Artspeak Reference Guide

By

Ron Schnell

Introduction

Artspeak is a computer language written in 1970 by a brilliant math professor at NYU named Jacob Schwartz. It was taught to students at the Courant Institute of Mathematical Sciences at NYU for not too many years after that, including by Henry Mullish, who wrote a programmer's manual for it.

Artspeak only ran on the CDC 6600; one of the early supercomputers created by Seymour Cray. It took input only via punch cards, and output via a plotter with black ink. I was able to program in Artspeak in 1981 at NYU, and as a teenage computer and math enthusiast, thoroughly enjoyed it.

Artspeak has since disappeared off the face of the planet. Some might think that its time has passed, but I think it is still a great language for computer and math enthusiasts.

I recently unearthed an early draft of Henry Mullish's programmer's guide, and have written this interpreter, mostly based upon its content.

For the handful of people who might remember how Artspeak worked, I tried to make this implementation as true to the original as possible. There are some differences, some of which are due to programming convenience, and others are due to the fact that punch cards are not being used. For example:

- Although line labels and comment characters must be in column 1 like in the original, there are no restrictions on the starting column of statements
- The original Artspeak was always in upper case, because of the restrictions of the keypunch. This version is case insensitive.
- The original Artspeak plotted on a 10" x 10" sheet of paper, and all of the coordinates and distances were interpreted as inches. For this interpreter, the scale of coordinates and distances is the same as the original, but the values are essentially multiplied by 100, and drawn onto a 1000 x 1000 image outputted onto a 1001 x 1001 .PNG file.
- At this time, multi-segment non-linear curves are not supported, and thus the expression of these curves in the LET statement is slightly different.
- Unlike the original, spaces or commas are required between statements and operands.

I have dedicated this interpreter to the memory of Jack Schwartz and to Henry Mullish, who were a big part of my young computer experience, and played a pivotal role in my enthusiasm that turned into a successful career. Many of the example programs were shown to me by Professor Mullish, and some were in his draft book that I recently unearthed.

A Quick Example

```
LET X1 BE VALUE 3
LET X2 BE VALUE 4
LET P1 BE POINT (X1, X2)
LET C1 BE CIRCLE, CENTER P1, RADIUS 1
DRAW C1
```

This simple program draws a circle centered at coordinate (3,4), with a radius of 2. Note that in the lines that define the point and circle, the coordinates and values can be indicated either by variables or constants themselves. Variables are used in this example, but are completely unnecessary. Programs that make use of mathematical functions or control flow statements can benefit from the use of these variables.

Also interesting to note is the DRAW command. Only two commands in ARTSPEAK cause something to be output: DRAW and CAPTION. In the example above, the DRAW statement actually draws the circle, which is output to .PNG.

See Figure 1 below to see the output of this program (with the optional grid on).

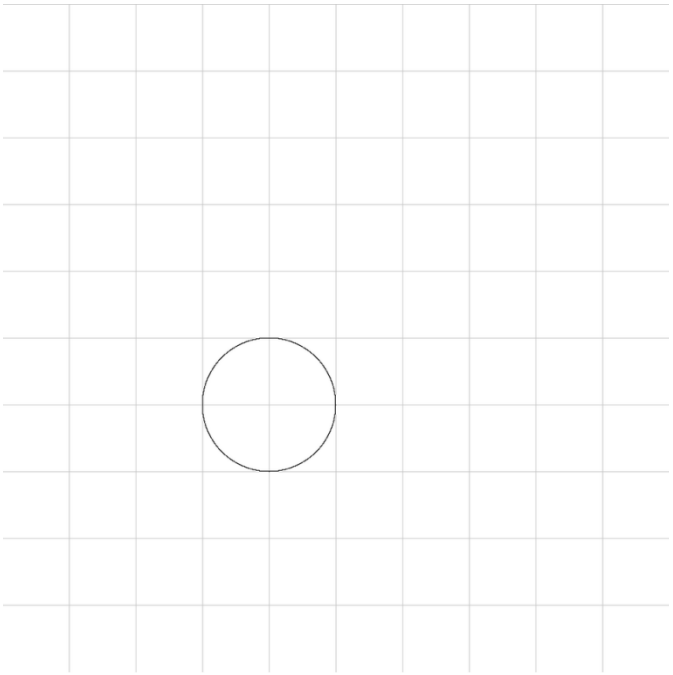

Figure 1

The LET Statement

The LET statement is the only way to define the geometric objects that are drawn in Artspeak. There are three types of variables that can be assigned using the LET statement: Values, Points, and Curves. Values and Points are for defining constants and coordinates respectively. Curve variables are used to define lines, circles and curves.

In Artspeak, variable names start with one of 'X', 'P', or 'C' (for Value, Point, or Curve), followed by an integer between 1 and 100, inclusive. Example valid variables are X5, P100, and C24.

The format for the LET statement for assigning a ***value*** to a variable is:

```
LET Xn BE VALUE n[.nnnn]
```

Examples:

```
LET X2 BE VALUE 3

LET X79 be VALUE 2.71818
```

The format for the LET statement for assigning a ***point*** to a variable is:

```
LET Pn BE POINT (x,y)
```

Note that in addition to the coordinates being able to be represented by a parenthetical set of axis values as above, the coordinate numbers can also be substituted by 'X' (value) variables, as in (X1, X2). For all of the examples below, coordinates can be expressed the same way, in addition to via 'P' variables.

The format for the LET statement for assigning a ***circle*** to a variable is:

```
LET Cn BE CIRCLE CENTER (x,y)|Pn RADIUS n[.nnnn]|Xn
```

Note that the RADIUS and CENTER operands can be in the reverse order.

The format for the LET statement for assigning a ***line*** to a variable is:

```
LET Cn BE LINE (x,y)|Pn (x2,y2)|Pn … (xn,yn)|Pn
```

A line declaration must contain two or more points, and can change direction in order to form a polygon. For example, the following will define a square with sides of length 2:

```
LET C1 BE LINE (0,0),(0,2),(2,2),(2,0),(0,0)
```

The format for the LET statement for assigning a ***curve*** to a variable is:

```
LET C1 be CURVE (Xe1, Ye1), (Xi, Yi), (Xe2, Ye2), CF
```

Artspeak has a unique way of defining curves. (Xe1,Ye1) and (Xe2,Ye2) are the two endpoints of the curve. (Xi,Yi) define an ***imaginary*** point to where what would otherwise be a straight line will now ***stretch***. The curve will be stretched towards the imaginary point with the ***curve factor*** described by CF,

which is a number between 0 and 1, inclusive. If the number is 0, it is a straight line. If the number is 1, the curve will appear as two straight line segments going from endpoint1 to the imaginary point (which is not so imaginary anymore), and then back to endpoint2. A value of .5 would have the curve's "apex" be halfway between what would be a straight line between the two endpoint, and the imaginary point.

To see what this means, the example program below:

```
LET C1 BE CURVE (2,2),(4,6),(6,2),0
DRAW C1
LET C1 BE CURVE (2,2),(4,6),(6,2),.1
DRAW C1
LET C1 BE CURVE (2,2),(4,6),(6,2),.2
DRAW C1
LET C1 BE CURVE (2,2),(4,6),(6,2),.5
DRAW C1
LET C1 BE CURVE (2,2),(4,6),(6,2),.9
DRAW C1
```

Would yield the following (cropped) output (Figure 2):

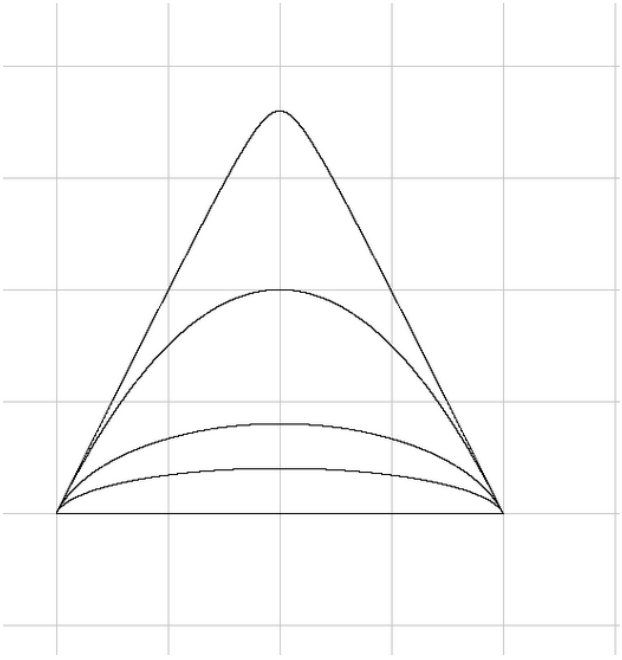

Figure 2

The grid line above the apex of the curve is 6 on the Y-axis, and as you would expect, the apex doesn't quite make it there because the highest curve factor is .9.

There are some additional special operands for the LET statement that are particular to the type of variable to which you are assigning. They are:

```
LET Pn BE START|END OF Cn
```

This sets the point variable to the first or last coordinate of the curve represented by the curve variable. This is invalid if the curve is a circle.

```
LET Pn BE VECTOR OF Cn
```

This sets the point variable to be equal to what the last coordinate of the curve *would be*, if the first coordinate was (0,0)

```
LET Xn BE LENGTH OF Cn
```

This sets the Value variable to be the "length" of the curve variable. Artspeak considers this length to be the straight line length between the first coordinate and the last coordinate.

DRAW Statement

The DRAW statement plots an object or objects. You must issue a DRAW statement if you want to cause any shapres to be output. The format of the DRAW statement is:

```
DRAW Ca [, Cb, Cc, Cd, …]
```

Note that multiple curve variables can be listed in a single DRAW statement

COPYxxx Statements

There are times when it is useful to copy objects to new variables. Artspeak has the following statements available for this:

```
COPYVALUE Xn TO Xm

COPYPOINT Pn TO Pm

COPYCURVE Cn TO Cm
```

All of these statements create a new object, and subsequently modifying the original object will not affect the new object.

EXPAND Statement

The EXPAND statement moves one point or curve away from another, by a specified factor:

```
EXPAND Pa FROM Pb, FACTOR n[.nnn]

EXPAND Ca FROM Pb, FACTOR n[.nnn]
```

Note that the object to be expanded is always expanded from a point. This point can be represented by variables or constants, but the object that is to be expanded *must* be a variable.

Here are two sample programs using EXPAND, and their outputs:

```
LET C1 BE CIRCLE, CENTER (3,3), RADIUS 1
DRAW C1
EXPAND C1 FROM (3,3), FACTOR 2
DRAW C1
```

(See figure 3)

```
LET C1 BE CIRCLE, CENTER (3,3), RADIUS 1
DRAW C1
EXPAND C1 FROM (3,2), FACTOR 2
DRAW C1
```

(See figure 4)

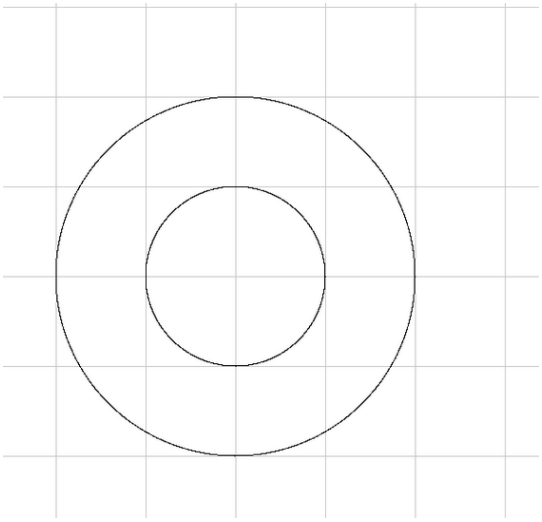

Figure 3

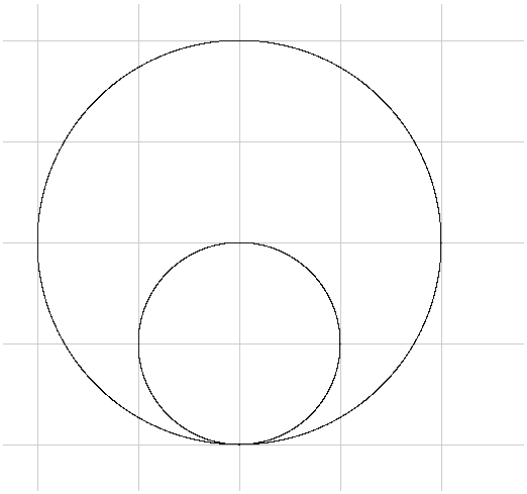

Figure 4

Note that EXPAND works on all of the types of curves (as well as points), and will expand each point that is part of the definition of the curve.

The MIRROR Statement

The MIRROR statement allows you to mirror a point or any curve to the other side of a line. The line is defined by exactly two points, but continues beyond those points to the ends of the graph bounds. As with EXPAND, each point defining the curve is affected.

```
MIRROR Pn IN (x1,y1)|Pn,(x2,y2)|Pn

MIRROR Cn IN (x1,y1)|Pn,(x2,y2)|Pn
```

Examples:

```
MIRROR C1 IN (2,5),(7,6)

MIRROR C1 IN P1,P2
```

The REFLECT Statement

The REFLECT statement is different from MIRROR in that it reflects about a point, as opposed to a line. The effect is like an old "pinhole camera".

```
REFLECT Pn IN (x1,y1)|Pn

REFLECT Cn IN (X1,y1)|Pn
```

Here is an example program and output (credit Henry Mullish):

```
LET C1 BE LINE (2,4),(2,1),(3,2),(2,1),(1,2)
DRAW C1
LET P1 BE POINT (4,3)
REFLECT C1 IN P1
DRAW C1
```

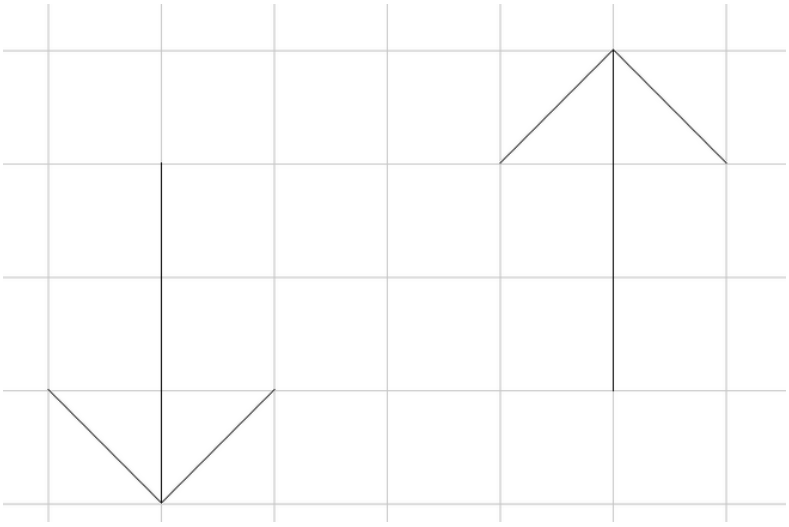

Figure 5

The ROTATE Statement

The ROTATE statement rotates a point or a curve a specified number of degrees around a point.

```
ROTATE Pn ABOUT (x,y)|Pm, ANGLE n[.nnn]|Xn

ROTATE Cn ABOUT (x,y)|Pm, ANGLE n[.nnn]|Xn
```

Here's a quick example:

```
LET P1 BE POINT (3,4)
LET C1 BE LINE P1,(5,6),(3,8),(1,6),(3,4)
DRAW C1
ROTATE C1 ABOUT P1 ANGLE 2
DRAW C1
ROTATE C1 ABOUT P1, ANGLE 2
DRAW C1
ROTATE C1 ABOUT P1, ANGLE 2
DRAW C1
ROTATE C1 ABOUT P1, ANGLE 2
DRAW C1
```

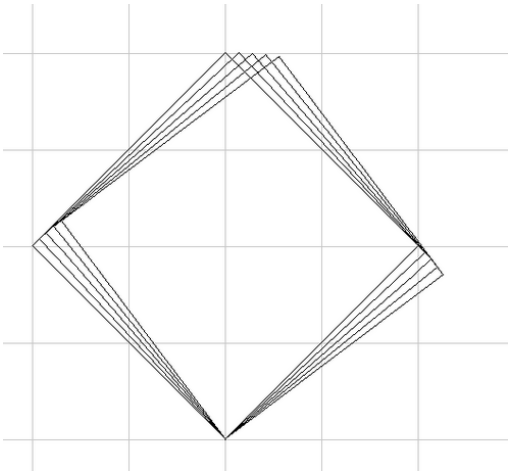

Figure 6

The REPEAT Statement and Labels

The tedium of that last example program calls out strongly for some sort of looping in Artspeak, and luckily, it does exist.

```
REPEAT Ln TO Lm, n TIMES
```

The same output as the above example could be accomplished as follows:

```
LET P1 BE POINT (3,4)
LET C1 BE LINE P1,(5,6),(3,8),(1,6),(3,4)
```

```
L1 DRAW C1
L2 ROTATE C1 ABOUT P1 ANGLE 2
REPEAT L1 TO L2, 4 TIMES
```

Note the "labels" before the DRAW and ROTATE statements. At any point later (or earlier) in the program these labels may (or may not) be used in a REPEAT statement or any other program flow statement described below. Labels must be at the start of a line. The lines to be repeated can be either *before or after* the line with the REPEAT statement.

The GO TO Statement

```
GO TO Ln
```

This statement simply transfers control to the line with the label indicated.

The DO Statement

```
DO Ln
```

This statement executes the line with the label indicated (only), and then continues from after the line with the DO statement.

The DUMMY Statement

The DUMMY statement does nothing.

The STOP Statement

The STOP statement terminates the program. It is not required, but is available.

The ADVANCE Statement

The ADVANCE statement moves a point by the value of another point.

```
ADVANCE Pn BY (x,y)|Pn
```

The X and Y values of the point variable are increased by the values of the 2nd operand. Note that the operand may contain negative X and Y values, for the purposes of moving in the opposite direction.

The MOVE Statement

The MOVE statement moves a curve by the x/y offsets of a given point or vector. The same conditions apply as to ADVANCE above, except that MOVE is for a curve.

```
MOVE Cn BY VECTOR (x,y)|Pn
```

The REDUCE Statement

```
REDUCE Xn BY Xm|n[.nnn]
```

While ADVANCE is only used for point, REDUCE is only used for values. Negative values may be used for Xm or n, and this is the only way to mathematically *increase* values of Value variables.

The MULTIPLY and DIVIDE Statements

```
MULTIPLY Xn BY Xm|n[.nnn]

MULTIPLY Pn BY (x,y)|Pm

DIVIDE Xn BY Xm|n[.nnn]

DIVIDE Pn BY (x,y)|Pm
```

The SCALE Statement

The SCALE statement only affect non-linear curves.

```
SCALE n[.nnn]
```

It sets a number by which the imaginary point and the 2nd endpoint coordinates for "curves" are multiplied for the duration of the program. The default, of course is 1.

The CAPTION Statement

The CAPTION statement allows the programmer to put text in the output.

```
CAPTION (x,y)|Pn ABC DEF GHI …
```

The coordinates (or point variable) specify where to *start* the text.

Comment Character

Any line with a 'C' in the first column will be treated as a comment.

The PRINT Statement

The PRINT statement can be used for debugging Artspeak programs. It can be used to display the value of point, curve, and value variables. These results will not be displayed on the graphical output, but on the console. You can debug multiple variables in the same PRINT statement.

```
PRINT Xn|Pn,Xm|Pm,…
```

Sample Programs, Followed by Output

```
LET C1 BE CIRCLE, CENTER (5,6), RADIUS 1
L1 DRAW C1
L2 ROTATE C1 ABOUT (5,5), ANGLE 6
REPEAT L1 TO L2, 59 TIMES
```

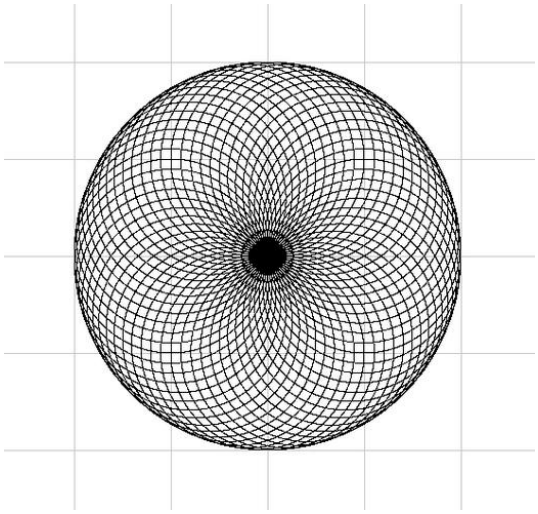

Figure 7

```
LET P1 BE POINT (3,3)
LET P2 BE POINT (3,7)
LET P3 BE POINT (7,7)
LET P4 BE POINT (7,3)
LET P5 BE POINT (5,5)
LET C1 BE LINE P1, P2, P3, P4, P1
L1 DRAW C1
EXPAND C1 FROM P5 FACTOR .98
L2 ROTATE C1 ABOUT P5, ANGLE 3
REPEAT L1 TO L2, 60 TIMES
```

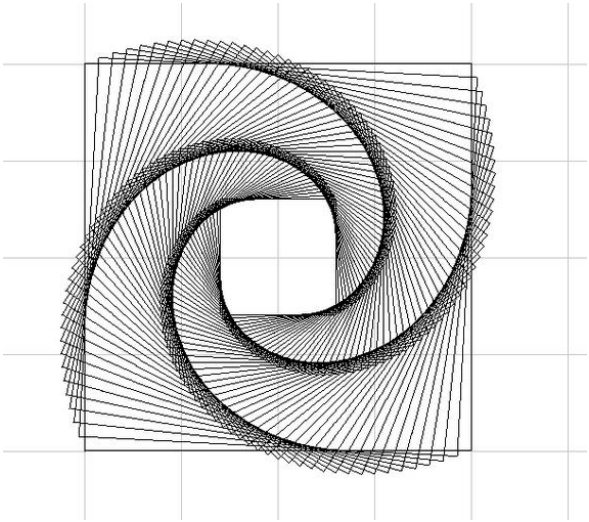

Figure 8

```
LET P1 BE POINT (2,2)
LET P2 BE POINT (2,8)
```

```
LET P3 BE POINT (8,8)
LET P4 BE POINT (8,2)
LET P5 BE POINT (5,5)
LET C1 BE LINE P1,P2,P3,P4,P1
L1 DRAW C1
ROTATE C1 ABOUT P5, ANGLE 180
DRAW C1
ADVANCE P2 BY (0,-.12)
ADVANCE P1 BY (.12,0)
L2 LET C1 BE LINE P1,P2
REPEAT L1 TO L2, 49 TIMES
```

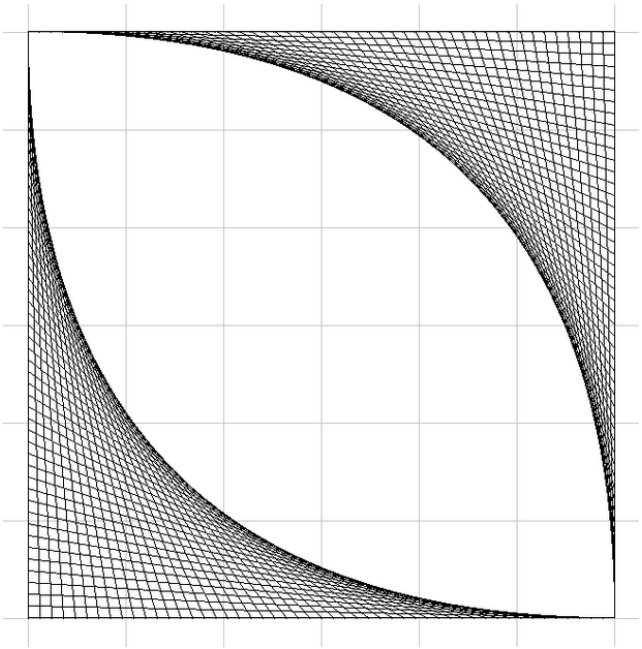

Figure 9

```
LET C4 BE LINE (3,3),(7,3),(7,7),(3,7),(3,3)
LET C5 BE LINE (4,4),(6,4),(6,6),(4,6),(4,4)
L1 DRAW C4
DRAW C5
ROTATE C4 ABOUT (5,5), ANGLE 5
L2 ROTATE C5 ABOUT (5,5), ANGLE 5
REPEAT L1 TO L2, 17 TIMES
```

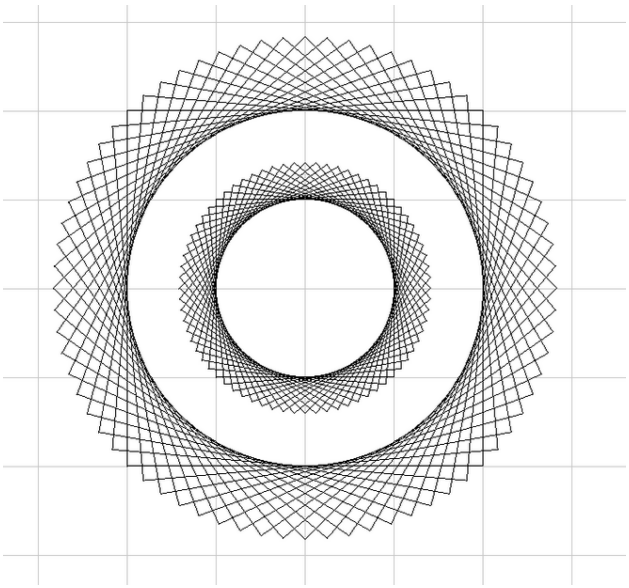

Figure 10

```
LET P5 BE POINT (5,5)
LET P1 BE POINT (5,2)
COPYPOINT P1 TO P2
L1 ROTATE P1 ABOUT P5, ANGLE 5
LET C1 BE LINE P1, P2
DRAW C1
L2 ROTATE P2 ABOUT P5, ANGLE 2.5
REPEAT L1 TO L2, 143 TIMES
```

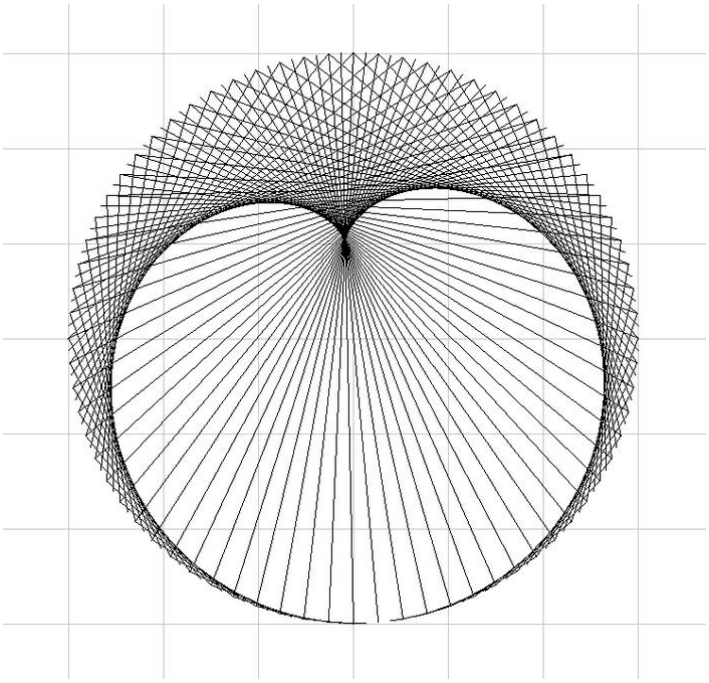

Figure 11

```
LET C1 BE LINE (2,3),(4,1),(5,5),(2,4),(3,6)
DRAW C1
MIRROR C1 IN (5,5),(6,5)
DRAW C1
```

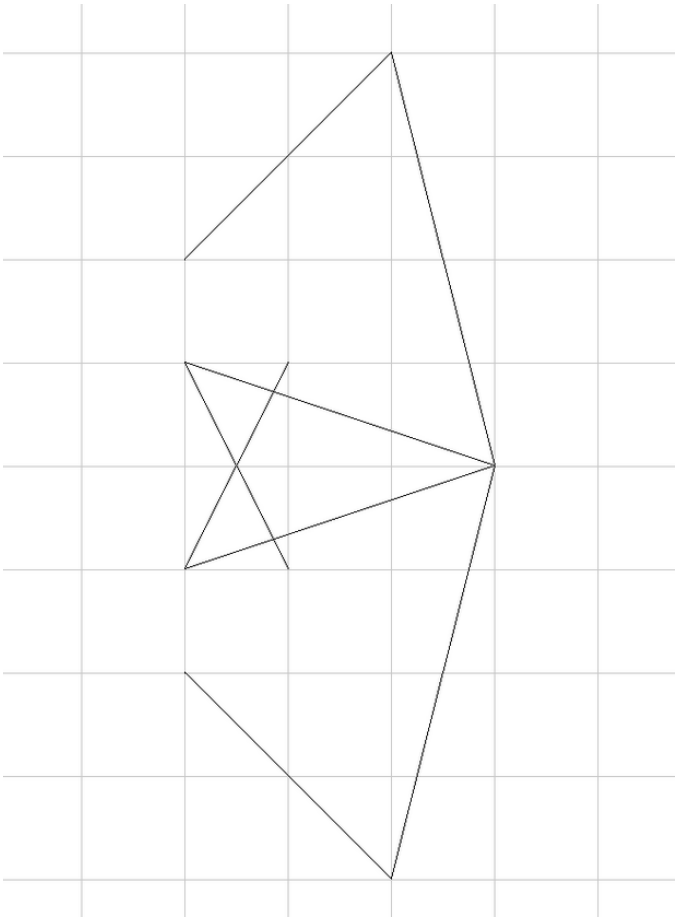

Figure 12

Here's one my daughter Mallory (Age 11) wrote:

```
let c1 be circle center (5,5) radius 2
draw c1
let c1 be line (5,3) (5,7)
draw c1
let c1 be line (7,5) (5,5)
rotate c1 about (5,5) angle 45
draw c1
rotate c1 about (5,5) angle 90
draw c1
```

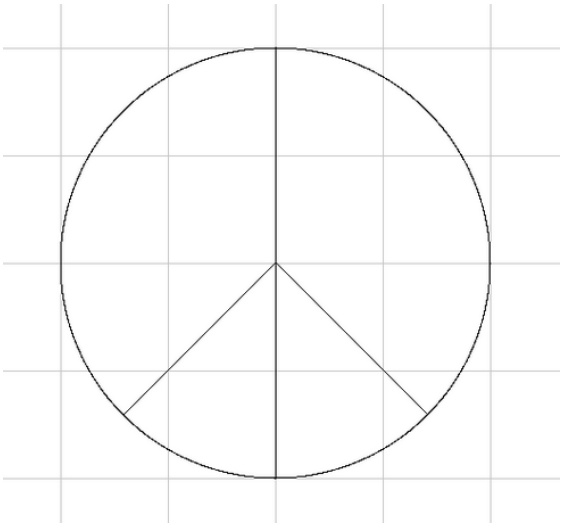

Figure 13