

19. 繼承、覆寫與重載

19.1 繼承性

Inheritance(繼承性)是指讓某一類別的物件繼承來自其它類別的屬性與行為。C++支援多重繼承，讓一個類別可以繼承多個類別。

□A is a kind of B.□

繼承性其實就是所謂的ISA關係 - IS-A relationship□是指兩個類別A與B間存在著□A類別就是B類別的一種特殊化。所謂的「特殊化(specialization)□是指A類別其實就是B類別，但A類別比B類別特殊些。我們將A類別稱為衍生類別(derived class)或子類別(child class)□B類別則稱為基礎類別(base class)或父類別(parent class)□

回顧我們在前一章開始介紹的Person類別，每個屬於這個類別的物件都會具有first name, last name等資料成員，以及showInfo()□set_firstname()□get_firstname()□set_lastname()□get_lastname()等成員函式：

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

#ifndef _PERSON_
#define _PERSON_

class Person
{
private:
    string firstname;
    string lastname;

public:
    Person();
    Person(string, string);
    void showInfo();

    void set_firstname(string fn);
    string get_firstname();

    void set_lastname(string ln);
    string get_lastname();
};
#endif
```

```
#include "person.h"

Person::Person()
{
}

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

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

void Person::set_firstname(string fn)
{
    firstname=fn;
}

void Person::set_lastname(string ln)
{
    lastname=ln;
}

string Person::get_firstname()
{
    return firstname;
}

string Person::get_lastname()
{
    return lastname;
}
```

Person類別的衍生類別 - Student類別設計

如果我們要設計一個學生成績管理系統(Student Score Management System, STUScoreMan)在我們所要建構的物件導向世界中，必然會存在有「學生」這種類別的物件。那要如何開始設計學生這個類別呢？先考慮學生只需具備firstname、lastname與學號三個資料成員，那麼我們可以撰寫以下的程式碼：

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

#ifdef _STUDENT_
```

```
#define _STUDENT_  
  
class Student  
{  
private:  
    string firstname;  
    string lastname;  
    string ID;  
};  
#endif
```

就如同前一章所介紹的，我們還應該為這個類別加上建構子、顯示學生資料的method、設定存取權限、設計setters與getters...

```
#include <iostream>  
using namespace std;  
  
#ifndef _STUDENT_  
#define _STUDENT_  
  
class Student  
{  
private:  
    string firstname;  
    string lastname;  
    string ID;  
  
public:  
    Student();  
    Student(string, string);  
    void showInfo();  
  
    void set_firstname(string fn);  
    string get_firstname();  
  
    void set_lastname(string ln);  
    string get_lastname();  
    void set_ID(string id);  
    string get_ID();  
};  
#endif
```

我們發現這個新設計的Student類別，與Person類別很相似....

「Student也是一種Person，只是比較特別！比起Person，Student還多了ID(目前為止)。」

以這樣的角度看問題時，我們可以說「**Student is a kind of Person!**」。我們可以用以下的程式碼，告訴電腦這件事：

```
#include "person.h"

class Student : public Person
{
};
```

□: public Person□是用以表示Student類別是「**衍生自(derived from)**□Person類別，用物件導向的術語來說：**Student類別繼承了Person類別**。在這個ISA的繼承關係中，我們將Person類別稱為「**基礎類別(base class)或父類別(parent class)**□□Student類別則稱為「**衍生類別(derived class)或子類別(child class)**□□

其中的□public□修飾字是用以說明此繼承為public繼承，稱為「**public derivation**」，所有在父類別中的public成員都會變成是在子類別中的public成員。但在父類別中的private成員，則僅能從繼承自父類別中的public(或protected)成員函式來存取。除此之外，除了預設的建構函式外，其餘的建構函式並不會被繼承。

<note> ++++ private與protected derivation | 除了public derivation之外，還有private derivation與protected derivation□可用以進行不同開放程度的繼承。

```
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};

class B : public A
{
    // x is public
    // y is protected
    // z is not accessible from B
};

class C : protected A
{
    // x is protected
    // y is protected
    // z is not accessible from C
};

class D : private A
{
    // x is private
    // y is private
};
```

```
// z is not accessible from D  
};
```

++++ </note>

一旦你完成這樣的設計，儘管現在在Student類別中，一行程式碼都還沒寫，但Person類別該有的Student類別也都會有，包含Person類別的屬性與行為。因此，我們可以在物件導向的世界中，把Student類別的物件視為是Person類別的物件，並且使用它所公開(public)的屬性與行為。請參考下面的程式碼：

```
#include <iostream>  
using namespace std;  
  
#include "student.h"  
  
int main()  
{  
    Student *amy = new Student;  
  
    amy->set_firstname("Amy");  
    amy->set_lastname("Chang");  
    amy->showInfo();  
  
    return 0;  
}
```

沒有意外地，其執行結果如下：

```
junwu@ws2 oop % ./a.out  
Name: Amy Chang  
junwu@ws2 oop %
```

但這不是我們要的結果，雖然Student已經是a kind of Person，但Student不夠特殊，它跟Person根本是一樣的。讓我們將新的屬性與行為加到Student類別中，請參考下面的程式碼：

```
#ifndef _STUDENT_  
#define _STUDENT_  
#include "person.h"  
  
class Student : public Person  
{  
private:  
    string ID;
```

```
public:
    void set_ID(string id);
    string get_ID();
};
#endif
```

```
#include "student.h"

void Student::set_ID(string id)
{
    ID=id;
}

string Student::get_ID()
{
    return ID;
}
```

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

#include "student.h"

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

    amy->set_firstname("Amy");
    amy->set_lastname("Chang");
    amy->set_ID("s111418099");
    amy->showInfo();

    return 0;
}
```

<note> 在Person類別中，`firstname`與`lastname`是定義為`private`，所以連其子類別都不能使用，必須透過`public`的`setters`與`getters`才能存取。 </note>

讓我們為這個小節做個總結：

□ Student IS A (kind of) Person □

在C++語言裡，子類別(subclass)可以繼承父類別(super class)所有的屬性與行為，除了那些被定義為私有的(private)。

19.2 衍生類別的建構與解構函式

19.2.1 預設的建構函式

在本章的範例中，我們使用public derivation來讓Student類別繼承了Person類別。在Person類別中，提供了Person(string, string)型式的建構函式，可以將firstname與lastname的初始值加以設定。現在，讓我們試著使用下列程式碼，來產生Student類別的物件並加以初始化其值：

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

#include "student.h"

int main()
{
    Student *stu = new Student("Yo-Yo", "Ma");
    return 0;
}
```

讓將之加以編譯，會得到以下的錯誤：

```
junwu@ws2 oop % c++ main.cpp
main.cpp: In function 'int main()':
main.cpp:14:35: error: no matching function for call to
'Student::Student(const char [4], const char [6])'
main.cpp:8:44: note: candidates are:
In file included from main.cpp:4:0:
student.h:5:7: note: Student::Student()
student.h:5:7: note:   candidate expects 0 arguments, 2 provided
student.h:5:7: note: Student::Student(const Student&)
student.h:5:7: note:   candidate expects 1 argument, 2 provided
junwu@ws2 oop %
```

其中error: no matching function for call to 'Student::Student(const char [5], const char [2])'含有兩點資訊：

1. 不存在兩個參數的建構函式
2. 字串常值的預設型態為const char[]

另外candidates are則列示了，可能的建構函式包含哪些，其中：

```
student.h:5:7: note: Student::Student()
student.h:5:7: note:   candidate expects 0 arguments, 2 provided
```

表示存在一個Student::Student()的建構函式，當然它是從Person類別繼承下來的，試著將程式修改如下：

```
Person::Person()  
{  
    cout << "A Person is created." << endl;  
}
```

並且使用無參數的Student()建構一個物件看看，你應該會看到以下的輸出。

```
A Person is created.
```

其實，在前面的討論中，我們已經說明過使用public derivation時，除此之外，除了預設的建構函式外，其餘的建構函式並不會被繼承(表示預設的無參數建構函式會被繼承)。但下面卻又告訴我們還有一個預設的建構函式存在：

```
student.h:5:7: note: Student::Student(const Student&)  
student.h:5:7: note:   candidate expects 1 argument, 2 provided
```

這是一個單一參數的建構函式Student::Student(const Student &)讓我們試試以下的程式碼：

```
#include <iostream>  
using namespace std;  
#include "student.h"  
  
int main()  
{  
    Student *amy = new Student();  
  
    amy->set_firstname("Amy");  
    amy->set_lastname("Chang");  
    amy->set_ID("s111418099");  
  
    Student *tony = new Student(*amy);  
  
    tony->set_firstname("Tony");  
  
    amy->showInfo();  
    tony->showInfo();  
  
    return 0;  
}
```

其執行結果為：

```
junwu@ws2 oop % ./a.out  
Name: Amy Chang  
Name: Tony Chang  
junwu@ws2 oop %
```

其實，這個並不是預設的建構函式，而是預設的List Initializer也就是說，這是預設的初始值給定。這會

以所傳入的物件參考的值，來設定新物件的值。

[請參考person3](#)

在解構函式部份，則與建構函式類似。衍生類別也會繼承基礎類別的解構函式：

[請參考person4](#)

19.2.2 設計新的建構與解構函式

在預設的情況下，一個子類別的物件建立時，必先呼叫父類別的建構函數，再呼叫本身的建構函數；且在解構時，必先呼叫本身的解構函數，再呼叫父類別的解構函數。例如：

```
Person::~~Person()  
{  
    cout << "A Person is removed." << endl;  
}  
  
Person::Person()  
{  
    cout << "A Person is created." << endl;  
}  
  
Student::Student()  
{  
    cout << "A Student is created." << endl;  
}  
  
Student::~~Student()  
{  
    cout << "A Student is removed." << endl;  
}
```

若執行下列程式：

```
int main()  
{  
    Student *amy = new Student();  
  
    amy->set_firstname("Amy");  
    amy->set_lastname("Chang");  
    amy->set_ID("s111418099");  
  
    Student *tony = new Student(*amy);  
  
    tony->set_firstname("Tony");  
}
```

```
amy->showInfo();
tony->showInfo();

delete amy;
delete tony;

return 0;
}
```

其執行結果為：

```
A Person is created.
A Student is created.
Name: Amy Chang
Name: Tony Chang
A Student is removed.
A Person is removed.
A Student is removed.
A Person is removed.
```

請參考[person5](#)

除了預設型式的建構函式外，我們也可以設計新的建構函式。

```
Student::Student(string fn, string ln, string id)
{
    firstname=fn;
    lastname=ln;
    ID=id;
}
```

但編譯時會得到以下的錯誤：

```
In file included from student.h:3:0,
                  from student.cpp:1:
person.h: In constructor 'Student::Student(std::string, std::string,
std::string)':
person.h:10:10: error: 'std::string Person::firstname' is private
student.cpp:10:3: error: within this context
In file included from student.h:3:0,
                  from student.cpp:1:
person.h:11:10: error: 'std::string Person::lastname' is private
student.cpp:11:3: error: within this context
make: *** [student.o] Error 1
```

從上面的訊息中，你可以瞭解問題在哪嗎？讓我們改寫這個建構函式如下：

```
Student::Student(string fn, string ln, string id)
{
    set_firstname(fn);
    set_lastname(ln);
    ID=id;
}
```

若執行下列程式：

```
int main()
{
    Student *amy = new Student("Amy", "Chang", "s111418099");
    amy->showInfo();
    delete amy;

    return 0;
}
```

可以得到以下的結果：

```
A Person is created.
Name: Amy Chang
A Student is removed.
A Person is removed.
```

請參考[person6](#)

我們也可以在衍生類別的建構函式中呼叫其基礎類別的建構函式：

```
Student::Student(string fn, string ln, string id):Person(fn, ln)
{
    ID=id;
}
```

請參考[person7](#)

由於在衍生類別的建構過程中，已呼叫了一個基礎類別的建構函式(兩個字串參數版本)，因此將不會再呼叫預設版本。我們也可以將上述建構式修改如下：

```
Student::Student(string fn, string ln, string id):Person(fn, ln), ID(id)
{
}
```

19.3 覆寫(overriding)

所謂的覆寫(overriding)是指，衍生類別將所繼承下來的成員函式改寫，提供自己的實作。例如，在我們前述的例子中，`Student`類別繼承了來自`Person`類別的`showInfo()`函式，但其不符合需求，因此，我們將其改寫如下：

```
#ifndef _STUDENT_
#define _STUDENT_
#include "person.h"

class Student : public Person
{
private:
    string ID;

public:
    Student();
    Student(string fn, string ln, string id);
    ~Student();
    void set_lastname(string ln);
    void set_firstname(string fn);
    void set_ID(string id);
    string get_ID();
    void showInfo();
};
#endif
```

```
void Student::showInfo()
{
    cout << "Name: " << get_firstname() << " " << get_lastname() << endl;
    cout << "ID: " << ID << endl;
}

void Student::set_firstname(string fn)
{
    for(int i=0;i<fn.size();i++)
    {
        fn[i]=toupper(fn[i]);
    }
    Person::set_firstname(fn);
}

void Student::set_lastname(string ln)
{
    for(int i=0;i<ln.size();i++)
    {
        ln[i]=toupper(ln[i]);
    }
}
```

```
}  
    Person::set_lastname(ln);  
}
```

請參考[person8](#)

當我們改寫了成員函式後，還是可以用基礎類別的名稱加上「:」來呼叫原本的版本，例如
`Person::set_lastname(ln);`

19.4 重載

所謂的重載(overloading)(又被稱為多載或覆載)是指在類別中，擁有一個以上相同名稱的成員函式。由於名稱相同，在呼叫時會以其參數來決定執行的版本。前面所提到的，一個類別可以有多個不同版本的建構函式，就是多載的例子。請參考下面的例子：

請參考[person9](#)

```
#ifndef _STUDENT_  
#define _STUDENT_  
#include "person.h"  
  
class Student : public Person  
{  
private:  
    string ID;  
  
public:  
    Student();  
    Student(string fn, string ln, string id);  
    ~Student();  
  
    void set_ID(string id);  
  
    void set_name(string fn, string ln);  
    void set_name(string n);  
  
    string get_ID();  
    void showInfo();  
};  
#endif
```

我們打算提供新的設定學生名字的方法，採用`set_name()`來完成其名稱的設定，並且提供兩種方法，首先是：

```
void Student::set_name(string fn, string ln)
```

```
{
    set_firstname(fn);
    set_lastname(ln);
}

void Student::set_name(string n)
{
    unsigned pos=n.find(" ");
    set_firstname( n.substr(0,pos));
    set_lastname( n.substr( pos+1, n.size()-(get_firstname()).size()));
}
```

將一個衍生類別的參考或指標轉換為其基礎類別的參考或指標，就稱為「upcasting」(向上轉型)，同時只要是public derivation的類別都允許這樣做。

From:

<https://junwu.nptu.edu.tw/dokuwiki/> - Jun Wu的教學網頁

國立屏東大學資訊工程學系

CSIE, NPTU

Total: 118140

Permanent link:

<https://junwu.nptu.edu.tw/dokuwiki/doku.php?id=cpp:inheritance>

Last update: **2022/05/05 16:15**

