Chapter 8. Recursion

"Well, you sure picked a good subject this time," said Morf, looking a bit puzzled.

"We touched on this before, don't you remember?" replied Logy. "You remember, recursion is when a procedure uses itself as part of the solution."

"Say what?"

Yes, strange as that may seem, a recursive procedure is one that calls itself as part of the total solution. You saw examples of this in earlier chapters.

It is something like the two turtles in the picture. Each turtle is using the other to draw itself.

Is Life Recursive

Here's a fun procedure to help you make some sense out of recursion, even though it really isn't Logo.

```
TO GET.THROUGH.LIFE
GET.THROUGH.TODAY
GET.THROUGH.LIFE
END
```

Think about it for a moment. This says that to get through life, you have to get through today. Once you are through today, you have to move on, right?

But where?

You can't go backward. You can't stop time. You have to get through life. But to get through life, you have to get through today. But each day is different. So while this may look like a simple loop, it really isn't. Have you ever repeated a day, doing exactly what you did yesterday?

Let's add another twist to this. Let's suppose that when you're standing in front of the Pearly Gates to Heaven, you want to take a look at your Book of Life.

Embedded recursion can actually help you with this. We talk about different types of recursion in the next section. Anyway, to see your Book of Life, all you have to do is add a couple of lines to your life procedure.

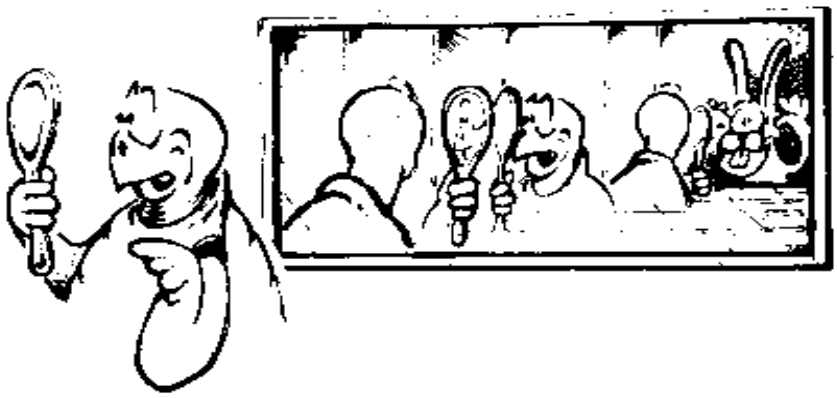
```
TO GET.THROUGH.LIFE
IF LIFE = "OVER [STOP]
GET.THROUGH.TODAY
GET.THROUGH.LIFE
PRINT BOOK.OF.LIFE
END
```

It's like every time you GET.THROUGH.TODAY, you write a page in your Book of Life. Only the pages are not printed until LIFE = "OVER. When the procedure stops, all the pages are printed starting with the last page it saved.

This may seem very confusing right now. But it will begin to make sense very shortly.

Tail-end Recursion

Yes, recursion is confusing.



It reminds you of the images you see when you look in two mirrors — without Morf getting in the way, that is. So let's try it with something we know something about.

```
TO MAZE :N
FD :N RT 90
MAZE :N + 5
END
```

This is an example of what you call "tail-end recursion." The recursive call is at the tail-end of the procedure.

To see just how this works, type

```
MAZE 20
```

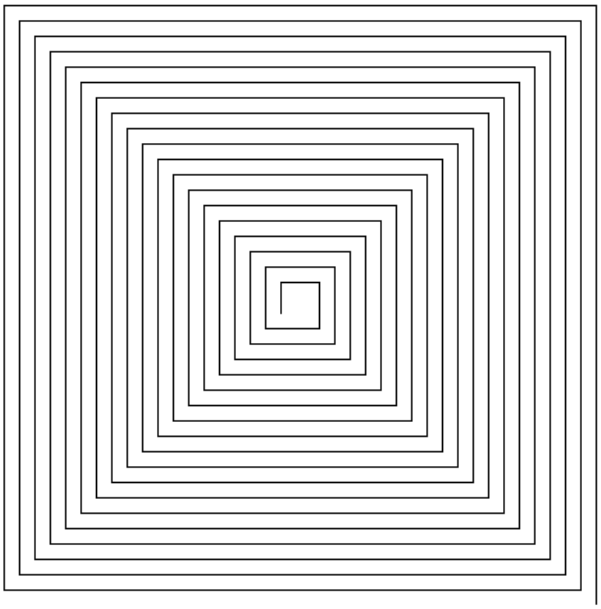
Now watch what happens.

The turtle goes forward 20 steps and then turns right 90 turns. Then the procedure tells :N (that's 20 to us) to become :N + 5 (that's 25 to us now). Then the procedure says

```
MAZE 25
```

and starts all over again.

MAZE 25 becomes MAZE 30. MAZE 30 become MAZE 35, and on and on and on. The screen soon looks something like what you see below. And it just keeps on going, gradually filling up the screen.



Of course, we can put a STOP in there if we want.

```
TO MAZE :N
IF :N = 200 [STOP]
FD :N RT 90
MAZE :N + 5
END
```

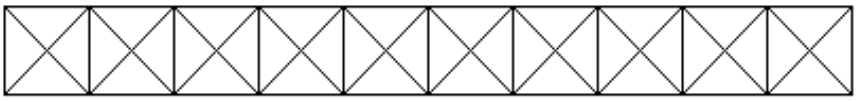
The first line sets up a conditional test. Each time the procedure runs, it tests :N to see if it equals 200. When :N does equal 200, the procedure stops.

The Turtle's Erector Set

Let's take a look at some other examples

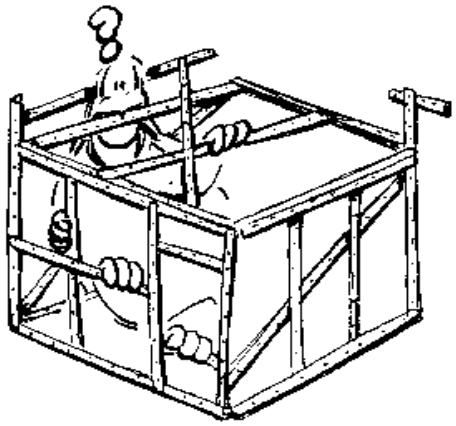
Did you ever play with an Erector Set? Did you ever build bridges? Maybe some buildings?

Well, Logo can help you draw your plans.



```
TO ERECTORSET :N :X
IF :X = 0 [STOP]
SECTION :N
MOVE :N
ERECTORSET :N :X - 1
END

TO SECTION :N
REPEAT 4 [TRI :N FD :N RT 90]
END

TO MOVE :N
RT 90 FD :N LT 90
END

TO TRI :N
FD :N RT 135
FD :N / SQRT 2 RT 90
FD :N / SQRT 2 RT 135
END
```

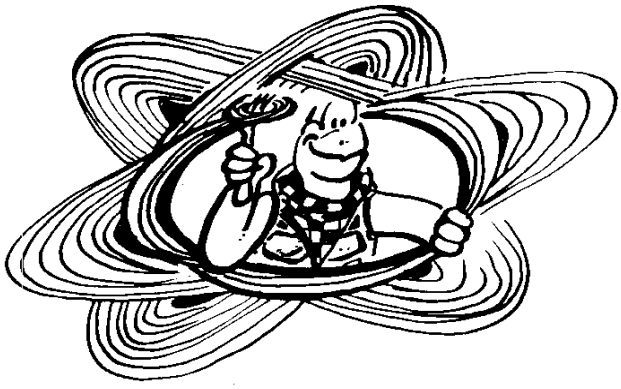
Uh, oh!



The Italian Turtle

Do you like Italian food? How about spaghetti?

```
TO SPAGHETTI
CIRCLE 5
CIRCLE 4
CIRCLE 3
CIRCLE 2
RT 45
SPAGHETTI
END
```

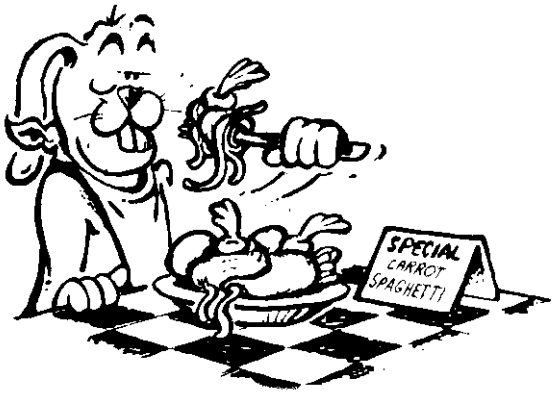


```
TO CIRCLE :N
REPEAT 36 [FD :N RT 10]
END
```

You can make the SPAGHETTI procedure even more variable with a few changes:

```
TO SPAGHETTI :N
CIRCLE :N + 5
CIRCLE :N + 4
CIRCLE :N + 3
```

```
CIRCLE :N + 2
RT 45
SPAGHETTI :N
END
```



How about this one?

```
TO SPAGHETTI :N
CIRCLE :N
IF :N = 0 [STOP]
RT 45
SPAGHETTI :N - 1
END
```

This recursive procedure is like some others you have used before. Will it draw spaghetti like the other procedures above? Why? Better yet, why not?

What about this one?

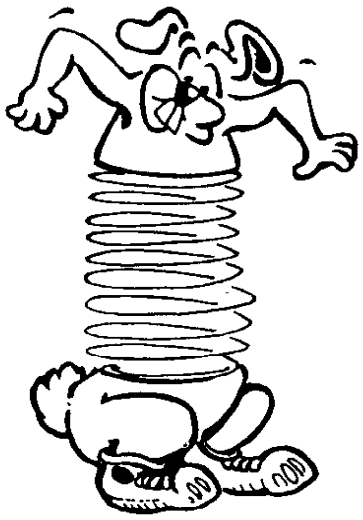
```
TO SPAGHETTI :N
IF :N > 200 [STOP]
CIRCLE :N
RT 45
SPAGHETTI :N + 5
END
```

In addition to spaghetti drawings, what else can you do with this circle procedure?

```
TO CIRCLE :N
REPEAT 36 [FD :N RT 10]
END
```

How About a Slinky?

```
TO SLINKY
LT 90
CIRCLES
END

TO CIRCLES
CIRCLE 5
FD 20
CIRCLES
END
```



You might think that recursion is just like a loop, that it just goes around in circles. Well, not quite! Sometimes things aren't what they seem to be.

Embedded Recursion

To check this out, let's look at a procedure that uses "embedded recursion." The recursive call is embedded in the middle of the procedure somewhere, like in the GET.THROUGH.LIFE procedure.

```
TO TEST.RECURSION :N
PRINT [IS THIS RECURSION?]
IF READWORD = "YES [TEST.RECURSION :N + 1]
PRINT :N
END
```

In the TEST.RECURSION procedure, the variable :N is used as a counter. It is used to keep track of your answers to the question, IS THIS RECURSION?

Watch and see. Type

TEST.RECURSION 1

When you start the procedure, the first thing you see on the screen is the question:

IS THIS RECURSION?

READWORD tells Logo to stop and wait for you to type an answer. Type YES. The counter knows that this is your first answer.

Then we come to a test. If the word you typed was YES, then the procedure calls itself. What happens?

You guessed it! There's that question again.

IS THIS RECURSION?

Is this what recursion does to you?

Type YES a few times when you see the question. Then type NO. What happens this time?

When you type NO, the procedure comes to the test. This time the word you typed doesn't match YES and so the computer reads the next line:

PRINT :N

WOW, what happened then?

Why were so many numbers printed? That's what makes recursion different from just a simple loop.

When you first look at this procedure, it seems as if it is going to go around in a loop. Every time it passes the TEST.RECURSION :N + 1 line, the counter is going to add 1. Then, when you type NO instead of YES, you'd think the procedure would simply print the current value of N.

Well, that isn't the way recursion works. Morf has one of his rabbit trails to show you what happens.

Rabbit Trail 22. Recursive Pages

Let's look at this procedure again. You'll need some blank paper, a pencil, and scissors for this one.

Fold the paper in half. Then fold it in half again. And again. Then one last time. Crease the edges nice and sharp and then open up your piece of paper.

Cut the paper along the folds. You should end up with 16 small pieces of paper. Now number these "pages" from 1 to 16 by writing a small number at the bottom of the page.

Get your pencil ready and then type

TEST.RECURSION 1

First, you see the question, IS THIS RECURSION?, on the screen. So write a big 1 on your first piece of paper and put that piece off by itself.

Type YES. What happens?

The question appears on the screen and :N becomes :N + 1 — or 2. Write 2 on your second piece of paper and put that on the pile with your first piece, the one with the 1 on it.

Type YES again.

What does :N become now?

Write 3 on the next piece of paper and put that piece on the number pile. Do this again three more times, writing the new number for :N each time. Put each piece of paper on the top of your growing pile of papers.

Now, when you type NO, what happens on the screen?

You see a list of numbers counting backward, right? From 7 back to 1.

Why?

Look at the screen. There are seven questions shown there. You typed "yes" six times and "no" once. In total, you typed 7 answers.

You should have two stacks of paper now. You have some blank pages left over in one stack, pages 8 to 16. The other stack has the pages you numbered from 1 to 7. Each page has a big number written on it.

Now put the pages back in order from 1 to 16. But how do you do that?

You put page #7 with the big number 7 on it, on top of page 8. You put page #6 on top of page #7, page #5 on top of page #6, and so on until you have all the pages back in a single stack again.

OK! Picture the memory in your computer like that stack of pages. Each time the procedure is run, another page is written to memory. When the procedure is stopped, Logo prints the pages.

Amazing Mazes

Remember the MAZE procedure, the example of tail-end recursion you read about earlier in this chapter?

```
TO MAZE :N
IF :N > 300 [STOP]
FD :N RT 90
MAZE :N +10
END
```

Now look at this procedure.

```
TO AMAZE :N
IF :N > 120 [STOP]
AMAZE :N + 10
FD :N RT 90
END
```

Here's another example of "embedded" recursion.

Will this procedure produce the same picture as the MAZE procedure or will it be different?

Try to picture what it will look like before you run it. Think about how recursion works, about how it reads and acts on procedures. Then start with :N as 50.

This is how Logo reads the procedure the first time.

TO AMAZE 50	
IF 50 > 300 [STOP]	Since 50 is smaller than 300, Logo goes to the next line.
AMAZE 50 + 10	AMAZE 50 becomes AMAZE 60 and AMAZE starts again.
FD 50 RT 90	**This line is held in memory.**
END	

Next, you have:

TO AMAZE 60	
IF 60 > 300 [STOP]	Since 60 is smaller than 300, Logo goes to the next line.
AMAZE 60 + 10	AMAZE 60 becomes AMAZE 70 and is called.
FD 60 RT 90	**This line is held in memory.**
END	

Next, you have:

TO AMAZE 70	
IF 70 > 300 [STOP]	Since 70 is smaller than 300, Logo goes to the next line.
AMAZE 80 + 10	AMAZE 70 becomes AMAZE 80 and is called.
FD 70 RT 90	**This line is held in memory.**
END	

and then

TO AMAZE 80	
IF 80 > 300 [STOP]	Since 80 is smaller than 300, Logo goes to the next line.
AMAZE 80 + 10	AMAZE 80 becomes AMAZE 90 and is called.
FD 80 RT 90	**This line is held in memory.**
END	

Each time Logo runs the procedure, it doesn't get to the last line. That's because the procedure calls itself. So it writes each last line on a "page" of the memory stack. It will keep going, writing pages for each line it did not run — AMAZE 90, 100, 110, and finally 300. Then it stops.

As Logo reads the pages, it puts them back in order sending the turtle

FD 300, RT 90
FD 290, RT 90
FD 280, RT 90
…back to where she started…
FD 50, RT 90.

So-o-o, are the pictures produced by MAZE and AMAZE the same?

The pictures look the same. The difference is that MAZE starts small and gets larger. AMAZE starts big and gets smaller.

There's another example of embedded recursion on the next page. This procedure produces a crazy drawing.

Why? Can you tell without running it?

```
TO TOWER :SIZE
IF :SIZE < 0 [STOP]
SQUARE :SIZE
TOWER :SIZE - 10
SQUARE :SIZE
FD :SIZE
END

TO SQUARE :SIZE
REPEAT 4 [FD :SIZE RT 90]
END
```

No, this is not like the TOWER procedure from Chapter 5 even though it looks something like it. To see what it looks like, type TOWER and a number for the length of one side of the SQUARE.

How would you change this procedure to make a better looking drawing? As a reminder, here's the TOWER procedure you saw earlier.

```
TO SQUARES :S
IF :S < 0 [STOP]
REPEAT 4 [FD :S RT 90]
FD :S
SQUARES :S - 5
END

TO TOWER :S :T
IF :T = 0 [STOP]
SQUARES :S
TOWER :S :T - 1
END
```

Spirals, Squirals, Polyspis, and Fractals

Do you know what they call the drawings that MAZE and AMAZE produce?

They're spirals. No, you don't need a telescope to spy on anything.

Spirals, squrials, polyspis, and fractals draw some of the prettiest drawings you can make using Logo.

MAZE and AMAZE produce square spirals, or squirals.

```
FD 50 RT 90
FD 60 RT 90
FD 70 RT 90
FD 80 RT 90
FD 90 RT 90
```

…and on and on and on.

But what about other types of spirals?

Remember that procedure you wrote to draw any kind of shape?

```
TO POLYGON :SIDE :REPEATS
REPEAT :REPEATS [FD :SIDE RT 360 / :REPEATS]
END
```

OK! Here's a challenge for you. Change this procedure into a recursive procedure that will draw the same kind of picture. How about this?

```
TO POLYGON :SIDE :REPEATS
FD :SIDE
RT 360 / :REPEATS
POLYGON :SIDE :REPEATS
END
```

You can make this easier by changing the :REPEATS variable to an :ANGLE variable.

```
TO POLYGON :SIDE :ANGLE
FD :SIDE
RT :ANGLE
POLYGON :SIDE :ANGLE
END
```

OK! If you set :SIDE to 100 and :ANGLE to 120, you will send the turtle on a continuous trip around a triangle.

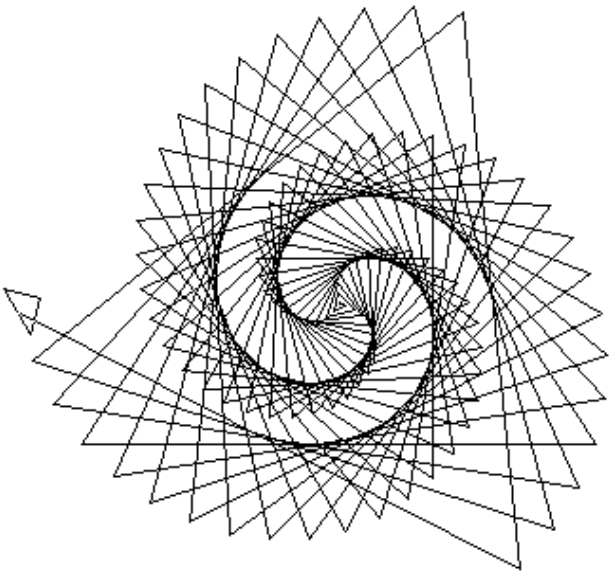
But that's no fun! So here's one way to handle it.

```
TO POLYGON :SIDE :ANGLE :AMT
IF :SIDE > 200 [STOP]
FD :SIDE  RT :ANGLE
POLYGON (:SIDE + :AMT) :ANGLE :AMT
END
```

What do you think the AMT variable does?

Well, here's a drawing produced by this procedure.

Does it help?



Play around with different numbers for the three variables of the POLYGON procedure. You'll be surprised at the things you can do.

What happens when you change 120 to 123?

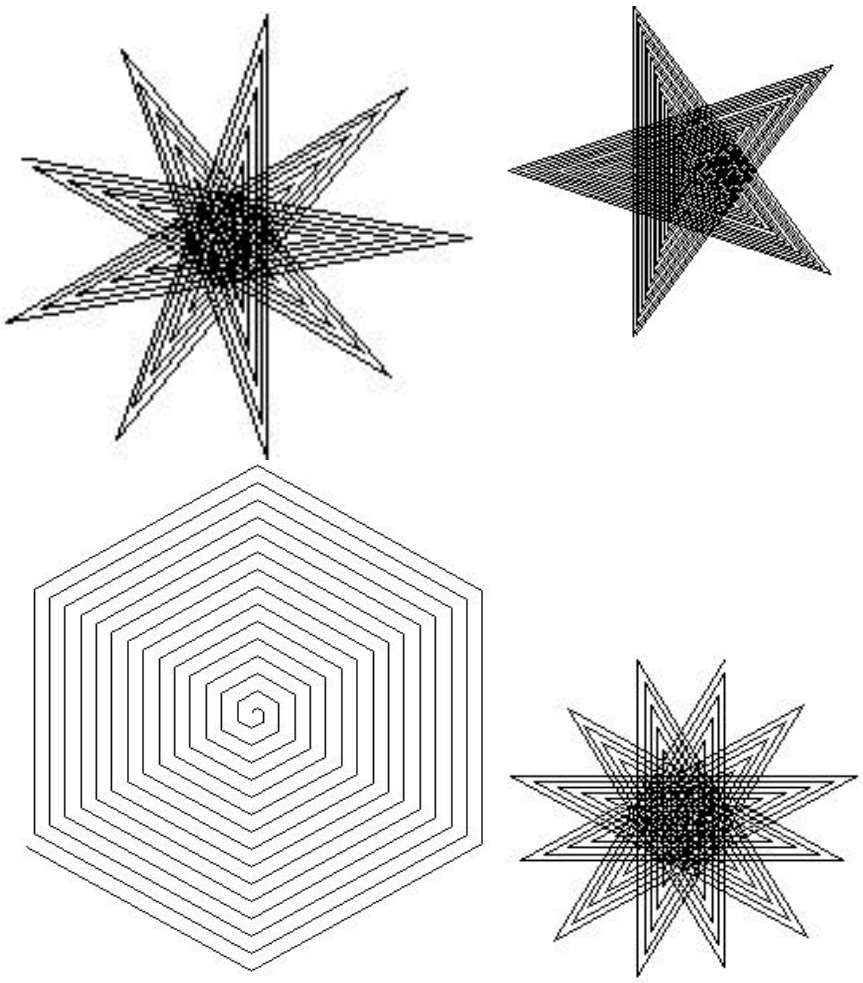
POLYGON 1 123 3

POLYGON 1 90 5

POLYGON 5 144 5

POLYGON 1 172 3

There are some more ideas on the next page.



Multiple Spirals

How would you put more than one spiral on the screen at the same time?

Here is a procedure a young student developed. The goal was to create two spirals within the same procedure.

What do you think of it — without running it, that is?

Will the procedure at the top of the next page draw two spirals?

```
TO SPIRAL :N
IF :N > 100 [STOP]
FD :N RT 90
SPIRAL :N + 5
FD 200
IF :N > 100 [STOP]
FD :N RT 90
SPIRAL :N + 5
END
```

It seemed perfectly logical to this student that the turtle would draw the first spiral, move 200, and then draw the second one. What that student overlooked is that the recursive call sends the turtle back to the beginning. The result of this procedure is a mess.

But how would you straighten it out?

One of the things this student overlooked was a very valuable lesson about working with Logo. You need to think in "chunks."

Logo has to process one "chunk" of information at a time. In the procedures below, SPIRAL is one chunk of information. When you want to process more than one chunk of information, you need to add a procedure that will process your chunks in the sequence that you want. This is what SPIRALS does for you.

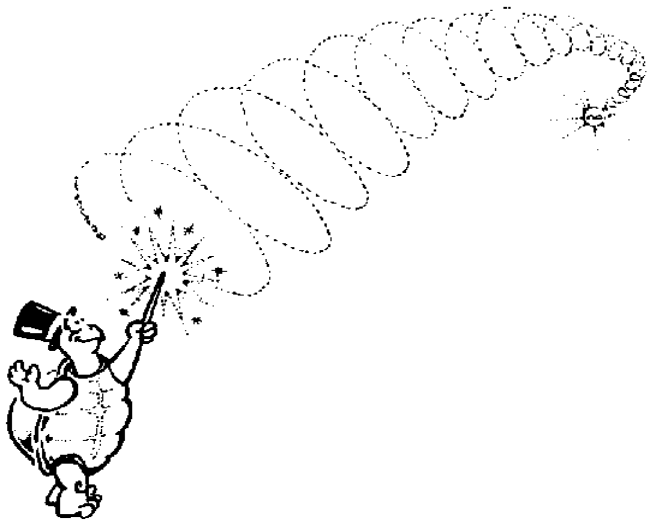
```
TO SPIRALS :N
SPIRAL :N
PU FD 200 PD
SPIRAL :N
END
```

```
TO SPIRAL :N
IF :N > 100 [STOP]
FD :N RT 90
SPIRAL :N + 5
END
```

The SPIRALS procedure draws two spirals. What would you have to do to make it draw four? Six? A variable number?

Polyspis and Inspis

POLYSPI and INSPI are variations on the POLYGON and SPIRAL procedures. You start with the basic POLYGON procedure.



```
TO POLYGON :SIDE :ANGLE
FD :SIDE RT :ANGLE
POLYGON :SIDE :ANGLE
END
```

Let's change this a bit.

```
TO POLYSPI :SIDE :ANGLE
FD :SIDE RT :ANGLE
POLYGON :SIDE + 3 :ANGLE
END
```

What does POLYSPI do to the POLYGON procedure?

It adds another variable so that you change how much the :SIDE changes.

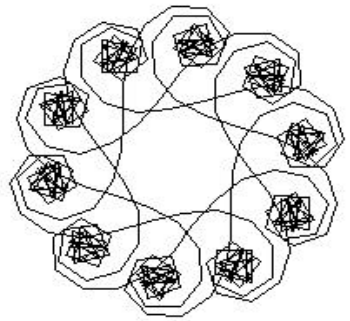
```
TO POLYSPI :SIDE :ANGLE :INC
FD :SIDE RT :ANGLE
POLYGON :SIDE + :INC :ANGLE :INC
END
```

Try this procedure with various inputs. Then let's change the procedure again. This time, we won't change the :SIDE. We'll change the :ANGLE.

```
TO INSPI :SIDE :ANGLE :INC.
FD :SIDE RT :ANGLE
INSPI :SIDE :ANGLE + :INC :INC
END
```

Now try these. Can you predict what they'll look like?

```
INSPI 10 0 15
INSPI 7 45 17
INSPI 10 4 20
INSPI 10 0 8
INSPI 3 45 30
```



Now try your own ideas. But before we move on, here's one more variation to explore.

Here's the POLY1 procedure.

```
TO POLY1 :SIDE :ANGLE
FD :SIDE RT 90 FD :SIDE RT :ANGLE * 2
POLY1 :SIDE :ANGLE
END
```

How's that different from this one?

```
TO POLY2 :SIDE :ANGLE
FD :SIDE RT 90 FD :SIDE RT :ANGLE
POLY2 :SIDE :ANGLE * 2
END
```

Rabbit Trail 23. String and Wire Art

Have you ever seen string or wire art?

These are beautiful patterns created by wrapping colored string or wire around pins or small nails hammered into a felt-covered board. You can find some very colorful string or wire art kits at a local hobby store.

What's even more fun is to transfer the art patterns to the screen. There you can begin to see the relationships that work together to create the pattern.

First, let's start with a shoe box. Paint the inside of the top using flat black paint. This creates a dull background to show off your string patterns.

The next job is to create an even pattern that you will use to punch tiny holes evenly around the edge of the box top. You can do this very easily on the computer. Here's a recursive procedure that should be pretty easy for you by now.

```
TO PATTRN :DIST :MARKS
IF :MARKS = 0 [STOP]
FD :DIST MARK :DIST
MAKE "MARKS :MARKS - 1
PATTRN :DIST :MARKS
END

TO MARK :DIST
RT 90 FD :DIST / 10
BK :DIST / 5 FD :DIST / 10 LT 90
END
```

This procedure divides the task of drawing a pattern into easily understood chunks. The big chunk is drawing the pattern. The little chunk draws the actual marks.

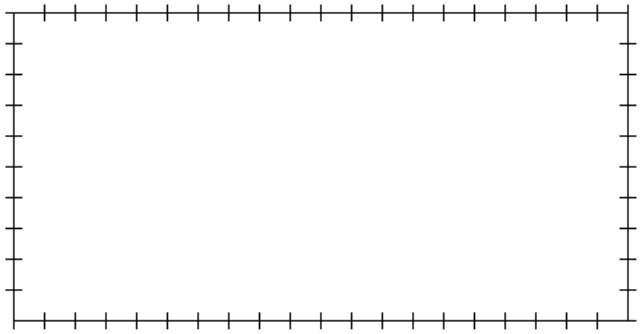
The variables let you set the number of marks (:MARKS) and the distance (:DIST) between them. For example, if you want to print 20 horizontal marks that are 25 turtle steps apart, type PATTERN 25 20 and press Enter.

Print the patterns and cut them into narrow strips. Then paste or tape them to the edge of your painted box top.

Curves From Straight Lines

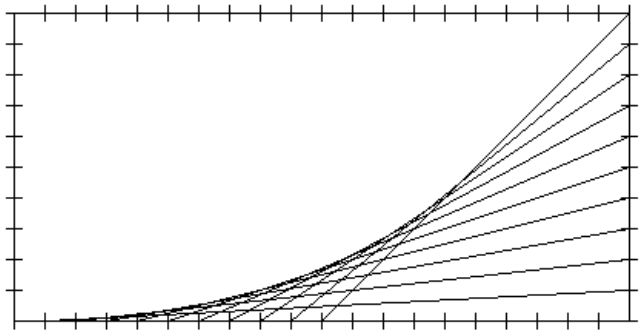
Now there are lots of things you can do. For one thing, you can use colored yarn and a needle to make curves from straight lines.

Here's a box top pattern.



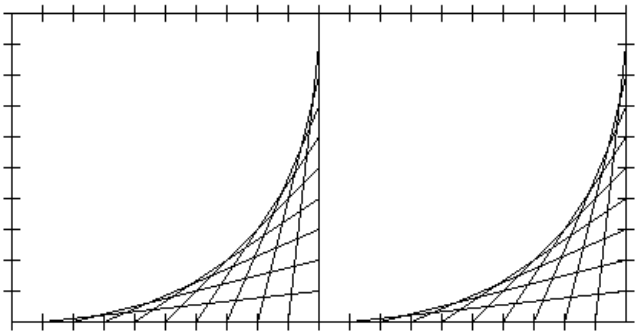
1. Start at the lower left hand corner.
2. Push the needle through the corner mark into the box top and then out through the mark at the lower right corner.
3. Move up to the first mark up the right side and push the needle from the outside into the box top.
4. Go to the first mark in from the left corner and push the needle from the inside to the outside of the box top.

Soon you will have a pattern that looks like this, a curve made from straight lines.

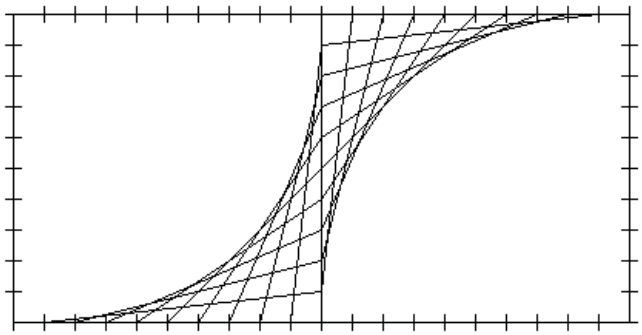


There are lots of other patterns you can make.

Why not try these?



Turn one of those upside down and look what you'll get.



There are all sorts of patterns you can make. If you want to dress them up a bit, try different colors of yarn for different parts of the design.

When you've used up all your old shoe boxes, you can try other designs on the Logo screen.

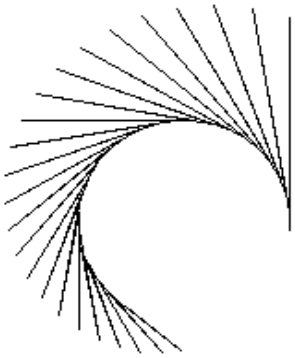
But, wait a minute! How are you going to do that?

Rabbit Trail 24. Curves From Straight Lines

Well, let's start with a pencil, a piece of paper, and a straightedge. A ruler makes a good tool for this project.

1. Put the ruler on the paper in a vertical position, so that it's going straight up and down.
2. Draw a line from the bottom of the ruler up to about six inches and back to one-half inch from the bottom.
3. Hold your pencil in place and turn the ruler about 10 degrees.
4. Repeat steps 2 and 3 several times.

Does your drawing look something like this?



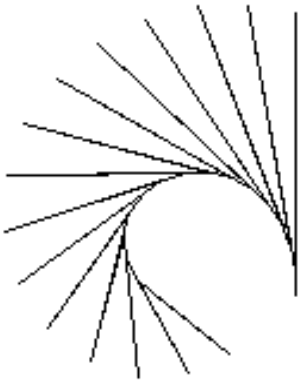
Not bad! Here's how you can do that on the computer.

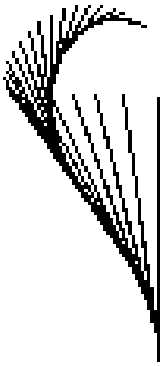
```
TO FANLEFT :DIST :ANGLE
IF :DIST < 0 [STOP]
FD :DIST BK :DIST - 10 LT :ANGLE
FANLEFT :DIST - 5 :ANGLE
END
```

What do you think would happen if you changed the angle *and* the distance each time a line was drawn?

```
TO LETSFINDOUT :DIST :ANGLE
IF :DIST < 0 [STOP]
FD :DIST BK :DIST - 10 LT :ANGLE
LETSFINDOUT :DIST - 5 :ANGLE + 2
END
```

If you can't see the difference here, try changing the number added to the ANGLE.





Here's a challenge for you.

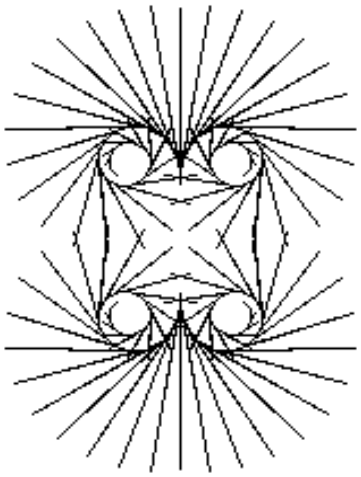
How would you create this drawing?

Here's a hint. Take a look at the angles between the lines.

Here are a few more ideas to play with. How about a FANRIGHT procedure? What would that one do? How would it be different?

What would happen if you combined them?

```
TO SWIRL :DIST :ANGLE
START1
FANLEFT :DIST :ANGLE
START1
FANRIGHT :DIST :ANGLE
START2
FANLEFT :DIST :ANGLE
START2
FANRIGHT :DIST :ANGLE
END
```

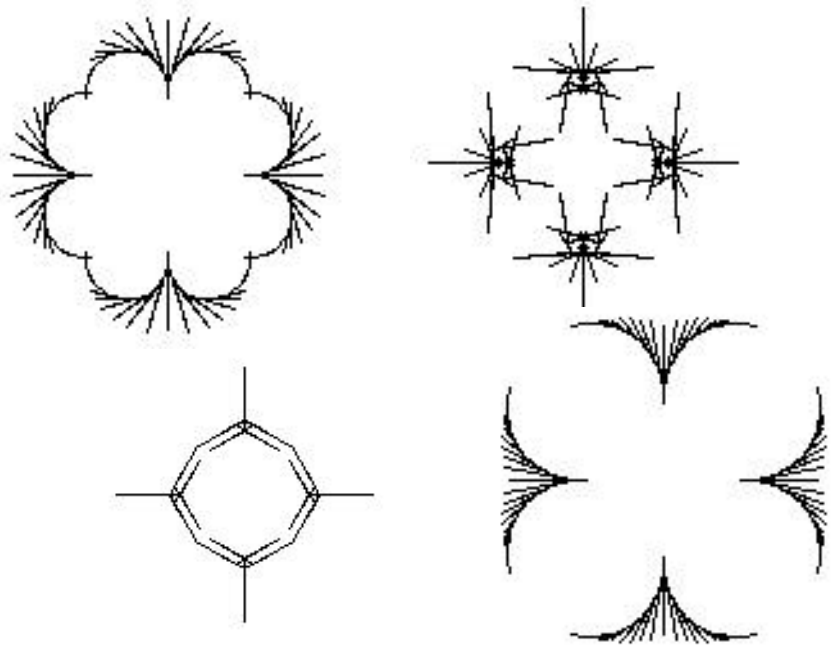


Note that the FANLEFT and FANRIGHT procedures are changed slightly to produce this drawing. Check the recursive statement in each procedure.

```
TO FANLEFT :DIST :ANGLE
IF :DIST < 0 [STOP]
FD :DIST BK :DIST - 5 LT :ANGLE
FANLEFT :DIST - 3 :ANGLE + 1
END

TO FANRIGHT :DIST :ANGLE
IF :DIST < 0 [STOP]
FD :DIST BK :DIST - 5 RT :ANGLE
FANRIGHT :DIST - 3 :ANGLE + 1
END

TO START1
PU HOME PD
END
```

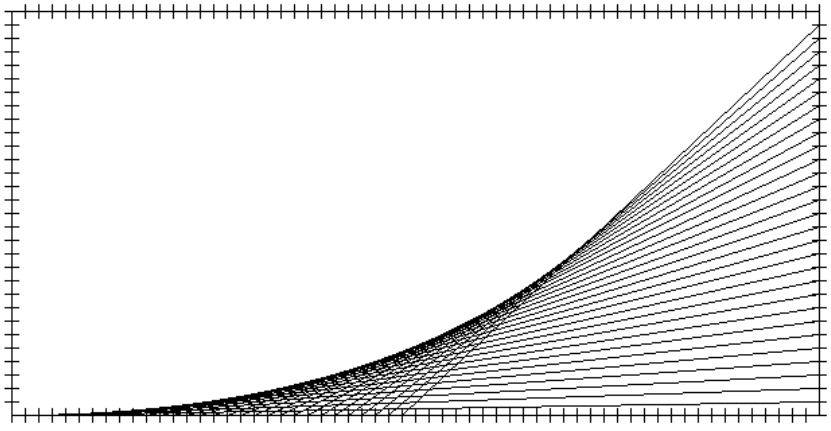
```
TO START2
PU HOME RT 180 FD 50 PD
END
```

Another thing you might want to try is to add a START3 and START4 so that you can have figures drawn at 90 degrees and 270 degrees. Here's a few simple ones.



String and Wire Art Procedures

Remember when you did some string and wire art earlier? You never did get around to the Logo procedures, did you? Guess what? You will now!



BOXTOP draws the box top. You define the size of the short side and the number of marks to appear on that side. For example:

```
BOXTOP 300 30
```

```
TO BOXTOP :DIST :MARK
PU SETX :DIST - :DIST * 2 PD
REPEAT 2 [MARKER :MARK RT 90 REPEAT 2
  [MARKER :MARK] RT 90]
END
```

The CURVE procedure looks complex. But it is simply the turtle doing the sewing that you did with a needle and colored yarn.

```
CURVE 30 10 -300 0 300 0
```

You draw 30 lines that are 10 steps apart (there's a GAP of 10 steps). You start at :X1 in the lower left where the X-coordinate is -300 and the :Y1 is 0. The turtle moves from :X1 and :Y1 to :X2 and :Y2, then back and forth 30 times.

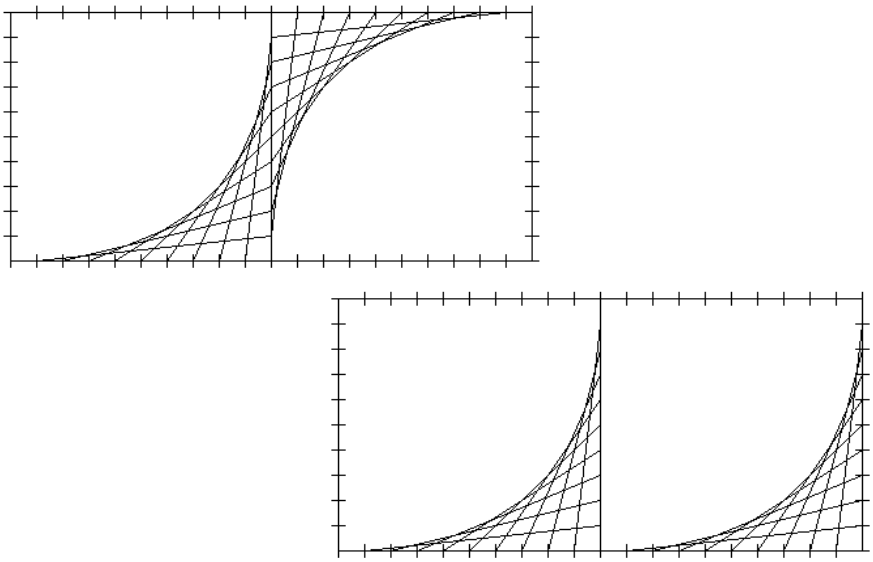
```
TO CURVE :T :GAP :X1 :Y1 :X2 :Y2
IF :T = 0 [STOP]
PU SETXY LIST :X1 :Y1 PD
SETXY LIST :X2 :Y2
MAKE "X1 :X1 + :GAP
MAKE "Y2 :Y2 + :GAP
CURVE :T - 1 :GAP :X1 :Y1 :X2 :Y2
END
```

```
TO MARKER :MARK
REPEAT :MARK [FD :DIST / :MARK MARKS]
END
```

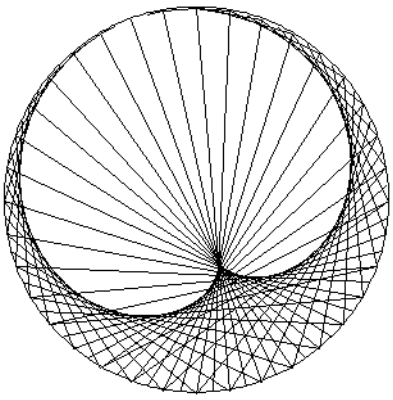
```
TO MARKS
LT 90 FD 5 BK 10 FD 5 RT 90
END
```

Go ahead. Play with a few other combinations. Do the same things on the screen that you did with yarn. What else can you dream up?

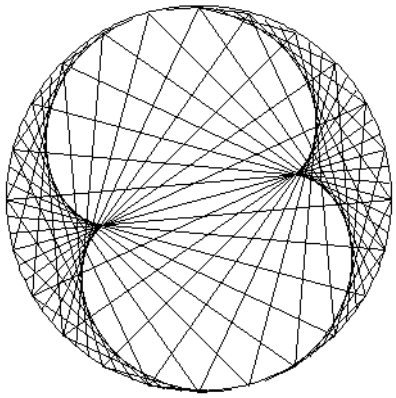
What would you have to do to draw the box top patterns shown earlier in this chapter?



Once you've played with the BOXTOP, the STRING procedures become a bit easier to understand.



STRING 150 2 2



STRING 150 3 2

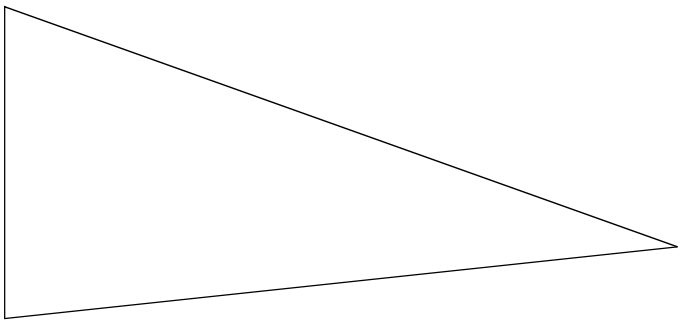
```
TO STRING :RADIUS :DIST :HEAD
CS HT MAKE "N 1 PU SETX :RADIUS PD
REPEAT 360 [FD :RADIUS * PI/180 LT 1] PU HOME
REPEAT 36 * :HEAD [FD :RADIUS MAKE "P POS
  HOME HDG FD :RADIUS PD SETXY LIST :P PU
  HOME HDG1 MAKE "N :N +1]
END

TO HDG
SETH REMAINDER (:N * 5 * :DIST) 360
END

TO HDG1
SETH REMAINDER (5 * (:N-1)) 360
END
```

Play with these procedures for awhile, trying different variables. Not only do they create some beautiful patterns, they give you a look at how positions and headings can be used.

But before we leave recursion, you can't overlook the fun you can have with fractals.

A Triangle in a Circle

Draw a triangle on the computer — any type of triangle will do.



Now draw a circle around that triangle so that the edge of the circle touches the three points of the triangle.

This problem shows a great use of recursion. The CHECK.DIST procedure keeps calling itself until it finds the center of the circle. It then draws the circle touching each corner of the triangle. Without recursion, this would be a difficult mathematical exercise.

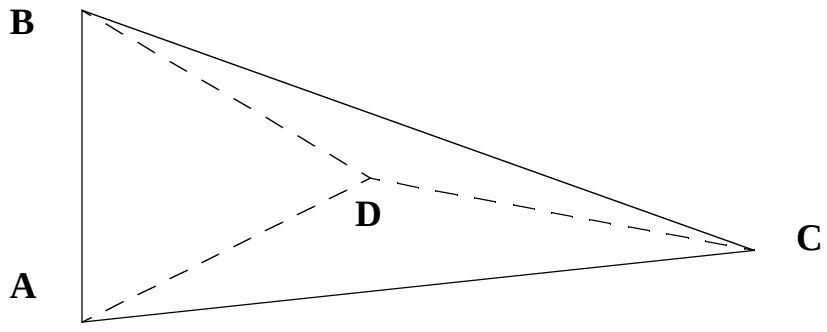
Now let's get to it. Just remember, the whole idea behind Logo is to break a problem down into its simplest parts. Start with what you know. Determine what you don't know. Then go find it.

What do you know?

You know that the three points of the triangle are going to be on the edge, or the circumference of the circle. If you can find a point that is the same distance from each of those points, then you have the center of the circle, right?

To make things easier to understand, let's label the points on the circle. Call them A, B, and C.

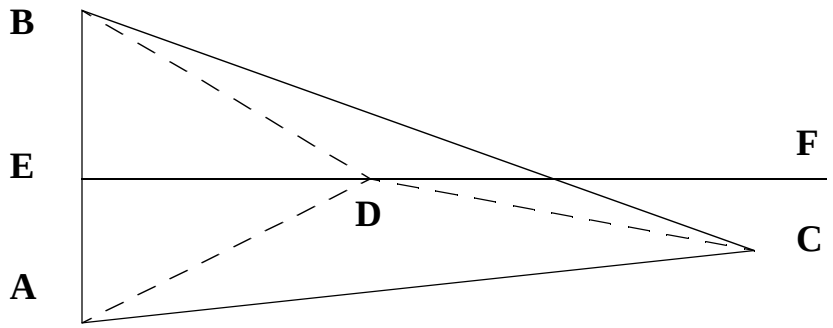
You have to find point D, a point inside the triangle that is the same distance from A as it is from B and C.



If point D is the same distance from A, B, and C, then point D must be the center of the circle and the three lines, AD, BD, and CD are each a radius of the circle you are supposed to draw.

Now, how can you prove that?

Draw the line EF so that it is perpendicular to the middle of line AB.



Perpendicular means that the line EF is at right angles to line AB.

What can you learn from this drawing now?

You have two triangles — ADE and BDE — that share one side and have two short sides that are equal. Therefore, the sides AD and BD must be equal.

OK, if you can find the point on line EF that makes these two lines equal to line CD, you have found the middle of the circle you want to draw.

Let’s do it.

The Random Triangle

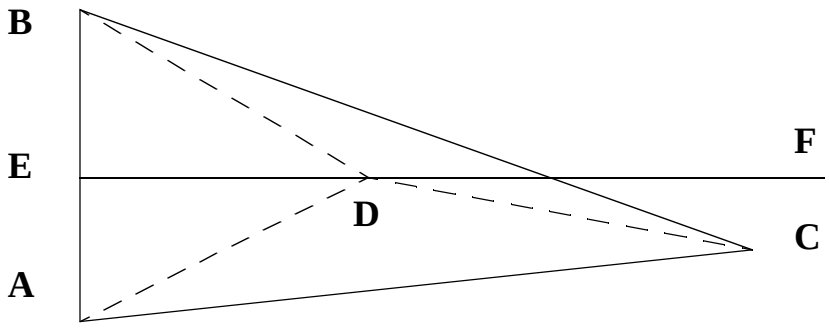
The first step is to create a random triangle, something like you have already drawn.

```
TO RANDOM.TRI
MAKE "POINTA POS
FD 100 RT 120 - RANDOM 30
MAKE POINTB POS
MAKE "DIST 250 - RANDOM 100 FD :DIST
MAKE "POINTC POS  HOME
END
```

This procedure starts from HOME, POINT A with coordinates 0,0. The turtle goes FD 100 and turns right a random angle, somewhere between 120 and 90. This is POINTB, coordinates 100,0.

The turtle then goes FD between 150 and 250 and sets POINTC. Then the turtle goes HOME. The next step is to draw the perpendicular line.

```
TO RT.ANGLE
SETPOS LIST :POINTA
FD 100 / 2 RT 90
MAKE "POINTE POS
FD 200 PU HOME PD
END
```



Now you have a drawing something like the one at the bottom of the last page, but without the dotted lines.

What do you need to know now to complete our circle? You need to find the point D on line EF that is the same distance from B as it is from C. You already know that AD and BD are going to be equal and that each is going to be a radius of our circle.

So if you can make one equal to line DC, the other is automatically equal to DC. The first thing you need for that is a distance procedure.

```
TO DIST :X1 :Y1 :X2 :Y2
OP DIST1 :X1 - :X2 :Y1 - :Y2
END

TO DIST1 :DX :DY
OP INT SQRT (:DX * :DX) + (:DY * :DY)
END
```

The DIST procedure measures the distance between two sets of coordinates. Logo measures that difference very precisely. So to keep things simple and easy to compare, the output is an integer, a whole number. (It's a lot easier to compare whole numbers than it is to compare long decimals.)

Now let's put the DISTance procedure to work. You'll use it to calculate two distances: the distance between B and D and the distance between C and D. When these are the same, you'll draw our circle.

```
TO CHECK.DIST
MAKE "BD DIST FIRST :POINTB LAST :POINTB ~
          FIRST :POINTD LAST :POINTD
```

```
MAKE "CD DIST INT FIRST :POINTC ~
        INT LAST :POINTC FIRST :POINTD ~
        LAST :POINTD
TEST :BD = :CD
IFTRUE [HT CIRCLE :POINTD :BD]
IFFALSE [FD 1 MAKE "POINTD POS CHECK.DIST]
END
```

Here are more new commands: FIRST and LAST. Well, actually you've seen them before when you made the POS procedure. This use of FIRST and LAST sort of explains itself.

```
TO POS :LIST
OP LIST FIRST :LIST LAST :LIST
END
```

That's a different way of writing a POS procedure but it gets the job done.

But back to CHECK.DIST. You already know that :POINTB is a list of two coordinates. So FIRST :POINTB must be the first coordinate. And if that's true, then LAST :POINTB must be the last element in the list or the y-coordinate. You'll learn more about characters, numbers, words, lists, FIRST, LAST, and other good stuff later on.

There's one more thing in CHECK.DIST. We used the TEST command:

```
TEST :BD = :CD
IFTRUE [HT CIRCLE :POINTD :BD]
IFFALSE [FD 1 MAKE "POINTD POS CHECK.DIST]
```

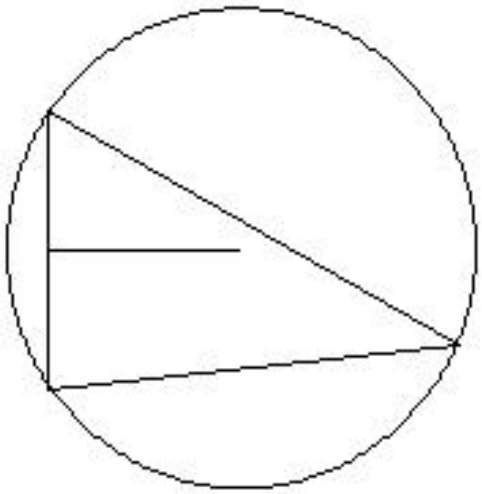
You can also write that as

```
IF :BD = :CD  [HT CIRCLE :POINTD :BD] [FD 1 ~
    MAKE "POINTD POS CHECK.DIST]
```

Now let's run through the CHECK.DIST procedure. The first two lines calculate the distances BD and CD. So that you can see how these distances change, the distances are printed in the Listener or Commander window.

Then Logo tests the two numbers. If :BD = :CD is true, if they are equal, Logo draws a circle with :POINTD as the center and a radius of :BD.

If the two distances are not equal, the turtle moves FD 1 and checks the distances again.



```
TO CIRCLE :CENTER :RADIUS
LOCAL "AMT
MAKE "AMT :RADIUS * PI / 180
PU SETXY LIST :CENTER
SETX XCOR - :RADIUS SETH 0 PD
REPEAT 360 [FD :AMT RT 1]
PU SETPOS :CENTER PD
END
```

To put the whole thing together, here's a place to start.

```
TO START
RANDOM.TRI
SETPOS :POINTA
FD 50 RT 90
MAKE "POINTD POS
CHECK.DIST
END
```

Take your time with this procedure. Come back to it when you're ready. This is a good stepping stone to some of the other procedures you'll see in the rest of this book.

Fun With Fractals

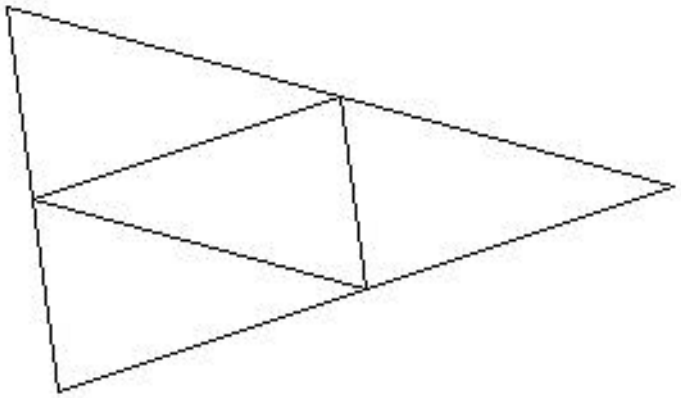
Fractals were once thought to be math monsters. No one could figure out what to do with them. But thanks to computers, we now know that these recursive monsters help make beautiful computer graphics.

Take a look at the COAST.LGO procedure in the Projects directory of the Sourcebook diskette. That shows you how to draw a random coastline. If you'd like to experiment a bit on your own, take a look at this MEDTRI.LGO procedure.

```
TO MEDIAL.TRIANGLE :X1 :Y1 :X2 :Y2 :X3 :Y3
COOR.TRIANGLE :X1 :Y1 :X2 :Y2 :X3 :Y3
MIDPOINT :X1 :Y1 :X2 :Y2
MIDPOINT :X2 :Y2 :X3 :Y3
MIDPOINT :X3 :Y3 :X1 :Y1
MIDPOINT :X1 :Y1 :X2 :Y2
END

TO COOR.TRIANGLE :X1 :Y1 :X2 :Y2 :X3 :Y3
SEGMENT :X1 :Y1 :X2 :Y2
SEGMENT :X2 :Y2 :X3 :Y3
SEGMENT :X3 :Y3 :X1 :Y1
END

TO MIDPOINT :X1 :Y1 :X2 :Y2
SETXY (:X1 + :X2) / 2 (:Y1 + :Y2) / 2
END
```

```
TO SEGMENT :X1 :Y1 :X2 :Y2
PENUP
SETXY :X1 :Y1
PENDOWN
SETXY :X2 :Y2
END
```

This procedure takes any triangle you define and creates a new triangle drawn from the midpoints of each side.



Here's a challenge for you — something you may want to come back to once you've read more about fractals.

1. Start with a triangle that is about as big as your screen.
2. Can you write a procedure that will draw a triangle at the midpoints of each new triangle you create?
3. Later in this book, you'll get a look at working with three dimensional space where you add a Z axis to X and Y. Think about changing the Z axis of each new triangle you create.

Imagine such a procedure run on a graphics workstation with lots of memory. As the triangles get smaller and smaller, and they begin to tilt in different directions, the picture begins to look like a mountain range. Add some color to make it more realistic. What you get is fractals in action.

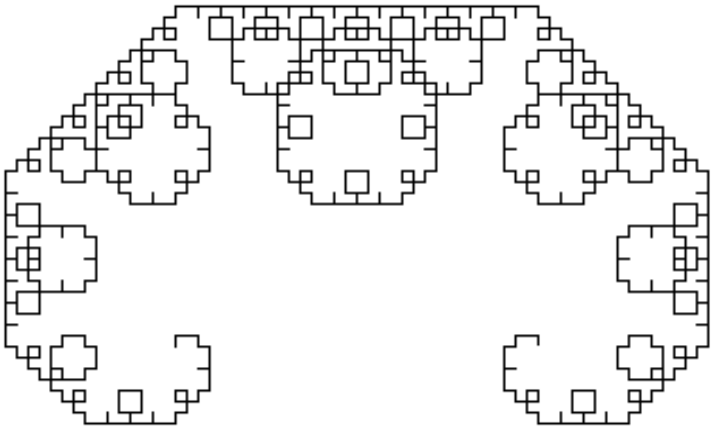
C Curves

There are lots of books on fractals that you can read. So rather than try to explain fractals, let's look at how they work. Here's the well-known C curve as a Logo procedure.

```
TO C :SIZE :LEVEL
IF :LEVEL = 0 [FD :SIZE STOP]
C :SIZE :LEVEL - 1 RT 90
C :SIZE :LEVEL - 1 LT 90
END
```

If you look at the procedure, you see that :SIZE is the variable used by FD. :LEVEL is a bit confusing, so let's watch it work first. Type

```
C 5 10
```



Wow! That's some pattern. Clear the screen and try

```
C 20 3
```

Automatic Startup

OK! Now add the SEE procedure on the next page. When you run SEE, you "see" how the turtle builds such complicated pictures.

```
TO SEE
IF :LEVEL = 11 [STOP]
C :SIZE :LEVEL WAIT 50 CS
SEE :SIZE :LEVEL + 1
END

MAKE "SIZE 10
MAKE "LEVEL 0
```

Hey! Wait a minute.

There's no procedure there at the end for those MAKE statements.

How can that be?

Don't you remember? We talked about how you can have procedures startup and do things when they're loaded into your workspace. In this case, you're telling tell Logo what you want the variables to be without writing a procedure. It saves you the trouble of putting the variables in the procedure title.

Time out for a moment. Here's a question for you. Are those variables local or global? Just checking to keep you on your toes.

While you've timed out, here's a couple of other things you can do.

1. To run SEE when it's first loaded, add the variables above and this line:

```
MAKE "STARTUP [SEE]
```

2. Rather than use the SEE procedure, you can type something like this in the Editor window:

```
CS C 10 1 WAIT 60
CS C 10 2 WAIT 60
CS C 10 3 WAIT 60
CS C 10 4 WAIT 60
CS C 10 8 WAIT 60
CS C 5 10 WAIT 60
```

Now when you load the C procedure, it will run six examples to show you how it works.

Figuring Out Fractals

Now, where were we? Run the SEE procedure. You're watching fractals in action.

To help you figure out fractals, here are some tips:

- Write the C and the SEE procedures on pieces of paper as you did in Morf's Rabbit Trail. This will help you follow the action.

- Change WAIT to 100 or 150 — long enough so that you can see the changes from one level to the next.

- Another thing to do is change the LEVEL variable to 5 or 6, large enough so you can watch how the procedure really works. The higher the level, the more complex the picture.

Dragons, Snowflakes, and other Fractals

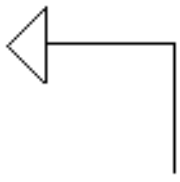
You'll find some other fractal procedures on the Logo diskette — SNOWFLAKE, HILBERT, DRAGON, SRPNSK (that's short for Serpinski) and others.

Take a look at the DRAGON procedure.

```
TO DRAGON :SIZE :LEVEL
LDRAGON :SIZE :LEVEL
END

TO LDRAGON :SIZE :LEVEL
IF :LEVEL = 0 [FD :SIZE STOP]
LDRAGON :SIZE :LEVEL - 1 LT 90
RDRAGON :SIZE :LEVEL - 1
END

TO RDRAGON :SIZE :LEVEL
IF :LEVEL = 0 [FD :SIZE STOP]
LDRAGON :SIZE :LEVEL - 1 RT 90
RDRAGON :SIZE :LEVEL - 1
END
```

Can you see what the DRAGON procedure does? What a drawing would look like?

Here's a picture for DRAGON 50 1.



What would DRAGON 50 0 look like? Try it and see. For a better look at how DRAGON works, turn on TRACE.

Tracing the Dragon

With TRACE turned on, type **DRAGON 50 1** and press **Enter**.

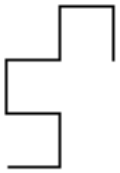
Then check the Commander window to see the sequence of operations that Logo went through.

```
TO DRAGON 50 1
LDRAGON 50 1
END

TO LDRAGON
IF 1 = 0 [FD 50 STOP]
LDRAGON 50 1 - 1 LT 90
RDRAGON 50 1 - 1
END

TO RDRAGON
IF 1 = 0 [FD 50 STOP]
LDRAGON 50 1 - 1 RT 90
RDRAGON 50 1 - 1
END
```

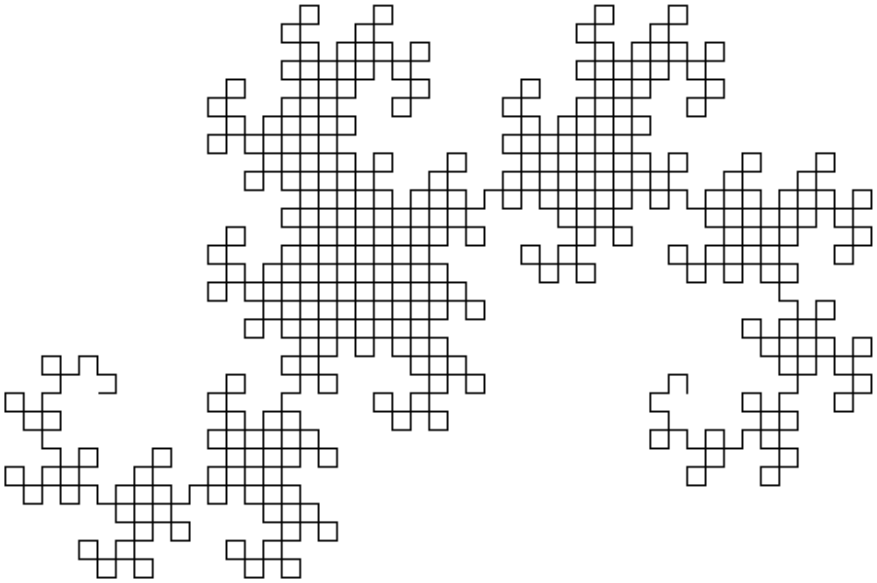
(NOTE: You can also use the STEP command, which steps you through each command of each procedure. The command is STEP [<*procedures to step through*>].)

Now try DRAGON 20 2

DRAGON 20 3



DRAGON 10 10



If you have trouble understanding the list in the Trace window, use a pad of paper and make stacks of recursive calls — the same way you did before.

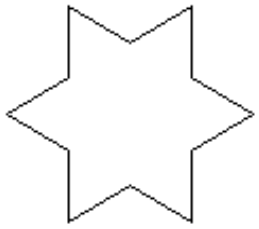
Snowflakes Again

Now take a look at the SNOWFLAKE procedure. Before the snowflakes were made using REPEAT 6 to create a unique six-pointed pattern. These are a bit different.

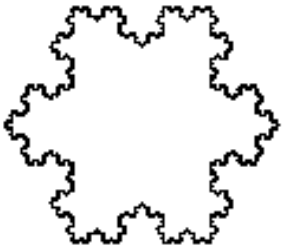
```
TO SNOWFLAKE :SIZE :LEVEL
REPEAT 3 [RT 120 SIDE :SIZE :LEVEL]
END

TO SIDE :SIZE :LEVEL
IF :LEVEL = 0 [FD :SIZE STOP]
SIDE :SIZE / 3 :LEVEL - 1 LT 60
SIDE :SIZE / 3 :LEVEL - 1 RT 120
SIDE :SIZE / 3 :LEVEL - 1 LT 60
SIDE :SIZE / 3 :LEVEL - 1
END
```

This procedure gets a bit more complex. What would SNOWFLAKE 50 0 look like. No fair trying it on the computer!

Here's a picture from SNOWFLAKE 100 1



Here's one from SNOWFLAKE 100 4



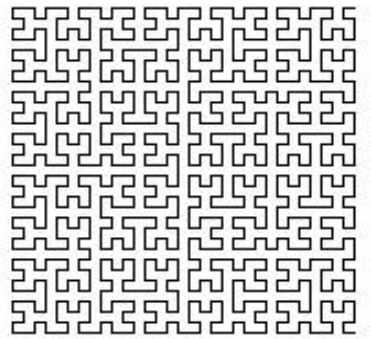
Want to see some colorful snowflakes? Try this procedure. It on the disk that came with this book.

```
TO START
CS PU SETPOS [-100 -100] PD
SETPC [0 0 255]
SNOWFLAKE 300 1 WAIT 30
SETPC [128 128 0]
SNOWFLAKE 300 2 WAIT 30
SETPC [128 0 0]
SNOWFLAKE 300 4
END
```

Hilbert Curve

Now take a look at HILBERT.LGO. It's a bit more complex than SNOWFLAKE or DRAGON — a really good challenge.

```
TO HILBERT  :SIZE :LEVEL
H :SIZE :LEVEL 1
END

TO H  :SIZE :LEV :PAR
IF :LEV = 0 [STOP]
LT :PAR * 90
H  :SIZE  :LEV - 1  0 - :PAR
FD :SIZE
RT :PAR * 90
H :SIZE  :LEV - 1  :PAR
FD :SIZE
H  :SIZE  :LEV - 1  :PAR
RT :PAR * 90
FD :SIZE
H :SIZE :LEV - 1  0 - :PAR
LT :PAR * 90
END
```



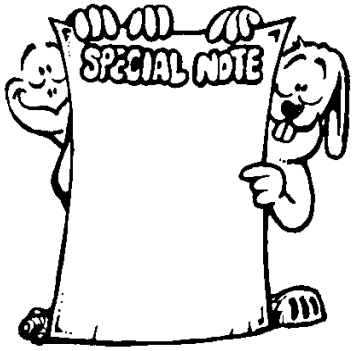
The HIL.LGO and LHILBERT.LGO procedures on the Sourcebook diskette offer another look at the Hilbert curve.

Serpinski Gaskets

For some more complex drawings, take a look at the Serpinski gasket procedures.

```
TO GASKET :SIZE :LEVEL
; SERPINSKI GASKET: MANDELBROT P. 142
; TEST CASE: GASKET 100 2
CS HT PU
BK 3 * :SIZE / 4
PD LT 30
```

```
REPEAT 3 [GGEN :SIZE :LEVEL RT 120]
RT 30 PU
FD :SIZE / 2 PD FILL
END

TO GGEN :SIZE :LEVEL
IF :LEVEL = 0 [FD :SIZE STOP]
IF :SIZE < 2 [FD :SIZE STOP]
(LOCAL "LEVEL1 "SIZE2)
MAKE "LEVEL1 :LEVEL - 1
MAKE "SIZE2 :SIZE / 2
GGEN :SIZE2 :LEVEL1
LT 120 REPEAT 3 [GGEN :SIZE2 :LEVEL1 RT 120]
PU RT 30 FD :SIZE2 / 2 PD
PU BK :SIZE2 / 2 LT 30 PD
RT 120 GGEN :SIZE2 :LEVEL1
END
```



SPECIAL NOTE: Hmmmm? Is that something new? If you ever want to write notes in your procedures, type a semicolon. Logo ignores anything that is on the rest of that line. You can put the semicolon anywhere on the line.

```
FD 100 RT 90 ;that's a corner
```

Here's one where you can create a colorful carpet design. Just add color.

```
TO CARPET :SIZE :LEVEL
CS HT
PU HOME SETH 45 BK :SIZE / 2 PD
SETH 0
REPEAT 4 [CGEN :SIZE :LEVEL RT 90]
```

```
PU RT 45 FD :SIZE / 2 PD
END

TO CGEN :SIZE :LEVEL
IF :LEVEL = 0 [FD :SIZE STOP]
IF :SIZE < 3 [FD :SIZE STOP]
(LOCAL "SIZE3 "LEV1)
MAKE "SIZE3 :SIZE / 3
MAKE "LEV1 :LEVEL - 1
REPEAT 3 [CGEN :SIZE3 :LEV1]
PU BK :SIZE BK :SIZE3 LT 90 FD :SIZE3 LT 90 PD
REPEAT 4 [CGEN :SIZE3 :LEV1 RT 90]
PU RT 45 FD :SIZE3 / 2 PD
PU BK :SIZE3 / 2 LT 45
BK :SIZE
PD REPEAT 4 [CGEN :SIZE3 :LEV1 RT 90]
PU RT 45 FD :SIZE3 / 2 PD
PU BK :SIZE3 / 2 LT 45
PU BK :SIZE3 RT 90 BK :SIZE3 RT 90 PD
END
```

You've got recursive calls embedded all over the place in this one.

Adding Color

Here's a Serpinski curve with color.

```
TO SIERP :S :LEVEL
SETPC [0 0 0]
SETPENSIZE [2 2]
MAKE "DIAG :S / SQRT 2
REPEAT 4 [ONE :S :DIAG :LEVEL  RT 45 ~
  FD :DIAG RT 45]
COLOR
END
```

```
TO ONE :S :DIAG :LEVEL
IF :LEVEL = 0 [STOP]
ONE :S :DIAG :LEVEL - 1
RT 45 FD :DIAG RT 45
ONE :S :DIAG :LEVEL - 1
LT 90 FD :S LT 90
ONE :S :DIAG :LEVEL - 1
RT 45 FD :DIAG RT 45
ONE :S :DIAG :LEVEL - 1
END

TO COLOR
PU RT 90 FD 15 PD
SETFC [255 0 0]
FILL
END
```

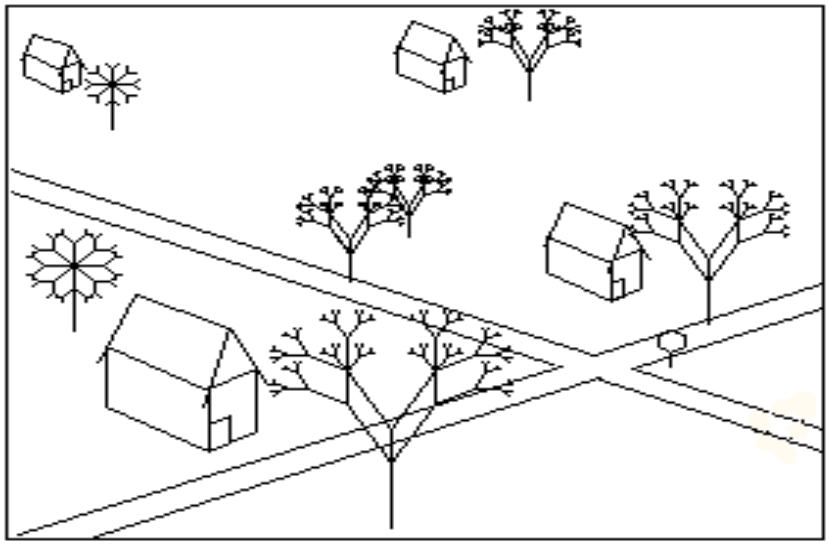
There are many, many books on fractals, from the most basic level to the very complex. Take a look at some of these, especially those that deal with computer art and landscapes.

Logo Trees

RUNNER.LGO, which you'll find on the Sourcebook diskette, is a great example of recursion. It's also a good example of animating the turtle.

Why not see what you can do with this procedure?

1. Add some color.
2. When the Road Runner reaches the Stop sign, it stops, looks both ways, and then plays two tones. If you have a sound card, why not play a wave file?
3. You'll soon read about changing the shape of the turtle. Why not draw a real Road Runner?
4. Have the Road Runner change directions and travel the other road.



Among other interesting things, RUNNER.LGO uses the classic TREE procedure. It is one of the better known examples of recursion.

```
TO TREE  :LENGTH
IF :LENGTH < 2 [STOP] LT 45 FD :LENGTH
TREE :LENGTH / 2 BACK :LENGTH RT 90
   FD :LENGTH
TREE :LENGTH / 2 BACK :LENGTH LT 45
END
```

If you can't seem to follow the action here, use Morf's pieces of paper to see how it works. It's really pretty neat! Or maybe you'll find these tree procedures easier to deal with.

```
TO FTREE :SIZE :COUNTER
IF :COUNTER = 0 [STOP]
LT 30 FD :SIZE * 2
FTREE :SIZE :COUNTER - 1
BK :SIZE * 2  RT 60 FD :SIZE
FTREE :SIZE :COUNTER - 1
```

```
BK :SIZE LT 30
END

TO TREE :SIZE :LIMIT
IF :SIZE < :LIMIT  [STOP]
LT 45 FD :SIZE
TREE :SIZE * 0.61803 :LIMIT
BK :SIZE RT 90 FD :SIZE
TREE :SIZE * 0.61803 :LIMIT
BK :SIZE LT 45
END
```

These are in the TREES.LGO procedure on the Sourcebook diskette. You'll also find some good examples of recursion in the next chapter.


MY JOURNAL
OR HOW I DID IT!