【第三講】

基於物件的程式設計

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

★ ★ ★ ★

★ ★ ★ ★

課程大綱

- 基於物件的程式設計
  - 一個類比的範例
- 大數問題
  - 使用結構與函式
  - 使用類別
  - 使用運算子多載

基於物件的程式設計

- 程序式程式設計：
  - Step 1. 思考能解決問題的演算法 (程式流程)
  - Step 2. 找到適當的函式或自行將部分步驟寫成函式
  - Step 3. 實作並完成
- 基於物件的程式設計
  - Step 1. 思考「可能的」流程，與「可能需要的」功能
  - Step 2. 找到或自行實作可能需要的類別
  - Step 3. 思考能解決問題的演算法 (程式流程)
  - Step 4. 實作並完成
- 基本上就是以流程或物件為設計主體的差異

一個類比的範例

- **你要怎麼從台北出發到墾丁？**
  - 自行開車、國道客運、飛機、高鐵、計程車、機車、腳踏車、走路、游泳等等
  - 那你覺得哪個最好？考量的因素是什麼？
- **那你會自己設計並做出一輛車子嗎？**
  - **.....** “你腦筋有問題嗎？ 我只要去買『一輛車子』，然後考個『駕照』就可以啦”
- **那為什麼有人願意去幫我們設計並做出一輛車子？**
  - 打發時間、人生志業或市場需求有錢賺
- **我們為什麼要買車子？**
  - 多功能性！上班、代步、出遊或約會都很方便喔
  - 高彈性！想去哪就去哪～

【練習】 數字計算

- 從頭到尾，寫一程式使用 **cin** 與 **cout** 從鍵盤讀入兩個整數 (使用 **int** 型態) 後，印出相乘的結果：

```
請輸入第一個整數： 10000
請輸入第二個整數： 10000
10000 * 10000 = 100000000
```

  - 999999999 * 123456789 == 12345678876543211

[練習] ex3A.cpp

大數問題

- 因為 **int** 可表示的範圍有限，當我們要表示超過 **int** 可表示的範圍時要怎麼辦？
  - 讓每個 int 只表示一個位數就好
  - 使用一個 int 陣列:

	0	0	0	0	0	0	0	0	0	0	0	9	9	9	9	9	9	9	9	9
*	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9

0	0	1	2	3	4	5	6	7	8	8	8	7	6	5	4	3	2	1	1

【範例】BitInt 結構與函式

```
struct BigInt {
 int data[20];
};

const BigInt Create(unsigned int v);

const BigInt Multiply(const BigInt &lhs,
                      const BigInt &rhs);

const BigInt Add(const BigInt &lhs,
                 const BigInt &rhs);

void Print(const BigInt &r);
```

為了簡化所以我們不處理負數

只使用結構與函式的缺點？

[範例] bigint.cpp

【練習】BitInt 結構與函式

- 試利用上述的 **BigInt** 結構與相關函式寫出一程式，算出下列運算式的結果後印出：

```
(1234567 + 9999999 * 7654321) * 1234567 == ?
```

[練習] ex3B.cpp

【範例】BitInt 類別

```
class BigInt {
 public:
  BigInt();                          // [預設建構式] (為什麼需要?)
  BigInt(unsigned int v);            // [建構式] 初始化為 v
  void Print() const;                // 印出值

  const BigInt Multiply(const BigInt &rhs) const;
  const BigInt Add(const BigInt &rhs) const;

 private:
  int data_[20];                     // 我們想保護資料 (實作細節)
};
```

為什麼 Multiply 和 Add 要放在類別裡面？

為什麼要加 const ?

[範例] classbigint.cpp

this 關鍵字

- 在類別成員函式裡，this 是一個指向物件本身的指標。如果該類別的名稱是 Grade，則 this 型態為 Grade *

```
class Grade {
 public:
  void Set(int v) {
    data_ = v;
  }

  int Get() const {
    return data_;
  }

 private:
  int data_;
};
```

相等於

```
class Grade {
 public:
  void Set(int v) {
    this->data_ = v;
  }

  int Get() const {
    return this->data_;
  }

 private:
  int data_;
};
```

【練習】BitInt 類別

- 試利用上述的 **BigInt** 類別寫出一程式，算出下列運算式的結果後印出：

```
(1234567 + 9999999 * 7654321) * 1234567 == ?
```

[練習] ex3C.cpp

【範例】BitInt 運算子多載

```
class BigInt {
 public:
  BigInt();                          // [預設建構式] (為什麼需要?)
  BigInt(unsigned int v);            // [建構式] 初始化為 v
  void Print() const;                // 印出值

  const BigInt Multiply(const BigInt &rhs) const;
  const BigInt Add(const BigInt &rhs) const;

  const BigInt operator*(const BigInt &rhs) const;
  const BigInt operator+(const BigInt &rhs) const;

 private:
  int data_[20];                     // 我們想把資料保護起來
};
std::ostream &operator<<(std::ostream &lhs,
                         const BigInt &rhs);
```

多載 operator<<(...) 讓我們可以使用 cout 印 BigInt

[範例] opbigint.cpp

再論運算子多載

```
Grade a = 10, b = 20;
Grade c = a + b;
```

- 此時 **a + b** 運算時會試著呼叫什麼？
  - 有一個以上的運算元是類別型態所以會試著呼叫 **operator+(a, b);**
  - 但是因為 **a** 是類別型態，所以也會試著呼叫 **a.operator+(b);**
  - 而且後者的優先順序比較高！
- 要多載運算子的方式要兩種
  - 寫一般函式或者第一個參數型態裡的成員函式

【練習】BitInt 運算子多載

- 試利用上述的 **BigInt** 類別寫出一程式，算出下列運算式的結果後印出：

```
(1234567 + 9999999 * 7654321) * 1234567 == ?
```

[練習] ex3D.cpp