C++的介紹

1. 介紹：

電腦上的應用軟體必須用程式語言來編寫，而一個好的語言可以讓程式設計師設計出一個效率極高的應用軟體或者是方便程式設計師寫出想要的軟體，以減少寫程式的時間進而增進效率，而C語言便是這樣子的一種程式語言，用它來寫軟體不僅容易編寫，而且所寫出來的軟體執行效率更是高。此外，用途廣泛更是C語言的另一項優點，舉凡一般的程式、網路上的cgi應用、編譯器、作業系統、甚至是控制硬體的應用程式(如控制印表機來列印物品的列印程式)，皆可以用C語言來編寫，用途廣大，鮮少程式語言可以比擬。

此外，繼C語言之後，又有一個強而有力的程式語言產生了，那就是C++。C++以不同的概念出發，並且繼承了不少C語言的語法，一樣是便利了世界上的程式設計師的程式開發工作。然而，儘管C++的語法跟C語言的語法差不多，它們的概念仍然是差太多了，以下將會介紹它有別於C語言的一些新概念。

2. C語言的歷史：

早期的系統軟體皆需用組合語言來編寫，而用組合語言來編寫程式雖然比機器碼純數字的程式內容在可讀性上來得更高、更有意義，但是仍是十分不足、難以了解內容且編寫程式不易，故有人便希望能發展出一個高階語言能取代組合語言，寫出可讀性更高的程式碼。但是一般的高階語言又有難以控制硬體的缺點，無法直接控制硬體，效率比不上組合語言，故又希望能夠設計出一個便於控制硬體的高階語言，而C語言便是為了因應此種情形而生的。

C語言的源頭，一開始可以追溯到1960年的ALGOL 60。ALGOL 60是一種解決面向問題的高階語言，當時不少的科學家都了解到其優點，也覺得該語言不錯，但是他們都知道ALGOL 60尚有一些缺陷，且偏離硬體太遠，不適合用來寫系統程式，故有人就嘗試設計出另一個比較好的語言來取代它來編寫系統程式。

之後，於1963年，英國的劍橋大學和倫敦大學以ALGOL 60為基礎，共同推出了CPL (Combined Programming Language)語言。CPL較ALGOL 60要來得更接近硬體，但是格局太大，編寫程式上仍嫌不方便，故在1967年的時候，英國劍橋大學的Martin Richards對CPL語言做了簡化，推出了BCPL (Base Combined Programming Language)語言。

後來，美國貝爾實驗室的Ken Thompson在使用了BCPL語言之後覺得不錯，但他認為如果想在一台DEC PDP-7的電腦上使用該語言的話，該語言必須還要再更精簡才行。於是在經過深入的研究之後，Ken Thompson於1970年以BCPL語言為基礎簡化了BCPL語言，設計出了十分接近硬體的B語言(取BCPL的第一個字母)。B語言十分地簡單，它

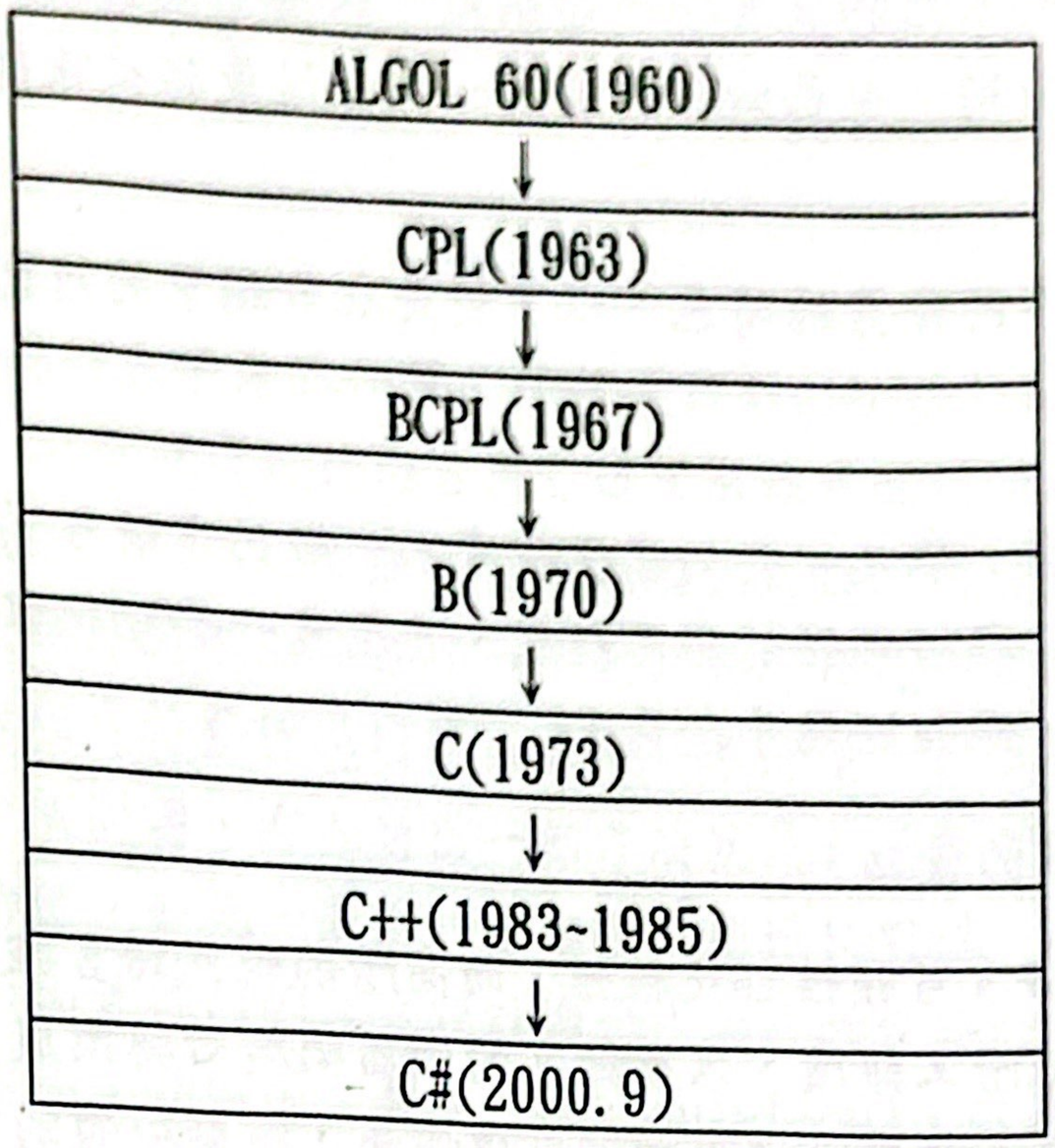
的資料是沒有在分資料型態的！(即所謂的 typeless)而且在語法上它也做了相當大的精簡！在設計出 B 語言之後，Ken Thompson 在貝爾實驗室裡用 B 語言寫出早期的 UNIX 作業系統。

然而由於資料沒有在分資料型態，再加上實在是太精簡了，功能實在是有限，故於 1972 年至 1973 年間，貝爾實驗室的 Dennis Ritchie 以 B 語言為基礎，結合了 B 語言及 BCPL 語言的一些重要觀念，再加資料型態的概念及其他的一些強而有力的觀念之後，設計出功能佳的 C 語言(取 BCPL 的第二個字母)。而由於 C 語言的功能實在是十分強大，所以在 Ken Thompson 洞悉有此語言之後，Ken Thompson 和 Dennis Ritchie 兩人又合力將 UNIX 作業系統重新再寫過一次，寫出了新版的 UNIX 作業系統，之後 C 語言亦經過了幾次的修正。然而雖然 C 語言好用，但是在 C 語言發跡之初也只有貝爾實驗室內部的人在用而已。C 語言廣泛為人所知是一直到第六版的 UNIX 系統出來了之後才有的事。當時第六版的 UNIX 系統出來後，C 語言突出的性能才受到廣大的注目，也因此那個時候 C 語言才開始廣為流傳。之後，Brain W. Kernighan 和 Dennis Ritchie 於 1978 年出了一本書叫做《The C Programming Language》，而書推出之後 C 語言再一次受到廣大的注意！此外，該本書所介紹的 C　語言成為了後來廣泛使用的 C 語言版本的基礎，因此該本書所介紹的 C 語言被稱為「標準 C」。

然而自從 1977 年出現了可不依賴具體機器語言的 C 語言的編譯文本《可移植 C 語言編譯程序》之後，不同硬體平台上的 C 語言移植的速度不僅變快了，也變得容易多了，而此種現象雖然使得C語言與其所寫出來的UNIX系統能夠在不同的硬體平台上廣泛地被流傳，卻也造成了不同平台上 C 語言語法相近但不相容的情形，故有人希望 C 語言能夠被標準化，進而使得程式設計師在設計程式時能比較方便，故有鑑於此，美國國家標準局(American National Standards Institute，即 ANSI)於 1983 年將現有的許多種 C 語言標準化，制定了一個標準化的 C 語言，叫做 ANSI C。之後 Brain W. Kernighan 和 Dennis Ritchie 在 1988 年時，根據 ANSI C 又重新寫了《The C Programming Language》一書。而在之前 1987 年時，ANSI 又公佈了新的標準叫做 87 ANSI C，而這個 87 ANSI C 便是我們目前的所用的標準化的 C 語言。

而在 C 語言被發明之後，為了增進程式碼的品質及增加程式碼的可再利用性，貝爾實驗室的 Bjarne Sroustrup 於 1983 年以前加入了少許的新概念於 C 語言之後，產生了一個新的 C 語言，而 Bjarne Sroustrup 稱它為”C with classes”。而在 1983 至 1985 年間，他又在 C 語言中加入了 Simula 語言的 class(類別)的概念，以及物件導向的概念，產生出 C++語言，於是我們所熟知的 C++便產生了！

此外，在 2000 年 9 月，Microsoft 在自己的.NET Framework 計畫中，以 C 語言為樣本，沿用了 C++的函式庫，自行研發了一個類似於 C 語言的 C#語言，而 C++和 C#都是 C 語言後來的演化。以下是簡易的圖示：



3. C語言的性質、特色、優點：

有趣的是，C語言的性質幾乎就是它的優點，以下將介紹它的一些性質：

(1) 結構化的設計流程：

C語言的程式流程是十分地有結構的，是按照程式內容的順序，由上而下一路執行下來的，並且提供 if-else 條件判斷，讓程式設計者能適時的利用條件判斷適當地執行應做的動作；此外，對於重覆執行的條件判斷，C語言提供了 for、while、do-while 等迴圈設計供程式設計者方便處理重覆執行類型的條件判斷。此外，對於繁複的 if-else 條件判斷式，C語言亦提供了 switch-case 型的條件判斷，方便程式設計師設計繁複的條件判斷式。而這樣的設計雖然有不少的程式語言也是如此，不過仍可算是C語言的一項優點。相較於所謂的第五代程式語言視窗式的程式設計流程而言，結構化的程式設計在結構上顯得比較有條理、組織且架構，在邏輯上邏輯性亦顯得比較強、比較穩固！

(2) 指標運算：

C語言提供了指標的運算，可直接儲存程式內設定的變數的記憶體位置編號，並且可對記憶體位置編號做加減等運算，進而快速讀取在記憶體連續排列的變數內容並做修改。而這個功能寫過C語言的人都知道，是C語言中一個十分重要且強而有力的功能！利用指標能對記憶體位置編號做運算的特性，指標甚至可以用來設計程式中的資料結構！功能可以算是十分強大的了！

(3) 使用函式庫：

用C語言所寫的程式在編譯成最終的應用程式時，總是需要連結到函式庫(library)來將最終的程式編譯出來。函式庫充斥著許多C語言中用到的函數的內容，而使用函式庫的概念優點很多，包括程式碼精簡、函式庫可替換、可共用等等；此外，使用標準函

式庫內定義的函數，不管到哪不用怕沒有那一些函數可以用！使用函式庫唯一的缺點是，當沒有自己所想要函式庫時，自己必須去編寫一個，這是程式設計師們普遍認為較麻煩的部分。

此外，C 語言充斥著許多的優秀的特色，以下便是在介紹它的特色：

(1) 使用範圍廣，用途廣大：

C 語言可以寫的程式很多，舉凡一般的應用程式、一般的系統程式(如作業系統)、網路上的應用(如 cgi 應用)、操控硬體的應用程式、甚至是一些驅動程式，皆可以用 C 語言來編寫！用途十分廣大，跟組語幾乎無異！

(2) 兼具高階語言和低階語言的優點：

一般而言，低階的程式語言可讀性都比較低，而高階的程式語言控制硬體的能力也比較差，而這是因為要將最低階又難讀、只有數字的機器碼精心轉換成一個可讀性高的程式語言是一件十分困難的事，故難以做出一個可讀性高、控制硬體的能力又好的高階語言。然而 C 語言卻突破了這一層障礙，它的程式碼不僅易讀易懂、可讀性高，同時又能輕鬆地控制硬體！而我想，這是其他的程式語言難以趕上的部分，可以算是 C 語言最大的幾個特色之一了！

(3) 執行效率高、速度快：

C 語言不僅在寫程式的過程中容易編寫，在控制硬體上亦比較輕鬆，同時在執行程式上，用 C 語言所編寫出來的軟體效率亦是十分地高的！執行速度很快！

(4) 跨平台能力強、可攜性大、平台範圍廣：

自從在 1977 年時出現了可不依賴具體機器語言的 C 語言的編譯文本《可移植 C 語言編譯程序》之後，C 語言的跨平台能力又比 1977 年以前更強了！然後藉由此事件的影響，C 語言的移植風更是旺盛，故 C 語言的平台範圍廣大！

(5) 語法較自由、較隨性：

寫過 C 語言的人都知道，C 語言的語法比想像地要來得自由多了！有時寫程式的新手在了解電腦聰明度上的限制之後，往往會用比較死板的語法來寫程式，因為他們知道電腦並不是十分地聰明；但是他們可能在學習 C 語言的過程之中會慢慢發現，C 語言比他們想像的要來得聰明多了！他們可以用更靈活、更有效率的寫法來寫程式！此外，C 語言的某一些語法和關鍵字的設計便是刻意要設計出自由度高的寫法的！而這些語法和關鍵字使得人們可以以比較自由、比較有效率的思維去編寫程式！而這些都是 C 語言語法自由的地方！

(6) 允許直接控制硬體：

C 語言可以利用設計得當的函式庫來直接操作硬體！許多地方存在有許多種處理硬體的函式庫可供程式設計者使用。倘若函式庫不足以程式設計者控制硬體的話，程式設

計者可以自行製作所需的函式庫。

4. C++的特色：

C++最為人所知的特色便是物件導向的程式設計(object-oriented programming，簡稱OOP)的概念了！以往程式設計師在用C語言寫程式的時候所使用的程式設計概念是結構化的程式設計概念，而之前有提到過說此種程式設計的優點，包括了邏輯性較強等等。但是此種的程式設計的概念存在有一個大缺點那就是：當程式設計師要寫一個十分大的程式時，容易對程式的整體邏輯架構掌握得不清楚，畢竟程式碼實在是太長了，容易出錯，故針對這一個問題，有人想出一個方法就是：將整個程式碼模組化，分成許多較小、較易掌握的物件，如此一來對於整個程式的邏輯架構，我們可以比較容易掌握得住，而這便是所謂的物件導向的程式設計概念！而在物件導向程式設計的概念之下，存在有最重要的三個概念，分別是：封裝(Encapsulation)、繼承(Inheritance)還有多型(polymorphism)，而這三個概念主要是在討論物件的性質。以下分別介紹他們的概念：

. 封裝(Encapsulation)：

封裝(Encapsulation)是指物件的封閉性。每個物件都像是一部又一部的機器一樣，被塑膠殼或鐵皮包了起來看不到內部，但是機器的上面有著一層操控的介面供人操作機器，且人們只要輸入某種實質的物品或是一道指令，機器內部被封起來不為人知的運作機制便會在一段時間之後輸出結果出來，人們都不需要知道機器內部的運作機制便可以完成此項動作。而物件也是一樣的！程式設計師在設計完一個物件之後，只要在物件所給了的介面上輸入某個訊息給物件的話，不必再知道內部的運作方式，物件便會將結果顯現出來！這便是所謂物件封裝的特性之一。

物件內包含的內容不是只有一堆用來處理資訊的程式碼而已，它還包括了一些資料的宣告！(在C++裡資料的宣告包括了變數的宣告還有函數的宣告，而函數的內容便是用來處理資料的程式碼。)而有一些資料是被隱藏起來保護的，而那些資料只有物件內部能夠處理，物件的外面無法去改變它；而有一些資料是公開的，物件外面的程式碼是可以改變它的，故因此這些公開的資料便被當作是與外面交流的一層媒介。我們在物件的外面改變這一些資料，之後那一些資料再送到物件的內部去處理，這便是物件用來與外界交流的方式！而這一些都算是物件封裝的性質之一！

. 繼承(Inheritance)：

各個物件彼此之間可以沒有關係，亦可以有繼承的關係。所謂繼承的關係就有如生物界有界、門、綱、目、科、屬、種等階層的關係，犬目之下有犬科，犬科之下有犬屬一樣，有歸屬的關係和繼承的關係，且下面的階層總是會擁有其上面的階層的一些性質，並且自己又有更多上面的階層所沒有的特性。而在物件導向的程式設計概念之中，物件的繼承關係也是一樣的，物件會擁有其上面物件的一些性質，又會有自己新增的一些性

質。而這樣的設計有一項運用上的優點，那便是當我想要新增某一個舊物件的新功能時，我可以不必再去修改原本舊的物件，我只需要再寫一個新的物件去繼承那一個舊的物件，我便可以使用那一個舊物件內的功能了！

上述所講的是一個物件去繼承另一個物件的例子，而一個物件繼承另一個物件叫做「單一繼承」；然而在繼承的這一項性質裡，我們還可以做「多重繼承」，即一個新物件去繼承另外多個舊物件，使那一些多個物件內所有的功能全都能被該新物件所使用，而這是一個容納許多功能的好方法；然而此種方法亦有其缺點，那就是如果有二個舊物件曾經繼承過同一個更舊的物件的話，那麼新物件在繼承該二個舊物件時，便會發生繼承功能上的一些問題！而如果要避免此種問題的話，事前必須做好妥善的規畫才行！

- 多型(polymorphism)：

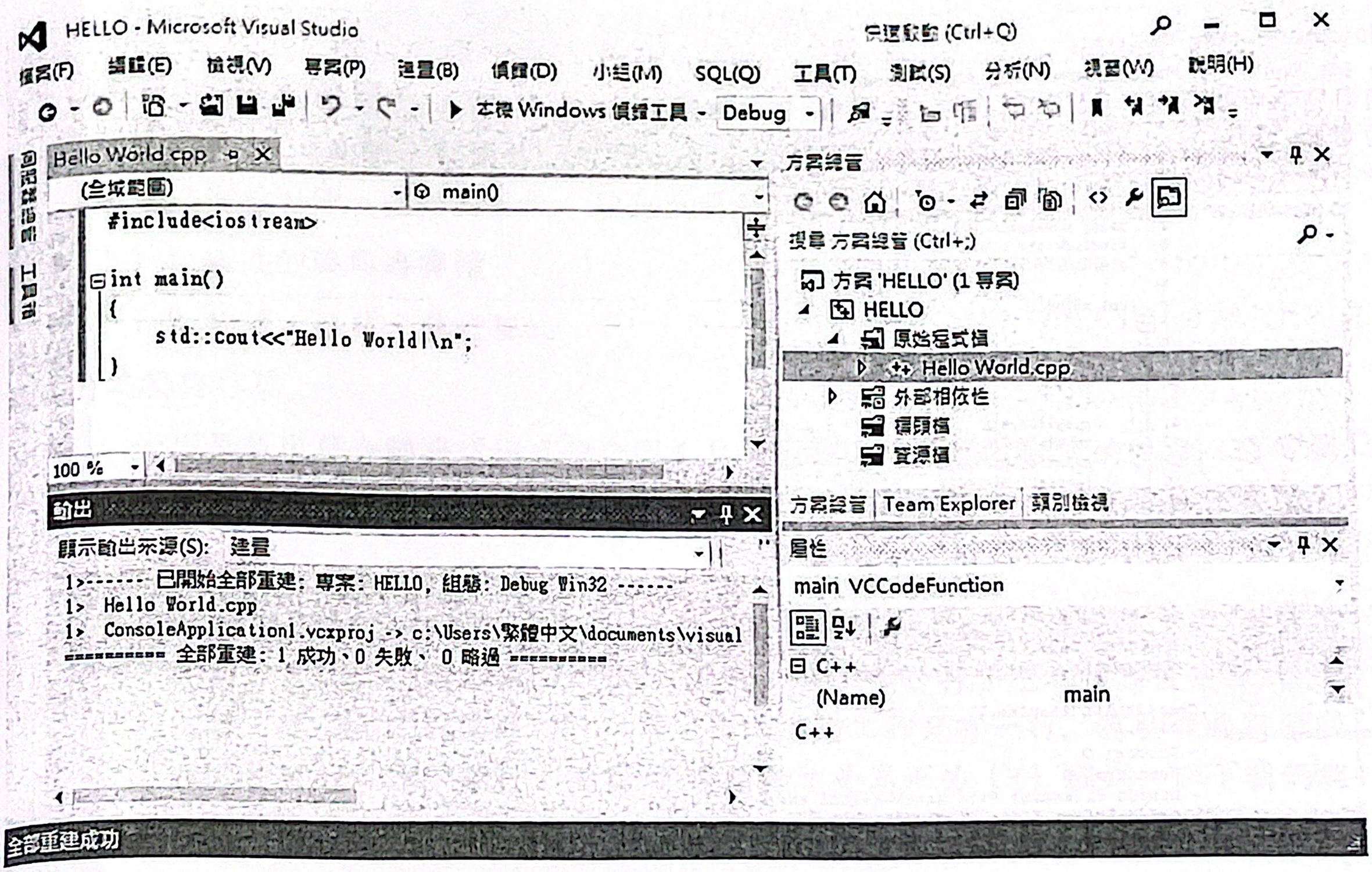
多型(polymorphism)是物件導向的程式設計中一個比較不好懂的概念。所謂的多型指的是物件內，一樣函數名稱的函數在被不同的下層物件所繼承時，裡面所做的動作可以不一樣！而這樣有時可以在寫程式時比較好理解，似乎程式碼維護也比較容易。

在 C++內，物件都是以製作 class(類別)的方式來表現出物件的特性的。Class 有別於 C 語言的 struct，它可以在裡面宣告變數和函數的名稱，並且可以編寫函數的內容，編寫處理資訊的方法，而變數和函數亦可以設成 public、private 以及 protected 三種，不僅跟 C 語言中僅能集體宣告一群變數而無法宣告函數的 struct 相差甚大，同時亦能符合以上所講的一些物件所應有的性質！

此外，C++物件導向程式設計的概念除了設了物件這麼一個獨特的概念外，亦設了一種叫做 template(樣版、模版)的東西。Template 是原本 C 語言中用來給予某一些名稱固定的值的符號#define 的延伸，目的亦是為了不重覆一些常被使用的程式碼所設立的。它的語法十分地特殊！有跟原本 C 語言語法不同的地方，對於剛學 template 的人而言有一點不易學，不過有時正因為它的語法比較特殊，故有人會比較傾向於使用它。它類似於某一些函數的性質，實質內容都被寫在其他的程式碼內了，但是語法卻跟函數完全不同！它比較適用於演算法固定，與資料型態沒關係的場合。而 C++之中存在有 STL(Standard Template Library，標準樣版函式庫)供程式設計師使用 template 時用。

Visual C++ 系列

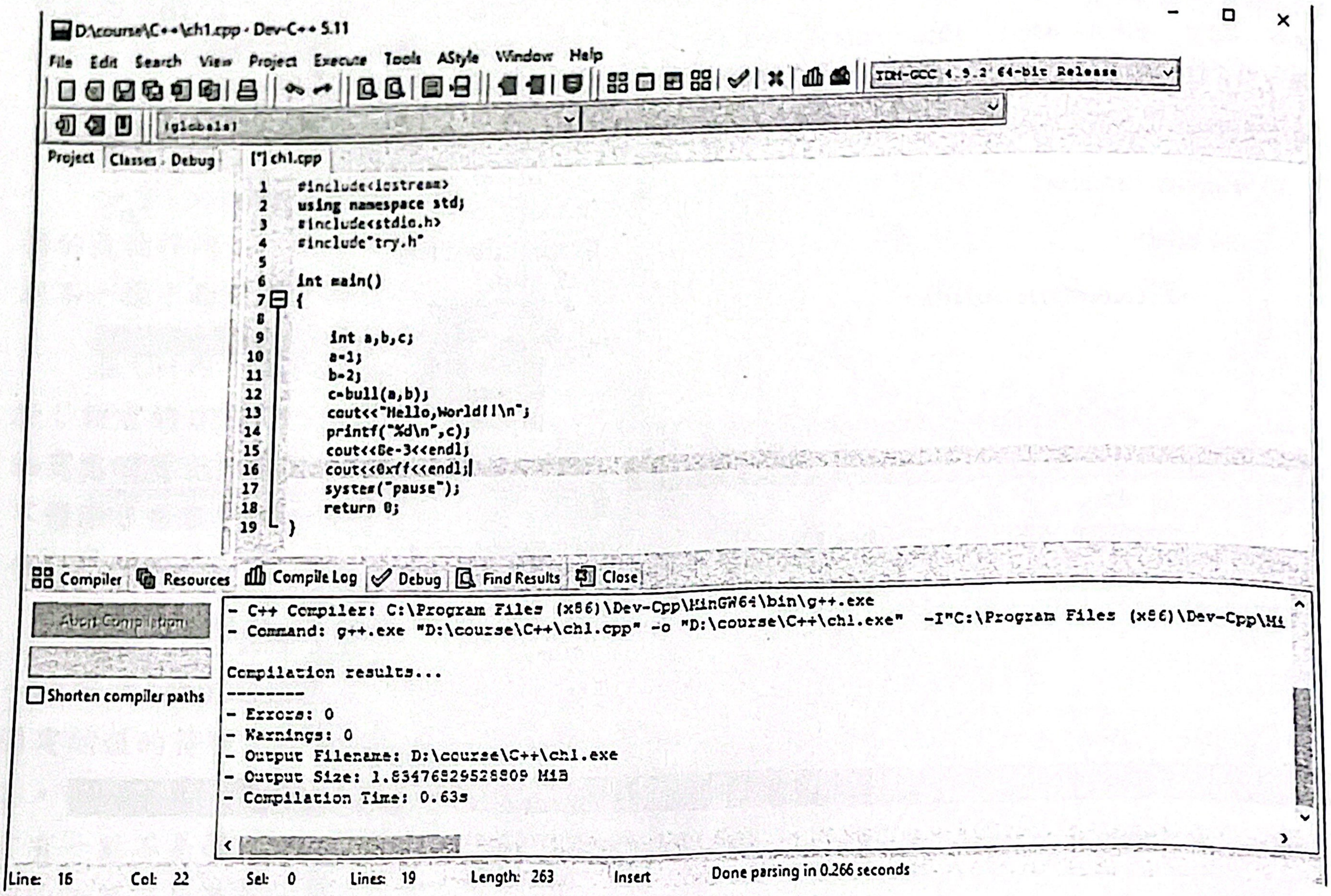
- 微軟公司的 Visual C++ 是 Windows 平台上主流的 C++ 整合開發環境之一。Visual C++ 的 IDE 和微軟公司其它程式語言產品（包括 Visual Basic、C#、J# 等）其實是共用一個名為 Visual Studio 的 IDE。對於想同時用微軟其它語言或技術的開發人員而言，由於只需熟悉一套操作介面即可，所以使用起來較為方便。



Microsoft Visual Studio

Dev C++ 系列

- Dev-C++是一套用於開發 C/C++（C++11）的自由的整合式開發環境（IDE），並以 GPL(GNU General Public License，縮寫 GNU GPL）作為散布許可。
- Dev-C++是一個 SourceForge 的計畫，是由 Colin Laplace 這位程式設計師及其公司 Bloodshed Software 所開始的。目前 Dev-C++一般用於撰寫執行於 Microsoft Windows 的程式。



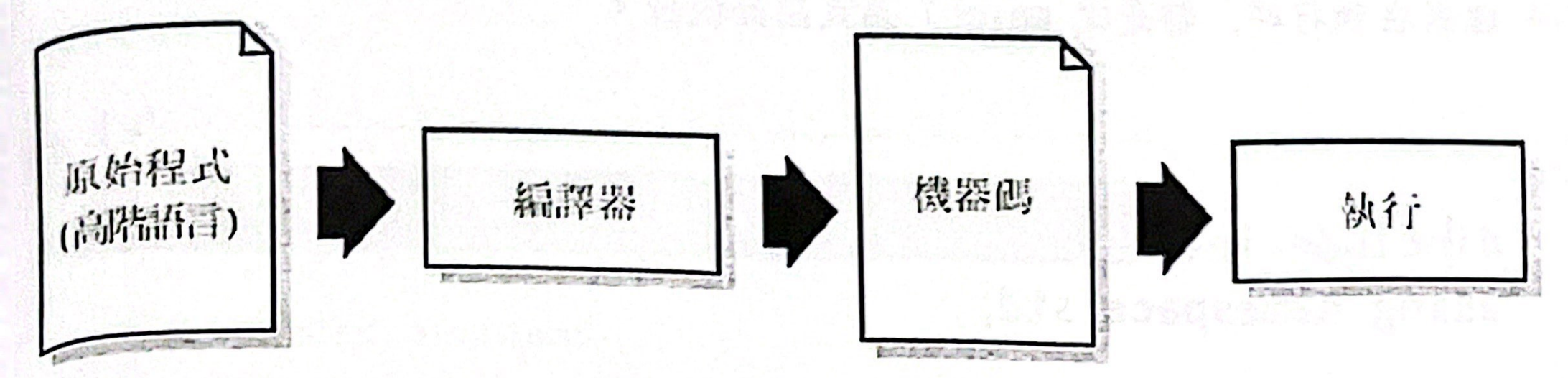
1-1 甚麼是程式語言？

- 身為電腦使用者，我們常會說：用 xx 程式來做 xx 事情。那這麼多能做不同事情、發揮不同功效的應用程式是如何產生的呢？簡單的說，是程式設計人員(programmer）寫出來的。而程式設計人員寫程式所用的語言，就稱為電腦程式語言，或簡稱程式語言 (Programming Language)。

高階語言

這樣不但更容易閱讀和理解，也比使用低階語言所寫出來的程式精簡多了。不過電腦並無法看懂這樣的程式，和組合語言一樣需要一個轉譯的動作，將使用高階語言所撰寫的程式轉換成電腦所能看懂的機器語言，然後才能依此執行。這個轉換的動作是由

各程式語言的編譯器（Compiler）或是解譯器（Interpreter）來進行



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。

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）。

main() 函式- C++ 程式的起點

- 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。

ch1.cpp

```
1   #include<iostream>
2   using namespace std;
3
4   int main()
5   {
6       cout<<"Hello,World!!\n";
7
8       return 0;
9   }
```

cout = console out（螢幕輸出）

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

練習：

請印出你的姓名，生日，星座

2-2 甚麼是變數？

- 當我們在進行心算時，要用大腦記住數字，然後將它們加減乘除，只要數字一多，就會發現我們的腦袋實在不爭氣，可能連要計算的數字都記不住，更不要說想要在腦中計算出答案了。這時候我們就會佩服在電視上表演心算的神童，也會覺得計算機（電腦）實在好用，手指頭按一按就能算出答案。
- 其實，電腦也沒有比我們高明多少，只不過它能清楚確實記住要計算的數字，也就是存放到記憶體中。
- 在這個例子中，數字就是所謂的資料；而計算的動作則稱為電腦在處理資料；計算出來的答案則是處理的結果（其實也算是資料）寫程式時，就是在命令電腦以我們想要的方式處理資料，而在程式中就是用變數（Variable）來表示儲存資料

的地方。抽象一點的說法，就直接說變數就是儲存資料的地方。

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 }
```

- 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，執行時會先出現如下的訊息：
- 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：

練習 1：輸入一個整數，並將輸出其平方

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。

1. 適當的分行：便於整篇程式的閱讀
2. 適當縮排
3. 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。

2-4 宣告變數

- 在這個程式中，第 7 行 的 "int i;" 敘述的功用，就是宣告 (Declare) 一個變數，它的名字叫 做 i，而最前面的 int 則是說這個叫做 i 的變數可以用來存放整數 (Integer) 型態的資料。當 C++ 編譯器看到這一行時，就會在程式執行時配置一塊記憶體空間，以存放資料。在這個例子中，存放的就是在第 4 行指定 (assign) 給 i 的 123 。

變數的名稱

- 在前面的範例中，變數的名字只是很簡單的 i，就字面來說，看不出有任何的意義。為了方便閱讀，最好可以為變數取個具有說明意義的名字。舉例來說，如果某個變數代表的是學生的年齡，那麼就可以幫這個變數取像是 age 這樣的名字

變數的命名規則

- C++ 程式中的變數名稱，需符合以下規範：
  - 可使用任何文字（英文字母、中文字都可以）或數字，以及 "_" 符號組成，其他符號都不能使用。
  - 在此要提醒讀者一點，C++ 語言標準雖然允許我們在變數名稱中使用中文(或其他語系的文字)，但一方面某些平台可能未支援此項功能，二來 C++ 語言本身定義的關鍵字及標準函式庫中定義的函式、變數名稱仍為英文，寫程式時若要中文、英文切來切去難免不便，所以建議還是只用英文來替變數命名。
  - 數字不能當成第 1 個字元。例如以下都是不合法的變數名稱：
  - 變數名稱不能重複。真實世界會有人同名同姓，但在程式中則不可有兩個變數使用相同的名稱，否則編譯器會發出重複定義的錯誤訊息。
- 除了以上 2 點，變數名稱也不能和 C++ 語言中的關鍵字 (keyword) 重複。

3-1 資料的種類：資料型別

- 使用任何變數之前，都需要先宣告 (declare)。宣告變數的目的，除了讓編譯器認識這個變數名稱外，另外也是告訴編譯器，這個變數是用來存放什麼類型的資料，這些資料種類，在程式語言中稱為資料型別 (Data Type)。
- 宣告變數的語法如下：

資料型別　變數名稱;

一般常用的資料型態為 1.整數 2. 浮點數 3 字元

1. 整數

用來表示整數值，可以區分為 short、int、long，配置的記憶體長度在不同編譯器上各不相同，可容納的大小各不相同，在 Windows 10 使用 MinGW-w64，GNU 編譯器版本是 8.1.0 型態的話，int 與 long 為 4 位元組，長度越長，表示可表示的整數值範圍越大。

2. 浮點數

用來表示小數值，可以區分為 float、double 與 long double，越後面的型態使用的記憶體空間越大，精度也就越高。

3. 字元

char 的 sizeof(char) 結果要是 1，基本上用來儲存字元資料，但沒有規定什麼是字元資料，也可用來儲存較小範圍的整數。

ch1_2.cpp　stdlib.h

```
1   #include<iostream>
2   using namespace std;
3   #include <stdio.h>
4   #include <stdlib.h>
5   int main()
6   {
7       printf("型態\t\t大小 (bytes) \n");
8       printf("short\t\t%d\n", sizeof(short));
9       printf("int\t\t%d\n", sizeof(int));
10      printf("long\t\t%d\n", sizeof(long));
11      printf("float\t\t%d\n", sizeof(float));
12      printf("double\t\t%d\n", sizeof(double));
13      printf("long double\t%d\n", sizeof(long double));
14      printf("char\t\t%d\n", sizeof(char));
15
16      return 0;
17  }
```

```
D:\course\C++\ch1_2.exe
型態                大小(bytes)
short               2
int                 4
long                4
float               4
double              8
long double         16
char                1

--------------------------------
Process exited after 0.8581 seconds with return value 0
請按任意鍵繼續 . . .
```

使用 printf 時要指定整數、浮點數、字元等進行顯示，要配合格式指定字(format specifier)，以下列出幾個可用的格式指定碼：

%c：以字元方式輸出
%d：10 進位整數輸出
%o：以 8 進位整數方式輸出
%u：無號整數輸出
%x、%X：將整數以 16 進位方式輸出
%f：浮點數輸出
%e、%E：使用科學記號顯示浮點數
%g、%G：浮點數輸出，取 %f 或 %e (%f 或 %E)，看哪個表示精簡
%%：顯示 %
%s：字串輸出
%lu：long unsigned 型態的整數
%p：指標型態

4-1 甚麼是運算式？

- 在 C++ 程式語言中，大部分的敘述都是由運算 式 (Expression) 所構成，至於運算式則是由一組一組的運算子 (Operator) 與運算元 (Operand) 組成。其中，運算子代表的是運算的種類 (或者說是運算符號)，而運算元則是要運算的資料。舉例來說：

- 就是一個運算式，其中 + 就是運算子，代表要進行加法運算，而要相加的則是 5 與 3 這兩個資料，所以 5 與 3 就是運算元。

遞增遞減運算子

在設計程式時，經常會需要將變數的內容遞增或是遞減（加一或減一），例如：

```
i = i + 1;        // 遞增，將 i 加 1 後設為 i 的新數值
j = j - 1;        // 遞減，將 j 減 1 後設為 j 的新數值
```

因此為方便做這類運算，C++ 特別提供了專用的運算子，可以用來代替如上以加減法運算子來將變數加減 1 的敘述。如果您需要幫變數加 1，可以使用 ++ 遞增運算子（Increment Operator）；如果需要幫變數減 1，則可以使用 -- 遞減運算子（Decrement Operator）：

比較運算子

- C++ 語言共有 6 個做為比較用的比較運算子（Relational Operator），其運算結果只傳回 true（整數 1）或 false（整數 0）。其使用法如下：

運算子	使用例	說明
>	i>j	比較 i 是否大於 j
<	i<j	比較 i 是否小於 j
==	i==j	比較 i 是否等於 j
>=	i>=j	比較 i 是否大於或等於 j
<=	i<=j	比較 i 是否小於或等於 j
!=	i!=j	比較 i 是否不等於 j

複合運算子

- 指定運算子還有個特別的功能，它可以和算術運算子或位元運算子結合在一起，使等號左邊的變數可與另一個運算式（或變數）進行算數或位元運算，並將結果再指定給變數，請參見下表：

運算子	使用例	相當於
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
% =	a %= b	a = a%b
>>=	a >>= b	a = a >> b
<<=	a <<= b	a = a << b
|=	a |= b	a = a | b
&=	a &= b	a = a & b
^=	a ^= b	a = a ^ b

如果複合運算子右邊為運算式，仍是先計算該算式，然後再做複合運算子的運算，例如：

```
a *= b + c;          // 相當於 a = a * (b + c);
                     // 而非 a = a * b + c
```

```
int a = 1, b = 1, c = 1, d = 1;
int aplus, bplus;
int cminus, dminus;
aplus = ++a;
bplus = b++;
cminus = --c;
dminus = d--;
printf("%d %d\n", aplus, bplus);
printf("%d %d\n", a, b);
printf("%d %d\n", cminus, dminus);
printf("%d %d\n", c, d);
```

D:\course\C++\3-9.exe

```
2 1
2 2
0 1
0 0
```

```
#include <iostream>
#include<stdio.h>
using namespace std;
int main()
{
    cout<<!(0==0)<<endl;
    cout<<(1!=0)<<endl;
    cout<<((5>2)&&(3>2))<<endl;
    cout<<((2>2)||(3>2))<<endl;

    return 0;
}
```

D:\course\C++\3-

```
0
1
1
1

--------------------
Process exited a
請按任意鍵繼續 .
```

練習 1：輸入攝氏度數，並將華氏度數輸出

練習 2：輸入兩個整數，並求其和，差，積，商

練習 3：輸入兩個浮點數，並求其和，差，積，商

巨集常數

巨集常數是以前置處理指令 #define 來定義，其格式如下：

ch1_2.cpp stdlib.h ch1.cpp

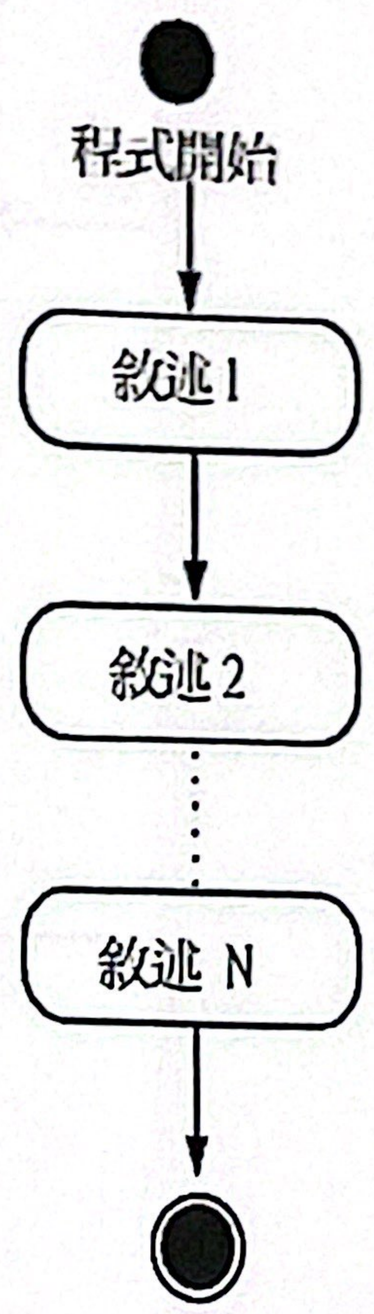
```
1   #include<iostream>
2   using namespace std;
3
4   #define bull(a,b) ((a)+(b))
5   int main()
6   {
7       int a,b,c;
8       a=1;
9       b=2;
10      c=bull(a,b);
11
12      printf("%d\n",c);
13      return 0;
14  }
```

D:\course\C++\ch1.exe

```
3

--------------------------------
Process exited after 0.4641 seconds with return value 0
請按任意鍵繼續 . . .
```

5-1 甚麼是流程控制？

- 在說明流程控制前，先了解何謂流程？以一天的生活為例，早上起床後，先刷牙洗臉，吃完早餐後就出門上課，上完了早上的三堂課，在餐廳吃自助餐，午休後繼續上下午的課...，晚上回宿舍唸書，最後上床睡覺，結束一天的流程。程式的執行也是相同的，如同第 2 章曾提及，程式在執行時是以敘述為單元，由上往下循序進行。如下圖：



- 由右圖不難發現，程式的執行就如同平常的生活一樣，是有順序性地在執行，整個執行的順序與過程，就是流程。

甚麼是流程控制？

- 因此，對於程式執行的流程順序以及因應不同狀況而選取不同的流程，即為流程控制 (flow control)。流程控制可說是電腦程式的靈魂，它包含：條件判斷、迴圈控制及無條件跳躍三大類：

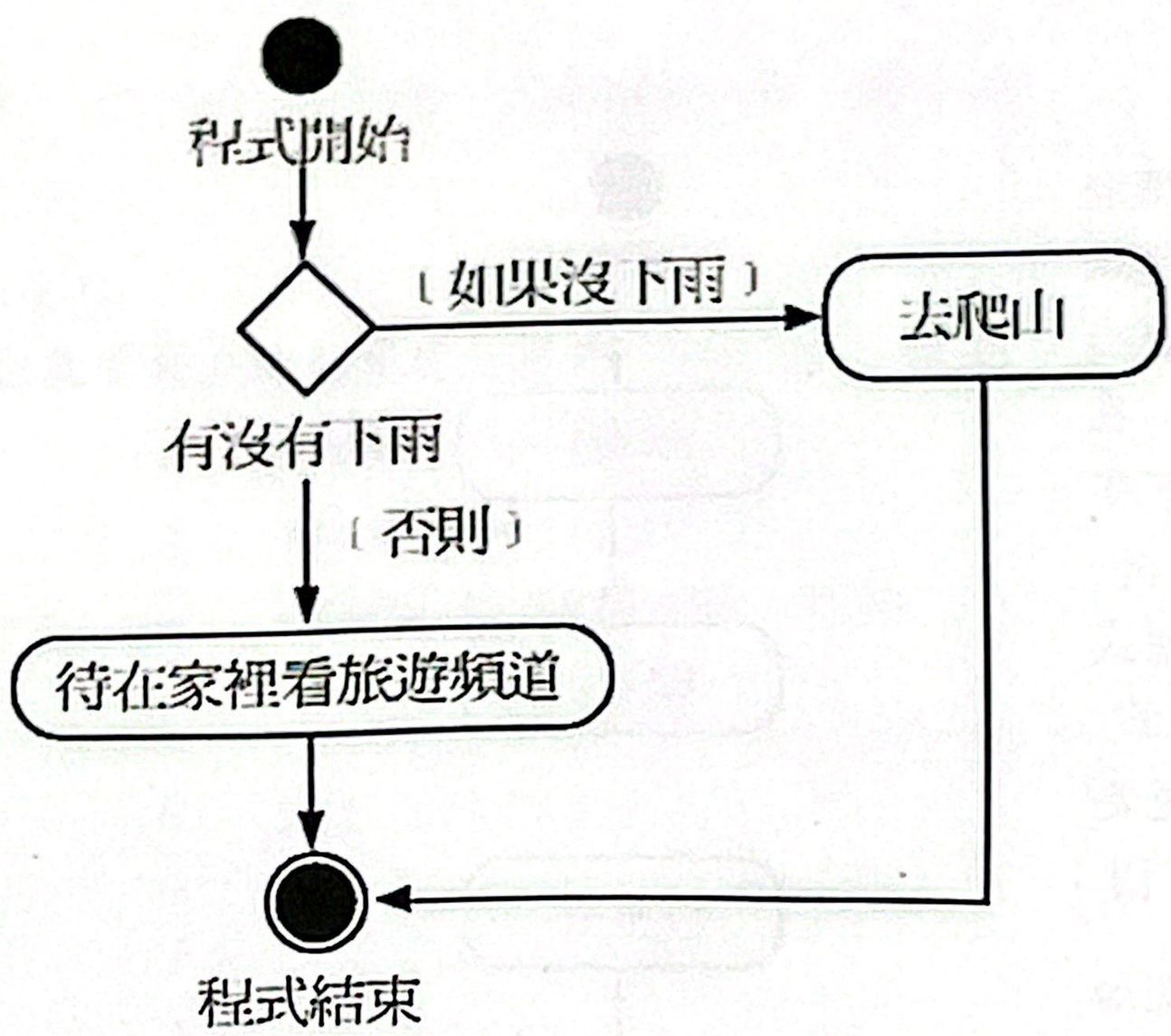
1. 條件判斷控制：判斷條件的真偽，然後程式依真偽的情形至指定的地方去執行程式。C++ 這方面的敘述有：if-else、switch-case 等 2 種。

2. 迴圈控制：程式依指定的條件做判斷，若條件成立則進入迴圈執行迴圈內的動作。每執行完一次迴圈內動作便再回頭做一次條件判斷，直到條件不成立後才結束迴圈，C++ 屬於這方面的流程控制敘述有：for 、while 和 do-while 3 種。

3. 無條件跳躍：當程式執行到無條件跳躍敘述時，程式立即依該敘述的指示跳到目的位置執行，由於無條件跳躍的強制性，容易使我們無法由程式本身看出其前因後果，造成閱讀及偵錯的困難，一般也都盡量不用。

5-2 if 條件分支

- 語法與執行流程
- 多條件運算式與巢狀 if
- 以 else 處理另一種狀況
- 條件運算子



語法與執行流程

- if：如果的意思。會根據條件運算式的結果，來判斷是否執行 { }（稱為區塊，block）中的程式。如果條件運算式的結果為 true，則執行區塊內的敘述；如果結果為 false，則跳過區塊。
- 條件運算式：通常由比較運算或邏輯運算所組成。
- 敘述：條件運算式結果為 true 時所要執行的動作。如果只有單一敘述，則大括號可省略；如果有好幾行敘述，則需用大括號將它們括起來。

巢狀 if

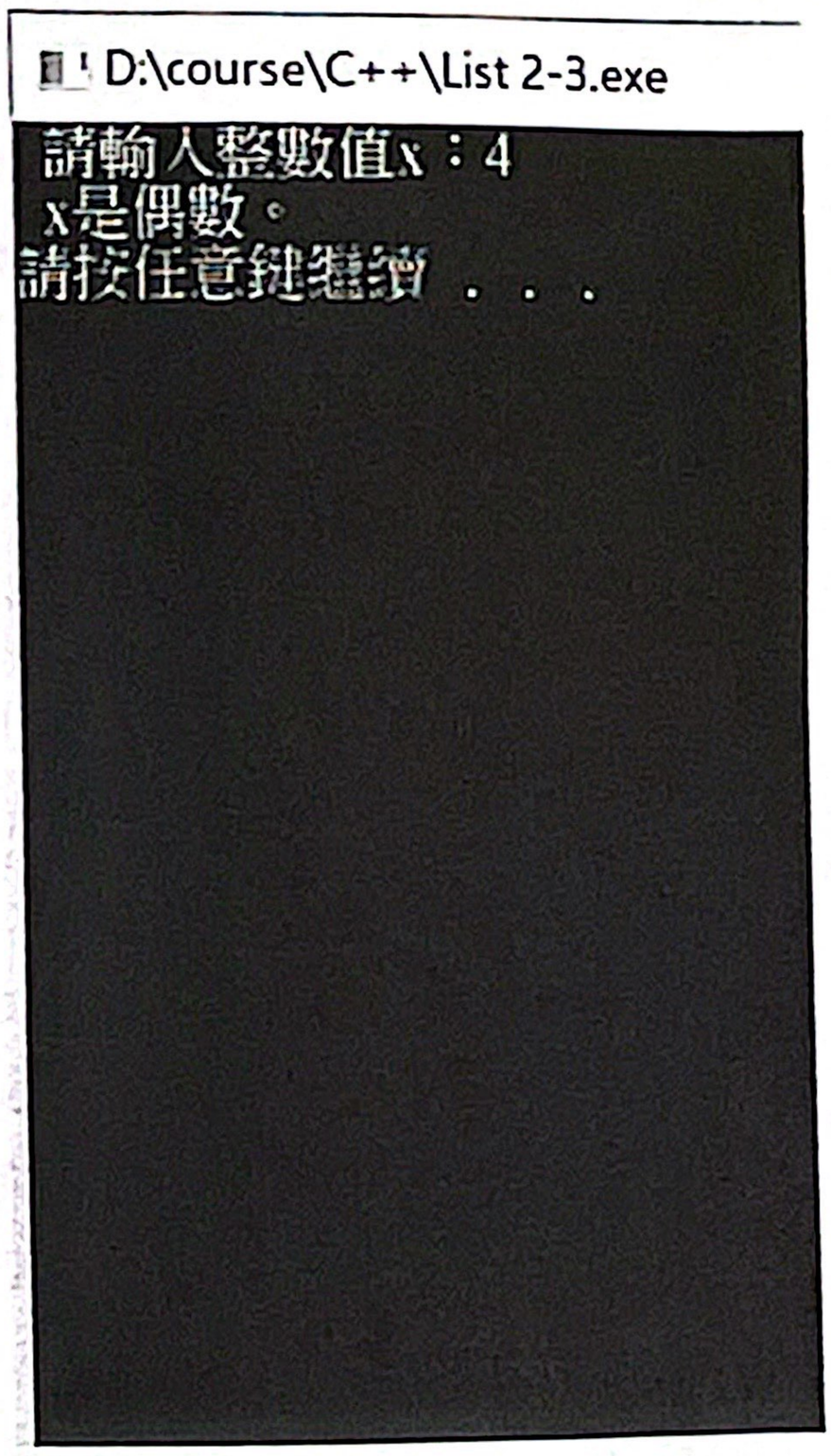
```
if (...) {
    if (...) // 這個 if 敘述是放在另一個 if 敘述的大括號下執行
        ...
}
```

以 else 處理另一種狀況

- if 敘述就像如果...就… 的處理方式，但平常我們也常做另一種形式的選擇：如果...就...，要不然就... 。C++ 當然也提供這種要不然就… 的處理方式，其語法是在 if 敘述後加上 else 子句，如下：

```
if ( 條件運算式 )
    { 動作 1 }        // 條件式運算式爲 true 時，執行動作 1
else
    { 動作 2 }        // 條件式運算式爲 false 時，執行動作 2
```

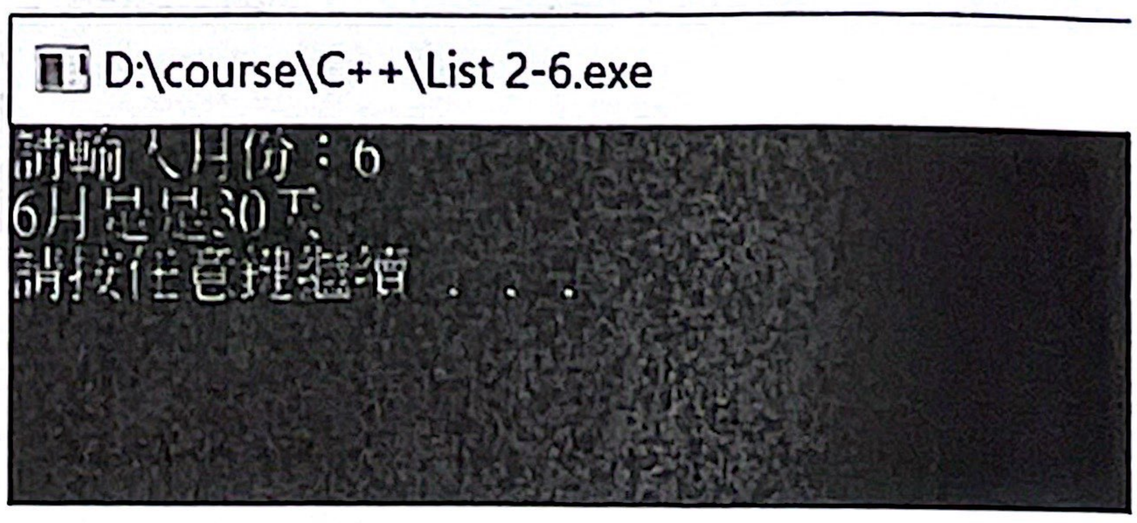
```
/* 判斷輸入的值是奇數還是偶數 */
#include<iostream>
using namespace std;
int main(void)
{
    int x;
    cout<<" 請輸入整數值x：";
    cin>>x;

    if(x%2)
        cout<<" x是奇數。\n";
    else
        cout<<" x是偶數。\n";

        system("pause");
return (0);
}
```



習題：輸入一整數，並判斷是否為正整數，或負整數

```
1  /* 顯示兩個整數值的大小關係(if..else if..) */
2  #include<iostream>
3  using namespace std;
4  int main(void)
5  {
6      int a,b;
7      cout<<"請輸入整數值a：";
8      cin>>a;
9      cout<<"請輸入整數值b：";
10     cin>>b;
11     if(a<b)
12         cout<<" a比b小。\n";
13     else if (a>b)
14         cout<<" a比b大。\n";
15     else
16         cout<<" a與b相等。\n";
17         system("pause");
18 return (0);
19 }
```

```
1  /*  顯示輸入月份的天數 */
2  #include<iostream>
3  using namespace std;
4  int main(void)
5  {
6      int month;
7      cout<<"請輸入月份：";
8      cin>>month;
9      cout<<month<<"月是";
10
11     if(month==2)
12         cout<<"是28天或是29天。\n";
13     else if (month==4 || month==6 || month==9 || month==11)
14         cout<<"是30天。\n";
15     else
16         cout<<"是31天。\n";
17         system("pause");
18 return (0);
19 }
```



習題 1：輸入西元記年，判斷該年是否為閏年

先問 整除 4　X 非閏年

V　再問，整除 100 嗎？　X　是閏年

V　再問，有整除 400 嗎？ V 是閏年

X 非閏年

5-3 switch 多條件分支

- switch 是一種多選一的敘述。舉個例子來說，在本年度初，我們對自己訂了幾個目標，如果年度考績拿到優，就出國去玩；如果拿到甲，就買台電腦犒賞自己；拿到乙，就去逛個街放鬆一下；如果考績是丙，就要罰自己吃兩個月饅頭：

如果年度考績爲

優 → 出國去玩

甲 → 買電腦犒賞自己

乙 → 去逛街放鬆心情

丙 → 吃兩個月饅頭

switch
多條件分支

- switch 多條件分支的用法與上述的情況十分類似，是由一個條件運算式的值來決定應執行的對應敘述，語法如右：

```
switch (條件運算式) {
  case 條件值 1:
    敘述 1
    //其他敘述...
    break;
  case 條件值 2:
    敘述 2
    //其他敘述...
    break;
  .
  .
  .
  case 條件值 N:
    敘述 N
    //其他敘述...
    break;
}
```

- switch：選擇.. 的意思，表示要根據條件運算式的結果，選擇接下來要執行哪一個 case 內的動作。switch 後的條件運算式其運算結果必須是整數或字元（也算是整數），否則編譯時會出現錯誤。

- case：列出個別的條件值，case 之後的條件值必須是常數或是由常數所構成的運算式，且不同 case 的條件值運算結果不能相同。switch 會根據條件運算式的運算結果，從各個 case 中挑選相同的條件值，並執行其下所列的敘述。

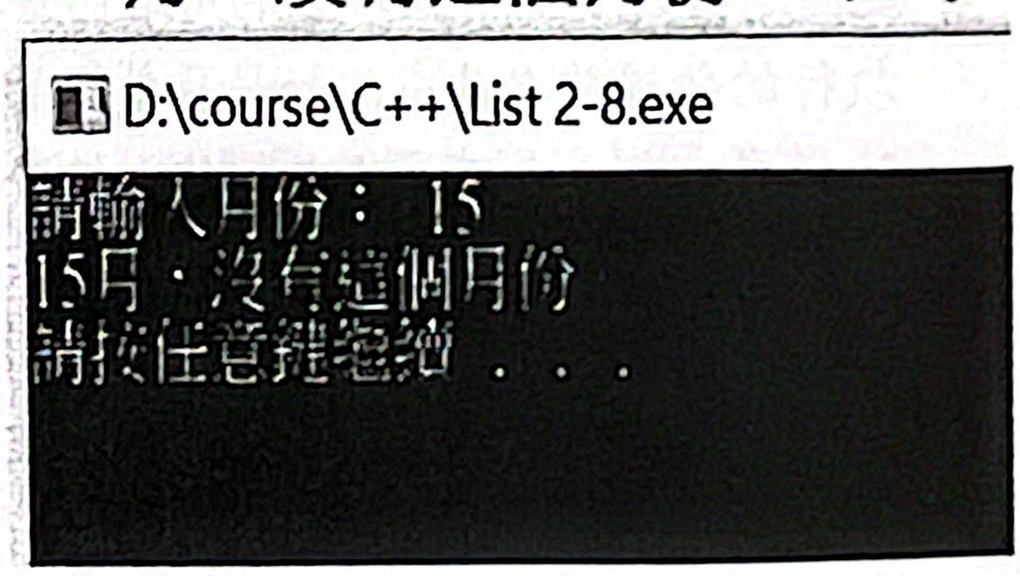
- break：結束這一段 case 的處理動作。

- 雖然 switch 表面上看起來跟使用判斷式的 if 條件運算式完全不同，但是 switch 私底下仍然是使用判斷式的真假來作為其控制流程的機制。

```
7        cout<<"請輸入月份：";
8        cin>>month;
9        switch (month)
10       {
11       case 2: cout<<month<<"月是28天或是29天。\n";
12               break;
13       case 4:
14       case 6:
15       case 9:
16       case 11:cout<<month<<"月是30天。\n";
17               break;
18       case 1:
19       case 3:
20       case 5:
21       case 7:
22       case 8:
23       case 10:
24       case 12:cout<<month<<"月是31天。\n";
25               break;
```

default：若沒有符合的數值或字元，會執行 default 後的陳述句，default 不需要時可以省略。

```
20          case 4:
21          case 6:
22          case 9:
23          case 11:cout<<month<<"月是30天。\n";
24                  break;
25          case 1:
26          case 3:
27          case 5:
28          case 7:
29          case 8:
30          case 10:
31          case 12:cout<<month<<"月是31天。\n";
32                  break;
33          default: cout<<month<<"月，沒有這個月份。\n";
34                  break;
35          }
36          system("pause");
37  return (0);
38  }
```



習題：利用 "switch" 判斷輸入月份為春,夏,秋,冬其中之一

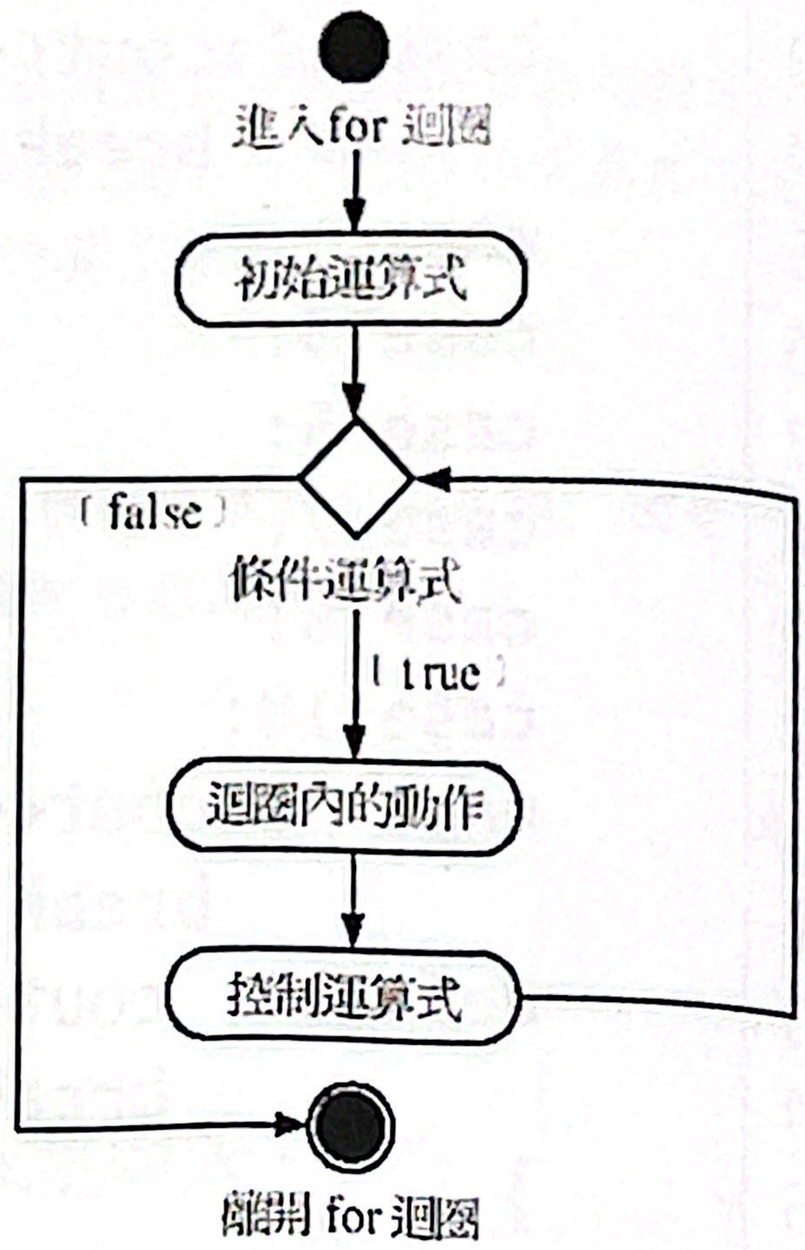
3 4 5春 6 7 8 夏 9 10 11 秋 12 1 2 冬

5-4 迴圈

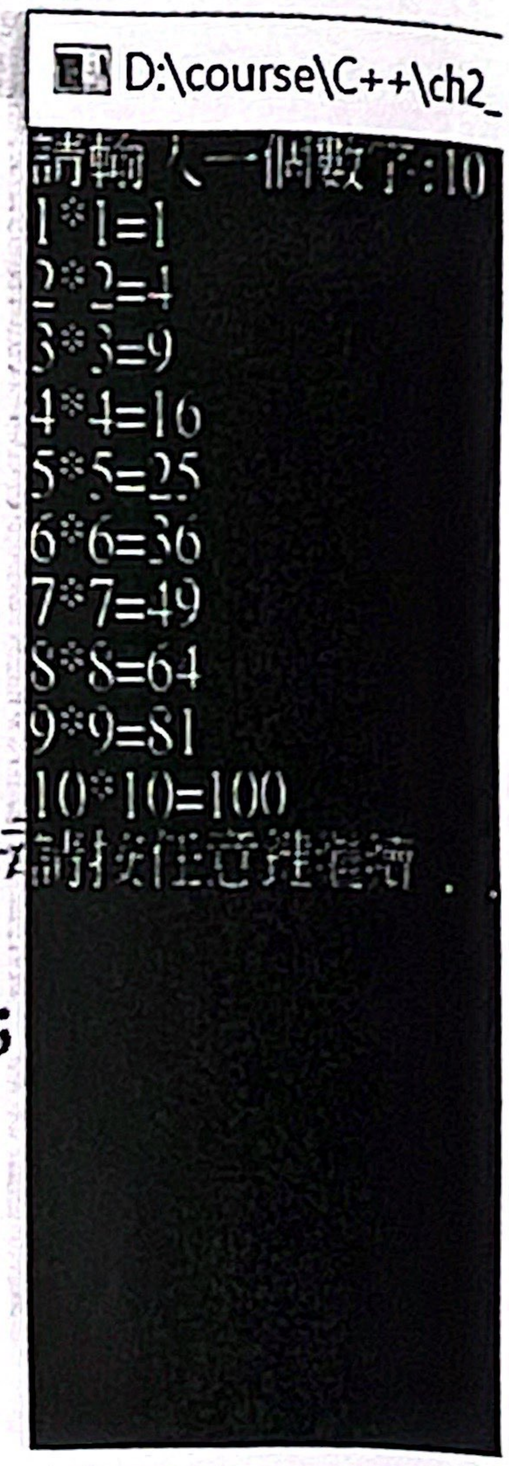
- 迴圈是一種能快速解決重複性工作（重複的執行動作）的敘述。在日常生活中，我們都會遇到一些例行性 (routine) 的工作：辦公人員每天重複的收發表格、操作員重複地把原料放到機器上等。這種重複性的工作即使是在寫程式時也很容易發生，此時我們就需要利用迴圈來解決此類程式問題。
- 簡單的說，迴圈就是用來執行需要重複執行的敘述。例如我們要用程式計算從 1 加到 100，而且每次加完都要顯示目前的累加成果。此時我們不必寫 100 行做加法運算的敘述、100 行 cout 敘述，只需將 2 行敘述放在迴圈中，讓迴圈執行 100 次即可。

for 迴圈

- 初始運算式：在第一次進入迴圈時，會先執行此處的運算式。在一般的情況下，我們都是在此設定條件運算式中會用到的變數之初始值。
- 條件運算式：用來判斷是否應執行迴圈中的動作敘述，傳回值需為布林值。此條件運算式會在每次迴圈開始時執行，以重新檢查讓迴圈執行的條件是否仍成立。比方說，若條件運算式為 "i<10"，那麼只有在 i 小於 10 的情形下，才會執行迴圈內的動作敘述；一旦 i 大於或等於 10，迴圈便結束。
- 迴圈動作敘述：將您希望利用迴圈重複執行的敘述放在此處，如果要執行的敘述只有一個，也可省略前後的大括號 { }。
- 整個 for 迴圈的執行流程圖如下：




```
1   //  顯示1到N的平方值在螢幕上
2   #include<iostream>
3   using namespace std;
4   int main()
5   {
6
7       int i;
8       int N;
9       cout<<"請輸入一個數字:";
10      cin>>N;                          //  由使用者
11      for(i=1;i<=N;i++){
12          cout<<i<<"*"<<i<<"="<<i*i<<endl;
13      }
14      system("pause");
15      return 0;
16  }
```



習題：請估計 1 + 1/2 + 1/3 + 1/4 + …

while 迴圈

- while 迴圈的結構和前面介紹的 if 條件判斷式看起來很類似，兩者都有個用括號括住的條件判斷式，加上一組由大括號括住的的動作敘述。

- while 迴圈有別於 for 迴圈的地方在於，while 迴圈不需要初始運算式及控制運算式，只需要條件運算式即可。如下述：

```
while (條件運算式) {
    迴圈動作敘述
}
```

- while：當.. 的意思。會根據條件運算式的真假，來決定是否執行迴圈內的動作。也就是說當條件運算式結果為真 (true)，就執行迴圈內的動作；若為假，則不執行括號內的敘述（跳出迴圈）。

- 條件運算式：可以為任何運算式、變數或數值。如果結果為真或非 0 的數值，則表示為 true，反之則為 false。

- 迴圈動作敘述：要用迴圈重複執行的敘述，如果要重複執行的敘述只有一個，則可省略大括號。

```
do {
    敘述...                    // 先執行一次迴圈動作
} while (條件運算式);          // 再檢查條件運算式是否成立
```

- 換句話說，不論條件式的結果為何，大括號內的敘述至少會執行一次，這是 do/while 迴圈的特性。

```
4   int main()
5   {
6       int index=1;                    //  這是for迴圈
7       while(index<=10){               //  這是for迴圈
8           cout<<index<<"*"<<index<<"=";
9           cout<<index*index<<endl;
10          index++;
11      }
12
13      system("pause");
14      return 0;
15  }
```

D:\course\C++\ch2_11.exe

```
1*1=1
2*2=4
3*3=9
4*4=16
5*5=25
6*6=36
7*7=49
8*8=64
9*9=81
10*10=100
請按任意鍵繼續 . . .
```

```
11      int index=1;
12      do
13      {
14          cout<<index<<"*"<<index<<"=";
15          cout<<index*index<<endl;
16          index++;
17      }
18
19      while(index<=9);
20
21      system("pause");
22      return 0;
23  }
```

D:\course\C++\ch2_12.exe

```
1*1=1
2*2=4
3*3=9
4*4=16
5*5=25
6*6=36
7*7=49
8*8=64
9*9=81
請按任意鍵繼續 . . .
```

習題1：用 while 計算兩正整數之最大公因數（使用輾轉相除法）

巢狀迴圈

- 在上述的迴圈範例中，我們都是以一維的方式去思考，比如說 1 加到 100 這種只需一個累加變數就能解決的問題。但是如果我們想要解決一個像九九乘法表這種二維的問題（x,y 兩累加變數相乘的情況)就必須將使用迴圈的方式做一些變化，也就是在迴圈中放入另一個迴圈，這種迴圈就稱為巢狀迴圈（nested loops）。

```
8       int i,j;
9       for(i=1;i<=9;i++)              //  外部迴圈
10      {
11          for(j=1;j<=9;j++)          //  內部迴圈
12              cout<<'\t'<<i<<'*'<<j<<'='<<i*j;
13          cout<<endl;
14      }
```

```
D:\course\C++\ch2_10.exe
        1*1=1   1*2=2   1*3=3   1*4=4   1*5=5   1*6=6   1*7=7   1*8=8   1*9=9
        2*1=2   2*2=4   2*3=6   2*4=8   2*5=10  2*6=12  2*7=14  2*8=16  2*9=18
        3*1=3   3*2=6   3*3=9   3*4=12  3*5=15  3*6=18  3*7=21  3*8=24  3*9=27
        4*1=4   4*2=8   4*3=12  4*4=16  4*5=20  4*6=24  4*7=28  4*8=32  4*9=36
        5*1=5   5*2=10  5*3=15  5*4=20  5*5=25  5*6=30  5*7=35  5*8=40  5*9=45
        6*1=6   6*2=12  6*3=18  6*4=24  6*5=30  6*6=36  6*7=42  6*8=48  6*9=54
        7*1=7   7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49  7*8=56  7*9=63
        8*1=8   8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64  8*9=72
        9*1=9   9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
請按任意鍵繼續 . . .
```

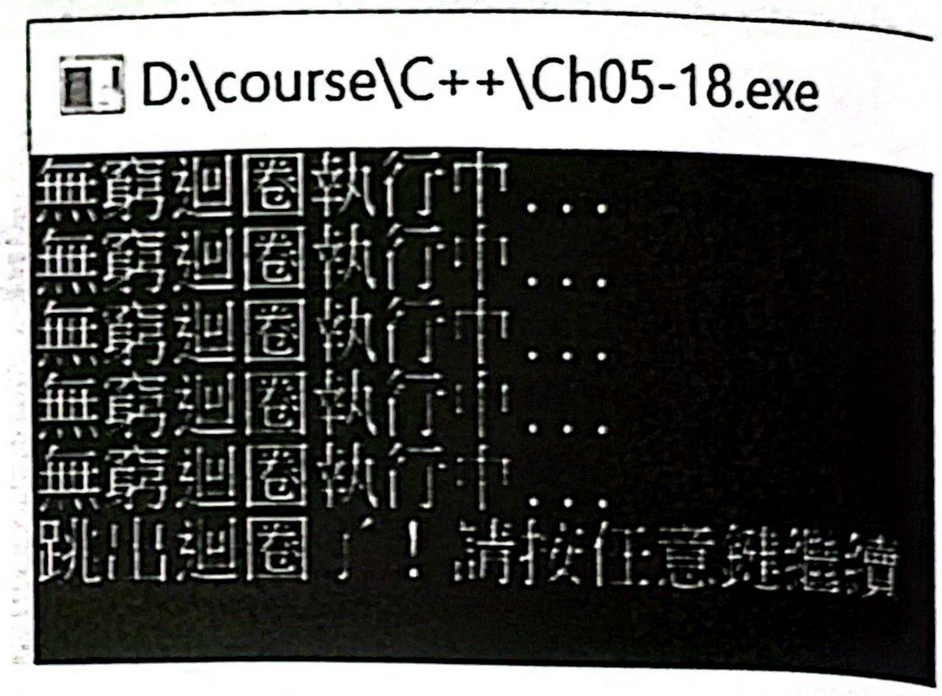
習題 1：請印出菱形的九九乘法表

習題 2：寫一個輸入層數，可以得到一個巴斯卡三角形形狀的程式

5-5 變更迴圈流程的 break 與 continue

- 原則上程式流程在進入迴圈後，都會把迴圈大括號內的所有敘述執行完，之後才檢查條件式是否成立，以決定是否繼續執行迴圈。但某些情況下，我們可打斷這個標準的執行流程，立即跳出整個迴圈（中止執行迴圈），或是跳出該輪迴圈。
- 前面介紹 switch 敘述時，提到要用 break 敘述放在每個 case 後面，讓程式執行流程會跳出該 switch 敘述外。同理，break 敘述也能放在迴圈之中，此時它將會中斷目前的迴圈執行，或者說是跳出迴圈。
- 當程式中遇到某種狀況，而不要繼續執行迴圈時，即可用 break 來中斷迴圈。

```
7   while (i>0) { // 無窮迴圈
8     cout << "無窮迴圈執行中...\n";
9     if (i == 5) // 當 i 為 5 時,
10      break;    // 就跳出迴圈
11    i++;
12  }
13  cout << "跳出迴圈了！";
14    system("pause");
15    return 0;
16 }
```



習題 1：請算出 0～100 的質數

習題 2：請計算 N!

習題 3：請計算第 100 個 Fibonacci 數

習題 4：請估計 1 + 1/4 + 1/9 + 1/16 + …

習題 5：請估計 1 - 1/2 + 1/3 + 1/4 + …

6-1 認識函式

- 隨著程式越寫越大，main() 函式的內容也越來越複雜，當我們在寫較長的程式時，可能會發現程式中有許多段功能相同的敘述會重複出現，而且由於程式的需求，這些程式碼無法被省略，只好重複一寫再寫。本章所要介紹的函式，讓我們能以更簡便的方式，取代會重複用到的一段程式碼，提昇寫作程式的效率，也讓程式的功能模組化。
- 所謂函式（function）就是一組敘述的集合，並且以一個函式名稱來代表此敘述集合。如果這一組敘述代表一項經常用到的功能，我們每次要用到這組敘述時，只要寫下函式的名稱，就是告訴編譯器我們要執行這項功能。
- 舉例來說，如果程式前前後後要計算階乘多次，且不是連續計算，因此不適合用迴圈，這時就變成要讓計算階乘的敘述在 main() 函式中出現多次：

使用函式的好處

- 簡單的說，使用函式的好處包括：
  - 將具有特定功能的敘述獨立成函式，可提高程式的可讀性。
  - 將程式模組化，讓程式碼可重複使用，提升寫程式的效率。
  - 將程式分解成函式，發生錯誤時，可以很容易找出問題在哪一個函式，提高除錯的效率。
- 函式和變數一樣，在使用前必須先定義。定義函式就是定義函式的資料型別、呼叫（call）函式（程式中使用函式的動作就稱為呼叫函式）時所需的參數、以及函式所要執行的動作（敘述）：

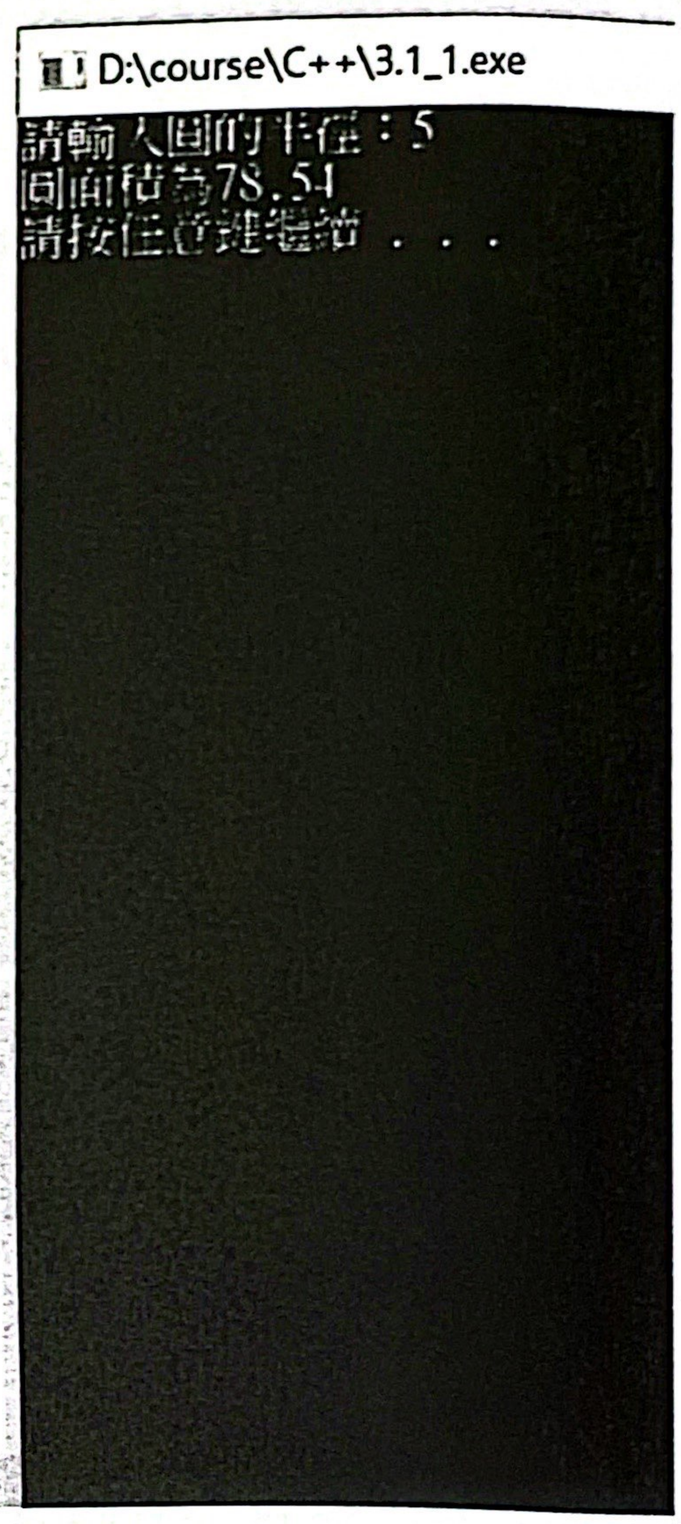
```
型別    函式名稱    (型別  參數1,    型別  參數2,  ...)
{
    敘述的集合...        // 函式內容
}
```

- 型別：函式和變數一樣都有型別，不過函式的型別並不是函式儲存的資料類型，而是函式傳回值的資料類型。函式處理完工作後，可以將處理的結果以傳回值的方式傳回給呼叫它的敘述，讓程式可據以做進一步的處理。要將資料傳回，需使用 return 敘述，例如 “return 0;” 表示傳回 0，若在 return 敘述後放變數或運算式，就表示傳回變數值或運算式的結果。函式也可以沒有傳回值，此時需將函式的型別宣告為 void。
- 函式名稱：函式的命名規則和變數相同，且不可與變數名稱重複。
- 參數：參數就像數學公式中的變數一樣，同一個公式以不同變數值代入計算，即可得到不同的結果。參數就是函式的變數，每次以不同的參數值呼叫函式，即可得到不同的結果。當然我們也可設計沒有參數的函式。此處所列的參數名稱，是在函式中用來代表參數的變數名稱，呼叫函式時，可使用變數、運算式、常數來呼叫。
- 函式本體：大括號的部份就稱為函式本體（function body），在大括號中可放入任何要執行的敘述。

// 求圓面積第一版

3.1_1.cpp

```
3   #include<iostream>
4   using namespace std;
5   #define PI 3.1416
6   int main()
7   {
8       float fRadius;   // 圓的半徑
9       float fArea;     // 圓的面積
10
11      cout<<"請輸入圓的半徑：";
12      cin>>fRadius;
13      if(fRadius<0)
14      {
15          cout<<"錯誤"<<endl;
16      }
17      else
18      {
19          fArea=PI*fRadius*fRadius;
20          cout<<"圓面積為"<<fArea<<endl;
21      }
22      system("pause");
23      return 0;
24  }
```



// 求圓面積第二版

3.1_2.cpp

```cpp
//  求圓面積第二版
#include<iostream>
using namespace std;
#define PI    3.1416
#define ERROR_MESSAGE cout<<"錯誤"<<endl
int main()
{
    float fRadius;  // 圓的半徑
    float fArea;    // 圓的面積
    cout<<"請輸入圓的半徑：";
    cin>>fRadius;
    if(fRadius<0)
    {
        ERROR_MESSAGE;
    }
    else
    {
        fArea=PI*fRadius*fRadius;
        cout<<"圓面積為"<<fArea<<
```

D:\course\C++\3.1_2.exe

```
請輸入圓的半徑：-3
錯誤
請按任意鍵繼續 . . .
```

求圓面積第三版

```cpp
//  求圓面積第三版
#include<iostream>
using namespace std;
#define PI    3.1416
#define ERROR_MESSAGE cout<<"錯誤"<<endl
#define AREA(R)   (PI*R*R)
int main()
{
    float fRadius;  // 圓的半徑
    cout<<"請輸入圓的半徑：";
    cin>>fRadius;
    if(fRadius<0)
        ERROR_MESSAGE;
    else
        cout<<"圓面積為"<<AREA(fRadius)<<endl;
    system("pause");
    return 0;
}
```

D:\course\C++\3.1_3.exe

```
請輸入圓的半徑：5
圓面積為78.54
請按任意鍵繼續 . . .
```

// 求圓面積第四版

[*] ch3.2_1.cpp

```
1   //  如何使用函數
2   //  求圓面積第四版
3   #include<iostream>
4   using namespace std;
5   #define PI 3.1416
6   //  在螢幕上顯示一個錯誤訊息
7   //  參數：無
8   //  傳回值：無
9   void ShowErrorMsg()
10  {
11      cout<<"錯誤"<<endl;
12      return;
13  }
14  //  參數：float fRadius:半徑
15  //  傳回值：float：圓面積
16  float CalcArea(float fRadius)
17  {
18      float fArea=PI*fRadius*fRadius;
19      return fArea;
20  }
21
22  int main()
23  {
24      //程式區塊B
25
26      float fRadius;  //圓的半徑
27      float fArea;    //圓的面積
28      cout<<"請輸入圓的半徑:";
29      cin>>fRadius;
30      if(fRadius<0)
31          ShowErrorMsg();
32      else
33      {
34          fArea=CalcArea(fRadius);
35          cout<<"圓面積為"<<fArea<<endl;
36      }
37      system("pause");
38      return 0;
39  }
```

習題：寫一個函數的梯形面積計算程式 。(輸入:上底、下底、高　輸出：面積)

習題：利用最大公因數的函數，求三個數的最大公因數

6-2 傳遞參數

沒有參數的函式功能有限，因為它的形式固定，能做的事情就少有變化。而透過參數的設計，每次呼叫函式時，就能將不同的變數傳遞給函式，讓函式能針對不同的數值、狀況進行處理，如此函式的執行結果就能有所變化。

C++在呼叫函式時的參數傳遞方式主要有三種：傳值（call by value)、傳址（call by address)、傳參考（call by reference)。

傳值：參數傳遞時的傳值就是只傳送變數的值給函式，這時就如同將變數的值指定給另一個變數，傳遞者與接受者兩個變數彼此各佔有一個記憶體，互不相干。目前為止的範例都屬於這一種。

```
void Function(int a,int b)
{
    a++;
    b++;
    cout<<"In Function:"<<'\t';
    cout<<a<<'\t'<<b<<endl;
}
int main()
{
    int a=5,b=3;
    Function(a,b);
    cout<<"In main:"<<'\t';
    cout<<a<<'\t'<<b<<endl;
    system("pause");
    return 0;
```

D:\course\C++\ch6_12.exe

```
In Function:    6       4
In main:        5       3
請按任意鍵繼續 . . .
```

傳址：就是將變數的記憶體告訴被呼叫的函式，函式的參數列使用指標來儲存這個記憶體位址，由於我們知道了變數的記憶體位址，我們可以直接對這塊記憶體作變動的動作

```
5   void Swap(int *a,int *b)
6   {
7       int c=*a;
8       *a=*b;
9       *b=c;
10      cout<<"In Swap:"<<'\t';
11      cout<<*a<<'\t'<<*b<<endl;
12  }
13  int main()
14  {
15      int a=5,b=3;
16      cout<<"Before Swap:"<<'\t';
17      cout<<a<<'\t'<<b<<endl;
18      Swap(&a,&b);
19      cout<<"After Swap:"<<'\t';
20      cout<<a<<'\t'<<b<<endl;
21      system("pause");
22      return 0;
23
24
25
```

D:\course\C++\ch6_14.exe

```
Before Swap:    5       3
In Swap:        3       5
After Swap:     3       5
請按任意鍵繼續 . . .
```

傳參考：其實它與傳址呼叫是一體的兩面，就我的觀點來說，它可以避免傳址呼叫上的可能語法錯誤

```
5   void Swap(int&a,int&b)
6   {
7       cout<<a<<'\t'<<b<<endl;
8       int c=a;
9       a=b;
10      b=c;
11  }
12  int main()
13  {
14      int a=5,b=3;
15      Swap(a,b);
16      cout<<a<<'\t'<<b<<endl;
17      system("pause");
18      return 0;
19  }
```

D:\course\C++\ch6_17.exe

```
20  5       3
21  3       5
22  請按任意鍵繼續 . . .
```

陣列

變數可以幫我們儲存一筆資料，非常方便；但是要儲存多筆資料的時候就顯得麻煩。
試想要記錄五個人的成績，若用只用變數宣告會寫成
int score1, score2, score3, score4, score5;
score 重複出現好幾次！

語法 - 陣列宣告
長度為 n 的陣列，可用資料為[0] ~ [n-1]， 共 n 個。
宣告時，陣列長度必須是實際數字，如 100、200。不能是變數。
資料型態 陣列名稱[陣列長度];

```
int score[5];
```

```
3   int main(void)
4   {
5       int i;
6       int ary[5];
7       cout<< "請輸入5個整數：\n";
8       for(i=0; i<5; i++){
9       cout<< "No." << i+1 <<":";
10      cin>>ary[i];
11      }
12
13      int sum=0;        //合計值
14      for(i=0; i<5; i++)
15          sum += ary[i];
16      cout<< "合計為" << sum <<"。\n";
17
18          system("pause");
19  return (0);
20  }
```

D:\course\C++\List 3-11.exe

```
請輸入5個整數：
No.1:1
No.2:2
No.3:3
No.4:4
No.5:5
合計為15
請按任意鍵繼續 . . .
```

習題：利用陣列計算五個數的標準差

```
9       int ninzu=0 ;         //未滿60分的學生人數
10      int tensu[6];         //儲存分數的陣列
11      int rakudai[6];       //未達60分學生的足標
12      cout<<" 請輸入6個人的分數： \n";
13      for( i=0 ; i < 6 ; i++)
14      {
15          cout<< i+1 << "號： ";
16          cin>> tensu[i];
17          if( tensu[i] < 60)
18          {
19              rakudai[ninzu] = i;
20              ninzu++;
21          }
22      }
23      cout<< " 未達60分有 " << ninzu << "人。 \n";
24      for( i = 0 ; i < ninzu ; i++)
25          cout<< rakudai[i] + 1 << " 號： " <<
26              tensu[rakudai[i]] << " 分\n ";
27
28          system("pause");
29  return (0);
30
```

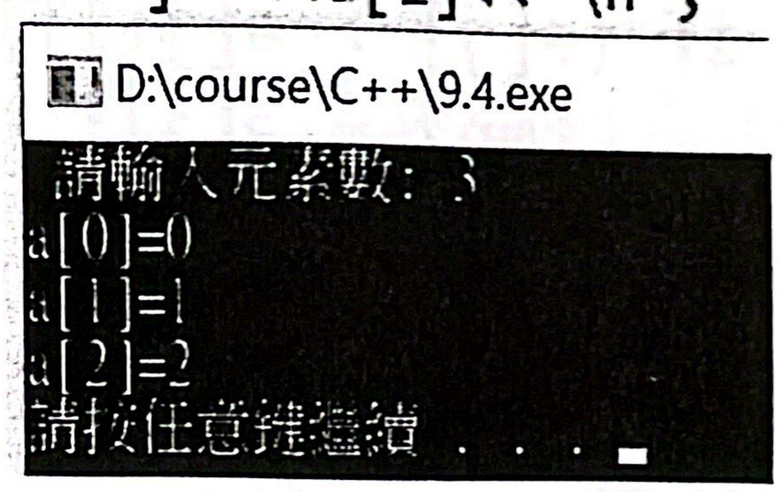
D:\course\C++\List 3-12.exe

```
請輸入6個人的分數：
1號： 58
2號： 69
3號： 70
4號： 50
5號： 88
6號： 45
未達60分有 3人
1 號： 58 分
4 號： 50 分
6 號： 45 分
請按任意鍵繼續 . . .
```

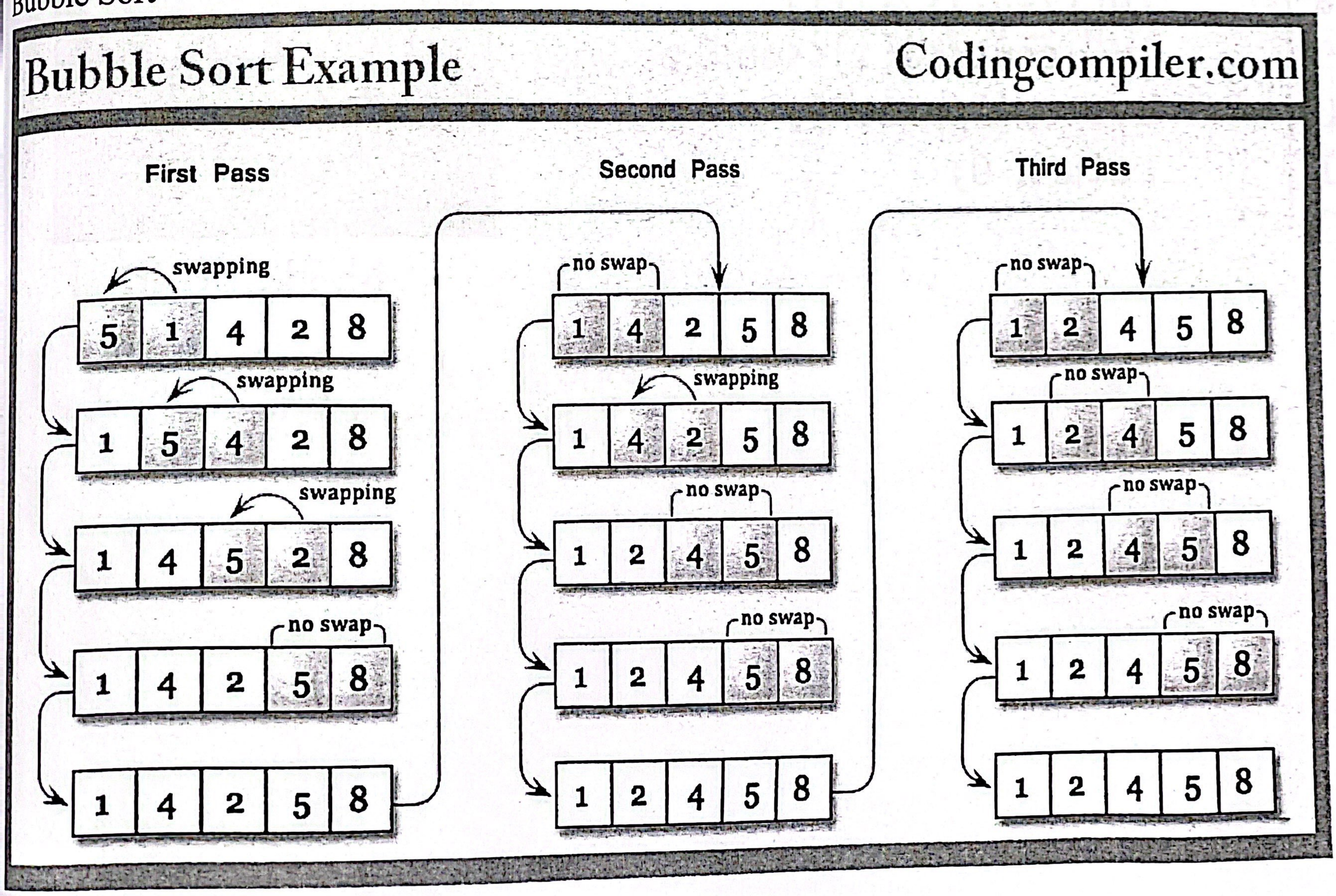
習題：利用陣列，寫一個程式，可輸入五人分數，列出低於平均分的號碼及分數

動態生成陣列

```
    int asize;
    int i;
    int* a;
    cout << " 請輸入元素數: ";
    cin >> asize;
    a=new int[asize];
    for(i=0; i<asize; i++)
        a[i]=i;
    for(i=0; i<asize; i++)
        cout<<"a["<<i<<"]="<<a[i]<<'\n';
    delete[] a;
    system("pause");
    return (0);
}
```



Bubble Sort



bubble sort.cpp

```
4   int main()
5   {
6       int i,j,k, tmp1, tmp2;
7       int a[5]={5,1,4,2,8};
8
9       for(i=0;i<4;i++){
10          for(j=0;j<4-i;j++){
11              if (a[j] > a[j+1]){
12                  tmp2 = a[j];
13                  a[j]=a[j+1];
14                  a[j+1]=tmp2;
15              }
16          }
17      }
18      for(i=0;i<5;i++)
19          cout<<a[i]<<endl;
20      system("pause");
21      return 0;
22  }
```

D:\course\C++\bubble sc

```
1
2
4
5
8
請按任意鍵繼續 . . .
```

記憶體位址

變數三要素

變數位址	變數值	變數名稱
0X0012FF74	2	b

記憶體位址

0X0012FF70	15
0X0012FF74	2
0X0012FF78	39
0X0012FF7C	180
0X0012FF80	67
0X0012FF84	...

從此位址連續寫入 4 個 byte
(因為整數型佔 4 個 byte)

跟記憶體要一塊空間

```
void main(){
    int a = 15;
    int b = 2;
    int c = 39;
    int d = 180;
    int e = 67;
}
```

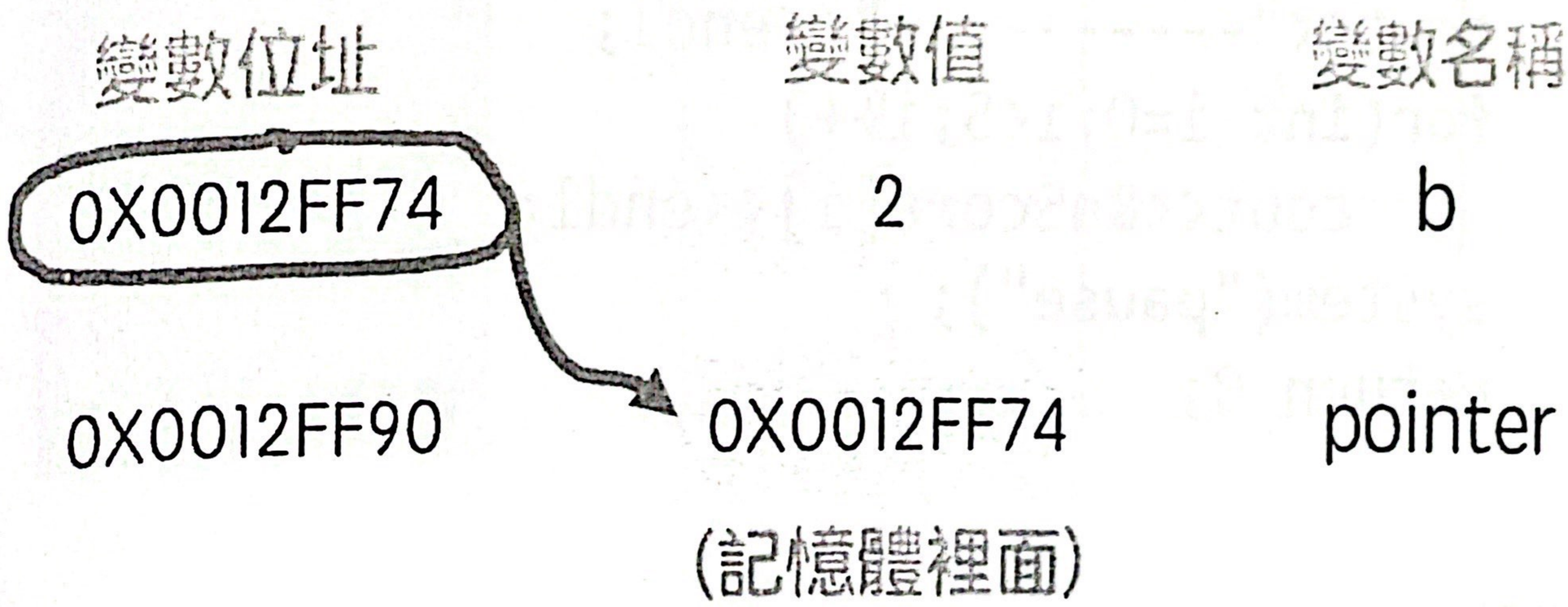
```
8       double a = 15.0;
9       int b = 2;
10
11      cout<<" a 的位址 "<<&a<<endl;
12      cout<<" a 的值   "<<a<<endl;
13      cout<<" b 的位址 "<<&b<<endl;
14      cout<<" b 的值   "<<b<<endl;
15
16
```

D:\course\C++\List 8-5.exe

```
a 的位址 0x6ffe08
a 的值   15
b 的位址 0x6ffe04
b 的值   2
```

指標變數

指標變數 pointer 指向了變數 b




```
int b = 2;
int *pointer;
pointer = &b;
cout<<" b       的位址 " <<&b<<endl;
cout<<" pointer 的位址 " <<&pointer<<endl;
cout<<" pointer 的值   " << pointer<<endl;
cout<<" pointer指向的值 "<<*pointer<<endl;
cout<<endl;
b = 5;
cout<<" pointer 指向的的值" <<*pointer<<endl;
```

D:\course\C++\List 8-5.exe

```
b       的位址 0x6ffe0c
pointer 的位址 0x6ffe00
pointer 的值   0x6ffe0c
pointer指向的值 2

pointer 指向的的值5
```

陣列與指標的關係一

```
{
    int nScore[5];
    cout<<nScore<<endl;
    cout<<"----------"<<endl;
    for(int i=0;i<5;i++)
        cout<<&nScore[i]<<endl;
    system("pause");
    return 0;
```

```
D:\course\C++\
0x6ffe00
----------
0x6ffe00
0x6ffe04
0x6ffe08
0x6ffe0c
0x6ffe10
請按任意鍵繼續
```

陣列與指標的關係二

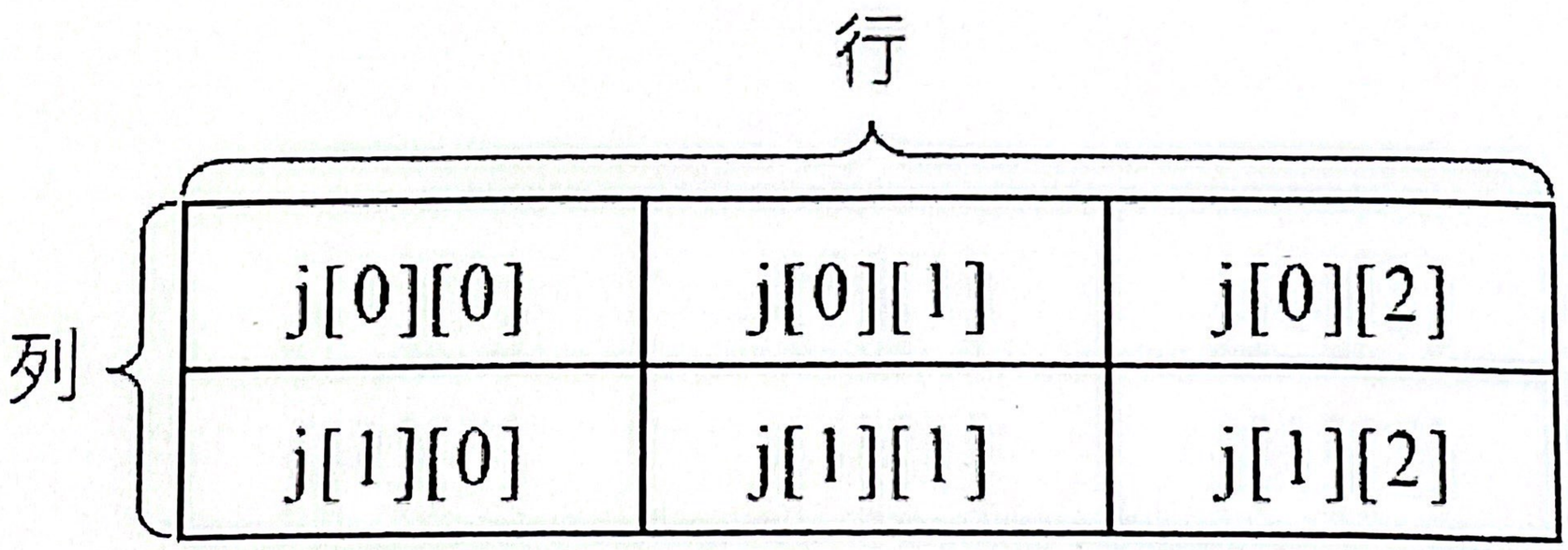
```
{
    int nScore[5]={1,2,3,4,5};
    for(int i=0;i<5;i++){
        cout<<nScore+i<<endl;
        cout<<nScore[i]<<endl;
    }
    system("pause");
    return 0;
}
```

```
D:\course\C++\ch
0x6ffe00
1
0x6ffe04
2
0x6ffe08
3
0x6ffe0c
4
0x6ffe10
5
請按任意鍵繼續
```

習題：利用這個範例用法，改寫 Bubble Sort

二維陣列

int j[2][3];



int j[2][3] = {0, 1, 2, 3, 4, 5};

0	1	2
3	4	5

```
6   void sum(int x[][3], int y[][3], int z[][3])
7   {
8       for(int i=0; i<2; i++)                          //求和
9           for(int j=0; j<3; j++)
10              z[i][j]=x[i][j]+y[i][j];
11  }
12  int main(void)
13  {
14      int x[2][3]={{1, 2, 3},{4, 5, 6}};
15      int y[2][3]={{1, 5, 8},{5, 3, 1}};
16      int z[2][3]={{0, 0, 0},{0, 0, 0}};
17      sum(x, y, z);
18          for(int i = 0; i < 2; i++)     //顯示結果
19      {
20          for(int j = 0; j < 3; j++)
21              cout<< setw(3) << z[i][j];
22          cout<< '\n';
23      }
```

D:\course\C++\twothree.exe

```
  2   7  11
  9   8   7
請按任意鍵繼續 . . .
```

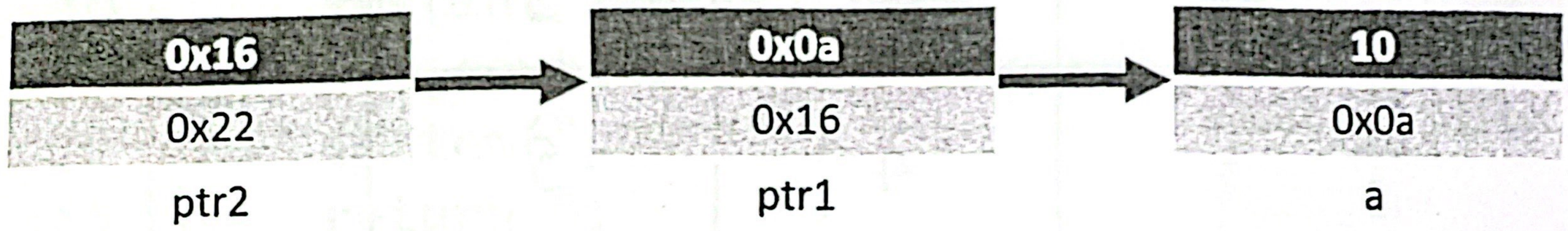
習題：利用 2 維陣列，輸入一個 3x3 陣列，傳入函數，並計算其行列式。

輸入:3x3 矩陣　　輸出:行列式值

雙重指標

那我們宣告一個指標變數，可以去指一個變數，那我們可不可以宣告一個指標變數，去指這個指標，而原本的指標因為指向了一個變數，所以這個指標變數最終也會指向變數。

我們先令一個指標變數 *ptr1 指向變數 a，而我們再宣告一個指標變數 **ptr2，用它來指向 *ptr1，我們最終就可以用**ptr2 指向 a。
假定這個 ptr2 的位址為 0x22，則我們可以讓這個 ptr2 指向 ptr1，最終藉由 ptr1 指到變數 a



雙重指標 ptr2

```
int value = 2;        //宣告整數變數
int *ptr_to_int = &value;   //宣告指向整數的指標
int ** ptr_to_ptr =  &ptr_to_int ;  //宣告指向指標的指標

cout << "   Name        |   Address      |  Value \n";     //印出標題列
cout << "-----------------------------------------------\n";
cout << "ptr_to_ptr   |  " << &ptr_to_ptr << "   |  " << ptr_to_ptr << endl;
                //列印 ptr_to_ptr 的位址與儲存值
cout << "ptr_to_int   |  " << &ptr_to_int << "   |  " << ptr_to_int << endl;
                //列印 ptr_to_int 的位址與儲存值
cout << "value        |  " << &value << "   |  " << value << endl;
                //列印 value 的位址與儲存值
cout << "\n*ptr_to_ptr  |  **********   |  " << *ptr_to_ptr << endl;
                //列印 *ptr_to_ptr 的位址與儲存值
cout << "**ptr_to_ptr |  **********   |  " << **ptr_to_ptr << endl;
                //列印 **ptr_to_ptr 的位址與儲存值
```

D:\course\C++\pointTopointer.exe

```
  Name        |   Address      |  Value
-----------------------------------------------
ptr_to_ptr    |  0x6ffdf8    |  0x6ffe00
ptr_to_int    |  0x6ffe00    |  0x6ffe0c
value         |  0x6ffe0c    |  2

*ptr_to_ptr   |  **********   |  0x6ffe0c
**ptr_to_ptr  |  **********   |  2
請按任意鍵繼續 . . .
```

動態生成二維陣列

```
#include <stdio.h>
//#include <stdlib.h>
#define ROW 3
#define COL 3
int main(void)
{
    int i,j;
    int **a; // 這叫做指標的指標.
    // 配置空間
    a = new int* [ROW];
    for ( i=0; i < ROW; i++ )
    {
        a[i]=new int[COL];
    }
    // 測試各自印下來
    for ( i=0; i < ROW; i++ ){
        for ( j=0; j < COL; j++ ){
            a[i][j]=i+j;
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
    // 把記憶體還回去
    for ( i=0; i < ROW; i++ ){
        delete [] a[i];
    }
    delete [] a;
```

D:\course\C++\ptr_to_ptr1.exe

```
0 1 2
1 2 3
2 3 4
請按任意鍵繼續 . . .
```

傳遞指標的指標

```
#include<iomanip>
#include<iostream>
using namespace std;
//--將2行3列的行列x，y的和儲存到z裡--//
void sum(int **x, int **y, int **z, int R, int C)
{
    for(int i=0; i<R; i++)
        for(int j=0; j<C; j++)
            z[i][j]=x[i][j]+y[i][j];
}
```

```
14  int main()
15  {
16      int i, Row, Column;
17
18      cout<<"請輸入列的維度 ";
19      cin>>Row;
20      cout<<"請輸入行的維度 ";
21      cin>>Column;
22
23      int **x=new int *[Row];
24      for(i=0; i<Row; i++)
25      {
26          x[i]=new int [Column];
27      }
```

```
    for(i=0; i<Row; i++)
        for(int j=0; j<Column; j++)
        {
            cout<<"請輸入y["<<i+1<<"]["<<j+1<<"]= ";
            cin>>y[i][j];
        }

    sum(x, y, z, Row, Column);
```

D:\course\C++\pointertopointer2.exe

```
請輸入列的維度 2
請輸入行的維度 3
請輸入x[1][1]= 1
請輸入x[1][2]= 2
請輸入x[1][3]= 3
請輸入x[2][1]= 4
請輸入x[2][2]= 5
請輸入x[2][3]= 6
請輸入y[1][1]= 7
請輸入y[1][2]= 8
請輸入y[1][3]= 9
請輸入y[2][1]= 10
請輸入y[2][2]= 11
請輸入y[2][3]= 12
  8 10 12
 14 16 18
請按任意鍵繼續 . . .
```

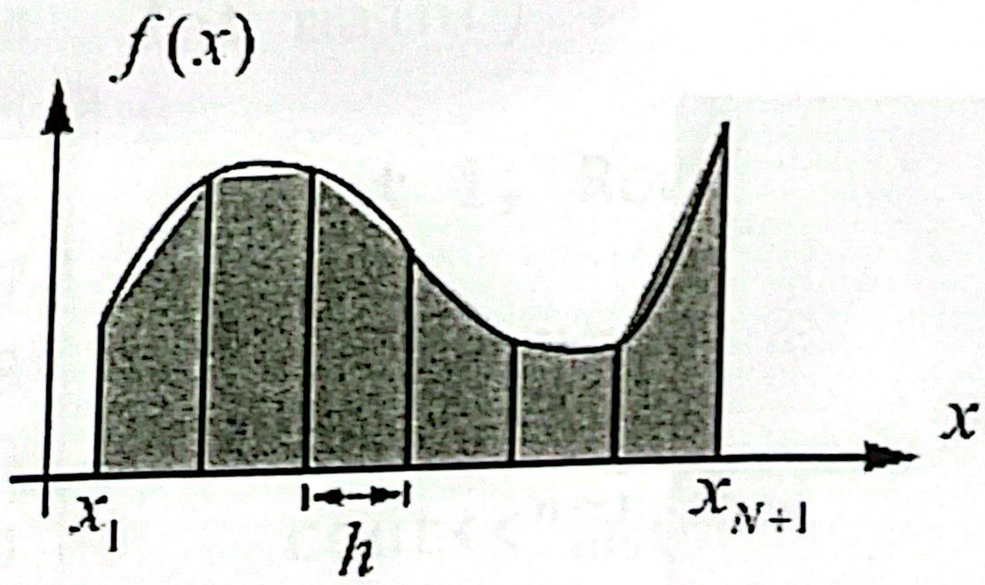
習題：利用動態 2 維陣列，輸入一個 3x3 陣列，傳入函數，並計算其行列式。

習題：寫一個程式，驗證 det(A*B) = det(A)*det(B)。(A,B 為兩方陣)

函數（function）雖然並非資料，然而編譯器（compiler）實際會替函數的程式碼配置記憶體位址，當呼叫函數時，也就是控制權移轉給被呼叫函數的時候，程式執行便會轉移到該記憶體位址的地方。因此，呼叫函數可以直接用指標（pointer），這是 C 語言提供的動態機制，有許多不可取代的地方。

梯型積分公式

$$\int_a^b f(x)dx \approx \frac{b-a}{2n}[f(x_0)+2f(x_1)+2f(x_2)+\cdots+2f(x_{n-1})+f(x_n)]$$




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

double f(double x)     //--- ??f ---//
{
    return (x * x);
}
double g(double x)     //--- ??g ---//
{
    return (x * x * x + 3 * x * x);
}
//--- 使用梯形公式將函數func(x) 分割成n等份從x1積到x2為止的函數 --//
double daikei(double x1, double x2, int n, double (*func)(double x))
{
    double s = 0.0;
    double step = (x2 - x1) / n;
    for (int i = 0; i < n; i++)
        s += ( (*func)(x1 + step*i) + (*func)(x1 + step*(i+1)) ) * step / 2.0;
    return (s);
}
int main(void)
{
    cout << "將f(x)分割成10等份從0.0積到1.0："<< daikei(0.0, 1.0, 10, f) << '\n';
    cout << "將g(x)分割成15等份從1.0積到3.0："<< daikei(1.0, 3.0, 15, g) << '\n';
        system("pause");
return (0);
```

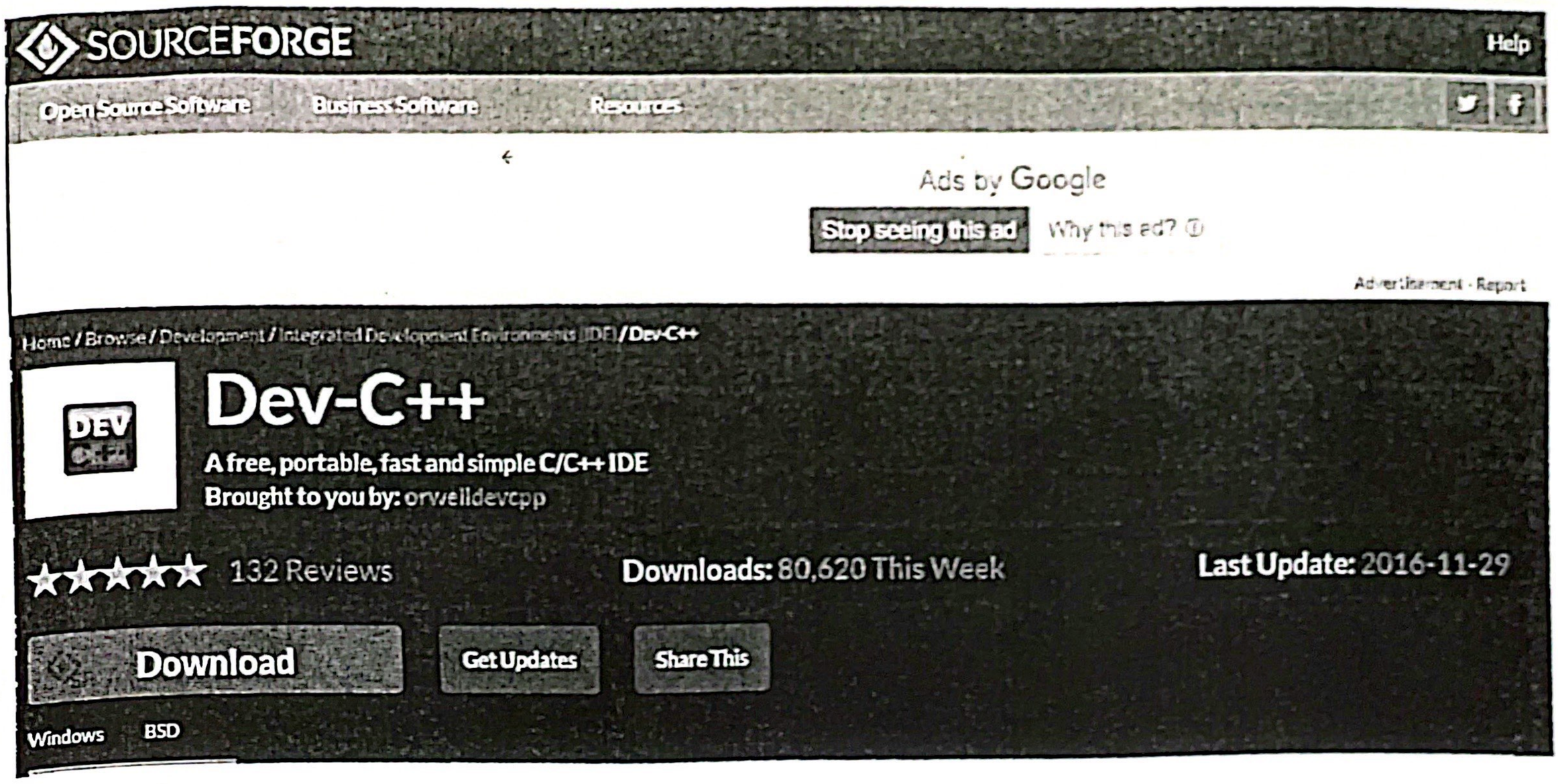
習題：利用"引入指標函數"將梯形法改成辛普森法(Simpson's Method)。

辛普森法公式

$$\int_a^b f(x)dx \approx \frac{\Delta x}{3}[f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+\cdots+2f(x_{n-2})+4f(x_{n-1})+f(x_n)]$$

安裝 **Dev C++**

可以到 **https://sourceforge.net/projects/orwelldevcpp/** 進行下載



本講義請配合以下網頁下載程式範例：

http://hp1.math.nsysu.edu.tw/course/IC/new_page_2.htm