【第十一章】

物件與類別

講師: 李根逸 (Ken-Yi Lee), E-mail: feis.tw@gmail.com

課程大綱

C++ 與 C

- **向 C 相容:**
  - C 語言大部分語法在 C++ 中依舊可用，所以我們可以將 C 檔案的附檔名存為 .cpp ，直接當做 C++ 檔案以 C++ 編譯器編譯。
- **物件導向:**
  - C++ 語言提供新的語法支援『物件導向設計』(OOP)
  - C 語言的主要設計方式是將程式分成很多實現不同功能的函式，利用傳入『結構』至不同的『函式』完成設計目的。物件導向設計則是利用『物件』間的互相呼叫完成目的。
    - 那物件跟函式或結構有什麼差 ？
      - 對 C 來說，物件可以想成是具有一群函式 (function / method) 與資料變數 (variable / property) 的結構 (struct)
      - 將資料與對這些資料做處理的函式包裝在一起！

《補充》C++常用新功能 (1)

- **命名空間 (Namespace)**
  - 為了解決函式或類別等名稱重複的問題，C++ 加入了命名空間的概念
    - 例如: 例如替 C 標準函式庫增加 std 命名空間，使得 printf 函式的全名為 std::printf
- **函式重載 (Function overloading)**
  - C++ 讓函式在參數型別/個數不同時可以擁有相同的名字
    - `int abs(int)`
    - `double abs(double)`
- **運算子多載 (Operator overloading)**
  - C++ 可以自訂大部分運算子的運算函式

《補充》C++常用新功能 (2)

- 參考 (Reference)
  - C++ 裡提供參考來減少指標的使用：
    - 原本指標宣告：

```
int grade;
int *ptr_grade = &grade;
*ptr_grade = 100;
```

ptr_grade 存放 grade 的記憶體位址

    - 可改寫成參考宣告：

```
int grade;
int &ref_grade = grade;
ref_grade = 100;
```

ref_grade 代表 grade 本身

    - 兩者都會修改原本的 grade 為 100

只有結構跟函式有什麼不好？

- 參考以下例子：

```
void read(Student *s);
void show(const Student *s);
int main() {
  Student s;
  read(&s);
  show(&s);
  return 0;
}
```

  - 哪些函式可能修改 s 的內容？
  - 有這樣的權力好嗎？

- 設計重點：提供介面但是隱藏資訊

類別定義

- 用 **class** 這個保留字來定義類別內容，類別內容包含『存取限制』、『資料成員宣告』與『函式成員宣告』：
  - 語法：

```
class 類別名稱 {
   存取限制:
      資料型態   資料成員;
      回傳值型態  函式成員([參數型態  參數名稱,]);
   ...
};
```

  - 每個屬性成員跟行為成員都可以有不同的存取限制：
    - public (公開) [任何函式都可存取呼叫], private (私有) [只有函式成員可以存取呼叫] 和 protected (保護)
    - 任何存取限制下所有函式成員都可以存取任意資料成員

類別函式成員定義

- 類別函式成員可以直接在類別定義內宣告或定義：

```
class Student {
    private:
        char _name[20];
        int _grade;
    public:
        void print() const;
        int set(int grade) {
            _grade = grade;
            return _grade;
        }
};
```

這裡的 _ 只是一種命名慣用法，另一種是在 **private** 成員名稱前加上 **m**

共用時類別宣告會放在標頭檔

成員函式宣告

成員函式定義

函式成員後加上 **const** 表示該函式不會修改資料成員

在成員函式內可以直接存取資料成員或其他成員函式

- 也可以用一般函式的方法定義 (需加上類別名稱::)

```
void Student::print() const {
  printf("%s (%f)\n", _name, _grade);
}
```

共用時成員函式定義會放在程式檔

物件與類別的使用

- 宣告好一個包含資料與函式的類別 **(class)** 後，可以直接當做資料型別 **(type)** 來使用。用類別宣告的變數就是一個物件 **(object)**
- 例如：
  - 宣告一個 Student 類別的 my_student 物件：

```
Student my_student;
```

  - 取得物件成員變數：

```
my_student._grade = 100;
```

  - 執行物件成員函式：

```
my_student.set(100);
```

用 . 來存取物件資料成員

注意的是此時因為 **_grade** 存取限制為 **private** 所以會編譯失敗

注意：存取物件指標的成員時要改用 **->**

用 . 來呼叫物件成員函式

《範例》Grade 類別

- 試寫一程式 (grade) 設計一類別 Grade，該類別的資料只能指定為不大於 100 的非負整數

《練習》Time 類別

- 試寫一程式 (time) 設計一個 Time 類別，提供下面的介面：

```
class Time {
  public:
    /* setTime(int, int, int):
       描述:   如果參數合理, 設定時間為 h 時 m 分 s 秒
       回傳值: true 表示設定成功, false 表示參數超出範圍）*/
    bool setTime(int h, int m, int s);

    /* print(): 印出 HH:MM:SS 格式的時間 */
    void print();
};
```

bool 是 C++ 一內建型態，有 true 跟 false 兩種值

類別的建構子與解構子

- **當你配置一個類別物件的時候，該類別定義裡對應的建構子 (constructor) 會被執行，通常用來作資料成員的初始化**
  - 建構子在類別定義中就是一個與類別名稱相同且沒有回傳值資料型態的函式
    - 例如: Student 類別內的 Student 函式
- **當你釋放一個物件的時候，該類別定義裡對應的解構子 (destructor) 會被執行，通常用來回收物件所佔用的資源**
  - 解構子在類別定義中就是一個是類別名稱前加上 ~ 符號且沒有回傳值資料型態的函式
    - 例如: Student 類別內的 ~Student 函式

動態配置與釋放物件

- **靜態配置的方法：**
  - 跟之前一樣，所有型別的變數會自動在宣告時配置記憶體，在結束生命週期時釋放記憶體
  - 以類別為型別的便是在宣告時還會呼叫建構子，在結束生命週期時還會呼叫解構子
- **動態配置的方法：**
  - 在 C 語言裡，我們使用 malloc 與 free 兩種函式來配置與釋放動態記憶體。
  - 在 C++ 語言裡，我們使用 new 與 delete 兩種運算子來配置與釋放物件
    - 配置物件時會配置記憶體與呼叫建構子
    - 釋放物件時會釋放記憶體與呼叫解構子

```
Student *s = new Student;
delete s;
```

《範例》類別的使用

- 請參照下列三個範例檔並了解之間的差異：
  - C 傳統版本: origin.cpp
  - C 函式版本: func.cpp
  - C 結構 (struct) 版本：struct.cpp
  - C++ 類別 (class) 版本：class.cpp
  - C++ 類別多檔版本：project.dev
    - main.cpp：主程式
    - class.hpp：類別定義
    - class.cpp：成員函式定義