

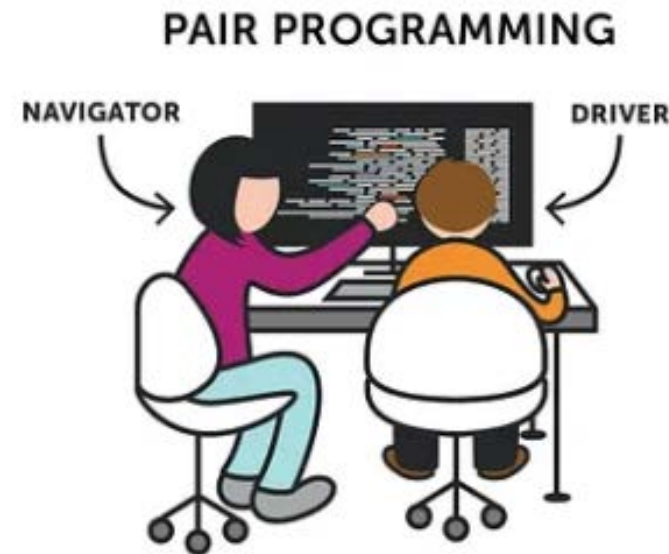
Object Oriented Programming

Code Section: Essential STL

Sun Chin-Yu (孫勤昱)

cysun@ntut.edu.tw

2023/10/06



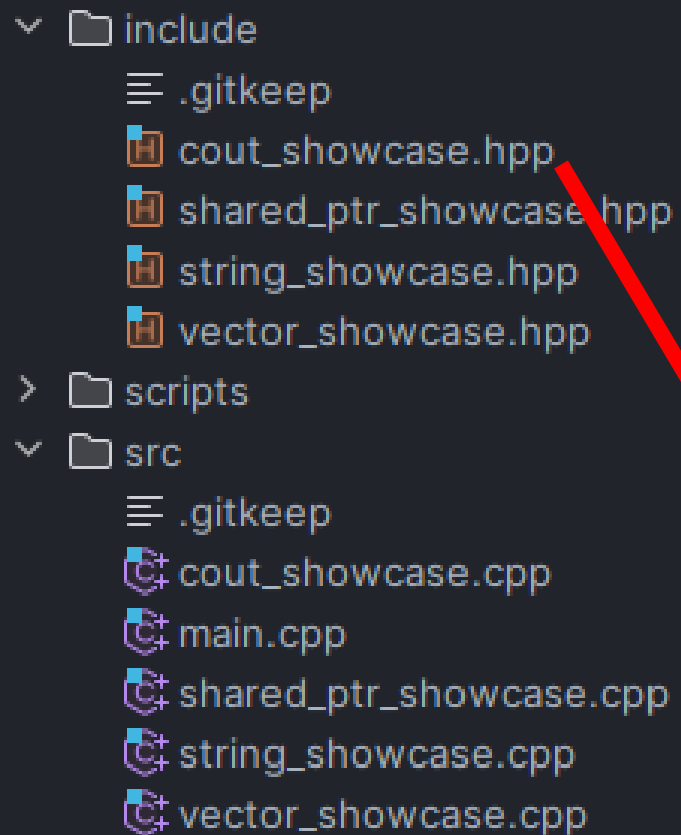
Agenda

- 給定一個main.cpp
 1. 實作cout_showcase();
 2. 實作string_showcase();
 3. 實作vector_showcase();
 4. 實作shared_ptr_showcase();

```
1  #include "cout_showcase.hpp"
2  #include "shared_ptr_showcase.hpp"
3  #include "string_showcase.hpp"
4  #include "vector_showcase.hpp"
5
6  ► int main() {
7      cout_showcase();           //1
8      string_showcase();         //2
9      vector_showcase();         //3
10     shared_ptr_showcase();     //4
11 }
```

程式碼架構

- 宣告放.hpp
 - Include guards
 - 將宣告放在head file中
- 實作放.cpp



```
include
├── .gitkeep
├── cout_showcase.hpp
├── shared_ptr_showcase.hpp
├── string_showcase.hpp
├── vector_showcase.hpp
├── scripts
└── src
    ├── .gitkeep
    ├── cout_showcase.cpp
    ├── main.cpp
    ├── shared_ptr_showcase.cpp
    ├── string_showcase.cpp
    └── vector_showcase.cpp
```

```
1  #ifndef COUT_SHOWCASE_HPP
2  #define COUT_SHOWCASE_HPP
3
4  void cout_showcase();
5
6  #endif
```

目標(一)

- 展示了如何使用C++的std::cout來輸出不同類型的資料和字串
 - 整數
 - 浮點數
 - 字串處理
 - 不同型態混合使用與迴圈
- Output畫面

```
This is an int: 5  
This is a float: 8.32  
This is a string: Never gonna give you up  
Current number: 0 Current number: 1 Current number: 2  
  
Process finished with exit code 0
```

cout_showcase.hpp

- 處理include
 - 需要cout_showcase.hpp
 - iostream
- 完成cout_showcase實作

```
1  #include "cout_showcase.hpp"
2  #include <iostream>
3
4  void cout_showcase() {
5      std::cout << "This is an int: " << 5 << "\n";
6
7      std::cout << "This is a float: " << 8.32F << "\n";
8
9      std::cout << "This is a string: "
10         << "Never gonna give you up"
11         << "\n";
12
13     for (int i = 0; i < 3; i++) {
14         std::cout << "Current number: " << i << " ";
15     }
16     std::cout << "\n";
17 }
```


目標(二)

- 展示了如何在C++中使用C-style字符串和std::string
 - C-style字串操作
 - C++ std::string字串操作



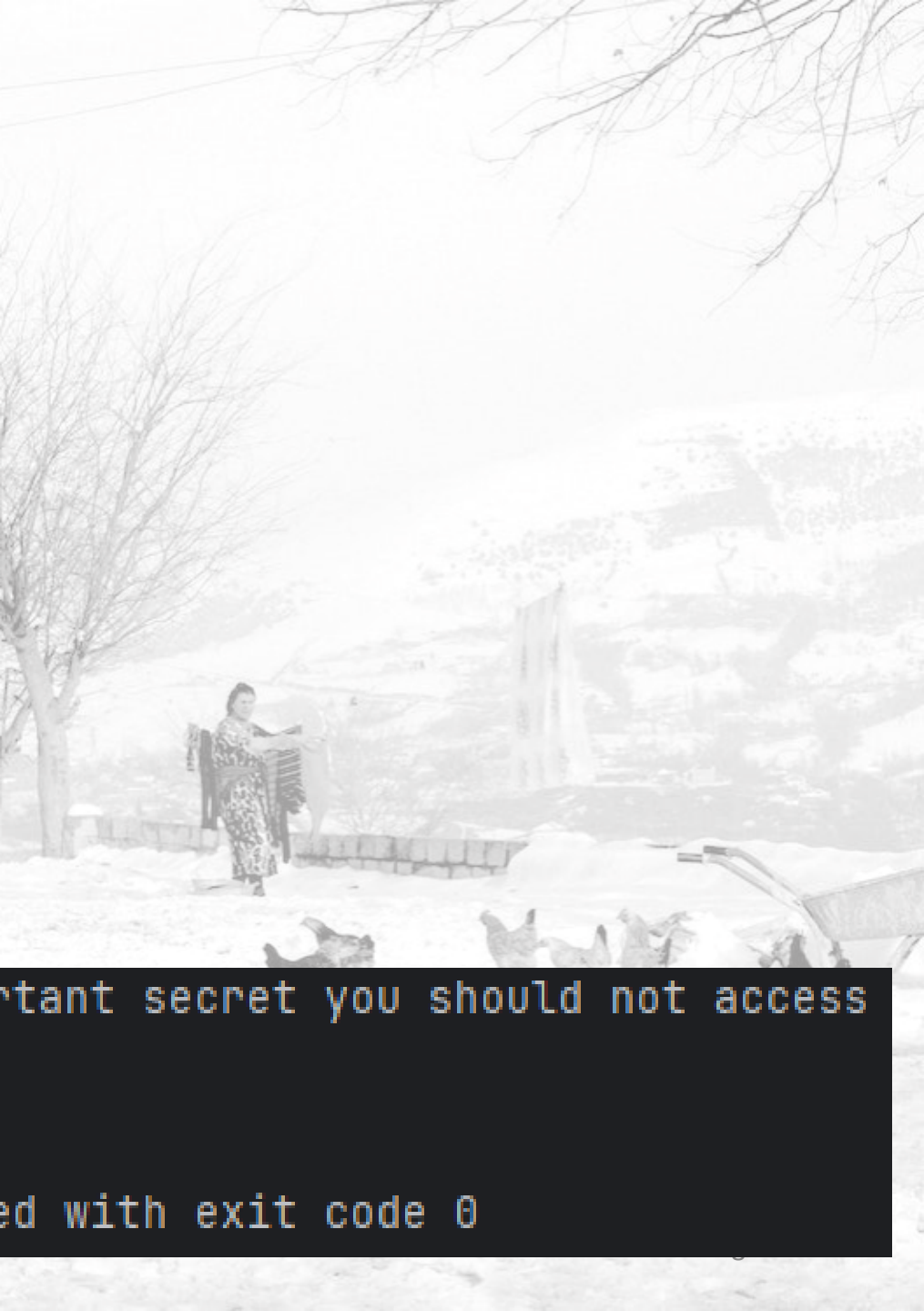
string_showcase.hpp

- 處理c_string()
- 處理cpp_string()
- 組合所有部分

```
1  #include "string_showcase.hpp"
2  #include <cstring>
3  #include <iostream>
4  #include <string>
5
6  void c_string() {
7      char *str = (char *)malloc( Size: 100);
8      strcpy( Dest: str, Source: "Some very important secret you should not access");
9      strcpy( Dest: str, Source: "Hello");
10     str[5] = '!';
11     printf( format: "%s\n", str);
12     free( Memory: str);
13 }
14
15 void cpp_string() {
16     std::string str = "Some very important secret you should not access";
17     str = "Hello";
18     str += "!";
19     std::cout << str << "\n";
20 }
21
22 void string_showcase() {
23     c_string();
24     cpp_string();
25 }
```

C vs. C++ 比較

- 記憶體管理
 - 在c_string函數中，使用了動態記憶體分配 (malloc) 來獲得100 bytes，並在結束時釋放 (free)
 - 在cpp_string函數中，使用std::string則無需明確管理記憶體，因為std::string會自動處理這部分
- 字串操作
 - 在c_string函數中，使用strcpy函數進行字串複製操作，並直接修改字串中的字符以添加一個驚嘆號
 - 在cpp_string函數中，展示了如何使用賦值和+=運算符進行C++字串操作
- 安全性



```
Hello!ery important secret you should not access  
Hello!
```

```
Process finished with exit code 0
```


目標(三): Vector

- 展示了如何使用C++的std::vector來存儲和操作字符串列表
 - Tradition for loop
 - Range-based for loop (C++11)
- 實作步驟
 - 將C++, JAVA, Python, Golang寫進去vector內
 - 使用Tradition for loop印出來
 - 在尾部增加一個新元素Rust
 - 使用Range-based for loop再次印出

```
C++ Java Python Golang
```

```
C++ Java Python Golang Rust
```

```
Process finished with exit code 0
```

3. vector_showcase()

- Tradition for loop
- Range-based for loop
- 組合

```
1  #include "vector_showcase.hpp"
2  #include <iostream>
3  #include <string>
4  #include <vector>
5
6  void print_all(std::vector<std::string> langs) {
7      for (int i = 0; i < langs.size(); i++) {
8          std::cout << langs[i] << " ";
9      }
10     std::cout << "\n";
11 }
12
13 void print_all_alternative(std::vector<std::string> langs) {
14     for (std::string lang : langs) {
15         std::cout << lang << " ";
16     }
17     std::cout << "\n";
18 }
19
20 void vector_showcase() {
21     std::vector<std::string> langs = {"C++", "Java", "Python", "GoLang"};
22     print_all(langs);
23     langs.push_back("Rust");
24     print_all_alternative(langs);
25 }
```

Range-base for loop

- 在C++11之前，我們通常使用傳統的for循環或while循環來遍歷容器
 - 但這有時會讓代碼變得較為冗長和容易出錯
- 為了解決這些問題，C++11引入了一種新的循環結構，稱為Range-based for loop
 - 更直觀、更簡潔地traverse容器或任何提供了迭代器的資料結構

```
1 for (declaration : range) {  
2     // code to execute for each element  
3 }
```

- declaration是每次迭代中當前元素的聲明
- range則是你想traverse的容器或迭代器
- 優點： Range-based for loop關心容器中的元素，而不是他們的索引值
- 缺點： 在「需要知道元素的索引」或「進行複雜的traversal」時，傳統的for loop可能更適合

目標(四): Memory management

- 展示展示了C++中兩種不同的記憶體管理方法
 - 手動記憶體管理
 - 使用智慧指標 (std::shared_ptr) 記憶體管理
- 實作步驟
 - 手動
 - 使用new來分配一個int, 並給他一個整數值來初始化
 - 使用new來分配一個std::string, 給他一個字串來初始化
 - 使用new來分配一個LargeObject類別
 - 定義了一個大的整數陣列, 其中包含500個整數元素
 - 使用完刪除這些空間
 - 使用智慧指標完成上述功能



shared_ptr_showcase.cpp

- 先處理include
 - #include "shared_ptr_showcase.hpp"
 - #include <iostream>
 - #include <string>
 - #include <Memory>



shared_ptr_showcase.cpp

- 手動
 - 使用new來分配一個int，並給他一個整數值來初始化
 - 使用new來分配一個std::string，給他一個字串來初始化
 - 使用new來分配一個LargeObject類別
 - 定義了一個大的整數陣列，其中包含500個整數元素
 - 使用完刪除這些空間

```
5 class LargeObject {  
6     private:  
7         int m_Arr[500];  
8 };
```

```
21 void manual_memory_management() {  
22     int *obj1 = new int(4);  
23     std::string *obj2 = new std::string("some string");  
24     LargeObject *obj3 = new LargeObject();  
25  
26     delete obj1;  
27     delete obj2;  
28     delete obj3;  
29 }
```

shared_ptr_showcase.cpp

- smart_pointer

```
31 void smart_pointer() {  
32     std::shared_ptr<int> obj1 = std::make_shared<int>(1);  
33     std::shared_ptr<std::string> obj2 =  
34         std::make_shared<std::string>("some string");  
35     std::shared_ptr<LargeObject> obj3 = std::make_shared<LargeObject>();  
36 }
```

side_effect()

```
11 class Entity {
12 public:
13     Entity(std::shared_ptr<int> ptr) { m_Ptr = ptr; }
14
15     void SetValue(int value) { *m_Ptr = value; }
16     int GetValue() const { return *m_Ptr; }
17
18 private:
19     std::shared_ptr<int> m_Ptr;
20 };
```

```
39 void side_effect() {
40     std::shared_ptr<int> p = std::make_shared<int>(1);
41     Entity e(ptr: p);
42     std::cout << e.GetValue() << "\n";
43     *p = 5;
44     std::cout << e.GetValue() << "\n";
45 }
```