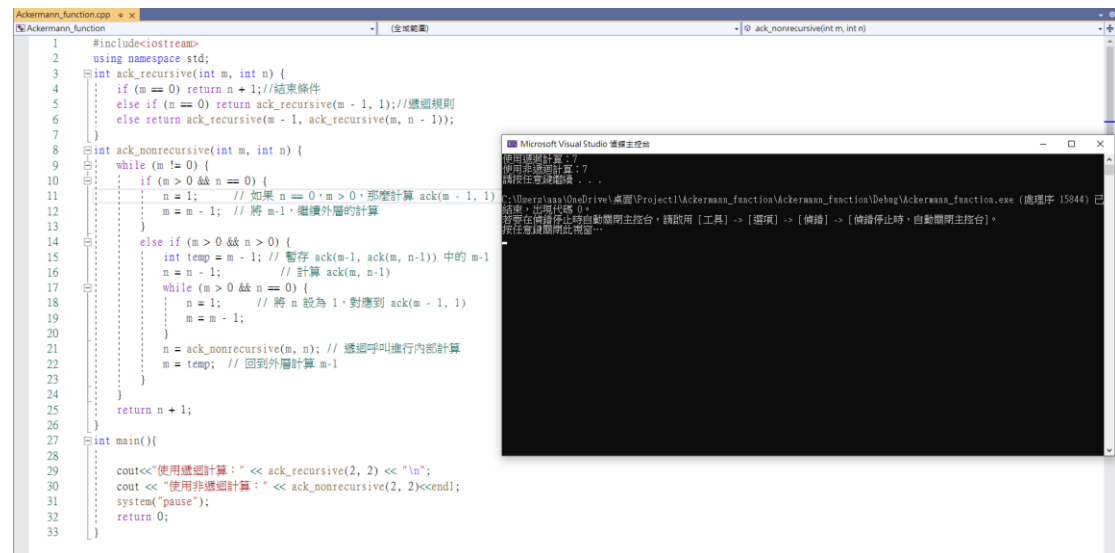


HW1

第一題



The screenshot shows the Visual Studio IDE with a C++ file named `Ackermann_function.cpp`. The code implements two versions of the Ackermann function: `ack_recursive` and `ack_nonrecursive`. The recursive version is a simple base case and recursive call. The non-recursive version uses a while loop to simulate the recursive process. The `main` function calls both functions with arguments (2, 2) and prints the results. A debug window is open, showing the execution of `ack_nonrecursive(2, 2)` and the output of the program.

```
1 #include <iostream>
2 using namespace std;
3 int ack_recursive(int m, int n) {
4     if (m == 0) return n + 1; // 結束條件
5     else if (n == 0) return ack_recursive(m - 1, 1); // 遞迴規則
6     else return ack_recursive(m - 1, ack_recursive(m, n - 1));
7 }
8
9 int ack_nonrecursive(int m, int n) {
10    while (m != 0) {
11        if (m > 0 && n == 0) {
12            n = 1; // 如果 n == 0, m > 0, 那麼計算 ack(m - 1, 1)
13            m = m - 1; // 將 m-1, 繼續外層的計算
14        }
15        else if (m > 0 && n > 0) {
16            int temp = m - 1; // 暫存 ack(m-1, ack(m, n-1)) 中的 m-1
17            n = n - 1; // 計算 ack(m, n-1)
18            while (m > 0 && n == 0) {
19                n = 1; // 將 n 設為 1, 對應到 ack(m - 1, 1)
20                m = m - 1;
21            }
22            n = ack_nonrecursive(m, n); // 遞迴呼叫進行內部計算
23            m = temp; // 回到外層計算 m-1
24        }
25    }
26    return n + 1;
27 }
28
29 int main() {
30     cout << "使用遞迴計算: " << ack_recursive(2, 2) << "\n";
31     cout << "使用非遞迴計算: " << ack_nonrecursive(2, 2) << endl;
32     system("pause");
33     return 0;
34 }
```

Microsoft Visual Studio 偵錯主控台

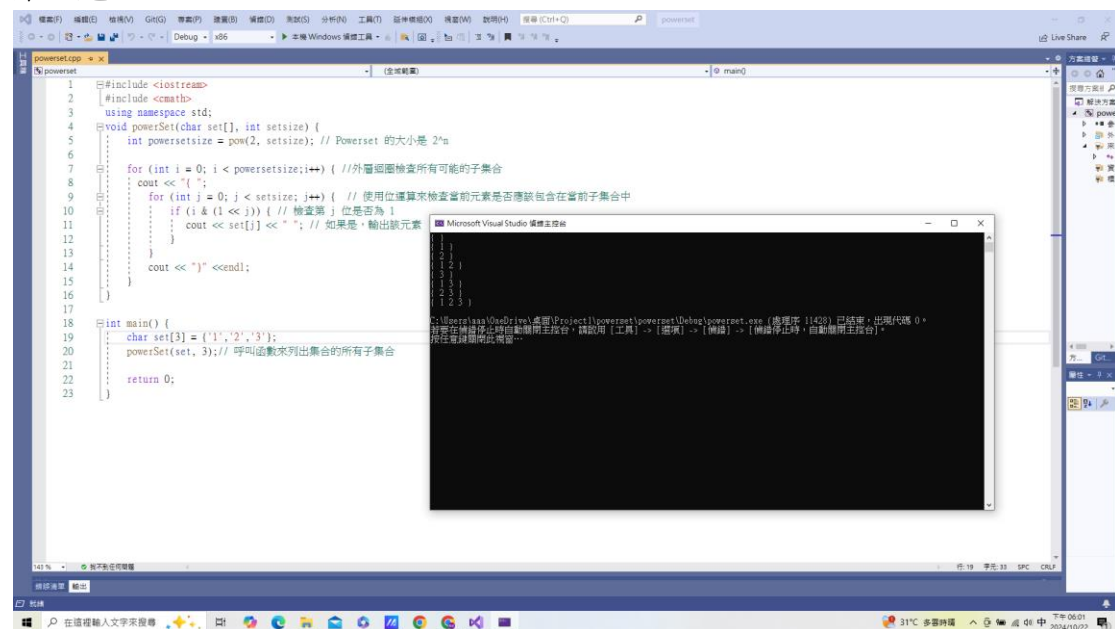
```
使用遞迴計算: 7
使用非遞迴計算: 7
請按任意鍵繼續...
```

C:\Users\aaa\OneDrive\桌面\Project\Ackermann_function\Ackermann_function\Debug\Ackermann_function.exe (線序 15844) 已結束, 出現代碼 0。

若要查看儲存本專案的變數主控台, 請啟用 [工具] -> [變數] -> [偵錯] -> [偵錯停止時, 自動關閉主控台]。

按任意鍵關閉此視窗...

第二題



The screenshot shows the Visual Studio IDE with a C++ file named `powerSet.cpp`. The code implements a function `powerSet` that generates all possible subsets of a given set. It uses a nested loop to iterate through all possible combinations of elements. The `main` function calls `powerSet` with a set containing the characters '1', '2', and '3'. A debug window is open, showing the execution of `powerSet` and the output of the program.

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 void powerSet(char set[], int setsize) {
5     int powersetsize = pow(2, setsize); // Powerset 的大小是 2^m
6     for (int i = 0; i < powersetsize; i++) { // 外層迴圈檢查所有可能的子集
7         cout << "{ ";
8         for (int j = 0; j < setsize; j++) { // 使用位運算來檢查當前元素是否應該包含在當前子集中
9             if (i & (1 << j)) { // 檢查第 j 位是否為 1
10                 cout << set[j] << " "; // 如果是, 輸出該元素
11             }
12         }
13         cout << "}" << endl;
14     }
15 }
16
17 int main() {
18     char set[] = {'1', '2', '3'};
19     powerSet(set, 3); // 呼叫函數來列出集合的所有子集
20     return 0;
21 }
22
23 }
```

Microsoft Visual Studio 偵錯主控台

```
{ }
{ 1 }
{ 2 }
{ 3 }
{ 1 2 }
{ 1 3 }
{ 2 3 }
{ 1 2 3 }
```

C:\Users\aaa\OneDrive\桌面\Project\powerSet\powerSet\Debug\powerSet.exe (線序 11428) 已結束, 出現代碼 0。

若要查看儲存本專案的變數主控台, 請啟用 [工具] -> [變數] -> [偵錯] -> [偵錯停止時, 自動關閉主控台]。

按任意鍵關閉此視窗...