C++程式設計：模板（Template）

目錄

1. 基本概念

你寫了一個 Student 類別處理成績（int），但之後又要處理名字（string）、身高（float）…

C++ 的「模板」就是為了解決這種問題誕生的

模板 = 幫類別或函式「保留型別」不決定，之後再決定用什麼型別使用。

1.1 沒有模板的做法

假設我要寫一個 box 的 class，裡面只能放一個變數

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

class IntBox {
public:
    int value;
};

class FloatBox {
public:
    int value;
};
```

```
class StringBox {
public:
    int value;
};

int main(){
    IntBox intBox;
    FloatBox floatBox;
    StringBox stringBox;
    return 0;
}
/* 要一次寫很多類別... */
```

1.2 使用模板的做法

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

template <typename T>
class Box {
public:
    T value;
};

int main(){
    Box<int> intBox;        // 裝 int
    Box<float> floatBox;    // 裝 float
    Box<string> strBox;     // 裝 string
    return 0;
}
```

2. 類別模板

2.1 範例：寫一個「Box」類別，可以裝任何型別的資料

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

template <typename T>
class Box{
private:
```

```
    T item;
public:
    void set(T value) {
        item = value;
    }
    T get() {
        return item;
    }
};

int main(){
    Box<int> intBox;
    intBox.set(42);
    cout << "整數盒子：" << intBox.get() << endl;

    Box<string> strBox;
    strBox.set("Hello, 模板!");
    cout << "字串盒子：" << strBox.get() << endl;
}
```

2.2 範例：實作 pair

像是座標 (x, y)，可以是 (int, int)、(string, int)、(string, string)

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

template <typename A, typename B>
class Pair{
    A first;
    B second;
public:
    void set(A a, B b) {
        first = a;
        second = b;
    }
    void show() {
        cout << "第一個: " << first << ", 第二個: " << second << endl;
    }
};
```

```
int main(){
    Pair<int, string> student;
    student.set(1001, "小明");
    student.show();

    Pair<string, float> product;
    product.set("可樂", 35.5);
    product.show();
    return 0;
}
```

2.3 範例：ArrayPrinter 類別（列印陣列的所有元素）

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

template <typename T>
class ArrayPrinter{
public:
    void print(T arr[], int size) {
        for (int i = 0; i < size; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};


int main(){
    ArrayPrinter<int> printer1;
    int arr1[] = {1, 2, 3, 4, 5};
    printer1.print(arr1, 5);

    ArrayPrinter<string> printer2;
    string arr2[] = {"蘋果", "香蕉", "葡萄"};
    printer2.print(arr2, 3);
    return 0;
}
```

2.4 範例：例用模板實作 stack

```
// #pragma once
```

```cpp
#include <memory>
#include <iostream>
using namespace std;

// STEP01：實作 ListNode 模板
template <class T>
class ListNode {
private:
    T key;
    shared_ptr<ListNode<T>> next;
public:
    ListNode(T k, shared_ptr<ListNode<T>> n = nullptr){
        key = k;
        next = n;
    }
    T getKey() {
        return key;
    }

    shared_ptr<ListNode<T>> getNext() {
        return next;
    }
};

// STEP02：實作 LinkedList 模板
template <class T>
class LinkedList {
public:
    shared_ptr<ListNode<T>> head;
    LinkedList() : head(nullptr) {}

    shared_ptr<ListNode<T>> getHead() {
        return head;
    }

    void insertHead(T key) {
        auto node = make_shared<ListNode<T>>(key, head); // 內建函數
        head = node;
    }
```

```
    T deleteHead() {
        if (!head) throw runtime_error("Stack underflow!");
        T key = head->getKey();
        head = head->getNext();
        return key;
    }
};

// STEP03：實作 Stack 模板
template <class T>
class Stack : public LinkedList<T> {
public:
    bool empty() {
        return this->head == nullptr;
    }

    void push(T k) {
        this->insertHead(k);
    }

    T pop() {
        return this->deleteHead();
    }
};


int main() {
    Stack<int> s;
    s.push(10);
    s.push(20);
    s.push(30);

    while (!s.empty()) {
        cout << s.pop() << endl;
    }
    return 0;
}
```

3. 函式模板（function template）

優點：

- 重複利用：不用重寫多個類型版本的函式
- 自動推導型別：傳入參數，編譯器自動判斷
- 配合 STL：像 sort, max, swap 等都是函式模板
- 可與類別模板一起用：如：Stack<T> 配上 print<T> 函式模板

```
template <typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}

// 使用方式
cout << max(10, 20) << endl;       // int
cout << max(3.14, 2.72) << endl;   // double
cout << max('a', 'z') << endl;     // char
```

3.1 範例：最大值 max

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

template <typename T>
T myMax(T a, T b) {
    return (a > b) ? a : b;
}

int main(){
    cout << myMax(10, 20) << endl;          // 輸出：20
    cout << myMax(3.14, 2.71) << endl;      // 輸出：3.14
    cout << myMax('a', 'z') << endl;        // 輸出：z
}
```

3.2 範例：寫一個泛型的 swap

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

template <typename T>
void mySwap(T& a, T& b) {
    T temp = a;
```

```
    a = b;
    b = temp;
}

int main(){
    int x = 10, y = 20;
    string a = "1", b = "2";
    mySwap(x, y);
    mySwap(a, b);
    cout << x << " " << y << endl;
    cout << a << " " << b << endl;
}
```

3.3 範例：陣列總和 sumArray

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

template <typename T>
T sumArray(T arr[], int size) {
    T sum = 0;
    for (int i = 0; i < size; ++i)
        sum += arr[i];
    return sum;
}

int main(){
    int a[] = {1, 2, 3};
    double b[] = {1.1, 2.2, 3.3};
    cout << sumArray(a, 3) << endl;    // 輸出：6
    cout << sumArray(b, 3) << endl;    // 輸出：6.6
}
```

3.4 範例：找陣列最大值（泛型 + 指標）

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

template <typename T>
T findMax(T* arr, int size) {
    T max = arr[0];
    for (int i = 1; i < size; ++i)
```

```
        if (arr[i] > max)
            max = arr[i];
    return max;
}


int main(){
    float arr[] = {1.5, 3.2, 2.7};
    cout << findMax(arr, 3) << endl;   // 輸出：3.2
}
```