Lab of Object-Oriented Programming:

Inheritance

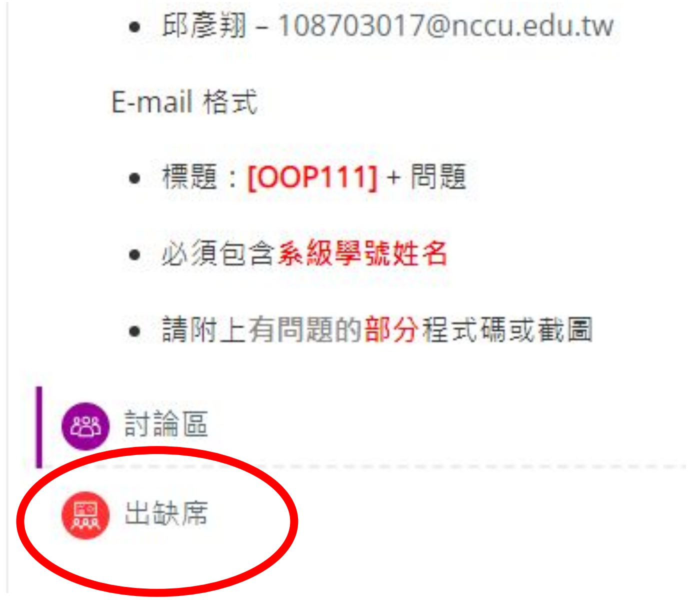
黃威、陳岳紘、邱彥翔

2022

使用 moodle 點名

請登入實習課的 moodle 課程

點擊出缺席並完成今日的點名



- 邱彥翔 – 108703017@nccu.edu.tw

E-mail 格式

- 標題：[OOP111] + 問題
- 必須包含系級學號姓名
- 請附上有問題的部分程式碼或截圖

Agenda

- Inheritance
- Assign5

Inheritance

```
class  <derived_class_name> : <access-specifier> <base_class_name>
{
        //body
};
```

How to Use Inheritance

- `class Orange : public Fruit {};      // 較常用`
- `class Orange : protected Fruit {};`
- `class Orange : private Fruit {};`

What is Inheritance？

- 將某個基礎類別的定義加以擴充，設計出一個新的衍生類別

- 被繼承的類別稱為基礎類別(Base Class)，繼承的類別稱為衍生類別(Derived Class)

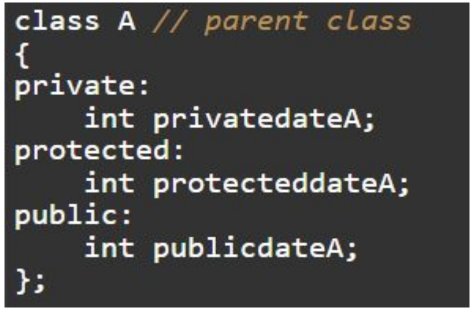
- 基礎類別又稱為父類別(Parent Class)，衍生類別又稱為子類別(Child Class)

- e.g.：所有的橘子都是水果(Is-a)，所以橘子類別(Derived Class)可以繼承水果類別(Base Class)

Why use Inheritance？

- 程式碼的 reuse
- 節省開發時間
- 使用已測試完成的程式碼，可以降低程式的錯誤
- 在開發大型專案時，利用測試完成的程式碼，會比不斷重新設計新程式好

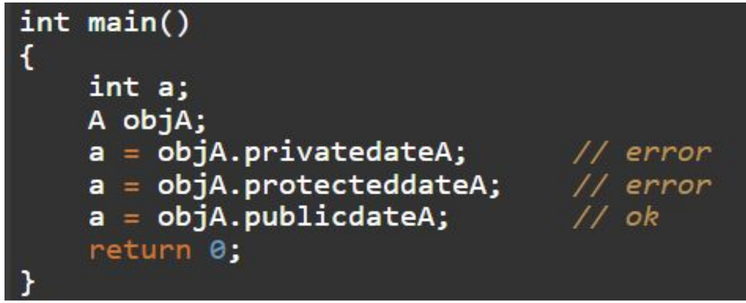
Encapsulation



```
class A // parent class
{
private:
    int privatedateA;
protected:
    int protecteddateA;
public:
    int publicdateA;
};
```

public, protected and private member

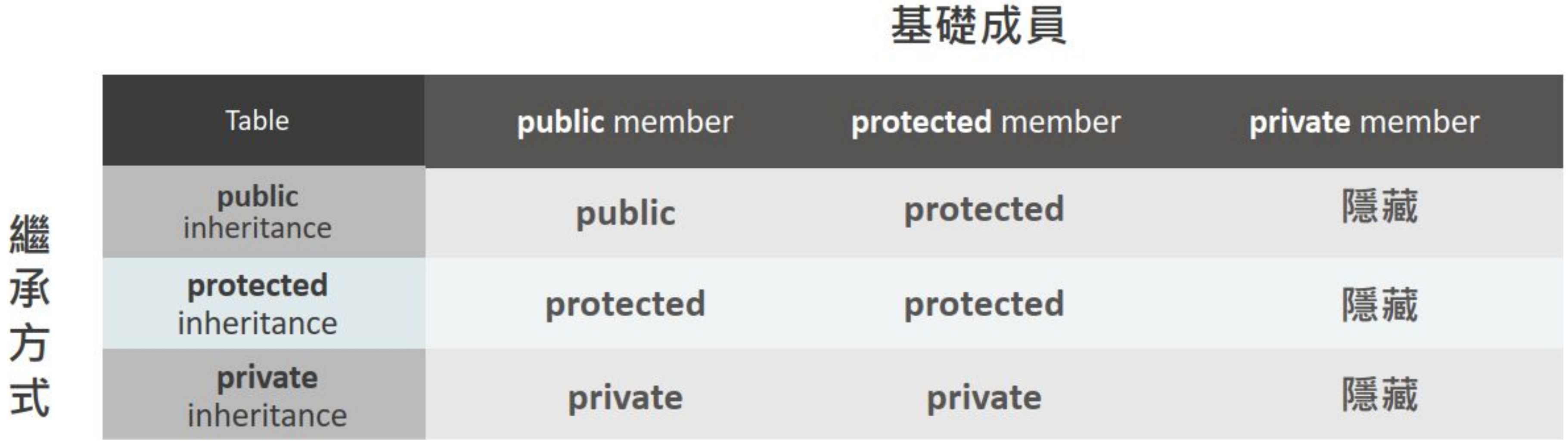
- public: 在子類、父類、子物件、父物件中都可以使用
- protected: 在子類、父類中可以使用
- private: 僅在父類中可使用



```
int main()
{
    int a;
    A objA;
    a = objA.privatedateA;      // error
    a = objA.protecteddateA;    // error
    a = objA.publicdateA;       // ok
    return 0;
}
```

父類(自身), 子類(繼承之物)

Inheritance - Graph



		基礎成員		
	Table	**public** member	**protected** member	**private** member
繼承方式	**public** inheritance	public	protected	隱藏
	protected inheritance	protected	protected	隱藏
	private inheritance	private	private	隱藏

Inheritance - Graph

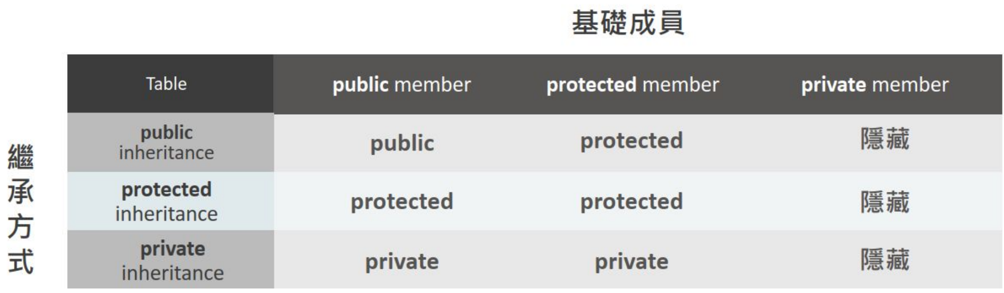
基礎成員 父類 member

繼承方式

Table	**public** member	**protected** member	**private** member
public inheritance	public	protected	隱藏
protected inheritance	protected	protected	隱藏
private inheritance	private	private	隱藏

子類 member

Inheritance - Graph



基礎成員

繼承方式

Table	public member	protected member	private member
public inheritance	public	protected	隱藏
protected inheritance	protected	protected	隱藏
private inheritance	private	private	隱藏

繼承圖怎麼解讀？

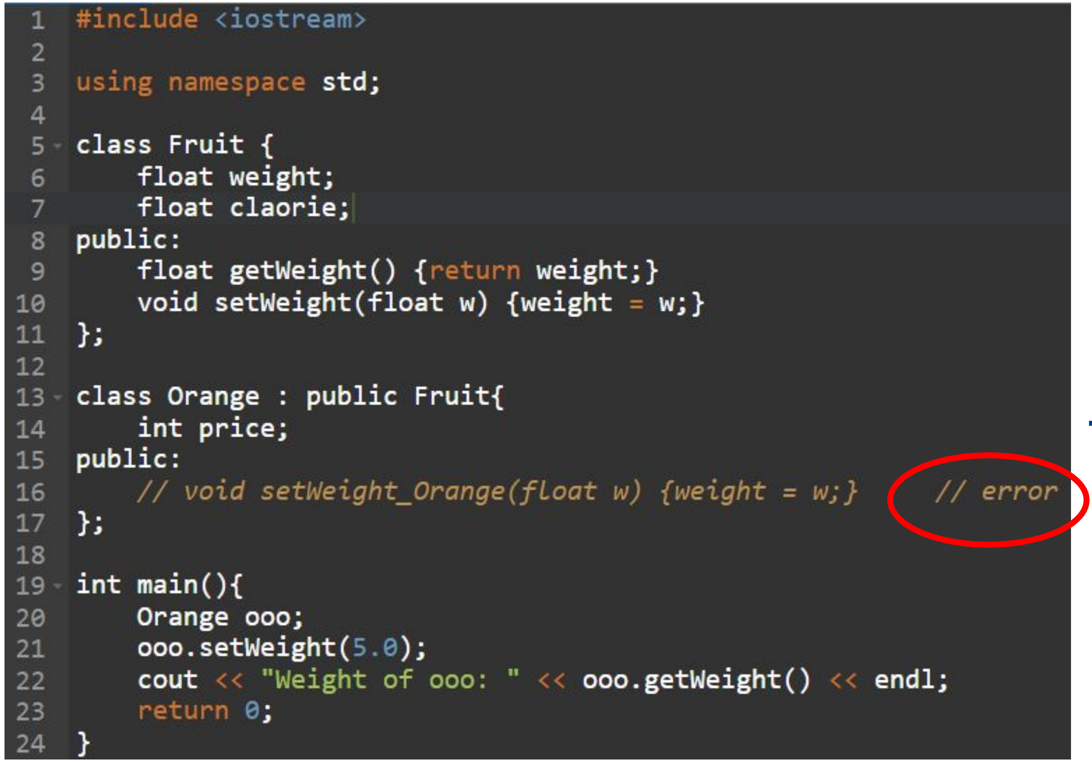
遵循兩大要點:

1. 父類的 private 成員能被子類繼承但無法進行存取

2. inheritance access specifier ∩ parent class member access specifier

交集即為取上限

Inheritance - private member



```
#include <iostream>

using namespace std;

class Fruit {
    float weight;
    float claorie;
public:
    float getWeight() {return weight;}
    void setWeight(float w) {weight = w;}
};

class Orange : public Fruit{
    int price;
public:
    // void setWeight_Orange(float w) {weight = w;}      // error
};

int main(){
    Orange ooo;
    ooo.setWeight(5.0);
    cout << "Weight of ooo: " << ooo.getWeight() << endl;
    return 0;
}
```

private member 只能被父類存取

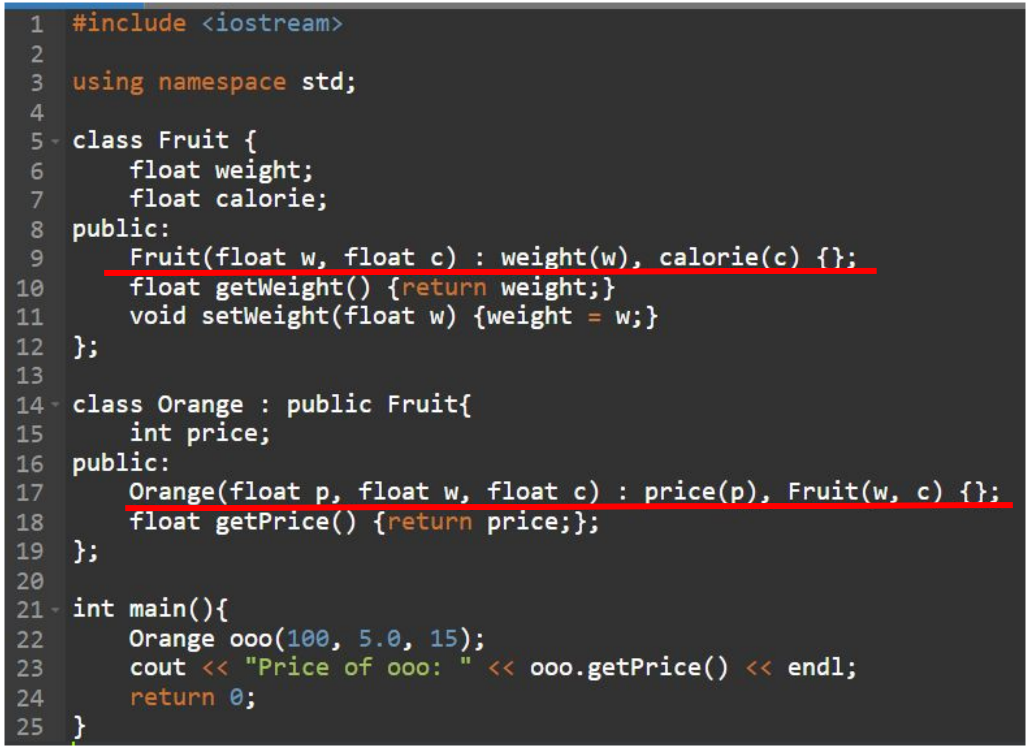
https://onlinegdb.com/4GTYdRiCP

Inheritance - Constructor

子類不會繼承父類的 Constructor
須自行實作子類的 Constructor

因為子類無法存取父類的 priavte member
意謂著其無法直接存取父類的 Constructor
若要設定父類的 private member, 使用 initialization list

Inheritance - Constructor

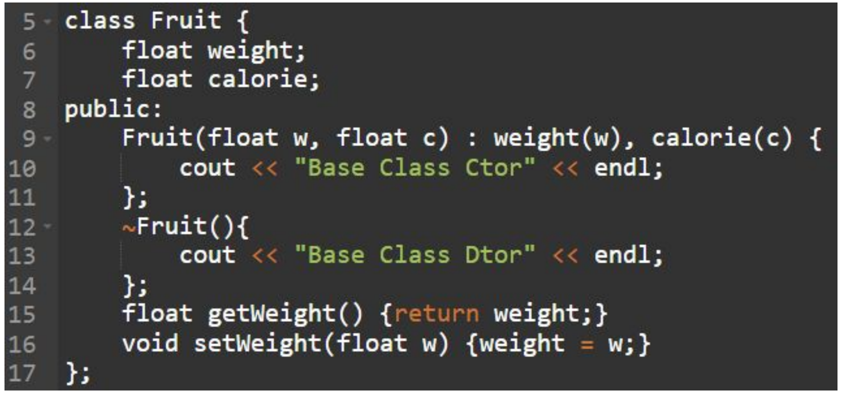


```
#include <iostream>

using namespace std;

class Fruit {
    float weight;
    float calorie;
public:
    Fruit(float w, float c) : weight(w), calorie(c) {};
    float getWeight() {return weight;}
    void setWeight(float w) {weight = w;}
};

class Orange : public Fruit{
    int price;
public:
    Orange(float p, float w, float c) : price(p), Fruit(w, c) {};
    float getPrice() {return price;};
};

int main(){
    Orange ooo(100, 5.0, 15);
    cout << "Price of ooo: " << ooo.getPrice() << endl;
    return 0;
}
```

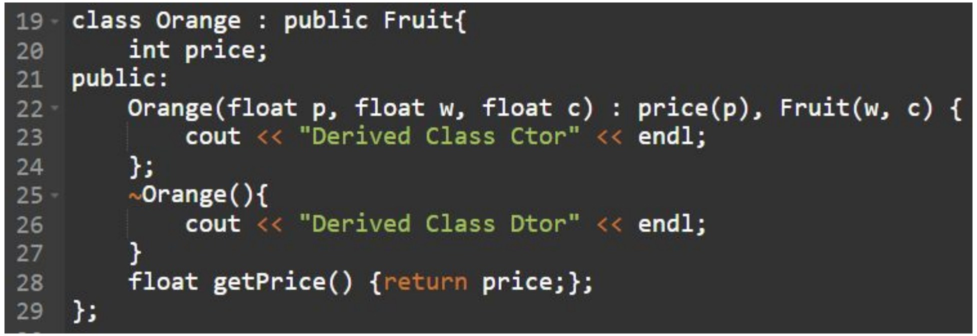
使用 initialization list 設定父類的 private 成員變數

https://onlinegdb.com/n6a54ie9y

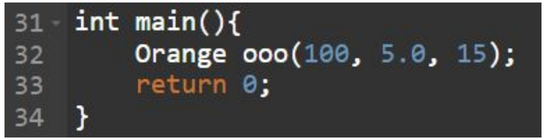
Inheritance - Constructor/Destructor Order



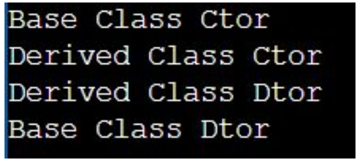
```cpp
class Fruit {
    float weight;
    float calorie;
public:
    Fruit(float w, float c) : weight(w), calorie(c) {
        cout << "Base Class Ctor" << endl;
    };
    ~Fruit(){
        cout << "Base Class Dtor" << endl;
    };
    float getWeight() {return weight;}
    void setWeight(float w) {weight = w;}
};
```



```cpp
class Orange : public Fruit{
    int price;
public:
    Orange(float p, float w, float c) : price(p), Fruit(w, c) {
        cout << "Derived Class Ctor" << endl;
    };
    ~Orange(){
        cout << "Derived Class Dtor" << endl;
    }
    float getPrice() {return price;};
};
```



```cpp
int main(){
    Orange ooo(100, 5.0, 15);
    return 0;
}
```



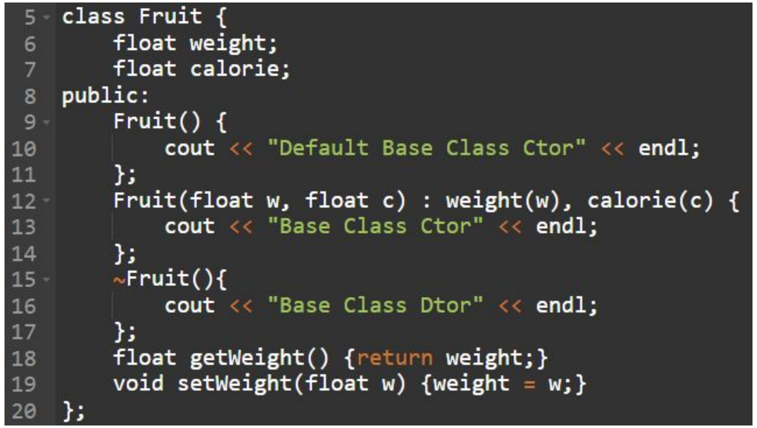
```
Base Class Ctor
Derived Class Ctor
Derived Class Dtor
Base Class Dtor
```

Constructor 建構順序:
先建構父類, 再建構子類

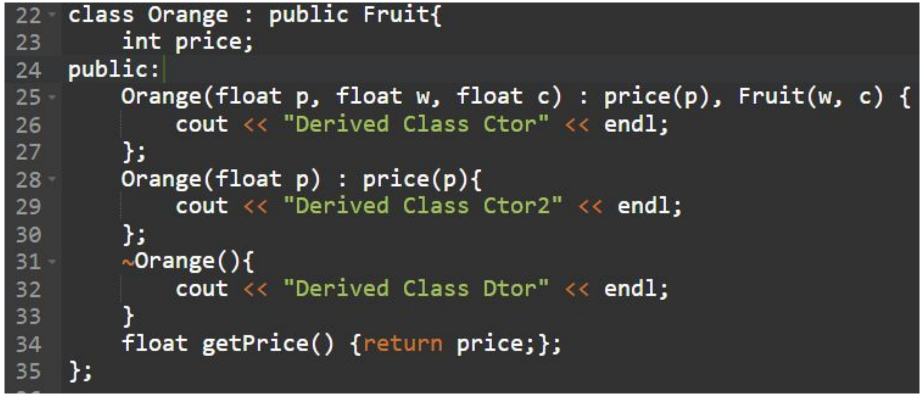
Destructor 解構順序
先解構子類, 再解構父類

https://onlinegdb.com/HAhdSixia

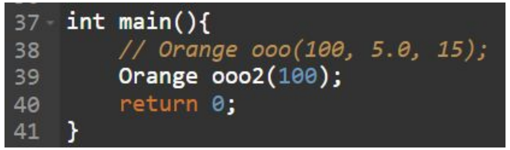
Inheritance - Default Constructor



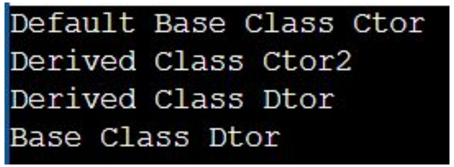
```cpp
class Fruit {
    float weight;
    float calorie;
public:
    Fruit() {
        cout << "Default Base Class Ctor" << endl;
    };
    Fruit(float w, float c) : weight(w), calorie(c) {
        cout << "Base Class Ctor" << endl;
    };
    ~Fruit(){
        cout << "Base Class Dtor" << endl;
    };
    float getWeight() {return weight;}
    void setWeight(float w) {weight = w;}
};
```



```cpp
class Orange : public Fruit{
    int price;
public:
    Orange(float p, float w, float c) : price(p), Fruit(w, c) {
        cout << "Derived Class Ctor" << endl;
    };
    Orange(float p) : price(p){
        cout << "Derived Class Ctor2" << endl;
    };
    ~Orange(){
        cout << "Derived Class Dtor" << endl;
    }
    float getPrice() {return price;};
};
```



```cpp
int main(){
    // Orange ooo(100, 5.0, 15);
    Orange ooo2(100);
    return 0;
}
```

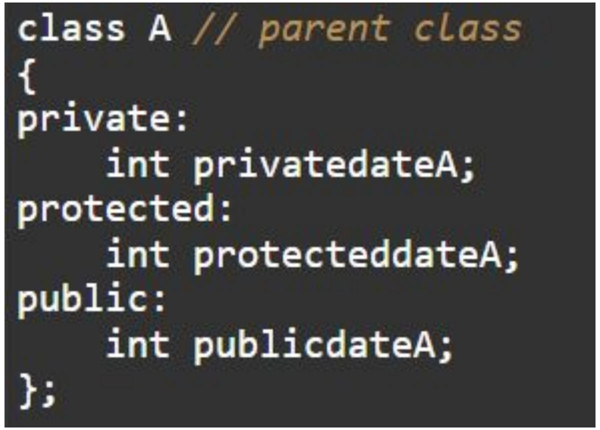


```
Default Base Class Ctor
Derived Class Ctor2
Derived Class Dtor
Base Class Dtor
```

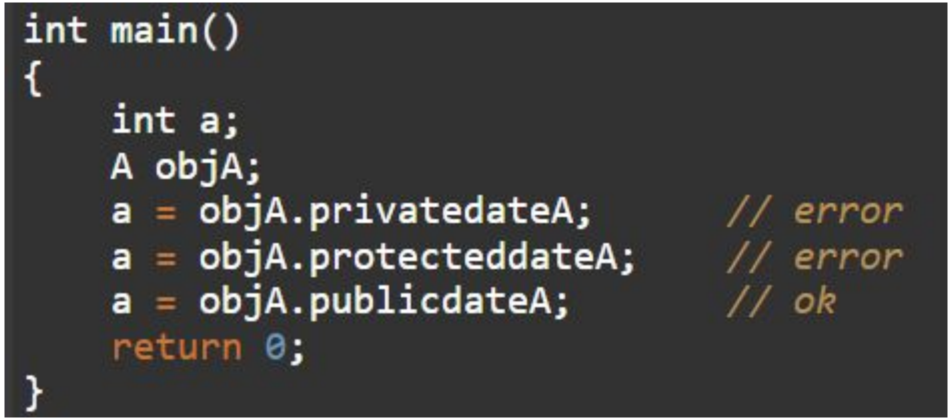
子類 Constructor 未指定父類 Construcor 時，compiler 會自動呼叫父類的預設 Constructor

https://onlinegdb.com/ducYHAVtp

Inheritance - Summary



```
class A // parent class
{
private:
    int privatedateA;
protected:
    int protecteddateA;
public:
    int publicdateA;
};
```



```
int main()
{
    int a;
    A objA;
    a = objA.privatedateA;      // error
    a = objA.protecteddateA;    // error
    a = objA.publicdateA;       // ok
    return 0;
}
```

Inheritance - Summary

class

Public

```
class B :public A          //基類A的衍生類B
 { public:
        void funct() {
            int b;
            b=privatedateA;      //error：基類中私有成員在衍生類中是不可見的
            b=protecteddateA;   //ok：基類的保護成員在衍生類中為保護成員
            b=publicdateA;       //ok：基類的公共成員在衍生類中為公共成員  }};
```

Protected

```
class C :protected A     //基類A的衍生類C
{ public:
        void funct() {
            int c;
            c=privatedateA;      //error：基類中私有成員在衍生類中是不可見的
            c=protecteddateA;   //ok：基類的保護成員在衍生類中為保護成員
            c=publicdateA;       //ok：基類的公共成員在衍生類中為保護成員  }};
```

Private

```
class D :private A         //基類A的衍生類D
{ public:
        void funct() {
            int d;
            d=privatedateA;      //error：基類中私有成員在衍生類中是不可見的
            d=protecteddateA;   //ok：基類的保護成員在衍生類中為私有成員
            d=publicdateA;       //ok：基類的公共成員在衍生類中為私有成員  }};
```

Inheritance - Summary

object

Public

```
int main() {
    int a;
    B objB;
    a = objB.privatedateA;      //error：基類中私有成員在衍生類中是不可見的，對外部對象不可見
    a = objB.protecteddateA;    //error：基類的保護成員在衍生類中為保護成員，對外部對象不可見
    a = objB.publicdateA;       //ok：基類的公共成員在衍生類中為公共成員，對外部對象可見
```

Protected

```
    C objC;
    a=objC.privatedateA;        //error：基類中私有成員在衍生類中是不可見的，對外部對象不可見
    a=objC.protecteddateA;      //error：基類的保護成員在衍生類中為保護成員，對外部對象不可見
    a=objC.publicdateA;         //error：基類的公共成員在衍生類中為保護成員，對外部對象不可見
```

Private

```
    D objD;
    a=objD.privatedateA;        //error：基類中私有成員在衍生類中是不可見的，對外部對象不可見
    a=objD.protecteddateA;      //error：基類的保護成員在衍生類中為私有成員，對外部對象不可見
    a=objD.publicdateA;         //error：基類的公共成員在衍生類中為私有成員，對外部對象不可見

    return 0;
}
```

Assign5

Description:

The rules and requirements in this assignment remain the same as in the third assignment. However, we will use class inheritance to relate SHPlayer and SHDealer. We mentioned that it makes more sense to inherit the SHDealer class from the SHPlayer class since they have a "Is-A" relation. The SHDealer class needs to know how to start a new game, how to print cards to the screen, how to compute the total for a hand of cards, etc, and these functions are already implemented by the SHPlayer class. Thus, a SHDealer is a SHPlayer.

Assign5

Description:

In this assignment, you are asked to change the containment relation (object within object) to an inheritance relation. The SHDealer class will simply inherit and use most data members and member functions in the SHPlayer class except for the start() function. The SHDealer class needs to behave differently since it needs to set up the cards for distribution.

Assign5 配分

未使用繼承實現，Assign5 的分數 / 2

項目	配分
有交（含屍體）	20
可以編譯	15
按 1 可以拿牌	10
按 2 放棄這一輪並顯示點數	10
按 3 可以重新開始	10
可以完成一輪完整的遊戲	20
按 4 可以離開並印出學號	10
印出玩家及莊家的名字	5

Assign5 Hint

未使用繼承實現，Assign5 的分數 / 2

SHDealer 繼承 SHPlayer
除了start()，start() 雖然兩個函式都有，但 內部實現方式完全不同

addCard()
openFirstCard()
showCard()

SHDealer 在使用以上三個成員函數時，要繼承 SHPlayer 的方法

Inheritance (Next Week)

https://onlinegdb.com/VJqjhu9QR

Any questions?