# 10. Inheritance in C++

我們已介紹了物件導向程式的第一個主要特性，即程式可模組化成為類別(物件)，類別具有資料封裝的特性。接下來我們要介紹物件導向程式的另一個主要特性，那就是類別具有繼承的功能。繼承就是重複使用程式，將現有的類別的屬性與行為，加以修改或重新定義(overridden)，即可製作出新的類別，達到軟體再使用的目標。

當我們要產生一個新類別，如其與某一個已經定義好的類別性質有許多相似之處，只有部分不同需修改或加強，我們可以不需以重新製作的方式來定義這個新類別的資料成員與函式成員，只需使用繼承的功能，指定其去繼承此一已經定義好的類別(基本類別, base class)，經由繼承某基本類別來產生的新類別稱為其之衍生類別(derived class)。

衍生類別會繼承基本類別所有的資料成員與函式成員，當繼承而來的函式成員不適於衍生類別時，就可以在衍生類別之定義內將其重新定義(overridden)，並可進一步加入新的成員(資料或函式)，以符合衍生類別應有的屬性與行為。

繼承功能真正強大之處，在於我們可以在衍生類別裡定義基本類別所沒有的資料及功能，或取代改良繼承自基本類別的功能，將其功能增強、取代和改良。換句話說，衍生類別通常會增加屬於他自己的資料成員與函式成員，故衍生類別一般都比其基本類別要來的大，但對的其也變的比較特別化，所能表示的物件範圍較為狹隘。

每一個經由繼承產生的衍生類別又可以被繼承成為其下衍生類別的基本類別，此外，對於 C++，一個衍生類別可以繼承一個以上的基本類別，故 C++之類別繼承為階層式的交錯樹狀架構。

C++提供了三種繼承方式，public、protected 與 private，我們主要將介紹 public 繼承(public inheritance)，並簡要說明其他兩種繼承。對於 public 繼承，衍生類別的每一個物件也可視為其基本類別的物件，但是，基本類別的物件則不是衍生類別的物件。

對於物件間的關係，主要有「is a」和「has a」關係之設計；「is a」關係就是繼承，衍生類別之物件也可視為其基本類別的物件；「has a」關係就是合成，意指某類別之物件使用其他類別的物件做其成員。

## 10.1 Base Class and Derived Classes

一類別的物件常常也可視為另一類別的物件(其中一種或其下分支)，例如，研究生是學生的一種，我們可利用繼承的功能，將學生定義為基本類別，而將研究生定義為學生的衍生類別。

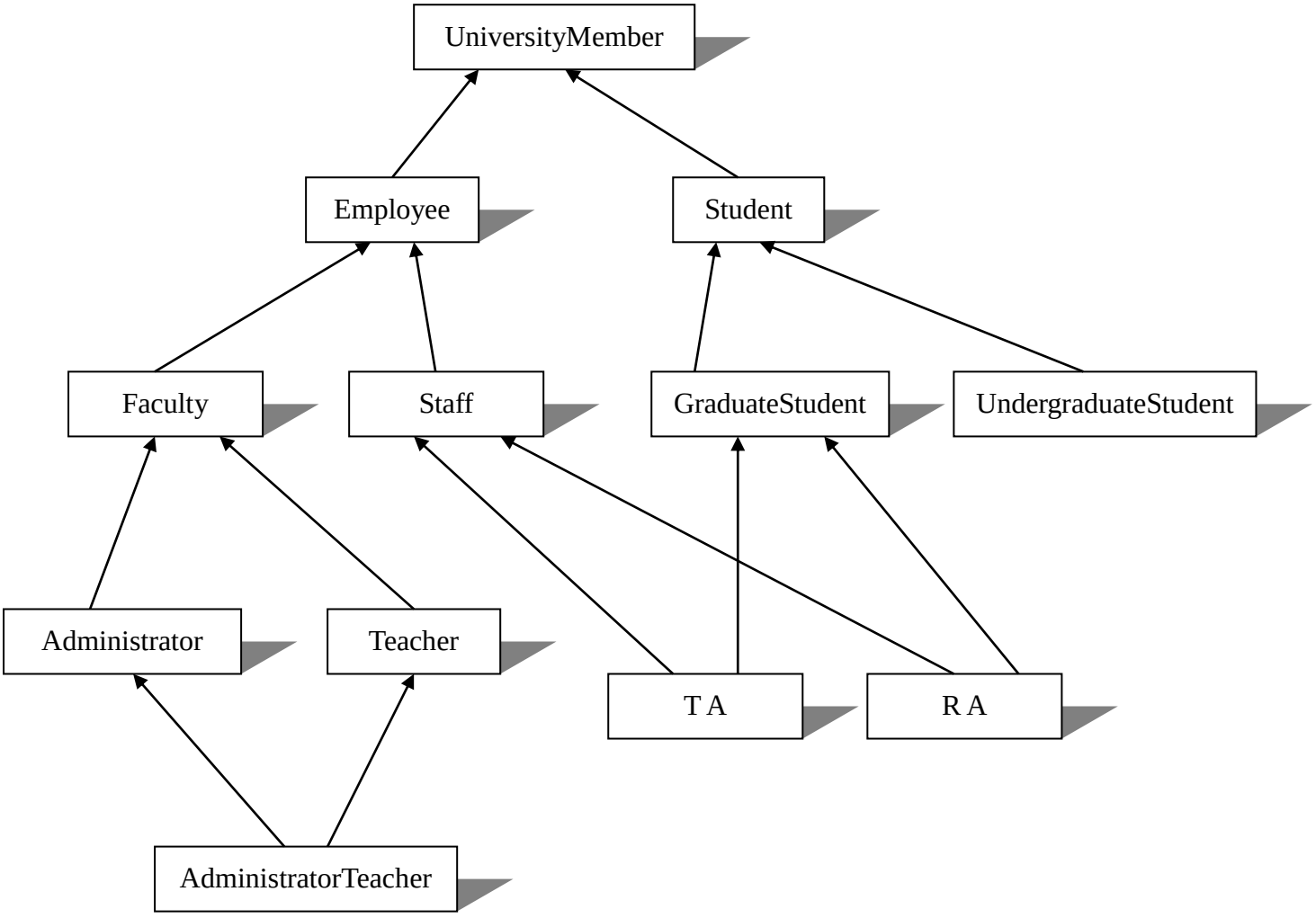
衍生類別的每一個物件同時也屬於此類別之基本類別的物件，但反之並不成立 – 基本類別的物件並不是此類別之衍生類別的物件。例如，在繼承的關係上，研究生是學生的一種特殊類型，反之學生並不一定是研究生，語意上是如此，也為類別繼承的規則。

有時，「基本類別」又稱為「父類別」，「衍生類別」又稱為「子類別」。有關於基本類別與衍生類別的例子如下：

| **Base Class** | **Derived Class + Base Class** | **Derived Class** |
|---|---|---|
| Student | GraduateStudent | MasterStudent<br>PhDStudent |
| | UndergraduateStudent | FullTimeStudent<br>PartTimeStudent |
| Shape | 2DShape | Circle<br>Triangle<br>Rectangular |
| | 3DShape | Sphere<br>Cube |
| Vehicle | Car<br>Bus | |
| Employee | ContractEmployee<br>HourlyEmployee | |
| Account | SavingAccount<br>CheckingAccount | |
| Matrix | FullMatrix<br>SymmetricMatrix<br>SparseMatrix | |
| Element | TrussElement<br>BeamElement<br>BeamColumnElement<br>PlateElement | |

類別繼承為階層式的交錯樹狀架構，一類別可如先前獨立存在，但如其位於繼承的架構中，則其不是提供本身屬性與行為給其下類別的基本類別，就是繼承其他類別屬性與行為之衍生類別，或二者皆是。

單一繼承(single inheritance)是指一個衍生類別是由繼承單一個基本類別而來。如一衍生類別繼承了多個基本類別，則稱為多重繼承(multiple inheritance)。




「public 繼承」的格式如下：

```
class Y : public X
{
    新增之資料成員;
public:
    新增或重新定義之函式成員;
}
```

衍生類別會繼承基本類別所有的資料成員與函式成員(需注意這些成員並沒有出現於衍生類別的程式碼中，易弄混)，於「public 繼承」的情況下，衍生類別之函式成員只能直接存取或呼叫所繼承來的 public 與 protected 成員，但不能直接存取或呼叫所繼承來的 private 成員。

通常會有多個衍生類別所共同繼承自一基本類別的情形，則這些衍生類別同屬於基本類別之一類，例如，研究生與大學生都屬於學生。我們可將這些衍生類別所共有的資料與功能，定義於基本類別，如此衍生類別們就不必對這些共有的部分個別重複定義，例如，將研究生與大學生都有的姓名與學號定義於學生基本類別內。而特屬於衍生類別之屬性與行為則定義於衍生類別，例如，將研究生才有之組別與研究室定義於研究生衍生類別。

衍生類別繼承自基本類別的的 public 函式成員，可能會不符合衍生類別的需要，故我們可以在衍生類別內修改或重新定義(overridden)這些函式成員，使之合乎衍生類別應有的之意義與行為。衍生類別中函式成員之重新定義，常會先呼叫已定義於基本類別的 public 函式成員，再加上專屬於此衍生類別的部分；或者，完全將此函式成員重新定義亦可。

故繼承同一基本類別之類別的共同處即基本類別之屬性與行為，經由 public 繼承所產生的任何衍生類別之物件，均同時亦為基本類別的物件(當物件以指標或參考操作時)，因為對外部程式而言，其完全具有基本類別應有的屬性與行為，且存取權限與基本類別之設定完全相同。但基本類別之物件並非衍生類別之物件，因為衍生類別通常有屬於此衍生類別獨有的屬性與行為。

例：當 Manager 以 public 繼承了 Employee，Employee*可指到並操作 Manager 物件。

```
Manager * Mptr = new Manager (…. );
Employee * Eptr = Mptr;
```

以下則程式則不可被編譯器接受。

```
Employee * Eptr = new Employee (…. );
Manager * Mptr = Eptr;
```

### 10.1.1 Constructor and Destructor for a Derived Class

基本類別之建構函式、解構函式、與拷貝指定運算子不會繼承給衍生類別。不過衍生類別之建構函式、解構函式、與拷貝指定運算子，可以呼叫基本類別之建構函式、解構函式、與拷貝指定運算子。

衍生類別通常需定義自己的建構函式，衍生類別的建構函式，必須要呼叫其每個基本類別的建構函式，來初始化繼承自基本類別的資料成員，明確地呼叫基本類別的建構函式，其格式如下，如未明確地呼叫，則會自動呼叫基本類別之預設建構函式。

```
class Employee
{
    string name;
    int deptID;
    ……
public:
    Employee(string n, int d)
    {
        name=n;
        deptID = d;
    }
    print( );
    …….
};

class Manager : public Employee
{
    int rank;
    …….
public:
    Manager(string n, int d, int r)
        : Employee(n, d),          // initialize base members
          rank ( r ){ }            // initialize local members
    print( );                      // override “print”
    …….
}
```

衍生類別物件之建立的程序是：(1)先呼叫基本類別之建構函式初始化繼承來的 base members，(2)之後執行衍生類別之 member initializer，(3)最後執行建構函式{ }中之內容來設定 local members。

至於解構函式方面，在執行完衍生類別的解構函式中的命令後，系統**必定會**自動呼叫其基本類別的解構函式。故衍生類別物件之釋放為與物件建立相反的程序，(1)先釋放 local members，(2)然後執行基本類別之解構函式釋放 base members。

### 10.1.2 Calling/Accessing the Base Members from Derived Classes

衍生類別的函式成員可以存取或呼叫繼承自基本類別的 public 或 protected 資料成員或函式成員，其做法是使用基本類別的名稱與範圍解析運算子，例如：

```
void Employee::print( )
{
    cout<<name<<" "<<deptID<<endl;
}

void Manager::print( )
{
    Employee::print( );        //calling base member
    cout<<rank<<endl;
}
```

### 10.1.3 Overriding the Function Members from Derived Classes

由基本類別繼承來的函式成員，常不符基本類別的需要。衍生類別可將繼承自基本類別的函式成員，依衍生類別的需要修改或重新定義，稱為 override，其做法是我們須於衍生類別中重新宣告此一繼承來的函式成員，並提供其新的定義，例如上例之 print。

Override 的做法，可先呼叫基本類別中被 overridden 的函式成員，再補充屬於衍生類別的內容，如上例之 print；或完全重新定義，如需存取基本類別的 private members，需有 public 或 protected 的存取函式。

**程式範例: oop_ex93.cpp**

**注意要點:**

1. circle 類別以 public 的方式繼承了 point 類別，這項指定出現於 circle 類別定義的第一行，冒號(：)代表了繼承。
2. 衍生類別(circle)會繼承所有基本類別(point)的成員(x, y, moveTo, distTo, print…)，但於衍生類別的建構函式中，無法直接存取基本類別的私有成員(x, y)，為了對繼承自基本類別的私有成員初始化，必須要使用基本類別的建構函式，呼叫格式見範例。至於呼叫順序，是基本類別的建構函式先被呼叫，再執行衍生類別的建構函式。
3. 衍生類別可以直接使用繼承而來的 public 函式成員(如 moveTo, distTo)，並定義專屬於衍生類別的新函式成員(如 area, perimeter, getR)。
4. 於衍生類別中，如果要存取基本類別之私有成員，唯有透過基本類別所提供的函式成員(例如 getX, getY)，如基本類別沒有提供這些存取函式，則無法達成。
5. 如需要將繼承自基本類別的函式成員，依衍生類別的需要修改或重新定義(例如 print)，則我們須於衍生類別中重新宣告此一函式成員，並提供其新的定義，修改或重新定義後的函式將於衍生類別中取代繼承自基本類別的定義。
6. 如果對某一繼承自基本類別的函式成員，我們不需完全更改其原本的定義，只需要加入新的功能(例如 print)，我們可以不必重寫已定義於原基本類別之原函式的部分，而只要呼叫此一原函式，再加上新增的部分即可。
7. 在執行完衍生類別的解構函式中的命令後，系統必定會自動呼叫其基本類別的解構函式，因此，於衍生類別的解構函式中，不需對繼承自其基本類別的資料成員做處理，此項工作就交給基本類別的解構函式。至於呼叫順序，是衍生類別的構函式先被呼叫，再執行基本類別的構函式。

**程式範例: oop_ex94.cpp**

**注意要點:**

1. 一衍生類別可以當作是其基本類別來使用，例如：將一衍生類別物件指定給其基本類別的指標、參考，或是當作參數，輸入接受其基本類別的函式內(可參考之後將介紹的衍生類別之拷貝建構函式與拷貝指定運算子)。
2. 此之所以可行，是因為衍生類別繼承了其本類別所有的資料與功能。
3. 但反之，基本類別不可以作為其衍生類別來用。因基本類別不具備衍生類別

中所新宣告定義的資料成員與函式成員(如 radius, area)。

4. 指定給為其基本類別的指標或參考之衍生類別物件，如透過基本類別的指標或參考來操作，其表現出的行為將如同其基本類別一樣，而非於衍生類別中新定義的行為(例如 print)，也不具新增的功能(無法執行 area)。Note: 與下一章介紹之 Virtual Function 比較。
5. 基本類別的指標，可強制轉換(cast)成衍生類別的指標，使用轉換後之指標來操作衍生類別物件，即回復衍生類別的行為。

### 10.1.4 Casting Base-Class Pointers to Derived-Class Pointers

基本類別之物件並不可視為衍生類別之物件，不過程式設計者可明確使用強制轉換，將基本類別之指標轉換成為衍生類別之指標，此稱為 downcasting。需注意的是要確認指標所指物件的實際型別為何，雖可以用 downcasting 將基本類別之物件指定給衍生類別之指標，但存取不存在的成員會造成執行邏輯上的錯誤。

**程式範例: oop_ex95.cpp**

**注意要點:**

1. 請注意執行的結果，以 circle 指標操作 point 物件，由於 point 物件並沒有定義 radius 等成員，程式則將 radius 預期所在記憶體位址之現存值，直接輸出。此有如使用超出陣列範圍之索引值對陣列做存取之情況。

## 10.2 Protected Member

**說明:**

protected 成員保護程度介於 private 與 public 成員之間，一基本類別的 protected 成員，可被其衍生類別的成員或朋友直接存取，有如 public 成員。但對於其他的類別，則無法自由存取，有如 private 成員。

於「public 繼承」的情況下，衍生類別對其繼承而來之成員的存取權限：

(1) public: 所有程式包括其衍生類別均可直接存取某基本類別的 public 成員。
(2) protected: protected 為特別針對繼承情況的存取設定，衍生類別的函式成員與朋友可直接存取其基本類別的 protected 成員，而其他部分式則不可直接存取某基本類別的 protected 成員。
(3) private: 與所有非基本類別的程式一樣，衍生類別不能直接存取其基本類別的 private 成員，衍生類別對於基本類別的 private 成員之存取，需透過基本類別提供宣告為 public 或 protected 之存取函式。
(4)需注意的是，朋友函式並非基本類別之成員，故無法被繼承。

**程式範例: oop_ex96.cpp**

**注意要點:**

1. 衍生類別的建構函式可直接初始化 protected 成員，但一般還是建議使用基本類別的建構函式。
2. 非衍生類別之函式成員，例如主程式內無法存取 protected 成員。

## 10.3 public、protected and private Inheritance

C++有 public、protected 與 private 三種繼承方式，使用 protected 與 private 繼承是很少見的。不同的繼承方式會影響成員被繼承後，於衍生類別中之權限設定，其規則如下：

| 基本類別中之成員設定 | 繼承後於衍生類別之權限設定(對外) / 衍生類別函式成員存取方式(對內) | | |
|---|---|---|---|
| | public 繼承 | protected 繼承 | private 繼承 |
| public 成員 | public / 可直接存取 | protected / 可直接存取 | private / 可直接存取 |
| protected 成員 | protected / 可直接存取 | protected / 可直接存取 | private / 可直接存取 |
| private 成員 | hidden / 需透過基本類別提供之存取函式 | hidden / 需透過基本類別提供之存取函式 | hidden / 需透過基本類別提供之存取函式 |

**程式範例: oop_ex97.cpp oop_ex98.cpp oop_ex99.cpp**

**注意要點:**

1. protected 與 private 繼承均不是「is a」關係。
2. base class 可分為 direct base class 與 indirect base class：direct base class 為宣告衍生類別時，明確以「：」指定；indirect base class 並非於宣告衍生類別時明確指定，而是衍生類別時經由二或多層繼承，間接由 direct base class 繼承來的。

## 10.4 Copy Constructor and Copy Assignment Operator for Derived Classes

衍生類別的拷貝建構函式與拷貝指定運算子也必須要呼叫其基本類別的相對應函式來初始化 base members，其呼叫方式如下：

```
class Employee
{
    ……
public:
    Employee(const Employee& other){……..}
    Employee& operator=(const Employee& other){……..}
    …….
};

class Manager : public Employee
{
    …….
public:
    Manager(const Manager& other) : Employee(other){……}
    Manager& operator=(const Manager& other)
    {
        Employee::operator=(other);
        …….
    }
    ……
}
```

**程式範例: oop_ex100.cpp**

**注意要點:**

1. 各位可注意到呼叫基本類別的拷貝建構函式與拷貝指定運算子時,所輸入參數是衍生類別的物件，由於衍生類別可當作是其基本類別來使用(指定給基本類別之指標或參考)，故此做法並沒有問題。

## 10.5 Friend Functions in Inheritance

適用於基本類別的朋友函式亦適用於其衍生類別。但其原因並不是衍生類別繼承了基本類別的朋友函式(朋友函式並非一類別的成員)，而是衍生類別之物件可當作其基本類別之物件來使用。

**程式範例: oop_ex101.cpp**

**注意要點:**

1. 朋友函式不能直接存取衍生類別專有的 private 成員(radius)，故可知其並不是 circle 的朋友，而只是 point 的朋友。故可以說一類別的朋友函式並不具有繼承的功能，因為其根本並不是類別的成員之一。

## 10.6 Relationship between Classes

### 10.6.1 Composition vs. Inheritance

類別間如為 public 繼承，則為「is a」之繼承關係，即衍生類別之物件可視為其基本類別之物件，例如 Manager 「is a」 Employee，Car「is a」 Vehicle.

如類別將其他類別之物件當做成員，則為「has a」之合成關係，例如 Team「has a」Person，Car「has a」Engine， 於合成關係下，物件可透過彼此公開之介面交互作用。

10.6.2 “Uses a” and “Know a” Relationship

除了繼承與合成，類別間還有「uses a」與 「know a」關係。例如 Person 與 Car 間非繼承與合成之關係，但 Person 可駕駛 Car，故其間為 Person「uses a」Car 之關係。

物件以指標或參考來參照到其他物件，稱為「knows a」關係，或稱為 association，表示物件間可透過指標或參考相關聯，物件間如為「has a」或「uses a」關係，均可以「knows a」方式相關聯。

## 10.7 Multiple Inheritance

C++之類別可以繼承自多個基本類別，這種產生衍生類別的方式稱為多重繼承(multiple inheritance)，此促成軟體之重複使用，但會造成許多混淆的問題。

多重繼承的宣告格式如下，繼承之所有基本類別均指定於「：」後，並以「，」分開。

```
class Derived : public Base1, public Base2, ….
{
      ……….
}
```

**程式範例: oop_ex102.cpp**

**注意要點:**

1. TA 包含了兩個 name 資料成員，一個繼承自 Staff，一個繼承自 Student，於 TA 中對其存取會有混淆的情形發生，需使用範圍解析運算子，將其區隔出來。對於沒有名稱衝突者，如 salary 與 level，則可直接存取。
2. 同樣的情形也發生於 TA 所繼承之 setDept 與 getDept 函式成員上。
3. 「is a」關係，也適用於多重繼承，例如 TA 之物件可視為 Student 之物件，也可視為 Staff 之物件。
4. 本範例為簡單的多重繼承，C++之多重繼承尚有其他的問題，例如 diamond hierarchies，未來將再做介紹。