

C++程式設計基礎

lesson 3

陳毅

上節回顧

- 條件控制
- 迴圈控制

條件控制

- 關係運算符 (>, <, >=, <=, ==, !=)
- 邏輯運算符 (&& (AND), || (OR), ! (NOT))
- if-else if-else
- switch

```
if ( 條件式1 ){  
    程式區塊1;  
    ...  
} else if ( 條件式2 ){  
    程式區塊2;  
    ...  
    ...  
} else if ( 條件式N ){  
    程式區塊N;  
    ...  
} else {  
    程式區塊N+1;  
}
```

```
switch ( 變數/運算式 ){  
    case 值1:  
        程式敘述1;  
        ...  
        break;  
    case 值2:  
        程式敘述2;  
        ...  
        break;  
    ...  
    case 值N:  
        程式敘述N;  
        ...  
        break;  
    default:  
        程式敘述N+1;  
        ...  
        break;  
}
```

迴圈控制

- for迴圈
- while迴圈
- do-while迴圈
- continue敘述
- break敘述

```
for (起始式; 判斷式; 運算式){  
    程式敘述;  
    ...  
}
```

```
while (判斷式){  
    程式敘述;  
    ...  
}
```

```
do {  
    程式敘述;  
    ...  
} while (判斷式);
```

練習

- 以亂數擲10000次骰子，並分別列出出現1、2、3、4、5、6點的次數。(Dice.cpp)
- 輸入兩個數字，求最大公因數，與最小公倍數。
- 列出1~100中的所有質數。
- 列出九九乘法表。

本節概要

- 使用者函數（自定義函數）
 - 函數的架構
 - 引數（參數）的傳遞
 - 區域與全域變數
 - 遞迴函數
- 陣列
 - 一維陣列的使用
 - 字串表示
 - 二維與多維陣列
 - 陣列的傳遞
- 補充：運算子介紹(三元運算子、sizeof運算子)、基礎讀檔

使用者函數（自定義函數）

好處

- 將重複功能的部分，使用函數替代，增加程式碼的可利用性。
- 將複雜的程式切分成數個較小且簡單的問題，在維護和修改上會更為方便。
- 程式語言中的函數並不單單只是數學函數，它真正有用的是，可以實作並包裝一個功能，讓程式設計師能有效率地去拼組出一個複雜的程式。
- 黑盒子
 - 與他人合作開發時，可將自己負責的部分進行封裝，別人只需要知道輸入與輸出，不需要知道實作方式，即可使用。

函數的架構

函數雛型(Declaration)

用來告訴編譯器，這個程式會有哪些函數。

型態 函數名稱(引數1型態, 引數2型態, ...);

函數宣告(Definition)

用來定義一個函數實際的執行內容。

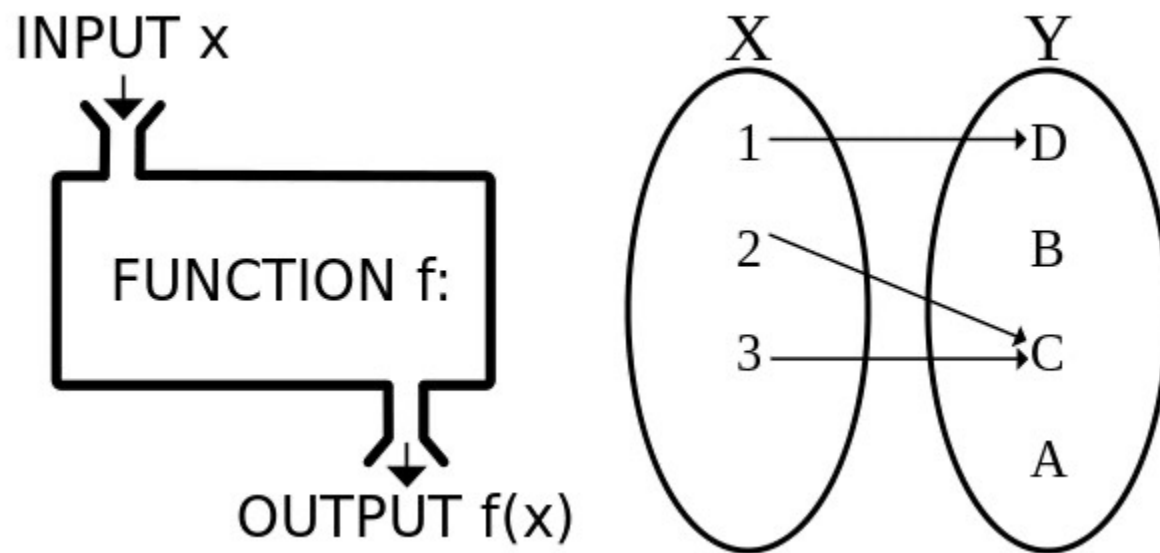
```
型態 函數名稱(引數1, 引數2, ...){  
    程式敘述;  
    ...  
}
```

```
#include <iostream>  
using namespace std;  
  
void I_AM_A_FUNCTION(int, int);  
  
int main(){  
  
    I_AM_A_FUNCTION(1, 2);  
  
    return 0;  
}  
  
void I_AM_A_FUNCTION(int a, int b){  
    cout << a << " " << b << endl;  
    return;  
}
```

函數的架構

- **return**敘述

- 可以將變數傳回呼叫它的函數內。
- 傳回的值必須與函數宣告時的型態相同。
- 在函數中，一旦執行到**return**敘述，程式將直接結束這個函數的執行。



練習

- 請寫一個函數，用以計算整數的次方。
 - 輸入：int a, int n
 - 輸出： a^n
- 請寫一個函數，找出四個整數中的最大值。
 - 輸入：int a, int b, int c, int d
 - 輸出：max(a,b,c,d)
- 請寫一個函數，在螢幕上輸出10行「Hello, world!」。
 - 輸入：無
 - 輸出：無

引數的傳遞

- 每一個函數都是獨立的，一般來說，函數只了解自己程式區塊的資料，並不認識函數外的任何變數。
- 當函數要修改到外部資料時，就必須將資料以引數的方法傳遞進函數。
- 傳遞方法分兩種：Call-by-Value、Call-by-Reference。
 - Call-by-Value：將所傳的變數資料複製一份，儲存在函數所宣告的自訂變數中。
 - Call-by-Reference：將所傳的變數之記憶體位址傳遞進函數，可以直接修改變數資料。

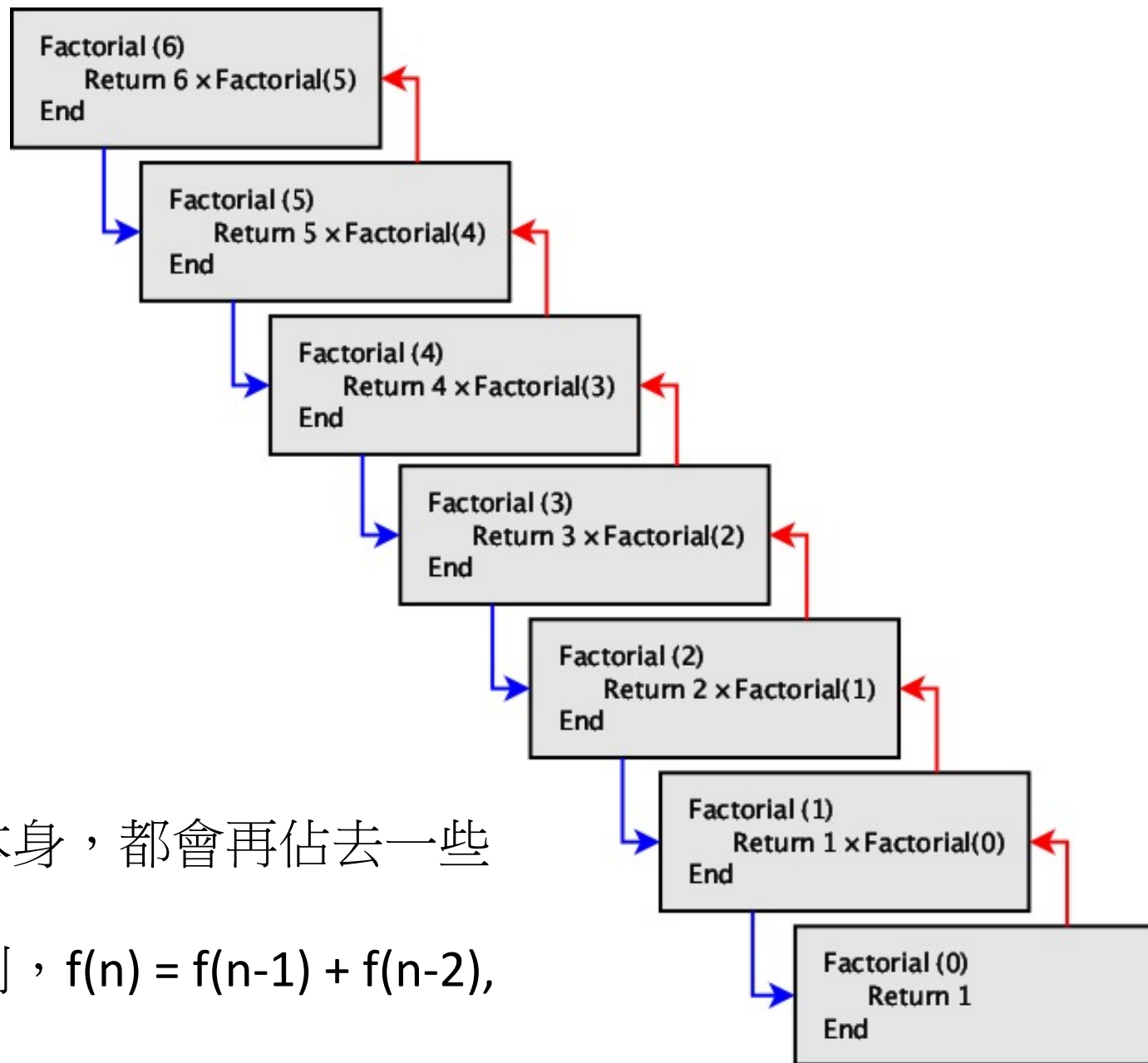
區域與全域函數

- 每一個變數都有自己的生命週期(scope)，當變數被宣告時，也決定了這個變數的存在範圍。
 - 區域變數
 - 只存在於程式區塊中，當離開了該程式區塊（離開大括號），該變數就會被銷毀。
 - 全域變數
 - 在程式一開始宣告，作用範圍在整個程式，不論在哪個區塊，都可以直接存取全域變數。
 - 在程式結束時，才會被銷毀。
 - 有時容易產生誤用，因此盡量不要使用。

遞迴函數

- 一個函數直接或間接的呼叫函數本身，稱作遞迴函數。
 - 直接：在程式敘述內直接呼叫函數本身。
 - 間接：程式敘述內呼叫其他函數，該函數又呼叫原先的函數。
- 許多數學公式都以遞迴的方式定義，例如： $n! = n * (n-1)!$ 。

遞迴函數



- 缺點

- 效率不佳，每次呼叫一次函數本身，都會再佔去一些記憶體空間。
- 容易重複計算，例如：費氏數列， $f(n) = f(n-1) + f(n-2)$, $f(n-1) = f(n-2) + f(n-3)$

陣列

一維陣列的使用

- 宣告一維陣列的語法如下：`型態 陣列名稱[元素個數];`

- 宣告一個10個元素的整數陣列，陣列名稱為array：

```
int array[10];
```

| | | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| array[0] | array[1] | array[2] | array[3] | array[4] | array[5] | array[6] | array[7] | array[8] | array[9] |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|

- 宣告時，給予初始值：

```
int array1[10] = {0, 2, 3, 7, 2, 3, 5, 7, 0, 1};  
int array2[5] = {1, 2, 3};  
int array3[] = {4, 3, 2, 1};
```

字串表示

- 一個字串是由很多字元組合而成，因此可以將字串用字元陣列表示。
- 字元陣列以 **'\0'** 為結尾。

| | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
| 'H' | 'e' | 'l' | 'l' | 'o' | ' ' | 'W' | 'o' | 'r' | 'l' | 'd' | '\0' |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|

- C++提供了<string>的header，裡面定義一個專門用來處理字串的物件。

二維和多維陣列

- 宣告二維以上陣列的語法：

型態 陣列名稱[元素個數(第1維)][元素個數(第2維)][元素個數(第3維)]...[元素個數(第n維)];

- 二維陣列

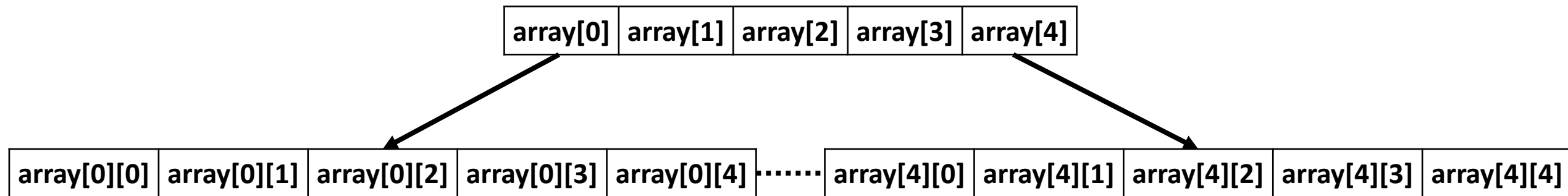
| | | | | |
|-------------|-------------|-------------|-------------|-------------|
| array[0][0] | array[0][1] | array[0][2] | array[0][3] | array[0][4] |
| array[1][0] | array[1][1] | array[1][2] | array[1][3] | array[1][4] |
| array[2][0] | array[2][1] | array[2][2] | array[2][3] | array[2][4] |
| array[3][0] | array[3][1] | array[3][2] | array[3][3] | array[3][4] |
| array[4][0] | array[4][1] | array[4][2] | array[4][3] | array[4][4] |

二維和多維陣列

- 宣告二維以上陣列的語法：

型態 陣列名稱[元素個數(第1維)][元素個數(第2維)][元素個數(第3維)]...[元素個數(第n維)];

- 二維陣列



陣列的傳遞

- 陣列也可以當作引數傳遞進函數中。
- 比較特殊的是，傳遞陣列時，並不會將整個陣列複製一份，而是將陣列的第一個元素的記憶體位址傳進函數中，所以在函數內修改陣列的值是有效的。

練習

- 學生成績系統
 - 撰寫一個程式，接受10個學生的成績，儲存在陣列中，之後計算10個學生的平均成績。
- 計算方程式
 - 撰寫一個程式，接受使用者輸入一元 n 次方程式的係數，使用陣列儲存，然後接受使用者輸入的變數值，計算方程式的值。
- 階乘計算（大數）
 - 電腦是現今人類一個重要的計算輔助工具，科學家時常利用電腦來計算一些方程式，其中可能牽扯到非常大的數字，假設今天我們想要計算 $50!$ ，不論何種型態的變數都無法大到可以儲存這個天文數字，因此程式設計師想出了一種方法，利用陣列表示數字，只要記憶體空間夠大，便可儲存非常大的整數。請撰寫一個程式，利用陣列儲存數字，計算 $50!$ 的大小。

運算子介紹

三元運算子

- 由兩個符號組成，分別是「？」與「：」。

```
int a, b, c;  
a = b > c ? b : c;
```

- 三元運算子會先看「？」前的條件式是否為真。
 - 若為真則指定「：」前的值給變數
 - 若為假則指定「：」後的值給變數
- 上面的這個例子，會比較**b**和**c**的大小，將數值比較大的值指定給變數**a**

sizeof 運算子

- 用來計算程式中變數所佔用的記憶體大小，這個運算子在動態配置記憶體時有很大的用處。

```
int a;  
char b;  
float c;  
double d;  
bool e;  
cout << "int: " << sizeof(a) << endl;  
cout << "char: " << sizeof(b) << endl;  
cout << "float: " << sizeof(c) << endl;  
cout << "double: " << sizeof(d) << endl;  
cout << "bool: " << sizeof(e) << endl;
```

基礎讀檔

ifstream類別 (input file stream)

- 定義於<fstream>中，因此若要使用，需要#include <fstream>。
- 用法：

```
ifstream myFile;  
myFile.open("file.txt", ios::in);  
if (!myFile){ // myFile.is_open()  
    cout << "開啟檔案失敗" << endl;  
    return 0;  
}  
myFile >> tmp;  
cout << tmp << endl;  
myFile.close();
```

宣告讀檔物件

開啟檔案

若開啟檔案失敗，則結束程式

讀取資料（用法與cin相同）

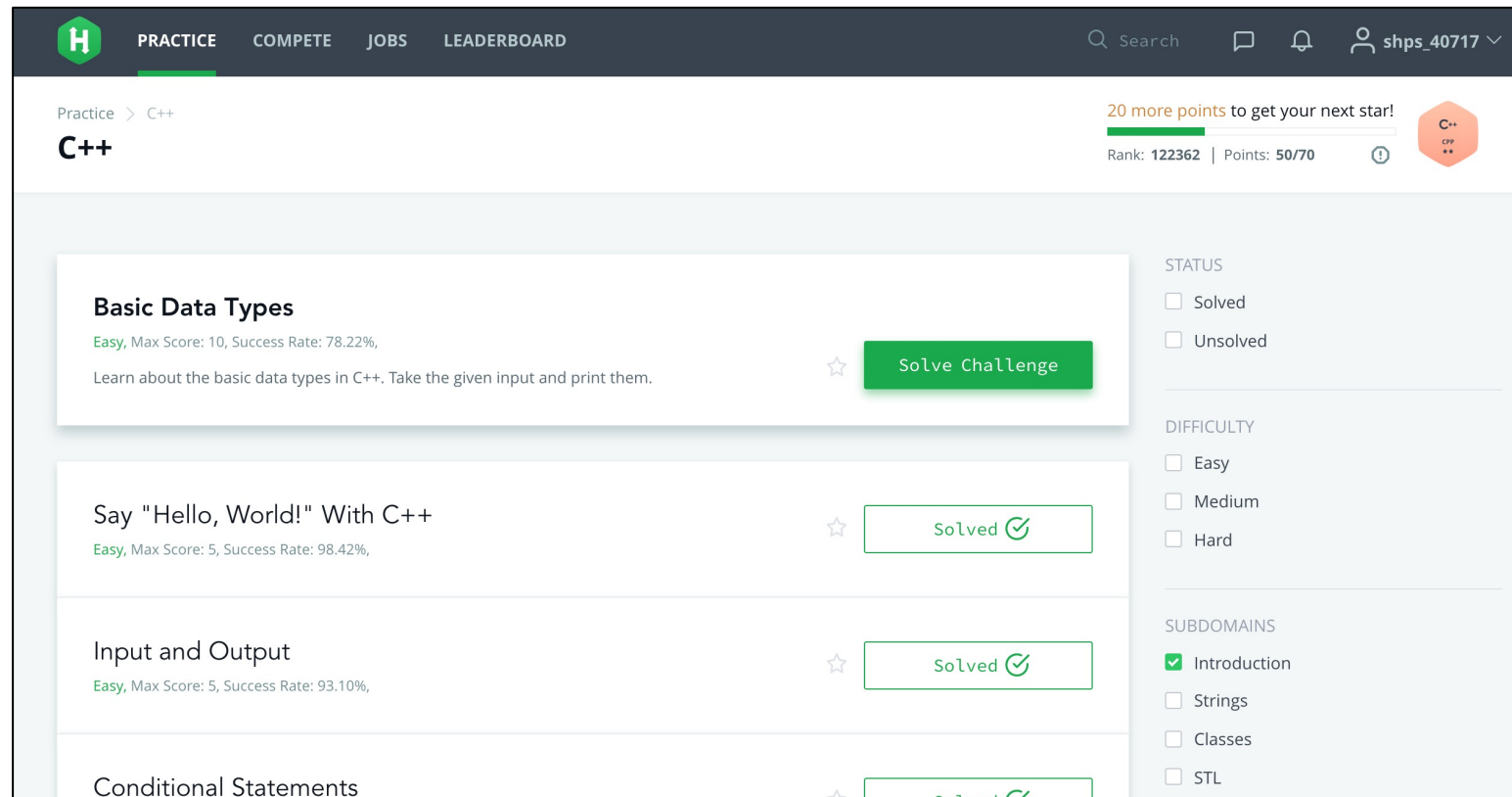
關閉檔案

挑戰

- 製作OOXX遊戲
 - 遊戲規則
 - 兩個玩家，在3x3的方格中，依序填入O和X，先連成一條線的人獲勝。
 - 程式設計思路
 - 需要一個變數來儲存現在的盤面。（hint: 可使用二維陣列）
 - 需要處理玩家的輸入，例如：要將O和X填入哪個格子中。（hint: 格式化輸入）
 - 需要更新遊戲盤面，並判斷玩家的輸入是否正確（hint: 函數）
 - 需要讓玩家看到遊戲盤面。（hint: 函數、格式化輸出）
 - 玩家每動一步，就要更新一次盤面，並判斷遊戲是否結束了。（hint: 函數）
 - 進階挑戰
 - 將遊戲擴增為「五子棋」
 - 更改遊戲規則，製作「踩地雷遊戲」

練習

- [HackerRank](#) (Practice -> C++ -> Introduction)
 - Say “Hello, World!” with C++
 - Input and Output
 - Conditional Statements
 - For Loop
 - Functions
 - Arrays Introduction



The screenshot shows the HackerRank C++ practice page. The top navigation bar includes links for PRACTICE, COMPETE, JOBS, and LEADERBOARD, along with a search bar and user profile (shps_40717). The main content area displays a list of challenges under the C++ section. The challenges listed are:

- Basic Data Types**: Easy, Max Score: 10, Success Rate: 78.22%. Description: Learn about the basic data types in C++. Take the given input and print them. Status: Unsolved.
- Say "Hello, World!" With C++**: Easy, Max Score: 5, Success Rate: 98.42%. Status: Solved.
- Input and Output**: Easy, Max Score: 5, Success Rate: 93.10%. Status: Solved.
- Conditional Statements**: Status: Solved.

The right sidebar contains filters for STATUS (Solved, Unsolved), DIFFICULTY (Easy, Medium, Hard), and SUBDOMAINS (Introduction, Strings, Classes, STL). The 'Introduction' subdomain is currently selected.

練習

- Zerojudge
 - a038 : 數字翻轉
 - a015 : 矩陣的翻轉
 - a016 : 數獨(SUDOKU)
 - d244 : 一堆石頭
 - b515 : 摩斯密碼-商競103
 - a248 : 新手訓練~陣列運用
 - a021 : 大數運算
- Reading: 課本Ch6~Ch7-4