บทที่ 5

ฟังก์ชันเวียนเกิดและฟังก์ชันทำซ้ำ

ในบทที่ 3 เราได้ศึกษาวิธีการสร้างฟังก์ชันในโปรแกรม Scilab เพื่อการใช้งานทั่ว ๆ ไปแล้ว แต่ในบางกรณี เราต้องการฟังก์ชันที่ต้องอาศัยการทำชุดคำสั่งเดิมซ้ำ ๆ กันหลายรอบ ตัวอย่างเช่น ต้องการสร้างฟังก์ชันเพื่อพิมพ์ข้อความเดียวกันหลาย ๆ ครั้ง โดยมีจำนวนครั้งตามที่กำหนดให้ เป็นต้น ในบทนี้เราจะศึกษาเทคนิคการสร้างฟังก์ชันที่มีลักษณะดังกล่าว 2 เทคนิค ได้แก่ การสร้าง**ฟังก์ชันเวียนเกิด** (recursive function) และการสร้าง**ฟังก์ชันทำซ้ำ** (iterative function)

อย่างไรก็ตาม เทคนิคทั้งสองมีหลักการทำงานที่แตกต่างกันมาก และในบางครั้งก็ส่งผลให้ฟังก์ชันของเราใช้เวลาในการทำงานมากน้อยแตกต่างกันด้วย ดังนั้นนักเขียนโปรแกรมจึงต้องรู้จักเลือกใช้เทคนิคดังกล่าวให้เหมาะสม เพื่อให้ฟังก์ชันที่ออกแบบมานั้น สามารถทำงานได้อย่างถูกต้อง รวดเร็ว และใช้ทรัพยากรของระบบโดยประหยัด

5.1 ฟังก์ชันเวียนเกิด

บทนิยาม 5.1.1

ฟังก์ชันเวียนเกิด (recursive function) คือ ฟังก์ชันซึ่งอาศัยหลักการทำงานโดยการเรียกใช้ตัวเองซ้ำ แต่เปลี่ยน input ให้มีขนาดเล็กลงกว่าเดิม

ตัวอย่าง 5.1.2 พิจารณาฟังก์ชันต่อไปนี้

```
function [y] = recursive_f(x)
    if x <= 0 then
        // output สำหรับกรณีฐาน (base case)
        y = 1;
    else
        // เรียกใช้ตัวเองซ้ำ แต่เปลี่ยน input ให้เล็กลง
        y = 2 + recursive_f(x-2)
    end
endfunction
```

จงหาค่าของ output ที่ได้จากการเรียกใช้ `recursive_f(7)`

<u>วิธีทำ</u>

ข้อสังเกต

1. โดยทั่วไปแล้ว การที่เราจะเรียกใช้ตัวแปรหรือฟังก์ชันใด ๆ ได้นั้น เราจะต้องสร้างตัวแปรหรือฟังก์ชันดังกล่าวไว้ก่อนแล้วเสมอ จะเห็นได้ว่าในกรณีของฟังก์ชันเวียนเกิดนั้นก็ยังคงไม่ขัดกับหลักการนี้ เนื่องจากว่า Scilab จะทราบถึงความมีอยู่ (existence) ของฟังก์ชันเวียนเกิดได้นับตั้งแต่ที่ประมวลผลส่วนหัว (header) ของฟังก์ชันแล้ว
2. ในฟังก์ชันเวียนเกิดทุกตัว จะต้องมีการกำหนด output สำหรับ**กรณีฐาน** (base cases) เสมอ เพื่อป้องกันมิให้ฟังก์ชันเวียนเกิดทำงานไปเรื่อย ๆ โดยไม่มีที่สิ้นสุด ทั้งนี้ กรณีฐานอาจจะมีเพียง 1 กรณีหรือมากกว่านั้นก็ได้

ตัวอย่าง 5.1.3 จงสร้างฟังก์ชันเวียนเกิดชื่อ `MySequence` เพื่อหาค่าของ s_n เมื่อ n เป็นจำนวนเต็มบวกที่กำหนดให้ โดยที่

$$s_n = \begin{cases} 1 & \text{ถ้า } n = 1 \\ 2 & \text{ถ้า } n = 2 \\ 3/4 & \text{ถ้า } n = 3 \\ \dfrac{s_{n-1} + 2\sqrt{s_{n-2}}}{4s_{n-3}} & \text{ถ้า } n \geq 4 \end{cases}$$

วิธีทำ

5.2 ฟังก์ชันทำซ้ำ

บทนิยาม 5.2.1

ฟังก์ชันทำซ้ำ (iterative function) คือ ฟังก์ชันซึ่งอาศัยการทำงานแบบวนซ้ำ โดยไม่มีการเรียกใช้ตัวเองเช่นกรณีของฟังก์ชันเวียนเกิด (นั่นคือ ใช้เพียงคำสั่ง `for` หรือ `while` ในการทำซ้ำ)

โดยทั่วไปแล้ว ฟังก์ชันใดที่ออกแบบให้อยู่ในรูปของฟังก์ชันเวียนเกิด มักจะสามารถออกแบบให้อยู่ในรูปของฟังก์ชันทำซ้ำได้เช่นกัน

ตัวอย่าง 5.2.2 เราสามารถปรับฟังก์ชัน `recursive_f` ในตัวอย่าง 5.1.2 ซึ่งเป็นฟังก์ชันเวียนเกิด ให้อยู่ในรูปของฟังก์ชันทำซ้ำได้ ดังนี้

```
function [y] = iterative_f(x)
    y = 1 // ค่าของ output สำหรับกรณีฐาน
    while x > 0
        y = y + 2   // ปรับปรุงค่าของตัวแปร y ซึ่งเป็น output
        x = x - 2   // ปรับปรุงค่าของตัวแปร x ซึ่งเป็น input
    end
endfunction
```

จงหาค่าของ output ที่ได้จากการเรียกใช้ `iterative_f(7)`

วิธีทำ

ในการทำงานบางอย่าง ฟังก์ชันทำซ้ำจะออกแบบได้ยากกว่า แต่สามารถทำงานได้เร็วกว่าฟังก์ชันเวียนเกิด

ตัวอย่าง 5.2.3 จงเปลี่ยนฟังก์ชันเวียนเกิด `MySequence` ในตัวอย่าง 5.1.3 ให้อยู่ในรูปของฟังก์ชันทำซ้ำ
วิธีทำ

ตัวอย่าง 5.2.4 จงออกแบบฟังก์ชัน `mysum_recursive` และ `mysum_iterative` เพื่อคำนวณหาค่าของ $1+2+3+\cdots+n$ เมื่อ n เป็นจำนวนเต็มบวกที่กำหนดให้ โดยใช้ฟังก์ชันเวียนเกิดและฟังก์ชันทำซ้ำ ตามลำดับ
วิธีทำ