11. Virtual Functions and Polymorphism in C++

對於同一繼承架構中之不同型別物件的操作，我們雖可將其均視為基本類別的物件來做整理，但如欲使物件能表現出其本身應有的行為，仍需透過其所屬型別的指標或參考來做個別的操作，一般的做法是使用 switch 等條件判斷式，根據物件之屬性判斷出物件之型別，依型別來採取適當的行動，新增新的類別，也要新增其對應的個別操作。

使用 C++之虛擬函式與多型的功能，可使的程式的寫作更具綜合性，讓我們將階層中的所有類別之物件，都可以視為基本類別的物件，以一致的方式來操作使用這些類別共同的介面，而程式會自動判斷物件之實際類別，並根據所操作使用物件之實際類別，做不同的反應。

11.1 Virtual Functions

說明:

就之前的介紹，我們可以將衍生類別，當作其基本類別來使用，也就是將衍生類別的物件，指定給其基本類別的指標或參考。透過基本類別的指標或參考來操作一衍生類別的物件，其表現出的行為是如同一基本類別物件一樣(執行基本類別之程式碼)，例如呼叫函式成員，其所執行的是基本類別內原函式的定義，而非衍生類別內修改或重新定義過的函式(如: oop_ex94.cpp 中對 print 的呼叫)。

C++提供了虛擬函式的功能，其做法是將一函式成員於基本類別內宣告為虛擬函式(使用 virtual 關鍵字)，對於虛擬函式，如使用基本類別的指標或參考來指到衍生類別的物件，並使用基本類別的指標來呼叫虛擬函式，系統將透過動態連結的方式，執行被參照的物件其實際類別(衍生類別)所定義(override)的虛擬函式，而非操作其的指標或參考所屬類別(基本類別)的虛擬函式。

例如範例 103，Circle、Triangle、Rectangle 與 Square 等，均是衍生自 Shape 的子類別，語意上，它們都是 Shape 的一種，可統稱為 Shape，但各自有不同的描繪自己(draw)的能力。在 C++程式中，我們亦可將這些不同類別的物件都視為 Shape，共同整理為 Shape(指標)之陣列，但為使得這些 Shape 能表現出不同的能力，即需使用虛擬函式的功能，使的無論是描繪哪一種圖形，只要是執行其基本類別 Shape 的 draw 虛擬函式(以 Shape 的指標呼叫 draw)，程式執行時即可動態決定出執行 draw 指令的實際物件為何類別，並執行其 draw 之定義，不需類別判

斷與 downcasting 等動作，即可以統一的方式操作不同類別的物件，並表現出其應有的行為。

程式範例: oop_ex103.cpp　oop_ex103b.cpp

注意要點:

1. 凡是一函式成員被宣告為虛擬函式，衍生類別中之重新定義(override)均將繼承其設定，自動成為虛擬函式，不需一一使用 virtual 重新宣告。
2. 如一衍生類別未重新定義虛擬函式，則所執行的是與此類別上層最接近的基本類別中的虛擬函式定義。
3. 虛擬函式只能宣告於類別的函式成員。
4. 衍生類別隻指標可用範疇解析運算子「::」，指定呼叫基本類別的函式成員。
5. 解構函式可以(最好)宣告為 virtual。(ex103b)

11.2 Pure Virtual Functions and Abstract Classes

在許多的情況，我們所定義的類別不一定需能產生其物件，此類別為繼承其之衍生類別之統稱，例如 Shape，或 Vehicle，其為一抽象、統合的名稱，並沒有任何一種圖形稱為 Shape，或任何一種交通工具叫 Vehicle，其各別是各種圖形，交通工具的統稱，這樣的類別，我們稱為抽象類別(Abstract Classes)。這種類別通常是作為繼承關係中，各種性質相近類別的基本類別，故也常稱為抽象基本類別(Abstract Base Classes)。

抽象類別(Abstract Classes)無法產生實體，其主要作用，是提供/定義/規定共同的能力(介面)，或實作內容，讓其各衍生類別繼承，並依其本身的性質實作這些共同的介面。故繼承自同一抽象類別的衍生類別都具有定義於抽象類別的能力，但表現出的行為並不相同，例如 Shape 類別定義了 draw 為函式成員，繼承其之各種圖形均繼承之並對 draw 提供不同的實作，Shape 為抽象類別，不需提供 draw 的實作，只要定義 draw 為其下衍生類別所應共同有的能力即可。

抽象類別的做法，是將類別的一個或多個虛擬函式宣告為純虛擬函式(pure virtual function)，此類別即成為抽象類別，無法產生實體(指標/參考可)，純虛擬函式之宣告，是在函式成員之宣告時，將其指定「= 0」，不需提供其實作內容。例如：

```
class shape
```

```
{
public:
        virtual void move( ) = 0;
        virtual void draw( ) = 0;
};
```

許多物件導向的程式設計，類別階層的起始(root)類別，甚自最上幾層，都是使用抽象類別。例如 Shape，以下再分為 2DShape、3DShape，這些都是抽象類別，之下才是各種可實體化的具象類別(concrete class)。

程式範例: oop_ex104.cpp

注意要點:

1. 抽象類別，無法產生實體，但可產生其指標/參考。
2. 衍生類別繼承了於上層中尚未定義任何實作內容的 pure virtual function，如此衍生類別沒有對其提供實作加以定義，則此函式仍為純虛擬函式，此衍生類別亦為抽象類別。

11.2 Polymorphism

除了資料封裝與繼承，C++另提供了多型(Polymorphism)之功能，讓彼此具有繼承關係的不同類別，能對相同的程式命令(以共同的父類別之指標或參考呼叫函式成員)，有不同的反應。多型是透過虛擬函式之功能所達成，當使用基本類別之指標或參考，呼叫虛擬函式來達到某項功能，C++會選擇相關物件的適當類別，呼叫其中正確的重載函式(redefined function)。

使用 C++多型之功能，程式(例如電腦繪圖程式)在物件之使用與管理上，對於彼此具有繼承關係的不同類別(例如 circle, triangle, square)之物件，可以一致地都使用基本類別(例如 shape)的指標或參考，來管理這些類似的物件，對於物件之操作，也可以一致地都透過基本類別(例如 shape)的指標或參考來呼叫(例如 shape_ptr->draw)，物件自動表現出屬於其之應有反應。故在物件之使用與管理上，我們不需要去考慮分類/分辨這些類似的物件的問題，可均把他們當作是基本類別的物件來操作，物件自動會表現出所屬類別應有的反應。

程式範例: oop_ex105a.h 、 oop_ex105b.h

在繼承上，我們可以在衍生類別中新增基本類別沒有的(虛擬)函式成員，這些沒有宣告於基本類別，非繼承來的虛擬函式，因為並非是基本類別的成員，故無法透過基本類別的指標或參考來呼叫。

程式範例: oop_ex106.cpp

注意要點:

1. 充分發揮多型之功能，是繼承關係的不同類別均是有相同的公開介面(函式成員)，這些介面均在基本類別中宣告為純虛擬函式，故每個物件的所有介面均可透過基本類別的指標或參考來呼叫。
2. 對於某類別才獨有的能力/行為，不適於宣告成基本類別的函式成員，讓所有類別來共同繼承並實作者，只能透過此類別之指標來呼叫，如物件被指定給基本類別的指標，需知其實際的類別為何，並作 downcasting。

使用多型之功能，函式之接收參數可使用基本類別的指標或參考，來接收繼承其之不同類別的物件，函式之中只要以基本類別的指標或參考來操作，物件自動會表現出所屬類別應有的反應，不需分類/分辨傳入物件為何類別。

程式範例: oop_ex107.cpp

多型可提升程式的擴充性，新增新類別至繼承架構中，除了產生新物件之程式，需用新類別之名稱外，管理與操作此類物件之程式並不需要修改，因為新類別之物件亦可當作為基本類別之物件來管理操作。

程式範例: oop_ex108.cpp

多型之整理如下：

1. 能表現出多型的虛擬函式，其參數之型態、個數，需與基本類別所定義的完全相同(基於重載，呼叫型式要相同，不同則為覆載)。
2. 虛擬函式通常於最上層類別，宣告為 virtual，甚至為 pure(= 0)。
3. 具有虛擬函式之類別稱為 polymorphic type，愈使程式表現出多型的行為(polymorphic behavior)，函式需宣告為 virtual，且物件需透過指標或參考來操作。
4. 繼承架構下，絕大多數函式成員均是虛擬函式。