一個可愛的語言

許介彥
大葉大學電機工程學系
chsu@mail.dyu.edu.tw

寫程式難不難？

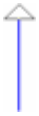
這個問題並不容易回答，因爲要視情況而定；什麼情況呢？大致上說來，有三個方面需要考慮，其一是「什麼人？」——程式是由誰來寫？每個人在程式設計方面的天份及努力的程度各有不同，因此難以一概而論。其二是「什麼事？」——想用程式完成什麼工作？可用電腦完成的工作種類繁多，有些簡單的工作連初學者都能完成，而較難的工作可能連專家都感到棘手。其三是「什麼語言？」——透過哪一種程式語言來跟電腦溝通？不同的語言在學習上本身就有難易之分，這當然直接影響到程式的撰寫。

本文的目地是向讀者介紹一個相當容易學習的語言——Logo；雖然它的用途廣泛，不過我們將只針對它在繪圖方面的功能來作介紹（繪圖是 Logo 的「強項」）。Logo 語言簡單到連小學生都能用它寫出漂亮的程式（以及產生漂亮的執行結果），世界各地有許多中小學即是以 Logo 做爲學生所接觸的第一個程式語言。

就像其他程式語言一樣，Logo 也存在著一些大同小異的版本，本文將以 MSW Logo 爲例來作說明；此軟體是免費的，檔案大小不到 2MB，讀者可透過上面的超連結來下載及安裝。同一個網頁裡還包含了許多不錯的入門教材及相關資料。

執行 MSW Logo 後在螢幕上首先出現一個空白的視窗，這就是我們的「畫紙」，其正中央有一個小小的尖端向上的三角形，這標示著「畫筆」的起始位置及方向，接下來我們即可透過在命令視窗鍵入指令來指揮畫筆移動，例如當我們鍵入 **`forward 100`** 並按下 Enter 鍵後將使得畫筆往前走 100 步（畫紙的預設大小是長寬各 1000 步），因此在畫紙上畫出了一條線段：

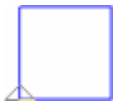
`forward 100`：



與改變畫筆的位置及方向有關的指令主要有前進、後退、左轉、右轉等四個，它們的用法如下：

例 子	可簡寫爲	畫筆的動作
`forward 100`	**`fd 100`**	前進 100 步
`backward 200`	**`bk 200`**	後退 200 步
`right 25`	**`rt 25`**	向右轉 25°
`left 319`	**`lt 319`**	向左轉 319°

利用這些指令已經不難畫出許多我們熟悉的幾何圖形，例如：向前走 100 步，向右轉 90°，並重複上述動作四次即可畫出一個正方形：



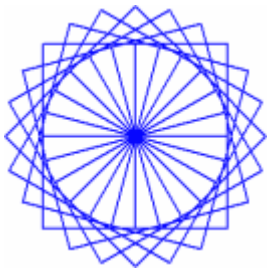
```
fd 100 rt 90 fd 100 rt 90 fd 100 rt 90 fd 100 rt 90
```

這也可透過迴圈來達成：

```
repeat 4 [fd 100 rt 90]
```

畫筆在畫完上面的正方形後正好回到了它出發前的位置及方向。

如果我們讓畫筆在畫完正方形回到原點後向右轉 15°，然後重新出發畫出第二個正方形，回到原點後再向右轉 15°，再出發畫出第三個正方形……，直到在原點繞完一整圈為止，那麼總共須畫出 24 個正方形（因為 24×15° = 360°），所得的圖形如下，而右方為其所需指令：



```
repeat 24 [repeat 4 [fd 100 rt 90] rt 15]
```

看似複雜的圖形竟只需這麼簡單的一行指令就能畫出來，是不是有點意思呢？

請留意上圖的繪製並不需要利用程式去計算各個正方形的頂點的「坐標」，如果需要的話麻煩就大了，程式將不可能寫得如此簡單（因為涉及坐標軸的旋轉）；這正是 Logo 繪圖能力強大的主要原因，也就是它的畫筆的位置及方向的改變可以是「相對於」目前的位置及方向，程式並不需要如解析幾何般去計算各個點在平面上的坐標；如果您曾經用 Turbo C、QuickBASIC 或其他可畫圖的語言畫過圖的話當更能體會 Logo 這個作法的好處。

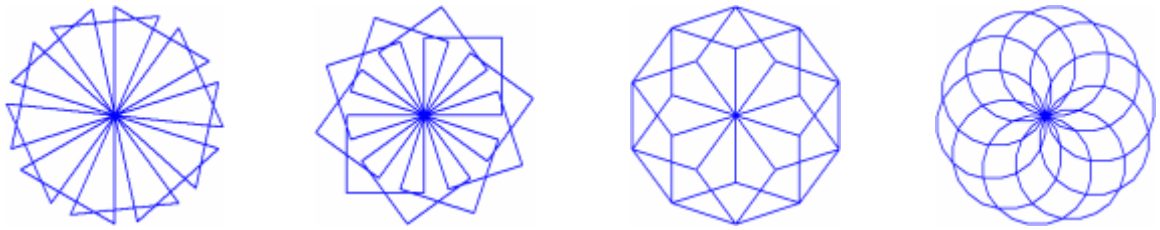
除了正方形外，讓我們也試試其他正多邊形的繪製。可以想見，畫其他正多邊形跟畫正方形基本上並沒有什麼不同，差別只是在當畫筆走到正方形的每個頂點時是轉了 360°/4 = 90°，而對任意正 *n* 邊形（ $n \geq 3$ ）而言，畫筆走到每個頂點時須轉的角度為 360°/*n*。以下的 **poly** 程序可用來畫出一個正 *n* 邊形，其中的 **n** 為程序的參數（Logo 的變數在使用時變數名稱前面須加冒號），而接下去的程序 **rotpoly** 則是透過持續呼叫 **poly** 程序 10 次來將 10 個正多邊形排成一圈（每次回到原點後向右轉 36°）：

```
to poly :n                          ；程序頭尾的 to 與 end 是 Logo 的保留字
repeat :n [fd 100 rt 360/:n]        ；畫出一個正 n 邊形
end

to rotpoly :n
repeat 10 [poly :n rt 36]           ；將 10 個正 n 邊形排成一圈
end
```

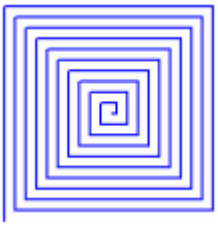
下面是呼叫 **rotpoly** 的幾個例子：

retpoly 3 **retpoly 4** **retpoly 5** **retpoly 20**



由上圖可看到當一個正多邊形的邊數越大，看起來就越接近圓。

如果我們讓畫筆如同畫正方形般每走到一個頂點就往右轉 90°，不過是讓每一「邊」所走的步數越來越小，例如第一次走 200 步，接下來走 195 步，接下來走 190 步等，每次減少 5 步，直到走到步數爲 0 爲止，所得的圖形將如下：



上圖要如何用程式畫出來呢？一開始當然是需要讓畫筆向前走 200 步，然後往右轉。接下來呢？接下來的工作其實和一開始的工作很像，差別只在不是從走 200 步開始而是從走 195 步開始，它們都是從走若干步開始直到走到步數爲 0 爲止；如果我們有一個程序（假設取名爲 **square**）可以接受一開始所需走的步數做爲參數，並且會持續走到步數爲 0 爲止，那麼我們所要完成的工作就是 **square 200**，而此工作可以在走了 200 步及向右轉後將剩下的工作交由 **square 195** 繼續完成（這用上了「遞迴」的觀念）；同樣地，**square 195** 在走了 195 步及向右轉後也可以將剩下的工作交由 **square 190** 繼續完成；隨著工作的持續交棒，我們的圖就被一段一段畫出來了。根據上面的想法我們寫出了以下的程序 **square**，而呼叫 **square 200** 即可畫出上圖。

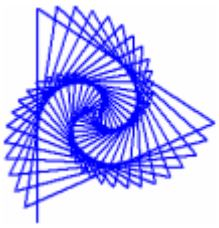
```
to square :n
if :n = 0 [stop]      ；如果步數已是 0 就不要再走了
fd :n rt 90           ；否則的話向前走 n 步然後向右轉 90°
square :n - 5         ；將剩下的工作交由 square :n-5 繼續完成
end
```

如果畫筆走到每個頂點時不是轉 90°而是轉其他角度，所得又是如何呢？以下我們將上面的 **square** 程序稍微改寫一下，讓它更有彈性一點，可接受三個參數：一開始走幾步、每個頂點轉幾度以及步數每次減少幾步；新的程序（取名 **spir**）如下：

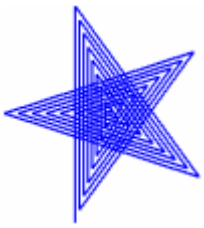
```
to spir :side :angle :dec
if :side < 0 [stop]
fd :side rt :angle
spir :side - :dec :angle :dec
end
```

下面是呼叫 **spir** 的幾個例子，圖形隨著輸入參數的不同可以有很大的變化：

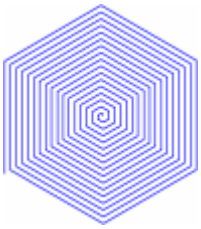
spir 200 123 3



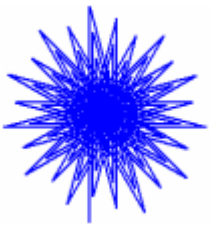
spir 200 144 5



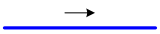
spir 200 60 2



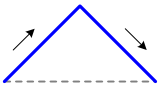
spir 200 165 2



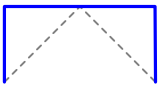
接下來我們將用 Logo 程式畫出一條數學上有名的曲線，此曲線可經由重複一個簡單的動作一步步建構出來。我們從畫出一條由左而右的水平線段開始：



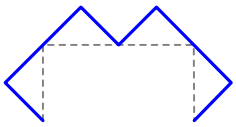
對這條線段我們做以下動作：將它擦掉，改以另外兩條線段取代（因此我們剛才其實根本不用將水平線段真的畫出來！），這兩條新線段位於原來的線段的上方，而且是以原來的線段做爲一個 45°-45°-90°三角形的斜邊。所得的新圖形如下，其中的虛線是被拿掉的舊線段：

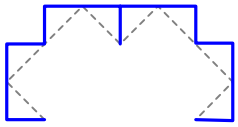


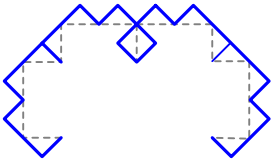
如果我們對上圖的兩條線段分別做跟剛才一樣的動作，所得的圖形將含有四條線段：



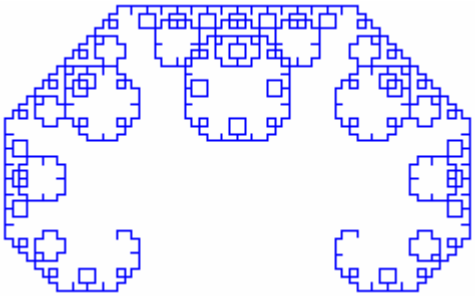
類似的步驟可以無窮盡地進行下去，圖形所含的線段數因此將由 1 而 2、由 2 而 4、由 4 而 8 等，所得的曲線稱爲 Lévy curve，是碎形的一種。以下是接下去的三個步驟所得的圖形：







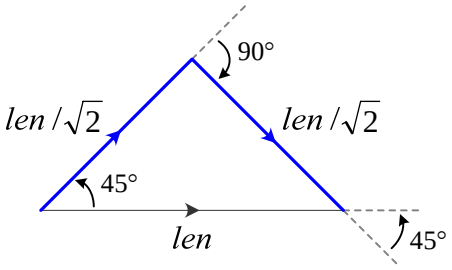
隨著步驟的持續進行，圖中每條線段的長度將越來越短（每條線段的長度都是上個階段的 $1/\sqrt{2}$ 倍）；下面是進行完 10 個步驟後所得的圖形以及可畫出此圖的程序：



```
to c :len :level
if :level = 0 [fd :len stop]
lt 45 c :len / sqrt 2 :level-1
rt 90 c :len / sqrt 2 :level-1
lt 45
end
```

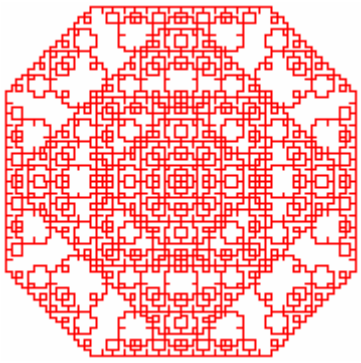
上圖是由呼叫 **`c 120 10`** 而得，而且我們假設在呼叫之前已先執行一個 **`rt 90`** 指令來將畫筆轉成水平向右的方向（純粹只是爲了美觀，否則圖形將是直立的）。

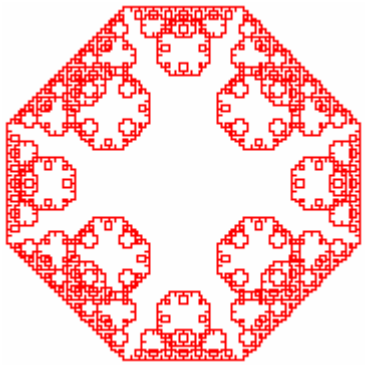
程序 **`c`** 並不難解讀，傳入的參數 **`level`** 是所要進行的步驟數，而參數 **`len`** 則是一開始的線段的長度。當我們要用遞迴的方式設計程式時，所需考慮的只是連續兩個階段之間的關係以及程序持續呼叫本身的動作該在什麼條件成立時結束；對我們所要畫的 Lévy curve 而言，連續兩個階段之間的關係就建立在將一條線段換成兩條線段這個基本的動作：



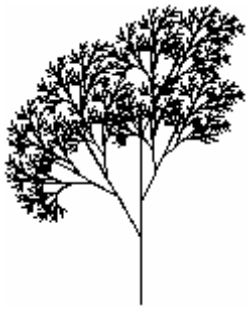
將上圖對照程序 **`c`** 就不難看出該程序是如何寫成的了（程序中的 **`sqrt 2`** 即 $\sqrt{2}$ ）。

畫 Lévy curve 時，如果我們不是從一條水平線段開始，而是從一個正方形的四條邊開始，所得的圖形更是美妙。下面的兩個圖分別是四條 Lévy curve 齊向正方形的內部及外部演化 10 個步驟所得的結果；我特別選用紅筆來畫，看起來很「吉祥」吧？





下面這棵「樹」也是用 Logo 畫出來的，是不是栩栩如生呢？程式同樣是用遞迴的手法寫成，請您自己參詳了。



```
to tree :len
if :len < 4 [fd :len bk :len stop]
fd :len/3
lt 30 tree :len*2/3 rt 30 fd :len/6
rt 25 tree :len/2 lt 25 fd :len/3
rt 25 tree :len/2 lt 25 fd :len/6
bk :len
end
```

寫程式難不難？我不知道；不過我相信對大部分人而言，只要透過適當的學習，寫程式其實可以很好玩。