He is rich enough that wants nothing.

– G. Herbert

無慾求，便是富足。

– 赫爾巴特–

26

多檔案程式的建立與圖書管理系統範例

各節標題

本章導讀

這一章是本書內容的總結，在這一章裡，將從無到有地示範建立**圖書管理系統**的完整流程。從這個系統的建立流程，將示範物件導向系統的系統分析、設計與建構的大略過程。讓各位體會一下，我們是如何利用物件導向觀念與技術，尋求問題的解答。雖然這個過程距離嚴謹的系統建構流程，仍有一段距離，但建構系統時的重要觀念卻仍具體地呈現出來。另外，在建立系統前將告訴您，如何建立一個多檔案程式，這對於程式的分工與程式架構的釐清上有相當的幫助。

26-1 建立多檔案程式 進階

為何要建立多檔案程式

對於一個較大的程式，也許是一個系統或者一套軟體，程式的開發可能由許多不同的程式設計師共同合作，有人負責螢幕的顯示，有人負責將資料的存取，還有人負責資料的運算。此時，透過多檔案程式的建立，有助於劃分出這些程式間的介面，以及各程式設計師的分工，這對於程式開發的分工合作是相當有助益的。

除了以團隊方式開發程式時，需要藉助多檔案程式的建立，方便程式開發的分工外。當一個人開發程式時，建立多檔案程式，可將不同功能的類別儲存在不同檔案裡，除方便管理程式外，亦可幫助您釐清程式之組織與架構。當程式稍具規模後，建議將其建立為多檔案程式。

如何建立一個多檔案程式

既然建立多檔案程式有這麼多的好處，但是建立多檔案程式的方法，卻並不難。在 13-2 節，已完成運用 Dev C++建立專案的說明，並在 13-3 節，告訴您如何自訂標頭檔。以下將運用第 15 章的 Hello_C++範例，改成由兩個檔案組成的程式，以說明如何建立與 include 自行建立的標頭檔。編譯時，只需要編譯 Hello_C++.cpp 檔即可。以下將把 Hello_C++範例程式，分成兩個檔案，一個是儲存主程式的 Hello_C++.cpp，另一個定義 Printer 類別的 Printer.h。以下為 Hello_C++.cpp 檔的內容。

檔案位置：ex26-1\Hello_c++.cpp

```
001   /*
002   範例檔名：Hello_c++.cpp
003   程式開發：郭尚君
004   */
005   #include <iostream>
006   #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
```

```
007   #include "Printer.h"
008
009   using namespace std; //使用 std 名稱空間
010
011   int main()   程式進入點
012   {
013      Printer My_Printer;  //依據 Printer 類別建立 My_Printer 物件
014
015      My_Printer.Hello();  //呼叫 My_Printer 物件的方法
016
017       system("PAUSE");
018       return 0;
019   } //主程式結束
```

下面是 Printer.h 檔的內容。

檔案 位置 ：ex26-1\Printer.h

```
001   /*
002   範例檔名：Printer.h
003   程式開發：郭尚君
004   */
005   #include <iostream>
006   #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007
008   using namespace std; //使用 std 名稱空間
009
010   class Printer  //宣告 Printer 類別
011   {
012   private:
013      int Serial_Number;  //宣告 Printer 類別的屬性
014   public:
015      void Hello()  //宣告 Printer 類別的方法
016      {
017          cout << "Hello C++ ! " << endl;  //印出 Hello C++ !
018      }
019   };    //完成 Printer 類別的宣告
```

26-2 圖書管理系統的發展

26-2-1 系統的發展過程 進階

這一章裡，將簡單示範建立一個小型系統。一方面讓讀者瞭解系統發展的過程，另一方面希望從這個系統的說明過程中，讓讀者體會建立以物件導向技術為基礎的系統，其設計理念、建構過程與系統運作架構。

開發電腦系統的過程大略可以區分為下列三個過程。

1. **系統分析**：分析系統所欲解決的問題為何？也可以說是使用者的需求為何？

2. **系統設計**：依照系統分析的結果，設計出符合使用者需求的系統架構。

3. **系統建構**：按照系統設計的結果，撰寫程式，建構系統。

系統分析、設計與系統建構的關係，就好像建築藍圖與建築大樓一般。在發展系統前，必須先確定出使用者對系統的需求，然後描繪出達成這些需求的系統，最後以程式撰寫出符合使用者需求的電腦系統。基本上，是一個確定問題（系統分析），研究解決方法（系統設計），最後解決問題（系統建構）的過程。必須強調一點，這三個流程並不是很單純地，由系統分析開始，然後進行系統設計，最後完成系統的建構。而是不斷在這三個步驟間來來回回，**反覆**思考的過程。

這三個步驟以物件導向的術語來說，稱為 OOA（Object Oriented Analysis）、OOD（Object Oriented Design）及 OOP（Object Oriented Programming）。

物件導向系統的分析、設計

物件導向觀念不論是在系統的分析與設計，抑或是在系統建構時，都有許多優點。其中有一個優點就是將系統的分析、設計與建構的過程所運用的觀念，統一在物件導向思維下，避免因系統發展的步驟中，使用了不同的觀念，產生因觀念差異，而造成步驟間的鴻溝。

在分析、設計一個物件導向系統時，除了瞭解使用者的需求，確定出系統目標外，當運用物件導向觀念分析、設計系統時，必須要做的工作還可以大致歸類為以下幾項：

1. 尋找系統中的物件，例如：圖書管理系統中的書籍與讀者。
2. 尋找物件應有的屬性與方法，例如：讀者有借書的動作，有姓名的屬性。
3. 尋找物件間的關係，如：繼承、聚合。
4. 釐清物件間如何交互運作以完成系統。

下圖是 14-5-2 節裡，說明物件導向觀念如何模擬真實世界時，所使用的圖。該圖以物件的特性、關係為經，靜態、動態的模擬為緯，清楚表達物件導向技術的各種觀念與模擬真實世界的關係。而物件導向觀念為基礎的系統分析、設計，就是利用這幾個角度來分析真實世界。

	靜態模擬	動態模擬
物件特性	屬性	方法
物件關係	繼承、聚合	訊息

在上面的步驟，並未區分哪些是系統分析，哪些是系統設計。事實上，它們之間的界線也是模糊的，且難以界定的。而實際上，也不太需要去區分它們。只是上述的四個步驟，同樣也是一個來回反覆修改的過程。剛開始時，您可以僅簡單地找出一些您認為系統中應該有的物件，然後進一步想出該物件應該有屬性與方法。接著，發展它們間的繼承關係。此時，您或許會發現

系統中應該再加入一些物件，所以您又回到了第一步。而當您想像系統的物件如何運作以完成使用者的需求時，您可能又發現剛剛設計的物件屬性、方法不敷使用，所以又為某些物件加了幾個屬性或者方法。更甚者，當開始建立系統時，您還會發現原先的設計是需要修正的。總而言之，一個系統的發展過程並不是單純地直線發展，而在各步驟中反反覆覆的，只不過大致發生的順序是如此罷了。

建立系統

在前面的系統分析、設計的過程裡，對物件的靜態、動態，都做了瞭解。接下來的重點是，如何落實進入系統中。這裡看出運用物件導向觀念的好處。直接將分析、設計所建立的物件，用程式寫出來就好了，很直接是不！？

26-2-2 圖書管理系統的分析與設計

前面大致介紹物件導向系統的大致發展過程，本節將以圖書管理系統為案例，來看看一個系統的發展過程。我們暫訂這個圖書管理系統的目標，在提供辦公室或者個人管理圖書。

確定需求

不論是設計哪一種系統，確定使用者的需求是相當重要的一個步驟。在尚未正確瞭解使用者的需求前，即貿然著手建立系統，就好像漫無目標地在沙漠中行走，想要達成使用者的需求，無異是緣木求魚。

確定需求的方式，有許多方式，最實際而可靠的無非是請教使用者，瞭解它們的作業方式，瞭解它們希望系統幫它們做什麼？對於一個圖書管理系統，使用上的需求大致可分為這幾個方面：

1.讀者資料的管理

2.書籍資料的管理

3.圖書的借閱與歸還

系統分析

確定了需求以後，接下來，開始尋找系統中的物件。

1.尋找系統中的物件

當要模擬真實世界時，我們該如何從真實世界裡，找出哪些是物件導向系統中的物件呢？這個問題並不難，只要尋找真實世界裡存在的名詞即可。以下就用找名詞這個方法，尋找圖書管理系統中應該有哪些物件？

讓我們回想一下曾去過的圖書館吧！進入圖書館以後，您會看到的是圖書館管理員坐在櫃檯上，為前來借/還書的讀者辦理手續。再進入書籍陳列室，接著映入眼簾的是一排排的書架，書架上擺了許多書。

在以上敘述裡，出現了管理員、櫃檯、讀者、書架、書，這些名詞，這些存在於真實圖書館裡的物件。每一種物件都需要在圖書管理系統中建立相對應的物件嗎？當然不是囉！不然，豈不是連圖書館裡有多少桌椅都必須模擬了。可是問題是篩選哪些物件應該模擬，哪些物件不該的那把尺在那裡？那把尺就是我們欲建立系統的目標，也就是系統欲解決的問題。我們來想想以圖書管理系統的建立目標來篩選，櫃檯、書架這些物件需要存在嗎？答案是否定的，因為不論是管理書籍、讀者的資料，或者辦理圖書的借閱與歸還，在圖書管理系統的運作裡，都不需要它們。因此，以圖書管理為目標篩選這些物件時，可以找出系統中應有的物件是 – 管理員、讀者、書。

二、尋找物件應有的屬性與方法

接下來的問題是確定這些物件應有的屬性與方法，也就是該如何**抽象化**系統中的物件。與前面尋找物件類似的，尋找物件應有的屬性與方法，所要找的就是真實世界裡，描述這些物件的**形容詞**（屬性）與**動詞**（方法）。下表將列出管理員、讀者、書這些物件的屬性。

物件	有的屬性	無的屬性
圖書館管理員	姓名	頭髮長短 有無駕照 年齡
讀者	姓名 生日 住址 借書記錄	頭髮長短 有無駕照 身份證字號
書	書名 作者 出版商 出版日期 是否在架上	書的大小 封面顏色

在上表中，簡要地列出了幾種屬性，我們可以看出這些屬性都是用於**形容**圖書管理系統中的物件。但是有些屬性對於圖書管理系統的運作，並沒有實質的用處，諸如：讀者有無駕照、書的封面顏色...等。這些屬性，並不需要抽象化為書物件的屬性，我們只需要哪些與圖書管理運作有關的屬性。接下來，表中將列出物件的方法。

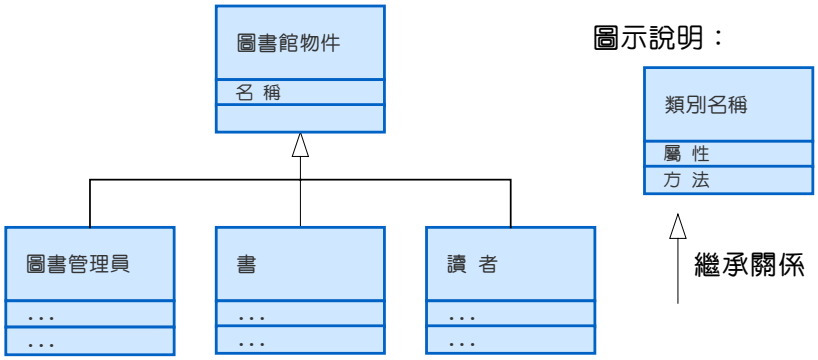
物件	有的屬性	無的屬性
圖書館管理員	辦理借書 辦理還書 維護書籍資料 維護讀者資料	開車 走路 說話 ...

物件	有屬性	無屬性
讀者	借書 還書	騎腳踏車 走路 說話 …
書		

上表中物件的方法，對應到真實世界，就是物件的**動作**。當然物件的動作很多，並不是每一種動作都需要模擬。例如：讀者會騎腳踏車，就是一個在圖書管理系統運作時，無用的方法。當然，篩選的那把尺還是系統的目標。

三、尋找物件間的關係

從第一個步驟裡，尋找出系統中應該存在管理員、讀者、書這三種物件，並將它們都歸類為**圖書館物件**。所以，在系統中，它們都衍生於圖書館物件。因此，我們將這些物件共同的屬性 - 名稱放進圖書館物件裡，讓各物件透過繼承圖書館物件的方式擁有名稱屬性。



四、釐清物件間如何交互運作以完成系統需求

在這個步驟裡，必須利用**事件**的觀念探討物件間如何交互運作，最後完成系統需求。以下將以借書事件為範例，說明圖書館中借書作業（人工方式）的流程描述。

1. 讀者拿書和借書證向圖書館管理員要求借書
2. 管理員將書的借閱記錄上蓋上借閱日期
3. 並將在讀者的借書證上記錄借閱了該書
4. 完成借書作業

下表將借書事件的發生流程，與物件的方法對應，由此可以看出系統中物件的交互運作過程。

步驟	真實世界的動作	電腦系統中物件的動作
1.	讀者向圖書館管理員要求借書	呼叫圖書管理員的借書作業，要求輸入讀者及書籍資料
2.	管理員將書的借閱記錄上蓋上借閱日期	圖書館管理員物件設定該書物件的狀態為借出
3.	並將在讀者的借書證上記錄借閱了該書	圖書館管理員物件在讀者物件的借書記錄上，增加借閱一本書
4.	完成借書作業	完成借書作業

系統設計

經過系統分析後，我們大致瞭解整個圖書管理系統中，應該有的物件，以及這些物件的屬性、方法。接下來的問題，是如何將它們轉化成為程式。下表整理經由前面分析所尋找出，圖書管理系統中物件應有的屬性與方法。

物件名稱	屬性	屬性	方法
圖書館物件（LibraryObject）		名稱	
圖書管理員（Librarian）	圖書館物件		辦理借書作業 辦理還書作業 維護書籍資料 維護讀者資料

物件名稱	屬性	屬性	方法
讀者（Reader）	圖書館物件	生日 住址 借書記錄	借書 還書
書（Book）	圖書館物件	作者 出版商 出版日期 是否在架上	

有了上表所整理出的物件屬性與方法，我們已經大致描繪出了系統中物件的模樣，此時若要立刻開始建立系統，也未嘗不可。不過，如果希望更詳細地描述系統，我們仍須考慮一些電腦運作上的一些問題，例如：如何儲存物件、各物件設定屬性的方法等。這些系統運作時所需要物件、屬性與方法，從真實世界的分析是看不出來的，因此，在系統分析時，並未被找出來。

對於圖書管理系統，運作之後的資料是必須被儲存下來的，否則程式結束後，所有資料都消失，根本沒辦法達到圖書管理的目的。因此，在圖書管理系統必須具備物件儲存的功能，其達成的方法有以下兩種。

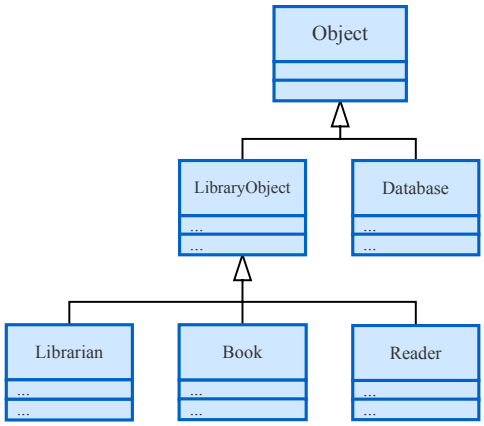
1. 利用每個物件的方法，操作物件的檔案輸出/入

2. 建立資料庫物件，操作所有物件的檔案輸出/入

這裡我們採用第二種方法，因此，系統中將增加一個資料庫物件，負責所有物件對檔案的輸出/入。而需要透過資料庫物件，進行檔案的輸出/入的物件（讀者、書），也必須增加屬性用以儲存存放資料的檔案名稱。資料庫物件的屬性與方法整理如下。

資料庫物件（Database）	
屬性	方法
欲操作的檔案名稱 每筆資料的設定 檔案的設定	新增物件資料 刪除物件資料 修改物件資料 查詢物件資料 顯示檔案中所有物件的資料

並增加一個 Object 類別，做為資料庫物件與圖書館物件的基礎類別。最後圖書管理系統內各類別的繼承架構將如下圖所示：



此外，我們在讀者與書物件，增加了編號屬性，以及一些供其他物件存取資料的介面（方法）。下表為系統設計後，所完成整個系統內物件之屬性、方法規劃。

物件名稱	屬性	方法
Object		
圖書館物件 （LibraryObject）	*編號 名稱	*設定/取得編號 *設定/取得名稱

物件名稱	屬性	方法
圖書管理員 （Librarian）		辦理借書作業 辦理還書作業 維護書籍資料 維護讀者資料
讀者（Reader）	生日 住址 借書記錄	借書 還書 *設定/取得住址 *設定/取得生日 *設定/取得借書記錄 *顯示物件資料
書（Book）	書名 作者 出版商 出版日期 是否在架上（是否被借出）	*設定/取得書名 *設定/取得作者 *設定/取得出版商 *設定/取得出版日期 *設定在架上 *設定不在架上
*資料庫物件 （Database）	*操作檔案的名稱 *每筆資料的長度 *檔案的長度	*新增物件資料 *刪除物件資料 *修改物件資料 *查詢物件資料 *顯示檔案中所有物件的資料

註： 1.標記星號（*）者表示在系統設計時所增加。
2.下加底線者，實做系統時，將予以省略。

為了讓讀者把注意力放在物件間的交互運作上，因此，系統實做時，我們將簡化設計降低整個系統的複雜度。所以，上表中下加底線的部份，將予以省略。下面將更進一步訂出各物件（類別）的詳細規格，其中包含屬性的名稱、型態與方法的名稱、回傳值、功能描述。

類別名稱	**Object**		
基礎類別	無	衍生類別	LibraryObject、Database
屬性			
名稱	型態	意義	備註
無			
方法			
原型	回傳值	功能描述	備註
無			

類別名稱	**Database**		
基礎類別	Object	衍生類別	無
屬性			
名稱	型態	意義	備註
FileName	string	操作檔案的名稱	
RecSize	long	每筆記錄的長度	
FileLen	long	檔案的長度	
方法			
原型	回傳值	功能描述	備註
Insert(LibraryObject &)	無	將傳入的物件插入檔案中	
Delete(string)	無	刪除與傳入字串名稱相同的物件	
Query(string)	物件	傳回與傳入字串名稱相同的物件，如果找不到則引出一個例外	

類別名稱	**LibraryObject**		
基礎類別	Object	衍生類別	Librarian、Reader、Book
屬性			
名稱	型態	意義	備註
index	int	物件的編號	
name	char[20]	物件的名稱	
方法			
原型	回傳值	功能描述	備註
GetName()	name	將物件的名稱傳出	
SetName(char *)	無	設定物件名稱	
GetIndex()	index	將物件的編號傳出	
SetIndex(int)	無	設定物件的編號	

類別名稱	**Reader**		
基礎類別	LibraryObjet	衍生類別	無
屬性			
名稱	型態	意義	備註
BroBookList	int[10]	已借閱書籍之編號	
FileName	char[15]	儲存 Book 物件的檔案名稱	類別成員，值為"Reader.txt"
方法			
原型	回傳值	功能描述	備註
BorrowBook(int)	無	傳入欲借閱書籍的編號，如果讀者已借閱超過 10 本書，則丟出一個例外	
ShowData()	無	輸出 Reader 物件的資料	
GetFileName()	char *	傳回儲存 Reader 物件的檔案名稱	類別成員

類別名稱	**Librarian**		
基礎類別	LibraryObjet	衍生類別	無
屬性			
名稱	型態	意義	備註
方法			
原型	回傳值	功能描述	備註
CheckOut()	無	執行借書作業	
BookData()	無	維護書籍資料	
ReaderData()	無	維護讀者資料	

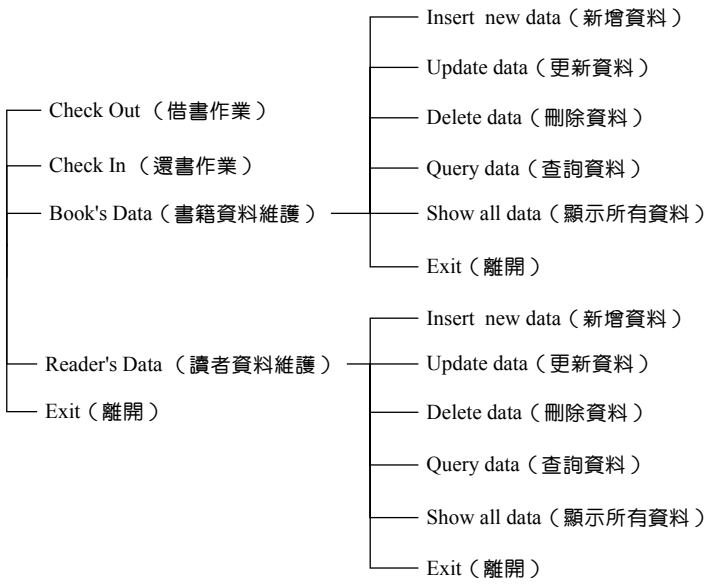
類別名稱	**Book**		
基礎類別	LibraryObjet	衍生類別	無
屬性			
名稱	型態	意義	備註
name	char[40]	書籍名稱	覆蓋繼承自 LibraryObjet 之屬性
OnShelf	bool	是否在架上	
FileName	char[15]	儲存 Book 物件的檔案名稱	類別成員，值為"Book.txt"
方法			
原型	回傳值	功能描述	備註
SetName()	無	設定書籍名稱	覆蓋繼承自 LibraryObjet 之方法
GetName()	無	取得書籍名稱	覆蓋繼承自 LibraryObjet 之方法
CheckOut()	無	確定書籍未被借出，如果不在架上引出例外，如果在架上將 OnShelf 設為 false，表示被借出	
ShowData()	無	輸出 Book 物件的資料	
GetFileName()	char *	傳回儲存 Book 物件的檔案名稱	類別成員

26-3 圖書管理系統的建構 進階

26-3-1 系統的架構

經過了上一節的分析、設計，整個圖書管理系統內應有的物件大致已經確定，接著，便是按照設計建立系統。然而分析、設計的結果，畢竟只是想像出來的結果，所以，在建立系統的過程裡，仍會修正，因此，最後建立出來的系統仍會有地方與原先的設計結果不一樣。

我們將整個圖書管理系統建立成多檔案程式，每個類別的宣告都放在一個自行建立的標頭檔中。整個系統有兩層選單（如下圖所示），供使用者選擇欲執行的各項作業，有借書作業（Check Out）、還書作業（Check In）、書籍資料維護（Book's Data）及讀者資料維護（Reader's Data）。當選擇維護書籍或讀者資料時，將進入第二層選單，這一層選單將提供資料維護的相關選項。



以下各小節，將列出主程式與定義各類別的標頭檔之原始碼，並為您說明。

26-3-2 主程式、Object 與 LibraryObject 類別

主程式

主程式在整個系統裡，基本上並沒有什麼實際執行功能，主要是載入定義類別的標頭檔，以及提供第一層選單，供使用者選擇欲執行的作業。而實際的執行動作，均由 Librarian 類別執行。因此，在主程式的第 22 行裡，宣告一個 Librarian 類別的 librarian 物件。第 30～53 行的 switch...case 判斷式裡，當使用者選擇執行管理系統的某功能時，均呼叫 librarian 物件的方法，完成各項操作。因此，若您希望了解系統各項功能完成的細部動作，僅需要深入瞭解 Librarian 類別的方法即可。

檔案 位置：Library\library_tmp_db.cpp

```
001   /*
002   範例檔名：library_tmp_db.cpp
003   程式開發：郭尚君
004   */
005
006   #include <iostream>
007   #include <map>
008   #include <string>
009   #include <fstream>
010   #include <string.h>
011   #include "LibraryObject.h"     //載入其他檔案
012   #include "Reader.h"
013   #include "Book.h"
014   #include "Database.h"
015   #include "Librarian.h"
016
017   using namespace std;
018
019   int main() //主程式開始 <=== 程式進入點
020   {
```

```
021      char choice = 'I';
022      Librarian librarian;
023
024      while( !(choice == 'E' || choice == 'e') )
025      {
026         cout << "check (O)ut, check (I)n, (B)ook's data,";
027         cout << " (R)eader's data, (E)xit :" << endl;
028         cin >> choice;//選擇欲執行的功能
029
030         switch(choice)
031         {
032         case 'o':
033         case 'O':   //借書作業
034            librarian.CheckOut(); //呼叫 Librarian 的借書作業
035            break;
036         case 'i':
037         case 'I':   //還書作業，留給您練習
038            cout << "Let you practice!" << endl;
039            break;
040         case 'b':
041         case 'B':   //維護書籍資料
042            librarian.BookData();
043            break;
044         case 'r':
045         case 'R':   //維護讀者資料
046            librarian.ReaderData();
047            break;
048         case 'e':
049         case 'E':   //離開系統
050            break;
051         default:
052            cout << "Don't provide this function!" << endl;
053         }
054      };
055   }  //主程式結束
```

Object 與 LibraryObject 類別

在系統裡，所有的類別都衍生於 Object 類別，不過該類別沒有任何屬性與方法。而系統中的 Librarian、Book、Reader 類別都衍生於 LibraryObject 類別。LibraryObject 裡定義了，圖書館中物件的共同屬性 – index、name，

並提供這些屬性的存取介面。下面是定義 Object 與 LibraryObject 類別的原始碼。

檔案位置：Library\LibraryObject.h

```
001  /*
002  範例檔名：LibraryObject.h
003  程式開發：郭尚君
004  */
005
006  #ifndef _LIBRARYOBJECT_H_
007  #define _LIBRARYOBJECT_H_
008
009  #include <iostream>
010  using namespace std;
011
012  class object{};  //定義 object 類別
013
014  class LibraryObject : public object
015  {
016  protected:
017     int index; //編號
018     char name[20];//名稱
019  public: //成員函數
020     LibraryObject (int index, const char * name);
021     LibraryObject (const char * i_name);
022     LibraryObject ();
023     const char * GetName ( );  //輸出名稱
024     int GetIndex ( );//輸出編號
025     virtual void ShowData();//輸出資料
026     void SetIndex(int i_index);//設定編號
027     void SetName(const char * i_name);//設定名稱
028  };
029  #endif // of _LIBRARYOBJECT_H_
```

檔案位置：Library\LibraryObject.cpp

```
001  /*
002  範例檔名：LibraryObject.cpp
003  程式開發：郭尚君
004  */
005
006  #include "LibraryObject.h"
007
```

```
008   //定義建構子
009   LibraryObject::LibraryObject(int index, const char * name)
010   : index(index) { SetName(name);}
011
012   LibraryObject::LibraryObject (const char * i_name)
013   { strcpy(name, i_name);}
014
015   LibraryObject::LibraryObject () {}
016
017   const char * LibraryObject::GetName ( ) //輸出名稱
018   { return name; }
019
020   int LibraryObject::GetIndex ( ) { return index; }     //輸出編號
021
022   void LibraryObject::ShowData() //輸出資料
023   { cout << index << "  " << name << "  ";}
024
025   void LibraryObject::SetIndex(int i_index) //設定編號
026   { index = i_index; }
027
028   void LibraryObject::SetName(const char * i_name) //設定名稱
029   { strcpy(name, i_name); }
```

26-3-3 Librarian 類別

Librarian 類別的定義

Librarian 是整個圖書管理系統中最重要的類別，定義於 Librarian.h 標頭檔中。Librarian 模擬了現實世界內的**圖書館管理員**，所以不論是借/還書作業，或者是書籍與讀者的資料維護都是由該類別完成。我們並沒有完全完成 Librarian 的所有方法，留下了還書作業（CheckIn()方法）讓讀者練習。已經完成的方法，分別是 CheckOut()（借書作業）、BookData()（書籍資料維護）、ReaderData()（讀者資料維護）。下面是 Librarian 類別的定義內，從 Librarian 類別的各個方法裡，將可以瞭解系統中，各物件間交互運作的過程。

檔案位置：Library\Librarian.h

```
001  /*
002  範例檔名：Librarian.h
003  程式開發：郭尚君
004  */
005  #include <iostream>
006
007  using namespace std;
008
009  #include "LibraryObject.h"
010
011  //定義 Librarian 繼承於 LibraryObject
012  class Librarian : public LibraryObject
013  {
014  public:
015     void CheckOut(); //借書作業
016     void BookData(); //維護書籍資料
017     void ReaderData();  //維護讀者資料
018  };
```

檔案位置：Library\Librarian.cpp

```
001  /*
002  範例檔名：Librarian.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
006  #include "Librarian.h"
007  #include "Database.h"
008  #include "Reader.h"
009  #include "Book.h"
010
011  using namespace std;
012
013  void Librarian::CheckOut()     //借書作業
014  {
015     char r_name[20], b_name[40];
016     Database<Reader> ReaderDB;
017     Database<Book> BookDB;
018     try{
019        cout << "Please input Reader's name :" << endl;
020        cin >> r_name;
021        Reader * reader = ReaderDB.Query(r_name);
022        //尋找欲借書的讀者資料
```

```
023
024         cout << "Please input Book's name :" << endl;
025         cin.ignore(1);
026         cin.get(b_name, 40, '\n') ;
027         Book * book = BookDB.Query(b_name);    //欲借出的書籍資料
028         book->CheckOut();   //設定該書被借出
029         reader->BorrowBook(book->GetIndex()); //增加讀者的借書記錄
030      }
031      catch(string s) { cerr << s;}
032   }
033
034   void Librarian::BookData()     //維護書籍資料
035   {
036      char choice = 'I';
037      char b_name[40];
038      Database<Book> BookDB; //宣告一個操作 Book 檔案的 Database 物件
039      Book * book;
040
041      while ( !(choice == 'E' || choice == 'e') )
042      {
043         cout << "Maintain Book Database" << endl;
044         cout << "(I)nsert new data, (U)pdate data, (D)elete data,"
045              <<   " (Q)uery data, (S)how all data, (E)xit  : " << endl;
046         cin >> choice;//選擇欲執行的動作
047
048         try{
049            switch(choice)
050            {
051            case 'i':
052            case 'I':  //新增書籍資料
053               cout << "Please input a Book's name : " << endl;
054               cin.ignore(1);
055               cin.get(b_name, 40, '\n');
056               book = new Book(b_name);
057               BookDB.Insert(*book);   //將書籍資料插入
058               break;
059            case 'u':
060            case 'U':  //更改書籍資料
061               cout << "Please input Book's name : " << endl;
062               cin.ignore(1);
063               cin.get(b_name, 40, '\n');
064
065               book = BookDB.Query(b_name); //尋找欲更改的書籍資料
066
```

```
067              cout << "Please input Book's new name :" << endl;
068              cin.ignore(1);
069              cin.get(b_name, 40, '\n');
070              book->SetName(b_name); //重新設定該書籍的名稱
071              break;
072           case 'd':
073           case 'D':  //刪除書籍資料
074              cout << "Please input Book's name : " << endl;
075              cin.ignore(1);
076              cin.get(b_name, 40, '\n');
077              BookDB.Delete(b_name); //刪除書籍資料
078              break;
079           case 'q':
080           case 'Q':  //查詢書籍資料，這個功能留給您練習
081              cout << "Let you practice!" << endl;
082              break;
083           case 's':
084           case 'S':  //顯示 Database 中所有書籍資料
085              BookDB.ShowAllData();
086              break;
087           case 'e':
088           case 'E':  //離開系統
089              break;
090           default:
091              cout << "Don't provide this function !" << endl;
092           }
093        }
094        catch(string s) { cerr << s;}
095     };
096  }
097
098  void Librarian::ReaderData()  //維護讀者資料
099  {
100     char choice = 'I' , r_name[40];
101     Database<Reader> ReaderDB;
102      //宣告一個操作 Reader 檔案的 Database 物件
103     Reader * reader;
104
105     while ( !(choice == 'E' || choice == 'e') )
106     {
107        cout << "Maintain Reader Database" << endl;
108        cout << "(I)nsert new data, (U)pdate data, (D)elete data,"
109             << " (Q)uery data, (S)how all data, (E)xit  : " << endl;
110        cin >> choice;//選擇欲執行的動作
```

```
111
112         try{
113            switch(choice)
114            {
115            case 'i':
116            case 'I':  //新增讀者資料
117               cout << "Please input a Reader's name : " << endl;
118               cin >> r_name;
119               reader = new Reader(r_name);
120               ReaderDB.Insert(*reader);
121               break;
122            case 'u':
123            case 'U':  //更改讀者資料
124               cout << "Please input Reader's name : " << endl;
125               cin >> r_name;
126               reader = ReaderDB.Query(r_name);
127                //尋找欲更改的讀者資料
128
129               cout << "Please input Reader's new name :" << endl;
130               cin >> r_name;
131               reader->SetName(r_name);     //重新設定該讀者的姓名
132               break;
133            case 'd':
134            case 'D':  //刪除讀者資料
135               cout << "Please input Reader's name : " << endl;
136               cin >> r_name;
137
138               ReaderDB.Delete(r_name);     //刪除讀者資料
139               break;
140            case 'q':
141            case 'Q':  //查詢讀者資料，這個功能留給您練習
142               cout << "Let you practice!" << endl;
143               break;
144            case 's':
145            case 'S':  //顯示 Database 中所有讀者資料
146               ReaderDB.ShowAllData();
147               break;
148            case 'e':
149            case 'E':  //離開系統
150               break;
151            default:
152               cout << "Don't provide this function !" << endl;
153            }
154         }
```

```
155         catch(string s) { cerr << s;}
156      };
157   }
```

Librarian::CheckOut()

Librarian 類別的 CheckOut 方法，用於完成讀者的借書作業。在 CheckOut() 方法裡，首先宣告了兩個 Database 類別物件（第 16、17 行），然後讀取輸入的書籍名稱與讀者姓名，利用 Database 物件的 Query()方法，從檔案中把書籍與讀者的資料讀取出來。若以上資料讀取動作均完成，且沒有因發生錯誤而丟出例外，則呼叫 book 物件（書籍）的 CheckOut()方法，設定該 book 的 OnShelf()為 false。最後，再呼叫 reader 物件（讀者）的 BorrowBook()方法，在讀者的借閱資料中增加借閱一本書。

在讀取書籍名稱時，由於書籍名稱可能存在空白，所以讀取書籍資料時，必須利用 iostream 類別的 get()方法（第 26 行），並指定欲讀取字串的大小及分隔字元，且必須將前一次讀取資料剩餘在資料流中的結尾字元忽略，所以在使用 get()方法前必須先使用 ignore 方法（第 25 行），忽略資料流中的殘餘字元。下面是 Librarian::CheckOut()的程式碼。

```
'摘自 Librarian.cpp 檔
013   void Librarian::CheckOut()     //借書作業
014   {
015      char r_name[20], b_name[40];
016      Database<Reader> ReaderDB;
017      Database<Book> BookDB;
018      try{
019         cout << "Please input Reader's name :" << endl;
020         cin >> r_name;
021         Reader * reader = ReaderDB.Query(r_name);
022         //尋找欲借書的讀者資料
023
024         cout << "Please input Book's name :" << endl;
025         cin.ignore(1);
026         cin.get(b_name, 40, '\n') ;
027         Book * book = BookDB.Query(b_name);    //欲借出的書籍資料
028         book->CheckOut();   //設定該書被借出
```

```
029          reader->BorrowBook(book->GetIndex()); //增加讀者的借書記錄
030       }
031       catch(string s) { cerr << s;}
032    }
```

26-3-4 Librarian::BookData()與 Librarian::ReaderData()

Librarian 的 BookData()與 ReaderData()，分別用於管理書籍與讀者的資料。在這兩個方法裡，都將分別宣告用於操作 Book 物件與 Reader 物件的 Database 物件，並利用 Database 的資料操作方法，完成資料的維護。所以，這兩個方法裡，執行的動作大多只是讀取使用者輸入的資料，與呼叫 Database 物件的方法。以下是 Librarian::BookData()的原始碼。

```
'摘自 Librarian.cpp 檔
034   void Librarian::BookData()     //維護書籍資料
035   {
036      char choice = 'I';
037      char b_name[40];
038      Database<Book> BookDB; //宣告一個操作 Book 檔案的 Database 物件
039      Book * book;
040
041      while ( !(choice == 'E' || choice == 'e') )
042      {
043         cout << "Maintain Book Database" << endl;
044         cout << "(I)nsert new data, (U)pdate data, (D)elete data,"
045              <<   " (Q)uery data, (S)how all data, (E)xit  : " << endl;
046         cin >> choice;//選擇欲執行的動作
047
048         try{
049            switch(choice)
050            {
051            case 'i':
052            case 'I':  //新增書籍資料
053               cout << "Please input a Book's name : " << endl;
054               cin.ignore(1);
055               cin.get(b_name, 40, '\n');
056               book = new Book(b_name);
057               BookDB.Insert(*book);   //將書籍資料插入
```

```
058                break;
059             case 'u':
060             case 'U':   //更改書籍資料
061                cout << "Please input Book's name : " << endl;
062                cin.ignore(1);
063                cin.get(b_name, 40, '\n');
064
065                book = BookDB.Query(b_name); //尋找欲更改的書籍資料
066
067                cout << "Please input Book's new name :" << endl;
068                cin.ignore(1);
069                cin.get(b_name, 40, '\n');
070                book->SetName(b_name);  //重新設定該書籍的名稱
071                break;
072             case 'd':
073             case 'D':   //刪除書籍資料
074                cout << "Please input Book's name : " << endl;
075                cin.ignore(1);
076                cin.get(b_name, 40, '\n');
077                BookDB.Delete(b_name);  //刪除書籍資料
078                break;
079             case 'q':
080             case 'Q':   //查詢書籍資料，這個功能留給您練習
081                cout << "Let you practice!" << endl;
082                break;
083             case 's':
084             case 'S':   //顯示 Database 中所有書籍資料
085                BookDB.ShowAllData();
086                break;
087             case 'e':
088             case 'E':   //離開系統
089                break;
090             default:
091                cout << "Don't provide this function !" << endl;
092             }
093          }
094          catch(string s) { cerr << s;}
095       };
096    }
```

以下是 Librarian::ReaderData()的原始碼。

```
'摘自 Librarian.cpp 檔
098   void Librarian::ReaderData()  //維護讀者資料
099   {
100      char choice = 'I' , r_name[40];
101      Database<Reader> ReaderDB;
102       //宣告一個操作 Reader 檔案的 Database 物件
103      Reader * reader;
104
105      while ( !(choice == 'E' || choice == 'e') )
106      {
107         cout << "Maintain Reader Database" << endl;
108         cout << "(I)nsert new data, (U)pdate data, (D)elete data,"
109              << " (Q)uery data, (S)how all data, (E)xit  : " << endl;
110         cin >> choice;//選擇欲執行的動作
111
112         try{
113            switch(choice)
114            {
115            case 'i':
116            case 'I':  //新增讀者資料
117               cout << "Please input a Reader's name : " << endl;
118               cin >> r_name;
119               reader = new Reader(r_name);
120               ReaderDB.Insert(*reader);
121               break;
122            case 'u':
123            case 'U':  //更改讀者資料
124               cout << "Please input Reader's name : " << endl;
125               cin >> r_name;
126               reader = ReaderDB.Query(r_name);
127                //尋找欲更改的讀者資料
128
129               cout << "Please input Reader's new name :" << endl;
130               cin >> r_name;
131               reader->SetName(r_name);     //重新設定該讀者的姓名
132               break;
133            case 'd':
134            case 'D':  //刪除讀者資料
135               cout << "Please input Reader's name : " << endl;
136               cin >> r_name;
137
138               ReaderDB.Delete(r_name);     //刪除讀者資料
139               break;
```

```
140             case 'q':
141             case 'Q':   //查詢讀者資料，這個功能留給您練習
142                cout << "Let you practice!" << endl;
143                break;
144             case 's':
145             case 'S':   //顯示 Database 中所有讀者資料
146                ReaderDB.ShowAllData();
147                break;
148             case 'e':
149             case 'E':   //離開系統
150                break;
151             default:
152                cout << "Don't provide this function !" << endl;
153             }
154          }
155          catch(string s) { cerr << s;}
156       };
157    }
```

26-3-5 Book 類別

Book 類別的定義

Book 類別以 public 方式繼承於 LibraryObject，但由於書籍的名稱比較長，所以，Book 類別裡將再定義一個 name 屬性。因此，對於名稱存取的介面 SetName()與 GetName()都必須重新建立，若沿用繼承於 LibraryObject 的存取介面，將存取到繼承於 LibraryObject 的 name 屬性。

在 Book 類別裡還定義了一個類別成員屬性 FileName，用以記錄儲存 Book 物件資料之檔案的名稱，Book 類別的預設值為 Book.txt。該屬性將透過 GetFileName()靜態成員函數，供 Database 物件取得。

檔案位置：library\book.h

```
001   /*
002   範例檔名：Book.h
003   程式開發：郭尚君
004   */
005   #include "LibraryObject.h"
```

```
006
007   //Book 繼承於 LibraryObject
008   class Book : public LibraryObject
009   {
010   private:
011      char name[40];//書名
012      bool OnShelf; //是否在架上
013      static char FileName[40];   //儲存 Book 物件的檔案名稱
014   public:  //成員函數
015      //建構子
016      Book(const char * i_name) : OnShelf(true)
017       { strcpy(Book::name, i_name); }
018      Book( ) {}
019      void CheckOut(); //被借出
020      const char * GetName ( ) { return name; }//取得書名
021      void ShowData(); //輸出書的資料
022      void SetName(const char * i_name) //設定書名
023       { strcpy(name, i_name); }
024      static char * GetFileName ( ) //輸出儲存 Book 物件的檔案名稱
025       { return FileName;}
026   };
```

檔案位置：library\book.cpp

```
001   /*
002   範例檔名：Book.cpp
003   程式開發：郭尚君
004   */
005
006   #include "Book.h"
007
008   char Book::FileName[40] = "C:\\C_C++\\ch26\\Library\\Book.txt";
009   //儲存 Book 物件的檔案爲 Book.txt
010
011   void Book::CheckOut()  //被借出
012   {  //如果書不在架上，則丟出例外
013      if(OnShelf == false)
014         throw("Book is not on shelf!\n");
015
016      OnShelf = false; //設定被借出
017   }
018
019   void Book::ShowData()  //輸出書的資料
020   {
```

```
021      cout << index << "  " << name << "  ";
022
023      if(OnShelf == false)
024         cout << "  Not on shelf" << endl;
025      else
026         cout << "  On shelf" << endl;
027   }
```

Book::CheckOut()與 Book::ShowData()

Book 類別與系統運作較為有關的兩個成員函數為 Book::CheckOut()與 Book::ShowData()。Book::CheckOut()用於當書籍要求被借出時，先確定書籍是否已被借出，如果書籍不在架上，則丟出一個例外。如果在架上則設定 OnShelf 為 false。Book::ShowData()用於顯示書籍的資料。

```
'摘自 Book.cpp
011   void Book::CheckOut()  //被借出
012   {  //如果書不在架上，則丟出例外
013      if(OnShelf == false)
014         throw("Book is not on shelf!\n");
015
016      OnShelf = false; //設定被借出
017   }
018
019   void Book::ShowData()  //輸出書的資料
020   {
021      cout << index << "  " << name << "  ";
022
023      if(OnShelf == false)
024         cout << "  Not on shelf" << endl;
025      else
026         cout << "  On shelf" << endl;
027   }
```

26-3-6 Reader 類別

Reader 類別的定義

Reader 類別與 Book 類別一樣，以 public 方式繼承於 LibraryObject 類別，

只是 Reader 未再定義 name 屬性。因此，Reader 類別將沿用 LibraryObject 的資料存取介面，且 Reader 建構子可呼叫 LbiraryObject 類別的建構子，設定繼承於 LbraryObject 的屬性，然後再起始化 BorBookList（借閱的書籍編號）陣列元素為 0，表示尚未借閱任何書籍。由於 Reader 物件也將儲存至檔案中，所以也必須提供 FileName 與 GetFileName()這兩個類別成員，供 Database 物件使用。

這裡有一個相當重要的觀念，為何 FileName 與 GetFileName()不宣告於 LibraryObject 類別中，然後讓 Book 類別與 Reader 類別將它們繼承下來呢？這樣做是不對的，因為 Book 類別與 Reader 類別都衍生於 LibraryObject 類別。因此，Book 類別與 Reader 類別都是 LibraryObject 類別的一種，而宣告於 LibraryObject 類別的類別成員- FileName 與 GetFileName()對整個類別而言只有一份，這樣造成的結果將是 Book 類別與 Reader 類別將共用一份 FileName 與 GetFileName()，這跟 Book 類別與 Reader 類別各別擁有一份 FileName 與 GetFileName()是不一樣的。

檔案位置：library\Reader.h

```
001   /*
002   範例檔名：Reader.h
003   程式開發：郭尚君
004   */
005   #include <iostream>
006   #include "LibraryObject.h"
007
008   using namespace std;
009
010   //Reader 繼承於 LibraryObject
011   class Reader : public LibraryObject
012   {
013   private:
014      int BorBookList[10];   //借書記錄
015      static const char FileName[40];
016       //儲存 Reader 物件的檔案名稱
017   public: //成員函數
018      Reader(const char *) ;//建構子
019      Reader( ) {}
```

```
020
021      void BorrowBook(int);  //借書
022      static const char * GetFileName( )   //輸出儲存資料的檔案
023       { return FileName;}
024      void ShowData(); //顯示資料
025   };
```

檔案位置：library\Reader.cpp

```
001   /*
002   範例檔名：Reader.cpp
003   程式開發：郭尚君
004   */
005   #include <iostream>
006   #include "Reader.h"
007   using namespace std;
008
009   const char Reader::FileName[40] =
                          "C:\\C_C++\\ch26\\Library\\Reader.txt";
010
011   Reader::Reader(const char * name) : LibraryObject(name)
012   {
013      for(int i = 0; i < 10; ++i)//起始設定借書記錄
014         BorBookList[i] = 0;
015   }
016
017   void Reader::BorrowBook(int BookID)     //借書
018   {
019      for(int i = 0; i < 10; ++i)
020      {
021         if (BorBookList[i] == 0)
022         {
023            BorBookList[i] = BookID;   //設定借書記錄
024            return;
025         }
026      }
027   }
028
029   void Reader::ShowData()   //顯示資料
030   {
031      LibraryObject::ShowData();
032
033      cout << endl << "Brorow book's index list : [ ";
034
035      for (int i = 0 ; i < 10; i++)
```

```
036         cout << BorBookList[i] << " | ";
037
038      cout << "]" << endl;
039   }
```

Reader::BorrowBook()與Reader::ShowData()

Reader 的 BorrowBook()成員函數，用於設定讀者的借閱記錄。借閱記錄是一個整數陣列，用於儲存被借閱書籍的編號，BorrowBook()函數裡運用一個迴圈尋找尚未被設定為 0 的陣列元素，然後將其儲存為欲借閱的書籍編號。Reader 類別 BorrowBook()成員函數的內容如下：

```
'摘自 Reader.cpp
017   void Reader::BorrowBook(int BookID)     //借書
018   {
019      for(int i = 0; i < 10; ++i)
020      {
021         if (BorBookList[i] == 0)
022         {
023            BorBookList[i] = BookID;   //設定借書記錄
024            return;
025         }
026      }
027   }
```

Reader 的 ShowData()將呼叫 LibraryObject 的 ShowData()，顯示繼承於 LibraryObject 的屬性，然後再輸出讀者的借閱記錄。ShowData()成員函數的內容如下：

```
'摘自 Reader.cpp
029   void Reader::ShowData()   //顯示資料
030   {
031      LibraryObject::ShowData();
032
033      cout << endl << "Brorow book's index list : [ ";
034
035      for (int i = 0 ; i < 10; i++)
036         cout << BorBookList[i] << " | ";
037
038      cout << "]" << endl;
039   }
```

26-3-7 Database 類別

Database 類別的定義

Database 類別在圖書管理系統中，負責從檔案中儲存與讀取物件。但對於每一種物件而言，這些檔案操作的動作是大同小異的。所以，在這種資料不同，但操作方式卻相同的情形下，我們將運用樣版的觀念建立 Database 類別。而 Database 類別操作物件由檔案輸出/入的方式，是利用 Database 類別的建構子，在建立操作某類別物件的 Database 物件時，將檔案內的物件全部讀取出來，並儲存於 map 容器中。然後在完成資料操作後，利用 Database 類別的解構子將物件再回存回檔案中。

將從檔案中讀取出的物件儲存至 map 容器中，該容器以檔案中物件的 name 為索引值，以檔案中物件為資料值。利用 map 容器儲存物件的最大優點在於操作資料時，可以直接使用 map 容器提供的方法，而不需要再自行建立。下面是定義 Database 類別的原始碼。

檔案位置：library\Database.h

```
001   /*
002   範例檔名：Database.h
003   程式開發：郭尚君
004   */
005   #include <iostream>
006   #include <fstream>
007   #include <map>
008   #include <string>
009   #include <typeinfo>
010   #include "LibraryObject.h"
011   using namespace std;
012
013   template <class T>
014   class Database : public object//Database 繼承於 object
015   {
016   private:
017      fstream File; //宣告一檔案資料流
018      string FileName; //檔案名稱
```

```
019      long FileLen, rec_size;     //檔案長度與一筆記錄的大小
020      typedef map<string, T, less<string> > mmtype;
021      //定義資料型態名稱 mmtype
022      mmtype RecMap;//宣告一個 map 容器
023      int MaxID; //物件中最大之編號
024   public: //成員函數
025       //建構子
026      Database() ;
027      ~Database(){SaveMap();}
028
029      void Insert(T &);   //將物件插入容器
030      void Delete(string);   //從容器中刪除物件
031      T * Query(string);  //查詢容器中的物件
032      void InitiMap(); //將檔案中的物件讀入容器中
033      void SaveMap();  //將容器中的物件輸出至檔案
034      void ShowAllData(); //顯示容器中所有元素的資料
035   };
036
037   template <class T>
038   Database<T>::Database() : FileName(T::GetFileName())
039   {
040      rec_size = sizeof(T);  //每筆記錄的大小
041      InitiMap();//將檔案中的物件讀入容器
042   }
043
044   template <class T>
045   void Database<T>::Insert(T & Object)    //將物件插入容器
046   {
047      MaxID++; //將容器中最大的編號加一
048      Object.SetIndex(MaxID);     //設定欲插入物件的編號
049
050      RecMap.insert(
            typename mmtype::value_type (Object.GetName(), Object) );
051      //將物件插入容器中
052   }
053
054   template <class T>
055   T * Database<T>::Query(string ObjName)  //從容器中刪除物件
056   {
057      typename mmtype::iterator iter;
058
059      iter = RecMap.find(ObjName);     //尋找物件
060
```

```
061      if(iter == RecMap.end())    //如果找不到物件則丟出例外
062      {
063         string c_name = typeid(T).name();
064         int NameLen = c_name.size();
065         string s = "Don't find this "
066                    + c_name.substr(6, NameLen) + "'s data!\n";
067         throw(s);
068      }
069
070      return &(iter->second);     //傳回找到的物件
071   }
072
073   template <class T>
074   void Database<T>::Delete(string ObjName) //查詢容器中的物件
075   {
076      Query(ObjName);  //尋找物件，如果找不到則會丟出例外
077      RecMap.erase(ObjName); //輸入欲刪除物件的名稱
078   }
079
080   template <class T>
081   void Database<T>::InitiMap()  //將檔案中的物件讀入容器中
082   {
083      fstream File;
084      MaxID = 0;
085
086      File.open(FileName.data(), ios::in | ios::binary);
087
088      File.seekg(0, ios::end);    //將檔案指標移向檔案結尾
089      FileLen = File.tellg();     //設定檔案長度
090
091      if(FileLen == 0){   //如果檔案大小爲 0，則關閉檔案
092         File.close();
093         return;
094      }
095
096      T Object;
097
098      File.seekg(0, ios::beg);
099
100      do{   //讀取物件，尋找最大物件編號，並插入容器內
101         File.read((char *) & Object, rec_size);
102
103         if(Object.GetIndex() > MaxID)
104            MaxID = Object.GetIndex();
```

```
105
106         RecMap.insert(
               typename mmtype::value_type (Object.GetName(), Object) );
107      }while(File.tellg() < FileLen);
108
109      File.close();
110   }
111
112   template <class T>
113   void Database<T>::SaveMap()    //將容器中的物件輸出至檔案
114   {
115      typename mmtype::const_iterator iter;
116      T Object;
117
118      File.open(FileName.data(), ios::out | ios::binary | ios::trunc);
119
120      for(iter = RecMap.begin() ; iter != RecMap.end() ; ++iter)
121         File.write((char *) & iter->second, rec_size);
122         //將物件寫入檔案
123
124      File.close();
125   }
126
127   template <class T>
128   void Database<T>::ShowAllData()    //顯示容器中所有元素的資料
129   {
130      typename mmtype::iterator iter;
131      T Object;
132
133      for(iter = RecMap.begin() ; iter != RecMap.end() ; ++iter)
134         (iter->second).ShowData();    //顯示物件資料
135   }
```

Database 類別的建構子

在宣告 Database 物件時，建構子將執行下列工作：

1. 取得欲操作的檔案名稱
2. 取得欲操作類別的物件大小，即儲存在檔案中的物件大小。
3. 將檔案中的物件讀取進入 map 容器中

下面是 Database 類別的建構子。

```
'摘自 Database.h
037   template <class T>
038   Database<T>::Database() : FileName(T::GetFileName())
039   {
040      rec_size = sizeof(T);  //每筆記錄的大小
041      InitiMap();//將檔案中的物件讀入容器
042   }
```

讀取與儲存物件

當建立 Database 物件時，必須將檔案中的物件讀入 map 容器中。完成這個動作的成員函數為 Initimap。該成員函數利用 File 物件的 read()方法，從檔案中讀取物件，然後利用 RecMap 容器的 insert()方法，將物件儲存進入 map 容器中。容器元素的索引值將利用物件的 GetName()方法取得，儲存資料則是讀取自檔案的物件。此外，在讀取檔案中的物件資料時，將順便取得儲存於檔案中物件的最大編號，供新增物件編號使用。

當完成資料的操作後，則利用 Database 類別的解構子呼叫 SaveMap()成員函數，將 RecMap 容器的元素中儲存的物件回存到檔案中。下面是 InitiMap()與 SaveMap()成員函數的原始碼。

```
'摘自 Database.h
080   template <class T>
081   void Database<T>::InitiMap()   //將檔案中的物件讀入容器中
082   {
083      fstream File;
084      MaxID = 0;
085
086      File.open(FileName.data(), ios::in | ios::binary);
087
088      File.seekg(0, ios::end);    //將檔案指標移向檔案結尾
089      FileLen = File.tellg();     //設定檔案長度
090
091      if(FileLen == 0){   //如果檔案大小爲 0，則關閉檔案
092         File.close();
093         return;
094      }
```

```
095
096      T Object;
097
098      File.seekg(0, ios::beg);
099
100      do{   //讀取物件，尋找最大物件編號，並插入容器內
101         File.read((char *) & Object, rec_size);
102
103         if(Object.GetIndex() > MaxID)
104            MaxID = Object.GetIndex();
105
106         RecMap.insert(
                typename mmtype::value_type (Object.GetName(), Object) );
107      }while(File.tellg() < FileLen);
108
109      File.close();
110   }
111
112   template <class T>
113   void Database<T>::SaveMap()    //將容器中的物件輸出至檔案
114   {
115      typename mmtype::const_iterator iter;
116      T Object;
117
118      File.open(FileName.data(), ios::out | ios::binary | ios::trunc);
119
120      for(iter = RecMap.begin() ; iter != RecMap.end() ; ++iter)
121         File.write((char *) & iter->second, rec_size);
122         //將物件寫入檔案
123
124      File.close();
125   }
```

與資料操作相關之方法

從 Database 類別操作資料的方法裡，將可體會到使用 map 容器的好處。不論新增、刪除、修改或者查詢，大多藉由 map 容器的成員完成，Database 類別的方法僅負責資料準備工作。

```
'摘自 Database.h
044   template <class T>
045   void Database<T>::Insert(T & Object)    //將物件插入容器
```

```
046   {
047      MaxID++; //將容器中最大的編號加一
048      Object.SetIndex(MaxID);     //設定欲插入物件的編號
049
050      RecMap.insert(mmtype::value_type (Object.GetName(), Object) );
051      //將物件插入容器中
052   }
053
054   template <class T>
055   T * Database<T>::Query(string ObjName) //從容器中刪除物件
056   {
057      typename mmtype::iterator iter;
058
059      iter = RecMap.find(ObjName);     //尋找物件
060
061      if(iter == RecMap.end())    //如果找不到物件則丟出例外
062      {
063         string c_name = typeid(T).name();
064         int NameLen = c_name.size();
065         string s = "Don't find this "
066                    + c_name.substr(6, NameLen) + "'s data!\n";
067         throw(s);
068      }
069
070      return &(iter->second);     //傳回找到的物件
071   }
072
073   template <class T>
074   void Database<T>::Delete(string ObjName) //查詢容器中的物件
075   {
076      Query(ObjName);  //尋找物件，如果找不到則會丟出例外
077      RecMap.erase(ObjName); //輸入欲刪除物件的名稱
078   }
…     ……
127   template <class T>
128   void Database<T>::ShowAllData()     //顯示容器中所有元素的資料
129   {
130      typename mmtype::iterator iter;
131      T Object;
132
133      for(iter = RecMap.begin() ; iter != RecMap.end() ; ++iter)
134         (iter->second).ShowData();    //顯示物件資料
135   }
```

使用 Database 類別的限制

對於使用 Database 類別操作物件儲存的類別，如：Book 類別、Reader 類別，都必須提供下列成員函數做為介面，讓 Database 類別可以取得操作該類別的相關資料。因此，當我們在圖書館裡系統中增加了一個需要利用 Database 類儲存物件的類別時，我們必須提供這些成員函數讓 Database 類別使用。下表我們將整理出這些成員函數的相關資料。

使用 Database 類別必須提供的成員函數	傳入的參數	傳回資料型態	用途	使用的方法
GetFileName()（類別成員）	無	char*	提供 Database 類別取得儲存物件之檔案名稱的介面	Database 的建構子
GetName()	無	char*	取得物件的名稱	Database::Insert Database::InitiMap
SetIndex()	物件的編號	無	設定物件的編號	Database::Insert