言程式的進入點，程式將會從其後的左大括號開始執行，直到右大括號為止，逐一將其中所包含的程式碼敘述加以執行。因此，設計一個C++語言的程式，就是將欲交由電腦執行的功能，依據C++語言的語法規則，將其寫在main()函式的大括號內。我們將在接下來的兩個小節，為讀者說明hello.cpp的main()函式裡的兩行敘述的意義與作用。

2.4.3 印出字串的cout物件

首先要為讀者說明的是在hello.cpp中的第9行程式碼：

```
cout << "Hello, <nowiki>C++</nowiki>!" << endl;
```

回顧上一小節的說明，由於這行程式是程式進入點（也就是main()函式）裡的第一行程式碼，因此在程式執行時，這一行程式碼將會率先被加以執行。請參考figure 27，這行程式碼使用「cout」來將「Hello, C++!」字串輸出到螢幕上，並將游標移到下一行。

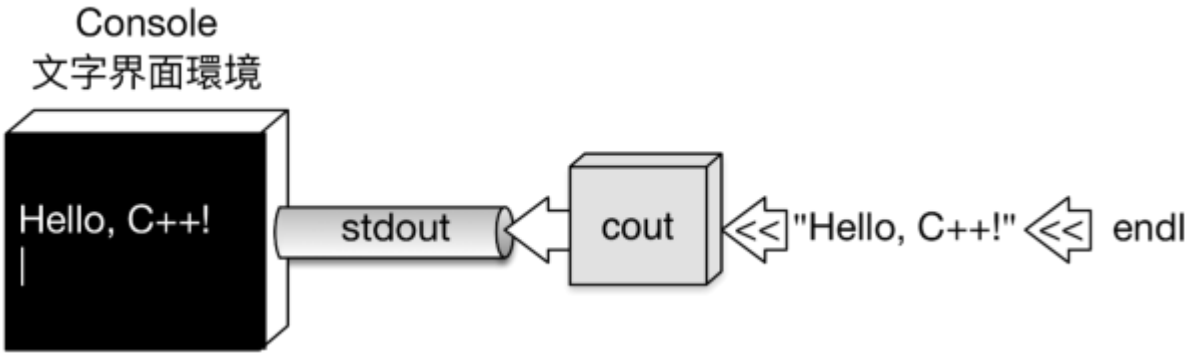

Fig. 27: cout透過stdout將資料輸出到Console

具體來說，cout是一個物件（object）[28]顯示在Console模式的文字界面環境。在使用上是透過「«」運算子[29]，來將所要輸出的資料交給cout物件，然後cout就會幫我們把資料經由stdout渠道輸出到Console，如

果有多項資料要輸出，還可以使用多個「«」來將資料串接給cout輸出，以下的例子都可以將Hello, C++!輸出到螢幕上：

```
cout << "Hello, <nowiki>C++</nowiki>!";
```

或是

```
cout << "Hello, " << "<nowiki>C++</nowiki>!";
```

從上面這些例子你可以發現，「«」運算子就好像是在指定資料流動的方向，把“Hello, C++!”或是“Hello, “與 ” C++!”流向cout所連接的stdout來顯示在Console文字界面環境中。但細心的讀者可能會問：「這些例子所輸出的不是“Hello, C++!”嗎？為何顯示時只看到Hello, C++而沒有看到雙引號？」其實使用雙引號所包裹的文字內容，被稱為「字串String，cout會將流向它的字串內容原封不動地顯示在螢幕上，並不包含用以標註字串的雙引號。換句話說，雙引號的存在只是要將字串內容加以包裹標註，而所謂的字串內容指得是一些字元Character的組合，例如以下是正確定義字串內容的例子：

```
"Hello" // 字串內容可以由大小寫英文字母組成
"123" // 字串內容不一定是由英文字母組成，使用數字也可以
"@_@" // 字串內容也可以由符號組成
"Hello @_@ 123" // 混合大小寫英文字母、數字與符號也可以
```

但是以下是錯誤的例子：

```
Hello, <nowiki>C++</nowiki>!    // 少了雙引號標註，並不是正確的字串
"Hello, <nowiki>C++</nowiki>!   // 缺乏右方的雙引號標註，所以不是正確的字串
```

除了“Hello, C++!”字串外，在hello.cpp的第9行中，還使用了第二個「«」運算子來將endl串接在後面加以輸出。但是此處的endl並不是一個字串（它沒有使用雙引號包裹標註），而是一個特別賦與的意義 — End of Line也就是「換行」[30]的意思，因此cout « “Hello, C++!” « endl;的作用就是輸出Hello, C++!字串內容並且加以換行。

最後要提醒讀者注意cout « “Hello, C++!” « endl;這行程式碼正如我們在程式基本構成元素：函式與敘述小節所說明過的，它其實是一個敘述Statement而且是一個運算敘述Expression Statement必須使用分號「；」做為其結尾。可千萬不要忘了這一個小小的分號，如果少了它，你所撰寫的程式就會因為語法上的不正確，而無法通過編譯。

2.4.4 return敘述

在前面的幾個小節中，我們已經反覆地說明過在main()函式執行時，從其後的左大括號開始，其中所包含的敘述將會被逐一地執行，直到遇到對應的右大括號為止。不過如果在右大括號前，就先遇到「return敘述」，那麼程式就會提前結束並傳回一個代表程式的結束狀況的整數值給作業系統。例如Example 1的hello.cpp程式，其第10行就是在main()函式的右大括號前的「return 0;」敘述。這行程式碼讓hello.cpp程

式在執行到main()函式的右大括號前，就提前完成main()函式（也就是程式進入點）的執行：當然，程式的執行也會隨之結束。除此之外，return 0;還會在程式結束前將代表「正常結束」的整數0傳回給作業系統，以便讓作業系統能夠知道程式是在正常的情況下結束[31)]。

由於return敘述改變了程式原本執行的動線（原本應該要在遇到main()函式的右大括號時，才會結束程式的執行），因此return敘述是一個跳躍敘述（Jump Statement），必須要使用「;」做為結束。

要特別注意的是，return敘述並不是必要的，省略return敘述的程式仍然能正確地執行。

2.4.5 函式標頭檔與命名空間

在Example 1的hello.cpp的程式碼中，還有以下這兩行：

```
#include <iostream>
using namespace std;
```

我們先從其中的「#include <iostream>」開始說明：這行程式碼是用來載入iostream這個檔案，其中包含有與輸入、輸出相關的定義。你可以試著將這行移除後再加以編譯，是不是發現許多編譯的錯誤？原因是包含cout與endl都是定義在iostream這個檔案之中，如果將「#include<iostream>」從程式中移除，那麼在進行編譯時就會發生無法識別它們的問題。其實，許多C++語言所提供的功能，都是所謂的C++標準函式庫（C++ Standard Library）的一部份，被事先定義在iostream等標頭檔（Header File）中。絕大多數的C++程式都會需要使用到這個標準函式庫，我們必須使用「#include < >」來將相關的標頭檔加以載入，才能在程式中使用所需的功能。做為C++語言的初學者，此時你只要先暫時知道如果要使用cout來進行資料的輸出時，一定要先載入iostream這個檔案即可。

要注意的是，「#include <iostream>」這行程式碼後面並沒有使用「;」做為結束。因為這行程式碼並不是所謂的敘述，而是一個「前置處理指令（Preprocessor Directive）」，它是在程式被編譯前，由「前置處理器（Preprocessor）」先加以處理的指令。具體來說，它會去系統內尋找iostream檔案，並將其內容在原本的「#include <iostream>」處，以類似複製貼上的概念，代換了原本的「#include <iostream>」。

沒有副檔名的標頭檔
為識別起見，過去C語言規定標頭檔應使用.h做為其副檔名；到了C++語言，一開始這個習慣也是被延用，所以早期的C++語言提供了許多副檔名為.h或是.hpp的檔案供我們載入到程式中使用。所以在早期開發的C++程式中，你可能會發現部份檔案的副檔名為.h或.hpp。不過現在的做法已經不再使用副檔名。所以「#include <iostream>」所載入的就是「iostream」這個檔案，你可以在作業系統中預設存放C++語言標頭檔的目錄中，找到iostream以及其它眾多的標頭檔，例如在Linux系統中，可以在「/usr/include」或是「/usr/include/C++」目錄中找到這些檔案。

至於「using namespace std;」則是一行敘述，必須使用「;」做為結尾。這行敘述的作用是定義所謂的「命

名空間（Namespace）：命名空間是C++語言用來管理識別字（Identifier）[32] 的一種方法。包含C++標準函式庫在內，許多程式都可以自行定義所需的識別字，因此也就可能會發生不同程式使用了相同識別字的衝突問題。透過命名空間的管理，就可以避免此一情形，例如C++標準函式庫使用「std」這個命名空間來管理其識別字，因此以「cout」這個識別字為例，其全名為「std::cout」。若是我們自行開發的程式中，也想使用「cout」這個識別字，只要使用不同的命名空間，就可以避免衝突，例如以「my」做為我們的命名空間，那麼「my::cout」與「std::cout」就可以區分為不同的識別字；我們將此處的「my::」與「std::」稱之為識別字的前綴（Prefix）。

現在請你先將hello.cpp中的「using namespace std;」這行程式碼暫時移除，然後試著再進行編譯，是不是發現編譯器會發出不認識「cout」的錯誤訊息呢？此時，我們只要將hello.cpp中的「cout」與「endl」改為「std::cout」與「std::endl」就可以讓編譯器瞭解到我們要使用的是定義在「std」這個命名空間裡的「cout」與「endl」這兩個識別字。如果我們所撰寫的程式，常常會使用到「cout」與「endl」這兩個識別字，就必須在每次使用時都在其前方加上「std::」前綴來表明所使用的是定義在「std」命名空間的「cout」與「endl」，如此一來編譯器才能正確地完成程式碼的編譯。為了便利起見，我們可以使用「using namespace std;」來告訴編譯器：每當遇到不認識的識別字時，可以試著在「std」這個命名空間中找尋相關的定義。如此一來，我們就不需要每次都在「cout」與「endl」的前面冠以「std::」，編譯器仍然能正確地進行編譯。

2.4.6 註解（Comment）

最後，讓我們來關注在hello.cpp的程式碼開頭處（也就是第1行及第2行程式碼）的以下內容：

```
/* Hello, <nowiki>C++</nowiki>! */
// This is my first <nowiki>C++</nowiki> program
```

其實這兩行是所謂的「註解（comment）」，編譯器在進行編譯時都會略過此一部份，對於程式的執行並沒有任何作用。在C++語言中註解使用的方式有以下兩種方式：

- 1.以「//」開頭到行尾的部份，皆視為註解。此種方式的註解可以放在每一行的開頭，也可以放在行中的任意位置，例如：

```
// This is my first <nowiki>C++</nowiki> program
int main() // 這是<nowiki>C++</nowiki>語言的進入點
```

- 2.以「/*」開頭到「*/」的部份也會被視為是程式的註解。使用這種方式可將多行的內容都視為註解，例如：

```
/* 這是單行的註解 */

/* 這種方式也可以用在多行的註解
  在需要較多說明的時候
就可以使用這種方式
*/
```

雖然註解在執行時並沒有作用，不過我們通常會使用註解來提供一些關於程式的說明，其內容不外乎版本、版權宣告、作者資訊、撰寫日期及程式碼的說明等，例如：

```
/*
  Filename: hello.cpp
  Author: Jun Wu
  Date: April 25th, 2018
 */

// This is my first <nowiki>C++</nowiki> program
```

註解並不是只能做為程式碼的說明之用，有時我們還會暫時把可能有問題的程式碼加以註解，讓其失去作用。例如我們可以將hello.cpp的第9行程式碼加以註解，然後執行看看缺少這行程式碼對於程式執行的影響為何？有時也可以將特定的程式碼加以註解，以便找出程式可能的錯誤，這在程式設計上是常用的一種除錯（debug）方式。

除錯（Debug）是衡量一個程式設計師能力的重要指標，優秀的程式設計失必須具備能夠找出程式碼的錯誤，以及修正錯誤的能力。一般而言，我們將程式碼中的錯誤稱之為「蟲（bug）」，除錯（除蟲，Debug）意味者將程式碼中錯誤的部份除去。對於初學者來說，學習並解讀編譯器所提供的錯誤訊息，是養成除錯能力的主要方法。要增進個人的除錯能力，除了必須學習程式語言的語法規則，還必須累積一定程度的程式開發經驗，從錯誤中學習正確的程式碼語法是唯一的方法。

2.5 本章內容回顧

至此我們已經為你簡要說明了在hello.cpp中每一行程式碼的意義，請讀者務必先依照本章的內容準備好C++語言的程式開發環境，才能夠配合本書後續的章節內容，開始你的C++語言程式設計學習之旅。以下我們為讀者彙整了本章相關的重點：

- C++98（C++語言的第一個國際標準（ISO/IEC 14882:1998））
- C++03（C++語言的第二個國際標準（ISO/IEC 14882:2003））
- C++11（C++語言的第三個國際標準（ISO/IEC 14882:2011））
- C++14（C++語言的第四個國際標準（ISO/IEC 14882:2014））
- C++17（C++語言的第五個國際標準（ISO/IEC 14882:2017））
- Code Editor（程式碼編輯器，用以撰寫程式的工具軟體。
- Text Editor（文字編輯器，一般用以撰寫純文字檔亦可用以撰寫程式碼。
- Compiler（編譯器，用以將原始程式碼轉換為可執行的程式檔案。
- IDE（整合開發環境（Integrated Development Environment））可以在單一環境中進行原始程式碼的撰寫、編譯與執行等功能的軟體。
- main()（C++語言的程式進入點（Entry Point））所有程式皆從此處開始執行。
- 敘述（Statement））用以定義程式所欲進行的動作。
- cout（透過stdout將資料以「«」來輸出到Console的文字界面環境。
- string（字串，由一組雙引號所包裹標記的字元（Character）組合。
- endl（行結尾（end of line））表示換行。
- return 0;（結束main()函式的執行，並傳回代表程式正常結束的整數0。
- #include <iostream>（載入iostream標頭檔，以便使用包含cout在內的輸出與輸入功能。
- using namespace std;（使用（std）命名空間。程式中如果缺少這行程式碼，就必須使用（std::）才能使用定義在std命名空間中的識別字，例如（std::cout））

- ：多行註解。程式設計師通常使用註解來進行程式碼的說明，或是提供版權、作者等相關資訊。有時亦可利用註解進行程式碼的除錯。
- 從「/*」開始到「*/」結束的範圍，都將會被視為註解。
- *：單行註解。從「」開始到同一行的行末都是註解。*

---

[1] 讀音為C plus plus在台灣一般的唸法是混和中英文唸做「西加加」。
[2] 比雅尼．史特勞斯特魯普(Bjarne Stroustrup)英國劍橋大學計算機科學博士，現為美國哥倫比亞大學客座教授以及摩根史丹利技術部門董事總經理，被資訊領域尊稱為C++之父。
[3] 組合語言assembly language是接近處理器指令的低階程式語言，以其執行的效率著稱。
[4] BCPL全名為Basic Combined Programming Language是由馬丁．里察德Martin Richards於1966年所發表的程式語言，是第一個使用一組大括號（{}）做為程式區塊的程式語言。其後續所推出的B語言即為C語言之前身，因此又有人將BCPL戲稱為Before C Programming Language
[5] Brian Kernighan and Dennis Ritchie, The C Programming Language, 2nd Edition, Prentice Hall, ISBN 0-13-110362-8, 1988.
[6] 艾茲赫爾．戴克斯特拉(Edsger W. Dijkstra)是著名荷蘭電腦科學家，曾獲得1972年圖靈獎(Turing Award)1974年IEEE哈里．古德紀念獎(Harry Goode Memorial Award)以及1989年ACM SIGSE電腦科學教育傑出貢獻獎(Award for Outstanding Contribution to Computer Science Education)等榮譽，在作業系統、演算法、程式語言等領域貢獻卓越。他的著名事蹟包含在作業系統方面所提出的號誌機(Semaphore)同步機制，以及解決了哲學家吃飯問題；在演算法方面，著名的最短路徑演算法(Shortest Path First Algorithm)以及銀行家演算法(Banker's Algorithm)都是由他所提出的；此外，他也是Algol 60程式語言編譯器的作者，並提出了以高階程式語言實現遞迴(Recursion)的方法。
[7] Edsger W. Dijkstra, “The Humble Programmer”, Communications of ACM, Volume 15, Issue 10, pp. 859-866, October 1972.
[8] 軟體危機(Software Crisis)一詞，首見於1968年，北大西洋公約組織(NATO)在西德所召開的會議，詳見Report on a conference sponsored by the NATO SCIENCE COMMITTEE, Garmisch, Germany, October 7-11, 1968.
[9] 軟體工程(Software Engineering)一詞，同樣首見於1968年的NATO會議。針對當時已注意到的軟體危機，軟體工程是希望制定出好的程序方法，讓軟體產業能如傳統製造業或營造業一樣，能夠在有限的時間與預算內，開發出具有高品質的軟體。
[10] 全名為Bjarne Stroustrup音譯為「比啊倪 史哆.史抓」），英國劍橋大學計算機科學博士，現為美國哥倫比亞大學客座教授以及摩根史丹利技術部門董事總經理，被資訊領域尊稱為C++之父。
[11] Bell Laboratories 貝爾實驗室）同時也是Unix作業系統與C語言的發源地。
[12] 通用程式語言General-Purpose Programming Language係指無特定應用目的限制，可廣泛應用於各種應用領域。相對的，特殊用途程式語言Special-Purpose Programming Language則是專門為特定應用領域所設計，例如應用於資料庫系統的SQL語言，以及專門為排版而設計的LaTex語言皆為特殊用途程式語言的例子。
[13] <nowiki>C++語言的命名最先是由Rick Mascitti所提出。由於在C語言中，++為遞增運算子，因此C++可以視為是將C的數值累加1。此命名巧妙地結合了C語言的增強版本或是下一個版本的意涵</nowiki>
[14] 依據Bruce Eckel所著的Thinking in C++一書C語言與C++語言所開發的程式，其效能的差距約在5%以內。
[15] 不論你使用何種作業系統，其通常都會預先安裝有文字編輯器可供你使用，例如Microsoft Windows系統中的記事本或是在Linux系統中的vi或emacs等。
[16] 關於vi的操作方法已超出本書的範圍，請有興趣的讀者自行參閱相關的書籍或教學網站，在此不予贅述。
[17] Code::Blocks官方網站的網址為http://codeblocks.org
[18] Eclipse官方網站的網址為http://www.eclipse.org
[19] Netbeans官方網站的網址為https://netbeans.org
[20] Sublime Text官方網站的網址為https://www.sublimetext.com
[21] Atom官方網站的網址為https://atom.io
[22] Brackets官方網站的網址為http://brackets.io
[23] Dev-C++是基於GNU通用公眾授權條款General Public License（GPL）的自由軟體，任何人都可以免費、

自由地使用。

[24] 檔名中的TDM-GCC 4.9.2代表該版本所使用的編譯器版本。

[25] 第4行顯示Dev-C++所使用的編譯器為TDM-GCC 4.9.2，這是在Windows系統上的一個GCC的分支版本。關於GCC可以參閱本書在Linux系統上開發<nowiki>C++</nowiki>程式小節的說明。

[26] 使用Dev-C++選單中的「檔案>開啟舊檔」，就可以載入這些程式範例的原始程式。

[27] Function一詞在數學領域被譯為函數，為區別起見，本書將C++語言中的Function譯為函式。

[28] 做為本書的開頭，此處實在無法和讀者們解釋何謂「物件」，只能請讀者暫時將cout視為是一個軟體元件，我們可以透過它來將資料顯示在螢幕上。至於「物件」與cout更詳細的說明，請分別參考本書後續的說名。簡單來說cout的作用為將資料透過標準輸出渠道standard output stdout[(除了用以輸出資料的標準輸出stdout之外，另外還有用以輸入資料與輸出錯誤訊息的stdin與stderr等標準輸入與輸出渠道。

[29] 例如「3+7」裡的+，就是一個運算子。

[30] 用白話文來說：「這行的輸出結束了，幫我換個行吧！」

[31] 我們也可以在一個C++程式中，透過程式碼來執行其它的程式。此時，程式結束時的傳回值就不是回傳給作業系統，而是傳給執行它的程式

[32] 識別字Identifier係指在程式中具有特定意義的符號組合，例如coutendl等都是識別字。