計算機程式設計

Computer Programming

From C to C++

W18: Self-Learning

GitHub repo

Outline

- Some main difference between C and C++
  - **Compiler and Header files**
  - **class** (absent in C)

[Usage] Compiler in C++

https://gcc.gnu.org/onlinedocs/gcc-3.3.6/gcc/G_002b_002b-and-GCC.html

- When referring to C++ compilation, it is usual to call the compiler “G++”. Since there is only one compiler, it is also accurate to call it “GCC” (GNU Compiler Collection) no matter what the language context.
- For some code editors, you may need to switch to G++ when running C++ code.

[Usage] Header files

- 有些原本定義在C語言函式庫的標頭檔，在C++中前面會多一個c，且不用加.h副檔名
- 舉例來說：

C Header file	C++ Header file
#include <stdio.h>	#include <cstdio>
#include <stdlib.h>	#include <cstdlib>
#include <string.h>	#include <cstring>
#include <math.h>	#include <cmath>
#include <ctype.h>	#include <cctype>

[Usage] iostream

- iostream 是 C++ 中跟 input / output 函式有關的標頭檔
- iostream 中定義了 cout 跟 cin，分別具備輸出與輸入的功能
  - cout 相當於 C 的 printf，但不需要像 printf 中指定 format specifier
  - cin 相當於 C 的 scanf，但不需要像 scanf 中指 format specifier

[Usage] Documents of cout and cin

cout 和 cin 被定義在一個叫做 std 的命名空間 (namespace)

std::cout

<iostream>

```
extern ostream cout;
```

https://cplusplus.com/reference/iostream/cout/

<iostream> 所有的內容都定義在std命名空間中
可以看 https://cplusplus.com/reference/iostream/

std::cin

<iostream>

```
extern istream cin;
```

https://cplusplus.com/reference/iostream/cin/

[Definition] Namespace (命名空間)

- Namespace is a feature in C++ (not in C).
- Namespace allows us to fix naming conflicts.
- For example, you wrote some code that has a function called xyz() and there is another library available which is also having same function xyz(). A namespace is designed to overcome this difficulty
- In the slides, we are not going to tell you how to create a namespace in C++. You can search and try on your own.

[Usage] Namespace (命名空間)

C-course-materials/11-C++/hello.cpp

只要加上一行 using namespace std;
就可以指定使用 std 中所定義的函式

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

int main(void){
    cout << "Hello, World!" << endl;
    return 0;
}
```

加了 using namespace std;
可以直接使用iostream的函式

```cpp
#include <iostream>

int main(void){
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

不加 using namespace std;
使用iostream的函式時，
要加命名空間std::

Print in C++

C-course-materials/11-C++/hello.cpp

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

int main(void){
    cout << "Hello, World!" << endl;
    return 0;
}
```

cout: 輸出函式，搭配 << 可以印出內容

endl: 輸出換行符號

注意順序！是先 << 印出 Hello, World!”後再 << endl 印出換行符號

[Usage] << and >>

- <<: insertion operator (印出)
- >>: extraction operator (讀入)
- Check here for more usage

Input via your keyboard

C-course-materials/11-C++/print_keyboard_input.cpp

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

int main(void){
    int num1;
    cin >> num1;
    cout << "You entered " << num1 << endl;
}
```

宣告變數的方式同 C

Input via your keyboard

C-course-materials/11-C++/print_keyboard_input.cpp

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

int main(void){
    int num1;
    cin >> num1;
    cout << "You entered " << num1 << endl;
}
```

cin 取得鍵盤輸入的值，接著將值賦予給變數 num1

Input via your keyboard

C-course-materials/11-C++/print_keyboard_input.cpp

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

int main(void){
    int num1;
    cin >> num1;
    cout << "You entered " << num1 << endl;
}
```

第二個<<後的num1是接在You entered 後印出

[Definition] Function Overloading

- In C++, multiple functions can share the same name, as long as their parameter lists differ in type, number (參數數量), or both.
- This feature is called function overloading (only in C++), and it allows functions to perform similar operations while operating on different types or numbers of arguments.

Function Overloading (1)

C-course-materials/11-C++/func_overloading_same_params.cpp

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

void print(int num) {
    cout << "Integer: " << num << endl;
}
void print(double num) {
    cout << "Double: " << num << endl;
}
void print(string str) {
    cout << "String: " << str << endl;
}
int main(void) {
    print(42);           // call print(int)
    print(3.14);         // call print(double)
    print("Hello");      // call print(string)
    return 0;
}
```

For function overloading, a C++ program chooses the function with the matched type and number of parameters.

Function Overloading (1)

C-course-materials/11-C++/func_overloading_same_params.cpp

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

void print(int num) {
    cout << "Integer: " << num << endl;
}
void print(double num) {
    cout << "Double: " << num << endl;
}
void print(string str) {
    cout << "String: " << str << endl;
}
int main(void) {
    print(42);           // call print(int)
    print(3.14);         // call print(double)
    print("Hello");      // call print(string)
    return 0;
}
```

For function overloading, a C++ program chooses the function with the matched type and number of parameters.

Function Overloading (1)

C-course-materials/11-C++/func_overloading_same_params.cpp

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

void print(int num) {
    cout << "Integer: " << num << endl;
}
void print(double num) {
    cout << "Double: " << num << endl;
}
void print(string str) {
    cout << "String: " << str << endl;
}
int main(void) {
    print(42);           // call print(int)
    print(3.14);         // call print(double)
    print("Hello");      // call print(string)
    return 0;
}
```

For function overloading, a C++ program chooses the function with the matched type and number of parameters.

Function Overloading (2)

C-course-materials/11-C++/func_overloading_diff_params.cpp

Let's see another example, where one of the functions takes no parameter.

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

void print_star(void) {
    cout << "Print 5 stars: *****" << endl;
}
void print_star(int num) {
    cout << "Print " << num << " stars: ";
    for (int i = 0; i < num; i++) {
        cout << "*";
    }
}
int main(void) {
    print_star();
    print_star(3);
}
```

Function Overloading (2)

C-course-materials/11-C++/func_overloading_diff_params.cpp

Let's see another example, where one of the functions takes no parameter.

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

void print_star(void) {
    cout << "Print 5 stars: *****" << endl;
}
void print_star(int num) {
    cout << "Print " << num << " stars: ";
    for (int i = 0; i < num; i++) {
        cout << "*";
    }
}
int main(void) {
    print_star();
    print_star(3);
}
```

For function overloading, a C++ program chooses the function with the matched type and number of parameters.

Function Overloading (2)

C-course-materials/11-C++/func_overloading_diff_params.cpp

Let's see another example, where one of the functions takes no parameter.

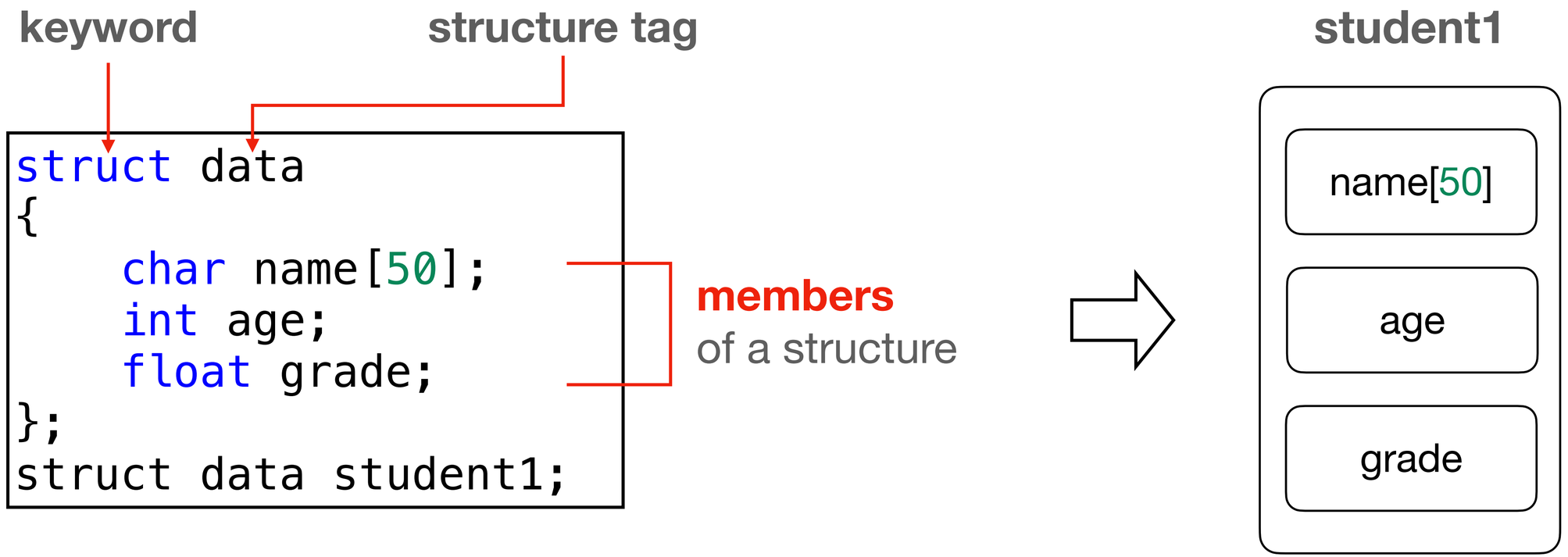
```
#include <iostream>
using namespace std;

void print_star(void) {
    cout << "Print 5 stars: *****" << endl;
}
void print_star(int num) {
    cout << "Print " << num << " stars: ";
    for (int i = 0; i < num; i++) {
        cout << "*";
    }
}
int main(void) {
    print_star();
    print_star(3);
}
```

For function overloading, a C++ program chooses the function with the matched type and number of parameters.

Class

[Recap] What are structures?

- Declare a structure:



[Recap] Example of struct

C-course-materials/11-C++/struct_example.cpp

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

struct Window {
    char id;
    int width;
    int height;
};
int area(struct Window w) {
    return w.width * w.height;
}
int main(void){
    Window w1; // typedef is not needed in C++
    // Window w1 = {'A', 10, 20};
    w1.id = 'A';
    w1.width = 10;
    w1.height = 20;
    cout << "Area of window " << w1.id << " is: " << area(w1) << endl;
    cout << "Size of W1: " << sizeof(w1) << " bytes" << endl;
}
```

[Usage] Skeleton of a class

keyword

class name

Very similar to struct

```
class Window {
    public: // Declarations under this line are public
        type1 name1;
        type2 name2;
        ...

        return_type1 func_name1(p_type1 p_name1, p_type2 p_name2, ...) {
            FUNCTION DESCRIPTIONS
            return EXPRESSION;
        }
        return_type2 func_name2(p_type1 p_name1, p_type2 p_name2, ...) {
            FUNCTION DESCRIPTIONS
            return EXPRESSION;
        }
        ...
};
```

Data members

Function members

[Usage] Skeleton of a class

keyword

class name

Very similar to struct

```
class Window {
    public: // Declarations under this line are public
        type1 name1;
        type2 name2;
        ...

        return_type1 func_name1(p_type1 p_name1, p_type2 p_name2, ...) {
            FUNCTION DESCRIPTIONS
            return EXPRESSION;
        }
        return_type2 func_name2(p_type1 p_name1, p_type2 p_name2, ...) {
            FUNCTION DESCRIPTIONS
            return EXPRESSION;
        }
        ...
};
```

We don’t use public in struct because members in struct are public in default.

Data members

Function members

Example of class

C-course-materials/11-C++/class_example.cpp

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

class Window {
    public: // Declarations under this line are public
        char id;
        int width;
        int height;
        int area(void) {
            return width * height;
        }
};
int main(void){
    Window w1;
    w1.id = 'A';
    w1.width = 10;
    w1.height = 20;
    cout << "Area of window " << w1.id << " is: " << w1.area() << endl;
    cout << "Size of W1: " << sizeof(w1) << " bytes" << endl;
}
```

We can use a dot operator to assign or access a value from a class member.

[Definitions] Member accessibility

- Public: members are accessible and adjustable from outside the class.
- Private: members cannot be accessed or modified from outside the class.
  - Usually, setting members as private can reduce unsafe behaviors.

[Important Notes] Comparison between struct and class

	struct in C	**struct in C++**	**class (C++ only)**
Member accessibility	Unadjustable. Members are always public.	Adjustable. Default in public.	Adjustable. Default in private.
Data members	O	O	O
Function members	X	O	O
Inheritance	X	O	O

Call a function in a class

C-course-materials/11-C++/call_func_in_class.cpp

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

class Window {
    public: // Declarations under this line are public
        char id;
        int width;
        int height;
        int area(void) {
            return width * height;
        }
        void print_area(void) {
            cout << "Area of window ";
            cout << id << " is: " << area() << endl;
        }
};
int main(void){
    Window w1;
    w1.id = 'A';
    w1.width = 10;
    w1.height = 20;
    w1.print_area();
}
```

Pass values to a class function

C-course-materials/11-C++/pass_val_to_class_func.cpp

```
class Window {
    public: // Declarations under this line are public
        char id;
        int width;
        int height;

        int area(void) {
            return width * height;
        }
        void print_area(void) {
            cout << "Area of window ";
            cout << id << " is: " << area() << endl;
        }
        void set_data(char i, int w, int h) {
            id = i;
            width = w;
            height = h;
        }
};
```

```
int main(void){
    Window w1;
    w1.set_data('A', 10, 20);
    w1.print_area();
}
```

Pass a class (as an object) to a function

C-course-materials/11-C++/pass_class_to_func.cpp

```
class Window {
    public: // Declarations under this line are public
        char id;
        int width;
        int height;

        int area(void) {
            return width * height;
        }
        void set_data(char i, int w, int h) {
            id = i;
            width = w;
            height = h;
        }
};
void print_area(Window w) {
    cout << "Area of window ";
    cout << w.id << " is: " << w.area() << endl;
}
```

```
int main(void){
    Window w1;
    w1.set_data('A', 10, 20);
    print_area(w1);
}
```

At this time, w1 is viewed as an object.

Function Overloading in a class

C-course-materials/11-C++/class_func_overloading.cpp

```
class Window {
    public: // Declarations under this line are public
        char id;
        int width;
        int height;

        int area(void) {
            return width * height;
        }
        void print_area(void) {
            cout << "Area of window ";
            cout << id << " is: " << area() << endl;
        }
        void set_data(char i, int w, int h) {
            id = i;
            width = w;
            height = h;
        }
        void set_data(char i) {
            id = i;
        }
        void set_data(int w, int h) {
            width = w;
            height = h;
        }
};
```

```
int main(void){
    Window w1, w2;
    w1.set_data('A', 10, 20);
    w2.set_data('B');
    w2.set_data(30, 40);
    w1.print_area();
    w2.print_area();
}
```

We can also use function overloading in a class!

Class (with private members)

Setting private members in a class (wrong)

C-course-materials/11-C++/class_private_member_wrong.cpp

```
class Window {
    private: // Declarations under this line are private
        char id;
        int width;
        int height;
    public: // Declarations under this line are public
        int area(void) {
            return width * height;
        }
        void print_area(void) {
            cout << "Area of window ";
            cout << id << " is: " << area() << endl;
        }
};

int main(void){
    Window w1;
    w1.id = 'A'; // Error! `id` cannot be accessed (private).
    w1.width = 10; // Error! `width` cannot be accessed (private).
    w1.height = -20; // Error! `height` cannot be accessed (private).
    w1.print_area();
}
```

This is a wrong case!!

Setting private members in a class

C-course-materials/11-C++/class_private_member.cpp

```
class Window {
    private: // Declarations under this line are private
        char id;
        int width;
        int height;
    public: // Declarations under this line are public
        int area(void) {
            return width * height;
        }
        void print_area(void) {
            cout << "Area of window ";
            cout << id << " is: " << area() << endl;
        }
        void set_data(char i, int w, int h) {
            id = i;
            width = w;
            height = h;
        }
};
```

```
int main(void){
    Window w1;
    w1.set_data('A', 10, 20);
    w1.print_area();
}
```

We can use another function to overcome the inaccessibility of private members.