6. Classes and Data Encapsulation in C++

基於類別之概念的提出，程式設計的單元由程序式程式中的函式，進化到物件導向程式中的類別。

類別(Class)是結構體的延伸，與結構體相同，類別也是程式設計者自訂的資料型態，將程式中彼此相關的資料與函式整合在一起，成為一個類別。類別與結構體是非常類似的，均為資料成員與相關函式的組合，格式也幾乎相同，惟類別有較多的功能，其一是支援資料封裝(Data Encapsulation)，此外是類別具有繼承(inheritance)的功能。

類別是一種自定資料型態的定義，其所產生的實際資料稱為物件(Object)。如結構體一樣，宣告定義了一類別之後，即可產生許多屬於此一類別的物件。但與結構體不同的是，由於資料封裝，產生一物件並對其資料指定初值需使用建構函式(Constructor)，如一物件不再使用，需使用解構函式(Destructor)來釋放屬於此物件的動態記憶體。

6.1 Access Control for Class Member in C++

類別(class)與結構體(struct)之主要不同之一，是類別的成員(資料或函式)可使用 private、public、protected 等關鍵字，設定其之存取權限。

private: 只可讓同類別本身所定義的函式成員直接存取。
A private member is accessible only to the methods defined for the class itself.

public: 任何部份之程式均可直接存取，有如結構體之成員。
A public member is visible everywhere in a program.

protected: 類別本身及其衍生類別可對其直接存取，其他部分之程式則不可。
A protected member is public to the class itself and its derived class, but private to the rest of program.

公開程度；public -> protected -> private

程式範例: oop_ex_b.cpp

6.2 Data Encapsulation(資料封裝)

說明:

我們之前所介紹的結構體，其內部的資料是完全公開的，其構成的資料成員或函式成員均可透過「.」或「->」運算子來被自由的使用或更改。

資料封裝的意義是指一類別的資料成員或函式成員的存取權限，可由類別本身的定義來訂定，一物件內的資料可設定為不可被外部(類別定義區塊之外)程式做自由的存取(private)，而被保護於此類別的內部(類別定義區塊之內)；或如同結構體一樣，開放給外部程式自由存取(public)。

如某類別的資料成員或函式成員被定義為 private，則這些成員為類別內部私有的成員，所有的私有成員只有此類別內部的函式成員之程式命令可以對其作存取，其他外部程式命令都不允許透過「.」或「->」直接存取私有成員。任何物件內之私有資料成員之存取，只能透過公開的函式介面。

如某類別的資料成員或函式成員被定義為 public，則這些成員為公開，公開的成員可任意為外部程式命令透過「.」或「->」直接存取，與結構體的成員相同。

一般而言，我們通常將一類別的資料成員定義為 private，而將其函式成員定義為 public。如成員未宣告其存取權限，則其預設值為 private。

資料封裝使得物件彼此間雖可透過事先定義好的介面(公開函式成員)進行溝通，但卻無法知道其他類別運作的實際內容(私有成員與函式內容)，有關實作的細節則被隱藏在類別裡面。

格式:

```
class  類別名稱
{
private:
      私有成員宣告;
public:
      公開成員宣告;
};
```

例:

```
class User
{
private:
    string name;
    string id;
    int age;
    int tel;
public:
    void readData();
    void printData();
};
```

程式範例: oop_ex49.cpp

注意要點:

1. 成員未宣告其存取權限，則使用預設值 private。
2. 且於結構體相同，宣告時不能為資料成員設定預設值。
3. public:與 private:標籤可以交叉重複使用，但建議將所有的 private 成員列再一起宣告，之後再一起宣告 public 成員。
4. 由 oop_ex49.cpp 可見，private 成員無法被外界直接存取，因此無法對其直接指定初值。如果為了成員的初始化而將其設定為 public，則變為與結構體相同，失去了資料封裝的意義。
5. 私有成員初始化解決方法：提供公開的介面(函式成員)來負責私有成員的初始化。

程式範例: cpp_ex50.cpp

1. 加入函式成員到類別的方法，與結構體完全相同。唯對類別的函式成員，亦須設定其使用權限為 public 或 private，public 之函式成員可被外界程式命令呼叫使用，private 的函式成員則只可供內部的其他函式來使用。
2. 對於私有資料成員的初值設定或輸出列印等存取之需求，可於類別內定義專門的 public 函式成員來做處理。
3. C++為類別保留了專門的函式(與類別同名之函式)，稱為建構函式(Constructor)來專門處理物件的私有資料成員之初始化。

程式範例: oop_ex51.cpp, oop_ex52.cpp, oop_ex53.cpp，oop_ex_c.cpp，oop_ex_d.cpp

1. 一般而言，我們通常將一類別的資料成員定義為 private，而將函式成員定義為 public。
2. 如此一來，一類別內部的實際製作可與外界獨立，只提供函式成員給外界使用。此設計可將隱藏類別的實際製作，使程式模組化，易於管理與擴充，由於資料成員不可為外界直接存取，對一類別程式的維護大部分只需考慮其本身。
3. 此並可增進程式的易修改性，對程式中某一類別內部的製作作變更，對其他類別的定義製作一般沒有影響，除非此類別的 public 函式介面有調整，但資料封裝亦可將此影響降到最低，外部程式只需針對介面的改變作調整。
4. 物件導向程式的架構為物件的組合，物件之間資料獨立，但透過彼此公開的函式介面交互作用。此使程式模組化增進了程式的延展性，進一步發展程式的方式可於各類別內做個別延伸，不影響其他類別，或者加入新類別與現有類別合作互動。
5. 使用公開函式成員來設定私有資料成員，可防止資料成員被設定了不合理之值，ex:　u.age = -5;
6. 函式成員應避免以傳位址法或傳參考法的方式，傳回私有資料成員，而破壞資料封裝。

6.3 Constructors (建構函式) in C++

說明:

產生物件時，其資料成員可經由建構函式(constructors)來進行初值設定。建構函式是一個與類別名稱同名的函式成員。定義了建構函式後，每當有此類別的物件產生(動態或靜態)時其建構函式(之一)會被自動的呼叫。建構函式主要用來對物件初始化(設定資料成員的初值 + 必要之動態記憶體配置)。

我們知道，類別的資料成員是不能在類別定義時設定初值的，他們可在此類別的建構函式內初始化，或在物件產生後再來設定其值，像之前的例子，一般情況均會使用建構函式，提供類別必要的建構函式可視為物件導向程式的規則。

建構函式也是函式的一種，其性質與一般函式相同，例如也具有覆載(C++ and Java)與預設參數初值(C++ only)的功能，但也有其特殊之處，其一是必須與類別同名。

此外，建構函式不能有回傳值，故不能也不需指定回傳值型態。建構函式之參數列通常用來輸入用來初始化的值，當宣告/動態記憶體配置產生某一類別的物件時，我們可將欲設的初值用括號括起放於物件名稱的右邊，這些值如同函式呼叫的參數傳入建構函式。(物件的宣告/動態記憶體配置即同時呼叫建構函式)

類別可使用函式覆載(function overloading)的方式提供各種不同的建構函式以供不同需要使用。啟動那一個建構函式則視呼叫的參數，根據函式覆載的規則而定。

格式:

```
類別名稱(參數列)
{
    初值設定;
}
```

例:

```
class User
{
    ……
public:
    User(string n, string i, int a, int t)
    {
        name = n;
        id = i;
        age = a;
        tel = t;
    }
    …….
}

User userObject(“Mike”,”A123456”,20,7777777);
User* userPtr = new User(“Mike”,”A123456”,20,7777777);
```

程式範例: oop_ex54.cpp (vs. oop_ex51.cpp)

注意要點:

1. 建構函式可以有函式覆載(function overloading)，其作法與規定都與一般函式相同，只要參數個數不同或資料型態不一樣，即是合法的覆載格式。
2. 建構函式不可定義傳回值型態。
3. 當一個建構函式的參數列沒有任何參數時，此建構函式稱為預設建構函式(Default Constructor, or called No-Arg. Constructor)，如產生物件而不輸入任何初值時，則使用預設建構函式。
4. 一類別至少要有一個建構函式，如沒有定義任何建構函式的話，編譯器會自動為您產生一個預設建構函式(沒有任何參數列)，這個預設建構函式不會執行任何成員初始化的動作，所以在物件產生後其內容不保證為合理的。
5. 編譯器只有在一類別完全沒有任何建構函式時才會自動產生沒有內容的建構函式，如已有其他建構函式存在則不會產生。

程式範例: oop_ex55.cpp

注意要點:

1. C++之建構函式可以對其參數列預設初值，其作法與規定都與一般函式相同。
2. 如對參數列中所有的參數都設有預設值，則其等於包含了預設建構函式。

6.4 Destructors (解構函式) in C++

說明:

相對於建構函式，解構函式(Destructors)是當一物件不再使用時，用來清除此物件的記憶體配置，主要是要釋放在執行建構函式值時所配置的動態記憶體。

與建構函式相同，解構函式是類別的一個特別的函式成員，其名稱是以~加上類別的名稱。解構函式不允許接受任何參數也不允許傳回任何值，故無覆載的問題，每一種類別只能有一個解構函式。

解構函式的呼叫時機：(1)當一物件離開其有效範圍(scope)時(如函式呼叫結束或程式結束)，即自動執行解構函式。(2)如物件之產生是使用動態記憶體配置，則須由使用者來呼叫解構函式釋放記憶體。呼叫方法為 delete 物件指標，其中物件指標儲存此物件之記憶體位址的指標。

解構函式的內容只需將於執行建構函式時有做動態記憶體配置的資料成員用 delete 釋放其記憶體配置，對一般變數之資料成員不須有任何處理。除非我們於建構函式中作動態記憶體配置(有使用 new)，簡單的類別不需使用到解構函式，但一般習慣仍會將其定義出來(無內容)。

格式:

```
~類別名稱( )
{
    動態記憶體釋放;
}
```

例:

```
class User
{
    char* name;
    char* id;
    ……
public:
    User()
    {
        name = new char[15];
        id = new char[10];
        age = 0;
        tel = 0;
    }
    …….
    ~User()
    {
        delete [] name;
        delete [] id;
    }
}

User* userPtr = new User();
    ……
delete userPtr;
```

程式範例: oop_ex56.cpp

注意要點:

1. 除了物件之建立是使用動態記憶體配置的情形外，解構函式會於物件離開其有效範圍時被呼叫，其呼叫順序與建構函式之呼叫順序相反。
2. 物件之建立是使用動態記憶體配置的情形時，解構函式不會自動被呼叫，必須由使用者來呼叫以釋放動態記憶體。
3. 原則上一物件之解構函式只能執行一次，執行後物件即消失，對一已被釋放的物件呼叫其解構函式會發生錯誤。
4. 需特別注意，執行解構函式，其中所釋放(delete)的資料成員均已事先配置過記憶體。
5. 一類別只有一個解構函式，其內容是普遍針對每一個可能被用來產生物件的建構函式，故每一個建構函式均須有對應於解構函式內容的記憶體配置。
6. 常見的錯誤是有些建構函式沒有對某資料成員做動態記憶體配置，如以這些建構函式產生物件，執行解構函式釋放此資料成員會產生錯誤。

6.5 Standard Constructors for C++ Objects

6.5.1 Default Constructor (預設建構函式)

說明:

預設建構函式又稱為無參數建構函式(No-Arg. Constructor)，此建構函式沒有輸入參數。

如一類別如沒有定義任何建構函式的話，編譯器會自動為您產生一個預設建構函式，這個預設建構函式，沒有任何內容，不會執行任何初始化的動作。

但如果一類別已有其他建構函式，則編譯器不會自動產生此無內容的預設建構函式，產生物件只能使用已定義的建構函式，無參數之物件產生方式將因無對應之建構函式而發生錯誤。

格式:

```
類別名稱( )
{
    初值設定;
}
```

例:

```
class User
{
    char* name;
    char* id;
    ……
public:
    User( )
    {
        name = new char[15];
        id = new char[10];
        age = 0;
        tel = 0;
    }
    …….
}
```

程式範例: oop_ex54.cpp　oop_ex57.cpp

一般而言，我們會對每一個類別定義一個預設建構函式，以因應無參數產生物件之情況，例如產生物件陣列，如類別無預設建構函式，以下 statement 將不被接受。

```
User　user_list[10];
User　 u;
```

6.5.2 Copy Constructors (拷貝建構函式)

說明:

我們常會想要基於一存在的物件，拷貝產生另一新的物件，產生之新物件的資料成員與原物件具有相同的值。如欲由一物件複製產生出其他的物件，這些物件各有獨立的記憶體，且初值內容都與原始物件相同，則必須使用拷貝建構函式

(copy constructor)來達成。拷貝建構函式的作用相當於對一已存在的物件克隆(clone)。

拷貝建構函式對一個類別的定義而言並非必要，但對程式物件導向程度要求較高的設計師，會為其定義的每一個類別定義拷貝建構函式。

拷貝建構函式需接收一已存在的同類別物件作為其參數，為避免傳值法時拷貝資料的低效率，一般建議使用傳參考的方式，但為避免現有物件的資料於拷貝建構函式中不小心被更改，我們通常將此參數宣告為 const。

格式:

```
類別名稱(const 類別名&  物件參數)
{
    資料逐員拷貝，必要時需配置記憶體;
}
```

例:

```
class User
{
    char* name;
    char* id;
    int age;
    int tel;
public:
    User(const User& other)
    {
        name = new char[15];
        strcpy_s(name, other.name);
        id = new char[10];
        strcpy_s(id, other.id);
        age = other.age;
        tel = other.tel;
    }
    ……
}
```

程式範例: oop_ex58.cpp

注意要點:

1. 拷貝建構函式是將一物件傳遞到相同類別的物件中，由於輸入的物件與將產生物件屬同一類別，因此於拷貝建構函式中「可直接存取輸入物件的私有資料成員」。故我們可知資料封裝的界線是指不同的類別，而非不同的物件。因此，如果一類別中的函式成員程式中有此類別的物件時，(例如拷貝建構函式或相同類別物件的運算函式)，可直接對此物件的資料進行存取。
2. 對一般的資料成員，可直接進行拷貝，但如果物件的成員是指標，其對應到一快動態配置的記憶體，則需要先為此指標配置獨立的記憶體，再將傳入物件中指標成員所對應的資料拷貝進來。
3. 如直接以(=)對指標成員進行拷貝，所拷貝的是傳入物件中配置的記憶體位址，如此一來兩物件對應到相同的一塊記憶體空間及其中資料，產生的物件並無獨立的記憶體及資料，如此會產生問題。

程式範例: cpp_ex59.cpp

注意要點:

1. 使用「=」可將等號右邊物件的資料成員之儲存值拷貝設定給等號左邊的同類物件的資料成員，這種設定動作是以逐員拷貝(memberwise copy)的方式來進行，B=A 即將物件 A 的每一個成員考貝給物件 B 相對應的每一成員。
2. 當類別的資料成員含有動態記憶體配置的指標時，此逐員拷貝會產生如前述3.的問題，亦即拷貝的是指標所存之動態記憶體配置位址，而非配置記憶體中之資料。因此我們需要拷貝建構函式。

6.5.3 Copy Assignment Operator (拷貝指定運算子)

說明:

使用「=」可將等號右邊物件的資料成員之儲存值拷貝設定給等號左邊的同類物件的資料成員，這種設定動作是以逐員拷貝(memberwise copy)的方式來進行，B=A 即將物件 A 的每一個資料成員拷貝給物件 B 相對應的每一資料成員。

當類別的成員是配置有記憶體的指標時，則問題產生。因「=」對指標的意義並非其所參照記憶體內資料之拷貝，而是將指標指到等號右邊的記憶體位址，如此一來，二物件的指標資料成員參照到相同的記憶體。

對 C++而言，「=」運算子亦為一函式，除了拷貝建構函式外，程式設計者可用 C++之運算子覆載的功能，為其設計的物件定義拷貝指定運算子「=」，避免直接使用「=」時的問題發生。

格式:

```
類別名稱& operator=(const 類別名稱&)
{
    資料逐員拷貝，必要時需配置記憶體;
    傳回物件本身；
}
```

例:

```
User& operator=(const User& other)
{
    if(this != &other)
    {
        delete [] name;
        delete [] id;
        name = new char[15];
        strcpy_s(name, other.name);
        id = new char[10];
        strcpy_s(id, other.id);
        age = other.age;
        tel = other.tel;
    }
    return *this;    //需傳回物件本身
}
```

程式範例: oop_ex60.cpp

注意要點:

1. 就編譯器而言，obj1 = obj2 被解讀為 obj1.operator=(obj2)。
2. 一般而言，我們會對每一個物件定義一個拷貝指定運算子，以因應物件以等號相連之情況。
3. 嚴謹的 C++類別，必須要有的成員有：預設建構函式、拷貝建構函式、拷貝指定運算子、與解構函式。