5. Structures/Simple Classes in C++

結構體是程式設計者自訂的資料型態(data type)，一結構體是由多個彼此相關之基本資料型態之資料所構成的複合式資料型態。程式設計者可將程式中彼此相關、且類型不同的資料整合在一起，定義為結構體，此新的資料型態宣告建立後，便可產生屬於此結構體類型(定義)的變數(實體)，此有助於資料的管理。

結構體與陣列都屬於複合式的資料型態，即由多筆資料成員或元素所組成，但二者的性質並不相同。陣列是相同型別資料的集合，資料通常屬於一集合中的不同實體，而結構體是不同型態資料的集合，這些資料通常屬於現實中某一實體之屬性(attributes)或構成元，稱為此結構體之資料成員(data members)。

結構體是使用者自訂的資料類別，與基本的資料型別(primitive data types)一樣，除了可宣告產生某種結構體的變數(structure variables)外，亦可宣告產生某結構體的指標、參考及陣列，以及對其作動態記憶體配置。不同的是，某結構體需先行定義後才能產生此類資料，且一結構體資料為多筆資料成員所構成，對其內部構成成員各別進行存取，需使用運算子。

結構體相當於一成員均為公開(public)且不支援繼承的類別(class)。故以下內容均適用於 C++之類別或物件(object)。類別相當於結構體之定義(類別為物件之定義)，物件相當於結構體變數(物件為類別之實體)。

5.1 Structure Definitions

由於結構體是由使用者自訂(與自訂函式類似)，故需先將一結構體定義完成之後，才能依定義宣告產生資料。結構體之定義如下，struct 為 C/C++關鍵字，用來宣告結構體，之後為使用者指定之結構體類型之名稱(name)，而大括弧(body)中宣告了構成此結構體之資料成員(data member)。

結構體定義並不會佔用記憶體空間，而是建立了新的資料型別，可用於宣告結構體變數(物件)，產生如定義之資料。

格式:

```
struct  結構體名稱                          class  類別名稱
{                                           {public:
      結構體成員;          相當於=>               類別內容成員;
};                                          };
```

例:

```
struct User
{
    string name;
    string id;
    int age;
    int tel;
};
```

```
struct Sphere
{
    string id;
    double x, y, z;
    double radious;
    double r, g, b;
};
```

```
struct Time
{
    int hour;
    int minute;
    int second;
};
```

5.2 Structure Variable Declarations

於結構體定義了之後，我們即可使用結構體定義所指定的結構體名稱來產生屬於此結構體類型的變數/物件、指標或陣列。

格式:

```
結構體名稱     結構體變數名;
結構體名稱*    結構體指標名;
結構體名稱     結構體陣列名[n];
```

例:

```
User   userObject,   *userPtr,   userArray[10];
Sphere   sphereObject,   *spherePtr,   sphereArray[10];
Time   timeObject,   *timePtr,   timeArray[10];
```

結構體之定義與宣告亦可結合在一起。

```
struct User
{
    string name;
    string id;
    int age;
    int tel;
} userObject;
```

5.3 Accessing Members of Structure

當我們宣告了結構體變數/物件之後，如一般變數，需能存取(access)其資料，與一般變數不同的是，結構體是由多筆資料成員所構成，存取的對象應是組成其之各資料成員。(有如陣列之存取是針對陣列元素)。

對一結構體之各資料成員做存取，有兩種方式，其一是透過結構體變數/物件，並使用「.」運算子。另一是透過指到此結構體之指標(位址)，並使用「->」運算子。(有如陣列使用[ ])。

至於存取的是哪一個資料成員，則是由結構體定義時所指定的成員名稱做為識別。(有如陣列使用索引值)。

5.3.1 Accessing Members through Structure Variables/Objects

結構體變數 . 資料成員名
結構體陣列[i] . 資料成員名;

例:

```
userObject.name = “Micheal”;
userObject.id = “A111111111”;
userObject.age = 20;
userObject.tel = 1234567;

cout << userArray[3].name << endl;
cout << userArray[3].id << endl;
cout << userArray[3].age << endl;
cout << userArray[3].tel << endl;
```

程式範例: oop_ex40.cpp

注意要點:

1. 結構體定義時不能為資料成員設定預設值，亦不能做動態記憶體配置。
2. 結構體相當於一成員均為公開(public)且不支援繼承的類別(class)。

5.3.2 Accessing Members through Structure Pointers

結構體指標->資料成員名;

例:

```
userPtr->name = “Micheal”;
userPtr->id = “A111111111”;
userPtr->age = 20;
userPtr->tel = 1234567;
```

程式範例: oop_ex41.cpp

注意要點:

1. 透過指標(位址)來存取結構體內資料的方式經常會被用到，因為我們經常會以動態記憶體配置的方式產生結構體，動態配置後傳回之結構體記憶體位址通常指定是給一結構體指標。
2. 對一個指到某結構體的指標，我們用「->」來直接存取結構體的資料成員，等於先使用「*」依指取值，之後再用「.」來存取結構體內的資料成員。

5.3 Dynamic Memory Allocation for Structure

說明:

與基本資料型態的資料相同，我們亦可對自訂的結構體變數/物件與陣列做動態記憶體配置，同樣的，配置完成的記憶體區塊之起始位址(第一個陣列元素之起始位址)，將被傳回，一般的情形下，將會以一此結構體類型的指標來接收。

格式:

```
結構體名稱*   結構體指標 =   new  結構體名稱;   //配置一結構體
結構體名稱*   結構體指標 =   new  結構體名稱[陣列長]; //配置一陣列
```

程式範例: oop_ex42.cpp

5.3 Passing Structure Variables or Objects to Functions

說明:

與基本變數型態相同，我們亦可將結構體資料傳遞至函式。傳遞的方式同樣有傳值法、傳位址法與傳參考法等三種，使用這些傳遞法的特性亦相同。於函式中對傳入結構體之資料成員的存取方式，則視函式內之對應之參數型態而定。

例:

```
//函式定義
void printByValue(User u)
{
    cout<<u.name<<" "<<u.age<<" "<<u.tel<<endl;
}

void printByAddress(User* u)
{
    cout<<u->name<<" "<<u->age<<" "<<u->tel<<endl;
}

void printByReference(User& u)
{
    cout<<u.name<<" "<<u.age<<" "<<u.tel<<endl;
}

//函式呼叫
printByValue(userObject);
printByAddress(&userObject);
printByReference(userObject);
```

程式範例: oop_ex43.cpp

5.4 Function Members

說明:

在 C++所提供的結構體定義規則，允許我們將與某結構體相關的函式加入其中，成為結構體的一部分，所加入的函式稱為此結構體的函式成員(function members)。此功能使得我們除了可以將程式中相關的資料(屬性)整合外，也可以

更進一步將相關的函式(行為)整合封裝在一起，使得程式更易於管理與發展。

一結構體的函式成員，不需傳遞的動作，即可直接存取同一結構體的資料成員，有如其是宣告於此函式內，亦可直接使用同一結構體的其他函式成員。此是因為宣告在同一結構體內之所有成員均屬於同一範疇(struct 下之{ }程式區塊)。

函式成員的宣告必須再結構體的定義區塊內，但函式成員的定義可以寫在結構體的定義區塊之外，但必須在其定義之前，加上所屬結構體名稱與範圍運算子(::)。 ----- 註：Java 之函式(成員)並不支援如 C++之將原型宣告與定義分開之語法，故 Java 類別之函式成員是完全定義於其所屬類別之 body 內。

至於函式成員的呼叫方法，則與結構體之資料成員之存取方式相同。亦即如為一結構體之變數(或參考)，則使用「.」運算子；如為一結構體之指標，則使用「->」運算子。

就物件導向的角度而言，資料成員(data members)為某物件的屬性(attributes)或構成元件，函式成員(function members)為某物件之能力或行為(behaviors)。

格式:

```
//結構體定義
struct  結構體名稱
{
    資料成員;
    函式成員;
};

//或

struct  結構體名稱
{
    資料成員;
    函式成員宣告;
};

結構體名稱::函式成員定義{ }
```

```
//呼叫結構體之函式成員
結構體變數.函式成員(參數列);
//或
結構體指標->函式成員(參數列);
```

例:

```
//結構體定義
struct User
{
    string name;
    int age;
    int tel;

    void print( )
    {
        cout<<name<<" "<<age<<" "<<tel<<endl;
    }
};

//或

struct User
{
    string name;
    int age;
    int tel;

    void print( );
};

void User::print( )
{
    cout<<name<<" "<<age<<" "<<tel<<endl;
}
```

```
//呼叫結構體函式成員
User userObject;
User* userPtr = & userObject;
…….
userObject.print();
userPtr->print();
```

程式範例: cpp_ex44.cpp , cpp_ex45.cpp , cpp_ex46.cpp

注意要點:

1. 函式成員可以有參數列，輸入參數可用來改變資料成員的值。
2. 寫在結構體定義外之函式成員定義，定義之前，必須加上結構體名稱與範圍運算子(::)。此是因為函式成員的宣告是再結構體的定義內，故需將其所屬範疇指定出來。
3. 就程式的語意，對結構體之資料成員，函式成員之定義內容為「第一人稱」，獨立函式之定義內容為「第三人稱」。

5.5 Using Structure Variables or Objects as Data Members

說明:

一結構體變數亦可作為另一結構體的資料成員，遞迴式的結構體宣告(本身之成員)亦可。

程式範例: oop_ex47.cpp , oop_ex48.cpp