第 2 章
初探 C++

本章提要

- 2-1 撰寫第一 個 C++ 程式
- 2-2 C++ 程式的組成及語法規則
- 2-3 良好的程式撰寫習慣

2-1 撰寫第一個 C++ 程式

- 開發 C++ 程式的第一個動作，當然就是**寫**程式了，程式寫好後才能依照前一章所述，進行編譯、連結的動作，以產生可執行的程式檔。
  - 2-1-1 程式的編輯與存檔
  - 2-1-2 編譯、連結、執行程式

程式的編輯與存檔

- 不管您要使用整合開發環境或命令列工具, 您都必須先輸入程式的內容, 通常稱之為原始碼 (Source Code), 而儲存原始碼的檔案則稱為原始檔或原始程式檔。請在您的整合開發環境或文字編輯器中輸入以下的程式內容：

程式的編輯與存檔

程式 Ch02-01.cpp 第一個 C++ 程式

```
01 #include<iostream>
02
03 int main()
04 {
05    std::cout << "Hello, 我的第一個 C++ 程式";
06    return 0;                    // 這一行可省略不寫
07 }
```

程式的編輯與存檔

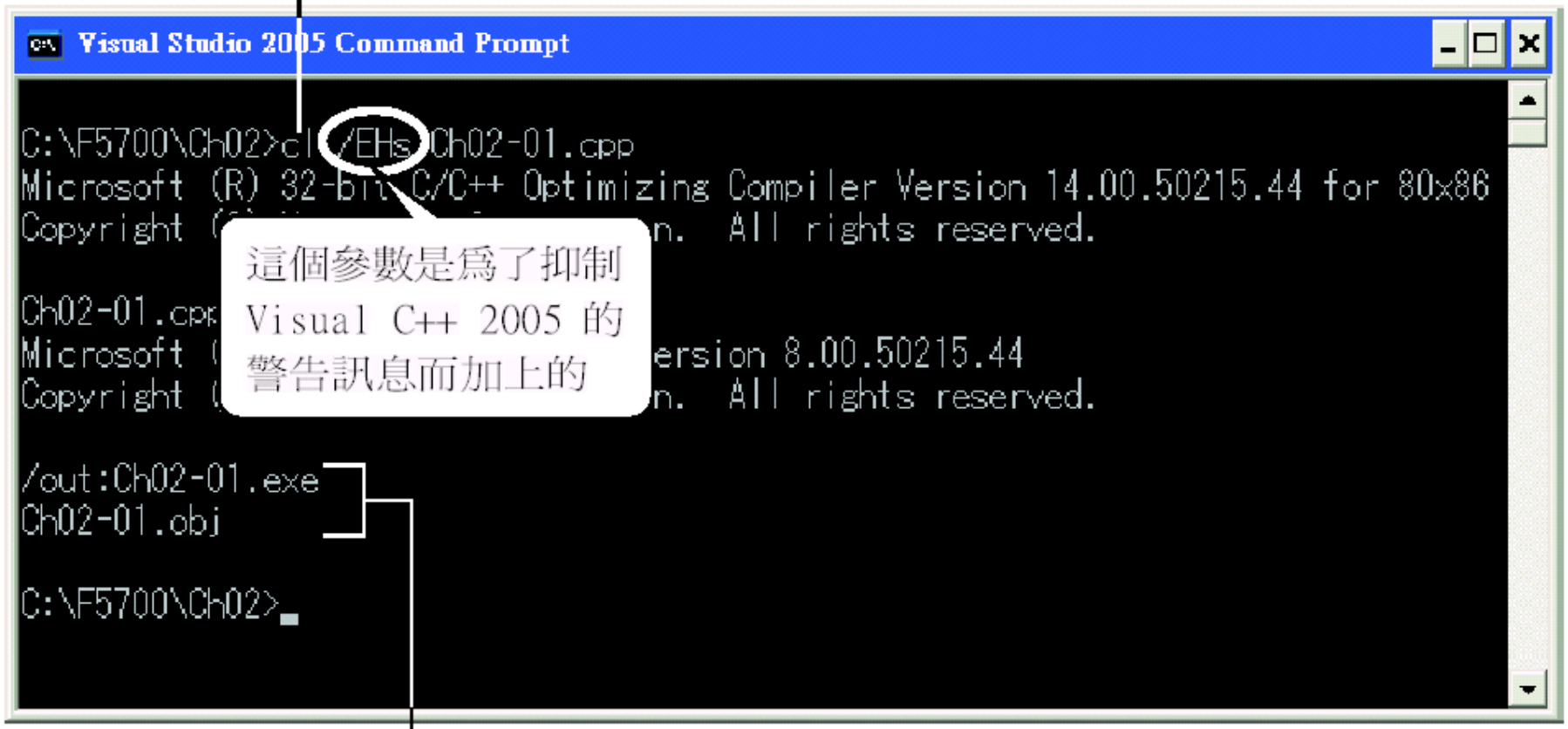
- 請注意, 輸入以上的程式內容時, 必須和書上的**大小寫完全一致**, 否則稍後將無法成功編譯程式。輸入完畢後, 以 Ch02-01.cpp 為檔名存檔, cpp 是目前一般所用的 C++ 原始程式副檔名。接下來即可編譯 / 連結程式, 並看看程式的執行結果。

編譯、連結、執行程式

- 使用命令列工具者, 請在文字模式下執行編譯器, 並在後面加上 C++ 程式檔名 (剛才存的檔名為 Ch02-01.cpp) 即可。目前多數的編譯器工具, 會在編譯成功後自動呼叫連結器進行連結, 以產生可執行檔, 所以您可能只看到幾行簡短的訊息或甚至沒有任何訊息, 就表示程式已完成編譯和連結。例如以下即為使用 Visual C++ 2005 Express 所附的命令列工具程式編譯時, 所出現的訊息：

編譯、連結、執行程式

"cl" 是 Visual C++ 2005 的編譯器執行檔檔名

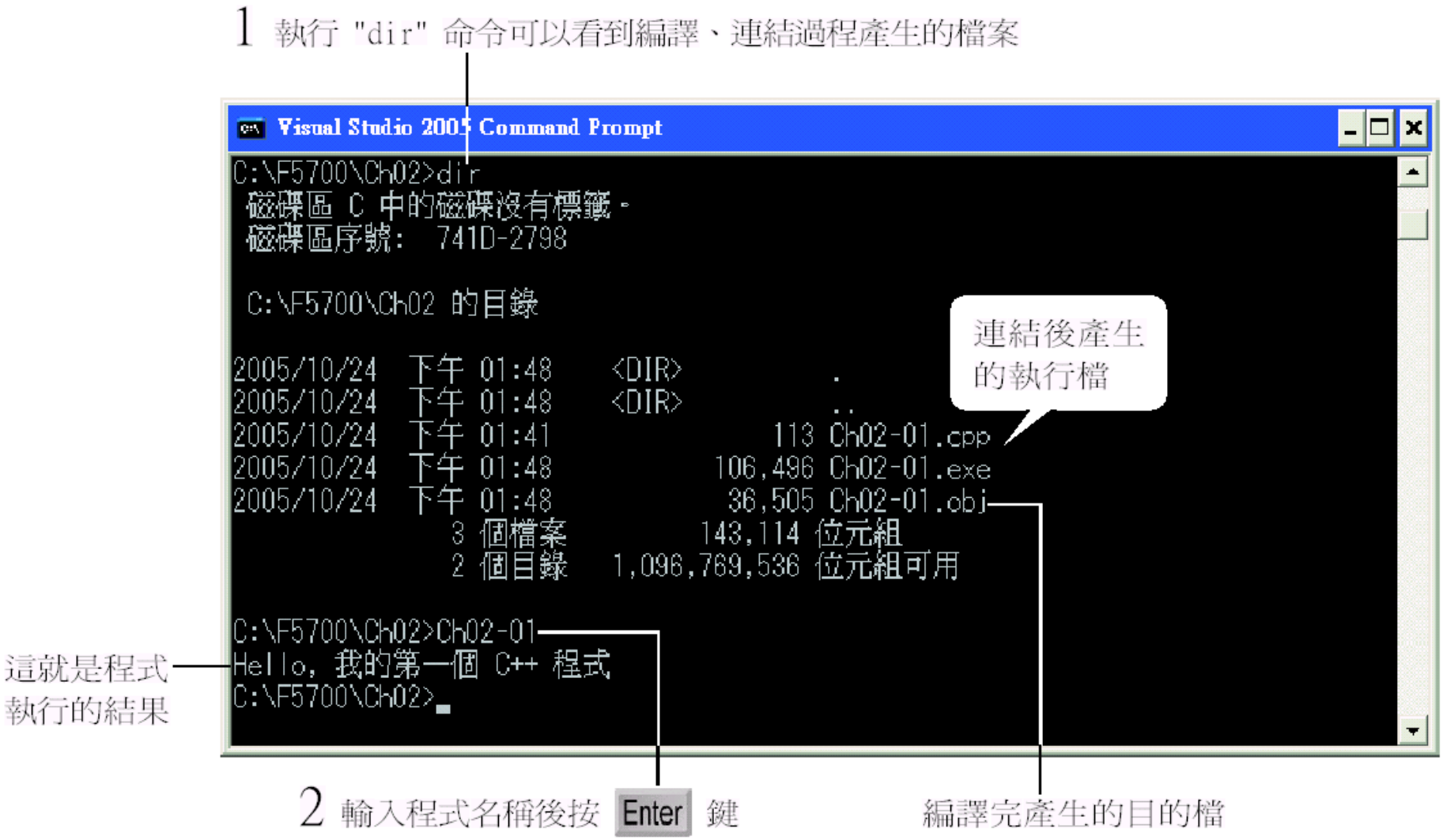


編譯、連結成功所產生的目的檔和執行檔名稱

執行程式

- 編譯 / 連結成功後, 預設建立的執行檔主檔名與程式的主檔名相同, 所以只要在**命令提示字元**下輸入程式的名稱, 即可執行程式：(如下頁)
- 這個程式的執行結果, 就是在螢幕上顯示一行訊息：**Hello, 我的第一個 C++ 程式**。

執行程式


1 執行 "dir" 命令可以看到編譯、連結過程產生的檔案
Visual Studio 2005 Command Prompt
C:\F5700\Ch02>dir
磁碟區 C 中的磁碟沒有標籤。
磁碟區序號: 741D-2798
C:\F5700\Ch02 的目錄
2005/10/24 下午 01:48 <DIR> .
2005/10/24 下午 01:48 <DIR> ..
2005/10/24 下午 01:41 113 Ch02-01.cpp
2005/10/24 下午 01:48 106,496 Ch02-01.exe
2005/10/24 下午 01:48 36,505 Ch02-01.obj
3 個檔案 143,114 位元組
2 個目錄 1,096,769,536 位元組可用
C:\F5700\Ch02>Ch02-01
Hello, 我的第一個 C++ 程式
C:\F5700\Ch02>
連結後產生的執行檔
這就是程式執行的結果
2 輸入程式名稱後按 Enter 鍵
編譯完產生的目的檔

編譯連結出現錯誤訊息

- 如果編譯或連結時, 編譯器顯示一些訊息, 且未產生可執行檔, 表示原始程式內容有問題, 編譯器無法正確解讀以進行編譯動作。此時必須回頭檢查程式, 你可檢查如下的項目：
  - 是否有大小寫與書上範例程式不同的地方, 並請將之更正為與書上相同。請切記, C++ 語言會**區分大小寫**, 也就是說同一個英文字母的大寫和小寫在 C++ 是 2 個不同的符號。

編譯連結出現錯誤訊息

- 是否有遺漏標點符號, 例如分號、括號、括弧等等。
- 是否誤用了全型的標點符號, C++ 語法只接受英文半型符號, 如果您不小心將某個標點符號輸入成中文全型符號, 請將之換回半型。

- 修正問題後再重新編譯 / 連結, 應該就能成功。

2-2
C++ 程式的組成及語法規則

- 看過一個簡單 的 C++ 程式後, 以下我們要來分析一下程式的結構, 並逐行解說各行程式的意義。

基本的程式輪廓

- 我們再回頭看一次程式的內容：

程式 Ch02-01.cpp 第一個 C++ 程式

```
01 #include<iostream>          ———————— 寫程式的準備工作
02
03 int main()
04 {
05    std::cout << "Hello, 我的第一個 C++ 程式";
06    return 0;                // 這一行可省略不寫
07 }
```

這部份是『可執行的程式碼』

基本的程式輪廓

- 程式大概可分為 2 個部份, 第 1 行並不算是 C++ 的程式, 但許多 C++ 程式都少不了它, 我們把它稱為寫 C++ 程式前的準備工作, 有了它之後稍後寫程式會比較方便。這個程式因為內容較簡單, 所以準備工作只有一行, 有些較複雜的程式, 這部份可能多達十數行；從第 2 行開始才是真正的 C++ 程式。以下我們就逐行解說各程式的內容。

前處理指令與含括檔

```
#include<iostream>
```

- 前面說過, 程式第一行並不算是 C++ 的敘述, #include 是個前置處理指令 (preprocessing directive, 或稱為**假指令**) 在 C++ 編譯器編譯程式前, 會先由前置處理器 (preprocessor) 處理程式所有以 # 開頭的前置處理指令, 例如前置處理器看到 #include 這個指令時, 會將指令後面所列的**含括檔** (include file, 或稱**表頭檔**, header file) 整個置入到 #include 指令所在的位置, 同時將原來 #include 這一行移除。這個置入的動作, 就稱之為**含括** (include)。

標準函式庫

- 為了方便我們設計有用的程式, C++ 語言提供了一組具備各式功能的工具箱, 這個工具箱一般稱為**標準函式庫** (standard library)。**函式** (function) 就是一段程式組合, 它能提供某項實用的功能, 像是專門計算正弦函數 (sin) 值的函式、用來將一組文字或數字排序的函式...等等。在 C++ 的標準函式庫中已提供許多現成的函式, 例如當我們需要計算正弦函數值, 就可直接用標準函式庫中的數學函式來計算, 省下自己設計**計算正弦函數值**程式的工夫。

標準函式庫

- C++ 標準函式庫提供的不只是函式, 也有許多實用的**類別** (class) 和**物件** (object) 可直接取用。
- 而 C++ 語言規定, 使用任何函式、類別、物件時, 必須先宣告 (declare), **宣告**的意思是說某個字 (名字) 是有特殊的意義。同樣的, 要使用標準函式庫所提供的函式、類別、物件, 也都必需在程式中一一宣告這些函式、類別、物件的名字。

標準函式庫

- 但這樣一來, 寫個簡單的程式, 就要先在程式開頭宣告一堆名字, 實在太不方便, 因此**標準函式庫**所提供的函式、類別、物件等, 其宣告都放在預先定義好的含括檔中, 我們只要用 #include 指令含括必要的含括檔, 就能使用其中所宣告的任何函式、類別、或物件。

#include 的格式

- #include 指令可有下列兩種格式：
  - #include <檔名> ： 指示前置處理器到系統的 INCLUDE 資料夾去找要含括的檔案。這種形式主要是用於含括標準函式庫中的含括檔使用。
  - #include “檔名”：指示前置處理器先到目前的工作目錄去找, 若找不到, 再到系統的 INCLUDE 資料夾去找。

#include 的格式

- 所以我們程式 第 1 行所含括的就是標準函式庫中的 iostream 含括檔, 檔案內含 C++ 所提供的標準輸出及輸入工具, 我們程式中就會用到其中一樣工具。
- 除了 iostream 外, C++ 還有很多含括檔, 以 Visual C++ 2005 為例, 在 “Program Files\Microsoft Visual Studio 8\VC\include” 路徑下即可看到 Visual C++ 2005 所提供的含括檔。

main() 函式- C++ 程式的起點

```
int main()
{
    std::cout << "Hello, 我的第一個 C++ 程式";
    return 0;                    // 這一行可省略不寫
}
```

- 剛剛已介紹過**函式**, 而範例程式第 3 行的 main() 就是一個函式, 稱為 **main() 函式**。所有的 C++ 程式都要有個 main() 函式, 它是程式執行的起點。也就是說 C++ 程式在執行時, 都是從 main() 函式開始執行。

main() 函式- C++ 程式的起點

- main() 函式開頭的 “int” 是整數 (Integer) 的意思, 表示 main() 函式在執行完會**傳回**一個整數給作業系統。至於作業系統如何使用這個**傳回值**, 在此不做討論。只要請您記得, “int main()” 是 C++ 程式的主體, 我們寫的程式都要有它。
- 第 4 到 7 行由大括號 { } 包住的部份, 稱為 main() 函式的**本體** (body), 也就是放 main() 函式的內容。

main() 函式- C++ 程式的起點

- main() 函式雖是程式執行的起點, 但它不一定要放在原始程式的開頭, 其 main() 函式可能會放在程式的後方, 這對程式執行並無影響。因為編譯器仍會把程式編譯成從 main() 函式開始執行。

單一敘述的組成

```
std::cout << "Hello, 我的第一個 C++ 程式";
return 0;                    // 這一行可省略不寫
```

- 在 main() 函式中有 2 行程式, 或者說是有 2 個**敘述** (statement), 敘述 是 C++ 程式的基本組成單位。回顧一下前一章控制機器人的例子, 我們下給機器人的指令, 就是一個敘述。C++ 的每個敘述都必須以分號 (;) 結尾, 因為 C++ 編譯器就是依分號來分辨一個敘述的開頭和結尾, 才能據以將之編譯成對應的機器語言。

單一敘述的組成

- 敘述乃是由一些符合語法的字串所組成, 常見的敘述有：宣告、定義、運算式、設定、函式呼叫、迴圈及流程控制等, 例如：
  - 第 5 行敘述的功用應不用再說明了, 對照程式的執行結果, 我們就能明白此敘述的功用就是在螢幕上輸出 “Hello, 我的第一 個 C++ 程式” 這段文字。

單一敘述的組成

- 第 6 行的敘述 “return 0” 則是呼應前面介紹過 的 main() 函式傳回值, 也就是說我們的 main() 函式在輸出文字後, 會傳回 0 這個數值給作業系統。一般慣例都是用傳回 0 表示程式執行未發生任何問題。

- 由於已成慣例, 所以這行敘述**可以省略不寫**, 當編譯器看到 main() 函式沒有傳回數值時, 會自動把它視為傳回 0。

字符與空白符號

- 敘述可以再細分為由一或多個字符 **(Token)** 所組成。例如第 5 行 **std::cout << “Hello, 我的第一 個 C++ 程式”;** 就是由 **std::、cout<<、“Hello, 我的第一 個 C++ 程式”、;** 這些字符所組成。字符與字符之間可以加上適當數量的空白符號 (Whitespace), 以茲識別。例如同樣這一行程式, 我們可以寫成下面這個樣子, 其意義是相同的：

字符與空白符號

```
        std  ::     cout
<<
   "Hello, 我的第一個 C++ 程式";
```

- 我們在幾個字符間加上額外的空白、甚至換行, 其意義仍是相同的。因為對 C++ 編譯器來說, 空白字元、定位字元 (Tab)、換行都算是空白符號。

字符與空白符號

- 簡單的說, **字符 (Token)** 就是對 C++ 編譯器有意義的符號, 例如 **std 、cout** 等是在 <iostream> 含括檔中宣告的；而 ::、<< 則都屬於 C++ 語言的運算子 (operator) 而分號 (;) 則是用來分隔敘述的符號。
- 再次提醒, 在 C++ 中大小寫是有分別的, 所以 std 不能寫成 “STD”、“Std”、“sTd”, C++ 編譯器會將它們視為不同的字符。

為程式加上註解

```
return 0;                    // 這一行可省略不寫
```

- 在第 6 行程式後面, 從兩條斜線開始的部份, 稱為**註解** (comment)。註解是我們寫給自己或其他人閱讀的說明文字, 而編譯器在遇到註解時, 會自動跳過。
- 註解的格式有 2 種：
  - 單行註解：也就是以 2 個斜線 (//) 開頭的註解。當編譯器看到 "//" 時, 就會將 "//" 到這一行結尾的文字完全忽略, 從下一行程式再開始編譯。

為程式加上註解

- 區塊註解：如果您要寫的註解有好幾行，就可使用這種方式，其格式是以成對的 /* 與 */ 來包含所要加入的註解說明。這一對符號不需在同一行，編譯器會自動跳過這一對符號間的所有文字，例如：(如下頁)

- 下述這個程式的執行結果和前一個程式相同，而第 14 行就是一個區塊註解。如果您將 C++ 敘述寫在其中，編譯器將不會看到它，所以也發生不了作用。

為程式加上註解

程式 Ch02-02.cpp 含註解的程式

```
01 /*   以下是我們所寫的第一個 C++ 程式
02       作者：施威銘研究室
03       版本：1.0
04       內容：在螢幕輸出一行訊息                    */
05 #include<iostream>
06
07 int main()
08 {
09    std::cout << "Hello, 我的第一個 C++ 程式";
10    return 0;                  // 這一行可省略不寫
11 }
```

C++ 的輸出與輸入

- C++ 語言本身並不具備輸出與輸入的功能，所有的輸出與輸入均是藉由函式呼叫的方式來達成。這些函式都已預先製作好了，並放在標準函式庫內供我們使用。輸出與輸入裝置的種類很多，有鍵盤、螢幕、檔案、印表機等，其中有一組特殊的裝置稱為標準輸入與輸出，一般而言就是指由鍵盤讀入和由螢幕輸出。

C++ 的輸出與輸入

- 我們回頭看 **std::cout << “Hello, 我的第一個 C++ 程式”;** 這行敘述, 其中 **std::cout** 是代表標準輸出的**串流物件**。而我們用 << 這個運算子將 “Hello, 我的第一 個 C++ 程式” 這個**字串**輸出到螢幕上, 所以我們能在畫面上看到這串文字。
- 利用同樣的方式, 我們可以輸出多行的文字字串, 例如下面這個程式：

2-2-6 C++ 的輸出與輸入

程式 Ch02-03.cpp 輸出多行訊息

```
01 #include<iostream>
02
03 int main()
04 {
05    std::cout << "Hello, 我的第一個 C++ 程式";
06    std::cout << std::endl;                    // 換行
07    std::cout << "Bye, C++ !";
08 }
```

執行結果

```
Hello, 我的第一個 C++ 程式
Bye, C++ !
```

C++ 的輸出與輸入

- 第 5 及 第 7 行的程式分別就是輸出**執行結果**所示的 2 行訊息。至於第 6 行 的 std::endl 則是定義在 iostream 含括檔中, 它代表的是一行的結尾 (END Line), 也就是讓輸出的內容換行, 所以 第 7 行程式輸出的訊息, 會出現在另一行。如果沒有第 6 行輸出 std::endl 的敘述, 則 “Bye, C++ !” 這幾個字會緊接在 “Hello, 我的第一 個 C++ 程式” 後面。

C++ 的輸出與輸入

- 像前面這樣輸出訊息非常不便, 每次都要打一串 “std::cout <<” 文字。如果程式要輸出 100 行訊息, 豈不是要重複 200 行都要打 “std::cout <<” (包括換行動作) 。
- 其實我們可透過 2 種方式簡化使用 “std::cout” 的敘述內容：
  - 在程式開頭即宣告使用 std 名稱空間。
  - 將多個 std::cout 敘述**串接**成單一敘述。

使用 std 名稱空間

- 其實標準輸出的物件名稱是 “cout”, 前面的 “std” 代表的是 cout 這個名字是定義在一個名 為 std 的**名稱空間** (namespace, 或稱命名空間) 簡單的說, **名稱空間**是避免大家使用同樣的名稱為類別、物件命名的一種機制。我們可做這樣的比喻：在現實生活中, 我們可能會在同一班就遇到同名同姓的人, 但 C++ 程式 (同一班) 卻不允許有同名同姓的情況發生, 但可允許不同班有同名同姓者。

使用 std 名稱空間

- **名稱空間**的功用, 是讓 C++ 程式中有同名同姓的人時, 可將他們區分為不同班 (不同的名稱空間), 如此 C++ 編譯器就不會弄錯, 造成編譯錯誤；因此 “std::cout” 的意思, 就是告訴 C++ 編譯器：這位 cout 同學 是 std 這一班的。
- 但我們的程式並沒有那麼複雜, 實在不需要去分班, 所以我們可在程式一開始就宣告說我們都是 std 這一班, 這樣一來, 使用 cout 時, 前面就不需加上 “std::” 了。宣告的方法是用 using namespace 敘述, 如以下程式所示：

使用 std 名稱空間

程式 Ch02-04.cpp 指定 std 名稱空間

```
01 #include<iostream>
02 using namespace std;     // 使用 std 名稱空間
03
04 int main()
05 {
06    cout << "Hello, 我的第一個 C++ 程式";
07    cout << endl;                // 換行
08    cout << "Bye, C++ !";
09 }
```

串接輸出

執行結果

```
Hello, 我的第一個 C++ 程式
Bye, C++ !
```

- 為了方便起見,“<<” 運算子可以**串接**起來使用,正如我們平常寫加減法的運算式時,可以將 +、-、*、/ 符號串在一起用一樣,例如前一個範例程式的第 6、7 行,可以合併成

```
cout << "Hello, 我的第一個 C++ 程式" << endl;
```

串接輸出

- 輸出的順序是由左至右逐一輸出, 所 以 cout 會先在螢幕上輸出一行 “Hello...” 的字串後, 就會緊接著換行。利用同樣的方式, 我們可以串接多個輸出內容。所以我們又可將程式簡化成：(如下頁)
- 請注意第 68 行總共只有一個分號, 所以它們其實是一個敘述, 只不過我們將它分成 3 行來寫。因為換行也算是空白符號, 所以像這樣分成多行, C++ 編譯器也能正常解讀並編譯程式。

串接輸出

程式 Ch02-05.cpp 串接輸出

```
01 #include<iostream>
02 using namespace std;     // 使用 std 名稱空間
03
04 int main()
05 {
06    cout << "Hello, 我的第一個 C++ 程式"
07         << endl
08         << "Bye, C++ !";
09 }
```

輸入資料

- std::cout 是用於輸出資料, 至於要想讓使用者由鍵盤輸入資料, 則需使用 std::cin 物件。cin 稱為**標準輸入** (Standard Input), 在個人電腦上通常就是指鍵盤, 所以使用 cin 就是要從鍵盤輸入資料。cin 的用法很簡單, 最基本的語法如下：

```
cin >> 儲存物件的變數;
```

輸入資料

- cin >> 儲存物件的變數; 由 cout 輸出時用的 是 “<<” 運算子, 由 cin 輸入時則換個方向使用 “>>” 運算子。由外界輸入資料時, 程式要準備一個儲存空間來放這筆資料, 而最簡單的方式就是準備一個**變數 (variable)** 來存放輸入的資料。我們先簡單示範 cin 的用法：

輸入資料

程式 Ch02-06.cpp 由鍵盤取得輸入資料

```
01 #include<iostream>
02 using namespace std;     // 使用 std 名稱空間
03
04 int main()
05 {
06    cout << "請輸入一個數字：";
07    int i;                // 宣告一個整數變數 i
08    cin >> i;             // 將鍵盤輸入的資料存到 i
09    cout << "您輸入的數字是 " << i;
10 }
```

輸入資料

- 其中第 7 行的敘述是宣告一個用來存放整數 (int) 的變數, 變數名稱為 i, 在第 8 行就將由鍵盤輸入的資料存到 i 之中, 在第 9 行則將 i 所存的數值接在字串後一起輸出。
- 編譯好程式後, 執行時會先現如下的訊息：

```
請輸入一個數字：
```

- 這時候程式會暫停, 等待我們輸入資料, 例如我們輸入 123 後按 , 程式就會接著執行, 也就是輸出前列第 9 行的訊息：

輸入資料

```
請輸入一個數字：123  Enter
您輸入的數字是 123          ←輸入資料後才會出現這行訊息
```

- 請注意, 由於用來接受輸入的變數 i 只能存放整數的資料, 所以若我們輸入文字或符號等非數字資料, cin 就會視為輸入錯誤, 而不會將輸入的文字或符號存入 i, 此時 i 將維持其預設值 1, 如下所示：

```
請輸入一個數字：abc  ←輸入文字
您輸入的數字是 1     ←結果 i 的值仍是 1
```

2-3 良好的程式撰寫習慣

- 寫程式的目的不外乎是能正常執行，但是要如何讓您的程式簡明易懂也是相當重要的。因為我們可能在程式寫好過了半個月、三個月後要回頭參考或修改，此時要讀幾個月前寫的程式，若看到程式碼排得亂七八糟，縱然記憶力再好，恐怕也很難記得當初程式所想表達的含意。為了避免這種情形，一般都會依循幾個原則，建立良好的程式撰寫習慣。

良好的程式撰寫習慣

- 前面說過, C++ 是自由格式的程式語言, 只要有確實遵照字符間加上必要的空格、敘述結尾加上分號... 等基本的語法規則, 不管我們的程式如何排列, C++ 編譯器仍能正常讀取進而編譯。但寫程式並不只是寫給 C++ 編譯器看, 很多時候程式也要給除了自己以外的人閱讀。所以如何提高程式的**可讀性**, 就是我們在撰寫程式時所要注意的。

適當的分行：
便於整篇程式的閱讀

- 在 C++ 語言中並未限制單行的長度，所以就算把整個程式寫成非常長的一行，只要每段敘述都有正確地以分號標開，編譯程式時也不會發生錯誤。問題是，這樣的程式讀起來，感覺會像讀一篇沒有標點符號的文章一樣，不知道哪裡該停頓，哪裡是段落結束：

```
#include<iostream> using namespace std;int main(){cout<<...
```

前置處理指令、C++ 敘述都擠在同一行

適當的分行：
便於整篇程式的閱讀

- 因此一般習慣都是將每個敘述、前置處理指令都個別放在一行，而函式的名稱、大括號也是各佔一行。就像我們前面的範例程式所示：

```
#include<iostream>
using namespace std;     // 使用 std 名稱空間

int main()
{
...
```

適當的分行：
便於整篇程式的閱讀

- 如果敘述很長時, 可以在適當的地方將之換行, 例如前面已看過的例子：

```
cout << "Hello, 我的第一個 C++ 程式"
     << endl
     << "Bye, C++ !";
```

被分成 3 行的敘述

- 這 3 行程式就等於：

```
cout << "Hello, 我的第一個 C++ 程式" << endl << "Bye, C++ !";
```

適當的分行：便於整篇程式的閱讀

- 如果要用 cout 輸出的字串本身就很長時，可將字串切割成多個子字串，個別輸出，例如：

```
cout << "我想用 cout 輸出的雙引號字串很長, 超過三十個字...";
```

- 只要適當的切割，就能將它分成 2 個子字串，分列 在 2 行 (注意！切割後的每個子字串前後都要用引號“括起來)：

```
cout << "我想用 cout 輸出的雙引號字串很長, "
     << "超過三十個字...";
```

適當的分行：
便於整篇程式的閱讀

- 或改用 2 個敘述個別輸出：

```
cout << "我想用 cout 輸出的雙引號字串很長, ";
cout << "超過三十個字...";
```

- 其它類型的敘述, 也都可採類似的技巧。

適當縮排

- 為了提高可讀性，寫程式要像寫文章一樣：段落明顯，章節分明。要讓程式看起來有段落、章節的感覺，就需善用內縮的技巧。

```
int main()
{
   cout << "Hello, 我的第一個 C++ 程式"
        << endl
        << "Bye, C++ !";
}
```

大括號中每行都內縮

被分成多行的敘述，第 2 行以下再內縮

適當縮排

- 一般習慣只要用到大括號時, 則大括號以內的程式都會比大括號的位置內縮一些。至於內縮幾格, 視每個人的習慣不同, 有些人習慣空 2 或 4 格, 也有人是直接用 Tab 鍵來做縮排。而像上列 cout 敘述的 第 23 行, 其內縮方式則是讓 “<<” 符號彼此對齊。利用這種方式, 我們就能讓程式看起來有段落、層次感, 閱讀起來也比較能找出條理。

程式碼的註解：幫助瞭解程式

- 最後一點要注意的，就是要適時的替程式加上註解。為什麼要加註解呢？一來有時程式的邏輯會較為巧妙，無法一眼看出，此時適當的加上說明，可幫助閱讀。
- 其次從維護程式的考量，對自己，可能早就忘了程式在寫什麼；對別人來說，則可能完全不能瞭解程式的內容。為了提升效率，避免大家浪費太多時間，若能在一開始寫程式時，就在適當的地方加上各種說明註解，這樣維護程式也會變得比較輕鬆自在。

程式碼的註解：幫助瞭解程式

- 在註解您的程式時, 建議可依如下的原則：
  - 盡量使用 “//” 來標示註解, 除非是多行的連續註解, 才使用 /* 與 */。畢竟只在註解開頭輸入 “//”, 比要在註解前後輸入 “/*” 與 “*/” 來得方便。
  - 當註解太長時需要換行時, 新的一行也必須在 /* 與 */ 之間。
  - 註解越詳細越好, 但也不要浮濫, 讓程式檔充斥一堆**非程式**的文字, 反而影響閱讀。