API Design for C++

Introduction

大綱

- 前言
- 甚麼是應用程式介面　(Application Programming Interface)?
- API 設計有甚麼不同？
- 為什麼你應該使用 API?
- 甚麼時候應該避免使用 API?
- API 的層級
- 關於本次實作

前言

- 用 C++ 撰寫大型應用程式，是一個既複雜又困難的任務。
- 穩定、易於使用及維護，可重用 (reusable) 的 C++ 介面，是我們在設計應用程式介面 (Applications Programming Interface, API) 的目標。
- API 主要呈現一個軟體元件的邏輯介面，並隱藏該元件所需的內部細節。
- API 提供模組的一個高層次抽象，並允許多個應用程式共享相同的功能，促進了程式碼的重用。

前言(續)

- 從底層的應用程式框架 (framework)，到資料格式的 API 和圖形使用者介面 (GUI) 的框架，現代的軟體開發高度依賴 APIs。
- 常見的軟體工程術語，例如模組化開發、程式碼重用、元件化、動態連結函式庫或 DLL、軟體框架、分散式運算和服務導向架構等，都需要 API 的設計技巧。
- 常見 C 和 C++ 的 API，包括標準樣板庫 (STL)、Microsoft Windows API (Win32)、Microsoft 的基礎類別 (MFC)、libtiff、OpenGL、OpenCV、MySQL 及 QT 等。
- 許多在 sourceforge.net 網站上的開放原始碼也是 C++ 設計的。
- 故 API 設計是當代軟體開發的一個重要環節。

前言(續)

- 這些 API 用於軟體開發的各個方面，從視窗程式應用、行動運算到嵌入式系統等。
- Firefox 網頁瀏覽器建立在 80 多個動態函式庫上，每個函式庫提供了一個或多個 API 的實作。
- API 設計和標準的應用開發不同的是，變更管理的需求會大很多。
- 變更在軟體開發是必然因素；但 API 是用來提供給不同的應用程式開發者，即使是些微變更，都有可能造成應用程式開發上的問題。
- 良好的 API 設計，首要目標是發行新版本時，即提供客戶所需要的功能，但對應用程式開發造成的影響最小。

前言 – 為什麼要學習 API

- 如果你是一個 API 設計者，而另一個工程師依賴於你寫的 C++ 程式碼，那麼這門課就是為你而設計。
- 介面是你所寫的最重要的程式碼，因為介面問題的修復比實作中的錯誤更費時。例如：介面的改變可能會使所有依賴你程式碼的應用程式，必須全面更新。
- 若只是實作的改變，當客戶端採用新的 API 版本時，便可僅移植函式庫 (實作部份)，輕鬆地整合到他們的應用程式。
- 介面設計不當，則會嚴重降低函式庫的長期生存空間。
- 學習創造高品質的函式庫介面是一個重要的工程技能，也是重點。

前言 – 為什麼要學習API(續)

- 本課程專注於代表 C++ 的模組化介面技術，因此將不會非常深入探討如何實作這些介面背後程式碼的問題。
- 目標對象：
  - 軟體實作工程師和架構師
  - 技術經理
  - 學生和教育工作者

前言 – 專注於 C++

- API 必須以一種特定的程式語言表達，本課程專注在以單一語言 (C++) 設計 API ，而不是抽象化使其適用於所有程式語言。
- C++ 仍然是著重效能的程式碼中最受歡迎的選擇。
- C++ 的特定主題，如樣板、封裝、繼承、命名空間、運算子、 const 正確性、記憶體管理、 STL 的使用及 pimpl 慣用法等，將會被討論。

前言 - 慣例

- 在 API 設計方面，使用者一詞的慣例是指一個軟體開發人員創建應用程式，並使用 API 來實作。
- 本課程使用 .cpp 及 .h 副檔名來識別 C++ 原始碼文件和表頭檔。
- 這邊會使用模組和元件來表示單一的 .cpp 和 .h 檔案組。這些不等同於一個類別，因為一個元件或模組可能包含多個類別。
- 這邊也會使用函式庫 (library) 來表示模組的物理集合，也就是 **函式庫 > 模組 / 組件 > 類別**。

前言 – 慣例（續）

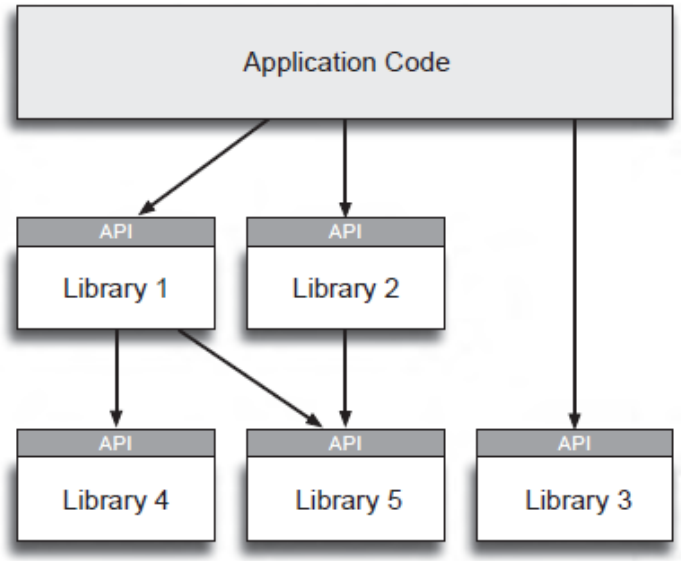
- 方法 (method) 一詞，一般為物件導向程式社群所理解，但嚴格來說不是 C++ 的術語，其相當於 C++ 的成員函數 (member function)。
- 同樣，資料成員 (data member) 是更正確的 C++ 表達，但這裡以視成員變數 (member variable) 為其代名詞。

甚麼是應用程式介面？

- API 是 Application Programming Interface 的縮寫。
- API 提供了一個問題的抽象概念，並指出使用者該如何跟解決這個問題的軟體元件互動。
- 元件通常以一個軟體函式庫 (library) 呈現，使它們能夠在多個應用程式中使用。
- API 可以為自己、公司中的其他工程師、其他公司工程師或為開發社群而寫。
- API 可為單一功能，或涉及到幾百個類別、方法、資料型別 (data type) 、列舉 (enumeration) 和常數。

甚麼是應用程式介面？(續)

- 現代應用程式通常是建立在許多 API 之上，其中一些更進一步依賴在別的 API 之上。
- 下圖說明了一個應用程式直接依賴三個函式庫的 API ，而其中一些 API 更進一步依賴兩個函式庫 API 的例子。

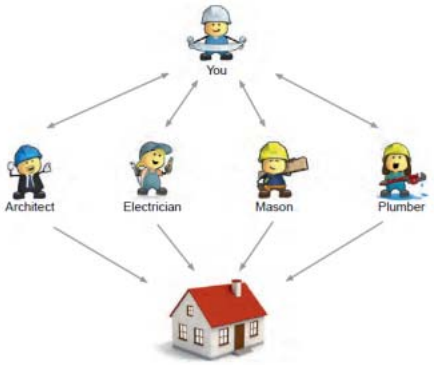


甚麼是應用程式介面？(續)

- 例如：一個影像瀏覽應用程式，可以使用載入 GIF 圖像的 API，而這個 API 本身可能是建立在低階的資料壓縮及解壓縮的 API 上。
- 再次強調，API 的開發在現代軟體開發中無處不在，其目的是提供元件功能的一個邏輯介面，同時隱藏實作細節。
- 例如：載入 GIF 圖像的 API 可能只有提供 LoadImage() 方法，它接受一個檔案名稱後，返回二維陣列的像素值。
- 所有的檔案格式和資料壓縮的細節都隱藏在這個簡單的介面背後。

合約和承包商

- 以自建房屋來說，如果完全靠自己在建房子，需要擁有對建築、水電、木工、泥瓦工和許多其他行業的透徹了解，還必須親自執行每一項任務，並追蹤案子裡各方面的細節。
- 在任何時間點只能執行單一任務，因而完成此專案的時間將會非常冗長。
- 另一個策略如下圖：聘請專業承包商來執行關鍵任務：
  - 建築師：設計房子
  - 木匠：所有木工
  - 水管工人：安裝水管及污水處理系統
  - 水電工人：配置電力系統



合約和承包商（續）

- 和各個承包商協議合約，告訴他們想要他們完成哪些事項，並商議價格，然後他們執行所需要的工作。
- 以這個策略類推，不知要蓋房子的所有相關知識，反而可以採取更高層的監督角色，選擇最佳承包商，確保每個承包的工作組合在一起，會達成所需要的結果。
- 上述的例子，用來比喻 API 是顯而易見的：你要建造的房子，相當於要寫的軟體程式。

  房屋 ←→ 軟體程式

- 承包商提供 API，抽象化需要執行的任務，並隱藏工作所涉及的實作細節。

合約和承包商（續）

- 開發者的任務就轉變成，為應用方式選擇適當的 API ，並把它們整合到你的軟體。
- 免費提供服務的 API ，例如開放原始碼函式庫；對比商用函式庫需要付出許可費 (License) 在軟體中使用。
- 這個比喻甚至可以延伸到承包商僱用一些分包商，相當於某些 API 需依賴其他 API 來執行它們的任務。

甚麼是應用程式介面？(續)

- 應用程式介面 ,API 是一個已定義的介面，提供特定服務給軟體的其他部份。
  - API 為函式庫 (應用程式介面) ，控制硬體或是使用特定軟體功能的介面
  - SDK(Software Development kit) 一般包括使用手冊及開發工具的集合，說明 API 如何呼叫及使用
  - KIT 為工具套件，一般有現成範例或是簡易工具，讓使用者快速實現相關功能

C++ 的 API

- API 只是如何跟元件互動的一個描述；它提供了元件的抽象概念和功能規格。
- 許多軟體工程師會將 API 的縮寫解釋為抽象程式介面 (Abstract Programming Interface)，而不是應用程式介面。

C++ 的 API (續)

- 在 C/C++ ， API 包含在一個或多個標頭檔 (.h) ，加上支援的文件。
- 特定 API 的實作，通常以函式庫 (library) 檔案呈現，並可以鏈結 (link) 到使用者的應用程式。
- 它可能是一個靜態函式庫，如 Windows 上的 .lib 檔案或 Mac OS X 和 Linux 上的 .a 檔。
- 也可能是動態函式庫，如 Windows 的 .dll 檔案、在 Mac 上的 dylib 或 Linux 上的 .so 。

C++ 的 API (續)

- 整理 C/C++ 的 API 一般包含以下：
  - 標頭 (header): .h 標頭檔的集合，定義介面並允許使用者端程式碼用該介面來編譯。開放原始碼的 API 還包括 API 實作的原始碼 (.cpp 檔案)
  - 函式庫 (library): 一個或多個靜態或動態函式庫檔案，提供 API 的實作部份。使用者可以把這些函式庫檔案鏈結到自己的程式碼，增添功能到他們的應用程式
  - 文件：概述資訊用以介紹如何使用 API ，通常包括 API 中所有類別和函式的文件。

C++ 的 API (續)

- 眾所皆知的 API – 微軟的 Windows API (經常被稱為 Win32 API) ，是 C 函式、資料類型、常數的集合，讓程式開發人員能夠編寫在 Windows 平台上運行的應用程式。
- 包含檔案處理、行程 (process) 和執行緒 (thread) 的管理，創建圖形使用介面等功能。
- 注意：Win32 API 是 C API 的例子，但可以從 C++ 的程式直接使用 C API 。
- C++ API 可以用標準樣版函式庫 (Standard Template Library, STL) 來舉例。
- STL 代表的是操作元素集合的一個邏輯介面，不公開每個演算法的任何內部實作細節。

API 設計有甚麼不同

API 是軟體元件的邏輯介面，它隱藏實作所需的任何內部細節

API 設計有甚麼不同(續)

- 再次強調!! 開發人員編寫的程式碼中，介面是最重要的。因為修正介面問題比相關程式碼實作問題的代價更高。
- 共享的 API 開發過程比標準的應用程式或圖形使用者介面 (GUI) 的開發，需要投入更多的心力。
- API 開發關鍵差異因素包括：
  - API 為程式開發人員的一個使用者介面， API 可能被數以千計世界各地的開發人員以未知的方式使用。
  - 多個應用程式可共享 API ，這表示應用程式中的程式碼只會影響該應用程式，但在 API 中的錯誤會影響所有依賴於該功能的應用程式。
  - 當改變一個 API ，必須具備回溯相容。如果介面做了不相容的改變，客戶端的程式碼可能會編譯失敗，或者可以編譯但表現不同或當掉。

API 設計有甚麼不同(續)

- 由於需要回溯相容，關鍵的一點是要有一個變更控制過程。在正常的開發過程中，許多開發人員可能會修復錯誤或增加 API 的新功能，這需要管控。許多開放原始碼的 API ，在加入變更到原始碼前，還需要執行變更請求。
- API 經常會被使用很長時間，產生良好的 API 可能會有很大的前置成本，因為必要的規劃、設計、版本、審查等額外工作。然而，如果做得好，長期的維護成本就可以大幅減輕。
- 寫 API 時，良好的文件說明至為重要，尤其是當沒有實作原始碼時。使用者可以看表頭檔，慢慢的了解如何使用 API ，但是這並沒有定義 API 的行為，例如可接受的輸入值或錯誤條件。
- 自動化測試的需要也同樣重要。

- 寫出良好的 API 很困難。雖然必要的技能仍基於一般的軟體設計原則，但還需要更多的知識和發展過程。然而， API 設計的原則和技術很少教授給工程師。

API 設計有甚麼不同(續)

API 說明了軟體已被其他工程師使用來建立應用程式。正因為如此，它必須是精心設計的、文件化、已迴歸測試，並在發行時是穩定的。

為什麼你應該使用 API

- 如果正在寫一個被其他開發人員使用的模組，無論是要給公司的同事或給外部客戶使用的函式庫，為他們創造一個 API 來存取功能，有以下好處：
  - 隱藏實作：藉由隱藏模組的實作細節，獲得靈活性，在未來的實作變更中，也不會造成使用者的困擾。
  - 提高軟體壽命：避免公佈實作細節，造成程式碼邏輯扭曲、互相糾纏，且難以閱讀。
  - 促進模組化：一個 API 通常是設計來解決特定的任務或使用情況，在一組 API 集合上開發應用程式，模組的行為不依賴另一個模組的內部細節，是為模組化架構。

為什麼你應該使用 API(續)

- 減少重覆程式碼：重覆的程式碼是軟體工程的大罪之一。讓所有程式碼的邏輯及實作細節放在介面的後面，使用者端必須使用介面，就會把行為集中在一起。
- 易於改變實作：如果已經把模組的所有實作細節都隱藏在公共介面的背後，計可以改變實作細節而不影響任何依賴 API 的應用程式碼。
- 易於優化：因為實作細節已隱藏，就可以優化 API 的效能而不必更改使用者端的程式碼。

程式碼重用

- 程式碼重用是利用現有的軟體建立新的軟體。
- 再次強調!! 這是現代軟體開發的聚寶盆。API 提供了可以讓程式碼重用的一種機制。
- 在最初的軟體開發，公司為自家生產的應用程式編寫所有的程式碼是常見的。
- 如今，優良的商業和開放原始碼函式庫激增，可以簡單地重用別人寫的程式碼。
- 基本上，軟體開發已經變得更加模組化，我們可以使用不同的元件，像堆積木般堆積成應用程式，並且透過它們公開的 API 來交流。

程式碼重用(續)

- 再次強調!! 你不需要了解每個軟體元件的細節，如先前蓋房子的比喻，你可以委派這些細節給專業承包者，這樣做會加快開發週期。
- 專注於自己的核心業務邏輯，而不是重新打造基礎。
- 實現程式碼重用的困難之一，就是得做到更泛用的介面。這是因為其他客戶可能有額外的期望或要求。
- 有效的程式碼重用，源自於對客戶的深刻瞭解，以及設計一個系統可以顧及自己與客戶的共同利益。

平行開發

- 即使是內部使用的軟體，你的同事可能需要使用你的程式碼來寫別的程式。
- 使用良好的 API 設計技術，可以令開發人員在相互依賴的程式碼中平行工作。
- 有效率利用時間的方法是，商定一個適當的 API ，然後可以讓佔位 (placeholder) 的功能代替該 API 的實作，讓對方可立即呼叫。
- 在現實中，很可能會有在寫程式碼之前沒有預料到的介面問題， API 可能要來回幾次才會弄對，但此方式依然可以在最省時間的情況下平行工作。

甚麼時候應該避免使用 API

- 許可證的限制：
  - 有個 API 可能提供了所需的功能，但許可限制造成無法使用。例如，在 GNU 通用公共許可證 (GPL) 下發布的開放原始碼套件，這表示在程式中使用此套件，就需要釋出任何相關的延伸作品，對商用程式來說無法被接受。
  - GNU 寬鬆通用公共許可證 (LGPL) 較自由，常見於軟體函式庫中。
  - 採用商業用函式庫價格過高，或是許可條款過於嚴格，例如要求每個使用者支付許可費用。

甚麼時候應該避免使用 API(續)

- 功能不匹配：
  - 無法符合應用程式的限制或功能上的要求。
  - API 無法在支援的平臺上工作，或者是不符合應用程式要求的效能標準。
- 缺乏原始碼：
  - 函式庫有錯誤將無法檢查。
  - 需要花時間等待該 API 擁有者來檢查錯誤，軟體專案的進度將受影響。
- 缺乏文件

API 的層級

- API 可以是任意大小，從單一功能到很大的類別集合。從底層的作業系統呼叫到 GUI 工具包。以下列出各種常見的 API ：
  - 作業系統 (OS) 的 API: 每個作業系統都必須提供一套標準的 API ，允許程式存取作業系統級的服務，如 Win32 的 CreateProcess() 等，這必須是穩定的底層 API 。
  - 語言的 API: C/C++ 等都提供各自的標準函式庫。
  - 圖像 API: 實作讀寫圖檔，[illegible]體套件， libjpeg 、 libtiff 以 libpn

```
TIFF *tif = TIFFOpen("image.tiff", "r");
if (tif)
{
    uint32 w, h;
    TIFFGetField(tif, TIFFTAG_IMAGEWIDTH, &w);
    TIFFGetField(tif, TIFFTAG_IMAGELENGTH, &h);
    printf("Image size  %d x %d pixels\n", w, h);
    TIFFClose(tif);
}
```

API 的層級 (續)

- 三維圖形的 API: OpenGL 和 DirectX 是兩個經典的 3D 圖形 API ，可定義基本圖形的 3D 物件，下面程式碼顯示 OpenGL API 用不同顏色畫出三角形的每個頂點。

```
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_TRIANGLES);
   glColor3f(0.0, 0.0, 1.0); /* blue */
   glVertex2i(0, 0);
   glColor3f(0.0, 1.0, 0.0); /* green */
   glVertex2i(200, 200);
   glColor3f(1.0, 0.0, 0.0); /* red */
   glVertex2i(20, 200);
glEnd();
glFlush();
```

API 的層級 (續)

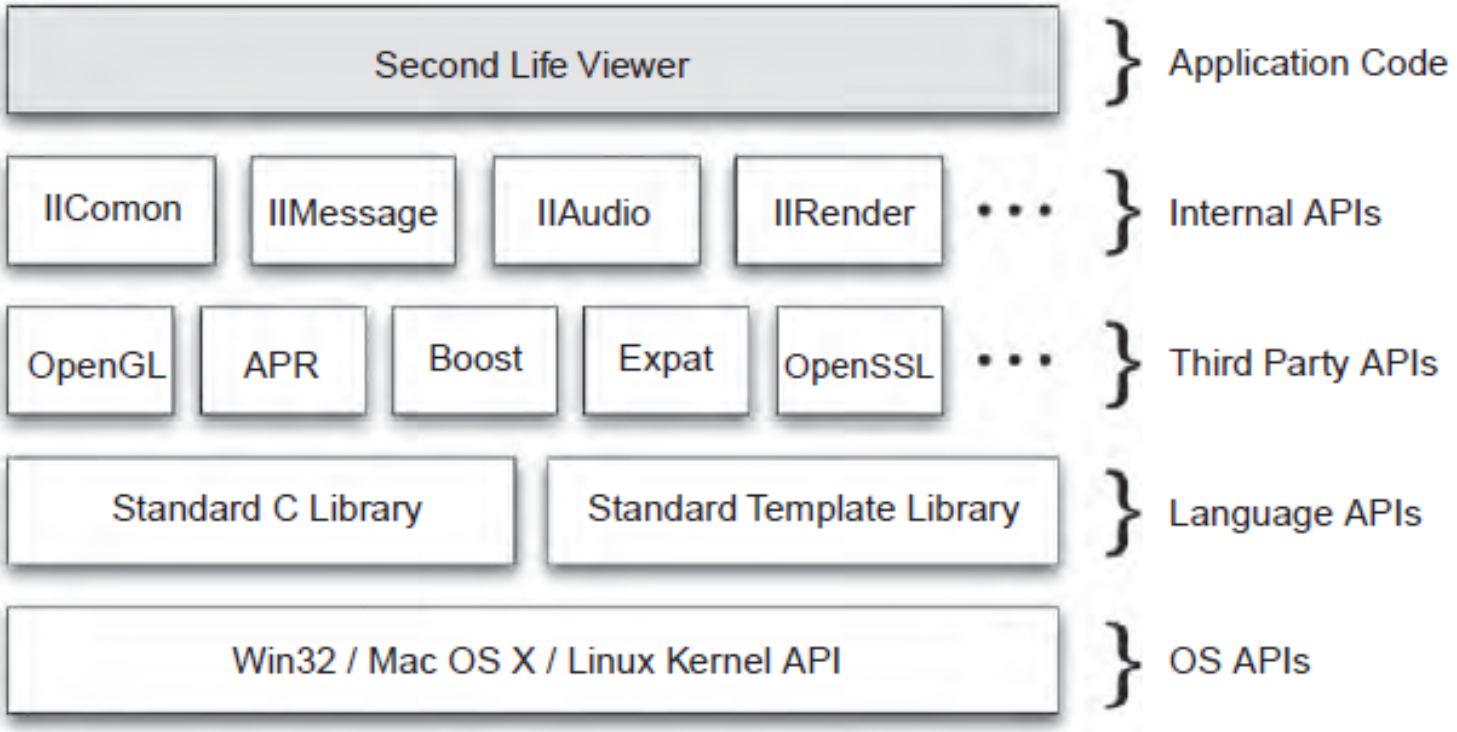
- 圖形使用者介面 API: 任何要開啟視窗的應用程式，需要使用 GUI 工具，它能夠創建視窗、按鈕、文字框、對話框、圖標及選單等。 這 API 通常還提供事件模型，捕捉滑鼠和鍵盤事件。 底下以跨平台的 QT 程式，來說明簡單的 GUI API 範例。

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QPushButton hello("Hello world!");
    hello.resize(100, 30);
    hello.show();
    return app.exec();
}
```

API 的層級 (續)

- 實際案例：



API 和 SDK

- SDK 是安裝在一個特定平臺的軟體套件，會使用其中一個或多個 API 來建立應用程式。
- SDK 會包括表頭檔讓你可以編譯程式，和函式庫檔案，提供的 API 實作讓你可以鏈結到應用程式。
- 一個 SDK 也可能包含其他資源，用以幫助使用者應用 API，如文件、範例的原始碼，以及支援工具。
- 舉例：Apple 提供了 iPhone SDK，其中包含實作各種 iPhone API 的框架 (表頭檔和函式庫)。編譯和鏈結這些檔案可得到 API 背後的實作功能。
- iPhone SDK 還包含 API 文件、範例程式碼、 XCode(IDE) 和 iPhone 模擬器，可以在 Windows 上運行 iPhone 應用程式。

關於本次實作

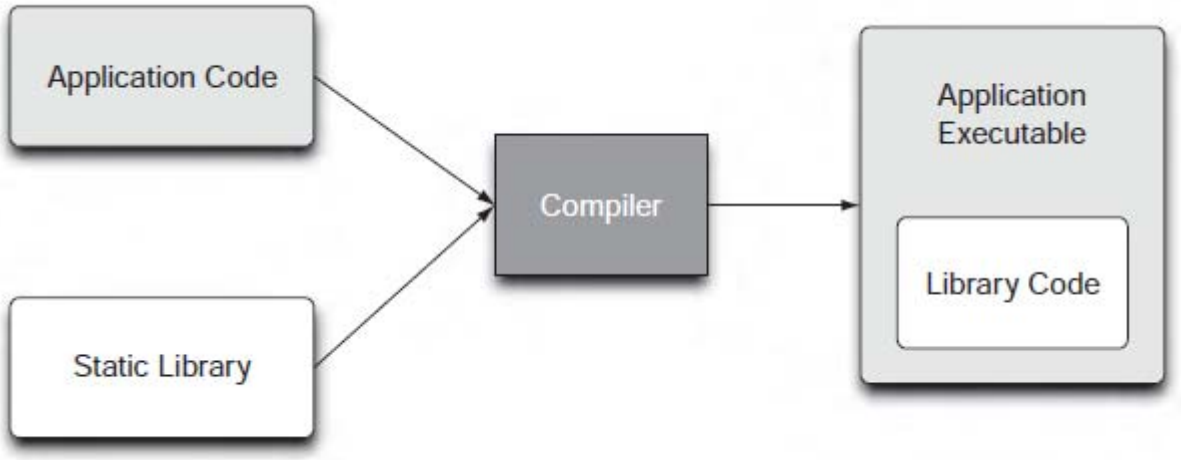
- 本次上課已經涵蓋 API 設計的基礎知識，以及 API 開發的利弊。
- 再來將深入到更細節，例如：如何設計良好的 API 、如何用 C++ 實作出高效率的 API ，和如何改版而不破壞其回溯相容性。
- 如何定義 API 的品質：
  - 品質：良好 API 的定義，這將涵蓋在設計 API 時應該要知道的品質，如資料隱藏、最小完整性和鬆散耦合，將以許多 C++ 原始碼說明這個概念。

函式庫的建立

- 函式庫 (Library) 讓你打包 API 實作編譯後的程式碼和資料，使用者可以嵌入到自己的應用程式中。
- 如先前所提及，函式庫是模組化的工具。
- 以下將說明 API 設計的實體方面，即在函式庫檔案的符號導出表 (symbol export table) 公開 API 函式的公共符號。
- 函式庫的特性、用法和支援工具本質上是特定於平台。在 Windows 運用動態連結庫 (DLL) 方式與在 UNIX 上動態共享物件 (DSO) 的工作方式是不同的。

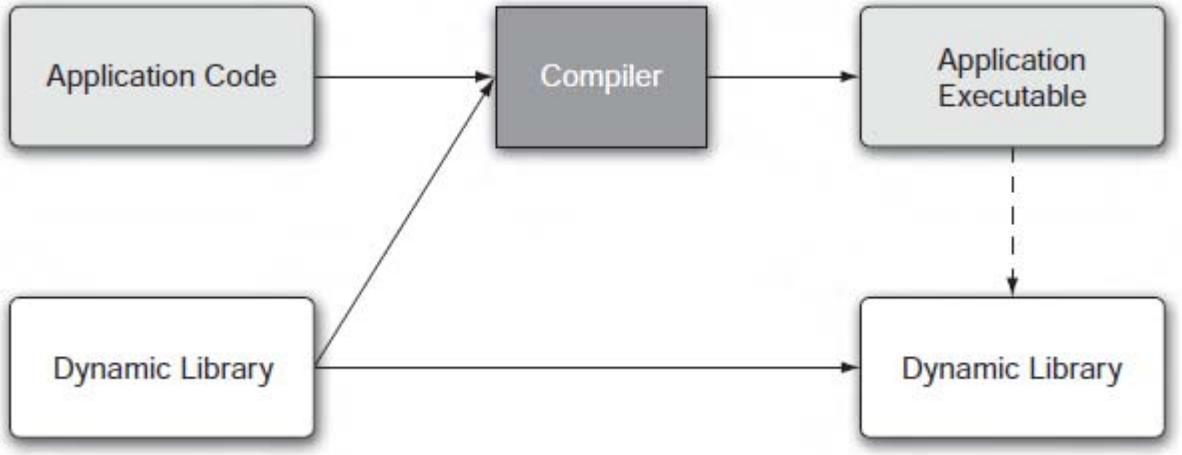
函式庫的建立(續)

- 函式庫有兩種主要的型式：靜態與動態函式庫。
- 靜態函式庫包含目的碼 (object code)，與使用者應用程式連結，然後成為可執行檔的一部份，下圖說明此一概念：



函式庫的建立(續)

- 靜態函式庫的本質是包裝已編譯的目的檔，在 Windows 上是 .lib ，例如 libjpeg.a 或 jpeg.lib 。
- 動態函式庫是在編譯時連結，以解決未定義的參考檔案，然後和使用者的應用程式一起發布，讓應用程式可以在執行時才加載函式庫。



函式庫的建立(續)

- 動態函式庫有時被稱為共享函式庫，因為它們可以由多個程式共享，在 Windows 上是 .dll ，例如 jpeg.dll 。
- 在 Windows 上，靜態函式庫以 .lib 檔案表示，而動態函式庫有 .dll 擴展檔名，此外每一個 .dll 檔必須伴隨著一個導入函式庫的 .lib 檔案。
- 導入函式庫是用來解決在 .dll 中導出的符號引用。
- 例如，在 Win32 使用者介面 API 的實作 user32.dll 附帶 user32.lib 。
- 注意!! 雖然靜態函式庫和導入函式庫使用相同的 .lib 檔名，它們實際上是不同的檔案類型。

函式庫的建立(續)

- 在 Windows 中，有幾個其他的檔案格式實際上是實作為 DLL，包括：
  - ActiveX 控制檔案 (.ocx)
  - 設備驅動檔案 (.drv)
  - 控制面板檔案 (.cpl)

導入和導出函式

- 如果想在 Windows 上呼叫 DLL 的函式，就必須使用以關鍵字明確的標示其宣告：

```
__declspec(dllexport)
```

- 例如：

```
__declspec(dllexport) void MyFunction();
class __declspec(dllexport) MyClass;
```

導入和導出函式

- 在建構 API 時，經常採用預處理器巨集來作導出宣告，但在應用程式中使用這相同的 API 是導入的修飾詞。下面提供預處理的範例程式碼：

```
// On Windows, compile with /D " EXPORTING" to build the DLL
#ifdef  WIN32
#ifdef  EXPORTING
#define DECLSPEC   declspec(dllexport)
#else
#define DECLSPEC   declspec(dllimport)
#endif
#else
#define DECLSPEC
#endif
```

- 然後，可能宣告所有想從 DLL 導出的符號，如下：

```
DECLSPEC void MyFunction();
class DECLSPEC MyClass;
```

導入和導出函式

- DLL 可以提供一個選擇性的進入點函式，在執行緒或程序加載 DLL 時，初始化資料結構，或是當 DLL 卸載時清理記憶體，這是由函式 DllMain() 管理。
- 如果進入點函式返回 FALSE ，會被認為是一個致命錯誤，應用程式將無法啟動，以下提供 DLL 進入點的範例：

```
BOOL APIENTRY DllMain(HANDLE dllHandle,
                      DWORD reason,
                      LPVOID lpReserved)
{
   switch (reason)
   {
   case DLL_PROCESS_ATTACHED:
      // A process is loading the DLL
      break;
   case DLL_PROCESS_DETACH:
      // A process unloads the DLL
      break;
```

在 Windows 上建立函式庫

- 以下步驟描述如何在 Windows 中建立一個靜態函式庫
  - 選擇選單 File -> New -> Project
  - 選擇 Visual C++ -> Win32 的選項和 Win32 Project 圖示
  - 出現 Win32 應用程式精靈在應用程式類型下選擇靜態函式庫選項

在 Windows 上建立函式庫

- 以下步驟描述如何在 Windows 中建立一個動態函式庫
  - 選擇選單 File -> New -> Project
  - 選擇 Visual C++ -> Win32 的選項和 Win32 Project 圖示
  - 出現 Win32 應用程式精靈在應用程式類型下選擇 DLL 選項