高二至高三的C++歷程

新竹女中 周語泠

檔案目錄

00 / 選課動機

從高二必修的程式設計課以及參加許多與程式有關的營隊，我便發現了自己對撰寫程式有著濃厚的興趣！
希望自己成功選到這門課，透過選修課學到更進階的C++技巧，以程式解決生活中看似複雜的問題，同時訓練邏輯與問題解決的能力。
更期許自己大學能就讀與程式設計相關的科系，活用高中所學並精進自我，一窺電腦科學的奧妙。

01 / 高二課程目錄

- **變數運算**
- **流程控制 - if**
- **迴圈 - for & while**
- **一維與二維陣列**
- **字串**
  - 作業精選例題 + 程式碼截圖
  - 例題撰寫邏輯
  - 例題AI 修改建議
- **函式與遞迴**

註. 僅挑選字串例題做精華檢討與反思！
以上目錄為高二上所有學過的章節

字串精選例題

在回文比賽中，參賽者輸出越多組回文字串，
就能夠得到越多獎勵
其中回文定義是『由左至右』或『由右至左』
讀法都一樣的句子

因為參賽者非常踴躍
請撰寫程式協助主辦方判斷輸入的字串是否回文

輸入說明	輸出說明
字串 s	Yes 或 No
範例輸入 level	**範例輸出** Yes

原程式碼

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

int main ()
{
    string s;
    cin >> s;

    for(int i = 0; i < s.length (); ++i)
    {
        if('A' <= s[i] && s[i] <= 'Z')
            s[i] = (s[i]-'A') + 'a';
    }

    int dill = 0, N = s.length();
    for(int i = 0; i < N; ++i)
    {
        if(s[i] != s[N-i-1])
            ++dill;
    }
    if (dill ==0)   cout << "Yes" << endl;
    else            cout << "No" << endl;
}
```

AI 建議程式碼

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

bool is_palindrome(const string& s)
{
    int len = s.length();
    for (int i = 0; i < len / 2; ++i)
    {
        if (s[i] != s[len - 1 - i])
        {
            return false;
        }
    }
    return true;
}

int main()
{
    string s;
    cin >> s;

    if (is_palindrome(s))
    {
        cout << "Yes" << endl;
    }
    else
    {
        cout << "No" << endl;
    }
}
```

字串精選例題 - 撰寫邏輯

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

int main ()
{
    string s;
    cin >> s;

    for(int i = 0; i < s.length (); ++i)
    {
        if('A' <= s[i] && s[i] <= 'Z')
            s[i] = (s[i]-'A') + 'a';
    }

    int dill = 0, N = s.length();
    for(int i = 0; i < N; ++i)
    {
        if(s[i] != s[N-i-1])
            ++dill;
    }
    if (dill ==0)   cout << "Yes" << endl;
    else            cout << "No" << endl;
}
```

將大寫字母轉換為小寫

- 遍歷字串 s，檢查是否為大寫字母（範圍 'A' 到 'Z'）。
- 若是，則**轉換為小寫**：
  - s[i] - 'A' 計算字母與 'A' 的距離（例如 'C' - 'A' = 2）
  - + 'a' 將結果轉換回對應的小寫字母（例如 2 + 'a' = 'c'）
  - 這樣能夠確保回文判斷時不受大小寫影響。

檢查是否為回文

- dill 用來計算不同的字符數量，初始值為 0。
- 使用 for 迴圈從 0 遍歷到 N-1：
  - s[i] 是從左邊的字母。
  - s[N-i-1] 是從右邊對應的字母。
  - 如果兩者不同，則 dill++ 記錄一次不相等。

字串精選例題 - 利用AI 修改建議

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

bool is_palindrome(const string& s)
{
    int len = s.length();
    for (int i = 0; i < len / 2; ++i)
    {
        if (s[i] != s[len - 1 - i])
        {
            return false;
        }
    }
    return true;
}

int main()
{
    string s;
    cin >> s;

    if (is_palindrome(s))
    {
        cout << "Yes" << endl;
    }
    else
    {
        cout << "No" << endl;
    }
}
```

- **使用 bool is_palindrome(const string& s) 的原因**

string& 不會複製整個字串，避免額外的記憶體開銷，更提高效率。

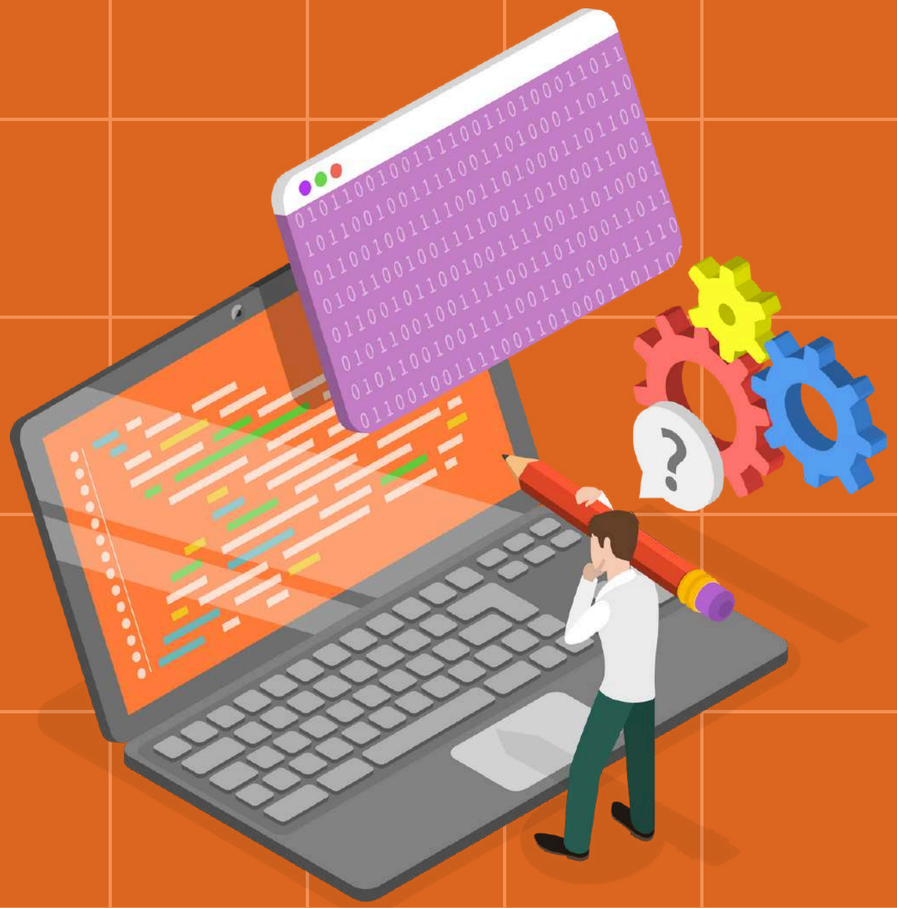
- **為什麼要用 const string& 而不是 string？**

假設用 string s，每次呼叫 is_palindrome() 時，整個字串都會被複製一份，這在處理大字串時會增加效能開銷。
而使用 const string& 能避免不必要的拷貝，只傳遞字串的**引用**。
且 const 限制了函式內部對 s 的修改，確保輸入字串不會被改變。

簡單比喻

- string s → 將整本書影印一份給函式，函式讀影本，不影響原本的書，但影印很花時間。
- const string& s → 你把書的書名給函式，函式直接讀原書，不影響原本內容，也不佔用額外空間。

02 / 高三課程目錄



- **資料結構 - queue & stack**
  - queue 作業精選例題 + 程式碼截圖
  - 例題改進
- **動態陣列 - vector**
  - 作業精選例題 + 程式碼截圖
  - 撰寫過程錯誤&檢討
- **內建排序 - sort**
- **時間複雜度**
- **圖論 - dfs & bfs**
  - bfs 作業精選例題 + 程式碼截圖
  - 例題撰寫邏輯
  - 例題AI 修改建議
- **動態規劃 - dp**

佇列精選例題

玩家會依序進入等待區直到主辦方開設房間才能進入房間對戰。房間有人數上限，如果等候區人數大於上限，則會依照玩家進入房間的順序取上限值人數進入房間。

請模擬連線過程，過程中指令輸入格式如下：

1 K: 編號 K 的玩家進入等待區

2 K: 讓至多 K 人進到對戰房間的房間，若等待區不滿 K 人，則所有等待區的人都可進入

輸入說明	輸出說明
第一行為正整數 N 接著有 N 行指令，指令格式如題目所述	在每次產生房間時，印出可進房間的玩家

範例輸入	範例輸出
7 1 20 1 30 1 10 1 50 2 3 1 60 2 5	enter room 20 30 10 enter room 50 60

原程式碼

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

int main()
{
    int N, op, K;
    cin >> N ;

    queue <int> q;

    for (int i = 0; i < N; ++i)
    {
        cin >> op >> K;
        if (op == 1) q.push(K);
        else
        {
            cout << "enter room";
            for (int j = 0; j < K; ++j)
            {
                if (q.size() > 0)
                {
                    cout << " " << q.front() ;
                    q.pop();
                }
            }
                    cout << endl;
        }
    }
}
```

佇列精選例題 - 改進

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

int main()
{
    int N, op, K;
    cin >> N ;

    queue <int> q;

    for (int i = 0; i < N; ++i)
    {
        cin >> op >> K;
        if (op == 1) q.push(K);
        else
        {
            cout << "enter room";
            for (int j = 0; j < K; ++j)
            {
                if (q.size() > 0)
                    {
                        cout << " " << q.front() ;
                        q.pop();
                    }
            }
                    cout << endl;
        }
    }
}
```

程式碼改進

- 防止輸入錯誤時 <u>op 沒有 K 值</u>：

op == 2 時應確保 K 存在，而不是直接讀取。

- 修正 if (q.size() > 0) 為 if (!q.empty())：

更直觀且符合標準用法。

- 優化 cout：

使用 vector 暫存要輸出的數字，減少 cout 操作，提高效率。

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

int main()
{
    int N, op, K;
    cin >> N;

    queue<int> q;

    for (int i = 0; i < N; ++i)
    {
        cin >> op >> K;
        if (op == 1)
        {
            q.push(K);
        }
        else if (op == 2)
        {
            cout << "enter room";
            while (K-- > 0 && !q.empty())
            {
                cout << " " << q.front();
                q.pop();
            }
            cout << endl;
        }
    }
}
```

動態陣列精選例題

現在共有 N 個可訂餐廳，
每個餐廳皆有名稱 S、美味度 V 和價錢 C
只有當 CP 值(即美味度/價錢)大於 1 時，才會被列入口袋名單

請印出被列入口袋名單的餐廳總數、餐廳名稱
以及因為報復性消費，也請印出在口袋名單的所有餐廳都消費的消費總額

輸入說明	輸出說明	範例輸入	範例輸出
第一行為正整數 N 接著有 N 行，每行有 名稱 S、美味度 V、價錢 C	由上至下，印出口袋名單中總數、餐廳名稱、總額	4 KFC 100 60 MOS 50 100 Macdonald 80 80 BurgerKing 100 50	2 KFC BurgerKing 110

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

int main()
{
    int N, V, C, total = 0;
    string S;
    vector <string> restaurant;

    cin >> N;
    for (int i= 0; i < N; ++i)
    {
        cin >> S >> V >> C;
        if (V > C)
        {
            restaurant.push_back(S) ;
            total += C ;
        }
    }
    cout << restaurant.size() << endl;

    for (int i = 0; i < restaurant.size(); ++i)
    {
        cout << restaurant[i] << endl;
    }

    cout << total << endl;
}
```

動態陣列精選例題 - 撰寫過程錯誤&檢討

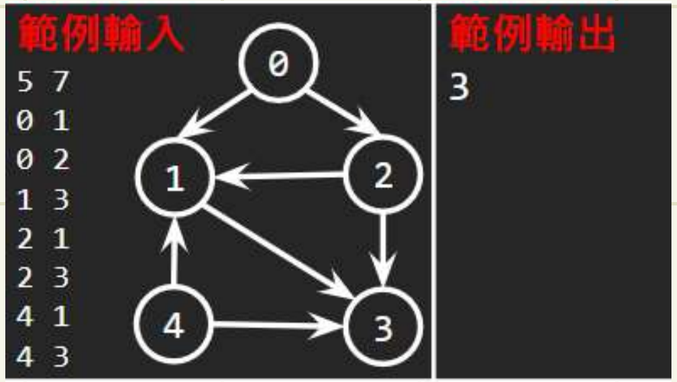
- int N, V, C, total = 0; // 利用 total 紀錄被放進名單的餐廳總和，需先歸零。
- vector <string> restaurant; // 宣告 vector 餐廳口袋名單，記得是要放string，因restaurant是字串！
- if (V > C)
  // 注意在C++中：
  若**除出來的數字不是整數則會無條件捨去顯示**，所以不建議使用除法，而是用**乘法**表示 V/C > 1
  {
  restaurant.push_back(S) ; // 將餐廳名稱放進 vector 中
  total += C ;
  // 更新被放進名單的餐廳總額 而+=其實就是把前面已經放進去的餐廳金額再加後面進來的，較 "total = total + C" 好因為執行較快速
  }
- for (int i = 0; i < restaurant.size(); ++i) // 在已放進去 restaurant 的陣列中印出**尚未超過此陣列大小**的編號

BFS 精選例題

對任意兩人，只要有 直接 或 間接 的共通好友就有機會認識彼此，進而成為朋友
同樣地，在原本不認識的兩人變成好友以後將有機會介紹更多朋友

已知有 N 個人(編號 0, 1, ..., N-1) 和 M 組單向朋友關係 (u, v)
請計算對於編號 0 的人，最多有多少人能夠和他成為朋友

輸入說明	輸出說明
第一行有正整數 N 和 M 代表 N 個點 M 條邊 接著有 M 行單向邊 (u, v)	能夠和編號 0 的人 成為朋友的數量



原程式碼

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

int main()
{
       int N, M, u, v, d[10] = {0};
    cin >> N >> M;

    vector<int> edge[10];

    for(int i= 0; i < M; ++i)
    {
        cin >> u >> v;
        edge[u].push_back(v);
    }
    for(int i =0; i < N; ++i)
        d[i] = 99999;

    queue <int> q;
    q.push(0);
    d[0] = 0;

    while(!q.empty())
    {
        int cur = q.front();
        q.pop();

        for(int nxt:edge[cur])
        {
            if(d[nxt] > d[cur] +1)
            {
                q.push(nxt);
                d[nxt] = d[cur] + 1;
            }
        }
    }

    int ans = 0;
    for(int i = 0; i < N; ++i)
    {
        if(d[i] !=99999)
            ++ans;
    }
    cout << ans-1 << endl;
}
```

AI 建議程式碼

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

const int INF = 1e9; // 定義一個極大值

int main()
{
    int N, M, u, v;
    cin >> N >> M;

    vector<vector<int>> edge(N); // 動態大小的鄰接表
    vector<int> d(N, INF); // 儲存最短路徑長，初始值為無窮大

    for (int i = 0; i < M; ++i) // 圖的輸入
    {
        cin >> u >> v;
        edge[u].push_back(v);
    }

    queue<int> q;
    q.push(0);
    d[0] = 0; // 設定起點距離為 0

    while (!q.empty())
    {
        int cur = q.front();
        q.pop();

        for (int nxt : edge[cur])
        { // 遍歷當前節點的所有相鄰節點
            if (d[nxt] > d[cur] + 1)
            { // 發現更短的路徑時更新
                d[nxt] = d[cur] + 1;
                q.push(nxt);
            }
        }
    }

    int ans = 0;
    for (int i = 0; i < N; ++i)
    {
        if (d[i] != INF) // 計算所有可達節點數量
            ++ans;
    }

    cout << ans - 1 << endl; // 扣掉自己
}
```

BFS 精選例題 - 撰寫邏輯

<圖論> 重點！

BFS (廣度優先) 第一次走一定保證是最短的路徑

但DFS (深度優先) 無法保證任何事情！！！

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

int main()
{
       int N, M, u, v, d[10] = {0};      // d[i] 記錄起點到i的「最短」路徑長
    cin >> N >> M;

    vector<int> edge[10];

    for(int i= 0; i < M; ++i) // 圖的儲存
    {
        cin >> u >> v;
        edge[u].push_back(v);
    }
    for(int i =0; i < N; ++i) // 預設極大值
        d[i] = 99999;

    queue <int> q;
    q.push(0);
    d[0] = 0;
    while(!q.empty()) // 只要 queue 中仍有待走點
    {
        int cur = q.front(); // pop 目前位置 i
        q.pop();

        for(int nxt:edge[cur]) // 嘗試走訪 i 可走到的下個點 j
        {
            if(d[nxt] > d[cur] +1) // 如果找到更好的解，則更新
            {
                q.push(nxt);
                d[nxt] = d[cur] + 1;
            }
        }
    }

    int ans = 0;
    for(int i = 0; i < N; ++i)
    {
        if(d[i] !=99999)
            ++ans;              // -1是為了要扣掉自己跟自己不能當朋友啦
    }
    cout << ans-1 << endl;
}
```

此處的扣一非常重要！容易遺忘！

不要忘記大前提：是跟「別人」交朋友！

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

const int INF = 1e9; // 定義一個極大值

int main()
{
    int N, M, u, v;
    cin >> N >> M;

    vector<vector<int>> edge(N); // 動態大小的鄰接表
    vector<int> d(N, INF); // 儲存最短路徑長，初始值為無窮大

    for (int i = 0; i < M; ++i) // 圖的輸入
    {
        cin >> u >> v;
        edge[u].push_back(v);
    }

    queue<int> q;
    q.push(0);
    d[0] = 0; // 設定起點距離為 0

    while (!q.empty())
    {
        int cur = q.front();
        q.pop();

        for (int nxt : edge[cur])
        { // 遍歷當前節點的所有相鄰節點
            if (d[nxt] > d[cur] + 1)
            { // 發現更短的路徑時更新
                d[nxt] = d[cur] + 1;
                q.push(nxt);
            }
        }
    }

    int ans = 0;
    for (int i = 0; i < N; ++i)
    {
        if (d[i] != INF) // 計算所有可達節點數量
            ++ans;
    }

    cout << ans - 1 << endl; // 扣掉自己
}
```

BFS 精選例題 - 利用AI 修改建議

- **避免使用固定大小陣列 (d[10])**

改用 vector<int> d(N, INF);，確保支援更大範圍的 N。

- **定義 const int INF = 1e9;**

1e9 (1000000000) 比 99999 更適合作為無限大值。

- **使用 vector<vector<int>> edge(N);**

vector<int> edge[10]; 只能支援最多 10 個節點，改成動態大小的 vector<vector<int>> 可以解決結點可能不夠多的問題。

- **queue<int> 直接操作 cur**

cur 變數仍然有意義，q.front() 可直接拿來用。

queue<int> 用來存放等待處理的節點，確保 BFS 逐層擴展。

cur = q.front(); 取得當前要處理的節點。

for (int nxt : edge[cur]) 訪問所有鄰居，確保 BFS 正確執行。

這種**先進先出**的特性確保了 BFS 會先探索較短路徑的節點，因此常用於最短路徑搜尋和樹/圖的遍歷。

03 - 01 / TOI練習賽

圖書館 (Library) (點按查看題目)

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

int main()
{
    int N, S, D, total = 0;
    vector <int> book;
    cin >> N;
    for (int i = 0; i < N; ++i)
    {
        cin >> S >> D;
        if (D > 100)
        {
            book.push_back (S);
            total += (D-100)*5;
        }
    }
    sort (book.begin(), book.end());

    if (book.empty())
    {
        cout << "0" << endl;
    }

    else
    {
        for (int i=0 ; i < book.size(); ++i)
    {
        cout << book[i] << " ";
    }

    cout << endl << total << endl;
    }
}
```

```
5
40 21
83 182
15 102
51 203
90 88
15 51 83
935

Process returned 0 (0x0)   execution time : 37.476 s
Press any key to continue.
```

- 迴圈處理每本書：
  - 這段程式碼會執行 N 次，每次讀入一組 S 和 D 值。
  - 篩選條件：如果 D > 100，則：
    - 將 S（書的價格）存入 book 容器。
    - 計算額外費用 (D - 100) * 5，並加到 total。
- 使用 sort() 函式將 book 內的價格從小到大排序。
- 若 book 為空（即沒有符合條件的書），則直接輸出 0，代表沒有書需要額外費用。
- 若 book 不為空：
  - 依序輸出所有符合條件的書籍價格（已排序）。
  - 換行後輸出 total，表示額外的總費用。

03 - 02 / APCS

最少相異字母 (點按查看題目)

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int countLetters(const string& word)
{
    vector <bool> freq(26, false);  // 記錄 A-Z 是否出現過
    for (char ch : word)
    {
        freq[ch - 'A'] = true;
    }
    return count(freq.begin(), freq.end(), true);  // 計算有幾個字母出現過
}

int main ()
{
    int N;
    cin >> N;
    vector <string> words(N);

    for (int i = 0; i < N; ++i)
    {
        cin >> words[i];
    }

    string best_word;
    int min_unique_chars = 27;

    for (const string& word : words)
    {
        int unique_count = countLetters(word);

        if (unique_count < min_unique_chars || (unique_count == min_unique_chars && word < best_word))
        {
            best_word = word;
            min_unique_chars = unique_count;
        }
    }

    cout << best_word << endl;
}
```

```
3
ABBCAAB
AABBACC
AAPPCCSS
AABBACC

Process returned 0 (0x0)   execution time : 4.895 s
Press any key to continue.
```

- CountLetters **函式**：
  - 用 vector<bool> freq(26, false) **來記錄字母是否出現過**。
  - 遍歷字串時，將對應的 freq[ch - 'A'] 設為 true。
  - 使用 count(freq.begin(), freq.end(), true) 來**計算 true 的數量，即相異字母數**。
- 範圍型迴圈：
  - ch 是一個字母，將 A~Z 轉換成 0~25 的索引，方便儲存到 letters 陣列（或 vector<bool>）。
    - 例如：'H' - 'A' = 72 - 65 = 7 → 對應索引 7
  - letters[ch - 'A'] = true;
    - letters 是一個 bool 陣列，初始時所有值都是 false。
    - 將該字母對應的索引位置設為 true，表示這個字母有出現過。

04 / 學習心得與反思

從高二開始接觸程式設計，我對於電腦科學的世界充滿好奇，卻也對程式的概念不夠熟悉，面對錯誤訊息經常感到挫折。隨著課程的深入，我發現程式設計不僅僅是撰寫程式碼，而是一種**邏輯推理與問題解決的能力**。我開始理解演算法與資料結構的應用，並學會以更有系統的方式解決問題。這段歷程不僅**提升了我的邏輯思維**，也**培養了面對問題時的分析能力**，也讓我對**資訊工程**領域產生了更濃厚的興趣。

- **提升的關鍵點**

過程中，我發現「**拆解問題**」與「**測試驗證**」是提升程式能力的重要關鍵。起初，我經常想著要一次呵成寫完整個程式，但結果往往是錯誤百出，甚至難以除錯。後來我學會**將問題分解成小部分**，**逐步驗證**各個功能是否正確，大幅減少了錯誤發生的機率。此外，與同學討論不同的解法使我意識到程式**並非只有單一解法**，讓我有勇氣**跳脫固有思維**，也學習到多樣的思考方式。

- **挑戰與突破**

我曾經因對函式與布林陣列的理解不夠深入，解決某些題目時遇到瓶頸。例如在練習APCS試題(p.16)時，有許多用法是課堂未學習的，但這並非逃避考驗的藉口。我透過**閱讀相關資料**並**觀察字母索引表**才真正理解「編號相減」能將電腦語言轉換為人類的ABC，若無法跳脫人類語言的**思考邏輯**應當是很難想到這種解法的。但多虧這次經驗讓我知道廣泛查詢資料、**勇於面對錯誤的精神和跳脫思考框架**才是程式撰寫的真諦，期許自己之後能進一步**提升程序執行效率**，並培養面對各種複雜問題時能不怯步的勇氣。

04 / 學習心得與反思

程式設計如何培養資工領域的重要能力

- **邏輯思考與問題拆解能力**

競程題目中我學會將一個**龐大的問題拆解成更小的部分**，運用條件判斷、遞迴等方式來構築解決方案。例如一個長度為 N 的陣列需找出其中「出現次數最多的數字」，如果有多個則選字典序最小者。可使用 unordered_map <char, int> 遍歷map統計每個字母的出現次數，也能確保時間複雜度最優化。這樣的拆解方式讓問題變得有條理，也幫助我理解如何選擇合適的資料結構提升運行效率。不僅讓我在競賽與學習中獲益，也培養了**面對複雜問題時的冷靜分析能力**。

- **數據處理與優化能力**

在學習演算法時，我開始意識到「**效率**」的重要性。例如當初我嘗試暴力解法解決「最短路徑」問題，卻發現程式運行時間過長。促使我深入學習Dijkstra與Floyd-Warshall演算法，並思考如何優化時間與空間複雜度。這種數據分析與優化的思維，正是**資工領域解決大規模數據問題時不可或缺的能力**。

- **程式語言與人工智慧接軌**

近年來，隨著人工智慧與機器學習快速發展，我也開始接觸Python，並嘗試學習基本的 AI 演算法，如線性回歸與神經網路。我發現 AI 的核心仍然建立在**數據結構與演算法**之上，而我在程式設計課程中學到的數據處理能力，對於**未來學習 AI 及大數據分析**有極大的幫助。

04 / 學習心得與反思

- **人工智慧與機器學習**

程式設計已不僅僅是一種技術，而是一種**解決問題的思維模式**。這門課程不只讓我學會了如何撰寫程式，更讓我理解到資訊技術如何與現實世界結合。過去我更參加了陽明交通大學的**人工智慧人才培育計畫**，程式設計課結束我將兩者的知識結合，並若在未來有機會就讀資訊工程相關學系，我期望自己能在機器學習與演算法等層面上擁有自己**獨特的見解**。

- **學習程式設計的價值與未來展望**

回顧這段學習歷程，我不僅掌握了程式語言的基礎知識，也培養了解決問題的能力。程式設計讓我學會用「**系統化思維**」來分析問題，並透過**不斷試錯**來找到最佳解決方案。更重要的是，我理解了資訊科技如何改變世界，並**啟發了我對人工智慧與資工領域的興趣**。未來我希望能持續精進演算法能力，並深入學習AI與大數據分析，讓程式設計成為我探索科技時代的最佳工具，為未來的智慧社會貢獻一份力量。這段學習歷程不僅是一種技能的累積，更是思維與態度的轉變。程式設計讓我學會**在錯誤中成長**，透過系統化的方式解決問題，也讓我對未來的科技時代充滿期待。

THANK YOU

BY 新竹女中 周語泠