Chapter 1.

C++ 基礎程式設計導論 I

Introduction to C++ Program Design I

嘉義女中數資班資訊組課程

National Chia-Yi Girls' Senior High School Information Course

講師：陳俊安 Colten

自我介紹 - 陳俊安 Colten

- 國立成功大學資訊工程系 大一 (特選甲組)
- 熱愛競技程式設計、演算法、資料結構
- APCS 實作滿級分
- 台南女中資訊研究社 37th C++ 進階班講師
- 2023 資訊之芽培訓計畫 南區算法班 講師
- 2022 ICPC Asia Taoyuan Regional Programming Contest Bronze Medal
- NHDK Ten Point Round Problem Setter
- NHDK 南北四校聯合資研暑期選手培訓課程 講師

簡單的課程進度說明

- C++ 基礎語法 (APCS 實做 3)
- 演算法、資料結構 (APCS 實做 4 ~ 5)
- 正常來說我大概會兩個星期來一次，如果那一週沒來也會有影片ouob

C++

- 跟 C 語言是朋友，最大的差別在於 C++ 是可物件導向的程式語言之一
  - 你可能也有聽過 C#，但我覺得語法上差蠻多的XD
- C++ 是高階程式語言
  - 低階程式語言 Low-level programming language
    - 低階語言一般來說都指 機器碼 或 組合語言
  - 高階程式語言 High-level programming language
    - 語法與人類慣用的語言相近

低階語言與高階語言

- 在物件導向出現之後，像 C 這種非物件導向程式語言常常也有人稱它為低階或中階
- 物件導向程式設計
  - Object-oriented programming (OOP)
  - 可增加程式的靈活性和可維護性

為什麼這一次選擇使用 C++ 這一個語言

- 要承認一件事情
  - 會選擇 C++ 跟物件導向無關，所以這一次課程不會有物件導向相關的東西
  - 如果這一次還要上物件導向的東西的話我 3 天內一定講不完XD
  - 雖然 struct 的概念也很類似就是为
- 有一些好用的工具可以直接用，且語法比 C 語言更容易上手
- ~~C++ 是我第一個接觸的程式語言，我 4 年前用到現在~~

Online Judge 線上評測系統

- 一般來說我們平時練習的題目都會到某個 Online Judge 上
  - 對於平時這樣的練習來說我們叫做 刷題
- 每一個題目裡面會有一些測試資料去檢查你的程式有沒有滿足題目要求
  - Accepted or Correct (AC) 你的程式碼通過所有測試資料
  - Wrong Answer (WA) 你的程式碼在某些測試資料輸出錯誤
  - Runtime Error (RE) 你的程式碼意外停止
  - Time Limit Exceed (TLE)程式碼跑某個測試資料的時候超出時間限制
  - Memory Limit Exceed (MLE) 你的程式碼使用的記憶體量超出限制

C++ 的一些語法特色

- 每一個描述後面都會有一個分號；表示該描述句的結束
- 程式碼執行順序由上到下 (其實基本上程式語言都這樣)
- 偏嚴謹，在做一些運算的時候對於型態一板一眼
  - Example:
    - 使用 max(a,b) 函數的時候 a,b 型態需要一致
    - 如果 a 跟 b 都是整數，但如果是不同類型的整數也不行
    - 整數有許多類型後面會介紹到

變數與型態

Lecture 1

變數

- 會變的數字？
- 生活中常見的變數
  - 你的郵局存款
  - 遊戲裡的人物血量
  - 你一卡通裡面的金額

變數與型態

- 每一個變數可能存在不同的形態
  - 一卡通裡面的金額 -> 整數 eg. 200 元
  - 遊戲裡面的人物寫量 -> 浮點數 (小數) eg. HP: 45.3
- 所以每一個變數都有一個屬於自己的型態
  - 就很像我們人有分 男生 跟 女生

常用型態

- 整數
  - int (-2^31 ~ 2^31 - 1)
  - long long (-2^63 ~ 2^63 - 1)
  - unsigned long long (0 ~ 2^64 - 1)
- 浮點數 (小數)
  - float (單精度浮點數) 最大有效位置小數點後 7 位
  - double (雙精度浮點數) 最大有效位至小數點後 16 位
  - long double
- 字元 char (eg. 'a')
- 字串 string (eg. "Colten")
- 布林 bool (true or false)

變數宣告

- 在 C++ 中如果我們要宣告一個變數，其語法如下
  - [變數型態] [變數名稱]; // 記得分號一定要加
    - int a; // 宣告 a 是一個整數變數
    - char b; // 宣告 b 是一個字元變數
    - bool apple; // 宣告 apple 是一個布林變數

```
3 int a,x,y,z;
4
5 char b;
6
7 bool apple;
```

主程式與輸入輸出

Lecture 2

主程式 main function

- 你的 C++ 程式碼開始執行時，就會先呼叫的 Function
  - 關於函數呼叫到後面某一個章節時會細講
- 最一開始大家就把它想像成是一個起手式，主要架構即可

```cpp
#include <iostream>

using namespace std;

int main()
{
    return 0;
}
```

#include <iostream>

- <iostream> 我們稱之為 **標頭檔** (header file)
  - 類似一個工具箱，每一個標頭檔案都有一些特定的工具可以使用
  - 想要使用某一個工具，就必須拿出 (#include) 那一個工具箱
  - 常見標頭檔
    - <iostream> // 最基本的標頭檔 (cin , cout …)
    - <cmath> // 數學相關 (sin , cos , tan , pow …)
    - <iomamip> // 格式化輸出相關
    - <algorithm> // 某些演算法相關 (sort, min, max …)

using namespace std; (記得分號)

- 命名空間 (namespace)
- 可以把它想像成是某一個群組的名稱
- 可能某兩個群組都有一個人叫做 a，但是這兩個 a 是不同人
- 所以假設第一個群組叫做 apple，第二個叫做 banana
- 那麼在 C++ 中為了區分我們可以寫成：
  - apple:a 與 banana:a，讓程式知道我們說的 a 是哪一個群組的 a
- 而我們在基礎語法階段會使用到的群組是 std，為了寫的時候可以省略掉 std:: (類似指定群組) 的這一個動作，我們會透過這樣的方式讓 std 可以被省略
- (請注意，這一個寫法在專案開發上是很不建議的動作，這邊只是為了方便)

int main()

- 在還沒學到 Function 之前，大家先有幾個認知就可以了
- main function 的型態必須是 32 位元的整數，所以才會寫 int main()
- int main() 這一個動作是 宣告函式
- 要寫東西在 main 函式裡面要使用大括號把要寫的東西包在裡面

```
4
5 int main()
6 {
7     int a,b,c;
8     return 0;
9 }
```

return 0;

- 由於 main 函式是整數型態，因此回傳的東西必須是一個整數
- 不過由於這一個函式是 main function，所以我們通常也用回傳 0 的方式來讓我們知道這一個程式是正常終止的
- return 的用途 Function 那邊還會細講，這邊大家就先大概知道即可

```
4
5 int main()
6 {
7     int a,b,c;
8     return 0;
9 }
```

main function

- 有了這一個架構之後，你的程式碼基本上就可以正常編譯跟執行了
  - 編譯，電腦把你的程式碼轉成它看得懂的東西
  - 執行，執行程式碼被轉換後的執行檔案

```cpp
#include <iostream>

using namespace std;

int main()
{
    return 0;
}
```

輸入與輸出

- 讓程式可以跟使用者互動
  - 語法：
    - 輸入：cin
    - 輸出：cout
    - 輸入 in、輸出 out

cout 輸出一個東西

- cout << [你要輸出的東西];

```cpp
#include <iostream>

using namespace std;

int main()
{
    int ouo = 5;

    cout << ouo; // 輸出 5

    return 0;
}
```

cout 輸出兩個東西

- cout << [你要輸出的東西1] << [你要輸出的東西2];

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 10 , b = 5;

    cout << a << b; // 輸出 105
    // 105 是連在一起的哦！

    cout << a << " " << b; // 10 5
    // 這個才沒有連在一起

    return 0;
}
```

cout 輸出三個東西

- cout << [你要輸出的東西1] << [你要輸出的東西2] << [你要輸出的東西3];
- 要在同一個 cout 輸出多個變數，只要使用 << 區分開即可

cout 輸出字串

- 字串在 C++ 的語法中，被指定的文字必須使用 **雙引號** 包起來
  - “Hello!”
  - “OwO”
  - “Q”

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world";

    return 0;
}
```

更多例子

- 字串在 C++ 的語法中，被指定的文字必須使用 **雙引號** 包起來

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 5;

    cout << "Hello     " << a << "owo";

    return 0;
}
```

練習題

- 輸出字串 Hello world!
- 輸出整數(請不要使用 "2023" 作弊QQ) 2023
- 輸出 2023 3202 OwO!!!

換行

- 把 cout 分開寫並不表示它會換行哦！

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world";
    cout << "Hello world";

    // output: Hello worldHello world

    return 0;
}
```

我想換行QQ feat. 跳脫字元

- 需要有一個先備知識，跳脫字元 \ (我都叫他反斜線)
  - 具有兩個功能
  - 功能 1: 特殊用途
    - \n 換行
    - \t 四個空格
    - 更多特殊功能可以 Google 找看看
  - 功能 2: 化解衝突 (eg. " , \ , ' 等等)
    - 我想要輸出雙引號，但是雙引號是具有功能的字元
    - 為了避免衝突我們可以在雙引號前面加上 \ 避免衝突
    - ```
      cout << "  \"  "; // 輸出   \
      ```

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world\nHello world"; // 輸出兩行 Hello world

    return 0;
}
```
```

但 \n 其實是 C 的作法，C++ 是 endl

- 這兩個方法都建議知道一下比較好
- 順帶一提，使用 \n 的程式碼有時候會比 endl 的還要來的快上許多
  - 原因牽涉到 釋放緩衝區 的問題
  - 不過這一個東西偏底層了，有興趣的可以參考 Wiwiho 的筆記

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world" << endl << "Hello world"; // 輸出兩行 Hello world

    return 0;
}
```

練習題

- 輸出三行字串 OuOb
  - Hint: 跳脫字元當中的特殊功能 或是 使用 endl
- 輸出字串 ><"\\\\\'"///
  - Hint: 跳脫字元當中的化解衝突

輸入 cin

- 讓使用者能夠跟程式互動
- 語法：cin >> [要被輸入的變數];

```cpp
#include <iostream>

using namespace std;

int main(void)
{
    int a;

    cin >> a; // 輸入 4，這個時候 a = 4
    cout << a << endl; // 輸出 4
}
```

多個變數輸入 cin

- 讓使用者能夠跟程式互動
- 語法：cin >> [要被輸入的變數 1] >> [要被輸入的變數 2];

```cpp
#include <iostream>

using namespace std;

int main(void)
{
    int a,b;

    cin >> a >> b; // 輸入 10 20

    cout << b << " " << a << "\n"; // 輸出 20 10
}
```

輸入 cin

- 多個變數之間通常我們在輸入的時候會以 空白 分開
- 但是如果你輸入使用 換行 分開兩個不同的變數也是可以的
- Example (下面這兩個輸入的意思是一樣的) :

剛剛的特例

- 在字元中有例外情況，跟字元相鄰的變數是可以不用空白分開的
- Example:

```cpp
#include <iostream>

using namespace std;

int main(void)
{
    int a,c;
    char b;

    cin >> a >> b >> c; // 輸入 10t20
    // 10t20 是全部連在一起的

    cout << a << " " << b << " " << c; // 正確輸出 10 t 20
}
```

練習題

- 依序輸入三個整數 a,b,c 並以 c a b 的順序輸出
- 輸入一個字串，並將這一個字串輸出
  - 測試
    - 輸入 Hello!
    - 輸入 OuO!

字串的輸入

- 輸入字串 Hello world! 時，你會發現用我們剛剛教的寫法會只有出現 Hello 這一個字串，這是為什麼呢？

```cpp
#include <iostream>

using namespace std;

int main(void)
{
    string s;
    cin >> s;
    cout << s << "\n";
}
```

字串的輸入

- 還記得我們剛剛提到兩個變數之間輸入的時候會以 空白 隔開嗎？
- 雖然 "Hello World!" 是一個字串，但是程式上會認為你是在輸入
  - Hello
  - world!
- 因此第一個他得到的字串是 Hello，然後程式就結束了
- 那麼究竟我們應該要如何實現一次輸入 Hello world! 呢？

getline(cin,string);

- getline 可以幫助我們連空白字元都一起讀入
- 語法：getline(cin,[要被輸入的字串變數]);

```cpp
#include <iostream>

using namespace std;

int main(void)
{
    string s;
    getline(cin,s);
    cout << s << "\n";
}
```

getline 的一些小細節

- 如果使用 getline 的前一次輸入有使用一般的 cin，會發現東西不見了

```cpp
#include <iostream>

using namespace std;

int main(void)
{
    int q;
    cin >> q;

    string s;
    getline(cin,s);

    cout << q << " " << s << "\n";
}
```

getline 的一些小細節

- 如果使用 getline 的前一次輸入有使用一般的 cin，會發現東西不見了
- 這是為什麼？
  - 由於前一次 cin 的東西最後有一個空字串沒有被清掉，因此你的 getline 獨到的實際上是那一個沒有被清掉的空字串
  - 詳細的原因可以 Google
    - 關鍵字：cin 與 getline 混用

如何解決？

- 在 getline 之前使用 cin.ignore(); 這一個函數即可
- 或是你可以做兩次 getline，第一次把那一個空字串讀掉

```cpp
#include <iostream>

using namespace std;

int main(void)
{
    int q;
    cin >> q;

    string s;
    cin.ignore();
    getline(cin,s);

    cout << q << " " << s << "\n";
}
```

練習題

- 依序輸入一個數字一個字串 2023 跟字串 "Apple Banana"
  - Hint: getline(cin,string) and cin.ignore()

運算子

Lecture 3

運算子

- 簡單來說，運算子指的就是 **運算用的符號**
- 運算子又分為很多種
  - 算術運算子 (+, -, *, /)
  - 比較運算子 (<, >, <=, >=)
  - 位元運算子 (&, |, ^)

算術運算子

- 就跟大家平時用的數學一樣
- 在程式中也遵守著數學的規則
  - 做除法時，不能除以 0
  - 先乘除，後加減，有括號先算

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 10, b = 5;

    cout << a + b << "\n"; // 15
    cout << a - b << "\n"; // 5
    cout << a * b << "\n"; // 50
    cout << a / b << "\n"; // 2
    cout << a + b * a << "\n"; // 60
    cout << b * ( a + b ) << "\n"; // 75
}
```

算術運算子 %

- 取餘數

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 10, b = 7;

    cout << a % b << "\n"; // 3
}
```

算術運算子

- 在做運算的時候要特別小心型態的問題
- 請見下方程式碼
- Why?

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 10, b = 3;

    double a2 = 10.0;

    cout << a / b << "\n"; // 3
    cout << a2 / b << "\n"; // 3.33333
}
```

算術運算子

- 由於 a,b 都是整數，而整數跟整數做運算會預設結果出來也是整數
  - a / b = 3.33333 只取整數部分 -> 3
- 由於 a2 是浮點數了， 因此出來的結果為 3.33333

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 10, b = 3;

    double a2 = 10.0;

    cout << a / b << "\n"; // 3
    cout << a2 / b << "\n"; // 3.33333
}
```

算術運算子

- 還有一個常犯的錯誤
  - 如果 a,b 都是 int
  - 當 a + b 會超出 int 範圍且沒有使用 long long 儲存的情況下會發生 overflow (溢位) 的問題，造成出來的結果不如預期

比較運算子

- 如果在程式中使用比較運算子做運算，得到的結果會是 布林 的型態
  - 布林 bool
- 當敘述成立時得到的結果為 true (1)，反之為 false (0)
- 常見比較運算子 (<, <=, >, >=, !=, ==)
  - ! 通常我們會稱之為 NOT，因此 != 就是 不等於 的意思
  - == 是指相等的意思，是兩個等於！
  - 用括號包起來避免衝突

```
int a = 10 , b = 5;

cout << ( a > b ) << "\n"; // 1
cout << ( a < b ) << "\n"; // 0
cout << ( a == b ) << "\n"; // 0
```

位元運算子

- 會將數字轉換成 二進位 做完運算後再將結果轉回 十進位 回傳
- 大家應該都對二進位與十進位沒什麼概念，因此這邊我們介紹一下

十進位 與 二進位

- 平時我們在現實生活中都是使用十進位做運算
- 為什麼被稱之為十進位？可以想像成我們做加法的時候都是 10 要進位
- 舉一反三，二進位當然就是 2 的時候要進位
  - 因此十進位的數字會出現 0 ~ 9
  - 而二進位的數字指會出現 0 或 1

十進位轉二進位

- 短除法
  - 將一個數字不斷的使用 2 做短除法，並將餘數紀錄起來
  - 紀錄的餘數倒著回去就會是該數字轉成二進位的結果
  - Example: (因此 12 的二進位是 1100)
    - 12 … 0
    - 6 … 0
    - 3 … 1
    - 1 … 1

二進位轉十進位

- 先想像一個十進位的數字是怎麼被組合出來的
  - 1234
    - 0 * 10 + 1 = 1
    - 1 * 10 + 2 = 12
    - 12 * 10 + 3 = 123
    - 123 * 10 + 4 = 1234

二進位轉十進位的第一種方法

- 那我們把同樣的策略放在二進位上
  - 1011
    - 0 * 2 + 1 = 1
    - 1 * 2 + 0 = 2
    - 2 * 2 + 1 = 5
    - 5 * 2 + 1 = 11
  - 因此 1011 轉成十進位的結果是 11

二進位轉十進位的第二種方法

- 我們一樣先看看十進位的數字
  - 1234
    - $1000 * 1 + 100 * 2 + 10 * 3 + 1 * 4 = 1234$
    - $1000 = 10^3$
    - $100 = 10^2$
    - $10 = 10^1$
    - $1 = 10^0$

二進位轉十進位的第二種方法

- 我們換成二進位的數字
  - 1011
    - $8 * 1 + 4 * 0 + 2 * 1 + 1 * 1 = 11$
    - $8 = 2^3$
    - $4 = 2^2$
    - $2 = 2^1$
    - $1 = 2^0$

AND 運算

- 兩個同位元的數字必須同時為 1，AND 出來的結果才會是 1
  - 1 AND 0 = 0
  - 0 AND 1 = 0
  - 0 AND 0 = 0
  - 1 AND 1 = 1

OR 運算

- 兩個同位元的數字其中一個是 1，OR 出來的結果就會是 1
  - 1 OR 0 = 1
  - 0 OR 1 = 1
  - 0 OR 0 = 0
  - 1 OR 1 = 1

XOR 運算

- 兩個同位元的數字不一樣，XOR 出來的結果就會是 1
  - 1 XOR 0 = 1
  - 0 XOR 1 = 1
  - 0 XOR 0 = 0
  - 1 XOR 1 = 0

實際運算

- 11 XOR 13
  - 先把這兩個數字轉成二進位
    - 1011 (11)
    - 1101 (13)
    - 同位置的位元去做運算
      - 1 XOR 1 = 0
      - 0 XOR 1 = 1
      - 1 XOR 0 = 1
      - 1 XOR 1 = 0
    - 因此 11 XOR 13 的二進位結果為 0110 = 110 轉成十進位就是 6

實際運算

- 小心平時在做位元運算的時候會有 **優先順序** 的問題，為了避免造成不是自己要的結果，我們通常會使用 **括號** 來避免
  - 因為括號的優先權最大，有括號先算

```
5 int main()
6 {
7     int a = ( ( 10 ^ 20 ) & 30 );
8 }
```

補充：向左位移 <<

- 將十進位數字轉成二進位後向左邊後推 k 位，新的位置補 0
- Example：
  - 二進位 1101 向左位移 2 位會變成二進位的 110100

```
 4
 5 int main()
 6 {
 7     int a = 11;
 8
 9     cout << ( a << 2 );
10 }
```

補充：向右位移 >>

- 將十進位數字轉成二進位後向右邊後推 k 位，少掉的部分直接丟掉
- Example：
  - 二進位 1101 向右位移 2 位會變成二進位的 11

```
 5 int main()
 6 {
 7     int a = 2023;
 8
 9     cout << ( a >> 2 );
10 }
```

補充：向左、右位移的一些小祕密

- 我們來活用一下剛學到的二進位觀念
- 由於向左位移 k 位會多出 k 個位置，我們在二進位轉十進位時，每到下一個位置就要 *2，換句話說多 k 位出來我們就要多乘 k 次 2
- 因此藉由這一個想法我們可以得出一個結論
  - 十進位的 a 向左位移 k 位的結果會變成 a * (2 的 k 次方)
- 那麼向右位移 k 位？
  - 十進位的 a 向右位移 k 位的結果會變成 …
  - 留給大家舉一反三當作練習

學完運算子你還必須要知道的事情

- 平常如果我們把一個變數 = 某一個值，這一個動作我們就稱為賦值
  - Example:
    - a = 10
- 程式在做這一個動作時，會先講等於右邊的東西處理完
- 右邊的東西處理完之後再將處理完的結果賦予給等於左邊的變數

```
7     int q = 10 + 20;
8
9     cout << q; // 30
```

現在 q = 10, 我想把 q 的值加上 p

```
 5 int main()
 6 {
 7     int q = 10 , p = 20;
 8
 9     q = q + p; // q + p = 30 , 把 30 的結果賦予給 q
10
11     cout << q << " " << p << "\n"; // q = 30, p = 20
12 }
```

學完運算子你還必須要知道的事情

- 平時我們在寫 a = a + b 這類的東西時可以簡寫成
  - a += b
- 只要是運算類的符號基本上都可以適用
  - a -= b
  - a *= b
  - a /= b
  - a ^= b
  - a <<= b

```
 5 int main()
 6 {
 7     int a = 2023;
 8
 9     a += 1;
10
11     cout << a; // 2024 Happy New Year!
12 }
```

學完運算子你還必須要知道的事情

- 把某個變數 +1 或 -1 還可以再縮寫成
  - 變數++ 或 變數--

```
 5 int main()
 6 {
 7     int a = 3;
 8
 9     a++;
10     cout << a << "\n"; // 4
11     a--;
12     cout << a << "\n"; // 3
13 }
```

初始化

- 做一些變數運算時記得小心變數初始化的問題
- 就很像那個變數不知道是什麼東西，但你把它拿去做運算
- 就會發生無法預測的事情

```
 5 int main()
 6 {
 7     int q; // q 不知道是什麼
 8
 9     q++; // 不知道 q 是什麼卻把它 +1
10 }
```

補充，前置與後置

- 加加還有減減也可以放在變數的前面，意思是不一樣的
  - 放在後面表示 **後運算**，也就是該做的東西都處理完才做 +1 或 -1
  - 反之則為 **前運算**，也就是先 +1 或 -1 才處理接下來的事情

```
 5 int main()
 6 {
 7     int a = 3;
 8
 9     cout << a++ << "\n"; // 3
10     cout << a << "\n"; // 4
11 }
```

```
 5 int main()
 6 {
 7     int a = 3;
 8
 9     cout << ++a << "\n"; // 4
10     cout << a << "\n"; // 4
11 }
```

條件判斷

Lecture. 4

條件判斷

- if 判斷式
  - 語法:
    - if(條件式) { 要做的事情 }
    - 只要 if 小括號裡面運算的結果為 true 就會執行大括號內的內容

```
int a = 5;

if( a == 5 ) // 注意這邊沒有分號
{
    a += 3;
}

cout << a << "\n"; // 8
```

更多例子

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     if( true )
8     {
9         cout << "Hello world\n"; // 會輸出
10    }
11    if(1)
12    {
13        cout << 1; // 會輸出
14    }
15 }
```

更多例子

```
 7      int a = 5 , b = 8;
 8
 9      if( a < b )
10      {
11          cout << 123 << "\n"; // 會輸出 123
12
13          if( b > a )
14          {
15              cout << 456 << "\n"; // 會輸出 456
16          }
17      }
```

更多例子

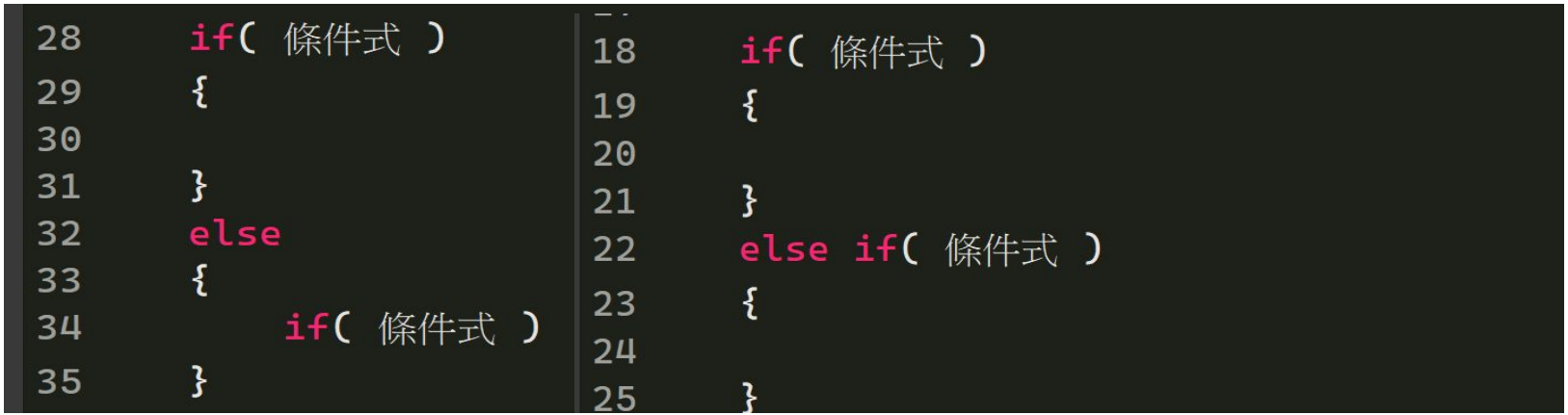
```cpp
int a = 5 , b = 8;

if( a < b )
{
    cout << "owo\n"; // 會輸出
}

if( a > b )
{
    cout << "oao\n"; // 不會輸出
}
```

條件判斷

```
 7    int a = 5 , b = 8;
 8
 9    if( a > b )
10    {
11        cout << "owo\n"; // 不會輸出
12    }
13    else
14    {
15        cout << "ouo\n"; // 會輸出
16    }
```

- else
  - 語法:
    - else { 要做的事情 }
    - else 之前一定必須有 if
    - 如果 if 的條件沒有成立，就會執行 else 大括號內的敘述

條件判斷



```
28      if( 條件式 )
29      {
30
31      }
32      else
33      {
34          if( 條件式 )
35      }
```

```
18      if( 條件式 )
19      {
20
21      }
22      else if( 條件式 )
23      {
24
25      }
```

- else if (條件式)
  - 其實 C++ 中沒有 else if 這一個語法，只是意義上剛好等價於上方的圖片
  - 也就是說如果上面的 if 不成立，就會繼續檢查 else if 的條件式
    - 如果 else if 裡面的條件式成立就會執行他大括號裡面的內容
    - 否則繼續往下一個 else if 檢查或是進入 else ， 沒東西就結束這一組判斷

範例

```
    int a = 5 , b = 8;

    if( a > b )
    {
        cout << "owo\n"; // 不會輸出
    }

    // 沒東西了, 結束這一組判斷

    if( a < b )
    {
        cout << "oao\n"; // 會輸出
    } // 上方的 if 已經沒有下一個 else 或 else if 了，因此結束當前這一組判斷
    if( true )
    {
        cout << 123; // 會輸出
    }
    else
    {
        cout << "owo\n"; // 不會輸出，因為上方的 if 已經成立
    }
```

範例

```
 9    if( a > b )
10    {
11        cout << "owo\n"; // 不會輸出
12    }
13    else if( a == b )
14    {
15        cout << "oao\n"; // 不會輸出
16    } // 由於上方 if , else if 的條件式都沒有成立，因此繼續往下看下一個 else if
17    else if( a < b )
18    {
19        cout << 123 << "\n"; // 會輸出 123
20    }
21    else
22    {
23        cout << "qq\n"; // 上方的 else if( a < b ) 已經成立，因此不會進入這一個 else
24    }
25    if( a < 10 )
26    {
27        cout << 456 << "\n"; // 會輸出 456，因為這一個 if 跟上面是不同組的
28    }
```

範例

```
 7     int a = 5 , b = 5;
 8
 9     if( a > b )
10     {
11         cout << "owo\n"; // 不會輸出
12     }
13     else if( a == b )
14     {
15         cout << "ouo\n"; // 會輸出
16     }
17     else
18     {
19         cout << "oao\n"; // 不會輸出
20     }
```

範例: Group 1 的結果不影響 Group 2

Group 1

到 else if 就沒有繼續了

Group 2

```
 5  int main()
 6  {
 7      int a = 5 , b = 5;
 8
 9      if( a > b )
10      {
11          cout << "owo\n"; // No output
12      }
13      else if( a == b )
14      {
15          cout << "ouo\n"; // Output: ouo
16      }
17      if( a <= b )
18      {
19          cout << "oao\n"; //  Output: oao
20      }
21  }
```

條件判斷 - 多個條件同時相等 &&

- if([條件1] && [條件2] && [條件3]) { 要做的事情 }

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 5 , b = 10;

    if( a == 5 && b == 10 )
    {
        cout << "Hello world\n";
    }
}
```

條件判斷 - 其中一個條件相等 ||

- if([條件1] || [條件2]) { 要做的事情 }

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 2023 , b = 10;

    if( a == 5 || b == 10 )
    {
        cout << "Hello world\n";
    }
}
```

條件判斷 - 大雜燴

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 2023 , b = 10;

    if( ( a == 5 || b == 10 ) && a == 2023 )
    {
        cout << "Hello world\n";
    }
}
```

迴圈

Lecture 5

迴圈

- 如果你現在想要輸出 10 次 Hello World!
- 你可能會這樣寫，那如果 100 次？1000 次？

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!\n";
    cout << "Hello world!\n";
    cout << "Hello world!\n";
    cout << "Hello world!\n";
    cout << "Hello world!\n";
    cout << "Hello world!\n";
    cout << "Hello world!\n";
    cout << "Hello world!\n";
    cout << "Hello world!\n";
    cout << "Hello world!\n";
}
```

迴圈

- 迴圈就是當我們要 **重複做同一件事情** 好幾次的時候派上用場！
- C++ 中的迴圈有三種
  - while 迴圈
  - for 迴圈
  - do-while 迴圈

while 迴圈

- 語法：
  - while(條件式) { 要重複執行的事情 }
  - 條件式的部分跟 if/else 那一個部份一樣
  - 條件式的結果為 true 就會執行

```cpp
#include <iostream>

using namespace std;

int main()
{
    int q = 0;

    while( q < 10 )
    {
        cout << "Hello world\n";
        q++;
    }
}
```

for 迴圈

- 語法：
  - for([第一次進迴圈要做的事];[條件式];[每一次迴圈結束後要做的事情])
    { 要重複執行的事情 }

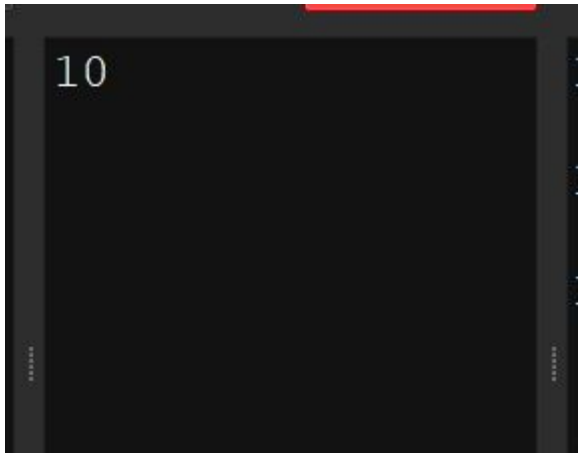
```
 7      for(int i=0;i<10;i++)
 8      {
 9          cout << i << "\n";
10      }
```

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

do-while 迴圈

- 語法：
  - do { 要做的事情 } while(條件式);
  - 不管怎麼樣都先執行一次要做的事情，最後才判斷要不要執行下一次

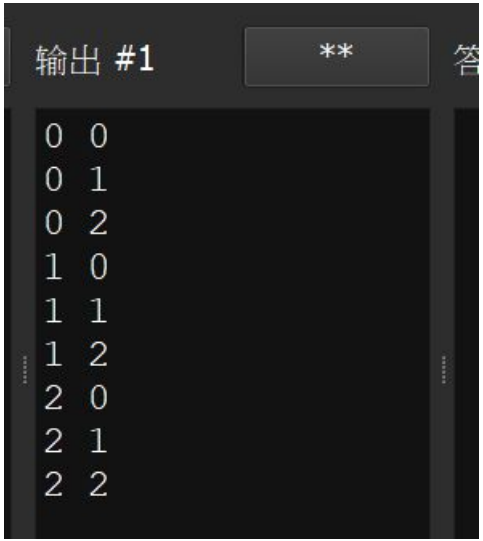
```
 5 int main()
 6 {
 7     int q = 10;
 8
 9     do
10     {
11         cout << q << "\n";
12     }while( q < 10 );
13 }
```



兩層以上的迴圈

- 迴圈裡面再包迴圈
- 不用想的太難，我們一樣照著程式的邏輯還有步驟走下去
- 我們先看一個範例

```
 5 int main()
 6 {
 7     for(int i=0;i<3;i++)
 8     {
 9         for(int k=0;k<3;k++)
10         {
11             cout << i << " " << k << "\n";
12         }
13     }
14 }
```



兩層以上的迴圈

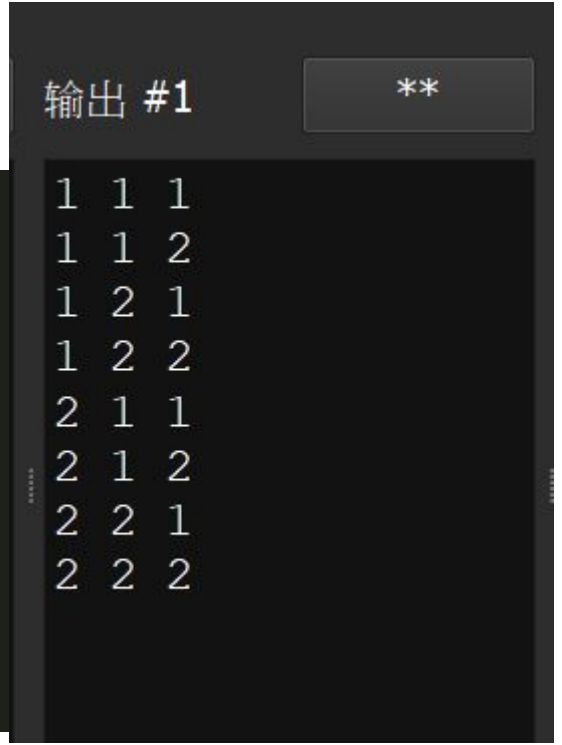
- 你會發現你可以把它理解成會執行 3 次裡面那一層 for 迴圈

```
 5 int main()
 6 {
 7     for(int i=0;i<3;i++)
 8     {
 9         for(int k=0;k<3;k++)
10         {
11             cout << i << " " << k << "\n";
12         }
13     }
14 }
```

兩層以上的迴圈

- 試試看三層吧！！

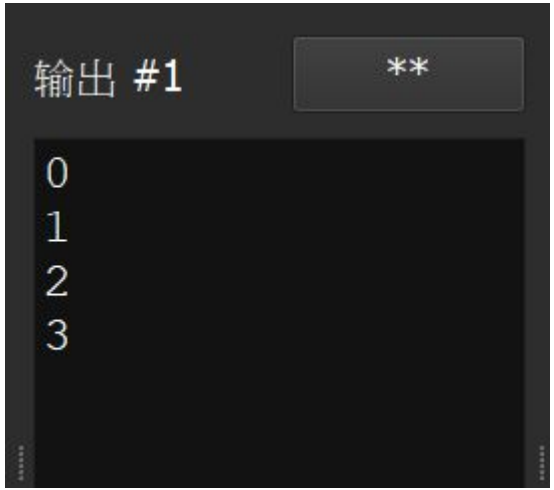
```
int main()
{
    for(int i=1;i<3;i++)
    {
        for(int k=1;k<3;k++)
        {
            for(int j=1;j<3;j++)
            {
                cout << i << " " << k << " " << j << "\n";
            }
        }
    }
}
```



break 跳出迴圈

- 直接跳出迴圈，不繼續在同一個迴圈執行

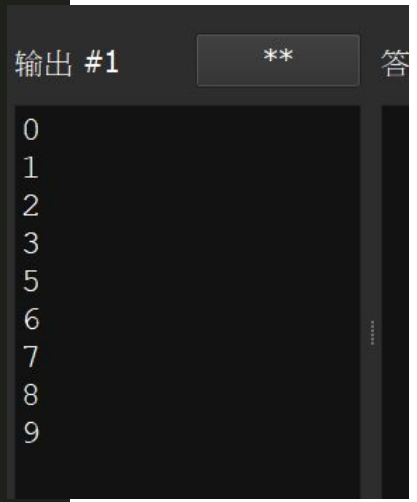
```
 5 int main()
 6 {
 7     for(int i=0;i<10;i++)
 8     {
 9         if( i == 4 ) // i = 4 的時候跳出迴圈
10         {
11             break;
12         }
13
14         cout << i << "\n";
15     }
16 }
```



continue 結束當前該次的迴圈

- 直接結束當前該次迴圈的執行，但沒有離開迴圈

```
 5 int main()
 6 {
 7     for(int i=0;i<10;i++)
 8     {
 9         if( i == 4 ) // i = 4 的時候結束該次迴圈，i++ 後繼續執行下一次
10         {
11             continue;
12         }
13
14         cout << i << "\n";
15     }
16 }
```



break 與 continue

- 使用 break 與 continue 控制迴圈時，每一次所指定的迴圈都是離他們最近的那一個，不會因為一個 break 或 continue 就對兩個以上的迴圈產生作用

```
 5 int main()
 6 {
 7     for(int i=0;i<10;i++)
 8     {
 9         for(int k=0;k<10;k++)
10         {
11             break;
12         }
13
14         cout << i << "\n";
15     }
16 }
```

兩層以上的迴圈練習題 - TOJ 104 星星樹

大家都知道，星星是長在樹上的。星星樹，自然就是上面長了星星的樹。

園丁JD3工作是專門維護長在資訊社社部的星星樹。工作包括每天澆水，拔雜草，防治蟲害（DEBUG）等等。當然，還有把星星樹定期修剪成整齊的形狀。

星星樹的樹枝發展相當有規律，呈現層狀。一棵好看的星星樹並且從樹的底端開始，往上呈現整齊的圓錐狀。為了保持穩定不傾倒，每一層的星星會修剪的比下面一層少正好兩顆。星星樹最頂端，則總是最閃亮的，那一顆的星星。

輸入說明
一個數字N，代表星星樹的高度。

輸出說明
星星樹。

範例輸入
```
4
```

範例輸出
```
   *
  ***
 *****
*******
```

兩層以上的迴圈練習題 - TOJ 104 星星樹

- 觀察可以發現如果輸入 N，我們就必須輸出 N 行
- 第 i 行會有 2i - 1 個星星
- 第 i 行輸出星星前必須先輸出 N - i 個空格

```
5 int main()
6 {
7     int n;
8     cin >> n;
9
10    for(int i=1;i≤n;i++)
11    {
12        for(int k=0;k<n-i;k++)
13        {
14            cout << " ";
15        }
16        for(int k=0;k<2*i-1;k++)
17        {
18            cout << "*";
19        }
20
21        cout << "\n";
22    }
23 }
```

範例輸入

4

範例輸出

```
   *
  ***
 *****
*******
```