

- **Instructor**
 - Hsueh-Wen Tseng 曾學文, hwtseng@nchu.edu.tw
- **Textbook**
 - C++程式設計風格與藝術 (O'Reilly).
- **Requirements**
 - Assignment x ? 100%
- **TAs**

第一章 概觀C++

1-2 二種版本的C++

1-5 初步檢視類別

1-1 何謂物件導向程式設計

1-8 C++的關鍵字

1-2 二種版本的C++

//傳統的C++程式

```
#include <iostream.h>
```

```
int main() {
```

```
    //程式碼
```

```
    return 0 ;
```

```
}
```

//現代的C++程式

```
#include <iostream>
```

```
using namespace std ;
```

```
int main() {
```

```
    //程式碼
```

```
    return 0 ;
```

```
}
```

C++的新式標頭

```
// 傳統C++函數庫
#include <iostram.h>
// C函式庫
#include <math.h>
#include <string.h>
int main() {
    ....
    return 0 ;
}
```

```
#include <iostream>
#include <cmath>
#include <cstring>
using namespace std ;
int main() {
    ....
    return 0 ;
}
```

1-5 初步檢視類別

- **class:** 用來訂定物件的規格
- 物件 = 屬性(Attributes) + 動作(Actions)

I am a bird.
My name is Cathy.



屬性: name, feather, claw,

動作: fly, sleep, eat

C++的class

```
class bird {
```

```
private: // data member
```

```
    string name, feather ;
```

```
    int claws ;
```

```
public: // member functions
```

```
    void fly() {...}
```

```
    void sleep() {...}
```

```
    void eat() {...}
```

```
};
```

屬性

動作

```
//產生物件Cathy
```

```
void main() {
```

```
    bird Cathy ;
```

```
    Cathy.fly() ;
```

```
    Cathy.sleep();
```

```
    Cathy.eat() ;
```

```
}
```

成員函數的訂定

```
class test {  
    int a ; // private default  
public:  
    void set_a(int n) {  
        a = n ;  
    }  
    int get_a() { return a;}  
};
```

```
class test{  
    int a;  
public:  
    void set_a(int n) ;  
    int get_a() ;  
};  
void test::set_a(int  
    n){a=n;}  
int test::get_a() { return a;}
```

範例一：成員函數的使用

```
class myclass {
    int a ; // private default
public:
    void set_a(int n) {a = n ;}
    int get_a() { return a;}
};

void main() {
    myclass ob1, ob2 ;
    ob1.set_a(10) ; ob2.set_a(99) ;
    cout << ob1.get_a() << ob2.get_a() <<endl ;
}
```


範例二: 不可取用private members

```
void main() {  
    myclass ob ;  
    ob.a = 10 ;  
    .....  
}
```

範例四: Stack類別

- 參考課本

1-1 何謂物件導向程式設計

- **(Object-Oriented Programming)**
物件導向程式設計概念
 - 封裝(Encapsulation)
 - 繼承(Inheritance)
 - 多型(Polymorphism)

封裝(Encapsulation)

- 結合資料(**data**)與處理函式為一體(物件)。
 - 使用**class**
- 使用者不必瞭解物件中的詳細資料結構與函數實作，就可使用該物件
- 避免外界干擾或誤用內部資料及函式
 - 使用**private**、**protected**及**public**

封裝

使用者: 使用Stack的push與pop功能

(封裝)

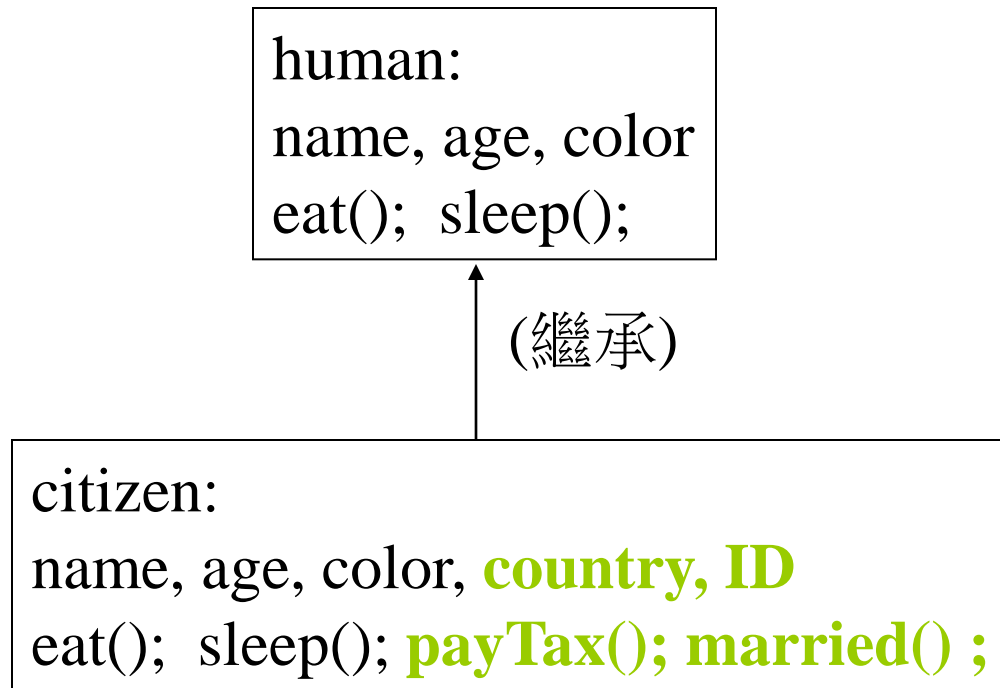
Stack (功能規格):
push(), pop()

Stack 實作一

Stack 實作二

Stack實作三

繼承(Inheritance)



多型(Polymorphism)

- 同一個函數名稱可以有不同的詮釋
- 編譯時期多型
 - function overloading
 - operator overloading
- 執行時期多型

1-8 C++的關鍵字

- 1-40頁

2-1 建構子與解構子 簡介類別

2-2 接受參數的建構子

2-3 簡介繼承 ✕

2-4 物件指標

2-5 類別、結構與聯合彼此相關

2-6 行內函數(inline functions)

2-7 自動化行內

2-1 建構函數與解構函數

- 建構子函數(**constructor functions**)
 - 物件生成時所呼叫的成員函數
- 解構子函數(**destructor functions**)
 - 物件消滅時所呼叫的成員函數

物件的生成與消滅

```
1 int main() {  
2     myclass ob1 ;  
3     ob1.show() ;  
4     fun() ;  
5     return 0;  
6 }  
7 void fun() {  
8     myclass ob2;  
9     ob2.show() ;  
10 }
```

建構子

EX: 將屬於myclass的物件的初始值設為10

```
class myclass {  
    int a ;  
public:  
    myclass(); // default constructor  
    void show() ;  
};
```

```
myclass::myclass() { a= 10 ; }
```

```
void myclass::show() { cout << a<<endl;}
```

```
int main() { myclass ob; ob.show(); return 0; }
```

建構子(constructor):

- (1)也是成員函數
- (2)與class同名
- (3)沒有回傳值
- (4)可以有很多個
- (5)允許傳參數

物件生成時，會呼叫建構子

解構子(Destructor)

```
class myclass {  
    int a ;  
public:  
    myclass() {a=10;}  
    ~myclass() {cout <<“物件消滅” <<endl; } //解構子  
    void show() { cout << a<<endl; }  
};  
int main() { myclass ob; ob.show(); return 0; }
```

解構子(destructor):

- (1)也是成員函數
- (2)~ + classname
- (3) 沒有回傳值
- (4)只能有一個
- (5)不允許傳參數

範例一

```
class stack {  
    char stack[10]; int tos ;  
public:  
    stack() { tos = 0 ; }  
    .....  
};  
void main() { stack s1, s2 ; .....}
```

範例二：動態記憶體配置與 costructor、destructor

```
class strtype {  
    char *p; int len ;  
public:  
    strtype() { p = new char[255]; *p='\0'; len = 0 ; }  
    ~strtype() { delete[] p ; }  
    void set(char *ptr) { strcpy(p,ptr) ; len = strlen(p);}  
};  
int main() { strtype p ; p.set("Hello"); return 0;}
```

範例三：有趣的計數器

```
#include <ctime> // 原time.h
class timer {
    clock_t start ;
public:
    timer() {start=clock();}
    ~timer() {
        clock_t end = clock();
        cout << “經過時間:” << (end-start)/CLOCKS_PER_SEC
        << endl ;
    }
};
```


範例三：有趣的計數器(續)

```
int main() {  
    timer ob ;  
    // ....做你的事  
    /* for(int i=0;i<30000;i++)  
        for(int j=0;j<30000;j++)  
        {  
        }*/  
    .....  
}
```

2-2 接受參數的建構子

```
void main() {  
    myclass ob1, ob2(2), ob3(ob1) ;  
    ob1.show() ;  
    ob2.show() ;  
    ob3.show() ;  
}
```

接受參數的建構子

```
1 class myclass {  
2     int a;  
3     public:  
4         myclass() { a = 10 ;} // 名字?  
5         myclass(int x) { a = x; } //名字?  
6         myclass(const myclass& ob) { a = ob.a; }  
7         void show(){cout<<a<<endl;}  
8     };  
9     void main() { myclass ob1, ob2(2),  
    ob3(ob1) ;...}
```

注意

建造類別時，不可缺少的建構子：

(1) default constructor

(2) copy constructor

範例一：多個參數的建構子

```
class myclass {  
    int a, b;  
public:  
    myclass(int x, int y) { a = x; b=y ;}  
    ...  
};  
void main() { myclass ob(5,3) ; ..... }
```

範例二：替物件取名

```
class stack {  
    char stk[20]; int tos; string who;  
public:  
    stack(string s) { tos = 0 ; who = s ; }  
    .....  
}  
void main() {stack s("Purchase Record") ;...}
```

範例三

```
class strtype {
    char *p; int len ;
public:
    strtype(char *s) {
        p = new char[strlen(s)+1] ;
        len = strlen(s) ;
    }
};
void main() { strtype str("Hello") ; ..... }
```

(1) 沒檢查char *s
(2) new之後沒檢查

範例四：讓使用者指定初值

```
class myclass{
    int i, j ;
public:
    myclass(int a, int b) { i = a; j = b; }
    ...
};
int main() {
    int x, y ; cin >> x >>y ;
    myclass ob(x,y) ; .....
}
```


2-4 物件指標

```
#include <iostream>
using namespace std ;
class myclass { int a; .... set(int x) ... get()...} ;
void main() {
    myclass ob(120) ;
    myclass * p ; // p的資料型態? p儲存什麼?
    p = &ob ;
    ob.show() ; p->show() ;
}
```

物件指標

```
int main() {  
    myclass ob(120) ; ob.show() ;  
    cubic(&ob) ; ob.show() ;  
    return 0 ;  
}  
  
void cubic(myclass* p) {  
    p->set(pow(p->get(),3)) ;  
}
```

2-5 類別、結構與聯合彼此相關

```
class complex {
```

```
};
```

```
struct complex {
```

```
};
```

```
union complex {
```

```
};
```

class vs. struct

```
class complex {
    int a, b; // a+bi
public:
    set(int x, int y) ;
    print() ;
};
void main() {
    complex c ;
    c.set(5,3); c.print() ;
}
```

```
struct complex {
    set(int x, int y) ;
    print() ;
private:
    int a, b;
};
void main() {
    complex c ;
    c.set(5,3); c.print() ;
}
```

Class vs. Struct

- 異同
 - 都可以宣告資料成員(data members)與成員函數(member functions)
 - 預設的存取限制: private vs. public
- 討論
 - 既生struct ，何生class

Class vs. Struct

- 習慣

```
class complex {  
    int a, b ;  
public:  
    set(int x, int y) ;  
    .....  
};
```

```
struct student {  
    string name ;  
    double GPA;
```

```
};
```

Union

- 多個變數共用一個記憶體區塊

struct sbits {

double d ;

char c[sizeof(double)] ;

};

union ubits {

double d ;

char c[sizeof(double)] ;

};

double d;	8-bytes
char c[8];	8-bytes

d, c[]共用	8-bytes
-----------------	---------

union vs. struct

```
void main() {  
    sbits a ; //使用struct  
    a.d = 314898;  
    strcpy(a.c, "Hello") ;  
    ubits b ; //使用union  
    b.d = 314898;  
    strcpy(b.c, "Hello") ;  
}
```

d:314898

c[]: ????

d:314898

c[]: "Hello"

314898

"Hello"

匿名聯合(anonymous)

// 不必特別宣告變數

```
void main() {  
    union fourbits {int i ; char ch[4]; } ;  
    fourbits x ;  
    x.i = 10; strcpy((char *)x.ch,"sam") ;  
    union { int i ; char ch[4];}; //匿名聯合  
    i = 10 ;  
    strcpy(ch, "sam") ;  
}
```

範例二: **union**也可以有成員函數

//自行run一次

```
union bits {  
    bits(double n) ;  
    void show_bits() ; //show 出每一個位元  
    double d ;  
    unsigned char c[sizeof(double)] ;  
};  
bits::bits(double n) { d = n; }  
void bits::show_bits(){cout<<d<<c<<endl;}  
.....
```

2-6 行內函數(inline functions)

- 編譯時會被展開的函數 (通常適用於小函數)

```
bool odd(int x) { return (x%2)!=0 ; }
```

```
inline bool even(int x) { return (x%2)==0; }
```

```
int main() {
```

```
    int a ; cin >> a ;
```

```
    if (odd(a)) cout << a << "is odd" ;
```

```
    if (even(a)) cout << a << "is even" ;
```

```
    if (even(3.14)) cout << 3.14<<"is even" ;
```

```
}
```

行內函數的優缺點

- 增加程式執行效率
 - － 沒有呼叫與回傳的動作
- 可執行檔的大小可能變大
 - － 如果呼叫次數很多，而且函數又不只一行

注意事項

- 行內函數必須在使用前先被定義過
- 並不是宣告成**inline**就會被展開
 - 編譯器會執行cost/benefit analysis

ex:

```
inline int fun() {  
    if.... for .... switch....for...  
}
```

範例一

```
class samp {  
    int i, j;  
public:  
    samp(int a, int b) ;  
    bool divisible() ; // i是否為j的除數  
};  
samp::samp(int a, int b) { i = a; j =b;}  
inline bool samp::divisible() { return (i%j)==0; }
```

範例二: inline 函數也可以超載

```
inline int min(int a, int b) {  
    return (a<b)?a:b ;  
}
```

```
inline char* min(char *s1, char *s2) {  
    return (strcmp(s1, s2)<0)?s1, s2 ;
```

2-7 自動化行內 (Automatic Inline)

// 以下哪些成員函數會被視為inline functions

```
class samp {
```

```
    int i, j ;
```

```
public:
```

```
    samp(int a, int b) {i = a; j = b ; }
```

```
    bool divisible() { return (i%j)==0; }
```

```
    int gcd();
```

```
};
```

```
int samp::gcd() {.....}
```


範例一、二

- 自行參考課本