HW2

(deadline 2019/10/15)

手寫題:

4. One of the methods to calculate the square root of a number is Newton's method. The formula for Newton's method is shown in Figure 2-15. Write the pseudocode for a recursive algorithm to compute a square root using Newton's method. Verify your algorithm by using it to manually calculate the following test cases: `squareRoot` (5, 2, 0.01) and `squareRoot` (4, 2, 0.01). *Note*: in the formula, tol is an abbreviation for *tolerance*.

$$\text{squareRoot (num, ans, tol)} = \begin{bmatrix} \text{ans} & \text{if } |\text{ans}^2 - \text{num}| \le \text{tol} \\ \text{squareRoot(num, (ans}^2 + \text{num)/(2} \times \text{ans), tol)} & \text{otherwise} \end{bmatrix}$$

6. Ackerman's number, used in mathematical logic, can be calculated using the formula shown in Figure 2-17. Write a recursive algorithm that calculates Ackerman's number. Verify your algorithm by using it to manually calculate the following test cases: `Ackerman(2, 3)`, `Ackerman(2, 5)`, `Ackerman(0, 3)`, and `Ackerman(3, 0)`.

$$\text{Ackerman } (m, n) = \begin{bmatrix} n + 1 & \text{if } m = 0 \\ \text{Ackerman } (m-1, 1) & \text{if } n = 0 \text{ and } m > 0 \\ \text{Ackerman } (m-1, \text{Ackerman } (m, n-1)) & \text{otherwise} \end{bmatrix}$$

FIGURE 2-17 Ackerman Formula for Problem 6

程式題:

18. The combination of *n* objects, such as balls in a basket, taken *k* at a time can be calculated recursively using the combination formula. (See Exercise 5.) Write a C function to calculate the number of combinations present in a number. Test your function by computing and printing `C(10, 3)`.

5. The combination of *n* objects, such as balls in a basket, taken *k* at a time can be calculated recursively using the formula shown in Figure 2-16. This is a useful formula. For example, several state lotteries require players to choose six numbers out of a series of possible numbers. This formula can be used to calculate the number of possible combinations, *k*, of *n* objects. For example, for 49 numbers, there are `C(49, 6)`, or 13,983,816, different combinations of six numbers. Write a recursive algorithm to calculate the combination of *n* objects taken *k* at a time.

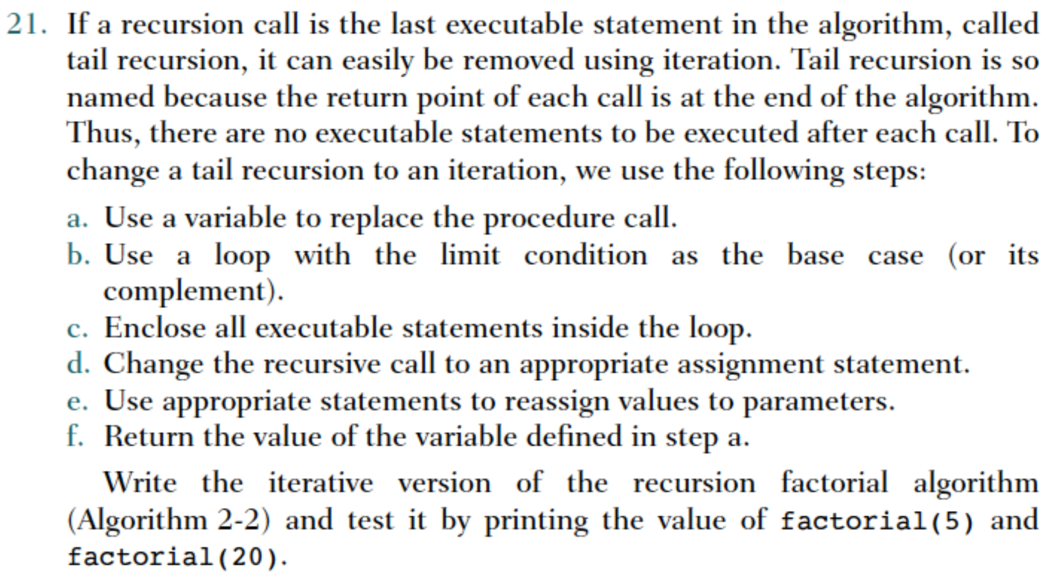
$$C(n, k) = \begin{bmatrix} 1 & \text{if } k = 0 \text{ or } n = k \\ C(n, k) = C(n-1, k) + C(n-1, k-1) & \text{if } n > k > 0 \end{bmatrix}$$

FIGURE 2-16 Selection Algorithm for Exercise 5

<Exercise 5：參考用，不需作答!!!>

＊請讓程式可以用 Command line 輸入，輸出為一個值~

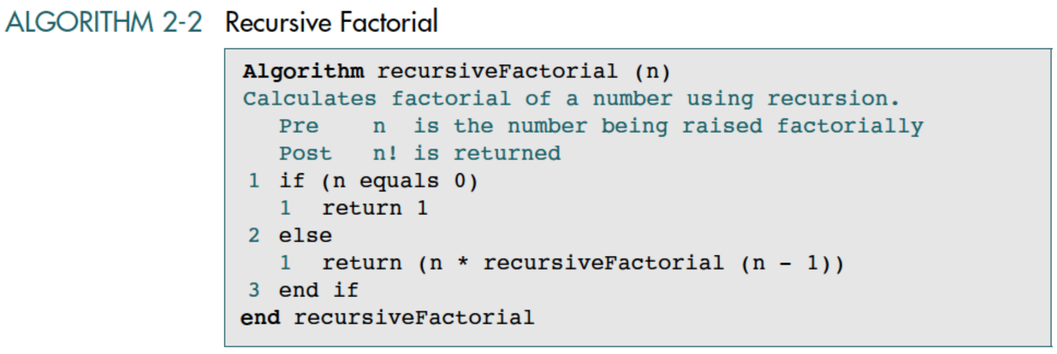
例：輸入 3,2　　輸出 3



21. If a recursion call is the last executable statement in the algorithm, called tail recursion, it can easily be removed using iteration. Tail recursion is so named because the return point of each call is at the end of the algorithm. Thus, there are no executable statements to be executed after each call. To change a tail recursion to an iteration, we use the following steps:

    a. Use a variable to replace the procedure call.
    b. Use a loop with the limit condition as the base case (or its complement).
    c. Enclose all executable statements inside the loop.
    d. Change the recursive call to an appropriate assignment statement.
    e. Use appropriate statements to reassign values to parameters.
    f. Return the value of the variable defined in step a.

    Write the iterative version of the recursion factorial algorithm (Algorithm 2-2) and test it by printing the value of `factorial(5)` and `factorial(20)`.



ALGORITHM 2-2 Recursive Factorial

```
Algorithm recursiveFactorial (n)
Calculates factorial of a number using recursion.
   Pre    n  is the number being raised factorially
   Post   n! is returned
1  if (n equals 0)
   1  return 1
2  else
   1  return (n * recursiveFactorial (n - 1))
3  end if
end recursiveFactorial
```

＊請遵循步驟 a-f 利用 Iterative version of the recursion 直接輸出 5!與 20!的值~