基礎 C++ 程式設計導論 I

Introduction to Basic C++ Program Design I

課程內容

- C++ 基礎程式設計
  - 基本語法
  - 邏輯控制

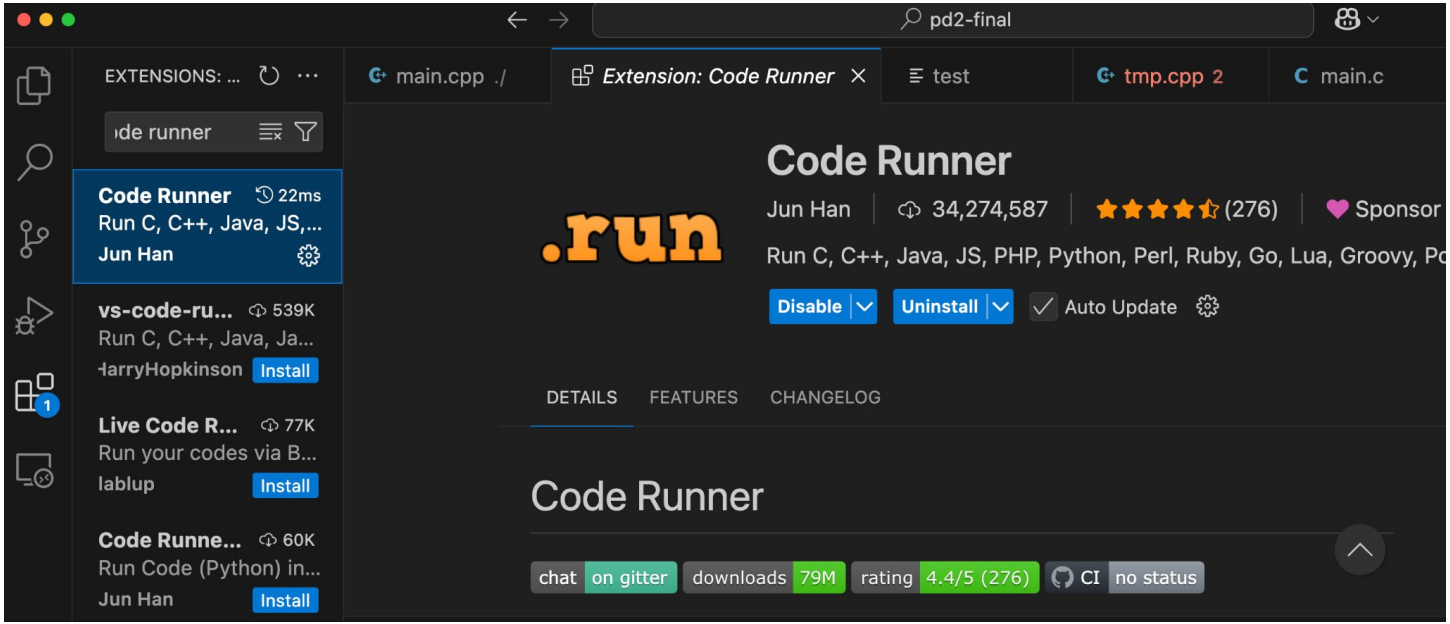
C++

- 出生於 1983 年 (現版本：C++23)
- 編譯式程式語言
- 程式語言裡面的老大哥
- C 語言延伸出來的一種語言

環境安裝 (TDM-GCC & vscode)

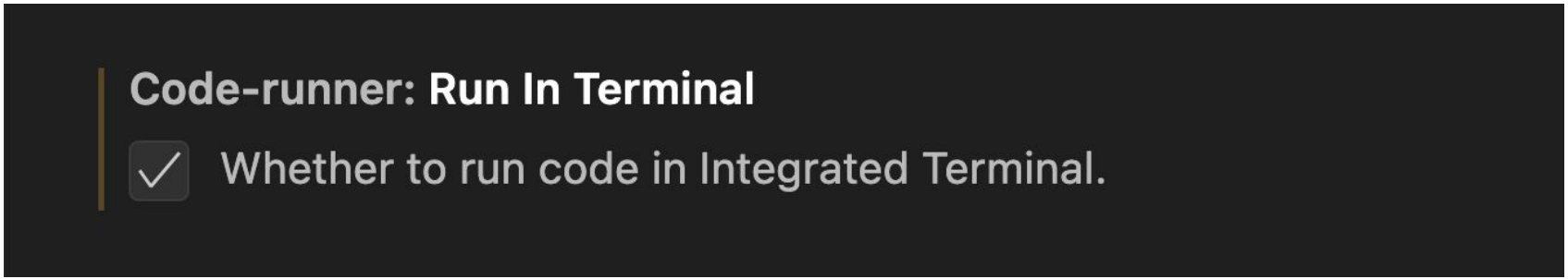
- TDM-GCC 官網 (MinGW-w64base)
- vscode 官網

vscode 插件安裝 (Code Runner)



Code Runner 設定

在設定把 Run in Termainal 打勾



變數與型態

Lecture 1

變數

- 會變的數字？
- 生活中常見的變數
  - 你的銀行存款
  - 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
  - 遊戲裡面的人物血量 -> 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)

常用型態

- 懶人包：
  - 整數
  - 浮點數 (小數)
  - 布林 (True or False)
  - 字串 (“Hello”)
  - 字元 (‘a’)

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)

- C++ 內有個比較進階的概念命名空間 (namespace)
- 命名空間用來區分不同群組之間的工具
  - Example:
    - 群組 a 有一個 solve(x,y) = x + y
    - 群組 b 有一個 solve(x,y) = x - y
    - 這時直接使用 solve 編譯器會不知道你指的是誰
    - 因此需要透過命名空間來區分 a::solve(x,y), b::solve(x,y)

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

```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 }

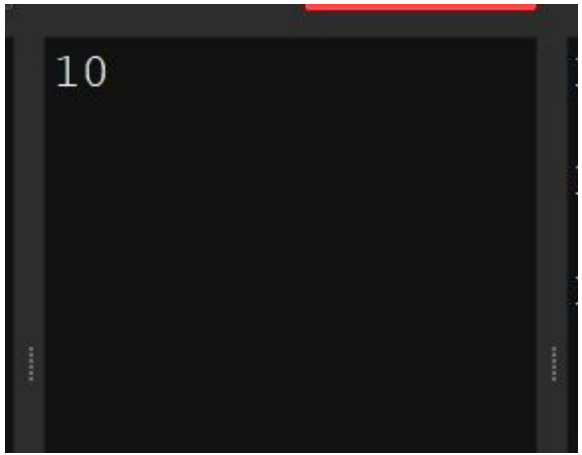
```
 7      for(int i=0;i<10;i++)
 8      {
 9          cout << i << "\n";
10      }
```

```
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

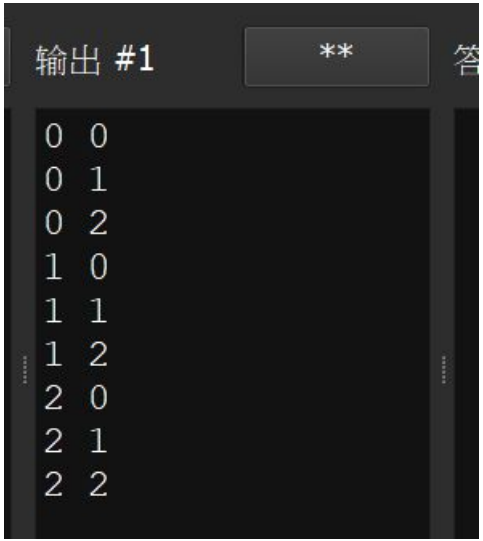
```
 5 int main()
 6 {
 7     int q = 10;
 8
 9     do
10     {
11         cout << q << "\n";
12     }while( q < 10 );
13 }
```



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

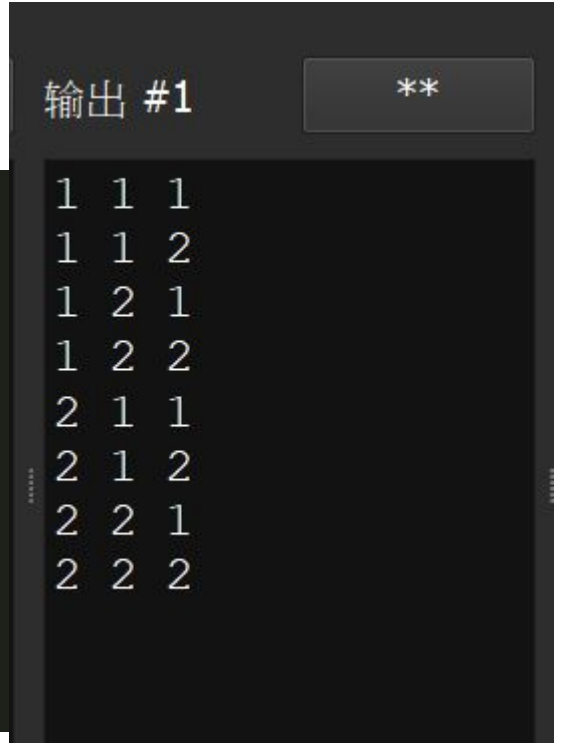
- 你會發現你可以把它理解成會執行 3 次裡面那一層 for 迴圈

```
 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！！

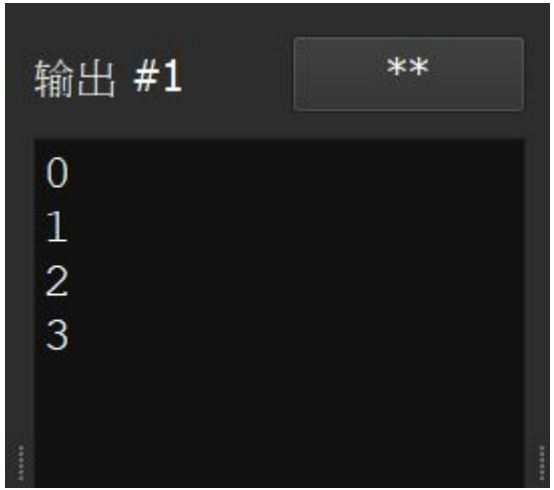
```
int main()
{
    for(int i=1;i<3;i++)
    {
        for(int k=1;k<3;k++)
        {
            for(int j=1;j<3;j++)
            {
                cout << i << " " << k << " " << j << "\n";
            }
        }
    }
}
```



break 跳出迴圈

- 直接跳出迴圈，不繼續在同一個迴圈執行

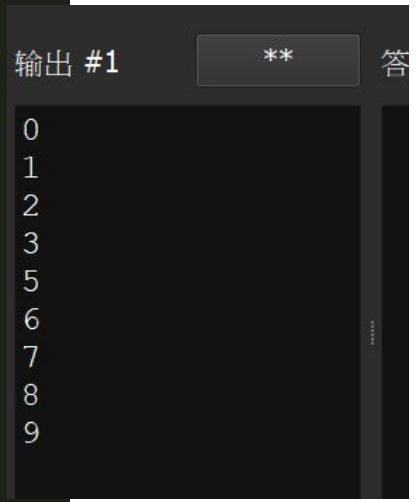
```
 5 int main()
 6 {
 7     for(int i=0;i<10;i++)
 8     {
 9         if( i == 4 ) // i = 4 的時候跳出迴圈
10         {
11             break;
12         }
13
14         cout << i << "\n";
15     }
16 }
```



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

```
int main()
{
    int n;
    cin >> n;

    for(int i=1;i≤n;i++)
    {
        for(int k=0;k<n-i;k++)
        {
            cout << " ";
        }
        for(int k=0;k<2*i-1;k++)
        {
            cout << "*";
        }

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

範例輸入

```
4
```

範例輸出

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

陣列 Array

- 當我想要一次宣告大量變數時，就會需要陣列的幫忙
- 陣列宣告：型態 名字[大小];

```
28      int a[20]; // 宣告 20 個名字叫做 a 的整數
```

陣列 Array

- 可以把陣列想像成很多個箱子，每一個箱子都有一個編號
- int a[20]; 宣告了 20 個名字叫做 a 的整數
- 那如果我想讓第 5 個箱子等於 100，應該怎麼做？
  - a[4] = 100;
  - 記得編號 (index 索引 值) 是從 0 開始的

陣列 Array

- 假設我有 n 個整數要輸入（1 <= n <= 100）

```
17 int a[100];
18
19 int main()
20 {
21     int n;
22     cin >> n;
23
24     for(int i=0;i<n;i++)
25     {
26         cin >> a[i];
27     }
28 }
```

陣列 Array

- 陣列初始化，如果想要一開始讓陣列所有元素為 0 可以這樣寫

```
2
3 int a[10] = {};
```

陣列 Array

- 一開始就給予陣列每一個位置的 value

```
int a[10] = {1,2,3,4,5,6,7,8,9,10};
cout << a[0] << "\n"; // 1
```

二維陣列

- 給你一個 n * m 的表格，每一個表格上的資訊都是整數
- 請你把所有資訊輸入近來
- 陣列好像只能輸入序列？

二維陣列

- 陣列是可以多維宣告的，一個中括號表示一個維度
- 因此我們可以宣告 int a[3][3]; 來表示 3 * 3 的表格
- 把資訊輸入近來


```cpp
int a[3][3];

int main()
{
    for(int i=0;i<3;i++)
    {
        for(int k=0;k<3;k++)
        {
            cin >> a[i][k];
        }
    }
}
```

二維陣列

- 二維陣列初始化

```
12      int a[3][3] = {
13          1 , 2 , 3,
14          4 , 5 , 6,
15          7 , 8 , 9,
16      };
17
18      cout << a[1][2] << "\n"; // 6
```

陣列的使用

- 如果陣列的大小可能會開很大，建議把宣告變數放到 main 函數的上面
  - 因為程式內部記憶體配置的關係
  - 如果

```
5 using namespace std;
6
7 int a[200005];
8
9 int main(void)
10 {
```

陣列的使用

- 用變數宣告陣列大小是非常不推薦的作法，很容易出問題
  - int a[n];
- 題目通常都會告訴你數字最多會輸入幾個，以那個數字作為依據即可
  - 如果題目說最多輸入 1 萬個數字就開 1 萬大小的陣列

字元與字串 char and string

- char & string
- stringstream
- ASCII

string 字串

- 字串就是一個被很多字元串接起來的東西
  - 因此在 C 中，字串的表示法為一個字元陣列
- 既然字串是一個字元陣列，那我們應該如何存取字串中某一個字元？
- 我們可以直接將字串當一般陣列使用，用索引指定字元位置
- 字串名稱.size() 這個操作可以取得字串的長度

```
string s = "Hello";

cout << s[0] << " " << s[2] << " " << s.size() << "\n"; // H l 5
```

string 字串加法

- 字串跟字串之間是可以做相加的
- 相加完的結果會是兩個字串接起來的結果
- 舉例：
  - Apple + Banana = AppleBanana

```
12      string a = "abc" , b = "OwO";
13
14      cout << a + b << "\n"; // abcOwO
```

ASCII Code

- 這個會輸出什麼東西？
- 為什麼字元可以被轉型成數字？依據是什麼？

```
19
20      cout << (int)'c' << "\n"; // 99
21
```

ASCII Code

16進制表示法：0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F、10 ...

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01		SOH
^B	2	02		STX
^C	3	03		ETX
^D	4	04		EOT
^E	5	05		ENQ
^F	6	06		ACK
^G	7	07		BEL
^H	8	08		BS
^I	9	09		HT
^J	10	0A		LF
^K	11	0B		VT
^L	12	0C		FF
^M	13	0D		CR
^N	14	0E		SO
^O	15	0F		SI
^P	16	10		DLE
^Q	17	11		DC1
^R	18	12		DC2
^S	19	13		DC3
^T	20	14		DC4
^U	21	15		NAK
^V	22	16		SYN
^W	23	17		ETB
^X	24	18		CAN
^Y	25	19		EM
^Z	26	1A		SUB
^[	27	1B		ESC
^\	28	1C		FS
^]	29	1D		GS
^^	30	1E	▲	RS
^-	31	1F	▼	US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂*

* ASCII 碼 127 具有代碼 DEL。在 MS-DOS 下，這個代碼與 ASCII 8 (BS) 的效果相同。DEL 代碼可以由 CTRL + BKSP 鍵產生。

ASCII Code

- 所以當你拿字元做加減乘除的運算的時候
- 程式會自動幫你把字元轉乘相對應的 ASCII Code 整數才做運算

ASCII Code

- 如果我現在想判斷一個數字有沒有某一個位數出現 2 或 3
  - 出現的話要把這一個位數 + x (1 <= x <= 5)
- 但我這個數字大到 10^100，使用 C++ 該怎麼做？
  - Example：12345 , x = 5
  - Answer：17845
- Hint: 可以掃過整個字串

ASCII Code

- 把整個字串掃過一次， 碰到 2 or 3 的時候就直接 +x
  - 字元 + 整數 x = 字元的 ASCII + x
  - 然後由於我們是存到字元裡面，所以新的字元的 ASCII Code 會等於 ASCII + x，程式會自動幫你轉回字元型態

```
20
21      int x;
22      string s;
23      cin >> s >> x;
24
25      for(int i=0;i<s.size();i++)
26      {
27          if( s[i] == '2' || s[i] == '3' ) s[i] += x;
28      }
29
30      cout << s << "\n";
```

stringstream

- #include <sstream>
- 一種處理字元的好工具
- 兩大功能
  - 切割字串
  - 型態轉型

stringstream 切割字串

- 一般陣列的題目都會先跟你說他會輸入幾個
- 但如果你遇到的題目一開始不會告訴你他要輸入幾個怎麼辦？
- 這個時候我們就只能先把整段字串先讀取進來了！

stringstream 切割字串

- stringstream 是一個工具，使用前必須先宣告

```
stringstream name;
```

stringstream 切割字串

- 接下來有兩種操作，假設我的 stringstream 的名字是 ss
  - ss << 變數; // 將變數放入 ss 裡面
  - ss >> 變數; // 將 ss 裡面的東西放入到變數

stringstream 切割字串

- 現在有一個字串 100 200 300，我想把他切成三份
- 每一份都是一個整數，依序存入 a[0],a[1],a[2]
- 如何用 stringstream 達成？
- 我們來解析一下吧！

```
21      string s;
22      getline(cin,s);
23
24      stringstream ss;
25
26      ss << s;
27
28      int number,a[3],idx = 0;
29
30      while( ss >> number )
31      {
32          a[idx] = number;
33          idx++;
34      }
```

getline(cin,s)

- 由於我們必須使用字串讀入，字串含有空白字元
- 所以我們必須使用 getline

```
21      string s;
22      getline(cin,s);
23
24      stringstream ss;
25
26      ss << s;
27
28      int number,a[3],idx = 0;
29
30      while( ss >> number )
31      {
32          a[idx] = number;
33          idx++;
34      }
```

stringstream ss; & ss << s;

- 宣告 stringstream
- 然後把我們要處理的字串丟入到這一個 stringstream 裡面

```
21      string s;
22      getline(cin,s);
23
24      stringstream ss;
25
26      ss << s;
27
28      int number,a[3],idx = 0;
29
30      while( ss >> number )
31      {
32          a[idx] = number;
33          idx++;
34      }
```

int number , a[3] , idx = 0;

- 宣告一個整數的 number，用來存 stringstream 切下來的東西
  - 因為我們切下來的東西是整數，所以 number 要是整數
- 宣告 a[3] 與 idx 來存取資料

```
21      string s;
22      getline(cin,s);
23
24      stringstream ss;
25
26      ss << s;
27
28      int number,a[3],idx = 0;
29
30      while( ss >> number )
31      {
32          a[idx] = number;
33          idx++;
34      }
```

while(ss >> number)

- stringstream 會依序把東西切下來然後放入 number
- 使用 while 迴圈的目的是：當沒有切到東西的時候才會停止

```
21      string s;
22      getline(cin,s);
23
24      stringstream ss;
25
26      ss << s;
27
28      int number,a[3],idx = 0;
29
30      while( ss >> number )
31      {
32          a[idx] = number;
33          idx++;
34      }
```

大功告成！

- 這麼一來我們的 a[0] = 100，a[1] = 200，a[2] = 300 了！
- 都是整數型態的哦！

```
21      string s;
22      getline(cin,s);
23
24      stringstream ss;
25
26      ss << s;
27
28      int number,a[3],idx = 0;
29
30      while( ss >> number )
31      {
32          a[idx] = number;
33          idx++;
34      }
```

stringstream 型態轉型

- 假設我有一個字串是 "1.414"，想要轉成 double 的型態？
- stringstream 也可以做到！寫法跟剛剛切字串很像

```
28      string s = "1.414";
29
30      stringstream ss;
31
32      ss << s;
33
34      double u;
35
36      ss >> u;
37
38      cout << u << "\n"; // 1.414
```

練習題

- 輸入字串 "10 20 30000" 請使用 stringstream 切割成 3 個整數
- 輸入字串 "5454"，請你使用 stringstream 轉成整數型態
- 呈上題，你能想到不使用 stringstream 的轉型方法　ㄇ？

兩個字串比大小

- 字串之間是可以比大小的
- 比較的規則使用 字典序，也就是 ASCII Code 的順序
- 我們來看看兩個字串是怎麼比較的吧！

兩個字串比大小

- 假設字串 a = "apple"，b = "apprl"，a 跟 b 誰比較大？
- 比較的時候會由左到右開始逐一字元比較，直到分出勝負
  - a = a (字元 1)
  - p = p (字元 2)
  - p = p (字元 3)
  - l < r (字元 4，分出勝負了，字串 b 比較大)

兩個字串比大小

- 即使字串長度不一樣也是可以比較的，規則一樣
  - Example：
    - abzzzzz < ac
    - gjosgjosjgojsojgosemnmn < z

兩個字串比大小 - 特殊情況

- 那 abb 跟 abba 誰比較大？
- 由於 abb 比到已經沒有東西了還沒分出勝負
- 因此 abba 比 abb 還要大

練習題

- 輸入兩個超過 long long 的正整數，請你寫一個程式判斷誰比較大
  - Hint: ASCII Code 字典序