【第五講】

泛型程式設計

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

★ ★ ★ ★

★ ★ ★ ★

課程大綱

- 模版 (`template`) 簡介
  - 實作多個相似內容的函式
  - 函式模版
  - 類別模版
- 自製陣列模版
- `static` 的用法
  - 修飾全域變數或一般函式
  - 修飾區域變數
  - 修飾類別成員
- C++ STL 的陣列模版 (`std::vector`)

實作多個相似內容的函式 [1]

- 寫 C/C++ 程式時常常會遇到許多實作內容相似卻不完全一樣的函式。在以前，我們會將這些函式不一樣的部份用函式參數的方式來傳遞，進而寫成同一個函式：
  - 例如：

```
void Set0(int &x) {              // Set0(a);
  x = 0;                         // Set5(a);
}
void Set5(int &x) {
  x = 5;
}
```

這樣設計的好處我們只要寫一次函式定義，比較容易統一相似的部分。缺點是每次呼叫的時候都可能需要複製一次 v

  - 可以設計成：

```
void Set(int &x, int v) {        // Set(a, 3);
  x = v;                         // Set(a, 5);
}
```

實作多個相似內容的函式 [2]

- 但是依照之前的作法，相似內容函式間不同的地方必須要能當做參數傳入，所以會有些限制：

```
void Set(double &x, double v) {
  x = v;                              // Set(a, 5);
}
void Set(int &x, int v) {
  x = v;                              // Set(b, 5);
}
```

  - 要怎麼把這兩個函式寫在一起？
    - 這兩個函式不一樣的地方是參數的『型態』而不是參數的『值』

模版 [1]

- **template (模版) 是進行泛型程式設計 (generic programming) 的主要語法：**
  - 使用模版可以定義一組相似內容的類別或函式
    - 在 C 裡面可能使用 #define 來模仿這個功能 (搜尋取代)
    - 宣告或定義語法：

```
template<模版參數型態 模版參數名稱, ...>
函式宣告定義或類別宣告定義
```

    - 例如，定義函式模版：

```
template<typename T>
void Set(T &x, T v) {
  x = v;
}
```

      - 呼叫模版產生的函式：

```
Set<int>(a, 5);
Set<double>(b, 5.5);
```

typename 表示這裡的 T 是個型態名稱。語法上這裡也可以使用 class，意義相同。

這裡的 T 是我們自己取的模版參數名稱

[範例] template.cpp

模版 [2]

- 函式模版本身不是函式，而是函式的模版、範本。在編譯的時候，編譯器視程式碼中使用的情況而自動依據模版產生需要的函式：

```
template<typename T>
void Swap(T &a, T &b) {
  T t = a;
  a = b;
  b = t;
}
```

Swap 是函式模版，而 Swap<int> 或 Swap<double> 等就是依據這個模組產生的函式們

在 <algorithm> 內有 std::swap 這個函式模版

  - 呼叫函式時，如果可以從函式參數直接判斷模版參數值的話，可以省略模版參數不用給：

```
int a = 3, b = 5;
double c = 3.5;
Swap<int>(a, b); // 會自動產生一個叫 Swap<int> 的函式
Swap<>(a, b);    // 我們可以從 a, b 的型態判斷 Type
Swap(a, b);      // 這也可以
Swap(a, c);      // [編譯錯誤]
```

[範例] swap.cpp

【練習】求最大值的函式

- 試著用模組寫一個可以求各種資料類型的最大值函式
  - 請修改原本的 Max 函式成為一個函式『模版』
  - 可以使用這個函式模版去計算各種不同資料型態的最大值

[練習] ex5A.cpp

類別模版

- 我們也可以定義類別的模版：

```
template<typename T>
class Grade {
 public:
  Grade();
  Grade(const T &v);
  void Set(const T &v);
  const T &Get() const;
  const Grade<T> operator+(const Grade<T> &rhs) const;

 private:
  T data_;
};
```

為什麼我們使用 const T & 而不是 T 作為參數或回傳值型態？

Grade<int> 是個用 int 型態儲存成績的類別。而 Grade<double> 是個用 double 型態儲存成績的類別。

在定義類別模版的成員函式時，很多時候可以省略模版參數。例如 Grade<T> 可寫成 Grade。但為了避免困惑，我們這門課大部分情況都不省略。

【特別】建構式名稱還是 Grade 而不是 Grade<T>

[範例] grade.cpp

【範例】自製陣列模版 [1]

```
template<typename ElemType>
class Array {
 public:
  Array();
  Array(const Array<ElemType> &rhs);
  explicit Array(int n);
  ~Array();

  int Size() const;
  ElemType &At(int i);
  const ElemType &At(int i);

  ElemType &operator[](int i);
  const ElemType &operator[](int i) const;

  Array<ElemType> &operator=(const Array<ElemType> &rhs);
};
```

使用模版可以更有延伸性

[範例] array_1.cpp

【範例】自製陣列模版 [2]

- 如果陣列的大小在編譯期就可以決定的話，我們可以將陣列大小也設定為一個模版參數，可以簡化程式碼和可能增加效率：

```
template<typename ElemType, int kNumOfElems>
class Array {
 public:
  int Size() const;
  ElemType &At(int i);
  const ElemType &At(int i); const;
  ElemType &operator[](int i);
  const ElemType &operator[](int i) const;

 private:
  ElemType data[kNumOfElems];
};
```

大三法則 (解構式、複製建構式和指定運算子) 到哪去了？

Array<int, 10> a; a 是一個大小為 10 的 int 陣列物件

[範例] array_2.cpp

static 的用法 [1]

- static 可以修飾全域變數或函式的宣告或定義：
  - 該全域變數或函式只能在同個檔案內使用
    - 在 C++ 中，這功能可以使用匿名命名空間取代

```
int a;            // 在所有檔案都可以使用
static int b;     // 就只有這個檔案可以使用
namespace {
  int c;          // 就只有這個檔案可以使用
};

int main() {
  cout << a << endl
       << b << endl
       << c << endl;
  return 0;
}
```

static 的用法 [2]

- static 可以修飾區域變數的定義：
  - 該區域變數只會在程式中產生與死亡一次！
  - 在第一次定義的時候初始化，在程式結束的時候死亡。

```
int &static_var() {
  static int a = 3;    // 在整個程式中這只會做一次
  return ++a;
}

int main() {
  for (int i = 0; i < 10; ++i) {
    cout << static_var() << endl;
  }
 return 0;
}
```

【思考】這裡可以回傳參考嗎？

[範例] static_1.cpp

static 的用法 [3]

- static 可以修飾類別內的成員宣告：
  - 該成員變數或成員函式就成為類別變數或類別函式，屬於該類別而不是個別物件，可以直接使用類別名稱去存取 (類別名稱::變數或函式名稱)。
    - 相對地，類別函式內部不能存取一般物件的成員變數或成員函式

```
class Grade {
 public:
   static const int kMaxGrade = 100;
   static const int kMinGrade = 0;
   int data;
};
int main() {
   cout << Grade::kMaxGrade << " "
        << Grade::kMinGrade << endl;
   cout << Grade::data << endl;  // [編譯錯誤]
   return 0;
}
```

我們不用產生任何物件就可以讀取類別變數或執行類別函式

[範例] static_2.cpp

【範例】自製陣列模版 [3]

- 成員函式的運作不牽涉到物件的成員變數或其他成員函式，可以宣告其為 **static**，也就是類別函式：

```
template<typename ElemType, int kNumOfElems>
class Array {
 public:
  static int Size();
  ElemType &At(int i);
  const ElemType &At(int i) const;
  ElemType &operator[](int i);
  const ElemType &operator[](int i) const;

 private:
  ElemType data[kNumOfElems];
};
```

Array<int, 10>::Size() 是 10

[範例] array_3.cpp

【練習】可變大小陣列的模版

- 試著做一個可變大小的陣列模版：

```
template<typename ElemType>
class Vector {
 public:
  Vector();
  explicit Vector(int n);
  Vector(const Vector<ElemType> &rhs);
  ...
};
```

[練習] ex5B.cpp

【範例】使用 STL 的陣列

- 在 **C++ STL** 內有 **std::vector<Type>** ，是可變大小陣列的模版
  - 需要 **#include <vector>**
  - 請試著練習查看說明文件
  - **std::vector<Type>** 的提供的操作:
    - 快速存取任意位置的元素 (**operator[]**)
    - 快速新增或刪除元素於尾端 (**push_back, pop_back**)
    - 可任意改變大小 (**resize**), 但可能需重新配置記憶體
    - 可以清除全部的陣列內容 (**clear**)
    - 事先保留足夠的空間 (**reserve**) 可以避免重新配置記憶體
    - 新增 (**insert**) 或刪除 (**erase**) 任意位置的元素比較慢
      - 這兩個操作需要用到迭代器 (**iterator**)，我們在後面一章節再解釋

[範例] stl.cpp

【練習】通訊錄

- 使用 **std::vector<Type>** 來完成一個通訊錄程式
  - 使用 **std::string** 儲存名字與電話號碼

```
struct Record {
  string name;      // 姓名
  string mobile;    // 電話號碼
};
```

[練習] ex5C.cpp