Chapter 7 Classes

Yu-Hsiang Cheng
鄭宇翔 (Slighten)
GOTC Instructor

http://m101.nthu.edu.tw/~s101021120/Myweb/index_ch.html

Outline

- The Motivation of Using Classes
- Introduction of Classes
- Using C++ STL
- Applications of C++

The Motivation of Using Classes

Motivation

- 假設你想要用程式模擬建造出一個機器人！
  - 他有一些「特徵」！(屬性, attributes, 狀態, states, 欄位, fields)
    => 用「變數」實踐
    - 名字：Dorakemon
    - 身高 129.3 cm、體重 129.3 kg
    - 顏色：藍
    - 所在位置 (xyz座標)
    - …
  - 他會做一些「動作」！(方法, methods, 行為, behaviors)
    => 用「函式」實踐
    - 打招呼
    - 移動
    - …

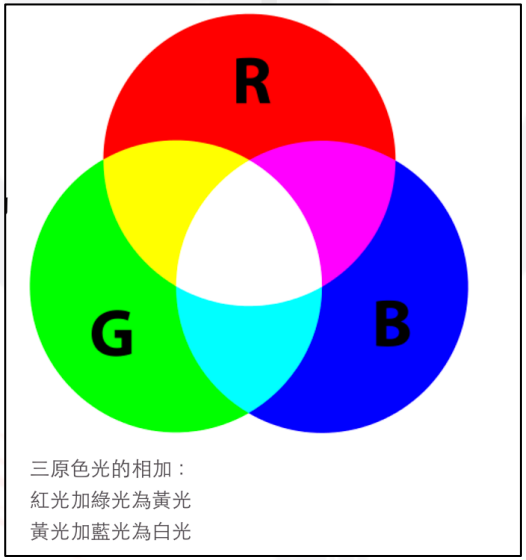
The Oldest (Worst) Way

- Recall: “ or ”(中文的雙引號) ≠ " (英文的雙引號)
- 一定要用英文的單、雙引號

```
5  string name = "Dorakemon”;
6  double height = 129.3, weight = 129.3;
7  int rgb = 0x0000ff;
8  int _x = 0, _y = 0, _z = 0;
9  void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
10 void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
```

What is rgb? (1/2)

- **三原色光模式（RGB color model），又稱RGB顏色模型或紅綠藍顏色模型**，是一種加色模型，將紅（Red）、綠（Green）、藍（Blue）三原色的色光以不同的比例相加，以產生多種多樣的色光。



三原色光的相加：
紅光加綠光為黃光
黃光加藍光為白光

- 簡單來說，就是一種在電腦裡的顏色表示法！

What is rgb? (2/2)

- 一個顏色顯示的描述是由三個數值控制的，分別為R、G、B。但三個數值位為最大時，顯示為白色，當三個數值最小時，顯示為黑色。
- 常見數值表示方式

方式	RGB 表示
浮點	(1.0, 0.0, 0.0)
百分比	(100%, 0%, 0%)
八位數字	(255, 0, 0) 或 #FF0000 (十六進位)
十六位數字	(65535, 0, 0)

- 以 0 ~ 255 表示顏色的明度 (小 ~ 大)
  - 0xFF0000 => R: 255 (Max), G: 0 (Min), B: 0 (Min) => Red
  - 0xFFFFFF => R: 255 (Max), G: 255 (Max), B: 255 (Max) => White
  - 0x000000 => R: 0 (Min), G: 0 (Min), B: 0 (Min) => Black
  - 0x808080 => R: 128, G: 128, B: 128 => Gray
  - 0x0000FF => R: 0 (Min), G: 0 (Min), B: 255 (Max) => Blue
- 想要自己配色？ Click Here: Html Color Codes

Back to the Traditional (Worst) Way

```
5  string name = "Dorakemon";
6  double height = 129.3, weight = 129.3;
7  int rgb = 0x0000ff;
8  int _x = 0, _y = 0, _z = 0;
9  void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
10 void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
```

- 如果想要再建一個機器人叫作 *Beimax*?
- 再一個機器人！
  - 他有一些「特徵」！(屬性, attributes, 狀態, states, 欄位, fields)
    - 名字： *Beimax*
    - 身高 210 cm、體重 3 kg
    - 顏色：白
    - 所在位置 (xyz座標)
    - …
  - 他會做一些「動作」！(方法, methods, 行為, behaviors)
    - 打招呼
    - 移動
    - 擁抱
    - …

The Traditional (Worst) Way

```
5  string name = "Dorakemon";
6  double height = 129.3, weight = 129.3;
7  int rgb = 0x0000ff;
8  int _x = 0, _y = 0, _z = 0;
9  void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
10 void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
11
12 string name2 = "Beimax";
13 double height = 250.0, weight = 3.0;
14 int rgb = 0xffffff;
15 int _x2 = 0, _y2 = 0, _z2 = 0;
16 void sayHello2(){ cout << "Hello! My name is " << name2 << ".\n"; }
17 void moveTo2(int x, int y, int z){ _x2 = x; _y2 = y; _z2 = z; }
18 void hug() { /* ... */ }
```

- Typo
  - line 13: height –> height2, weight –> weight2
  - line 14: rgb –> rgb2

A Little Bit Better Version (Array)

```
20 string name[2] = {"Dorakemon", "Beimax"};
21 double height[2] = {129.3, 250.0}, weight[2] = {129.3, 3.0};
22 int rgb[2] = {0x0000ff, 0xffffff};
23 int _x[2] = {0, 0}, _y[2] = {0, 0}, _z[2] = {0, 0};
24 void sayHello(string name){ cout << "Hello! My name is " << name << ".\n"; }
25 void moveTo(int id, int x, int y, int z){ _x[id] = x; _y[id] = y; _z[id] = z; }
26 void hug() { /* ... */ }
```

- 優點
  - 比前一頁方式的還簡潔
- 缺點
  1. function 需要多一些參數 (例如：id)來告訴 function 你是要指定哪一個機器人
  2. 不好擴增：第 3 隻、第 4 隻、第 100 隻 ... ?

A Little Better Version (Struct + Array)

```
28 typedef struct _ROBOT{
29     string name;
30     double height, weight;
31     int rgb;
32     int _x, _y, _z;
33 } Robot;
34
35 Robot robot[2] = {
36     {"Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0},
37     {"Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0}
38 };
39 void sayHello(string name){ cout << "Hello! My name is " << name << ".\n"; }
40 void moveTo(int id, int x, int y, int z){ robot[id]._x = x; robot[id]._y = y; robot[id]
       ._z = z; }
41 void hug() { /* ... */ }
```

結構定義(不是在宣告變數)

這裡才是宣告變數

- 優點
  1. 容易擴增： 1000 隻都沒問題
  2. 容易新增特徵：例如年齡
  3. Access: `robot[0].rgb;`
- 缺點
  1. 不包括函式 (事實上可以用函數指標來達成，但頗麻煩)
     - 也想要 `robot[0].sayHello();` 可以嗎？
  2. 變數初始化不太方便 (第 35~38 行)，尤其當你修改特徵之後更麻煩

```
28 typedef struct _ROBOT{
29     string name;
30     double height, weight;
31     int rgb;
32     int _x, _y, _z;
33     int age;
34 } Robot;
35
36 Robot robot[2] = {
37     {"Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0, 43},
38     {"Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0, 2}
39 };
```

A Total Solution: Class (類別) (1/2)

- 萬靈丹，一次搞定所有缺點。

```
50 class Robot {
51 public:
52
53     // Constructor
54     Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
55            int x = 0, int y = 0, int z = 0) :
56             name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
57     // Deconstructor
58     ~Robot() {}
59     void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
60     void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
61
62 private:
63
64     string name;
65     double height, weight;
66     int rgb;
67     int _x, _y, _z;
68
69 };
70
71 Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);
72 Robot r2 = Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0);
73
74 Robot robot[2] = {
75     Robot("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0),
76     Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0)
77 };
78
79 int main(){
80     robot[0].sayHello();
81 }
```

類別定義(不是在宣告變數)

這裡才是宣告變數

A Total Solution: Class (類別) (2/2)

- Class：就像是一個更廣義、更方便的 struct
  - 除了包含「特徵」(變數)，也可以包含「動作」(函式)
  - 也包含了如何初始化 (Constructor, 建構子)
  - public/private, 繼承 (inheritance) …
- Declaration (宣告變數): 型別(type) 變數(variable);

```
int i;
int &ri = i;
int *ptr = &i;
int arr[2] = {1, 2};
char c;
bool isDead;
```

- Instantiation(實體化、創造物件): 類別(class) 物件(object) = …;

```
Robot r2 = Robot("Beimax",
250.0, 3.0, 0xffffff, 0, 0, 0);
Robot *r3 = &r2;
Robot &r4 = r2;
Robot robot[2] = { r2, r4 };
string name;
```

- 宣告變數 == 創造物件
- 從此，寫程式從以前的先想「要做什麼」，而開始定義 function，改成：先想「有什麼」，而開始定義 class
- 這種以建造「物件」為主軸來思考的程式設計方式叫作物件導向程式設計 (OOP, object-oriented programming)
  - 多型 (polymorphism)、繼承 (inheritance)、泛型 (generic)…

Introduction of Classes

Terminology

- Class 包含的東西我們在 C++ 裡叫他們為「成員」(Members)
- 變數叫「成員變數」 Data members、函式叫 「成員函式」 Member functions

```
50 class Robot {
51 public:
52
53     // Constructor
54     Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
55           int x = 0, int y = 0, int z = 0) :
56            name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
57     // Deconstructor
58     ~Robot() {}
59     void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
60     void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
61
62 private:
63
64     string name;
65     double height, weight;
66     int rgb;
67     int _x, _y, _z;
68
69 };
```

Member functions

Data Members

Naming a Class

- 記得之前教的 Naming Convention
- class 名稱是大駝峰式命名法(UpperCamelCase)
- `class Robot { /*…*/ };`
- `class RobotArm { /*…*/ };`

```
50 class Robot {
51 public:
52
53     // Constructor
54     Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
55           int x = 0, int y = 0, int z = 0) :
56            name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
57     // Deconstructor
58     ~Robot() {}
59     void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
60     void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
61
62 private:
63
64     string name;
65     double height, weight;
66     int rgb;
67     int _x, _y, _z;
68
69 };
```

public/private ? (1/2)

- 成員的取用層級
- Access Level(高到低): `public/protected/private`
- `public`: 自己跟外部都可以 access (包括取得、更改) 自己的 members
- `protected`: 自己與自己的父親都能 access 自己的 members (等教到繼承(inheritance)會再細部說明)
- `private`: 只有自己能 access 自己的 members

```
50 class Robot {
51 public:
52
53     // Constructor
54     Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
55           int x = 0, int y = 0, int z = 0) :
56            name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
57     // Deconstructor
58     ~Robot() {}
59     void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
60     void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
61
62 private:
63
64     string name;
65     double height, weight;
66     int rgb;
67     int _x, _y, _z;
68
69 };
```

public/private ? (2/2)

```
50 class Robot {
51 public:
52
53     // Constructor
54     Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
55            int x = 0, int y = 0, int z = 0) :
56             name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
57     // Deconstructor
58     ~Robot() {}
59     void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
60     void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
61
62 private:
63
64     string name;
65     double height, weight;
66     int rgb;
67     int _x, _y, _z;
68
69 };
70
71 Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);
72 Robot r2 = Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0);
73
74 Robot robot[2] = {
75     Robot("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0),
76     Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0)
77 };
78
79 int main(){
80     robot[0].sayHello();
81     robot[0].rgb = 3;
82     cout << r1.name;
83 }
```

行 81~82 都會造成 compile error,
因為 access private members

Constructor (建構子) (1/6)

- 代表當你創造一個物件
  (例如宣告 r2 用
  `Robot r2 = Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0);`)
- 第一個會被呼叫的就是「建構子」函式
- 功能是：自訂成員變數的初始化方式

```
50 class Robot {
51 public:
52
53     // Constructor
54     Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
55           int x = 0, int y = 0, int z = 0) :
56            name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
57     // Deconstructor
58     ~Robot() {}
59     void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
60     void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
61
62 private:
63
64     string name;
65     double height, weight;
66     int rgb;
67     int _x, _y, _z;
68
69 };
```

Constructor (建構子) (2/6)

- 建構子函式名稱要 == class 名稱
- 不用回傳型別，因為就是回傳自己 class 這個型別 (Robot)
- 參數給幾個都可以，但必須與創造物件時給的參數個數相同
- 最簡單的形式，給 0 個參數，並不做任何的初始化，預設建構子 (default constructor): `Robot(){}`
- 那麼創建物件時就得 0 個
  - `Robot r1;`
  - ~~`Robot r2 = Robot;`~~
  - `Robot r3 = Robot();`
- 可以有多個同樣名稱的(建構子)函式：Overload （重載、多載）
- Overload 定義： 當有 2 個以上的函式名稱相同，但不同參數個數/型別，且回傳型別為 compatible (e.g.: `double`, `float`, `int`) 時，就稱函式的重載
- 創建物件時依照參數個數/型別編譯器就可以知道該用哪個建構子函式

```
// Constructor
Robot() {}
Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
      int x = 0, int y = 0, int z = 0) :
       name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
```

Constructor (建構子) (3/6)

- Member–initializer list
- `Robot (int x, int y, int z) : a(x), b(y), c(z) {}`
- 等同
- `Robot (int x, int y, int z) { a = x; b = y; c = z; }`

```
// Constructor
Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
       int x = 0, int y = 0, int z = 0) :
        name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
```

```
Robot(string str, double h = 0.0, double w = 0.0, int color = 0, int x = 0, int y = 0, int z = 0) {
    name = str;
    h = height;
    w = weight;
    rgb = color;
    _x = x;
    _y = y;
    _z = z;
}
```

Data Members

```
private:

    string name;
    double height, weight;
    int rgb;
    int _x, _y, _z;
```

Constructor (建構子) (4/6)

- explicit 關鍵字：prevent implicit type-conversion

```
// Constructor
explicit Robot(int a){ _x = a; }
```

- 若有 explicit 則創造物件時只能 Robot r1(3);
- 不能 Robot r2 = 3; (implicit type-conversion)
- 不能 Robot r3(3.0); (implicit type-conversion)
- http://www.cplusplus.com/forum/general/168292/

Constructor (建構子) (5/6)

- Default arguments (預設參數): 有 type name = value 的參數
- 例如參數 h, w, color, x, y, z
- 代表創建物件時若沒給值就給 default 值
- 一般: `Robot r1 = Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0);`
- 也可: `Robot r2 = Robot("Beimax", 250.0, 3.0 , 0xffffff, 0, 0);`
  - 代表參數 `z = 0;` 因此 `_z = 0;`
- 也可: `Robot r3 = Robot("Beimax", 250.0, 3.0);`
  - 代表參數 `color = 0; x = 0; y = 0; z = 0;` 因此 `rgb = 0; _x = 0; _y = 0; _z = 0;`
- 最少: `Robot r4 = Robot("Beimax");`
  - 剩下參數 `h = w = color = x = y = z = 0;`

```
// Constructor
Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
      int x = 0, int y = 0, int z = 0) :
       name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
```

Constructor (建構子) (6/6)

- Initialize with initializer list
- 創造物件語法可以下列方式
  - `Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);`
  - `Robot r2 = Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0);`
- 新版本還可使用 initializer list (初始子清單) (C++11 以後版本才有)
  - `Robot r1{"Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0};`
  - `Robot r2 = {"Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0};`
  - 編譯時需加 –std=c++11 選項: `g++ -std=c++11 test.cpp`

Destructor (解構子)

- 解構子函式格式: ~ (tilda) 開頭 + class 名稱
- 無參數，無回傳，無回傳型別
- 當物件死亡時被呼叫 (Recall 變數的生存期間與能見度)
- 功能為自訂當物件被破壞時要清理哪些成員變數，通常都是 `delete` some pointers，以節省記憶體空間
- 一個 class 一個 destructor
- 必須 `public`
- e.g. `~Robot(){}`

```
50 class Robot {
51 public:
52
53     // Constructor
54     Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
55            int x = 0, int y = 0, int z = 0) :
56             name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
57     // Deconstructor
58     ~Robot() {}
59     void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
60     void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
61
62 private:
63
64     string name;
65     double height, weight;
66     int rgb;
67     int _x, _y, _z;
68
69 };
```

Constructor (建構子) & Destructor (解構子)

- Question: constructor & destructor 內能否放 cout ?
- Answer: 當然可以，他們也是函式 (可以試試看，並在 `main()` 裡面創造物件 `Robot r1;` 並透過 `cout` 來判斷何時 constructor 被呼叫，何時 destructor 被呼叫)
- `Robot () { cout << "constructor" << endl; }`
- `~Robot () { cout << "destructor" << endl; }`
- 記得 `#include <iostream>` 與 `using namespace std;`
- Debug 時經常也會用到

Setters and Getters (1/3)

- 因為將成員變數設成 `private` 後，將無法透過 '.' 去 access 它們
- 但我們仍想取得與更改它們，How? Use setters and getters.
- Recall: public/private ? (2/2)

```
50 class Robot {
51 public:
52
53     // Constructor
54     Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
55           int x = 0, int y = 0, int z = 0) :
56            name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
57     // Deconstructor
58     ~Robot() {}
59     void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
60     void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
61
62 private:
63
64     string name;
65     double height, weight;
66     int rgb;
67     int _x, _y, _z;
68
69 };
70
71 Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);
72 Robot r2 = Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0);
73
74 Robot robot[2] = {
75     Robot("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0),
76     Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0)
77 };
78
79 int main(){
80     robot[0].sayHello();
81     robot[0].rgb = 3;
82     cout << r1.name;
83 }
```

行 81~82 都會造成 compile error,
因為 access private members

Setters and Getters (2/3)

- e.g. Want to access the data member -- rgb

```
50 class Robot {
51
52 public:
53     // Constructor
54     Robot(string str, double h = 0.0, double w = 0.0,
55           int color = 0, int x = 0, int y = 0, int z = 0) :
56     name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
57     // Destructor
58     ~Robot() {}
59     void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
60     void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
61     // A setter
62     void setRgb(int color) { rgb = color; }
63     // A getter
64     int getRgb() { return rgb; }
65
66 private:
67     string name;
68     double height, weight;
69     int rgb;
70     int _x, _y, _z;
71
72 };
73
74 Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);
75 Robot r2 = Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0);
76
77 int main(){
78     r1.sayHello();
79     r1.setRgb(3); // r1.rgb = 3;
80     cout << r1.getRgb(); // r1.rgb
81 }
```

Setters and Getters (3/3)

- 對於每個 private 成員變數都要有一個 setter 與 getter – 太麻煩
- 有時乾脆都 public，就不用 setters and getters – 方便但不安全

```
50 class Robot {
51
52 public:
53     // Constructor
54     Robot(string str, double h = 0.0, double w = 0.0,
55           int color = 0, int x = 0, int y = 0, int z = 0) :
56     name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
57     // Destructor
58     ~Robot() {}
59     void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
60     void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
61
62     string name;
63     double height, weight;
64     int rgb;
65     int _x, _y, _z;
66
67 };
68
69 Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);
70 Robot r2 = Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0);
71
72 int main(){
73     r1.sayHello();
74     r1.rgb = 3;
75     cout << r1.rgb << endl; // r1.rgb
76 }
```

Placing a Class in a Separate File I (1/5)

- 通常 class 很大，會放在另一個檔案 (標頭檔, header file)
- 原則上一個檔案一個 class，檔案名稱 = class 名稱
- e.g. 將 Robot class 放在 robot.h，將 main() 放在 main.cpp

robot.h — Edited

robot.h

robot.h 〉No Selection

```
#ifndef ROBOT_H
#define ROBOT_H

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

class Robot {

public:
    // Constructor
    Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
          int x = 0, int y = 0, int z = 0) :
    name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
    // Destructor
    ~Robot() {}
    void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
    void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
    // A setter
    void setRgb(int color) { rgb = color; }
    // A getter
    int getRgb() { return rgb; }

private:
    string name;
    double height, weight;
    int rgb;
    int _x, _y, _z;

};

#endif
```

main.cpp

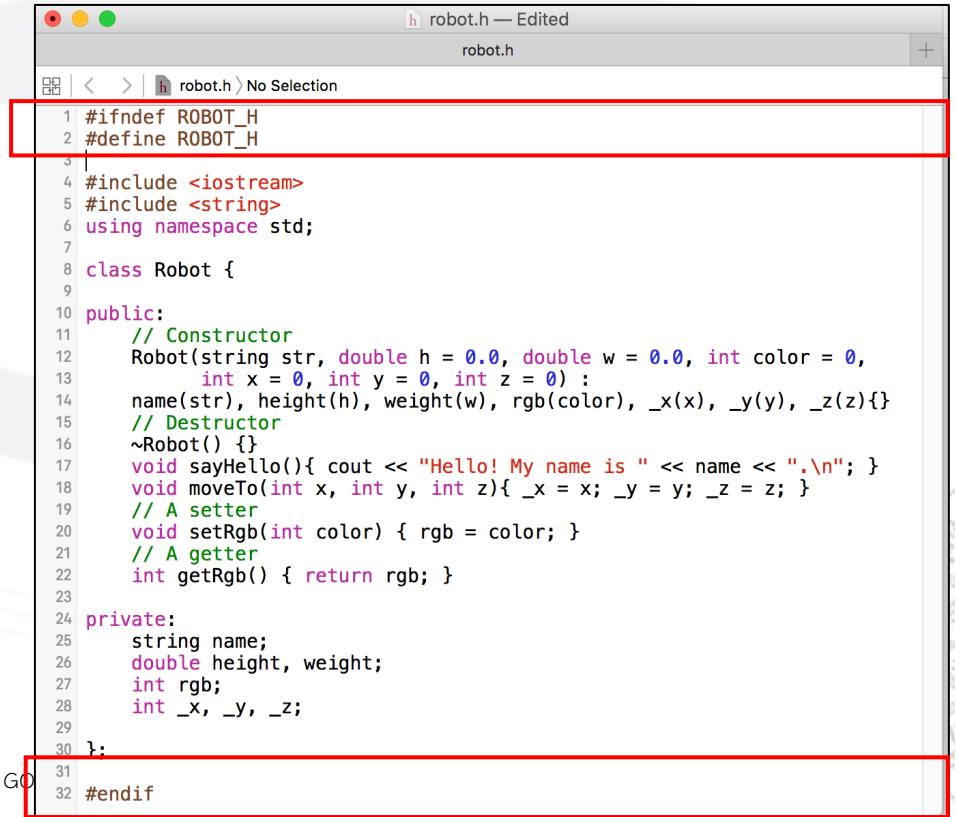
main.cpp

main.cpp 〉main()

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

int main(){
    Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);
    r1.sayHello();
    r1.setRgb(999); // r1.rgb = 3;
    cout << r1.getRgb() << endl;
}
```

Placing a Class in a Separate File I (2/5)

- 先看 robot.h
- A include guard: Preprocessor directive，避免此標頭檔被重複 include
-
```
#ifndef ROBOT_H // 如果沒有定義過 ROBOT_H (if not define)
#define ROBOT_H // 定義 ROBOT_H
    // ...
#endif // 就是 end if
```



Placing a Class in a Separate File I (3/5)

- 再看 main.cpp
- 記得把 robot.h 放在 main.cpp 旁邊，並 `#include "robot.h"`
- 如果 robot.h 是放在 main.cpp 旁邊的 header 資料夾底下，那就 `#include "header/robot.h"`

main.cpp

main.cpp

main.cpp 〉 main()

```
1  #include <iostream>
2  #include "robot.h"
3  using namespace std;
4
5  int main(){
6      Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);
7      r1.sayHello();
8      r1.setRgb(999); // r1.rgb = 3;
9      cout << r1.getRgb() << endl;
10 }
```

Placing a Class in a Separate File I (4/5)

- Question: 兩個檔案都有 `#include <iostream>`，那我在 main.cpp 裡`#include "robot.h"` 不就又再重複 `#include <iostream>` 一次嗎，為什麼沒問題？
- Answer: 因為 iostream.h 有自己的 include guard，請打開 iostream.h (command + click in MacOS Xcode) 看看。

robot.h — Edited

robot.h

robot.h 〉No Selection

```
#ifndef ROBOT_H
#define ROBOT_H

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

class Robot {

public:
    // Constructor
    Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
          int x = 0, int y = 0, int z = 0) :
    name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
    // Destructor
    ~Robot() {}
    void sayHello(){ cout << "Hello! My name is " << name << ".\n"; }
    void moveTo(int x, int y, int z){ _x = x; _y = y; _z = z; }
    // A setter
    void setRgb(int color) { rgb = color; }
    // A getter
    int getRgb() { return rgb; }

private:
    string name;
    double height, weight;
    int rgb;
    int _x, _y, _z;

};

#endif
```

main.cpp

main.cpp

main.cpp 〉main()

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

int main(){
    Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);
    r1.sayHello();
    r1.setRgb(999); // r1.rgb = 3;
    cout << r1.getRgb() << endl;
}
```

Placing a Class in a Separate File I (5/5)

- Question:兩個檔案都有 `#include <iostream>`，那我在 main.cpp 裡 `#include "robot.h"` 不就又再重複 `#include <iostream>` 一次嗎，為什麼沒問題？
- Answer: 因為 iostream.h 有自己的 include guard，請打開 iostream.h (command + click in MacOS) 看看。

iostream

```
// -*- C++ -*-
//===--------------------------- iostream ---------------------------------===//
//
//                     The LLVM Compiler Infrastructure
//
// This file is dual licensed under the MIT and the University of Illinois Open
// Source Licenses. See LICENSE.TXT for details.
//
//===----------------------------------------------------------------------===//

#ifndef _LIBCPP_IOSTREAM
#define _LIBCPP_IOSTREAM

/*
    iostream synopsis

#include <ios>
#include <streambuf>
#include <istream>
#include <ostream>

namespace std {

extern istream cin;
extern ostream cout;
extern ostream cerr;
extern ostream clog;
extern wistream wcin;
extern wostream wcout;
extern wostream wcerr;
extern wostream wclog;

}  // std

*/

#include <__config>
```

Placing a Class in a Separate File II (1/3)

- 通常 class 很大，會放在另一個檔案 (標頭檔, header file)
- 原則上一個檔案一個 class，檔案名稱 = class 名稱
- 但因為 class 的 member functions 可能很複雜，因此可能將 function declaration (function prototype) 放在 .h 檔，function definition 放在 .cpp 檔裡
- e.g. 將 class 放在 robot.h (改)，將 function definition 放在 robot.cpp ，將 main() 放在 main.cpp (同)

robot.h

```h
#ifndef ROBOT_H
#define ROBOT_H
#include <string>

class Robot {

public:
    // Default Constructor
    Robot(std::string , double, double, int,
          int, int, int);
    // Default Destructor
    ~Robot();
    void sayHello() const;
    void moveTo(int, int, int);
    // A setter
    void setRgb(int);
    // A getter
    int getRgb() const;

private:
    std::string name;
    double height, weight;
    int rgb;
    int _x, _y, _z;

};

#endif
```

robot.cpp

```cpp
#include <iostream>
#include <string>
#include "robot.h"
using namespace std;

Robot::Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
      int x = 0, int y = 0, int z = 0) :
name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}

Robot::~Robot() {}

void Robot::sayHello() const {
    cout << "Hello! My name is " << name << ".\n";
}

void Robot::moveTo(int x, int y, int z){
    _x = x;
    _y = y;
    _z = z;
}

// A setter
void Robot::setRgb(int color) {
    rgb = color;
}

// A getter
int Robot::getRgb() const {
    return rgb;
}
```

Placing a Class in a Separate File II (2/3)

- robot.h: member functions 只剩 prototype (Recall Chapter 4: Functions)
- 不用 include iostream 因為用不到，沒有 using namespace std; 就要補上 std:: (Recall Chapter 1: Introduction)

robot.h

robot.h > sayHello()

```
1  #ifndef ROBOT_H
2  #define ROBOT_H
3  #include <string>
4
5  class Robot {
6
7  public:
8      // Default Constructor
9      Robot(std::string , double, double, int,
10           int, int, int);
11     // Default Destructor
12     ~Robot();
13     void sayHello() const;
14     void moveTo(int, int, int);
15     // A setter
16     void setRgb(int);
17     // A getter
18     int getRgb() const;
19
20 private:
21     std::string name;
22     double height, weight;
23     int rgb;
24     int _x, _y, _z;
25
26 };
27
28 #endif
```

const 是?

const Objects and const Functions

- 創造一個 const object:
  `const Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);`
  => 代表物件 r1 被 construct 之後成員變數的值就不會再被改變
  - 呼叫 non-const functions 會 compile error: e.g. `r1.setRgb(30);`
- const function 代表這個 member function 不會更動 data members 的值
  - 更動就 compile error: e.g. 宣告 `void setRgb(int) const;`
- 因此 const objects 可以呼叫 const functions，但不能呼叫 non-const functions
- 一般 (non-const) objects 沒差都可

main.cpp

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

int main(){
    const Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0)
        ;
    r1.sayHello();
    r1.setRgb(999); // compile error
    cout << r1.getRgb() << endl;
}
```

robot.h

```cpp
#ifndef ROBOT_H
#define ROBOT_H
#include <string>

class Robot {

public:
    // Default Constructor
    Robot(std::string , double, double, int,
          int, int, int);
    // Default Destructor
    ~Robot();
    void sayHello() const;
    void moveTo(int, int, int);
    // A setter
    void setRgb(int);
    // A getter
    int getRgb() const;

private:
    std::string name;
    double height, weight;
    int rgb;
    int _x, _y, _z;

};

#endif
```

Placing a Class in a Separate File II (3/3)

- robot.cpp: function definition 都在這，記得 include robot.h 和每個 function 前都要有 Robot::，代表我要定義的 function 是 class Robot 的 member function
- :: = scope resolution operator

robot.cpp

robot.cpp

robot.cpp 〉No Selection

```
1  #include <iostream>
2  #include <string>
3  #include "robot.h"
4  using namespace std;
5
6  Robot::Robot(string str, double h = 0.0, double w = 0.0, int color = 0,
7        int x = 0, int y = 0, int z = 0) :
8  name(str), height(h), weight(w), rgb(color), _x(x), _y(y), _z(z){}
9
10 Robot::~Robot() {}
11
12 void Robot::sayHello() const {
13     cout << "Hello! My name is " << name << ".\n";
14 }
15
16 void Robot::moveTo(int x, int y, int z){
17     _x = x;
18     _y = y;
19     _z = z;
20 }
21
22 // A setter
23 void Robot::setRgb(int color) {
24     rgb = color;
25 }
26
27 // A getter
28 int Robot::getRgb() const {
29     return rgb;
30 }
```

Using Makefile

- 當你把程式碼分在很多檔案，編譯時你可能需要
  ```
  % g++ a.cpp b.cpp c.cpp -o program.exe
  ```
- 指令很長，還有可能要 link 某些自己寫的 library
- 可以只用一個 make 指令替代 (有人會誤認 make 就是編譯)
- `make clean` 可以刪掉所有已產生的執行檔
- 先將所有邊指令寫在 Makefile 裡，放在旁邊
- 其實 make 就只是照著 Makefile 的規定執行指令

An Example Makefile

- # 單行註解
- CC=編譯器名稱
- LDFLAGS=一些編譯設定
- SOURCES=all ".cpp" files
- EXECUTABLE=執行檔名稱
- INCLUDES=all ".h" files

1. Cygwin 要安裝 make
2. 這邊語法不詳述，
   自行google Makefile
3. Learn X in Y minutes — X = make
   https://learnxinyminutes.com/docs/make/

Makefile

```
#
# Makefile for HW2
#

CC=g++
# if you want to use debugger, add -g to CFLAGS and LDFLAGS
CFLAGS=
LDFLAGS=-std=c++11 -O3 -lm
SOURCES=main.cpp
OBJECTS=$(SOURCES:.c=.o)
EXECUTABLE=FM_Partitioner
INCLUDES=cell.h net.h

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@

%.o:  %.c  ${INCLUDES}
	$(CC) $(CFLAGS) $< -o $@

clean:
	rm -rf *.o $(EXECUTABLE)

```

`friend` Functions and `friend` Classes (1/2)

- A `friend` function of a class is a non-member function and has the right to access both `public` and `non-public` class members.
- Non-member function: 不用 Robot:: 開頭

robot.h

```
#ifndef ROBOT_H
#define ROBOT_H
#include <string>

class Robot {

    friend void setName(Robot &, std::string);

public:
    // Default Constructor
    Robot(std::string , double, double, int,
          int, int, int);
    // Default Destructor
    ~Robot();
    void sayHello() const;
    void moveTo(int, int, int);
    // A setter
    void setRgb(int);
    // A getter
    int getRgb() const;
    std::string getName() const { return name; }

private:
    std::string name;
    double height, weight;
    int rgb;
    int _x, _y, _z;

};

#endif
```

robot.cpp

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

void setName(Robot &r1, std::string str){
    r1.name = str;
}
```

main.cpp

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

int main(){
    Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);
    r1.sayHello();
    r1.setRgb(999);
    setName(r1, "Dodoro");
    r1.sayHello();
}
```

friend Functions and friend Classes (2/2)

- B is A's friend: class A 內 friend class B; 代表 class B 裡的所有 member functions 都是 class A 的 friend functions ，都可以 access class A 的 public 和 non-public class members.
- Granted not taken: B is A's friend，A 要說而不是 B 說
- Non-symmetric: B is A's friend ⇏ A is B's friend
- Non-transitive: A is B's friend B is C's friend ⇏ A is C's friend

```
class A {
    friend class B;
    /* … */
};

class B {
    /* … */
};
```

this Pointer

- this is a pointer variable that points to the caller object. (its value is caller's address)

robot.h

robot.h > class Robot

```
#ifndef ROBOT_H
#define ROBOT_H
#include <string>

class Robot {

    friend void setName(Robot &, std::string);

public:
    // Default Constructor
    Robot(std::string , double, double, int,
          int, int, int);
    // Default Destructor
    ~Robot();
    void sayHello() const;
    void moveTo(int, int, int);
    // setters
    void setRgb(int);
    void setHeight(double h) {
        this->height = h;
        // this->height == (*this).height == height
    }
    // getters
    int getRgb() const;
    std::string getName() const { return name; }
    // Operator Overloading
    Robot & operator + (Robot &r2) {
        this->name += "_plus_" + r2.name + "_ver2.0";
        return *this;
    }

private:
    std::string name;
    double height, weight;
    int rgb;
    int _x, _y, _z;

};

#endif
```

Operator Overloading (1/2)

- 重載運算子，+, -, *, /, +=, ++, >>, << 等... 都可自訂
- 重載 + 運算子函式
- 回傳 type = Robot &
  - 可以連加
  - r4 = r1 + r2 + r3;
- 函式名稱 = operator +
- 參數只有一個 Robot &r2
- Robot a, b;
- a + b; 時
  - *this == a
  - r2(形式參數) = b(真實參數)

robot.h

robot.h

robot.h 〉 class Robot

```
1  #ifndef ROBOT_H
2  #define ROBOT_H
3  #include <string>
4
5  class Robot {
6
7      friend void setName(Robot &, std::string);
8
9  public:
10     // Default Constructor
11     Robot(std::string , double, double, int,
12           int, int, int);
13     // Default Destructor
14     ~Robot();
15     void sayHello() const;
16     void moveTo(int, int, int);
17     // setters
18     void setRgb(int);
19     void setHeight(double h) {
20         this->height = h;
21         // this->height == (*this).height == height
22     }
23     // getters
24     int getRgb() const;
25     std::string getName() const { return name; }
26     // Operator Overloading
27     Robot & operator + (Robot &r2) {
28         this->name += "_plus_" + r2.name + "_ver2.0";
29         return *this;
30     }
31
32 private:
33     std::string name;
34     double height, weight;
35     int rgb;
36     int _x, _y, _z;
37
38 };
39
40 #endif
```

Operator Overloading (2/2)

- 在 main() 使用 +
- 輸出： Hello! My name is Dorakemon_plus_Beimax_ver_2.0.

main.cpp

main.cpp

main.cpp 〉 main()

```
1  #include <iostream>
2  #include "robot.h"
3  using namespace std;
4
5  int main(){
6      Robot r1("Dorakemon", 129.3, 129.3, 0x0000ff, 0, 0, 0);
7      Robot r2 = Robot("Beimax", 250.0, 3.0, 0xffffff, 0, 0, 0);
8      r1 = r1 + r2;
9      r1.sayHello();
10 }
```

Overloading ostream and friend it (1/2)

- 應用 friend 和 operator overloading 以自訂 cout << 要印什麼 (Why friend?)
- 重載 << 運算子函式
- 回傳 type = ostream &
  - 可以連 <<
  - cout << r1 << r2;
- 函式名稱 = operator <<
- 參數有二個
  - ostream &out
  - Robot &r
- Robot r1;
- cout << r1; 時
  - out (形式參數) == cout (真實參數)
  - r(形式參數) = r1(真實參數)

robot.h

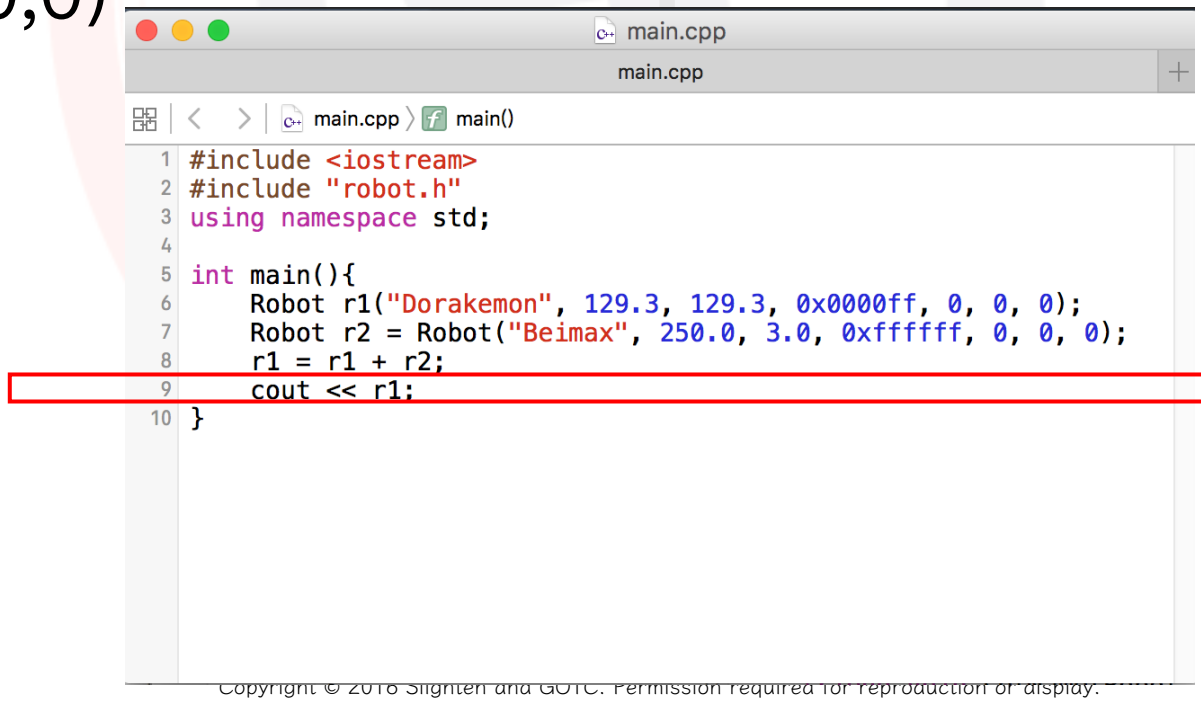
robot.h

robot.h 〉 operator << (ostream &out, Robot &r)

```
1  #ifndef ROBOT_H
2  #define ROBOT_H
3  #include <string>
4  #include <iostream>
5  using namespace std;
6
7  class Robot {
8
9      friend ostream & operator << (ostream &out, Robot &r){
10         r.sayHello();
11         out << "height/weight = " << r.height << "/" << r.weight <<
               endl;
12         out << "at (" << r._x << "," << r._y << ","
13                        << r._z << ")" << endl;
14
15         return out;
16     }
17     friend void setName(Robot &, std::string);
18
19 public:
20     // Default Constructor
21     Robot(std::string , double, double, int,
22           int, int, int);
23     // Default Destructor
24     ~Robot();
25     void sayHello() const;
26     void moveTo(int, int, int);
27     // setters
28     void setRgb(int);
29     void setHeight(double h) {
```

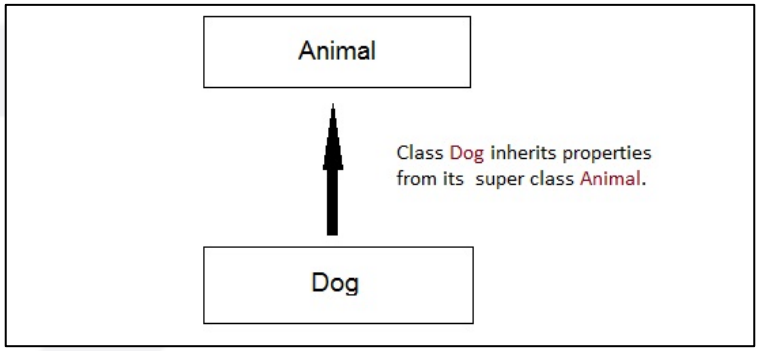
Overloading ostream and friend it (2/2)

- 在 main() 使用 `cout << r1;`
- 輸出：
  Hello! My name is Dorakemon_plus_Beimax_ver_2.0.
  height/weight = 129.3/129.3
  at (0,0,0)



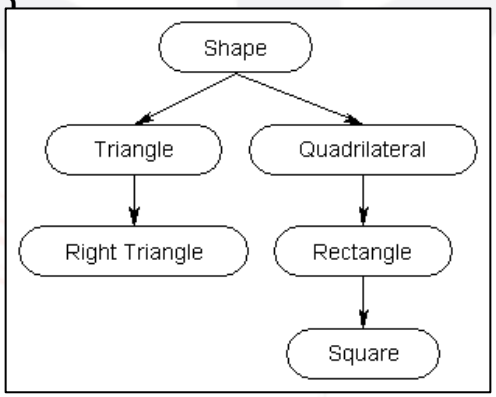
Introduction of Inheritance (繼承)

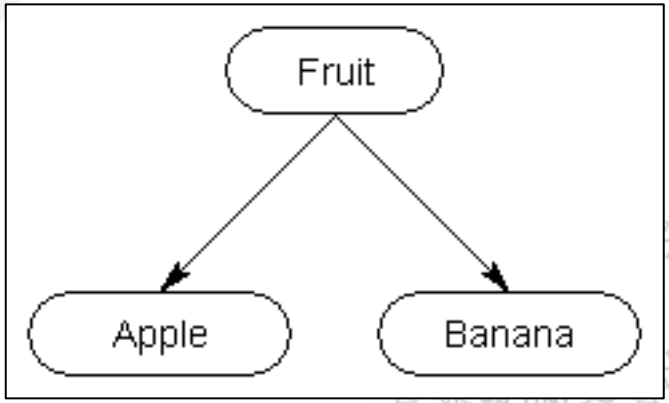
- Public Inheritance
  - (孩)子類別 (child classes) 繼承父(親)類別 (parent classes) 代表可以直接 access 父類別的所有 public members
- 好處：子類別不必自己再寫一次那些父親有的 members，直接用就好
- 當然子類別可能還有自己額外的 members




```
class Animal{
  /* … */
  public:
    void move() { cout << "move!\n"; }
}
class Dog : public Animal{
  /* … */
}
Dog dog;
dog.move();
```

- Dog is Animal's child.
- Anumal is Dog's parent.





Introduction of Polymorphism (多形) (1/2)

- 有繼承後，子類別也可以有自己定義的，並且與父類別 member function 相同名字的 member function，但此時父類別的 function 要加上 `virtual` 關鍵字，此動作稱為 Override (覆寫)
- 父類別的可以只是抽象類別 (Abstract Classes) 或介面 (Interfaces)，也就是說 member function 只有格式 (prototype) 沒有定義實作 (稱作 pure virtual function)，由子類別來自行決定如何實作它
- Virtual function: 子類別有實作就用子類別的，沒有就用父類別的
- Pure virtual function: 父類別不實作，子類別一定要實作，用子類別的 (`virtual function() = 0;`)

```
class Animal{
  public:
    virtual void move() = 0;
}
```

```
class Dog : public Animal{
  public:
    void move()
       { cout << "walk\n"; }
}
```

```
class Bird : public Animal{
  public:
    void move()
       { cout << "fly\n"; }
}
```

Introduction of Polymorphism (多形) (2/2)

```
class Animal{
  public:
    virtual void move() = 0;
}
```

```
class Dog : public Animal{
  public:
    void move()
       { cout << "walk\n"; }
}
```

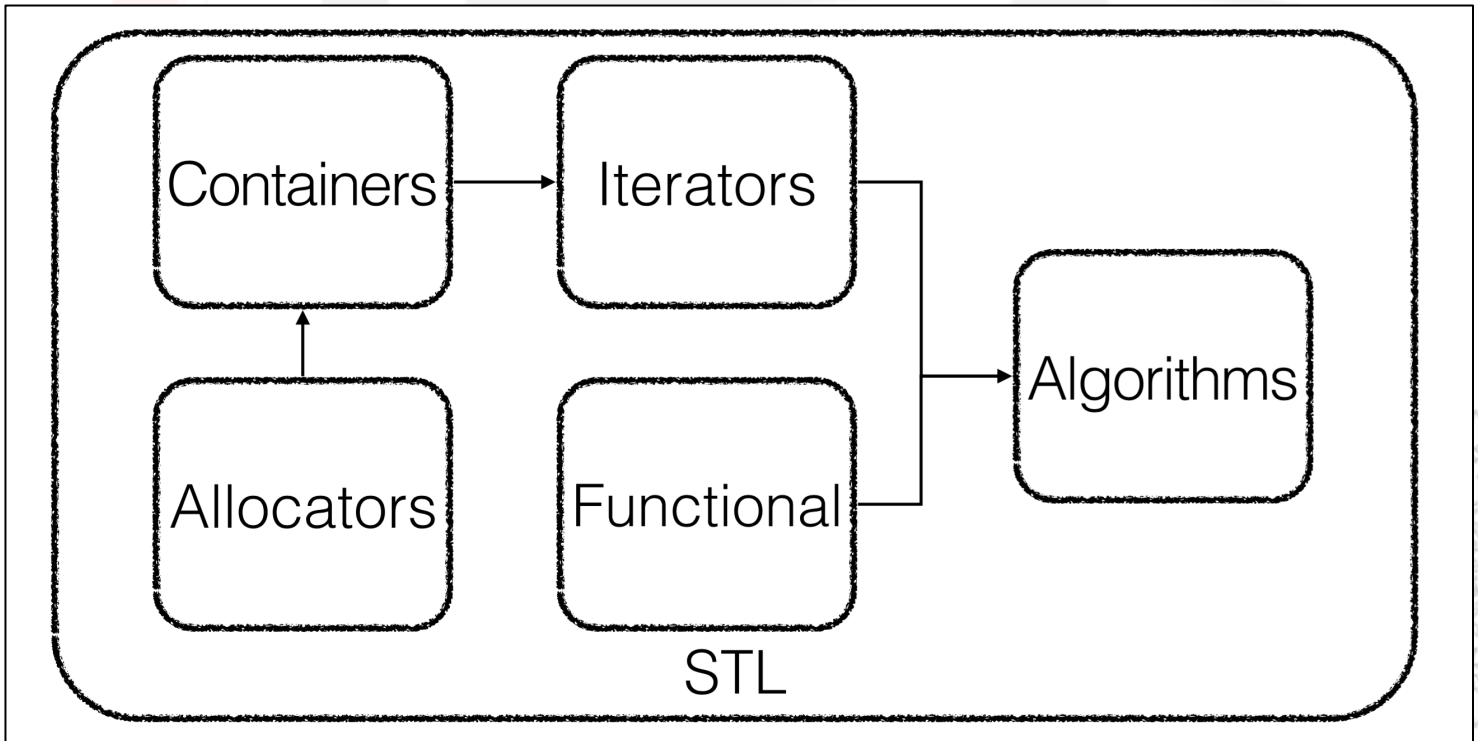
```
class Bird : public Animal{
  public:
    void move()
       { cout << "fly\n"; }
}
```

- Dog dog; Animal ani = dog; ani.move();
- 此時 ani 要 call 哪個 move()?
- **Polymorphism 起手式： Inheritance + Overriding + Upcasting**
- **Polymorphism**: Function 的執行不在編譯時期決定，而延後到執行時期才決定要呼叫哪一個 function。(也就是根據 object 本身的 type 決定)

Using C++ STL

What is STL? (1/2)

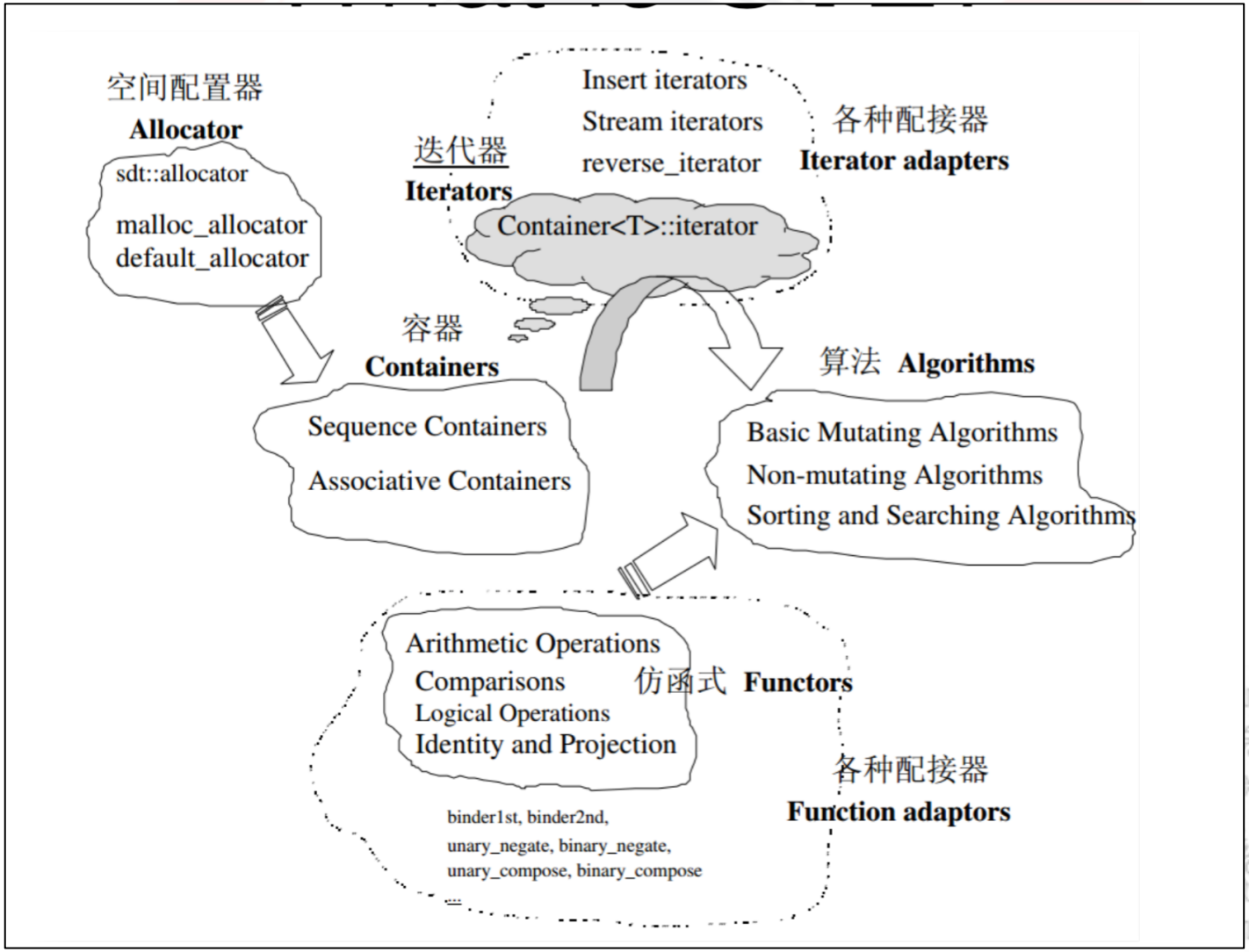
- **STL**: **S**tandard **T**emplate **L**ibrary, 標準模板庫
- 包括 Containers (容器)、 Iterators (迭代器)、 Functional (函式)、 Algorithms (演算法)、 Allocators (空間配置器) …



Why STL?

- **STL 有好用/常用的資料結構與演算法放在那給你用**
  - 好讀/好用/好改
  - 完整測試過/快速
- 所有更詳細資訊請 Read Documents (http://www.cplusplus.com/reference/)
- e.g. google 「C++ vector」 然後找有 C++ Reference – Cplusplus.com

What is STL? (2/2)

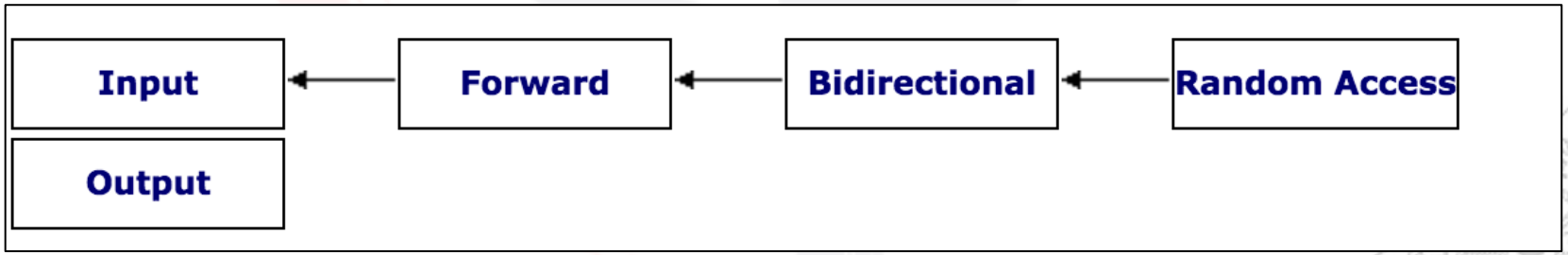


Containers (容器)

- 是 STL 裡最常用/最好用的部分，將資料結構與操作設計成一個模板，讓我們使用
- **Containers** are *templatized* data structures which manage collection of elements.
- 一個 container 一個 `#include`
- 元素的型別是什麼都可以，不論是基本型別 (`int, char, double`)，還是自創的 `class` (`Robot…`)
- **Simple container**
  - pair
- **Container adaptors**
  - stack (FIFO), queue (LIFO), priority_queue
- **Sequential containers** -- ordered collections (有序集)
  - vector, array, basic_string, deque (double-ended queue), list (doubly linked list), slist (singly linked list), forward_list
- **Associative containers** -- sorted collections (排序集)
  - set, multiset, map, multimap
- **Unordered associative containers** — unordered collections (無序集)
  - unordered_set, unordered_multiset, unordered_map, unordered_multimap
- **Others** -- bitmap, bitset, valarray

Iterators (迭代器)

- Iterators are objects that can navigate over elements
- 角色與功能就像 `for (int i = 0; i < 100; i++)` 裡的 `i`
- 每種 container 都有自己的 iterator
- A pair of iterators determines a sequence
  - `someContainerObject.begin()` & `someContainerObject.end()`
  - Move forward (++), get Value (*), compare(==)
- 分五大類



Iterators (迭代器)

- 分

category				properties	valid expressions
all categories				*copy-constructible*, *copy-assignable* and *destructible*	`X b(a);` `b = a;`
				Can be incremented	`++a` `a++`
Random Access	Bidirectional	Forward	Input	Supports equality/inequality comparisons	`a == b` `a != b`
				Can be dereferenced as an *rvalue*	`*a` `a->m`
			Output	Can be dereferenced as an *lvalue* (only for *mutable iterator types*)	`*a = t` `*a++ = t`
				default-constructible	`X a;` `X()`
				Multi-pass: neither dereferencing nor incrementing affects dereferenceability	`{ b=a; *a++;` `*b; }`

				Can be decremented	`--a`
					`a--`
					`*a--`

				Supports arithmetic operators + and –	`a + n`
					`n + a`
					`a - n`
					`a - b`

				Supports inequality comparisons (<, >, <= and >=) between iterators	`a < b`
					`a > b`
					`a <= b`
					`a >= b`

				Supports compound assignment operations += and -=	`a += n` `a -= n`
				Supports offset dereference operator ([])	`a[n]`

Recall: C String vs C++ String

- `std::string str1;`
  - C++ 字串
  - 動態決定大小的字元陣列
  - 長度不限
  - 有很多操作與重載運算子: `+`, `+=`, `==`, `!=`, `length()`, `size()`…
  - `#include <string>`
  - It's not a container since it doesn't have the generic concept
  - Actually, `string` is a specific type of the a container called `basic_string`
  - `typedef basic_string<char> string;`
- `char str2[100];`
  - C 字串
  - 給定大小的字元陣列
  - 長度超過 100 就完了
  - 所有字串操作要 `#include <cstring>`
    - `strlen()`, `strchr()`, `strstr()`, `strcat()`, `strcmp()`…

An Example of Container – vector (1/7)

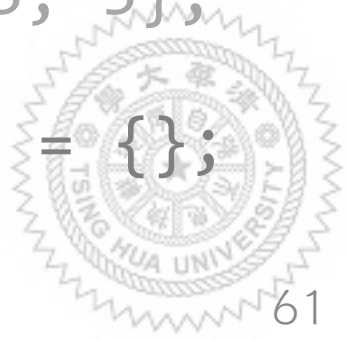
- vector, 向量：基本上就像是一個動態決定大小的陣列
  - 不必一開始給定大小
  - 空間不夠會自動 reallocate 使用者不必擔心
- `#include <vector>`
  `using namespace std;`
- 宣告變數 (創造物件): `ClassType object;`
  - `vector<int> vec;`
    - 一個名叫 `vec` 的向量，元素型別都是 `int`
    - 一開始是空的，大小是 `0`
    - 功能就像整數陣列： `int vec[] = {}; // illegal in C`

An Example of Container – vector (2/7)

- `vector<int> vec;`
- 上面 vector 是一種 class template
- class template 格式
  - `templateName <typeName T>`
  - 由角括號<>包住的 T 又稱作一個 type-parameter，代表該 template 裡元素的 type 變成參數，由使用者宣告時決定
- `vector<int> v1;` ≈ `int v1[] = {};`
- `vector<char> v2;` ≈ `char v2[] = {};`
- `vector<Robot> v3;` ≈ `Robot v3[] = {};`
- `vector<Robot*> v4;` ≈ `Robot* v4[] = {};`
- `vector<vector<int>> v5;` ≈ `int v5[][] = {};`
- This is called generic (泛型) programming

An Example of Container – vector (3/7)

- `vector<int> v1(4, 3); // ≈ int v1[4*] = {3, 3, 3, 3};`
- `vector<int> v2(v1.begin(), v1.end());`
  `// use iterators`
  `≈ int v2[4*]; memcpy(v2, v1, 4);`
  `// mean int v2[4*] = v1;`
  - `#include <cstdlib>`
    `memcpy(a, b, size);`
  - `≈ for (int i = 0; i < size; i++) a[i] = b[i];`
- `vector<int> v3(v2); // same as above`
- `vector<int> v4(2, 5); // int v4[2*] = {5, 5};`
  `v4 = v1; // v4 = {3, 3, 3, 3};`
  `v4 = vector<int>(); // empty vector, v4 = {};`



An Example of Container – vector (4/7)

- A simple example

```
vector<int> vec;
// size == 0, don't do vec[1] = 200, use push_back() instead.
for (int i = 0 ; i < 10; i++) vec.push_back((i+1)*100);

// 三種印出所有元素的方式
// 1. 最基礎的方式
for (int i = 0 ; i < 10; i++) cout << vec[i] << " "; cout << endl;
// 2. 用 iterator
for (vector<int>::iterator it = vec.begin(); it != vec.end(); it++)
    cout << *it << " "; cout << endl;
// 3. Range-based (C++11 以後才有), auto 可改成 int
for (auto i : vec) cout << i << " "; cout << endl;
```

```
100 200 300 400 500 600 700 800 900 1000
100 200 300 400 500 600 700 800 900 1000
100 200 300 400 500 600 700 800 900 1000
```

An Example of Container – vector (5/7)

- A simple example (Cont'd)

```
cout << "vec[3] = " << vec[3] << endl;
cout << "vec.at(4) = " << vec.at(4) << endl;
cout << "vec.front() = " << vec.front() << endl;
cout << "vec.back() = " << vec.back() << endl;

for (int i = 0 ; i < 3; i++) vec.pop_back();
for (int i : vec) cout << i << " "; cout << endl;

vec.clear(); // 清空

// size == 0, don't do vec[15] = 2;
cout << "vec.size() = " << vec.size() << endl;
cout << "vec.capacity() = " << vec.capacity() << endl;

vec.resize(20);
// size == 20, can do vec[15] = 2;
cout << "vec.size() = " << vec.size() << endl;
cout << "vec.capacity() = " << vec.capacity() << endl;
```

```
vec[3] = 400
vec.at(4) = 500
vec.front() = 100
vec.back() = 1000
100 200 300 400 500 600 700
vec.size() = 0
vec.capacity() = 16
vec.size() = 20
vec.capacity() = 32
```

An Example of Container – vector (6/7)

- Insert

```
//10, 10, 10
std::vector<int> myvector (3,10);
std::vector<int>::iterator it;

//20, 10, 10, 10
it = myvector.begin();
it = myvector.insert ( it , 20 );

//30, 30, 20, 10, 10, 10
myvector.insert (it,2,30);

// "it" no longer valid, get a new one:
it = myvector.begin();

//30, 30, 40, 40, 20, 10, 10, 10
std::vector<int> anothervector (2,40);
myvector.insert (it+2,anothervector.begin(),anothervector.end());

//51, 52, 53, 30, 30, 40, 40, 20, 10, 10, 10
int myarray [] = { 51,52,53 };
myvector.insert (myvector.begin(), myarray, myarray+3);
```

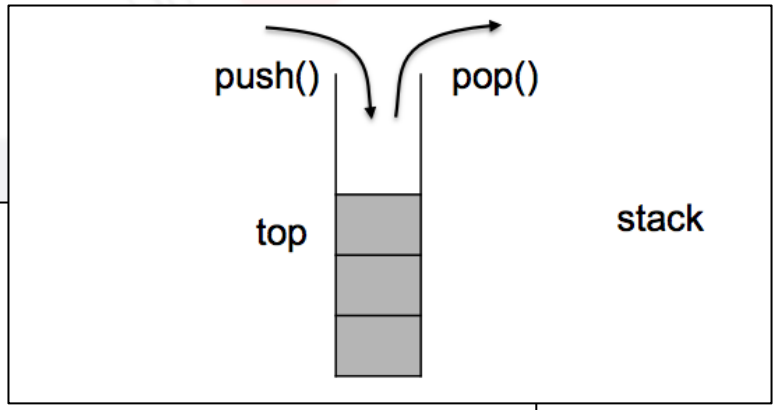
An Example of Container – vector (7/7)

- Erase

```
std::vector<int> myvector;

// set some values (from 1 to 10)
for (int i=1; i<=10; i++) myvector.push_back(i);

// erase the 6th element
myvector.erase (myvector.begin()+5);

// erase the first 3 elements:
myvector.erase (myvector.begin(),myvector.begin()+3);
```

An Example of Container – stack

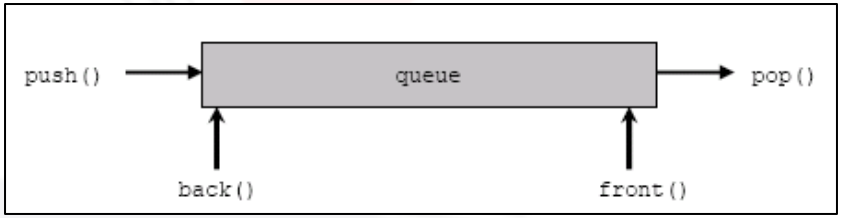
- stack, 堆疊: Last In First Out (LIFO)
- #include <stack>
- 有 push(), pop(), top()




```
std::stack<int> myStack;

//0, 10, 20, 30
for(int i=0;i<4;i++)
    myStack.push(10*i);

//0, 10
for(int i=0;i<2;i++)
    myStack.pop();

//true, 10
if( !myStack.empty() )
    std::cout << myStack.top() << std::endl;
```

An Example of Container – queue

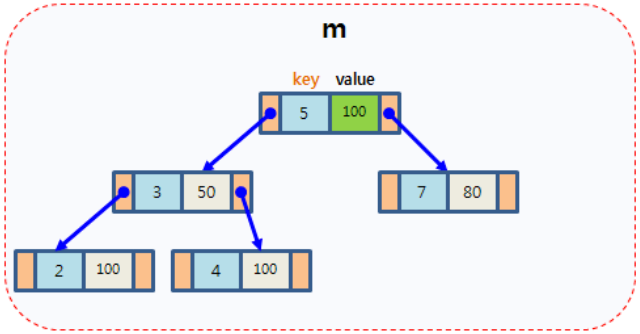
- queue, 佇列：First In First Out (FIFO)
- `#include <queue>`
- 有 `push(), pop(), front(), back()`
- 事實上正確術語為
  - Queue: enqueue, dequeue
  - Stack: push, pop




```
std::queue<int> myQueue;

//0, 10, 20, 30
for(int i=0;i<4;i++)
    myQueue.push(10*i);

//20, 30
for(int i=0;i<2;i++)
    myQueue.pop();

//true, 20
if( !myQueue.empty() )
    std::cout << myQueue.front() << std::endl;
```

An Example of Container – map (1/2)

- map, 映射：A set of *key–value* pairs
- 就像一個高級版的陣列
  - 陣列的 key 型別一定是整數
  - e.g. `double num[4] = {0.1, 0.5, 0.4, 1.8};`
  - `key->value` pairs: `0->0.1, 1->0.5, 2->0.4, 3->1.8`
  - Preferred `key->value` pairs: `"Amy"->98, "Alice"->95, "Bob"->100`
  - 有沒有一種陣列 `arr`
    - `arr["Amy"] = 98, arr["Alice"] = 95, arr["Bob"] = 100`
    - 要用到 map
    - ```
      #include <map>
      map<string, int> arr; // map<keyType, valueType>
      arr["Amy"] = 98;
      arr["Alice"] = 95;
      arr["Bob"] = 100;
      ```
- Search keys by binary search: $O(\log n)$


      ```

An Example of Container – map (2/2)

- ```
  #include <map>
  map<string, int> arr; // map<keyType, valueType>
  arr["Amy"] = 98;
  arr["Alice"] = 95;
  arr["Bob"] = 100;
  ```
- `cout << arr["Amy"] << endl;`
- `if (!arr.count("Colin")) arr["Colin"] = 0;`
- `count(key)` function:
  - 給一個 key 找是否存在此 key 的 key-value pair
  - 有：回傳 `1 (true)`
  - 無：回傳 `0 (false)`
- `map<int, string> m1; // m1[101021120] = "Slighten";`
- 仿二維陣列
  - `map<int, map<int, string>> m2; // m2[101][21120] = "Slighten";`
  ```

Time Complexity

Container	Insertion	Access	Erase	Find	Persistent Iterators
vector / string	Back: O(1) or O(n) Other: O(n)	O(1)	Back: O(1) Other: O(n)	Sorted: O(log n) Other: O(n)	No
deque	Back/Front: O(1) Other: O(n)	O(1)	Back/Front: O(1) Other: O(n)	Sorted: O(log n) Other: O(n)	Pointers only
list / forward_list	Back/Front: O(1) With iterator: O(1) Index: O(n)	Back/Front: O(1) With iterator: O(1) Index: O(n)	Back/Front: O(1) With iterator: O(1) Index: O(n)	O(n)	Yes
set / map	O(log n)	-	O(log n)	O(log n)	Yes
unordered_set / unordered_map	O(1) or O(n)	O(1) or O(n)	O(1) or O(n)	O(1) or O(n)	Pointers only
priority_queue	O(log n)	O(1)	O(log n)	-	-

Limitations of Containers

- Container 不能放任意 object, 以下三種不行
  1. 沒有 copy constructor 的 classes
  2. 沒有定義 operator = 的 classes
  3. 沒有 default constructor 的 classes

```
class Foo
{
private:
    Foo& operator=(const Foo&);
}
```

```
class Foo
{
private:
    Foo( const Foo & );
};
```

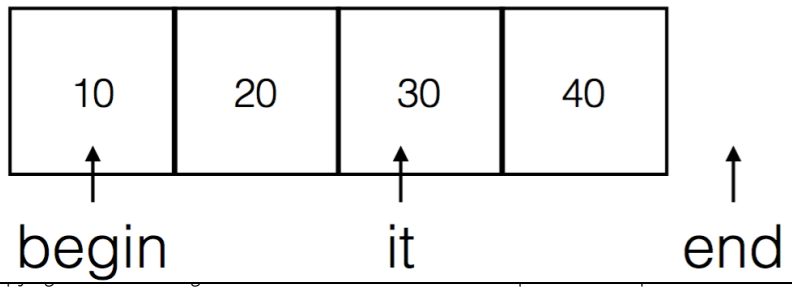
Algorithms (演算法)

- STL 第二好用的東西，提供許多方便的演算法 (函式)，並適用於任何 Container
  - `find(), sort(), min(), max(), nth_element(), shuffle(), rotate(), fill(), replace(), binary_search()....`
- Algorithms use iterators
- `#include <algorithm>`
- Go to C++ reference website and learn how to use them

Algorithms (演算法) -- Find

- #include <algorithm>

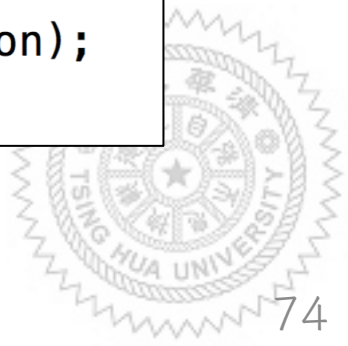
```
int myints[] = { 10, 20, 30, 40 };

// using std::find with vector and iterator:
std::vector<int> myvector ( myints, myints+4 );
std::vector<int>::iterator it;

it = std::find (myvector.begin(), myvector.end(), 30);

if (it != myvector.end())
    std::cout << "found:" << *it << '\n';
    else
        std::cout << "not found\n";
```



Algorithms (演算法) – Sort (1/2)

- #include <algorithm>

```
bool myfunction (int i,int j) { return (i<j); }

////////////////////////////////////////////////////////////

int myints[] = {32,71,12,45,26,80,53,33};
std::vector<int> myvector (myints, myints+8);
// 32 71 12 45 26 80 53 33

// using default comparison (operator <):
std::sort (myvector.begin(), myvector.begin()+4);
//(12 32 45 71) 26 80 53 33

// using function as comp
std::sort (myvector.begin()+4, myvector.end(), myfunction);
// 12 32 45 71 (26 33 53 80)
```



Algorithms (演算法) – Sort (2/2)

- #include <algorithm>

```
struct myclass {
    bool operator() (int i,int j) { return (i<j);}
} myobject;

////////////////////////////////////////////////////
int myints[] = {32,71,12,45,26,80,53,33};
std::vector<int> myvector (myints, myints+8);
// 32 71 12 45 26 80 53 33

// using function as comp
std::sort (myvector.begin()+4, myvector.end(), myobject);
// 32 71 12 45 (26 33 53 80)

// using function as comp
std::sort (myvector.begin(), myvector.begin()+4, std::less<int>() );
// (12 32 45 71)(26 33 53 80)
```

Conclusion

- C++ STL 相當好用，不會用等同不會 C++

- OOP 裡有很多專有名詞 (或稱術語、行話, jargan)，搞清楚概念最重要，重點是「為什麼」
- Override (覆寫), overload (重載), inheritance (繼承), polymorphism (多形), generic (泛型) …

補充：Boost

- Boost (C++ Libraries)：包含很多很實用的東西
  - Image processing
  - Regular expression
  - Unit testing
  - Linear algebra
  - pseudo-andom number generation
  - multithreading
  - …
- 其中有一個 Library 叫作 Boost Graph Library (BGL)，專門處理圖論問題

Applications of C++

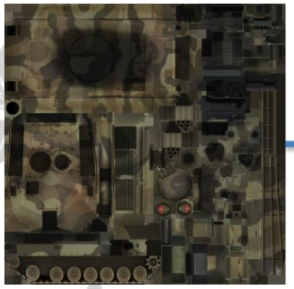
1. C++ OpenGL Programming： 2D/3D 繪圖
2. C++ Ot Programming: PC/Android 平台遊戲製作
3. 8051 Programming (Embedded System)
4. Arduino
5. PHP
6. ….

C++ OpenGL (1/2)

What is OpenGL?

Open Graphics Library

3D Model



Texture

Light =(1.5, 0.6, 0.2)
Material=……

OpenGL

Your OpenGL Application

Rendered Image

Source: Hung-Kuo Chu

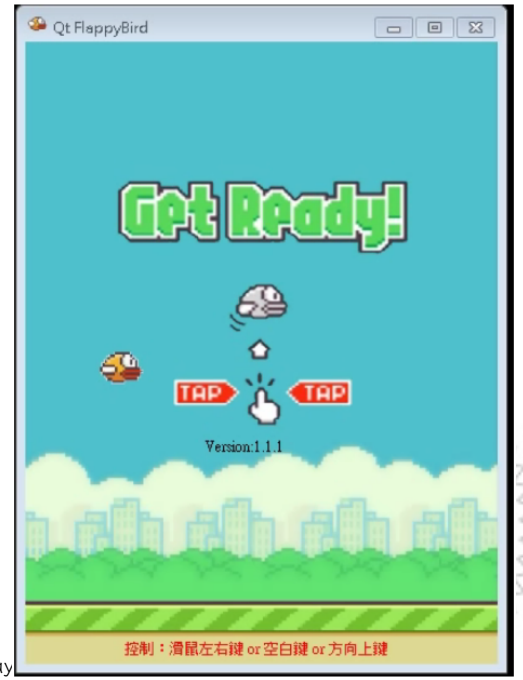
C++ OpenGL (2/2)

- Shader Programming
- Light/Shadow
- Fog/Bloom/Blur/Abstraction…

- Source: Hung-Kuo Chu

C++ Qt

- What is Qt?
  - Qt（發音同cute），是一個跨平台的 C++ 應用程式開發框架，廣泛用於開發GUI(使用者介面圖形) 程式
  - 擁有完善的C++圖形函式庫
  - 支援多種平台。會自動依平台的不同，表現平台特有的圖形介面風格(PC/Android)
  - 知名的Andoird App ー Flappy Bird 就是用 Qt 做的



嵌入式系統重要概念 (1/4)

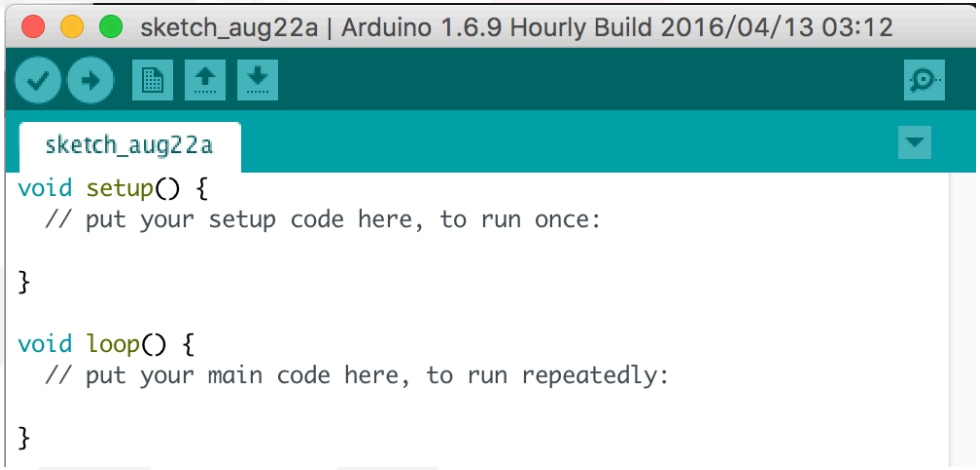
- **Interrupt vs Polling**
  - 一個機器不能 main() 完就 return 0;
  - 它是要打開就開始等使用者操作的
  - Two ways

1. Polling
   1) Waiting: `while (/* no user input */){}`
   2) Doing some Work:
      ```
      while (/* no input */){ /* Do something */ }
      ```
   - Problem: potentially slow response / long latency
2. Interrupt (Better way)
   - Let hardware test flag instead of software
   - Codes written in ISR (Interrupt Service Routine)

嵌入式系統重要概念 (2/4)

- 通常程式碼會長得像

```
main() {
        setup();
        while(1){
                if (xxx) xxx;
                else if (xxx) xxx;
                /* … */
        }
}
```

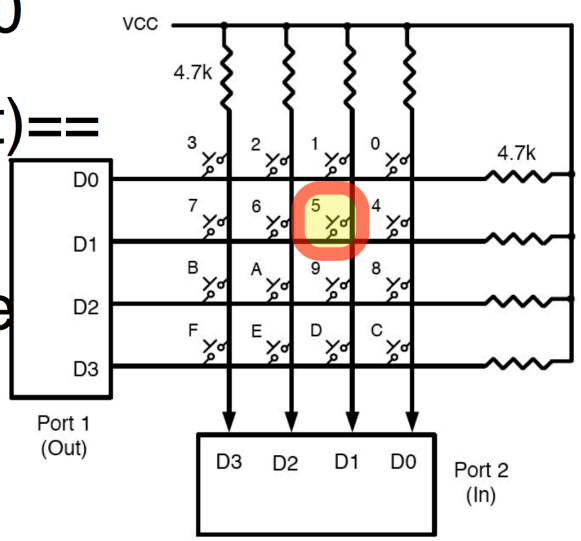



```
94  void Main(void) {
95
96      functionSet();
97      entryModeSet(1, 0); // increment and no shift
98      displayOnOffControl(1, 1, 1); // display on, cursor on and blinking on
99      sendString("EdSim51 LCD Module Simulation");
100     setDdRamAddress(0x40); // set address to start of second line
101     sendString("Based on Hitachi HD44780");
102
103     while (1) {
104         if (ret == 0) {
105             returnHome();
106         }
107         else {
108             if (left == 0 && right == 1) {
109                 cursorOrDisplayShift(1, 0); // shift display left
110             }
111             else if (left == 1 && right == 0) {
112                 cursorOrDisplayShift(1, 1); // shift display right
113             }
114         }
115     }
116 }
```

嵌入式系統重要概念 (3/4)

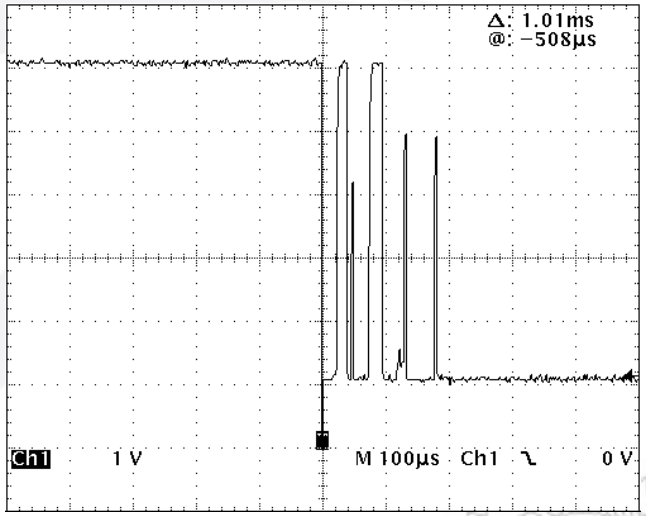
- 怎麼讀 key (*m* rows, *n* cols)?
  - 可以一個鍵一個 pin 腳，讀那個 pin 的值，但會很浪費 pin: 需要 $m \times n$ pins
  - Solution: **Row–Column Scanning**，只需 $m + n$ pins

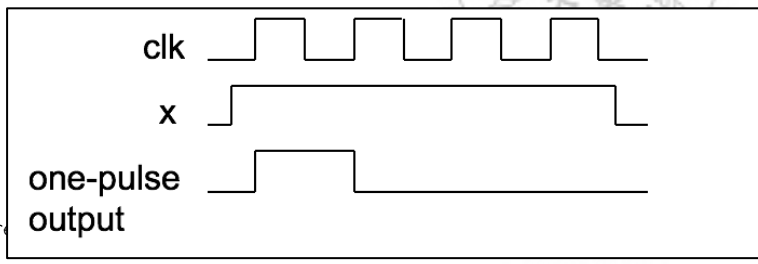
- MCU sets Rows(output)= 0111, 1011, 1101, 1110
- MCU reads Cols(input)== 1111, 1101, 1111, 1111
- 0s yield the coordinate of the pressed button => multi-button press can be detected too!



嵌入式系統重要概念 (4/4)

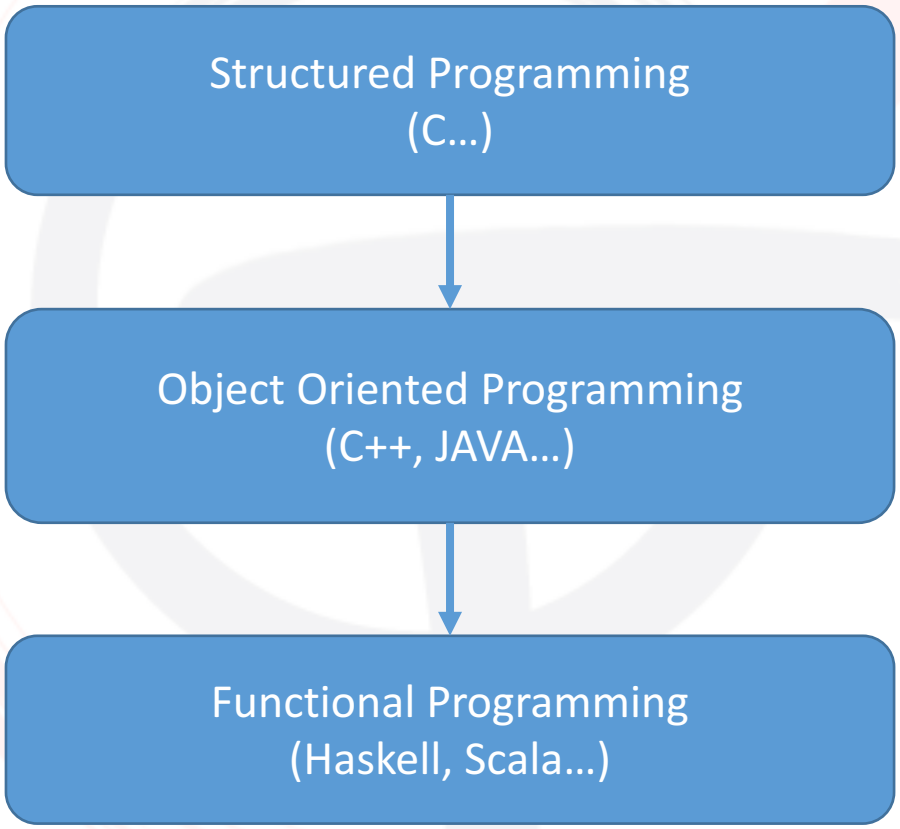
- **Key Bouncing**
  - 當你按按鍵，你以為你只按一下，但實際上硬體會認為你按了很多下 (物理)
  - 程式沒寫好會認為你是在連點按鍵
  - Solutions:
    1. Hardware: Debouncer
       1) SR Latch
       2) Schmitt Trigger
    2. Software: just delay() for ~20 ms
- **One Pulse**
  - 按住不會 auto-fire，當成只按一下
    - 鎖在一個 clock 內





Conclusion

- Programming paradigm (編程範式) 演變



- C/C++ 有很多應用，尤其是 C++，學好這個語言，無往不利
- 會一個語言，往後接觸其他語言都會很方便