國立屏東大學 資訊工程學系 物件導向程式設計

17. 類別與物件(Classes and Objects)

- 物件導向概念
- C++的物件導向設計
- Class定義與Object實體化
- Constructors
- Destructors

相信您　一定聽過物件導向(Object-Oriented)這個名詞，但具體的意義又是什麼呢？基本上，我們可以將物件導向視為一種新的思維、新的想法，應用在系統分析、設計上，就產生了OOA(Object-Oriented Analysis□物件導向分析)與OOD(Object-Oriented Design□物件導向設計)；應用在程式語言，則有了OOP(Object-Oriented Programming□物件導向程式設計)的誕生。目前C++□C#□Java□Object C等都是十分流行的物件導向程式語言。我們將先介紹物件導向的觀念與精神，再就C++語言本身的物件導向相關的程式語法做一說明。

17.1 物件導向概念

□Programming is problem solving.□

這句話講得好！一語道破程式設計的工作。寫程式就是為了要解決特定的問題，例如：

- 給定50個數字，其最大值為何？
- 計算物件導向程式設計課程的學期成績

這些都是可能的「問題」。要記住，問題不等於「需求」！程式設計師的工作之一，就是要分析問題並制定「需求」。需求可視為程式規格的描述，包含功能性與非功能性的需求。所謂的功能性需求概指程式應提供的功能，而非功能性的需求則是如完成期限、執行平台、使用的開發工具與程式語言等與問題求解非直接相關的需求。我們為前述的例子列舉一些功能需求如下：

- 給定50個數字，其最大值為何？
    1. 需求規格一
        - 取得使用者輸入的50個整數
        - 經計算後找出最大值
        - 輸出結果到螢幕上
    2. 需求規格二
        - 從input.txt檔案取得50個整數
        - 經計算後找出最大值
        - 輸出結果到output.txt檔案中
- 計算物件導向程式設計課程的學期成績
    - 從score.txt檔案中取得修課成績
    - 學期成績=平時成績佔百分之三十，期中考成績佔百分之三十，期末成績佔百分之四十
    - 將每位同學的姓名、學號與學期成績公告到網站
    - 網址為http://web.csie.npic.edu.tw/~junwu/oop/score/index.html

簡單來說，程式的需求規格即為□What to do?□的具體描述，上述的範例都還過於簡化，您可以再進一步提出更具體的需求規格。有了需求規格後，下一步就是構思□How to do?□□

傳統的程式設計概念，是將程式設計視為問題求解的過程：針對各種的應用問題，定義所需要的資料項目與處理問題的邏輯、流程與步驟、條件與限制，並撰寫程式碼以取得資料項目、進行資料處理，最後將問題的解答輸出。我們在進行設計程式以解決應用問題時，腦海中所思考的是：

> 要取得哪些資料？如何取得？資料的型態為何？要用什麼樣的資料結構來存放資料？資料的範圍為何？
> 取得資料後該進行哪些資料處理？要使用什麼方法來處理？有沒有特定的處理流程、條件、步驟或限制？
> 資料處理完成後，又要輸出哪些資訊給使用者？格式為何？要求為何？

我們可以說這是一種「以資料為核心」的思維方式。

至於現代的物件導向程式設計則是以「物件」為核心，將問題抽象化描述為一個由物件所構成的環境，透過物件的行為、屬性的改變、物件彼此間的互動、訊息的傳遞，來完成問題求解。我們所思考的往往是以下問題：

> 在應用問題所存在的真實或虛擬世界中，有哪些種類的物件存在？
> 這些物件有哪些屬性？有哪些狀態？有哪些行為？
> 物件與物件間有沒有關係？有沒有互動？問題的需求牽涉到那些物件？

這種以「物件為核心」的思維方式，就是物件導向程式設計的概念，例如:

- 給定50個數字，其最大值為何？
  1. 此問題之求解存在哪些類別的物件?
     - 50個數字(Number)類別的物件
     - 一個可存放50個數字類別物件的容器，此容器是稱為Box類別的物件
  2. 這些物件有哪些屬性？狀態？行為?
     - Number類別的物件 - 設定數值
     - Box類別的物件 - 加入Number類別的物件, 找出具備最大值的Number類別的物件
  3. 物件間有沒有關係？有沒有互動？問題的需求牽涉到那些物件？
     - 一個Box類別的物件可以含有多個Number類別的物件

- 計算物件導向程式設計課程的學期成績
  1. 此問題之求解存在哪些類別的物件?
     - 學生類別的物件
     - 成績單類別的物件
     - Web公告類別的物件
  2. 這些物件有哪些屬性？狀態？行為?
     - 學生類別的物件：設定個人資料(含姓名與學號)、設定平時成績、設定期中考成績、設定期末考成績、計算學期成績
     - 成績單類別：新增修課同學、輸出成績資料(文字格式)
     - Web公告類別：取得成績資料並產生網頁、上傳到Web Server
  3. 物件間有沒有關係？有沒有互動？問題的需求牽涉到那些物件？
     - 一個成績單類別的物件與多個學生類別的物件有關係

17.1.1 Objects

在物件導向的世界，萬事萬物都是「物件(objects)□□不論是真實或虛擬世界中有形或無形的事物都被視為物件：

> 對於供應鍊管理系統而言：特定的企業、產品、客戶、供應商、物流業者等是物件
> 對於生產管理系統而言：特定的原料、生產線、作業員、品管員、生產計畫等是物件

> 對於銀行存放款系統而言：特定的銀行、帳戶、貸款、信用卡等是物件
> 對於學校的資訊系統而言：特定的學校、系所、班級、課程、學生等是物件
> 對於線上遊戲的軟體而言：特定的女巫、怪獸、寶物、地圖、商店、玩家等是物件
> 對於視窗系統系統而言：特定的程式視窗、對話盒、功能表、按鈕等是物件

與問題相關或有助於解決問題的物件加入到所建構的物件導向世界中，但和解決問題無關的物件屬性和物件行為會被省略。

> 例如學生成績管理系統當然會把學生視為物件，其屬性包含學生姓名、學號、成績等(但不包含學生的身高、體重等屬性)，其行為包含課程選修、加選、退選，成績登錄與查詢等(但不含社團活動、設備借用等行為)。
>
> 例如人事管理系統會把員工視為物件，其屬性包含了姓名、部門、職級、薪資等資料(但不包含員工的血型、星座等屬性)，其行為則包含了加班申請、出差申請或上班簽到等(但不含吃飯、睡覺等行為)。

物件所應該擁有的屬性與行為完全是依據系統的要求而決定的，在不同的軟體系統中也許會有同樣的物件，但其屬性和行為可能完全不同。

Programming is problem solving!

物件導向程式設計(object-oriented programming，OOP)將所要解決的問題，描繪成一個物件導向世界，凡是和問題相關或有助於解決問題的物件都存在於這個世界中。問題的處理邏輯、流程與步驟、條件與限制等需求，將會對映到物件的行為與屬性，透過　物件彼此間的互動與訊息的傳遞，問題就可以被解決。

17.1.2 Classes

物件導向程式設計就是要建構一個虛擬的物件導向世界，在這個世界中存在著一些物件，透過這些物件的行為、屬性的改變、彼此的互動與訊息的傳遞，就可以完成特定的系統目的。問題是，在建構這個世界之前，您有沒有先定義這些物件的行為與屬性呢？

> 類別(class)就是物件的抽象定義，包含了屬性與行為的定義。
>
> 有了類別的定義後，我們才能在虛擬的物件導向世界中產生物件。

要在物件導向的程式中使用物件，您必須先行定義其所屬的類別，才能據以產生物件供程式使用。定義類別就是所謂的抽象化(abstraction)，將真實或虛擬世界中的人、事、時、地、物等抽象對映為程式語言中的類別。至於產生類別所屬之物件的動作則稱為實體化(instantiate)，所產生出的物件被稱為「實體(instance)」

至於C++語言如何定義類別與產生物件，後續會再加以說明。

17.2 類別定義與物件實體化

在討論類別的定義前，先讓我們看看下面這個使用結構體的例子：

```
#include <iostream>
using namespace std;
```

```
struct Person
{
  string firstname;
  string lastname;
};


void showInfo(Person p)
{
    cout << "Name: " << p.firstname << " " << p.lastname << endl;
}


int main()
{
  Person amy;

  amy.firstname="Amy";
  amy.lastname="Tang";
  showInfo(amy);
  return 0;
}
```

在這個例子中，我們定義了一個名為Person的結構體，定設計一個函式用以顯示其資訊。同樣的應用，若改以類別來定義，則可以同時將類別的資料與其處理方法定義在一起。先讓我們看看類別定義的語法：類別的定義必須包含類別名稱、屬性與行為，使用C++語言的術語，屬性被稱為資料成員(data member)，行為被稱為成員函式(member function)。以下是C++語言類別定義的語法：

```
class className //類別名稱
{
public: // access modifier用以表示此類別的物件內的資料成員與成員函式皆可供外界使用
   // data member declarations
   [DataType variableName[=value]?[,variableName[=value]?]*;]*

   // member function declarations and implementations
   [returnType methodName(parameters)
   {
      // method implementations
      statements
   }]*
};
```

我們以前述的例子，改以類別定義如下：

```
#include <iostream>
using namespace std;
```

```
class Person
{
public:
  string firstname;
  string lastname;

  void showInfo()
  {
    cout << "Name: " << firstname << " " << lastname << endl;
  }

};

int main()
{
  Person amy;

  amy.firstname="Amy";
  amy.lastname="Tang";
  amy.showInfo();
  return 0;
}
```

從上面這個例子可以發現，類別就好像是結構體一樣，只是它還多了函式的定義。回想在結構體定義時，我們可以在定義的同時，就直接宣告其結構體變數，例如：

```
struct Person
{
  string firstname;
  string lastname;
} amy={"Amy", "Wang"}, tony={.lastname="Wu"};
```

我們也可以在定義類別時，宣告其物件變數：

```
class Person
{
public:
  string firstname;
  string lastname;

  void showInfo()
  {
    cout << "Name: " << firstname << " " << lastname << endl;
  }

} tony, amy={"Amy","Chang"}, jack {.lastname="Wu"};
```

當類別的定義完成後，我們可以將「類別名稱」視為資料型態，來宣告其變數，例如：

```
Person amy;
Person amy={"Amy","Chang"};
Person amy {"Amy","Chang"};
Person amy {.lastname="Chang", .firstname="Amy" };
```

從上述的例子中，可以發現也可以使用List initialization設定物件的初始值。

如同一般的變數宣告一樣，編譯器會自動幫我們在記憶體內配置其所需的空間。對於某個類別的變數，其在記憶體中所配置到的空間，即稱為該類別的一個「實體(instance)」；而其變數名稱，一般稱為「物件變數」或者直接稱為是該類別的「物件(object)□□

在實務上，為了記憶體管理的需求，我們通常會動態地配置物件在記憶體中的空間，我們將其稱為「物件的實體化(Object Instantiation)」。下面則是動態產生物件的實體並以指標進行操作的例子：

```
#include <iostream>
using namespace std;

class Person
{
public:
  string firstname;
  string lastname;

  void showInfo()
  {
    cout << "Name: " << firstname << " " << lastname << endl;
  }

};

int main()
{
  Person *amy = new Person();

  amy->firstname="Amy";
  amy->lastname="Tang";
  amy->showInfo();
  (*amy).showInfo();
  return 0;
}
```

當然，我們也可以動態建立物件的陣列：

```

```

```
#include <iostream>
using namespace std;


class Person
{
public:
  string firstname;
  string lastname;

  void showInfo()
  {
    cout << "Name: " << firstname << " " << lastname << endl;
  }

};

int main()
{
  Person *people = new Person [5];
  Person *someone;

  people[0].firstname="Amy";
  people[0].lastname="Chang";
  people[0].showInfo();
  someone=people;
  someone++;
  someone->firstname="Jacky";
  someone->lastname="Chen";
  (*someone).showInfo();
  delete [] people;
  return 0;
}
```

17.3 類別定義與實作的程式架構

通常，我們會將類別的定義與實作分開放在不同的檔案：

類別定義

```
#ifndef _PERSON_
#define _PERSON_

class Person
{
```

```
public:
  string firstname;
  string lastname;

  void showInfo();
};
#endif
```

類別實作

```
#include <iostream>
#include "person.h"
using namespace std;

void Person::showInfo()
{
  cout << "Name: " << firstname << " " << lastname << endl;
}
```

實作定義在Person類別中的成員函式時，要在函式名稱前加上Person::表明其所屬的類別。

在含有main()函式的程式中使用類別

```
#include <iostream>
#include "person.h"
using namespace std;

int main()
{
  Person *amy = new Person;
  amy->firstname="Amy";
  amy->lastname="Chang";
  amy->showInfo();
}
```

使用Makefile

通常，將類別的定義與實作分開後，最麻煩的就是編譯時的步驟變多了，例如：

```
g++ -c person.cpp
g++ main.cpp person.o -o main
```

這時只要搭配Makefile就可以輕鬆地完成編譯的動作：

```
all: person.o
	g++ main.cpp person.o -o main

person.o: person.cpp
	g++ -c person.cpp

clean:
	rm -f *.o main *.*~ *~
```

本系系友Kito寫了一個不錯的入門網頁，大家可以去參考參考。

17.3.1 建立inline的member function

在C/C++語言中，我們可將函式以inline方式宣告，來增進效能。當我們呼叫一個inline function時，編譯器會幫助我們將inline function內的程式碼「複製」一份到其所被呼叫之處，並在該處執行程式碼，省略了函式「呼叫與傳回」的動作。因此，嚴格來說，inlinefunction並不是function，而是類似#define一樣，是由編譯器進行前置的程式碼替代動作。

在C++中，我們也可以將類別的member function以inline方式宣告，並有以下兩種方式：

1. 宣告時不須註明，而在在實作時前置inline保留字即可，請參考下列的範例：

```
class Person
{
public:
  string firstname;
  string lastname;

  void showInfo();
};


inline void Person::showInfo()
{
  cout << "Name: " << firstname << " " << lastname << endl;
}
```

2. 直接宣告並實作於類別定義內，但在宣告時同樣不須註明。所有實作在類別定義內的member function都會自動被轉換成inline function，請參考下列的例子：

```
class Person
```

```
{
public:
  string firstname;
  string lastname;

  void showInfo()
  {
     cout << "Name: " << firstname << " " << lastname << endl;
  }
};
```

要注意的是inline function並不是絕對可以使用，一切必須視所使用的編譯器而定，例如：

- 一些編譯器並不支援在inline function中使用迴圈switch或是goto等敘述；
- 通常，你不應該將inline function設計為遞迴型式；
- 在inline function中並不支援宣告static的變數。

17.4 建構函式/建構子

有沒有注意到這行Person *amy = new Person();或是Person *amy = new Person;其中的new Person()即為用來產生一個Person類別的實體。其實還不只是這樣，它還會幫你呼叫一個名為Person()的成員函式。

> 可是我沒有定義這個成員啊？
>
> > C++的編譯器預設會幫你在類別定義內產生一個與類別同名的成員函式，我們稱之為「建構函式(constructor)」，或稱為「建構子」
> >
> > 例如Person類別會自動包含以下的程式碼

```
Person() { }
```

> > 它是一個特殊的成員函式，沒有傳回型態的宣告，且其內容為空白
> > 在使用new Person();時(或者是new Person;)這個建構函式會被自動呼叫
>
> 內容空白，那不就沒用處？
>
> > 看你想要做什麼？你可以自己定義啊！例如：

```
Person()
{
   firstname="undefined";
   lastname ="undefined";
}
```

> > 你還可以提供不同版本的constructor例如:

```
Person(String f, String l)
{
   firstname=f;
   lastname =l;
}
```

> 這樣一來，你可以在實體化的同時順便把firstname與lastname設定好。例如以下的宣告:

```
Person *amy = new Person();
```

> 不過要特別注意，如果定義了自己的constructor那麼預設的constructor就不會再產生，所以下面的程式碼是錯誤的：

```
#ifndef _PERSON_
#define _PERSON_

class Person
{
public:
  string firstname;
  string lastname;

  Person(string, string);
  void showInfo();
};
#endif
```

```
#include <iostream>
#include "person.h"
using namespace std;

Person::Person(string fn, string ln)
{
  firstname=fn;
  lastname=ln;
}

void Person::showInfo()
{
  cout << "Name: " << firstname << " " << lastname << endl;
}
```

```
#include <iostream>
#include "person.h"
using namespace std;

int main()
{
  Person *amy = new Person("Amy","Chang");
  amy->showInfo();
}
```

> 瞭解了！所以如果有自定constructor就要小心實體化時呼叫的版本是否正確！
>
> > 您也可多提供一個預設的版本！例如：

```
Person::Person()
{
   firstname="unknown";
   lastname="unkonwn";
}
```

17.5 解構函式/解構子

除了建構函式之外，C++也會在我們進行delete以回收不再需要的物件實體時，呼叫一個特別的成員函式，稱為「解構函式(destructor)」或「解構子」。如果你並沒有提供解構函式，那麼編譯器會幫我們產生例如下面的函式：

```
Person::~Person() {}
```

當然，你也可以實作自己的解構函式：

```
Person::~Person()
{
   cout << "Bye!" << endl;
}
```

如果你在物件中有動態配置記憶體，那麼解構函式就是個適合去回收它們的地方。例如：

```
#ifndef _PERSON_
#define _PERSON_

class Person
```

```
{
public:
  char *firstname;
  char *lastname;

  Person(const char *, const char *);
  ~Person();
  void showInfo();
};
#endif
```

```
#include <iostream>
#include <cstring>
#include "person.h"
using namespace std;

Person::Person(const char *fn, const char *ln)
{
  if(firstname==NULL)
    firstname = new char[strlen(fn)+1];
  if(lastname==NULL)
    lastname = new char[strlen(ln)+1];

  strcpy(firstname,fn);
  strcpy(lastname,ln);
}

Person::~Person()
{
  cout << "deleting string ... ";
  delete firstname;
  delete lastname;
  cout << "done.";
}

void Person::showInfo()
{
  cout << "Name: " << firstname << " " << lastname << endl;
}
```

```
#include <iostream>
#include "person.h"
using namespace std;
```

```
int main()
{
  Person *amy = new Person("Amy", "Chang");
  amy->showInfo();
  delete amy;
}
```