HW2(deadline:2014/10/9)

2. Consider the following algorithm:

```
algorithm fun2 (x, y)
1 if (x < y)
   1  return -3
2 else
   1  return (fun2 (x - y, y + 3) + y)
3 end if
end fun2
```

What would be returned if `fun2` is called as

a. `fun2 (2, 7)`?
b. `fun2 (5, 3)`?
c. `fun2 (15, 3)`?

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)`.

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

FIGURE 2-17 Ackerman Formula for Problem 6

16. Write a recursive C function to calculate the square root of a number using Newton's method. (See Exercise 4.) Test your function by printing the square root of 125, 763, and 997.

22. Write the iterative version of the Fibonacci series algorithm using the hints given in Project 21. Note that step c in Project 21 will be different because factorial uses two recursive calls in the last statement.

(以下 4、21 為 16、22 題，所提到的參考，不用做)

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*.

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

FIGURE 2-15 Newton's Method for Exercise 4

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)`.