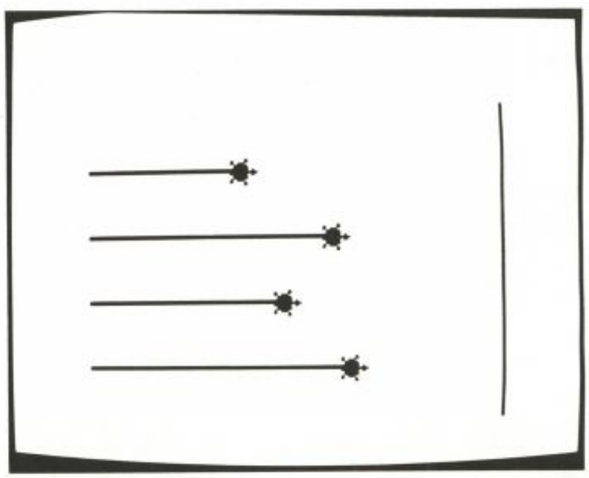
4

Turtle Geometry

Turtle Race

RACE shows four turtles racing from the left side of the screen to the right side. The winning turtle changes color. The background also changes color, to emphasize that someone has won the race. Here is a race in progress.



RACE.FROM does the real work by running first SETUP.RACE and then RUN.RACE. In other words, the program is divided into a setup part and an action part. The job of SETUP.RACE is to draw the racecourse and to assign colors and positions to the turtles. The job of RUN.RACE is to run the race.

```
TO RACE
RACE.FROM -120 120
END
```

The motion of the turtles is controlled by repeated use of the FORWARD command, not by using the dynamic (speed) ability of the turtles.

RACE is the top-level procedure. It knows where the left and right edges of the screen are and gives that information to RACE.FROM.

By Brian Harvey, with modifications by Jim Davis.

```
TO RACE.FROM :START :FINISH
SETUP.RACE
RUN.RACE
END
```

START and FINISH contain the *x* coordinates of the starting and ending positions. This information is used to set the turtles up at the start of the race and to find the winner.

Setting Up

SETUP.RACE has two tasks to do: set up the racecourse and prepare the four turtles as racers. It has one subprocedure to do each task. It hides the turtles when it starts, to avoid clutter during the setup.

```
TO SETUP.RACE
TELL [0 1 2 3]
HT
DRAW.RACETRACK
SETUP.RACERS
END
```

DRAW.RACETRACK is the procedure in charge of setting up the racecourse. This involves cleaning up the screen and setting its color, and then drawing the finish line.

```
TO DRAW.RACETRACK
SETBG 86
FS
CS
ASK 0 [DRAW.FINISHLINE]
END
```

DRAW.FINISHLINE draws a vertical line near the right edge of the screen.

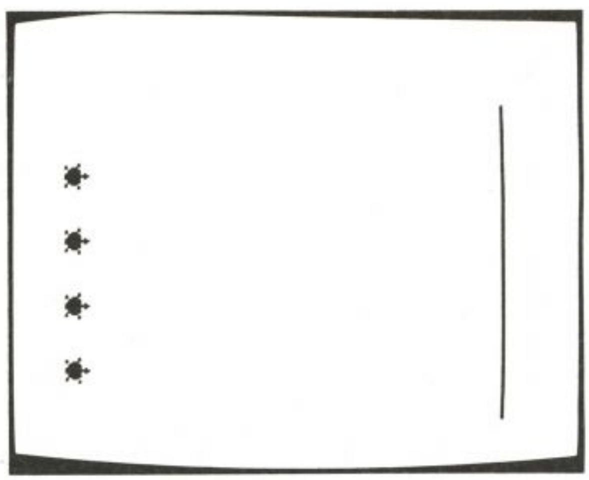
```
TO DRAW.FINISHLINE
PU
SETPOS LIST :FINISH 80
SETPN 1
SETPC 1 105
PD
BK 190
PU
END
```

SETUP.RACERS positions the four turtles at the starting point of the race, near the left edge of the screen. Some things are the same for all the turtles, like the RT 90 to point them toward the finish line. But two things are different for each turtle: the vertical position and the color. SETUP.RACERS uses the primitive command EACH to tell Logo to set these two properties for each turtle separately. In the instructions given as inputs to EACH, the particular value used for each turtle depends on the turtle number, represented by the primitive operation WHO. For example, the

SETC instruction will give turtle 0 color 11 (11+16 * 0), turtle 1 color 27 (11+16 * 1), and so on.

```
TO SETUP.RACERS
PU
EACH [SETPOS LIST :START WHO*40-80]     Vertical position different
RT 90                                               for each turtle.
EACH [SETC 11+16*WHO]                  Each turtle has different hue
ST                                                but same intensity.
SETPN 0
SETPC 0 90
PD
END
```

The result of running SETUP.RACE is shown here.



Running the Race

The race itself is handled by RUN.RACE, which moves the turtles one at a time until there is a winner. The command EACH is used to accomplish this one-at-a-time motion.

After each turtle moves, the operation WONP checks whether or not the turtle that moved has reached :FINISH and thus has won the race. If so, the procedure SHOW.WINNER is called to congratulate the winning turtle by changing its color. If there is no winner, the race continues.

```
TO RUN.RACE
EACH [MOVE1 IF WONP [SHOW.WINNER STOP]]
RUN.RACE
END
```

MOVE1 is invoked for each turtle in turn. It moves the turtle a small random amount. The distances are small, so repeating this procedure over and over will give a fairly smooth effect. The distances are random so that the race is different each time the program is run.

```
TO MOVE1
FD 6+RANDOM 20
END
```

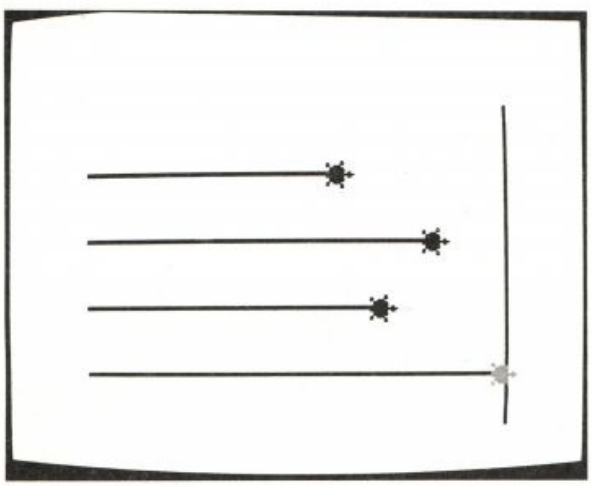
WONP checks the current turtle's position to see if it's past the finish line. If so, it outputs TRUE; otherwise, FALSE.

```
TO WONP
OP XCOR > :FINISH
END
```

SHOW.WINNER just changes the color of the winning turtle and the background color, to indicate that the race is over.

```
TO SHOW.WINNER
SETC 7
SETBG 84
END
```

This is the end of a race.



SUGGESTIONS

This race is unfair; lower-numbered turtles have a greater chance of winning, because they move first. Here's one way to fix it: judge the winner only when each turtle has had a chance to move. Then the operation WINNER would output a list of all winners. This is more egalitarian. Each could be bestowed an award.

How about a musical fanfare at the end?

On the other hand, if you like unfair races, perhaps the winning turtle should eat the other turtles, plunder their homelands, and so forth.

PROGRAM LISTING

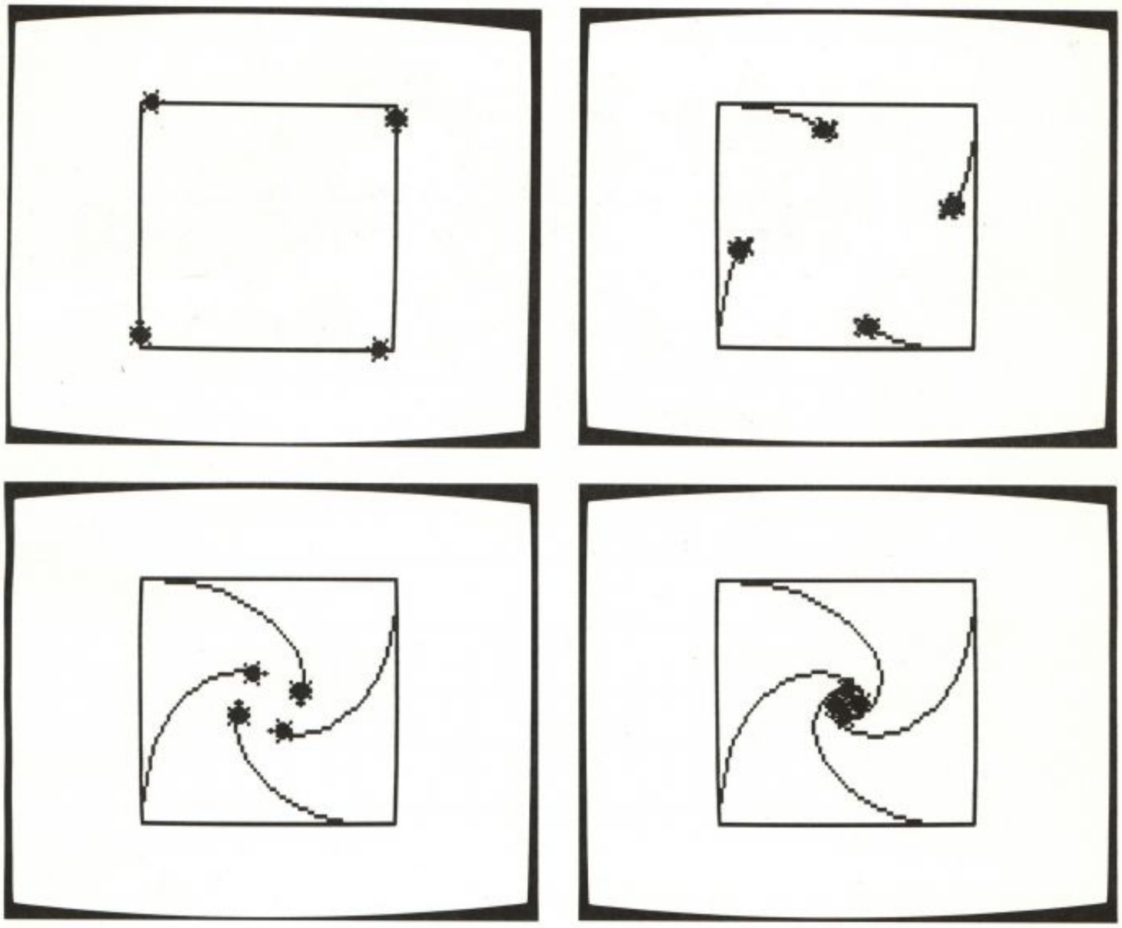
```
TO RACE
RACE.FROM -120 120
END

TO RACE.FROM :START :FINISH
SETUP.RACE
RUN.RACE
END

TO SETUP.RACE
TELL [0 1 2 3]
HT
DRAW.RACETRACK
SETUP.RACERS
END

TO DRAW.RACETRACK
SETBG 86
FS
CS
ASK 0 [DRAW.FINISHLINE]
END

TO DRAW.FINISHLINE
PU
SETPOS LIST :FINISH 80
SETPN 1
SETPC 1 105
PD
BK 190
PU
END

TO SETUP.RACERS
PU
EACH [SETPOS LIST :START WHO*40-80]
RT 90
EACH [SETC 11+16*WHO]
ST
SETPN 0
SETPC 0 90
PD
END

TO RUN.RACE
EACH [MOVE1 IF WONP [SHOW.WINNER ►
     STOP]]
RUN.RACE
END

TO MOVE1
FD 6+RANDOM 20
END

TO WONP
OP XCOR > :FINISH
END

TO SHOW.WINNER
SETC 7
SETBG 84
```

Four-Corner Problem

Here is a famous math problem: There are four ants, each at one corner of a square. Each ant faces the next one. They all start walking at the same time. As they walk, each ant turns so that it continues to face the same ant it was facing at the beginning. How far do the ants walk before they all meet at the center of the square?

This Logo program doesn't tell you how far they walk, but it does draw a picture to act out the problem. The only difference is that in this version of the problem we use turtles instead of ants.

By Brian Harvey.



PROGRAM LISTING

This project uses the TOWARDS procedure, which is shown later on in this chapter.

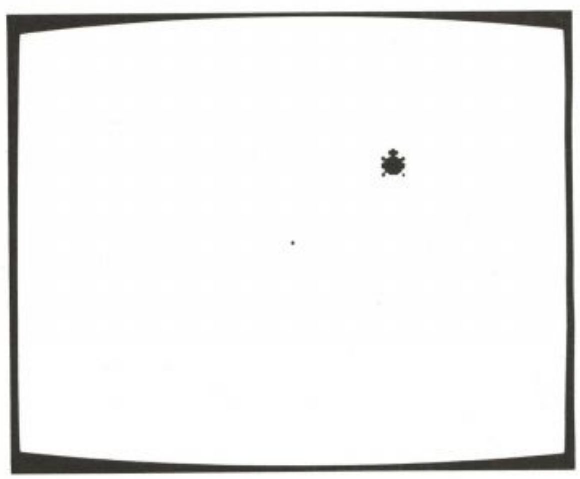
```
TO FOUR
TELL [Ø 1 2 3]
CT CS ST SETSH Ø PU FS
ASK Ø [SETPOS [-8Ø -8Ø]]
ASK 1 [SETPOS [-8Ø 8Ø]]
ASK 2 [SETPOS [8Ø 8Ø]]
ASK 3 [SETPOS [8Ø -8Ø]]
PD
ASK Ø [SETH Ø REPEAT 4 [FD 16Ø RT 9Ø]]
WAIT 6Ø SS
PR [EACH TURTLE KEEPS FACING THE NEXT]
PR [ONE AS THEY ALL MOVE FORWARD.]
WAIT 3ØØ
FS
FOUR.LOOP
END

TO FOUR.LOOP
EACH [SETH TOWARDS ASK REMAINDER (WHO+1) 4
[POS]] FD 1Ø
IF COND TOUCHING Ø 1 [STOP]
FOUR.LOOP
END
```

Towards and Arctan

TOWARDS is an operation that tells you how to turn the turtle to get it pointing toward a particular position. It takes one input, which is the position toward which you want to turn the turtle (in the form of a list of two coordinates). It outputs the heading to which the turtle should be turned in order to be facing from its current position to the input position. Here is an example. Start out with a clear screen with one dot in the middle.

```
CS
PD
FD Ø
PU
SETPOS [63 27]
```

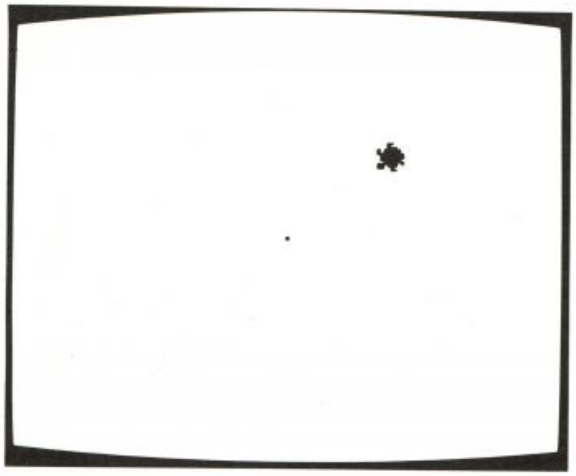
At this point, the turtle is facing north.



```
SETH TOWARDS [Ø Ø]
```

The turtle is now facing the dot we drew at the center of the screen.

This project is based on "Three Computer Mathematics Projects" by Hal Abelson, MIT Logo Laboratory Working Paper No. 16, June 20, 1974; write-up by Brian Harvey.



TOWARDS is defined in terms of the second tool in this package, the ARCTAN procedure. ARCTAN takes a number as input and gives as output the arctangent (in degrees) of that number. The procedure uses an approximation that is good to within about one degree, close enough for graphics!

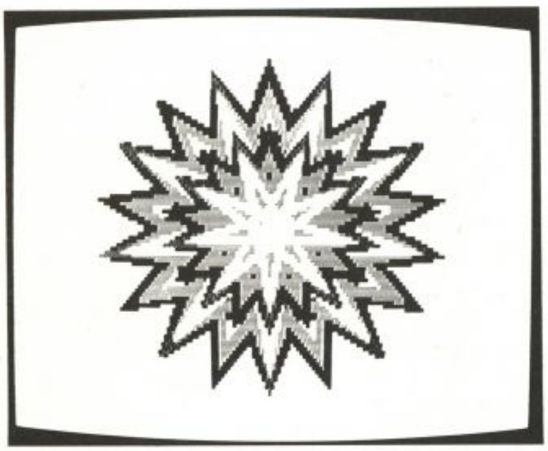
For those who have studied trigonometry: the TOWARDS procedure computes the differences between the x and y coordinates of the input position and those of the turtle's position, then takes the arctangent of $\Delta y/\Delta x$. The output from ARCTAN is the correct heading, except that attention must be paid to the positive or negative direction of the two differences. (If you haven't studied trig, don't worry about it. You can use TOWARDS without understanding its inner workings.)

PROGRAM LISTING

```
TO TOWARDS :POS
OP TOWARDS1 (FIRST :POS)-XCOR (LAST ▶
     :POS)-YCOR
END

TO TOWARDS1 :DX :DY
OP TOWARDS3 :DX :DY TOWARDS2 ABS :DX ▶
     ABS :DY
END

TO TOWARDS2 :DX :DY
IF :DX=Ø [OP Ø]
IF :DY=Ø [OP 9Ø]
OP ARCTAN (:DX/:DY)
END

TO TOWARDS3 :DX :DY :ANG
IF :DY<Ø [MAKE "ANG 18Ø-:ANG]
IF :DX<Ø [MAKE "ANG 36Ø-:ANG]
OP :ANG
END

TO ARCTAN :X
OP 57.3*ARCTAN.RAD :X
END

TO ARCTAN.RAD :X
IF :X>1 [OP 1.571-ARCTAN.RAD (1/:X)]
OP :X/(1+Ø.28*:X*:X)
END

TO ABS :X
OP IF :X<Ø [-:X] [:X]
END
```

Gongram:
Making Complex Polygon Designs

GONGRAM makes designs like the ones shown below.



To make the first design, type:

```
GONGRAM 14 11Ø 14Ø 16Ø 28 23 Ø
```

To make the second one, type:

```
GONGRAM 1Ø 11Ø 135 45 23 45 77
```

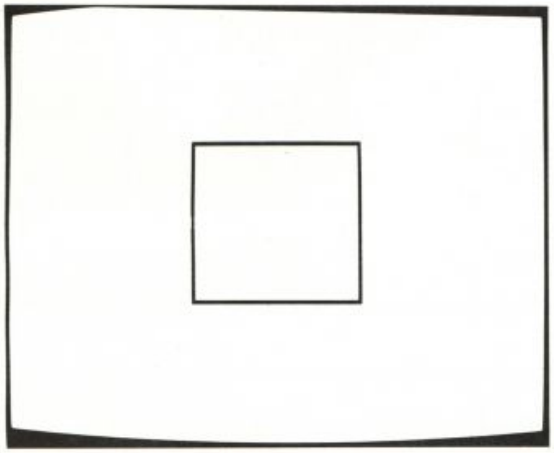
It takes a long time to make a gongram design since the turtle must draw many lines.

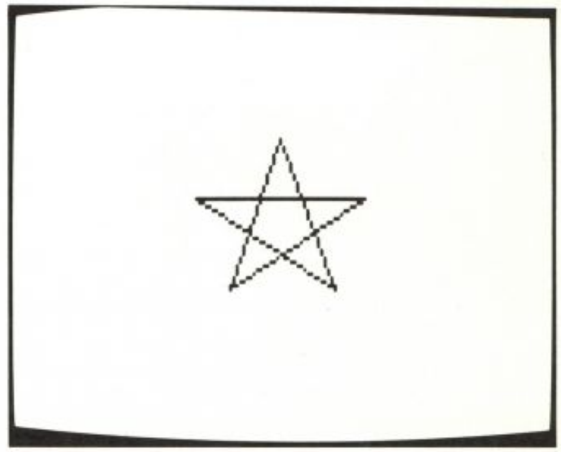
GONGRAM uses a variation of POLY, a procedure that makes a turtle draw polygons of different sizes and shapes. (See *Atari Logo Introduction to Programming Through Turtle Graphics,* p. 138, for a discussion of this procedure.)

```
TO POLY :SIDE :ANGLE
POLY1 :SIDE :ANGLE HEADING
END

TO POLY1 :SIDE :ANGLE :START
FD :SIDE
RT :ANGLE
IF HEADING = :START [STOP]
POLY1 :SIDE :ANGLE :START
END
```

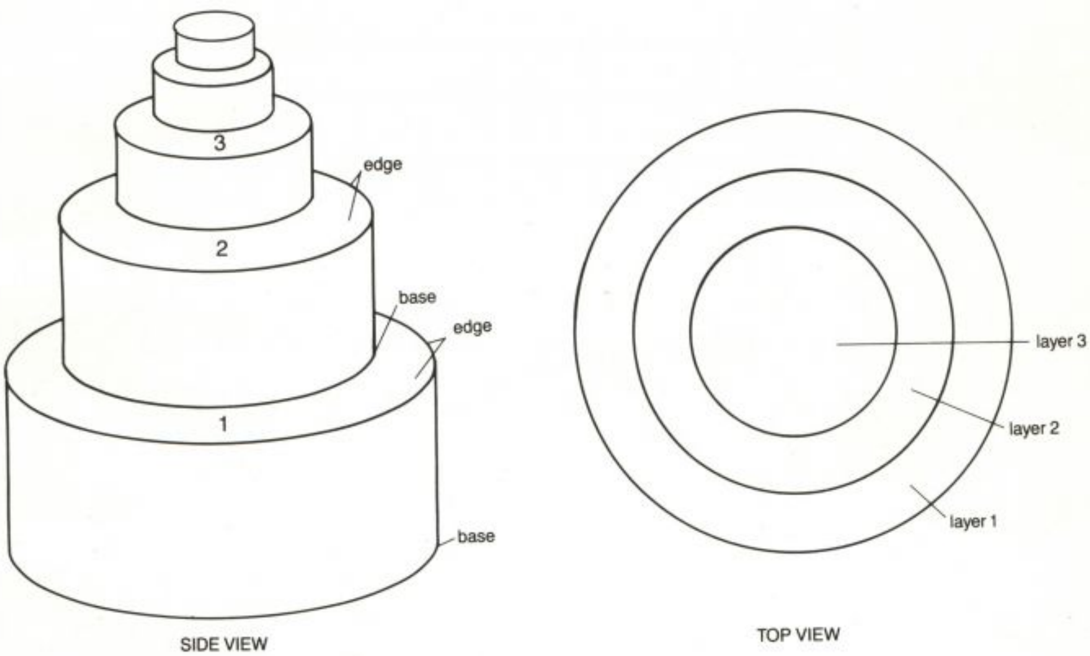
By Erric Solomon.

For example, `POLY 50 90` draws a square of side length 50; `POLY 50 144` draws a five-pointed star of side length 50.

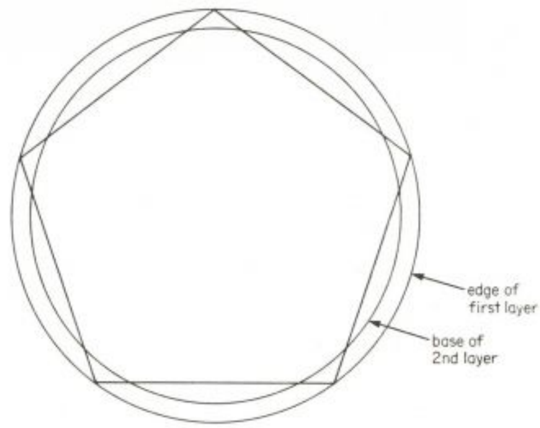




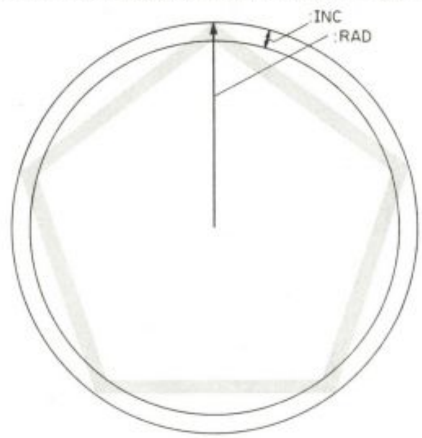
To help in understanding how a gongram design is made, imagine that you are directly above a layer cake.



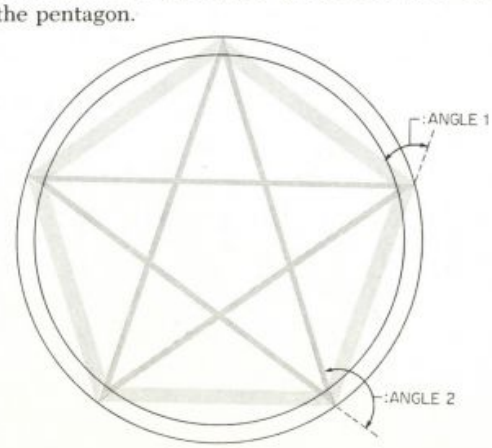
Each layer is slightly smaller than the one below it. Each layer of the cake is transparent, except when we draw on it. At the edge of the bottom layer, which we'll call the first layer, a pentagon is drawn.



Also drawn on the first layer is a smaller pentagon. It digs into the circle formed by the base of the layer above. (The second layer's base rests on the surface of layer 1.) The area between the two is filled in.



The result is a thick POLY shape. In a similar fashion we draw two five-pointed stars and fill the area between them in another color, sharing vertices with the pentagon.



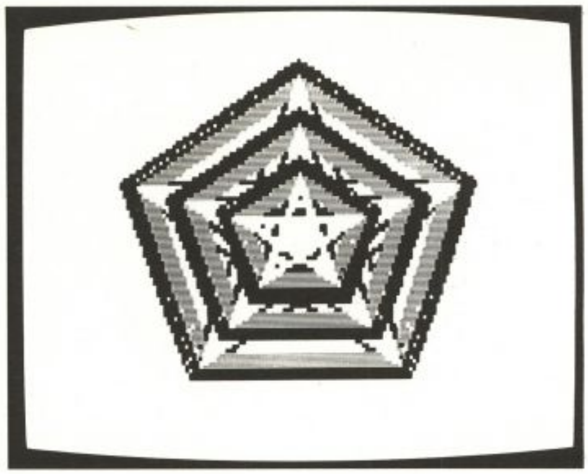
Notice that part of the pentagon is covered by the star. Now we move to the next layer. On the second layer we draw two new pentagons, one at the edge of the second layer, the next one inscribed into the circle formed by the base of the third layer. And as before, we fill in the area between them in a third color.

Notice that the view of parts of the star has been obstructed. Two new stars are inscribed in a similar manner, but instead of filling the area between them with a color, we rub an eraser over the area between the stars.

We skip over layer three, and at layer four we pretend that it is the bottom layer and repeat the process.

Of course, we didn't have to use a pentagon or a star. We could have chosen two other POLY-generated shapes. We could choose other colors.

Here is the completed design.



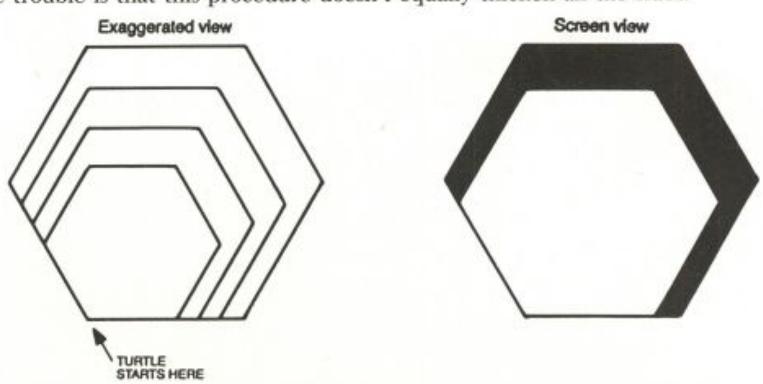
```
GONGRAM 10 110 72 144 30 62 102
```

Making a Filled-in POLY

Why not just use POLY several times, with a slightly different first input each time? That would produce several polygons of the same shape but slightly different sizes, one inside the other. For example, we could try this procedure:

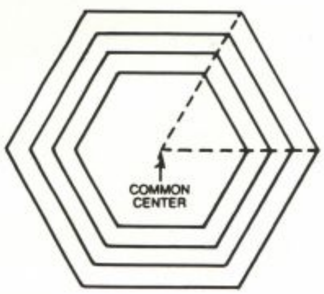
```
TO THICK.POLY :SIDE :ANGLE :THICKNESS
IF :THICKNESS=0 [STOP]
POLY :SIDE :ANGLE
THICK.POLY :SIDE - 1 :ANGLE :THICKNESS - 1
END
```

The trouble is that this procedure doesn't equally thicken all the sides.



We could try to solve this problem by moving the turtle in toward the center of the polygon a little before drawing the next polygon. But it's a bit complicated to figure out exactly how far to move the turtle, and in what direction, between POLYs.

The fundamental problem is that the successive POLYs are "anchored" to one vertex of the polygon, the one where the turtle starts. That vertex is at the same place on the screen for all the POLYs we draw. It would be better if we could anchor the polygons to a common *center* rather than to a common *vertex*.



This is how I approached the problem. First I noted that all the vertices of a POLY design are equidistant from the center of the design. Therefore, all the vertices of the POLY are points on the same circle, and each side of the design is a chord of the circle. Since :SIDE remains constant, each chord is the same length. It is possible, then, to inscribe a POLY into a circle by specifying the radius of the circle and the angle of the POLY design. If you inscribe a series of POLYs into concentric circles, then you get a thick or "filled-in" POLY.

I then wrote a different POLY and called it POLYGON. It places the turtle at points around a circle using the center of the screen as the center of the circle. As the turtle moves from point to point, the pen traces a line along each chord.

```
TO POLYGON :RADIUS :ANGLE
PU
SETPOS LIST :RADIUS * SIN HEADING :RADIUS * COS HEADING
PD
POLYGON1 :RADIUS :ANGLE HEADING
END

TO POLYGON1 :RADIUS :ANGLE :START
RT :ANGLE
SETPOS LIST :RADIUS * SIN HEADING :RADIUS * COS HEADING
IF HEADING = :START [STOP]
POLYGON1 :RADIUS :ANGLE :START
END
```

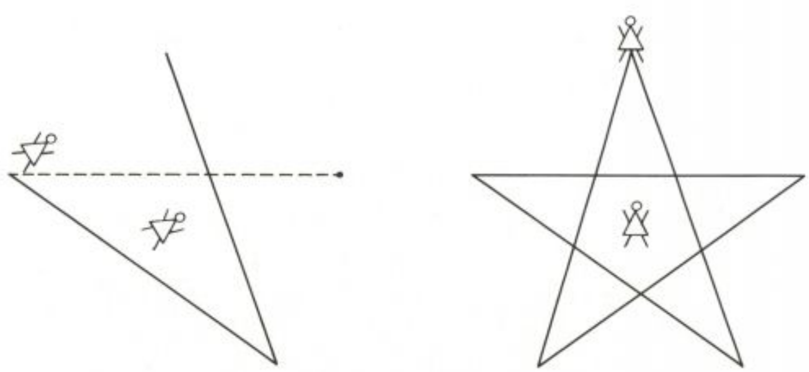
Another way to look at this procedure is to think of the turtle sitting in the middle of a circle. Each time the turtle turns, it points to a new vertex on the circle. If the turtle makes 90-degree turns each time, it will point to four distinct vertices. If the turtle turns 144 degrees each time, it will point

to five distinct vertices. These vertices could be connected in several different ways. The order in which the turtle points to them is the order in which they will be connected.

In the earlier version of POLY, the turtle always faces in the direction of the next side to be drawn. After it draws each side, the RIGHT turn points the turtle so that the next FORWARD will draw the next side.

In the new version, the turtle does *not* face in the direction of the next side. That's why the sides are drawn using SETPOS instead of FORWARD; we tell Logo where the next vertex is, instead of telling it the distance and direction. But the turtle's heading is still important in this version of POLY. It is always the heading that a turtle *in the center of the polygon* would face in order to point to the next vertex. To see how this works, watch a version of POLYGON where two turtles are visible. Turtle 0 will actually draw the polygon; turtle 1 will sit in the center of the screen, but will keep turning to retain the same heading as turtle 0.

```
TO VIEWPOLYGON :RADIUS :ANGLE
TELL [Ø 1] ST
PU
ASK Ø SETPOS LIST :RADIUS*SIN HEADING :RADIUS*COS HEADING]
PD
VIEWPOLYGON1 :RADIUS :ANGLE HEADING
END

TO VIEWPOLYGON1 :RADIUS :ANGLE :START
RT :ANGLE
ASK Ø [SETPOS LIST :RADIUS*SIN HEADING :RADIUS*COS HEADING]
IF HEADING = :START [STOP]
VIEWPOLYGON1 :RADIUS :ANGLE :START
END
```



Now that we've made a POLYGON that inscribes the design into a circle, a POLY.FILL is possible.

```
TO POLY.FILL :RADIUS :ANGLE
IF :RADIUS = Ø [STOP]
POLYGON :RADIUS :ANGLE
POLY.FILL :RADIUS - 1 :ANGLE
END
```

But we want to make a POLY.FILL that will make a POLYGON of any thickness.

```
TO POLY.FILL :HI :LO :ANGLE
POLYGON :HI :ANGLE
IF NOT :HI > :LO [STOP]
POLY.FILL :HI - 1 :LO :ANGLE
END
```

And now GONGRAM:

```
TO GONGRAM :INC :RAD :ANGLE1 :ANGLE2 :PC1 :PC2 :PC3
IF :RAD < 31 [STOP]
SETPC 1 :PC1
TELL Ø SETPN 1
POLY.FILL :RAD :RAD - :INC :ANGLE1
SETPC 2 :PC2 SETPN 2
POLY.FILL :RAD :RAD - :INC :ANGLE2
SETPC Ø :PC3 SETPN Ø
POLY.FILL :RAD - :INC :RAD - 2 * :INC :ANGLE1
ERASE.POLY.FILL :RAD - :INC :RAD - 2 * :INC :ANGLE2
GONGRAM :INC :RAD - 3 * :INC :ANGLE1 :ANGLE2  :PC1 :PC2 :PC3
END
```

GONGRAM takes seven inputs.

:INC	The distance between the edge of a layer and the base of the layer on top of it.
:RAD	The radius of the largest layer.
:ANGLE1	An angle that dictates the shape of one of the polygons inscribed on the cake. In our example it is 72 (the pentagon).
:ANGLE2	An angle that dictates the shape of one of the polygons inscribed on the cake. In our example it is 144 for the star.
:PC1	The pen color for pen 1. In this example it is 30 for red.
:PC2	The pen color for pen 2. In this example it is 62 for blue.
:PC3	The pen color for pen 0. In this example it is 102 for green.

ERASE.POLY.FILL is the only procedure I haven't mentioned. It is just like POLY.FILL except that it calls ERASE.POLY. ERASE.POLY is just like POLYGON except that it puts the pen into eraser, or PE, mode.

Here are some nice examples of GONGRAM.

```
GONGRAM 1Ø 11Ø 135 45 23 45 77
GONGRAM 15 11Ø 9Ø 135 23 45 77
GONGRAM 14 11Ø 14Ø 16Ø 28 23 Ø
GONGRAM 12 11Ø 9Ø 12Ø 45 6Ø 23
GONGRAM 5 11Ø 4Ø -1ØØØ 23 Ø 45
GONGRAM 15 11Ø 6Ø 12Ø 6Ø 45 23
GONGRAM 15 11Ø 12Ø 6Ø 45 23 77
```

Note: Some of these take a very long time to draw.

PROGRAM LISTING

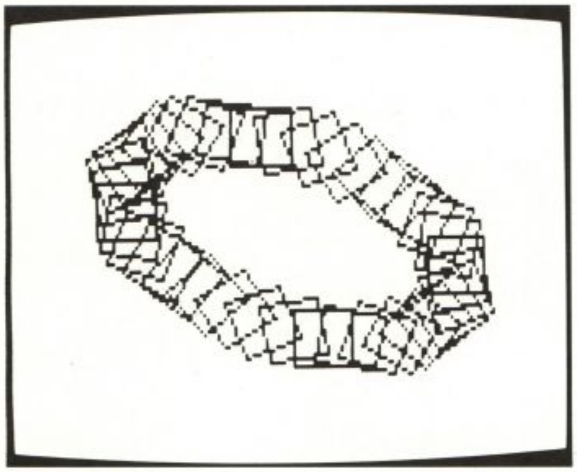
```
TO GONGRAM :INC :RAD :ANGLE1 :ANGLE2 ►
     :PC1 :PC2 :PC3
IF :RAD < 31 [STOP]
SETPC 1 :PC1
TELL Ø SETPN 1
POLY.FILL :RAD :RAD - :INC :ANGLE1
SETPC 2 :PC2 SETPN 2
POLY.FILL :RAD :RAD - :INC :ANGLE2
SETPC Ø :PC3 SETPN Ø
POLY.FILL :RAD - :INC :RAD - 2 * :INC ►
     :ANGLE1
ERASE.POLY.FILL :RAD - :INC :RAD - 2 * ►
     :INC :ANGLE2
GONGRAM :INC :RAD - 3 * :INC :ANGLE1 ►
     :ANGLE2  :PC1 :PC2 :PC3
END

TO POLY.FILL :HI :LO :ANGLE
POLYGON :HI :ANGLE
IF NOT :HI > :LO [STOP]
POLY.FILL :HI - 1 :LO :ANGLE
END

TO POLYGON :RADIUS :ANGLE
PU
SETPOS LIST  :RADIUS * SIN HEADING ►
     :RADIUS * COS HEADING
PD
POLYGON1 :RADIUS :ANGLE HEADING
END

TO POLYGON1 :RADIUS :ANGLE :START
RT :ANGLE
SETPOS LIST :RADIUS * SIN HEADING ►
     :RADIUS * COS HEADING
IF HEADING = :START [STOP]
POLYGON1 :RADIUS :ANGLE :START
END

TO ERASE.POLY.FILL :HI :LO :ANGLE
IF NOT :HI > :LO [STOP]
ERASE.POLY :LO :ANGLE
ERASE.POLY.FILL :HI :LO + 1 :ANGLE
END

TO ERASE.POLY :RADIUS :ANGLE
PU SETPOS LIST :RADIUS * SIN HEADING ►
     :RADIUS * COS HEADING
PE
RT :ANGLE
POLYGON :RADIUS :ANGLE
END
```

Polycirc

POLYCIRC makes designs by drawing polygons or lines around the circumference of an imaginary circle. As the turtle walks around the circumference, its heading changes as well as its position. Thus the polygons are drawn at different angles.

```
POLYCIRC 35 9Ø 1Ø 8Ø 1
```

By Erric Solomon.



POLYCIRC takes five inputs: :SIZE, :ANGLE, :INC, :RAD, :TIMES. POLYCIRC calls two procedures: POLY and NEXTPEN.

```
TO POLYCIRC :SIZE :ANGLE :INC :RAD :TIMES
RT :INC / :TIMES
PU SETPOS LIST (:RAD * COS :TIMES * HEADING)
               (:RAD * SIN :TIMES * HEADING)
PD
NEXTPEN
POLY :SIZE :ANGLE
IF HEADING = Ø [STOP]
POLYCIRC :SIZE :ANGLE :INC :RAD :TIMES
END
```

POLY draws polygons.

```
TO POLY :SIZE :ANGLE
POLY1 :SIZE :ANGLE HEADING
END
```

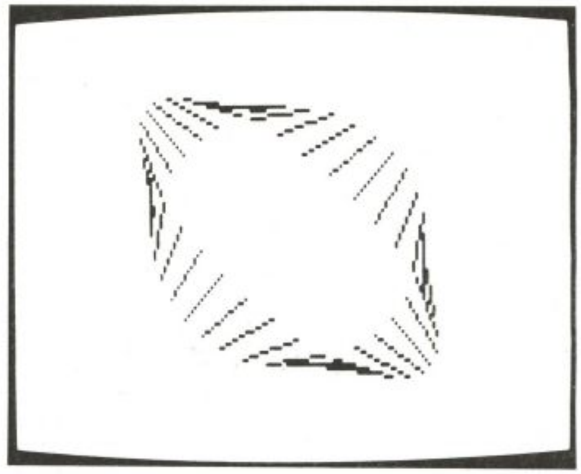
```
TO POLY1 :SIZE :ANGLE :HEAD
RT :ANGLE
FD :SIZE
IF HEADING = :HEAD [STOP]
POLY1 :SIZE :ANGLE :HEAD
END
```

NEXTPEN changes the pen each time it is called.

```
TO NEXTPEN
IF PN = 2 [SETPN Ø] [SETPN PN + 1]
END
```

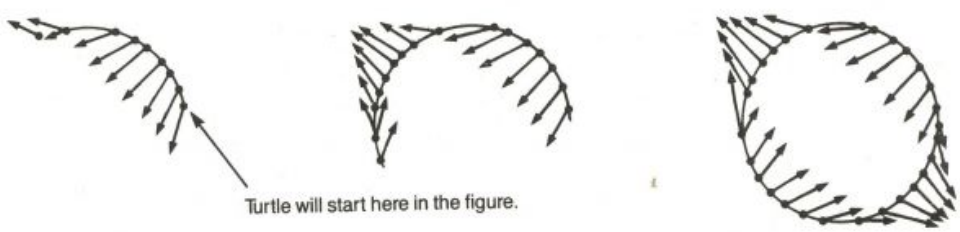
Try:

```
POLYCIRC 35 18Ø 1Ø 8Ø 1
```



Notice that this POLYCIRC is similar to the one in the first example, except that it draws a spoke instead of a square.

The following diagram might help you in understanding POLYCIRC.



Two circles are implicit in the figure. One is stationary and has its center in the center of the screen. The other can be thought of as a rolling wheel whose center is always found on the circumference of the stationary circle. This wheel has just one spoke. As the wheel rolls along the circumference of the central circle, this spoke turns. In the figure, the wheel turns one full revolution for every trip around the circle. At the same time, a trace of the spoke is left every 10 degrees around the circle.

To help you understand how the program works, you can make the central circle visible and then watch the spokes being drawn. First draw the central circle this way:

```
POLYCIRC 2 18Ø 2 5Ø 1
```

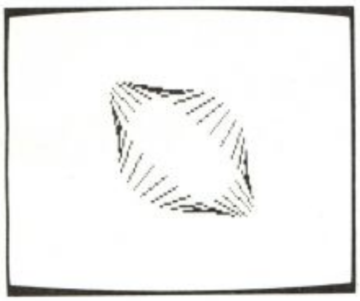
Then type the following to see the spokes being drawn around it.

```
POLYCIRC 5Ø 18Ø 1Ø 5Ø 1
```

We have used spokes in this example for their visual clarity, to help you understand how the program places polygons around a circle. A spoke is simply a POLY using an angle of 180 degrees. To draw other kinds of polygons, use a different angle. For example, the square POLYCIRC at the beginning of this section was drawn using an angle of 90 degrees.

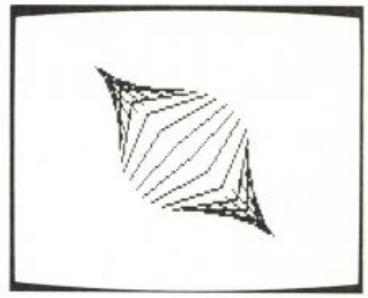
Several inputs or parameters of the design can be varied.

:SIZE, the first input, is the radius of the rolling wheel and thus the length of each side of the polygon.



POLYCIRC 30 180 10 60 1

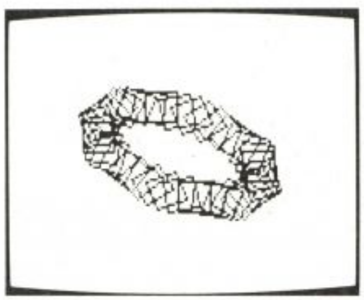
POLYCIRC 10 180 10 60 1



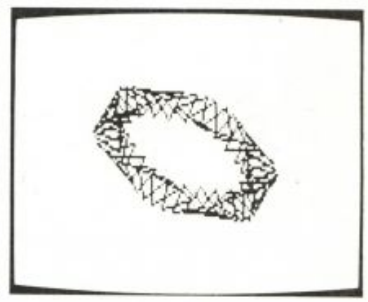
POLYCIRC 60 180 10 60 1

:ANGLE, the second input, is the angle that determines the shape of the polygon. If :ANGLE is 180, just a spoke is drawn.

POLYCIRC 30 180 10 60 1

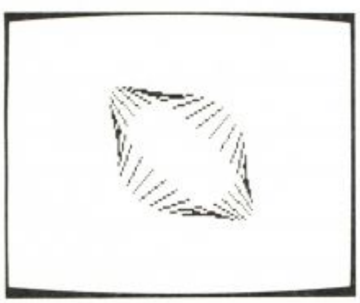


POLYCIRC 30 90 10 60 1

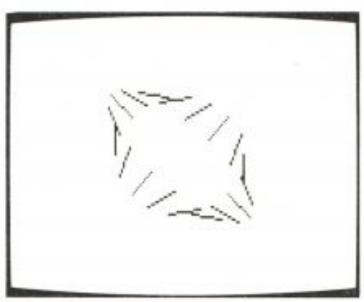


POLYCIRC 30 120 10 60 1

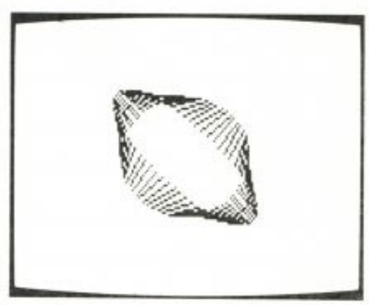
:INC, the third input, is inversely related to the density of the polygons.



POLYCIRC 30 180 10 60 1

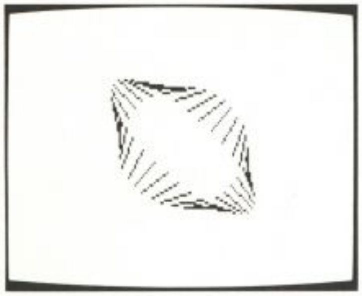


POLYCIRC 30 180 20 60 1



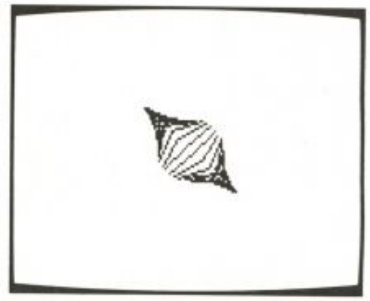
POLYCIRC 30 180 5 60 1

:RAD, the fourth input, is the radius of the center circle.



POLYCIRC 30 180 10 60 1

POLYCIRC 30 180 10 15 1

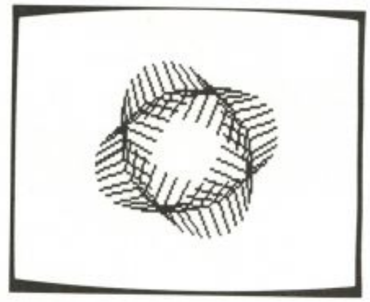


POLYCIRC 30 180 10 30 1

:TIMES, the fifth input, is the number of rotations of the wheel around its own center for each revolution it makes around the central circle. (For example, for each revolution of the earth around the sun, it makes 365 rotations, more or less.)*

POLYCIRC 30 180 10 60 1

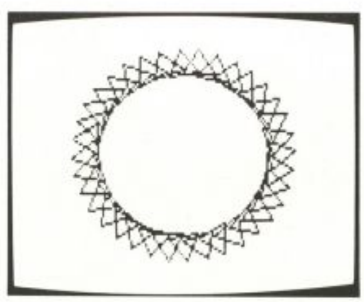
POLYCIRC 30 180 10 60 2



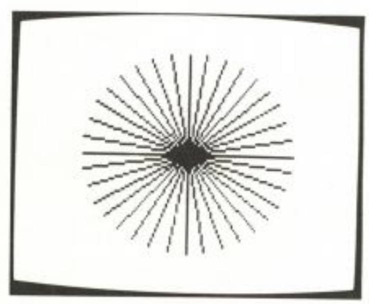
POLYCIRC 30 180 10 60 3

Other inputs for POLYCIRC that you might try are:

POLYCIRC 30 120 5 90 2



POLYCIRC 30 120 10 80 -1



POLYCIRC 100 180 10 0 1

*This input can never be zero.

PROGRAM LISTING

```
TO POLYCIRC :SIZE :ANGLE :INC :RAD ►
     :TIMES
RT :INC / :TIMES
PU SETPOS LIST (:RAD * COS :TIMES * ►
     HEADING) (:RAD * SIN :TIMES * ►
     HEADING)
PD
NEXTPEN
POLY :SIZE :ANGLE
IF HEADING = Ø [STOP]
POLYCIRC :SIZE :ANGLE :INC :RAD :TIMES
END

TO POLY :SIZE :ANGLE
POLY1 :SIZE :ANGLE HEADING
END

TO POLY1 :SIZE :ANGLE :HEAD
RT :ANGLE
FD :SIZE
IF HEADING = :HEAD [STOP]
POLY1 :SIZE :ANGLE :HEAD
END

TO NEXTPEN
IF PN = 2 [SETPN Ø] [SETPN PN + 1]
END
```

Animating Line Drawings

In Atari Logo you can change the color of lines already drawn on the screen. This feature can be used to animate drawings. I will give three examples. In each of them all three pens are used to make a drawing. Then the drawings are transformed from static to moving pictures. This is done by changing pen colors.

The first example is of spinning spokes. The other two examples show how color changes affect designs made by GONGRAM and POLYCIRC, programs that are described in other sections of this book.

Spinning Spokes

STAR draws 36 lines as if they were spokes of a wheel. As it draws lines, the turtle switches from pen 0, to pen 1, to pen 2, to pen 0, and so on until all the spokes are drawn. STAR puts the background's color in pens 1 and 2 so that their lines are not visible to the user while the design is being made. Lines drawn by pen 0 are visible; thus every third spoke is displayed on the screen.

After it draws each spoke, STAR calls NEXTPEN, which changes the pen.

By Erric Solomon.

```
TO STAR
SETPC 0 50
SETPC 1 BG
SETPC 2 BG
REPEAT 36 [FD 100 BK 100 RT 10 NEXTPEN]
END

TO NEXTPEN
IF PN = 2 [SETPN 0] [SETPN PN + 1]
END
```

Now try:

```
CYCLE 200 5
```

CYCLE animates the picture; it displays spoke after spoke by changing the pen colors. The first input to CYCLE is the number of times the animation will be repeated. The second input controls the delay (in sixtieths of a second) between shifts of pen colors. You can think of this time delay as the length of time between frames in the animation.

```
TO CYCLE :TIMES :DELAY
REPEAT :TIMES [CYC PC 1 :DELAY]
END

TO CYC :PC :DELAY
SETPC 1 PC 0
SETPC 0 PC 2
SETPC 2 :PC
WAIT :DELAY
END
```

The basic idea in this example is that two pens are always "hidden," but CYCLE keeps changing which two are hidden. Lines drawn by pen 1 change to the color previously assumed by lines drawn by pen 0. Pen 0's lines change to the color in pen 2. Lines drawn by pen 2 change to the color that used to be in pen 1. This color is given to CYC as an input.

Color Change with Gongram and Polycirc

CYCLE can work its magic in other situations as well. Let's try it with GONGRAM. In this example, all three pens have visible colors. (The last three inputs to GONGRAM are the colors for the pens.) Now run CYCLE and watch the result of this color shift.

```
GONGRAM 15 120 72 144 43 77 22
CYCLE 200 5
```

The following creates the same gongram pattern, but with two pens in the background color.

```
GONGRAM 15 130 72 144 43 BG BG
CYCLE 200 5
```

POLYCIRC is also animated by color shifting. Try this:

```
SETPC 1 BG
SETPC 2 BG
SETPC Ø 55
POLYCIRC 35 9Ø 1Ø 8Ø 1
CYCLE 3ØØ 5
```

You should see squares moving around in an elliptical path.

PROGRAM LISTING

For a listing of GONGRAM, see page 222; for POLYCIRC, see page 227.

```
TO STAR
SETPC Ø 5Ø
SETPC 1 BG
SETPC 2 BG
REPEAT 36 [FD 1ØØ BK 1ØØ RT 1Ø ►
     NEXTPEN]
END

TO NEXTPEN
IF PN = 2 [SETPN Ø] [SETPN PN + 1]
END

TO CYCLE :TIMES :DELAY
REPEAT :TIMES [CYC PC 1 :DELAY]
END

TO CYC :PC :DELAY
SETPC 1 PC Ø
SETPC Ø PC 2
SETPC 2 :PC
WAIT :DELAY
END
```