เรียนรู้ภาษาโลโก้

เทพพิทักษ์ การุญบุญญานันท์

พฤศจิกายน 2547

1 ความเป็นมา

โลโก้ (Logo) เป็นภาษาที่ใช้ฝึกเขียนโปรแกรมสำหรับเด็ก พัฒนาขึ้นโดย W. Feurzeig, D. Bobrow และ S. Papert แห่งบริษัท Bolt Beranek and Newman ที่เคมบริดจ์ รัฐแมสซาชูเซตต์ สหรัฐอเมริกา

ลักษณะของโปรแกรมเป็นการสั่งให้ "เต่า" วาดรูป ในยุคแรก ๆ นั้น จะใช้หุ่นยนต์รูปเต่าถือปากกาเดินไปบนกระดาษตามคำสั่ง และขีดเขียนลายเส้นบนกระดาษตามเส้นทางที่เต่าเดินไป แต่ในปัจจุบัน "เต่า" กลายเป็นรูปสามเหลี่ยมเหมือนหัวลูกศรเดินไปมาบนจอคอมพิวเตอร์

2 โปรแกรมที่ใช้

บทเรียนนี้ จะอ้างอิง Berkeley Logo ซึ่งมีมากับระบบปฏิบัติการ Debian GNU/Linux ผู้เรียนสามารถติดตั้งเองได้โดยสั่ง

```
apt-get install ucblogo
```

จากนั้น สามารถเรียกโปรแกรม `logo` ได้ที่หน้าต่างเทอร์มินัล จะมีเครื่องหมาย `?` รอรับคำสั่ง

```
$ logo
Welcome to Berkeley Logo version 5.3
?
```

โปรแกรมจะรับคำสั่งภาษาโลโก้ทีละบรรทัด และทำงานตามคำสั่งทันที หากมีคำสั่งวาดภาพ ก็จะเปิดอีกหน้าต่างหนึ่งสำหรับวาดภาพโดยเฉพาะต่างหาก เช่น เมื่อสั่งคำสั่ง `home` โปรแกรมจะเปิดหน้าต่างว่าง ๆ ที่มีแต่เต่าตัวหนึ่ง (เป็นรูปสามเหลี่ยม) ที่กลางจอ

ในบางครั้ง ถ้าต้องการหยุดโปรแกรมกลางคัน สามารถหยุดได้โดยกด `Ctrl-c`

3 ภาษาโลโก้

3.1 การอ้างตำแหน่ง

ในหน้าต่างวาดภาพของโลโก้ จะอ้างตำแหน่งของจุดต่าง ๆ ด้วยพิกัด (x, y) โดยจุดกึ่งกลางจอภาพคือจุด $(0, 0)$ ตำแหน่งในแนวนอนคือแกน X มีค่าเป็นบวกไปทางขวา และมีค่าเป็นลบไปทางซ้ายของจุด $(0, 0)$ ตำแหน่งในแนวตั้งคือแกร Y มีค่าเป็นบวกไปด้านบน และมีค่าเป็นลบไปทางด้านล่างของจุด $(0, 0)$

มุมเลี้ยวของเต่า มีหน่วยเป็นองศา ขณะที่โปรแกรมเริ่มทำงานนั้น เต่าจะอยู่ที่ *ตำแหน่งเหย้า (home)* คือตำแหน่ง $(0, 0)$ และหันไปทางด้านบน (คือแกน Y ด้านบวก) เราสามารถสั่งให้เต่ากลับมาที่ตำแหน่งเหย้านี้เมื่อไรก็ได้ที่ต้องการ โดยใช้คำสั่ง `home`

3.2 คำสั่งวาดรูป

รูปแบบคำสั่งของโลโก้ จะมีทั้งแบบย่อและแบบเต็ม จะใช้แบบไหนก็ได้ เริ่มจากคำสั่งล้างจอภาพก่อน ซึ่งจะล้างจอและย้ายเต่ามาที่ตำแหน่งเหย้า:

```
cs  clearscreen
```

คำสั่งวาดรูปที่ง่ายที่สุด ได้แก่คำสั่งให้เต่าเดินไปข้างหน้าหรือถอยหลัง และเลี้ยวซ้าย-เลี้ยวขวา ได้แก่คำสั่งต่อไปนี้

```
fd  forward
bk  backward
rt  right
lt  left
```

คำสั่งเหล่านี้ ต้องตามด้วยตัวเลข สำหรับคำสั่ง `fd` และ `bk` นั้น ต้องระบุระยะทางที่ให้เดินไป ส่วนคำสั่ง `rt` และ `lt` นั้น ต้องระบุมุมที่จะให้เลี้ยว (หน่วยเป็นองศา)

เรามาลองคำสั่งง่าย ๆ การสั่งคำสั่งนั้น จะสั่งทีละคำสั่งและจบด้วย Enter หรือสั่งหลายคำสั่งในบรรทัดเดียวก็ได้ แต่ต้องระบุค่าที่แต่ละคำสั่งต้องการให้ครบ มิฉะนั้นโลโก้จะบ่นและไม่ทำงานให้

คำสั่งต่อไปนี้:

```
fd 60 rt 120 fd 60 rt 120 fd 60 rt 120
```

จะสั่งให้เต่าวาดรูปสามเหลี่ยมด้านเท่า แต่จะสังเกตว่า คำสั่ง `fd 60 rt 120` นั้นถูกสั่งซ้ำกันสามครั้ง เราสามารถสั่งคำสั่งซ้ำแบบนี้ได้ด้วยคำสั่ง `repeat` ดังนี้

```
repeat 3 [fd 60 rt 120]
```

ซึ่งคำสั่งในวงเล็บเหลี่ยมจะถูกกระทำซ้ำสามรอบ

3.3 คำสั่งอื่น ๆ

โลโก้ยังมีคำสั่งอื่น ๆ สำหรับวาดรูปอีก เช่น

`pu`	`penup`
`pd`	`pendown`
`ht`	`hideturtle`
`st`	`showturtle`
`home`	
`label`	
`setxy`	

คำสั่ง `penup` และ `pendown` ใช้ยกปากกาและจดปากกา ตามลำดับ ถ้าปากกาถูกยก จะไม่เกิดรอยขีดเขียนตามที่เต่าเดินไป คำสั่ง `hideturtle` และ `showturtle` ก็ใช้ซ่อนและแสดงเต่า ซึ่งไม่ได้มีผลต่อการวาดรูปแต่อย่างใด ส่วนคำสั่ง `home` ใช้ย้ายเต่ากลับมาที่ตำแหน่งเหย้า (ตำแหน่ง $(0, 0)$ หรือกลางจอ หันไปทางด้านบน) ดังได้กล่าวมาแล้ว

คำสั่งทั้งห้าข้างต้น ไม่ต้องส่งค่าใด ๆ ตามไปอีก ส่วนคำสั่งที่เหลือ คือ `label` ใช้สั่งแสดงข้อความ ณ ตำแหน่งที่เต่าอยู่ จะรับค่าหนึ่งค่าเป็นข้อความ ซึ่งอาจใช้รูปแบบกำกับด้วยเครื่องหมายคำพูดเปิด (เช่น `"string`) หรือใช้วงเล็บเหลี่ยมครอบข้อความ (เช่น `[a string of letters]`) ก็ได้ และคำสั่ง `setxy` ใช้สั่งเต่าเดินไปที่ตำแหน่งพิกัดที่กำหนด จะรับค่าสองค่า คือค่าพิกัด X และ Y ตามลำดับ (อาจจะลากเส้นไปด้วยถ้าจดปากกาอยู่)

ทราบไหมว่า คำสั่งต่อไปนี้วาดรูปอะไร?

```
cs pu setxy -60 60 pd home rt 45 fd 85 lt 135 fd 120
```

3.4 ตัวแปร

ตัวแปรคือที่เก็บค่า โดยมีชื่อกำกับสำหรับอ้างในโปรแกรม ชื่อตัวแปรประกอบด้วยตัวอักษรหรือตัวเลข โดยขึ้นต้นด้วยตัวอักษรเท่านั้น เช่น `size` และเมื่อเราจะอ้างถึงค่าในตัวแปร จะใช้เครื่องหมายทวิภาค (`:`) กำกับ เช่น `:size`

การกำหนดค่าให้กับตัวแปร จะใช้คำสั่ง `make` ตามด้วยข้อความระบุชื่อตัวแปรและค่าที่จะกำหนด ตัวอย่างเช่น

```
make "size 60
```

เป็นการกำหนดค่า 60 ให้กับตัวแปร `size` สังเกตว่าเราใช้รูปแบบ `"size` ในการอ้างชื่อตัวแปร แต่ใช้ `:size` ในการอ่านค่าที่เก็บในตัวแปร

ถ้าตัวแปรที่กำหนดในคำสั่ง `make` ยังไม่ถูกอ้างมาก่อน คำสั่ง `make` ก็จะสร้างตัวแปรให้ แต่เราจะอ่านค่าตัวแปรที่ยังไม่เคยกำหนดมาก่อนไม่ได้

คำสั่งที่ใช้พิมพ์ค่าออกทางหน้าต่างคำสั่ง ได้แก่คำสั่ง `print`

```
? print 50
50
? print "hello
hello
? make "size 60
? make "name "Thongmee
? print :size
60
? print :name
Thongmee
```

3.5 เครื่องหมายเลขคณิต

โลโก้มีเครื่องหมายเลขคณิต บวก ลบ คูณ หาร โดยใช้เครื่องหมาย `+ - * /` ตามลำดับ เราจึงสามารถสั่งอะไรทำนองนี้ได้:

```
make "size 49/7
print 2*3
print :size - 5
```

คำสั่งอื่นที่มีประโยชน์ ได้แก่

- `sqrt` a หาค่า $\sqrt{a}$
- `power` a b หาค่า a^b
- `exp` a หาค่า e^a
- `ln` a หาค่า $\ln a$
- `log10` a หาค่า $\log a$

ทดลองโปรแกรมต่อไปนี้:

```
make "angle 0
repeat 1000 [fd 6 rt :angle make "angle :angle + 7]
```

ค่าของตัวแปร `angle` เปลี่ยนแปลงอย่างไรบ้าง? มีปมกี่ปมเกิดขึ้นในรูป? และถ้าตัวเลข 7 ถูกแทนที่ด้วย 11 หรือ 13 จะเกิดปมกี่ปม?

3.6 การสุ่มตัวเลข

บางครั้ง การสุ่มตัวเลขก็ทำให้โปรแกรมให้ผลลัพธ์ที่หลากหลาย โลโก้มีคำสั่ง `random` สำหรับสุ่มตัวเลขในช่วงที่กำหนด เช่น

```
print random 360
```

เป็นการสุ่มเลขขึ้นมาค่าหนึ่งจากช่วง 1 ถึง 360 แล้วพิมพ์ออกทางหน้าต่างคำสั่ง (ลองสั่งหลาย ๆ ครั้ง แล้วเทียบดูว่าผลลัพธ์ได้เท่าเดิมหรือไม่)

ลองเล่นอะไรกับเลขสุ่มดูบ้าง คำสั่งสองคำสั่งนี้ ให้ผลลัพธ์ต่างกันอย่างไร?

```
repeat 100 [fd random 80 rt 90]
repeat 1000 [fd 4 rt random 360]
```

3.7 โปรแกรมย่อย (Procedure)

เราสามารถเขียนชุดคำสั่งที่ถูกใช้บ่อย ๆ ไว้เป็นโปรแกรมย่อยสำหรับเรียกใช้ภายหลังได้ ซึ่งการเรียกใช้ ก็จะเหมือนกับการเรียกคำสั่งภายในของโลโก้ทุกประการ

การสร้างโปรแกรมย่อย จะใช้คำสั่ง `to` ตามด้วยชื่อโปรแกรมย่อย เนื้อหาของโปรแกรมย่อย และจบด้วยคำสั่ง `end` ตัวอย่างเช่น เราอาจจะสร้างโปรแกรมย่อย `random_walk` ที่เดินสุ่มเลี้ยวเป็นมุมฉาก

```
to random_walk
  repeat 100 [fd random 80 rt 90]
end
```

จากนั้นก็สามารถสั่ง `random_walk` ได้เหมือนเป็นคำสั่งโลโก้ปกติ

โปรแกรมย่อยสามารถเรียกโปรแกรมย่อยอื่นได้อีกด้วย ตัวอย่างเช่น สมมุติว่าเราต้องการจะวาดต้นไม้ โดยแบ่งส่วนวาดทีละส่วน เริ่มจากลำต้น ตามด้วยใบ และในส่วนลำต้นนั้น ก็แยกวาดส่วนซ้าย ส่วนยอด ส่วนขวา และส่วนล่างอีกที สุดท้ายแล้ว ก็อาจจะได้โปรแกรมในลักษณะนี้:

```
to left_side
  rt 20 fd 20 lt 20 fd 60
end

to top_side
  rt 90 fd 25 rt 90
end

to right_side
  fd 60 lt 20 fd 20 rt 20
end

to return_start
  rt 90 fd 40 rt 90
end

to trunk
  left_side
```

```
  top_side
  right_side
  return_start
end

to center_top
  pu fd 80 rt 90 fd 20 lt 90 pd
end

to circle
  repeat 360 [fd 1 rt 1]
end

to tree
  pu bk 100 pd
  trunk
  center_top
  lt 90
  circle
end
```

ลองสั่ง tree ก็จะได้ต้นไม้ที่อ ๆ มาหนึ่งอัน ถ้าไม่ชอบอาจจะตกแต่งโปรแกรมใหม่เองได้ตามใจชอบ

3.8 การสร้างโครงงาน

นอกจากการสั่งคำสั่งทีละขั้นแล้วทำงานสด ๆ เราสามารถเขียนโปรแกรมเตรียมไว้แล้วเรียก logo ให้อ่านโปรแกรมขึ้นมาทำงาน ซึ่งวิธีนี้จะสะดวกต่อการทำโครงงานเพราะสามารถเรียกโปรแกรมที่เขียนไว้ขึ้นมาแก้ไขได้ตามต้องการ

วิธีเรียกใช้โปรแกรมที่เตรียมไว้ ทำได้สองแบบ สมมุติว่าโปรแกรมวาดต้นไม้ข้างต้นถูกเก็บไว้ในไฟล์ชื่อ tree.logo วิธีแรกคือ หลังจากเรียกโปรแกรม logo ขึ้นมาแล้ว ก็ป้อนคำสั่งในบรรทัดคำสั่งของโลโก้:

```
? load "tree.logo
```

อีกวิธีหนึ่งคือ ระบุชื่อโปรแกรมตั้งแต่ขั้นตอนการเรียกโปรแกรม `logo` เลย โดยระบุชื่อต่อท้ายคำสั่ง

```
$ logo tree.logo
```

อนึ่ง การเขียนโปรแกรมยาว ๆ นั้น คำอธิบายประกอบโปรแกรมจะช่วยเตือนความจำได้ดี เมื่อจำเป็นต้องกลับมาอ่านโปรแกรมภายหลัง โลโก้จะไม่พยายามตีความเนื้อหาโปรแกรมที่อยู่หลังอัฒภาค (`;`) ไปจนจบบรรทัด เราจึงนำมาใช้เขียนคำอธิบายให้มนุษย์อ่านได้

3.9 การควบคุมเงื่อนไขการทำงาน

คำสั่ง `repeat` ช่วยให้กำหนดการทำงานหนึ่ง ๆ ซ้ำกันหลายรอบได้ โดยกำหนดจำนวนรอบ ตามด้วยชุดคำสั่งที่จะทำซ้ำภายใต้วงเล็บเหลี่ยม []

คำสั่ง `if` ใช้ตรวจสอบเงื่อนไขการทำงาน โดยจะทำคำสั่งที่กำหนดต่อเมื่อเงื่อนไขบางอย่างเป็นจริงเท่านั้น คำสั่ง `if` จะตามด้วย "เงื่อนไข" ที่จะตรวจสอบ และชุดคำสั่งที่จะทำงานภายใต้วงเล็บเหลี่ยม [] ตัวอย่างเช่น

```
if :size < 3 [stop]
```

จะหยุดและออกจากโปรแกรมย่อยถ้าค่าตัวแปร `size` น้อยกว่า 3

3.10 โปรแกรมย่อยแบบมีตัวแปร

โปรแกรมย่อยอาจมีประโยชน์มากขึ้นถ้าสามารถกำหนดค่าตัวแปรขณะทำงานได้ด้วย เช่น โปรแกรมย่อยต่อไปนี้สามารถวาดรูปสี่เหลี่ยมจัตุรัสขนาดใด ๆ ก็ได้ตามที่กำหนด:

```
to square :size
  repeat 4 [fd :size rt 90]
end
```

การเรียกใช้ ก็ต้องส่งค่าขนาดที่ต้องการมาให้ด้วย เช่น `square 60` หรือ `square :side` เมื่อ `side` คือค่าตัวแปรจำนวนเต็ม

โปรแกรมย่อย `house` ต่อไปนี้ ใช้วาดบ้านแบบหยาบ ๆ หลังหนึ่ง:

```
to square :size
  repeat 4 [fd :size rt 90] ; where is the turtle when this step completes?
end

to floor :size
  repeat 2 [fd :size rt 90 fd :size * 2 rt 90]
end

to house
  floor 60 fd 60 floor 60 ; where is the turtle at this point?

  pu fd 20 rt 90 fd 20 lt 90 pd
  square 20

  pu rt 90 fd 60 lt 90 pd
  square 20
end
```

3.11 โปรแกรมย่อยเรียกตัวเอง

นอกจากจะเรียกโปรแกรมย่อยอื่นแล้ว โปรแกรมย่อยสามารถเรียกตัวเองได้ด้วย โดยหลักการก็ไม่ได้ต่างอะไรกับการเรียกโปรแกรมย่อยอื่นตามปกติ เพียงแต่ว่าจะต้องมีเงื่อนไขการหยุดเรียกตัวเอง เพื่อป้องกันการวนซ้ำไม่รู้จบ

มีรูปร่างทางคณิตศาสตร์ประเภทหนึ่ง ที่มีคุณสมบัติที่น่าสนใจคือ ตัวมันเองบรรจุรูปแบบของตัวมันเองซ้อน ๆ กันหลายชั้นไม่รู้จบ ทำให้เมื่อขยายภาพตรงรายละเอียดเล็ก ๆ ก็จะยังเห็นรูปแบบของภาพใหญ่อยู่ในนั้นไม่รู้จบ รูปร่างพวกนี้เรียกว่า *แฟรกทัล (fractal)*

โปรแกรมต่อไปนี้แสดงตัวอย่างของแฟรกทัลที่มีชื่อเสียงอันหนึ่ง เป็นรูปสามเหลี่ยมที่บรรจุสามเหลี่ยมย่อย ๆ ซ้อนกันในตัวเอง:

```
to triangle :size
  repeat 3 [fd :size rt 120]
end
```

```
to triangle_fractal :size
  if :size < 6 [triangle :size stop]
  triangle_fractal :size/2
  pu fd :size/2 pd
  triangle_fractal :size/2
  pu rt 60 fd :size/2 rt 60 pd
  triangle_fractal :size/2
  pu rt 60 fd :size/2 rt 60 fd :size/2 rt 120 pd
end
```