

解題說明：

用遞迴和非遞迴做 ackermann 的程式

效能分析：

```
if (m == 0) return n + 1;  
if (n == 0) return a(m - 1, 1);  
else return a(m - 1, a(m, n - 1));  
ackermann 的用法
```

測試與驗證：

Input:1 2

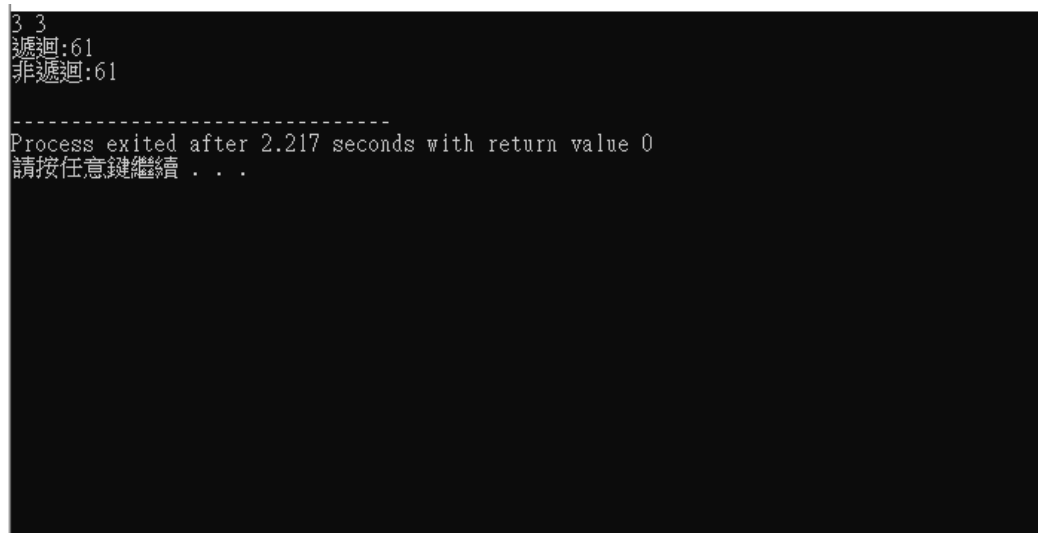
Output:4



```
Microsoft Visual Studio 偵錯主控台  
1 2  
遞迴:4  
非遞迴:4  
C:\Users\Lenovo\OneDrive\Desktop\HW1-1\Debug\HW1-1.exe (處理序 20256) 已結束，出現代碼 0。  
若要在偵錯停止時自動關閉主控台，請啟用 [工具] -> [選項] -> [偵錯] -> [偵錯停止時，自動關閉主控台]。  
按任意鍵關閉此視窗...
```

Input:3 3(最多到 3 3)

Output:61



```
3 3  
遞迴:61  
非遞迴:61  
-----  
Process exited after 2.217 seconds with return value 0  
請按任意鍵繼續 . . .
```

申論及心得：

依照 Ackermann 函式定義用遞迴的方式來寫

1 如果 $m=0$ 要回傳 $n+1$

2 如果 $n=0$ 要回傳 $a(m-1, 1)$ 。

3 如果都不符合前 2 條，就回傳 $a(m-1, a(m, n-1))$

要先宣告一個函數例如： $a(\text{int } m, \text{int } n)$ ，然後在把上述條件放進這個函數中，就可以達成 Ackermann 所要的功能了。

再來是非遞迴的 Ackermann 函式，也是一樣跟著他要的條件打

1 如果 $m=0$ 要回傳 $n+1$

2 如果 $n=0$ $n=1$ ， $m=m-1$

3 如果都不符合前 2 條 $n=b(m, n-1)$ ， $m=m-1$ ；

這些條件要放在無窮迴圈裡才可以一直運作直到數值 return 完，才會結束，所以這些條件要放在 while(1) 裡面做運作，非遞迴的輸出結果就會等於遞迴的輸出結果。

在實作非遞迴版本時，我將原本的遞迴轉換成一個 while 迴圈，並用紙張模擬遞迴的過程。這部分的困難在於如何處理每一次遞迴的運算，尤其是在遞迴中又要在遞迴的時候，必須要正確的模擬出原本的答案。這讓我對遞迴有了更多了解，並幫助我思考如何在實作中減少不必要的重複計算。

遞迴與非遞迴各自的優缺點也變得更加清晰，遞迴的程式碼簡潔、直觀，非遞迴雖然它的程式碼多了一點，但也更容易了解程式，增加程式的可讀性。

這次的實作不僅讓我更了解 Ackermann 函式的運作原理，還熟悉了我對遞迴轉換非遞迴過程的理解。我學會了如何依照需求選擇使用遞迴或非遞迴的方式解決問題，並且學到如何將看似複雜的遞迴邏輯轉換為迴圈的技巧。這些經驗對我未來的程式設計有很大的幫助。