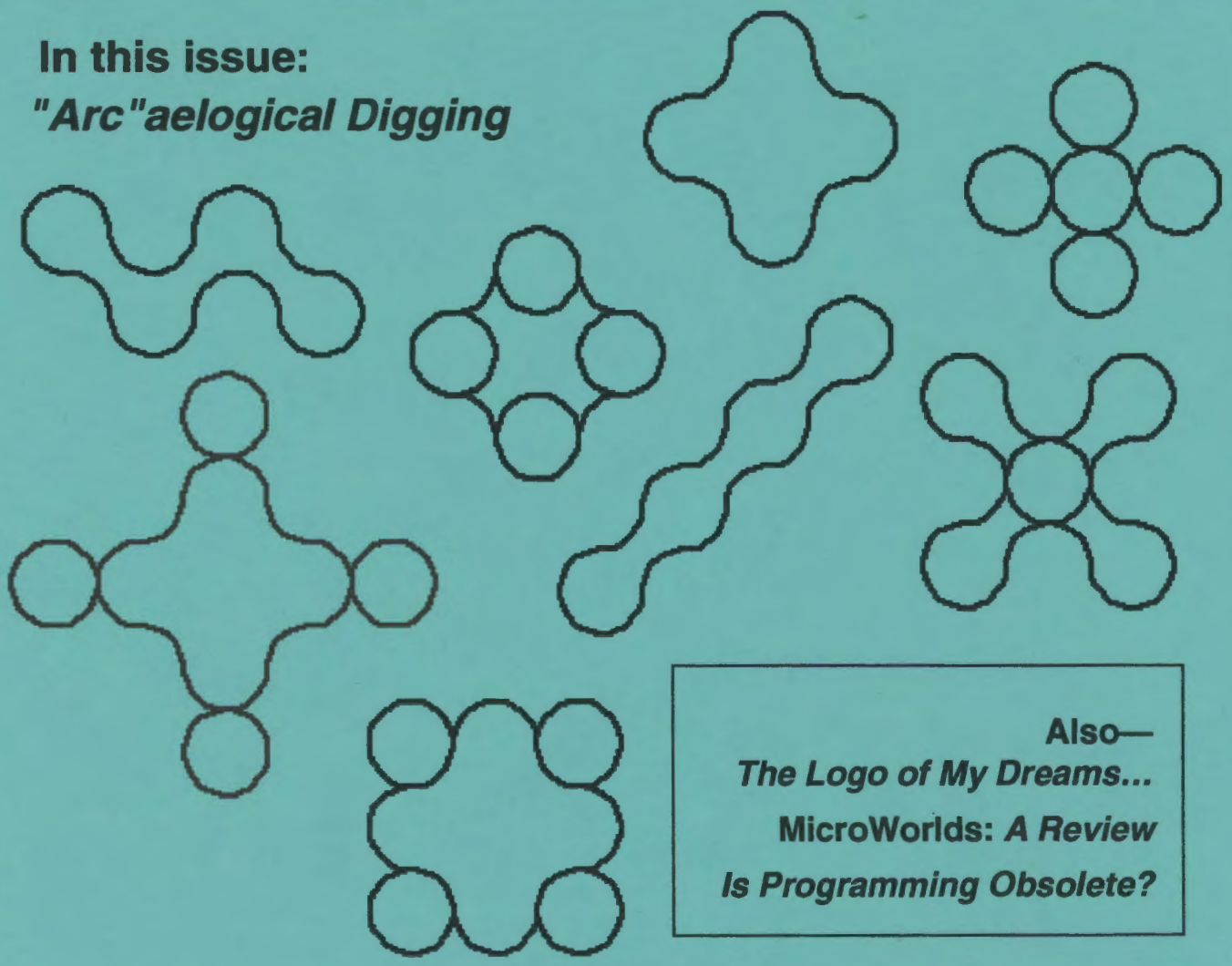
Journal
of the ISTE
Special Interest Group
for Logo-Using
Educators

# LOGO EXCHANGE

Fall 1993 Volume 12 Number 1

**In this issue:**

***"Arc"aelogical Digging***



**Also—**

***The Logo of My Dreams...***

**MicroWorlds: *A Review***

***Is Programming Obsolete?***

International Society for Technology in Education

LX Volume 12 Number 1



## Submission of Manuscripts

*Logo Exchange* is published quarterly by the International Society for Technology in Education Special Interest Group for Logo Using-Educators. *Logo Exchange* solicits articles on all aspects of Logo use in education. Articles appropriate to the International column should be submitted directly to Dennis Harper. Advanced articles should be submitted to Mark Horney, editor of the Extra for Experts column. Articles appropriate for the MathWorlds column should be sent directly to Sandy Dawson.

Manuscripts should be sent by surface mail on a 3.5" disk (where possible). Preferred format is Microsoft *Word* for the Macintosh. ASCII files in either Macintosh or DOS format are also welcome. Submissions may be made by electronic mail as well. Where possible, graphics should also be submitted electronically. Please include electronic copy, either on disk (preferred) or by electronic mail, with any paper submissions. Paper submissions alone will NOT be accepted.

### Send surface mail to:

Sharon Yoder
170 Education, DLIL
University of Oregon
Eugene, OR 97403

### Send electronic mail to:

Internet: YODER@oregon.uoregon.edu

## Deadlines

To be considered for publication, manuscripts must be received by the dates indicated below.

| | |
|---|---|
| Volume 12, Number 4 | Oct. 1, 1993 |
| Volume 13, Number 1 | Feb. 1, 1994 |
| Volume 13, Number 2 | Apr. 1, 1994 |
| Volume 13, Number 3 | July 1, 1994 |



*Logo Exchange* is published quarterly by the International Society for Technology in Education (ISTE), 1787 Agate Street, Eugene, OR 97403-1923, USA; 800/336-5191. This publication was produced using Aldus *PageMaker*®

Individual ISTE Members may join SIG/Logo for $20.00. Dues include a subscription to *Logo Exchange*. Add $10 for mailing outside the USA. Send membership dues to ISTE. Add $2.50 for processing if payment does not accompany your dues. VISA, Mastercard, and Discover accepted.

Advertising space in *Logo Exchange* is limited. Please contact ISTE's advertising coordinator for space availability and details.

*Logo Exchange* solicits articles on all topics of interest to Logo-using educators.Submission guidelines can be obtained by contacting the editor. Opinions expressed in this publication are those of the authors and do not necessarily represent or reflect the official policy of ISTE.



**POSTMASTER:** Send address changes to *Logo Exchange*, ISTE, 1787 Agate St., Eugene, OR 97403-1923. ISTE is a nonprofit organization with its main offices housed at the University of Oregon. **ISSN#** 0888-6790

# LOGO EXCHANGE

**Volume 12 Number 1** **Journal of the ISTE Special Interest Group for Logo-Using Educators** **Fall 1993**

## Contents

From the Editor

# Weary of the Latest and Greatest?

**by Sharon Yoder**

Turn on the television at any time of the day or night and you will be bombarded by advertisements telling you of products of every description that are the latest and the greatest. Magazines, newspapers, and advertisements in stores and on the radio all tell us constantly how products will make us healthier, wealthier, more beautiful, more intelligent, and more desirable. It is any wonder that when we see advertisements for some new computer product, we are immediately skeptical?

Those of us who have been working with Logo for a long time have heard a great deal of "hype" about Logo. We heard that it would teach kids to think and solve problems. It would revolutionize education. We all know those claims didn't materialize—students still need to learn thinking and problem-solving skills, and schools still have problems.

## Wrong Again!

So it's not surprising that nearly a year ago when I began to hear rumors of new Logos coming to the market, I was skeptical. While there are many Logo enthusiasts like you and me, many educators consider Logo passé. I found it hard to believe that several companies would be introducing new Logos into this environment. I paid little attention to the rumors, dismissing them as the chatter of over-zealous users of Logo.

However, late last winter, I was forced out of my jaded complacency. My phone began to ring. Copies of new versions of Logo began to arrive on my desk. Not just one—but several. (Yes, several!) Those rumors of new Logos were no longer rumors—they had to be taken seriously. Here they were: beta copies and final versions. Proof that indeed new Logo versions were continuing to appear. If you need more proof, take a careful look at the announcements and advertisements in this issue and in the Summer 1993 issue of *LX*—you see information on new versions for DOS machines, new versions for Macs.

## Is There a Message Here?

If you have been working with Logo for a long time, you probably remember a day when Logo was *the* software package that a school needed to own. Now Logo is usually far down on the wish-list, below word processors, spreadsheets, graphics tools, desktop-publishing packages, and hypermedia tools. Only a year ago, it seemed that new versions of Logo were quite unlikely.

But Logo is alive and healthy, it seems.

The new versions of Logo are quite different from the ones we used in the early 1980s. There will be purists among you who will be unhappy with some of the new Logos. You will feel that the essence of what is important to you about Logo is gone. What one group of you would describe as "dated," others would describe as their "perfect Logo." I'm sure some of you would be appalled to hear that Logo is to be the scripting language for a new version of *HyperStudio* for the Macintosh. Of course, others will be cheering.

It seems to me that there is a message being sent to us by the great variation in versions of Logo that are commercially viable. There simply is no such thing as a "Logo person" any more. In the early days, we all used versions of Logo that were essentially identical. Since that time, the paths taken by various companies in developing their versions of Logo have gone in different directions. That obviously means that the Logo community is much more diverse than it used to be.

## Forging a New Community

With the appearance of these new versions of Logo, I think that the Logo community needs to take stock of itself. We need to stop pointing fingers at each other and criticizing the Logo-of-choice used by others. Some of you are still delighted with one of the very early Logos. Others thoroughly enjoy *LogoWriter*. Still others are enamored of *Object Logo*. There will be those who adore Harvard Associates Logo that runs under *Windows*. There will be those who think *MicroWorlds* from LCSI is the greatest. There will be those committed to Terrapin's new Macintosh Logo. It doesn't matter. We all think Logo has something special to offer our students and our schools. So let us start this new school year with a renewed acceptance of those who think *their* version of Logo is the latest and the greatest. Let us glory in our mutual love of Logo.

Sharon Yoder
Education 170, DLIL
College of Education
University of Oregon
Eugene, OR 97403
Internet: yoder@oregon.uoregon.edu

# First of All

**by Tom Lough**

Ah, the first of a new school year. A time of excitement, suspense, anxiety, and anticipation. Another opportunity to introduce students to the power of Logo, and to help those familiar with Logo to discover even more capabilities.

Naturally, this idea got me to thinking about other firsts. First impressions. First Amendment. First words. First place. First aid. First gear. How many others can you and your students find?

Sooner or later, I began thinking about the FIRST command. (Incidentally, I also wondered if FIRST was the first Logo command developed. Any Logo historians out there?)

FIRST reports the initial element of a word or list. Here are a few examples.

```
SHOW FIRST "LOGO L

SHOW FIRST [LOGO IS DELIGHTFUL] LOGO
```

The FIRST command can also be chained with other FIRST commands.

```
SHOW FIRST FIRST [LOGO IS
   DELIGHTFUL] L
```

Then I started thinking about initials and acronyms and how FIRST could be used to form them. Here is a tool procedure you might enjoy.

```
TO INITIALS :WORDLIST
IF (COUNT :WORDLIST) > 1 [OUTPUT
   WORD FIRST FIRST :WORDLIST
   INITIALS BUTFIRST :WORDLIST]
OUTPUT FIRST FIRST :WORDLIST
END

SHOW INITIALS [LOGO IS DELIGHTFUL]
LID
```

Such a procedure could be used in many different ways. For example, I have a friend named Cathy Helgoe who loves chocolate. I once remarked that it is no coincidence the word "chocolate" begins with her initials. Later, I followed up with a list of words for the INITIALS procedure.

```
SHOW INITIALS [CATHY HELGOE OFTEN
   CONFIDENTLY ORDERS LUSCIOUS AND
   TASTY EDIBLES]
CHOCOLATE
```

What other opportunities can you find for such a procedure?

P. S. If you would like more details about how the INITIALS procedure functions, send me a self-addressed stamped envelope.

P.P.S. How would you change the INITIALS procedure so that words such as "of" and "for" are ignored?

P. P. P. S. Can you write a procedure called INISENTS that produces the INITIALS of a list of sentences?

```
SHOW INISENTS [ [LOGO IS DELIGHTFUL]
   [ONLY FAST FOOD] ] [LID OFF]
```

**FD 100!**

Tom Lough
Founding Editor
PO Box 394
Simsbury, CT 06070

# Renewal

by Eadie Adamson

There's a song in Bob Fosse's film *All That Jazz* that says, "Everything old is new again." The song has been running through my head as I think about columns for this year. New software adds further dimensions and extends thinking about the old ideas.

The more things change, the more...*interesting* they become. That might be a way to describe the new *MicroWorlds* Logo from Logo Computer Systems (LCSI). While *MicroWorlds* is quite a change from old versions of Logo (see the review of *MicroWorlds* by Sharon Yoder in this issue), it also offers us an opportunity to revisit old ideas and make them better. This column touches on just a few examples by first looking back at the old ideas and then sketching out how new computing environments with Logo change them. Future columns will look at other Logo ideas and how they are transformed in new environments.

## Signs of the Times

Being a careful observer of the world around you is an important skill. Many students need to "wake up" visually. Giving students an assignment to notice some particular aspect of life helps to focus their powers of observation and sharpen their skills of observing and describing things happening around them. That was the purpose of the signs project I developed some years ago, and which now, with new software, can be extended even further.

### First Signs

A group of my students was given an assignment to make flashing signs that were like signs they saw. Before the classwork began, they were asked to notice signs around them: on the street, as they drove in the country, on television, and in movies. Then it became an interesting challenge for students to see just how much they could do to replicate a flashing sign in a *LogoWriter* project.

In the first versions of *LogoWriter* for the Apple and MS-DOS, a flashing sign could be made with the **label** command. Commanding the turtle to do this with

```
label "Hello
```

causes the word Hello to appear on the screen just next to the turtle. If you hide the turtle, it can still **label** but you can see the entire word. Label the same text over the original and it disappears. Using **repeat** to label causes the message to appear to flash. It needs to be slowed down by adding a **wait** command. The previous Hello message would be look like this:

```
repeat 10 [label "Hello wait 5]
```

Odd-numbered inputs for **repeat** will leave the message on the screen (can you see why?), while even-numbered inputs for **repeat** will flash the message on and then turn it off.

One student working on the signs project made a wonderful visual pun on New Yorkers' tendency to ignore the Walk / Don't Walk signs. His version flashed "Don't Walk" alternately with "Run!" Another student observed a large neon sign he passed every weekend on his way out to the country. It alternately flashed text and icons:

```
Newport ♥ New York
```

He combined the **label** command with showing a turtle in the shape of a heart to make his replication of the illuminated billboard. A pair of students who had never been to Times Square made the most complex of the projects, a simulated weather sign meant to represent the text floating around the building in the center of Times Square.

### Other Ways to Make Signs Flash

The Macintosh version of *LogoWriter* added another dimension to labeling. Whatever color was set for the turtle also set the color for the labeled text. This required a slightly different approach to labeling a sign. First it had to be set; then it had to be erased. Commands to make this sign might look like this:

```
setc 4
repeat 4 [pd label "Hello wait 4 pe
   label "Hello wait 4]
```

When color was used, it was necessary to erase the text rather than to simply write over it, as in the earlier *LogoWriter* versions.

### Pictures Flash Too!

Any version of Logo that can load a picture file can have signs flash in another way. Recently we used *Kid Pix* to make some signs to add to some adventure

stories. We saved them as picture files. Then, since we were using *LogoWriter* for the Mac, we chose **Pics** from the **Gadgets** menu and checked the Set Position and Conform Size boxes.

Next we used the **loadpic** command to load the picture onto a page. We set the position and dragged the image out to the size we wanted. Then we saved the page with the graphic sized correctly as a picture file, using the **savepic** command. **Savepic** saves only graphics, not the text, so that a page did not have to be cleared of text in order to save the picture.

On the last page of the adventure story this procedure was added:

```
to startup
wait 250
repeat 4 [loadpic "win wait 5 cg]
loadpic "win
end
```

Before loading the picture we locked the page so that it would always save without the graphics. Now when users choose the page, they read the last few sentences of the story. Suddenly a colorful "You Win" sign flashes on the screen. The advantage of this method over the *LogoWriter* **label** command was that the size and style of text could be changed in many ways. Even "hand-drawn" letters could be used.

Other ideas could be combined with this to make a really flashy ending. Adding sounds is one obvious idea, or writing a little music to play is another. Have two pictures of a flag in two different positions. Load them in alternately.

## New Worlds With *MicroWorlds*

Now there's a truly exciting new way to look at a sign project: *MicroWorlds* from LCSI. The flashing signs that kids made with *Kid Pix* can now be made with the drawing tools that are integral to the *MicroWorlds* program. If they prefer, they can still make something with *Kid Pix* or any other drawing program and import it into their projects.

This all seems somewhat the same. Where's the change? *MicroWorlds* has a text box feature. Instead of allowing text to print on the page itself, a text box of any size is created. The box can be resized to fit the text in the box. Text within the box can have any font or size that is installed on your computer. Parts of the text can be in color, if you prefer. Words can be set apart by color, size and style of text. If you like, text can be "snapped" to the screen so that drawings show through. (Normally a text box obscures that part of the background it covers, unless it is snapped.)

### There's More!

That's not all there is to text boxes. Text boxes can hide or show with two Logo commands: **hidetext** and **showtext**. Thus, you can make a text box and flash it like this (here it's assumed this is your first text box, text1):

```
text1, repeat 4 [wait 2 showtext wait
   2]
```

Note that turtles, text boxes, buttons, and sliders in *MicroWorlds* are addressed by name, followed by a comma, as one would punctuate a sentence in English: "Mary, do this..." In addition, text boxes are numbered in order of creation on a page: text1, text2, and so on. You can also name them so that they reflect what the box is about. Then you address them by that name. For instance, Message might be the name of a text box. Attention is directed to that text box with the command

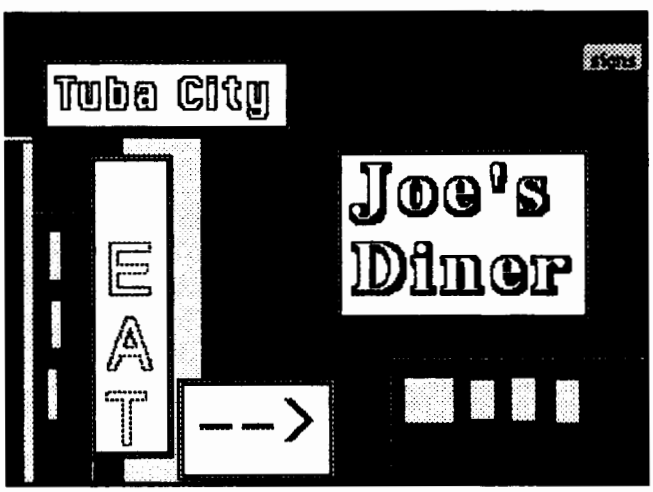
```
Message,
```

followed by instructions.

### More Revelations

Flashing text boxes are dramatic, but think of having a picture with a part of it revealed under the text box. For example, a sign for food might have a picture of food underneath, or even a second sign underneath. When the first sign hides, the background or the second sign is revealed.

In the following example, the Joe's Diner sign, when hidden, reveals a sign that says "Food." It could have had pictures of food hidden below it also. The Food sign was another text box that was put just beneath the Diner box and snapped to the screen. (There's a pull-down menu that lets you do this with just a mouse click.) The commands to hide and show the text box are directed to the Diner sign. When the Diner hides, the Food sign is revealed.



## But That's Not All

Now think about this: Often you'll see neon signs with a flashing outline. *MicroWorlds* has primitives that hide or show the frame of text boxes without disturbing the text: **hideframe** and **showframe**. To flash a frame for your text, you can address the box and use these commands:

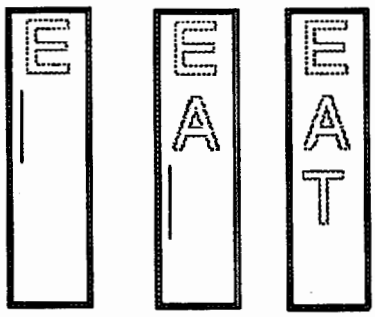
```
text1, repeat 4 [hideframe wait 2
   showframe wait 2]
```

The Tuba City sign in the previous picture used the **hideframe** and **showframe** commands to flash its frame. The commands to hide and show the text might also have been used to create still more motion.

## Moving Letters

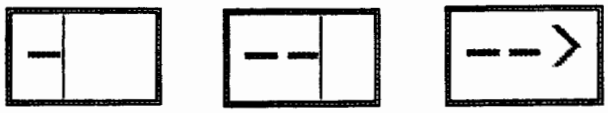
The Eat sign was more complicated. Text is entered a letter at a time, much as a neon sign can flash on one piece at a time. Just as in other versions of Logo in which you could print to the screen, in *MicroWorlds* you can print to a specific text box. That's what makes the Eat sign appear to move:

```
text3,
ct
print "E
wait 4
print "A
wait 4
print "T
wait 4
```



To make the arrow sign, use the **insert** command, which leaves the cursor at the end of the text inserted (print moves the cursor down to the next line):

```
text4,
ct
insert "-
insert "-
insert ">
```



Notice that the text is cleared before it inserts or prints. This starts the sign "fresh" each time the procedure is repeated.

## Add Procedures and Buttons

Procedures in *MicroWorlds* are on one page in each project, although a project may have many pages (18 pages the last time I checked). Just as in *LogoWriter* versions, the procedures page can be annotated. It makes sense, then, to mark off the procedures page so that procedures are clearly identified by pages.

*MicroWorlds* has parallel processing. This allows each sign to "do its own thing" without regard to what other objects (such as turtles) are doing. Thus, while you can still write procedures the old way by thinking of the smallest piece each sign does in a lengthy sequence of actions for all the signs, you can also set up a process for each sign. Here's an example of how to make the Eat sign run "forever":

```
to eat
text3,
forever [ct print "E wait 4 print "A
   wait 4 insert "T wait 4]
end
```

Running a process forever is analogous to invoking a recursive procedure. For this signs project, each sign had its own task to run forever. All of these procedures are combined into a signs procedure:

```
to signs
eat
arrows
diner
end
```

I made a signs button that ran "For Once" because the processes themselves run forever, or until you stop them. I clicked on the button—and sat back and watched, astonished as the screen came to life, with all the signs flashing at their own tempo.

## More and More and More

The drawing tools included with *MicroWorlds* came in handy for making a nice background. The bright colors of the text seemed to suit the "neon" project perfectly.

Obviously, this is only the beginning of an idea that could be developed quite extensively. The *MicroWorlds* shapes can be created in multiple colors. Each color has

a wide range, from the lightest pastel to a very dark tone. Drawing tools are all available in the Shapes Editor also. Very intricate shapes can be created. The size of the turtle—and the shape it's wearing—can also change, from a minimum size of 5 to a maximum of 100. A lot of fun could be had with "growing" signs as well! The shapes could be used to add additional animation to signs, either as moving objects around the frame or as simulation of a more complex neon display. And since there can be as many as a hundred turtles in a project, it's possible to think of choreographing some pretty fancy neon!

Expect more about new guises for old ideas in the next issue of *Logo Exchange*. Until then, notice the signs around you. Try making your own project. Send us your results. Who knows—you might end up on the cover of *LX*!

If you would like to explore signs some more, see the Advertising Project in the *MicroWorlds Language Arts* package for some other ideas to use with signs.

Eadie Adamson has been working over the past year as an independent educational consultant. A large part of her time was spent as a consultant with Logo Computer Systems, working on the *MicroWorlds* project. She can be reached at

1199 Park Avenue, Apt. 3A
New York, NY 10128
AppleLink: EadieA

# Anno's Introduction to Factorials

**by Robert Macdonald**

*[Editor's note: Because Sandy Dawson was unable to contribute a math column this month, we have included two of Robert Macdonald's Musings that are mathematical in nature.]*

**Musings**

American publishers produce a surprising number of books that are primarily geared to the interests of children but that also intrigue adults. Authors frequently either secure the talents of gifted illustrators or are noted illustrators themselves. The Australian writer Graeme Base comes quickly to mind. He blends text and illustration with telling effect. Last fall I purchased his *The Sign of the Seahorse: A Tale of Greed and High Adventure in Two Acts* and quickly loaned it to friends, pleading that they incorporate it into a science unit on the fragile ecology of a coral reef (Base, 1992).

There is hardly a teacher who hasn't marveled at the illustrative talents of Martin Handford. Trying to locate Waldo has engaged many of us the past few years. On occasion, a magnifying glass is an essential tool (Handford, 1989).

Many of these publications are rich sources for mathematics, language, and science encounters. They become priceless teaching tools.

## Anno Introduces Factorials

One of my favorite authors and illustrators is Mitsumasa Anno. For almost a decade I have delighted in the math applications made possible by reading through and applying the ideas generated by *Anno's Mysterious Multiplying Jar* (Anno & Anno, 1983).

With limited text but superb illustrations, Anno magically unfolds the marvels of factorials. It is a concept of great mathematical subtlety, but it is a concept we apply frequently. It is also a concept that may easily be presented to children. It would be difficult to envision a better vehicle for this than Anno's picturesque text and vivid illustrations.

To demonstrate, I will summarize his simple text:

- There is a jar filled with water. The water becomes a vast sea.
- In the sea, one island rises. On the island are found two countries.
- Each country possesses three mountains. Each mountain supports four walled kingdoms. Within each walled kingdom are five villages.
- Each village embraces six houses. Each house has seven rooms.
- Each room contains eight cupboards. In each cupboard are found nine boxes. Within each box are ten jars.

Thus the tale should expand eternally as each of the ten jars could be filled with water, the water in each becoming a vast sea.

At the conclusion of his illustrated story, Anno begins to represent the power of factorials with red dots. This is applied from 1 factorial to 8 factorial. Thereafter, he foregoes illustrating factorials. Anno determines that he would need more than 180 pages of tiny red dots to represent 10 factorial.

Let's consider just what a factorial is. Obviously *factorial* is a term applied by mathematicians to a certain type of numerical relationship. Surprisingly, the symbol of this relationship is the exclamation point. Marilyn Burns has published a delightful introduction to this exclamation point and its permutational ramifications in *Math for Smarty Pants* (Burns, 1982).

Suppose in your mathematical encounters you come across 4!. You read it as "four factorial." The exclamation point signals that the product of 4 and the next smaller number 3 is to be multiplied by the next smaller number 2, all the way down to 1. In brief:

```
4! = 4 * 3 * 2 * 1 = 24
```

By applying this numerical relationship, a mathematician can cause a rapid growth in numbers. For example:

```
1! = 1
2! = 2 * 1 = 2
3! = 3 * 2 * 1 = 6
4! = 4 * 3 * 2 * 1 = 24
5! = 5 * 4 * 3 * 2 * 1 = 120
6! = 6 * 5 * 4 * 3 * 2 * 1 = 720
7! = 7 * 6 * 5 * 4 * 3 * 2 * 1 = 5040
8! = 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 40320
9! = 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 362880
10! =10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 3628800
```

## A Factorial Program

While we could quickly verify the above results on a calculator, we can verify them more easily on a computer. Some versions of *LogoWriter* (e.g., Apple,

Version 2.0) do contain a factorial program in the Math.Tools. My Macintosh version does not. If your version of Logo does not contain a program for producing factorials, try this one:

```
to factorial :input
ifelse :input = 0 [output 1] [output
   :input * factorial (:input -1)]
end
```

Use a numerical input to factorial and the **print** command to get the value of a given factorial. For example,

```
print factorial 5
```

causes 120 to appear in the work area of the screen.

To avoid typing the word "factorial," use the following adaptation:

```
to f :input
output factorial :input
end
```

Again prefacing with a print command, give the command:

```
print f 5
```

Again, 120 appears on the screen.

## A Logo Calculator

Before doing some experiments with factorials, let's first write a computer program that will more easily permit inputting factorials and other operations. For some years I have used the following Logo program to provide some of the conveniences of a calculator application. With the program, I can avoid using the commands **print, show**, and **type**. It has some advantages over a calculator in that I can solve problems requiring a true or false reply. The program operates on the principle of the recursive printing and running of a input accepted by the reporter **readlist**. Only an operational error will cause the user to "kick-out" of the program. The direction provided in the Command Center helps the user get started again. My students have enjoyed using it, and I have been able to apply it in countless circumstances where it had advantages over a calculator. These will be demonstrated later in this column.

```
to startup
clearpage
type [If there is an error which
   causes an exit from the program,
   type C and touch RETURN to re-
   enter. Type S to stop program.]
type char 13
calculate
end

to clearpage
if not front? [flip]
rg
ct
ht
cc
end

to calculate
print run readlist
print [ ]
if readchar = "S [stop]
calculate
end

to C
calculate
end
```

## Experimenting With the Logo Calculator

Let's play around with the previous program. The startup procedure will bring the program up and activate it. Enter the following (I used the *LogoWriter* Macintosh version for this):

```
5 + 3          (press RETURN)
5 + 3 = 8      (press RETURN)
5 - 4 = 2      (press RETURN after each entry)
25 - 3 * 5 / 2
(25 - (3 * 5) / 2
763 - 45
(14 - 3) + 7
4 * 7
42 / 6
remainder 43 6
factorial 9    (You may have to enter
               the factorial program.)
f 9
(4 / 5) + (4 /5) + (4 / 5)
abs -24
round 46.8
sqrt 64
divisor? 6 42
```

Incidentally, the predicate *divisor?* is not found in the Macintosh version, although it may be found in the Math.Tools of Apple *LogoWriter* (Version 2). If your work with children makes it convenient for you to have a predicate that determines whether the first input

divides evenly into the second, use the following small program:

```
to divisor? :number1 :number2
output 0 = remainder :number2 :num;
   ber1
end
```

*Remainder* is provided among the primitives on the Macintosh version.

You may find that preparing worksheets for an activity such as this proves helpful the first few times you use the program. With a worksheet, you will have some control over what you may want a class to accomplish within a certain time frame. Individualizing instruction may also be enhanced.

## Working With Factorials

Factorials may open up a way for students to look at counting and, at a higher level, at the permutations (changes) inherent in that manner of counting. This aspect of factorials will be considered in a future article.

Working with factorials is another way to increase a younger student's appreciation of large numbers. If you are working with Apple *LogoWriter* (Version 2.0), you will not be able to go beyond 14 factorial. But the Macintosh *LogoWriter* version will permit you to go up to 265 factorial. This will provide yet another vista for the alert student. Enrichment is an activity that an intuitive teacher applies as the occasion arises.

I always liked to have my students prepare a worksheet on factorials they could use to solve some problems that utilized the adding, subtracting, multiplying, or dividing of factorials. Doing arithmetic functions on a computer with factorials may easily lead to stack overflow due to the large numbers involved. But by filling in a chart, it is an easy enough task to provide answers without resorting to a computer.

A list of factorials through 10 has already been provided, so let's compile a list from 11 factorial through 20 factorial.

```
11! = 39916800
12! = 479001600
13! = 6227020800.0
14! = 87178291200.0
15! = 1307674368000.0
16! = 20922789888000.0
17! = 355687428096000.0
18! = 6402373705728000.0
19! = 1.21645100408832e17
20! = 2.43290200817664e18
```

Of course, you would not provide this information to students. They would work it out on a computer for themselves. A worksheet for such an activity is provided at the conclusion of the article.

As you glance over the list of factorials, you will note that the Macintosh moves from integer numbers to decimals numbers from 13 factorial on, and the decimal numbers are coded in scientific notation from 19 factorial. This provides another opportunity to explore the fascinating aspects of large numbers.

## Scientific Notation

Scientific notation is a convenient manner of expressing not only very large numbers but also very small numbers. The notation expresses a number as a multiple of 10, 100, 1000, or some other power of 10. The Macintosh documentation provided for *LogoWriter* states that "the display of numbers changes from decimal format to scientific notation at approximately 9999999999" (page 24). However, in experimenting, I discovered that it did not occur until the number was much greater. In the Command Center try

```
show 9999999999999999
```

My result was

```
9999999999999999.0
```

Typing

```
show 99999999999999999
```

results in

```
1.0e17
```

while typing

```
show 99999999999999990
```

gives

```
9.999999999999999e16
```

This activity, which calls for experimentation and discovery, fascinates children. It doesn't take them very long to realize that numbers written out the "long way" are awkward to handle and easy to make mistakes with (Jacobs, 1970). Working in base 10, as we move from a decimal point one digit to the right, we move by some group of 10s. The *e* we read in scientific notation is an exponent. In brief, it specifies the number of places to move the decimal point to the right (for a positive exponent) or to the left (for a negative exponent). The latter does not concern us here (Ashley, 1980).

For introductory purposes most youngsters will be satisfied with a straightforward explanation. If 19! = 1.212645100408832e17, transform it into "long notation" by moving the decimal point 17 digits to the right. Therefore, 19! = 121645100408832000.0. In preparing a worksheet, you must determine how you want the students to handle scientific notation. Do you want

them to keep it short, or do you want them to transform the factorial into long notation?

## Concluding the Study

Now for some experimentation with and reflection on factorials. After the students complete a worksheet detailing the long notation of a group of factorials (see the worksheet at the end of this article), have them ponder the following:

- What is the last digit of every factorial number greater than 4!
- Does 6! = 3 * 2! (No, I didn't forget to place an exclamation point after the 3.)
- Does 3! * 4! = 12!
- Does 3! + 4! = 7!
- Does 9! / 3 = 3! (No, I did not forget the exclamation point after the divisor.)
- Does 14! - 3! = 11!

Using what you have discovered about factorials, see if you can supply an answer to the following.

- How many ways can you arrange the letters in the word SCRAM?
- How many ways can you arrange the letters in the word LOOK?

In the next article we will study letter arrangements (permutations) using the knowledge we have gained of factorials.

## References


Anno, Mitsumasa, & Anno, Masaichiro. (1983). *Anno's Mysterious Multiplying Jar.* New York: Philomel Books.

Ashley, Ruth. (1980). *Background Math for a Computer World.* New York: John Wiley and Sons.

Base, Graeme. (1992). *The Sign of the Seahorse: A Tale of Greed and High Adventure in Two Acts.* New York: Harry N. Abrams, Inc

Burns, Marilyn. (1982). *Math for Smarty Pants.* Boston: Little, Brown and Company.

Handford, Martin. (1989). *Among Others: The Great Waldo Search.* Boston: Little, Brown and Company.

Jacobs, Harold R. (1970). *Mathematics. A Human Endeavor.* San Francisco: W.H. Freeman and Company.



Robert Macdonald
Hawthorne Meadows
10225 Nancy's Blvd.
Grosse Ile, MI 48138

# Worksheet on Factorials

Name _____ Date _____

Please fill in the value of the following factorials from the computer program in "long notation." Please do not use scientific notation. To help you solve problems, use the information your teacher has supplied. If you attempt to solve your problems using factorials on a computer, you will probably cause a stack overflow.

1! = _____

2! = _____

3! = _____

4! = _____

5! = _____

6! = _____

7! = _____

8! = _____

9! = _____

10! = _____

11! = _____

12! = _____

13! = _____

14! = _____

15! = _____

16! = _____

17! = _____

18! = _____

19! = _____

20! = _____

# A Study of Factorials and Permutations...Continued

by Robert Macdonald

In a previous article we explored *Anno's Mysterious Multiplying Jar*, which presents a fascinating introduction to factorials (Anno & Anno, 1983). As a reprise, we might remark that a factorial is a means of representing large numbers in a brief form.

Anno states in an Afterward that factorials can "tell you some amusing or useful things" that may be applied in other instances.

Let's say that there are five persons, named A, B, C, D, and E, who have to arrange their desks in a schoolroom. There is no special seating order. Now how many ways are there to arrange the desks (Anno & Anno, 1983)?

## Permutations

Anno brings forward the idea of permutations, which is inherent when considering the power of factorials. The *Oxford English Dictionary* (1933) defines the verb *permute* as "to alter the order of; to rearrange in a different order" (Vol. VII, p. 713).

In *Math for Smarty Pants* Marilyn Burns (1982) conjures up an interest in permutations with some delightful illustrations. She does this by lining up three friends at an ice cream store for cones. How many ways of lining up are available to them? What if another friend joins the initial three? How many different ways can they now line up?

Let's extend Ms. Burns' premise. Suppose each student wants three scoops of ice cream placed on a single cone. The available flavors are vanilla, strawberry, and chocolate. How many different ways can the server place these scoops one on top of the other?

Initially, I like to present a permutation problem to the class at the beginning of the year by asking how many different ways the class could line up in a single file each time they exited the room. If there were 25 students in a room, do you think it would be practical to even try a different line-up formation each time the pupils left as a group? Remember, math can give us a handle on practicality.

## Some Practical Simulations

When I finally got around to seriously considering the concept of permutations as a group activity, I suggested that the students determine how they might sit down on one side of a table to eat dinner. (All students seem to have good appetites.) It was an easy enough task to demonstrate physically. Select a table and a group of four students as props. Attach a number to each of them: 1, 2, 3, or 4. They will perform this simulation sitting on one side of a table facing the rest of the class.

All of the other students will experiment with 3 x 5 inch file cards on which are written the numbers 1, 2, 3, and 4. As they watch the demonstration, the students facing the demonstrators will manipulate their cards.

With one student sitting down to eat, there obviously is no problem. Only one seating arrangement is possible. It represents a 1 factorial. Only one person is available to sit down in the only place available. Hence, $1 = 1!$

Now add another student. We now have two students and two places. Permute.

```
1 2
2 1
```

There are two possible permutations (arrangements). They represent 2 factorial. Two possible people may sit in the first position. Because one occupies that chair, only one remains to occupy the second. Make certain that students are not only observing but are also working with the file cards to show both permutations.

With three students the rearrangements increase markedly. At this point, having students study Marilyn Burns' illustrations of three children lining up for ice cream will be very helpful (Burns, 1982, p. 95). We named the children Skates, Boots, and Lincoln. Ms. Burns' permutations are to be read horizontally. However, glancing through the list vertically does show some interesting patterning.

```
Skates Boots Lincoln
Skates Lincoln Boots
Boots Skates Lincoln
Boots Lincoln Skates
Lincoln Skates Boots
Lincoln Boots Skates
```

There are six permutations in this set. Obviously it is a 3 factorial. Any of the three available students may occupy the first position. Since one is selected to occupy that chair, two possibilities remain for occupying the second chair. One does. That leaves the remaining student to occupy the third position. $3 \times 2 \times 1 = 3!$ Have

the students working with file cards assign a number to these three figures: Skates = 1, Boots = 2, Lincoln = 3. Now have the students do a simulation with numbers.

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

The patterning is striking. In essence, there are three pairs of permutations. One pair begins with 1, another with 2, and the final pair with the number 3. Each pair differs in that the last two numbers switch places.

For our final simulation, add one more child to Ms. Burns' group of three. We shall permute Skates, Lincoln, Boots, and Tiny. However, we shall make some small alterations to Ms. Burns' permutations. It will add some variety and point out some valid possibilities in permutations. The eagle-eyed student will enjoy the slight changes.

```
Skates Lincoln Boots Tiny
Skates Lincoln Tiny Boots
Skates Boots Lincoln Tiny
Skates Boots Tiny Lincoln
Skates Tiny Lincoln Boots
Skates Tiny Boots Lincoln
Lincoln Skates Boots Tiny
Lincoln Skates Tiny Boots
Lincoln Boots Skates Tiny
Lincoln Boots Tiny Skates
Lincoln Tiny Skates Boots
Lincoln Tiny Boots Skates
Boots Skates Lincoln Tiny
Boots Skates Tiny Lincoln
Boots Lincoln Skates Tiny
Boots Lincoln Tiny Skates
Boots Tiny Skates Lincoln
Boots Tiny Lincoln Skates
Tiny Skates Lincoln Boots
Tiny Skates Boots Lincoln
Tiny Lincoln Skates Boots
Tiny Lincoln Boots Skates
Tiny Boots Skates Lincoln
Tiny Boots Lincoln Skates
```

Again, note the patterning. There are four pairs of six permutations. Within these six permutations, try to discern an internal pattern. Compare that patterning to that of Burns'. Obviously we have a 4 factorial. To begin we had four possible names we could have placed in our first position. We chose Skates. This gave us three names to use in our second position. We chose Lincoln. This gave us two names for use in our third position. We selected Boots. Tiny then was left for the fourth position. 4 x 3 x 2 x 1 = 4!

The activities we have been carrying out cry out for a computer application. Therefore, we shall supply one.

## A Permutation Program

The following program will permute a list entered by the user. Give the command **enter.** The program will prompt "Please enter the list you wish to permute." You might enter "Tim Bill Eric." If you choose to enter more than one letter or word as an entity in the list, you will have to use brackets, for example, [black dog] [green turtle] [yellow cat].

```
to enter
clearpage
print [Please enter the list you
   wish to permute:]
make "list readlist
change :list
end

to clearpage
if not front? [flip]
rg
ht
ct
cc
end

to change :list
clearpage
move.around [ ] [ ] :list
end

to move.around :stationary :moved
   :unmoved
if (sentence :moved :unmoved) = [ ]
   [print :stationary stop]
if :unmoved = [ ] [stop]
move.around (lput first :unmoved
   :stationary) [ ] (sentence :moved
   butfirst :unmoved)
move.around :stationary (lput first
   :unmoved :moved) (butfirst
   :unmoved)
end
```

## Experimenting With the Program

Let's experiment with the program. Why not begin with the list suggested above? Give the command **enter.** When prompted, enter "[Black dog] [Green turtle] [Yellow cat]." The following will appear:

```
[Black dog] [Green Turtle] [Yellow cat]
[Black dog] [Yellow cat] [Green Turtle]
[Green Turtle] [Black dog] [Yellow cat]
[Green Turtle] [Yellow cat] [Black dog]
[Yellow cat] [Black dog] [Green Turtle]
[Yellow cat] [Green Turtle] [Black dog]
```

Our permutations for Ms. Burns' children (Skates, Lincoln, Boots, and Tiny) were generated by the computer program. You might like to check it.

At the conclusion of the preceding article introducing factorials, two questions were posed:

- How many ways can you arrange the letters in the word SCRAM?
- How many ways can you arrange the letters in the word LOOK?

If we count the number of different letters in SCRAM, we may infer that the number of permutations might equal 5 factorial, or 120. Input SCRAM and see if this can be validated. Remember to enter SCRAM as S C R A M. Each letter is part of a list and must be separated by a space.

Now enter LOOK. How many different permutations are generated? Does it represent a 4 factorial? Be careful to enter LOOK as L O O K.

| | | | |
|---|---|---|---|
| LOOK | OLOK | OLOK | KLOO |
| LOKO | OLKO | OLKO | KLOO |
| LOOK | OOLK | OOLK | KOLO |
| LOKO | OOKL | OOKL | KOOL |
| LKOO | OKLO | OKLO | KOLO |
| LKOO | OKOL | OKOL | KOOL |

Pair off the identical permutations. You will find 12 pairs. Why aren't there 24 different permutations rather than 12 different permutations? If all the letters differed, there would be 24 permutations. But two of the letters are the same. If we were to divide by 2 factorial, we might eliminate this difficulty:

```
4! / 2! = 24 / 2 = 12
```

different permutations. In the above list of permutations, count each pair of similar permutations as one.

What if you were to enter a four-letter word made up of two pairs of letters, for example, LULU? How many different permutations are possible? Mathematically we suggest

```
4! / 2! x 2! = 24 / 4 = 6
```

different permutations. Demonstrate this with the computer program.

How many different permutations would be possible in a four-letter word such as MUMM, in which three of the letters are the same and one is different. Let's solve the problem mathematically:

```
4! / 3! = 24 / 6 = 4
```

different permutations. Prove it with a computer generation. Remember, when you enter MUMM as an input, enter it as M U M M.

## Conclusion

To round off our study, let's go back to the ice cream line introduced earlier. Enter vanilla, strawberry, and chocolate. How many permutations are possible?

Returning to the example from the beginning of this article involving a line of 25 students, would it be practical to even attempt to line up a 25-member class to demonstrate permutations? Probably not, because 25! = 15511210043330986000000000, or, in scientific notation, 1.5511210043330986e25.

Factorials and permutations can help expand the horizons of younger students. Consider them as tools with which some youngsters may take a plunge into an area of investigation they might not otherwise venture. If you are interested in exploring this further, see the accompanying worksheet.

## References


Anno, Mitsumasa, & Anno, Masaichiro. (1983). *Anno's Mysterious Multiplying Jar.* New York: Philomel Books.

*Oxford English Dictionary.* (1933). Oxford: Oxford University Press.

Burns, Marilyn. (1982). *Math for Smarty Pants.* Boston: Little, Brown and Company.



Robert Macdonald
Hawthorne Meadows
10225 Nancy's Blvd.
Grosse Ile, MI 48138

# Worksheet

Name _____ Date _____

If you were to arrange the letters in the following words in as many different ways possible, how many different permutations would you find in each?

| | |
|---|---|
| ACE | OXEN |
| _____ | _____ |
| LULL | AN |
| _____ | _____ |
| EXIT | ENTER |
| _____ | _____ |
| SCRABBLE | SASS |
| _____ | _____ |
| ZOO | ELEPHANT |
| _____ | _____ |

# *MicroWorlds:* A Review

**by Sharon Yoder**

"Microworlds" is certainly a familiar word to most *Logo Exchange* readers. At some point in time each of us has become intrigued with the microworld called Logo. As we learned more, we began to create our own microworlds within the Logo environment of our choice. In fact, the building of microworlds is often central to the work we do with Logo.

But now, "microworlds" has taken on an entirely new meaning. Logo Computer Systems, Inc. (LCSI), has recently released three packages, all bearing the name *MicroWorlds*. *MicroWorlds Language Arts* and *MicroWorlds Math* are products aimed at those who have not used Logo before or who have minimal Logo experience. These two packages provide both disk and print material that allows a Logo novice to begin to explore the use of Logo in both language arts and mathematics environments.

At the heart of the two curriculum-based packages is the *MicroWorlds* programming environment, packaged under the name of *MicroWorlds Project Builder*. The *MicroWorlds Project Builder* is an exciting new extension of LCSI's *LogoWriter*. There is all the power of *LogoWriter* combined with features reminiscent of *HyperCard* and *Kid Pix*.

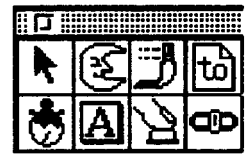
Because most readers of *LX* are at least novice users of Logo, this review will focus on the *MicroWorlds* environment. The projects available in the math and language arts packages will be discussed in other articles in this issue (see Logo Ideas) and in future articles in *LX*.

## So What *Is MicroWorlds?*

When you start *MicroWorlds Project Builder*, you see a screen familiar to *LogoWriter* users. There is a "page" and a Command Center. *MicroWorlds* projects are more like *HyperCard* stacks than they are like traditional Logo files. Each project consists of a number of pages, just as a *HyperCard* stack consists of a number of cards. Thus, the name at the top of the screen has both a project and a page name. Initially, you see

**Untitled (page1)**

at the top of the page. The other immediately obvious new feature is the Tool Palette found to the right of the Command Center.



The top row of the Tool Palette represents the tools available in *MicroWorlds;* the bottom row represents objects.

## Let's Look at the Tools

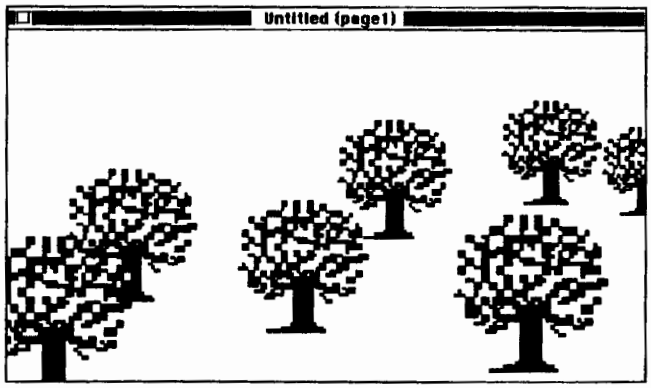
The leftmost tool, the arrow pointer, is called the Command Center tool. This tool is used to manipulate objects on the page or to place the pointer to work with text in the Command Center or on the page. For example, if you click on the turtle, you can drag it around the screen. Or, if you click in the Command Center, you can type Logo commands.

On the right side of the Tool Palette is the Procedure tool, the little page with a "to" in it. Clicking here takes you to the Procedures page. Unlike *LogoWriter*, there is a single procedure page for an entire project; that is, several pages share the same procedures.

So far, so good.

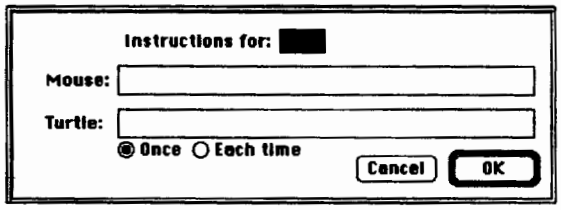
Now, if you click on the Shapes icon—the moon shape—you see the Shapes Center, which includes a series of shapes. The shapes are multicolored. In the Shapes Editor, you can modify or create shapes using 256 colors. You can rotate shapes and flip them with the click of a mouse. Among the tools available in the Shapes Center is the Stamper tool. Click on the Stamper tool and then click on a turtle, and the shape is stamped on the screen.

There are also Magnify tools. You can make a shape as large or as small as you want. (Do I hear the cheering of *LogoWriter* fans?) The following screen was created using a single turtle, changing its size, and repeatedly stamping its image on the page.




The paint brush next to the Shapes Tool is the Drawing tool. When you click on this tool, you see a range of graphics tools that will remind Mac users of most familiar paint programs. These tools are used to draw "backgrounds" for your scenes.

There's an exciting new use for the 256 colors available in *MicroWorlds* that goes beyond any graphics program you've ever seen. Each color family—a "column" of colors—is programmable. Any color family can be programmed to be sensitive to a turtle or a click of the mouse button. If you double click on a color you see




You can enter Logo commands or the names of Logo procedures.

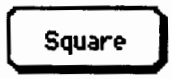
## Next, the Objects

In Logo, the turtle has always been an object, although most early Logo users would not have described it that way. Long before the **Objects** menu appeared in *HyperCard*, the turtle carried properties such as a heading, color, and pen state, just as *HyperCard* objects carry size, position, and style. In *MicroWorlds*, there are four kinds of objects: turtles, text boxes, buttons, and sliders.

At the bottom left of the Tools Palette is the Hatching Turtle tool. With it, you can create turtle after turtle. You are no longer limited to one turtle or four turtles. And, of course, each of these turtles can have its own shape.

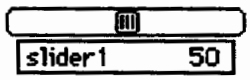
The next tool is the Text Box tool. Instead of typing text directly onto the page as you do in *LogoWriter* or *Logo Plus*, you create text boxes that resemble *HyperCard* fields. These text boxes have borders that can be visible or invisible, and the text boxes themselves can be made visible or invisible or can be "stamped" onto the painted background.

The third tool is used to create buttons. In *MicroWorlds*, buttons are named using either Logo commands or the name of Logo procedures.




Buttons can have the action assigned to happen once or many times. Buttons can be used to control the action of the screen. They can make turtles move, put text into text boxes, or cause the next page in a project to appear.

The last object is a slider.




Sliders are really reporters (or operations). But they are "variable" reporters. That is, as you drag the slide along the slider, the value "output" by the slider changes. Thus, if you type

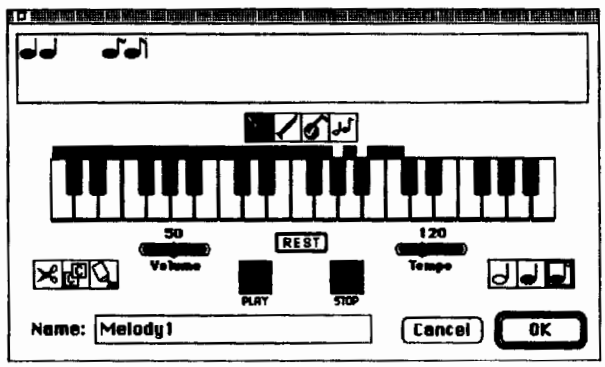
```
forward slider1
```

the distance that the turtle moves depends on the current setting of the slider. Sliders allow you to control the action on the screen that has been programmed by someone else.

## The Features Multiply

In addition to the power available using the Tool Palette, the menu bar provides even more power. Pages can be copied and pasted or even duplicated, allowing you to create a background and easily use that same background on many pages. Fonts can be changed; typeface, size, style, and color can all be modified. There's a powerful Help menu that offers a variety of ways to seek assistance.

For those of you who enjoy using sound, the Gadgets menu allows you to record sound or create "melodies":




(I think I hear that cheering again!)

As if this is not enough, multiple actions can occur at the same time. That is, *MicroWorlds* includes parallel processing. For example, you can have several turtles moving on the screen, each doing a different action at the same time. Or you can have the song play while the turtle moves. (I *know* I hear the cheering now.)

## Complaints Anyone?

With all of these features, how can there be anything negative about *MicroWorlds*? Without a doubt, those who are Logo purists will probably not like *MicroWorlds*. It "automates" many things that we have long thought encourage thinking skills and problem solving. My personal opinion is that *MicroWorlds* offers a new set of opportunities to learn problem solving and thinking skills, in much the same way that *LogoWriter* did. However, the skills that are exercised are frequently not the traditional Logo skills.

There is the problem that *MicroWorlds* runs only on a 256-color Macintosh. Few, if any, schools are equipped with labs and classrooms full of such powerful equipment. Clearly, LCSI is thinking of the future, but those many Apple IIe users out there will be very disappointed, as will all of you DOS and Windows users.



My greatest concern with *MicroWorlds* spills over from the desktop-publishing world. If I have a project with a number of buttons and sliders, I'd like to be able to click on a button, have a text box with instructions appear, and then have the slider controlling the action appear. When I've finished with the task at hand, I might like to hide the slider, text box, and even the button. Unfortunately, only the text box can be made visible and invisible. The result is that *MicroWorlds* pages can quickly become cluttered with buttons and sliders. It is rather difficult to design attractive pages on which a lot of action can occur.

As is usually the case with new Logos aimed at the K-12 market, high-end Logo users among you will be unhappy because some of your favorite primitives, such as **define** and **text**, are missing. But then you probably won't be a *MicroWorlds* fan anyway.

## And in Conclusion

*MicroWorlds* has obviously been carefully planned and beautifully designed. The software itself is strikingly beautiful. The documentation that I have seen as of this writing is much better than the documentation accompanying past versions of Logo from LCSI. It is thorough, attractive, and easy to use.

If I were teaching in an elementary or middle school and enjoyed using both Logo and *HyperCard* environments with my students, I would certainly be moving toward *MicroWorlds*. It provides the Logo programming language, hypermedia capabilities, color, and a wealth of easy-to-use tools for those just getting started. Secondary students would no doubt enjoy it as much as they enjoy *HyperCard* or *LinkWay*, if not more so. And Logo is, in my opinion, a much better first programming language than HyperTalk or *LinkWay* scripting.

At the university level, I find myself considering giving my hypermedia classes a choice among *HyperCard*, *MicroWorlds*, and *LinkWay*. After all, university students should have a chance to have fun too!


Sharon Yoder
Education 170, DLIL
College of Education
University of Oregon
Eugene, OR 97403
Internet: yoder@oregon.uoregon.edu

# "Arc"aeological Digging

by Dorothy Fitch

Welcome back to this column for beginners, where we investigate interesting ideas that do not involve lots of Logo expertise. This time we'll explore arcs in ways that kept me fascinated for hours. I hope you'll find them intriguing as well.

## The Basic Arc

Arcs may seem complicated to draw, but they are really just part of a circle. Can you draw a circle in Logo? A typical instruction is:

```
REPEAT 360 [FORWARD 1 RIGHT 1]
```

To draw a smaller circle, try:

```
REPEAT 18 [FORWARD 10 RIGHT 20]
```

The Total Turtle Trip Theorem guarantees that if the total number of degrees turned is 360, the turtle will end up pointing in its original direction.

Here are two simple arc procedures that each draw a quarter of a circle. You can tell that by adding the total number of turns. R draws an arc to the right; L draws an arc to the left. Try them out.

```
TO R
REPEAT 9 [FORWARD 3 RIGHT 10]
END
```

```
TO L
REPEAT 9 [FORWARD 3 LEFT 10]
END
```

## Arc Patterns

What can you do with these R and L arc procedures? Try using them to form patterns. For example, clear the screen and turn the turtle RIGHT 45. Then continue the pattern R L R L R L for a design like this, which may wrap differently around your screen:

The following repeated pattern produces a bone-shape design:

```
REPEAT 2 [R R L R]
```

Can you re-create the patterns on the Copy Me Page at the end of this column? How many new arc patterns can you and your students invent?

## Arc Explorer's Shortcut

To experiment with different patterns, you can test a sequence of R and L commands on one line and then use REPEAT to continue your pattern. You may also find this short single-keystroke procedure to be a useful tool. After you define it, type KEYS to run it. Press R or L to draw arcs, C to clear the screen, and Q to quit.

```
TO KEYS
MAKE "KEY RC
IF :KEY = "R [R]
IF :KEY = "L [L]
IF :KEY = "C [CG]
IF :KEY = "Q [STOP]
KEYS
END
```

Version Notes: Your Logo may need to use READCHAR instead of RC, and DRAW or CS instead of CG. You may need to press down your computer's Caps Lock key. You may need this IF syntax: IF :KEY = "R THEN STOP }

Press the R and L keys in a pattern to make a design. Then transcribe the keystrokes into a REPEAT instruction, if you want to save them. For example, if you repeated the pattern R R L four times, this procedure would draw that design:

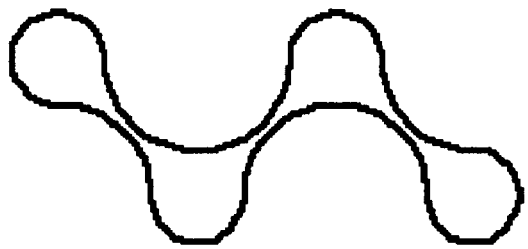
```
TO DESIGN
REPEAT 4 [R R L]
END
```

## An Interesting Detour

Another experiment is to make the R arc a different size from the L arc. An "inchworm" pattern (R R L L R R L R) made with equal-size arcs looks like this:

Now change the L arc procedure to this:

```
TO L
REPEAT 18 [FORWARD 3 LEFT 5]
END
```



The difference between the arc sizes makes this "inchworm" look rather warped! Try other designs with unequal arc sizes and see how the change affects them.

## Arc Flowers

Arcs also make great flower petals. For bigger petals, change the number after FORWARD in the R arc procedure, like this:

```
TO R
REPEAT 9 [FORWARD 10 RIGHT 10]
END
```

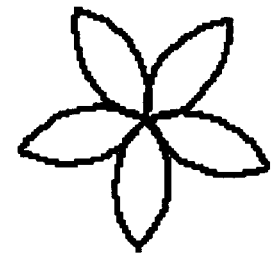
Can you figure out how to make a flower petal out of two arcs? Experiment to find the number after RIGHT in this procedure that completes the petal, leaving the turtle where it started.

```
TO ARC
REPEAT 2 [R RIGHT ___ ]
END
```

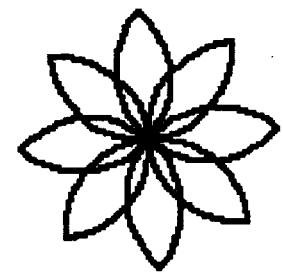
If it seems as though the Total Turtle Trip Theorem is not in effect, remember that there are also turns in the R procedure. The turtle still turns a total of 360°! (Answer: The number to fill in the blank after RIGHT in the procedure above is 90.)

Now you can use REPEAT to make some beautiful flower designs. Try these examples to see how simple changes in an instruction can make dramatic changes in the design. The more petals you draw, the more "layers" of complexity are added to the flower's design.

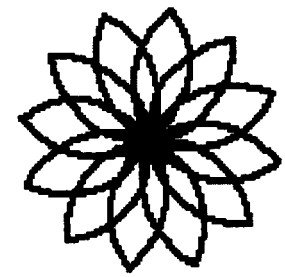
```
REPEAT 5 [ARC RIGHT 72]
```



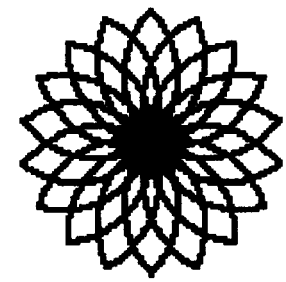
```
REPEAT 8 [ARC RIGHT 45]
```



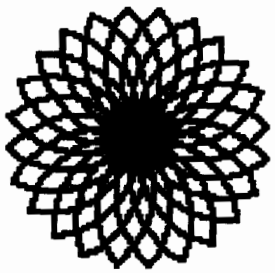
```
REPEAT 12 [ARC RIGHT 30]
```



```
REPEAT 18 [ARC RIGHT 20]
```



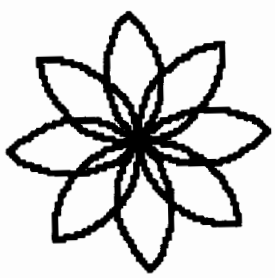
```
REPEAT 24 [ARC RIGHT 15]
```



Perhaps you want to fill the petals with color. Terrapin Logo for the Macintosh offers a FILLSH command, which takes as input a list of instructions for a shape it can fill. This is perfect for what we want to do.

Let's use the flower shape with eight petals. Here again is the basic outline:

```
REPEAT 8 [ARC RIGHT 45]
```

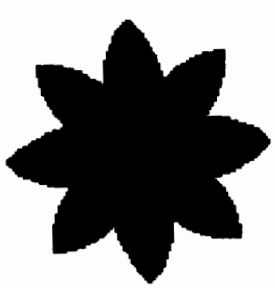


We can write a PETAL procedure that fills an arc, like this:

```
TO PETAL
FILLSH [ARC]
END
```

Then we can use REPEAT to draw this flower.

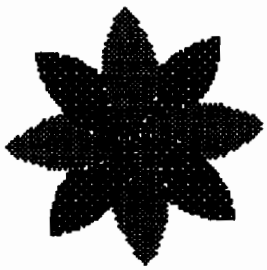
```
REPEAT 8 [PETAL RIGHT 45]
```



## Extra for Experts

(Sorry, beginners, but I got carried away and just couldn't resist the obvious next step!)

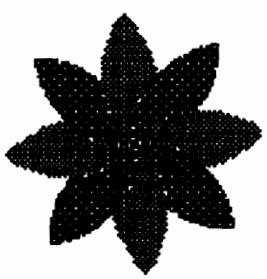
You might want to make the petals different colors, or, for the purpose of illustration in a black-and-white publication, we can show them in different pen patterns. If we alternate colors or patterns, the flower looks like this:



The procedure for this flower is:

```
TO MIXED.PETALS
REPEAT 4 [SETPPATTERN 2 PETAL RIGHT
   45 SETPPATTERN 4 PETAL RIGHT 45]
END
```

You may notice in the preceding design that the last petal drawn overlaps both darker petals beside it. It would be nice if its right side were tucked under the neighboring petal, as it would be in a pinwheel. We want it to look like this:



Can you see the difference between this flower and the preceding one? How can we solve this problem?

Drawing another petal wouldn't solve it; it would just move the problem one petal to the right. But we could draw half a petal to make the left edge of the darker petal overlap the right edge of the lighter, vertical petal. Here is how it would work.

```
TO HALFPETAL
SETPPATTERN 2
FILLSH [REPEAT 9 [FORWARD 10 RIGHT
   10] RIGHT 130 FORWARD 81.5]
RIGHT 140
END
```

I didn't compute the final turn and forward numbers mathematically—I confess to using the somewhat tedious, but very effective, trial-and-error method.

HALFPETAL can now be inserted as the last instruction in the MIXED.PETALS procedure, and we'll see the desired effect. There are probably other, perhaps simpler, solutions to this problem. I encourage you to look for one of your own.

Until next time, happy Logo adventures!

Dorothy Fitch has been director of product development at Terrapin since 1987. A former music educator, she has also directed a computer education classroom for teachers and students and provided inservice training and curriculum development for schools. She is the author of *Logo Data Toolkit* and co-author of *Kinderlogo*, a single-keystroke Logo curriculum for young learners. At Terrapin, she coordinates software development, edits curriculum materials, writes documentation, and presents sessions at regional and national conferences.

Dorothy Fitch
Terrapin Software, Inc.
400 Riverside Street
Portland, ME 04103-1068

CompuServe: 71760,366
Internet: 71760.366@compuserve.com
800/972-8200

*LX Copy-Me Page!*

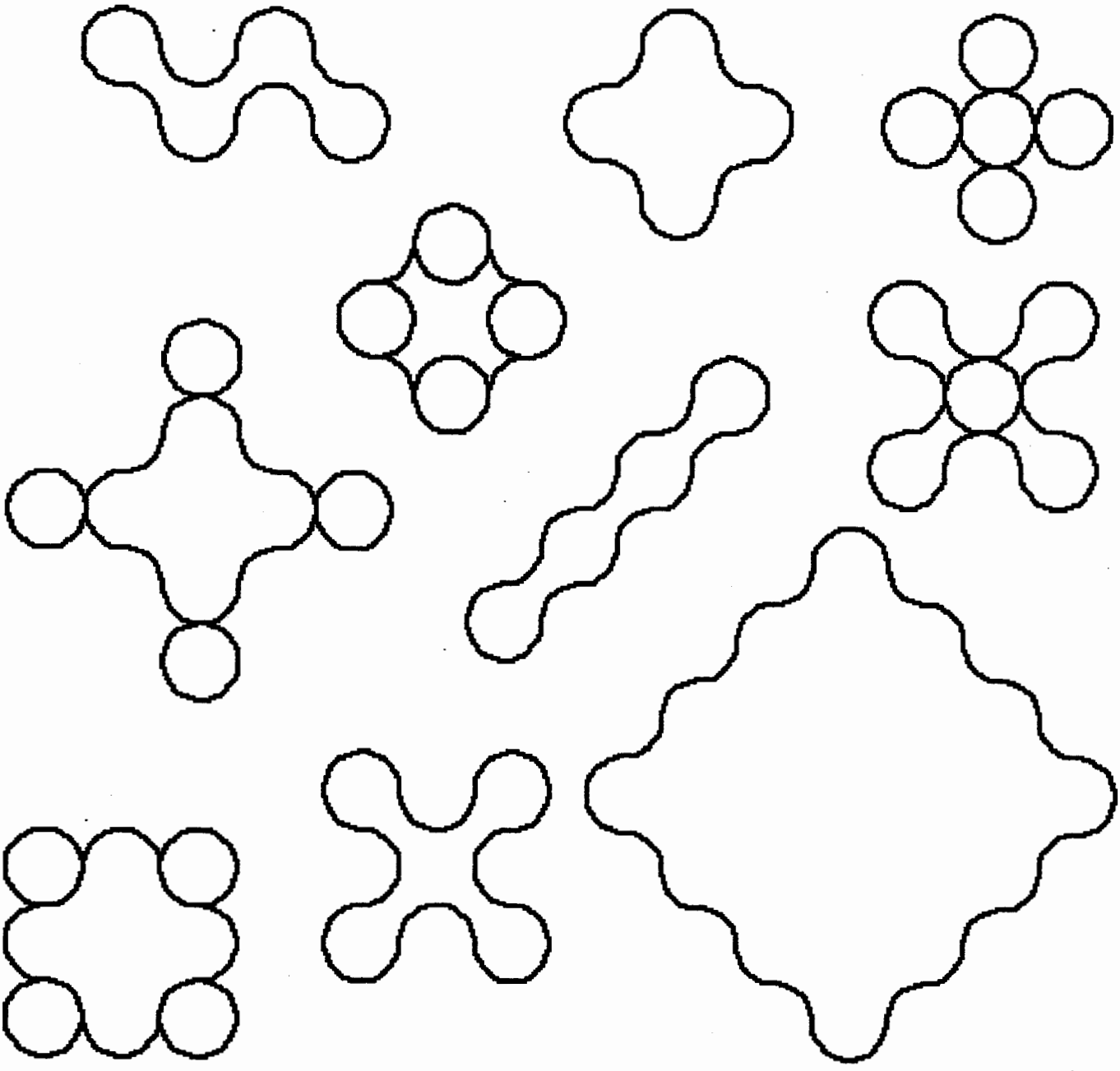
# "Arc"aeological Digging

Can you find the patterns that repeat to make these designs, using only these two procedures?

```
TO R
REPEAT 9 [FORWARD 3 RIGHT 10]
END

TO L
REPEAT 9 [FORWARD 3 LEFT 10]
END
```

*Example:* `REPEAT 2 [R R L R]`



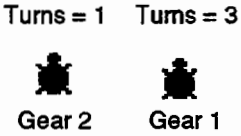
# Gear Ratios—A Simulation

by Glen L. Bull and Gina L. Bull

The history of Logo has always had links to gears and ratios. In the preface to *Mindstorms*, Papert describes his initial introduction to mathematics through the internal metaphor of gears. Later Logo robotics systems, such as LEGO-Logo, have provided actual gears and motors that can be controlled through Logo programs. This type of robotics system can provide practical experience with mathematics and gears. Children can become quite engaged in efforts to determine whether a larger or a smaller gear will be required to make a car run faster.

It is also possible to develop explorations with gears and ratios within Logo itself, even without any physical gears or external devices connected to the computer. In the 1980s, the term *microworld* was popularized in Logo circles. One popular microworld consisted of a "dynaturtle" representing a spaceship orbiting a planet. By playing with the dynaturtle, classes could experiment with the laws of physics.

In this column we will describe the beginnings of a gear simulation developed in *LogoWriter*. (This simulation can also be adapted for other versions of Logo.) The framework for the Gears program consists of spinning turtles (representing gears), and a counter that records how many times each gear (turtle) has turned.




In the preceding illustration, Turtle 1 (Gear 1) has made three complete turns, while Turtle 2 (Gear 2) has made a single turn. In other words, Turtle 2 completes one-third of a turn for every complete turn that Turtle 1 makes.

## Creating Startup and Reset Procedures

*LogoWriter* allows as many as four turtles to be displayed on the screen at one time. In this illustrative simulation, only two of the turtles will be employed as gears, but a more elaborate simulation could be employed using all four turtles. A startup procedure is needed to turn on two of the turtles. Traditionally Turtle 0 is the one that appears on the initial *LogoWriter* screen. However, we felt it would be simpler to refer to Turtles 1 and 2 instead of Turtles 0 and 1. For that reason, in the following startup procedure, Turtle 0 is hidden and Turtles 1 and 2 are displayed.

```
To StartUp
Tell 0 HT
Tell 1 ST
Tell 2 ST
Reset
End
```

After Turtles 1 and 2 are visible, counters are needed to record the number of times each turtle makes a complete revolution. A **Reset** procedure resets the counters for both turtles/gears to 0. In addition, the **Reset** procedure is used to establish the ratio between the two gears. The gear ratio determines how many times Gear 1 must turn around before Gear 2 makes a single turn. In this instance the gear ratio is set to 3; that is, Gear 1 must turn around three times before Gear 2 makes a complete turn.

```
To Reset
CC CG
Make "Gear1 0
Make "Gear2 0
Make "Gear.Ratio 3
Show.Gear.Ratio
End
```

The **Reset** procedure also displays the initial settings of the gear counters. This is accomplished in the procedure **Show.Gear.Ratio**. This is written as a separate subprocedure because it will also be used later by other procedures.

```
To Show.Gear.Ratio
CT
PRINT SENTENCE [Gear 1 =] :Gear1
PRINT SENTENCE [Gear 2 =] :Gear2
End
```

Once the **Startup**, **Reset**, and **Show.Gear.Ratio** procedures are completed, try them out by typing **Startup** in the Command Center at the bottom of the *LogoWriter* screen:

```
Startup
```

Two turtles should appear on the screen, and an initial value of 0:0 should appear in the counters in the top window:

```
Gear 1 = 0
Gear 2 = 0
```

## Gear Procedures

After the startup procedures have been created, a procedure to turn the gear is needed. To turn Gear 1/Turtle 1, the command "Tell 1" is invoked to let Turtle 1 know that we want to talk to it. Next we tell Turtle 1 to make a complete 360-degree turn. When that turn has been completed, "1" is added to the Gear 1 counter.

```
To Turn.Gear1
Tell 1
Repeat 360 [RT 1]
Make "Gear1 :Gear1 + 1
End
```

Try out the **Turn.Gear1** procedure by entering the command in the *LogoWriter* Command Center:

```
Turn.Gear1
```

Turtle 1 should spin around as soon as you press the Return key.

The rate at which the second gear/turtle spins around depends on Turtle 1. An initial gear ratio of 3:1 was specified in the **Reset** procedure, meaning that Turtle 1 will spin around three times for every complete turn made by Turtle 2. In order to maintain this ratio, we tell Turtle 2 to turn around by 360 degrees divided by the gear ratio. For example, with a gear ratio of 3, Turtle 2 will spin around by 360/3 (360/3 equals 120 degrees, or one-third of a turn).

```
To Turn.Gear2
Tell 2
Repeat (ROUND 360 / :Gear.Ratio) [LT
   1]
If (Remainder :Gear1 :Gear.Ratio) =
   0 [Make "Gear2 :Gear2 + 1]
End
```

The **Remainder** reporter is used to ensure that one count is added to the Gear 2 counter for every three turns of Gear 1. The **Remainder** reporter can be used to determine whether a number can be evenly divided by another number. For example,

```
PRINT REMAINDER 6 4
```

would produce a result of 2 (because 6 divided by 4 leaves a remainder of 2). On the other hand,

```
PRINT REMAINDER 6 3
```

produces a result of 0 (because 6 divided by 3 leaves a remainder of 0).

The **Remainder** reporter provides a method of determining when Gear 1 has completed three turns. When the Gear 1 counter is a number evenly divisible by 3 (3, 6, 9, etc.), the output of the **Remainder** reporter will be 0. When this is the case, the Gear 2 counter is incremented by 1.

The **Turn** procedure combines the **Turn.Gear1** and **Turn.Gear2** procedures, and displays the results with the **Show.Gear.Ratio** procedure.

```
To Turn
Turn.Gear1
Turn.Gear2
Show.Gear.Ratio
End
```

Each time a student types "Turn," Turtle 1 should turn around one time and Turtle 2 should turn around one-third of a turn. The gear ratio can be changed in the **Reset** procedure. For example, to change the gear ratio from 3:1 to 4:1, the number 3 should be changed to 4 in **Reset**:

```
To Reset
CC CG
Make "Gear1 0
Make "Gear2 0
Make "Gear.Ratio 4
Show.Gear.Ratio
End
```

After making this change in the editor, type **Reset** to reset the gear ratio to 4:1. Now how many times must Gear 1 turn before Gear 2 makes a complete revolution?

## Enhancing the Interface

If the program appears to be running successfully, you may want to enhance the user interface. We have found that the spacebar provides a convenient way to step the turtle through a series of turns, allowing the user to quickly examine the effect of different ratios. The following enhancement, **Gears**, allows the program to do this.

```
To Gears
Make "Input ReadChar
If :Input = "Q [Show [Exiting Gears]
   Stop]
Turn
Gears
End
```

Type "Q" to leave the Gears interface. The program will display the message "Exiting Gears" before stopping.

## Summary

We developed the Gears microworld as an illustration of a way in which classes without access to LEGO-

Logo gears might use Logo to explore gear ratios. The sample framework we outlined could be extended in a number of ways. For example, the third and fourth turtles could be used to add a third and fourth gear linked to the first two gears.

You may notice that due to round-off error, the second gear does not always make a complete turn for every three turns of the first gear. Real gears sometimes experience "slippage" of this kind as well. As an exercise, you may want to investigate ways of reducing or compensating for this electronic slippage.

There are a number of other programs that can be used to create Logo-like microworlds. We have been particularly impressed with *Interactive Physics II*, a program designed to allow students to quickly put together microworlds for exploration of physics concepts.

The simulation we have created in Logo is not completely realistic. For example, the turtles do not look like real gears with interlocking teeth, and we judge that it would be a real challenge to achieve this effect. Also, one of the gears would be larger than the other in a more realistic simulation, and the size of that gear would change depending on the gear ratio. (Can you tell which gear should be the larger one?) However, a simulation of this kind can provide a convenient way to explore the relationship between numeric abstractions and related events.

Glen Bull is an associate professor in the Instructional Technology Program of the Curry School of Education at the University of Virginia. Gina Bull is a system administrator in the Department of Computer Science at the University of Virginia. By day she works in a UNIX environment, by night in a Logo environment.

Internet Addresses: GBull@Virginia.edu, Gina@Virginia.edu
BITNET Addresses: GBull@Virginia, Gina@Virginia

# The Logo of My Dreams...

by John M. Sklar, MECA Editor

Several years ago I became enamored with Logo. However, I found the Logo implementations on the Macintosh and the MS-DOS platforms to be balky and, in a word, inelegant. Several weeks ago one of my students told me about a product from Harvard Associates, Inc., called PC Logo Version 4.0. I have taken a long look at this product, and I am impressed.

PC Logo 4.0 comes on a single disk and needs to be installed on a floppy or hard drive. The program supports most common MS-DOS graphics standards, including VGA. I found excellent performance on my Super VGA monitor and the Notebook display. The program is simple to install, and the publisher has included a short "Getting Started" booklet so that you need not thumb through the 300-page manual for setup instructions. If you have any computer experience or Logo experience, you may not even need the manual.

The program is the Logo of my dreams. It is fast, fast, fast; and it has a ton of convenient features I always wanted. For example, when you type a command it is stored in a buffer, and the up and down arrows scroll through them. This can save a lot of time and typing. Procedure definitions are accomplished right in the split-screen mode without going to the excellent full-screen editor. The program provides full mouse support; and, most importantly, it has a full Help system that is like a reference manual on disk. You simply type "help" and the command you need to know about. The program then presents you with a clear explanation and, in many cases, examples.

One feature that my son and I found very exciting was the ability to program several turtles at the same time. Here are three related procedures that will define six separate turtles, set each to a different color, and then send each on a different trip on the screen. The procedure ends by having each turtle draw a ball filled in with latitude- and longitude-type lines. The TELL command activates the six turtles, the ASK command addresses each separately, and the EACH command instructs each turtle to do something different based on the value of WHO. WHO is the currently active turtle. For the first iteration of the EACH loop, WHO is 0, so the right turn is 0; for turtle 1 it is 60; for turtle 2 it is 120; and so on, to turtle number 5, which results in RIGHT of 300. The two short procedures show the recursive capabilities of this language. In these procedures, the procedure seeds itself with a new value until a certain value is reached and then the procedure is abandoned. These procedures operate so quickly that the procedures literally jump onto the screen.

```
TO MANYCIRC
DRAW
HT
FS
TELL [0 1 2 3 4 5]
ASK 0 [SETPC 2]
ASK 1 [SETPC 4]
ASK 2 [SETPC 3]
ASK 3 [SETPC 5]
ASK 4 [SETPC 8]
ASK 5 [SETPC 9]
EACH [RIGHT 60 * WHO FORWARD 100]
MAKER_BALL 50
END

TO MAKER_BALL :N
IF :N = 0 THEN OTHER_BALL 50
EACH [STAMP OVAL 50 :N]
MAKER_BALL :N-5
END

TO OTHER_BALL :N
IF :N = 0 THEN TOPLEVEL
EACH [STAMPOVAL :N 50]
OTHER_BALL :N-5
END
```

The program supports more than 300 MS-DOS-compatible graphics printers, including the DeskJet 500. I don't think that anyone with an MS-DOS computer/printer combination will have trouble finding his or her printer in PC Logo's extensive list.

Harvard Associates, Inc., provides an excellent pricing structure. PC Logo is $100 plus the appropriate per-workstation fee based on the number of workstations licensed:

| | | |
|---|---|---|
| 1-20 | workstations | $15.00 each |
| 21-40 | workstations | $12.50 each |
| 41 + | workstations | $10.00 each |

A PC Logo licensee receives a complete set of materials, including software, documentation, license agreement, and home discount coupons for students. Additional documentation is available to licensed

schools or districts at $29.95 each. A PC Logo licensee may increase the number of licensed workstations at a future date by paying only the per-workstation fee. Policies like this encourage honesty and make it easy to keep your school lab "honest." Schools with stand-alone computers will find the class-pack licenses to be equally attractive, with a 20-pack selling for about $400.

The single-user package has a retail price of $99.95, with a street price of just under $70. Be sure to specify Version 4.0 if you decide to buy from a software dealer.

John M. Sklar, MECA Editor

Adapted from an article published in *MECA*, Milwaukee Educational Computing Association, January 1993 issue.

## New Paradigms in Classroom Research on Logo Learning

This monograph is the result of a research workshop at the National Educational Computing Conference (NECC) in 1991. These workshops provide a unique forum for the sharing of ideas by researchers in special areas of educational computing.

Each of the nine classroom research studies covered is a collaborative research project conducted by university researchers and classroom teachers. The focus is on effective teaching and on learning and assessment of Logo. The reports are organized into sections covering the cognitive outcomes of Logo learning, Logo learning in a social context, and the collaborative process itself.

Edited by Daniel Lynn Watt & Molly Lynn Watt
139 pages, 1993
ISBN 1-56484-041-7
Regular price: $19.95 plus $5.00 shipping
To order, contact ISTE at 800-336-5191.

## *PC Logo for Windows*™ Debuts

Harvard Associates is pleased to announce the immediate availability of *PC Logo for Windows*, a complete version of the Logo programming language for the *Microsoft Windows* ™ operating system.

"*PC Logo for Windows* is designed to appeal to those already familiar with Logo while extending its power and flexibility to the *Windows* environment with features like pull-down menus, buttons, multiple windows, advanced debugging tools, and a complete on-line Help system," according to Mr. William Glass, President of Harvard Associates. It joins PC Logo 4.0™, the latest release of the MS-DOS version of Logo developed and marketed by Harvard Associates.

Features of *PC Logo for Windows* include the full range of Logo capabilities, as well as multiple windows for commands, graphics, editing and debugging; multiple turtles that can change color, speed, and shape with the new shape editor; a complete music system for easy writing and playing tunes; and an automated, quick-install program. PC Logo provides control of the *Windows* environment via access to API and MCI for multimedia applications.

*PC Logo for Windows* comes complete with 500 pages of documentation, including a step-by-step tutorial and technical reference manual. Multiple-workstation licensing is available for educational institutions.

For more product information or to order *PC Logo for Windows* or DOS, please contact:

Harvard Associates, Inc.
10 Holworthy Street
Cambridge MA 02138
Phone: 617/492-0660
Fax: 617/492-4610

# The Ellipse and Logo

by Ken Large

## Preface

I have noticed in journals and among aficionados of Logo a sort of veneration for a small group of primitives—**forward, back, right,** and **left**—as if other primitives that can be used to give graphical results are somehow improper and therefore inferior.

In my view, this attitude severely limits the range and depth of interaction between mathematics and Logo, and interferes with the principle of open exploration and investigation that the idea of Logo sets out to uphold.

You will find no such skirting of taboo areas here. I prefer to adopt a pragmatic and hopefully broadminded approach, and use and adapt whatever I can find in Logo and mathematics to aid my quest for ellipses.

Some methods I describe require more mathematical expertise than others. Some require a fluency in Logo. All methods, I would claim, allow for further exploration and investigation into wider fields of both Logo and mathematics.

At the same time there is to be discovered a unifying structure hidden beneath my diversity of ways. That is to say, there is mathematically only one way to make an ellipse, and I display here only 18 variations of this single function.

## Preliminary Notes

In this article, you will find 18 ways to make ellipses or ellipse-like closed curves by using Logo.

Each section begins with a brief explanation. Then the procedures are given for drawing the curve. Finally, some specific instructions to type are given. Use whatever method your version of Logo provides for interrupting procedures to stop the curve-drawing process.

The procedure **easy** is used to clean the screen so that new lines can be seen more easily.

```
to easy :ch
if :ch = "c [clean]
end
```

In the procedures that need it, it is called by the line

```
if key? [easy readchar]
```

Pressing the c key clears the screen without stopping the procedure or changing the position of the turtle.

It might be of interest to compare the 18 elliptical shapes found in this article by matching height and width. **Compare** does this by integrating and matching two different methods.

```
to compare :x :a :b :ib :y :z
forward abs (:x * sin (:b))
make "p pos
pu
setpos sentence :y * cos :b :z *
   sin :b
pd
setc 2
setpos pos
setc 1
pu
setpos :p
pd
right :a
compare :x :a :b + :ib :ib :y :z
end
```

Thus, typing the following series of commands will simutaneously draw ellipses using two different processes.

```
cg
pu
ht
setpos [-90 0]
pd
compare 2 1 0 1 90 58
```

This particular version of **compare** compares **davell** and **trigell,** two of the procedures described below.

In these procedures, I use **pu setpos pos pd** to print dots. You may prefer to use a separate procedure, e.g.,

```
to dot
pu setpos pos pd
end
```

The procedure **abs** is used where the sign of the number must always be positive. If your version of Logo doesn't have **abs,** you can write your own procedure.

```
to abs :n
if :n < 0 [output minus :n]
output :n
end
```

If you don't have the primitive **minus,** you can use the negative sign directly, e.g., -:n or multiply the variable by -1, e.g., :n * -1.

The version of Logo used in the examples in this article is *LogoWriter 2.0.*

## 1. DISTANCE and Two Origins

In the procedure below, **od** is a global variable and stands for "old distance." The local variable **nd** stands for "new distance." In **seek**, the old distance is constantly compared to the new distance after the previous move. If the new distance is greater, the turtle turns right 12 degrees before moving forward. Otherwise, it just moves forward on the old heading. Origin 1 is at coordinate point [0 -30] and origin 2 is at [0 30]. At every turn the procedure **disell** compares the turtle's distance from the two origins. If the distance from origin 1—**d1**—is greater than the distance from origin two—**d2**—the first distance, **d1**, is used in **seek** as the new distance. If the opposite is true, the second distance, **d2**, is used in **seek** as the new distance. Note that the old distance could have come from either **d1** or **d2**. The procedure **seek** has been fine tuned by adjusting both numbers in **right 12** and **forward :nd / 14**. The turtle eventually settles down to a fairly tight orbit, and periodically typing **c** will allow you to see this clearly. The resulting shape is obviously not a true ellipse, but it is reasonably close.

```
to disell
make "d1 distance [0 -30]
make "d2 distance [0 30]
if key? [easy readchar]
if :d1 > :d2 [seek :d1]
if :d2 > :d1 [seek :d2]
disell
end

to seek :nd
if :nd > :od [right 12]
forward :nd / 14
name :nd "od
end
```

Then type

```
cg
make "od 0
right 20
forward 5
disell
```

## 2. TOWARDS and Two Origins

The method used in **towell** below is similar to **distell** above, but less complicated. It uses **towards** instead of the primitive **distance**. At each turn, the turtle moves one unit, approximately at right angles to origin 1—[50 0]—and then one unit toward origin 2—[-50 0]. In practice, **:a** is turned to 89 to give an elliptical shape, because **seth 90 + towards**, a true right angle, produces an elliptical spiral outwards.

```
to towell :a
if key? [easy readchar]
seth :a + towards [50 0]
forward 1
seth :a + towards [-50 0]
forward 1
towell :a
end
```

Then type

```
cg
forward 30
towell 89
```

This next procedure bisects the angle between the two headings to produce a smoother curve. It looks far more complicated than **towell** because of the inherent messiness of bisecting headings.

```
to towell2 :a
seth bisect :a forward 1
if key? [easy readchar]
towell2 :a
end

to bisect :a
make "h1 towards [50 0]
make "h2 towards [-50 0]
if :h2 - :h1 > 180 [output :a1 + :h1
   - (.5 * ((360 - :h2) + :h1 ))]
if :h2 - :h1 < 180 [output :a + :h1
   + .5 * (:h2 - :h1)]
end
```

Then type

```
cg
forward 30
towell2 89.3
```

## 3. Multiple Turtles

The next example uses turtle 0 and turtle 1 plus the primitives associated with multiple turtle use, namely, **rg**, **tell**, and **ask**. **Ttell** needs time to reach a stable elliptical orbit (remember to type **c** to clear). At any given turn, turtle 1 moves at a fixed speed approximately 90 degrees to turtle 0, which, of course, is also moving. The variable **d** controls the speed of reciprocating turtle 0. I suspect that **ttell** does not produce a true ellipse.

```
to ttell :a :d
tell 1
seth :a + towards ask 0 [pos]
forward 3
if key? [easy readchar]
tell 0
if or ycor > 20 ycor < -20
   [right 180] forward :d
ttell :a :d
end
```

Then type

```
rg
ttell 88 .5
```

The procedure **ttell2** has turtle 0 going in a circle instead of up and down. It also needs time to reach a stable elliptical orbit.

```
to ttell2 :a :d
tell 1 seth :a + towards sentence 0
   ask 0 [ycor]
forward 3
if key? [easy readchar]
tell 0
forward :d
right 3
ttell2 :a :d
end
```

Then type

```
rg
ttell2 88 3
```

## 4. Direct or Purist

The overt intention here is to produce an ellipse using only local geometry, i.e., direct manipulations of **forward, back, right,** and **left.**

```
to davell :x :a :b :ib
forward abs (:x * sin (:b))
right :a
davell :x :a :b + :ib :ib
end
```

(**Davell** is named after E. W. Davies, writing in the January 1990 issue of *Mathematics in School.*)
Then type

```
cg
ht
pu
setpos [-90 0]
pd
davell 2 1 0 1
```

Is this an ellipse or an ellipsoid? (See **compare** above.)

Here is an approach that is more direct than Davies' and that doesn't use trigonometry.

```
to oval :n :d
if or :n = :d :n = minus :d [make "n
   minus :n]
forward abs .02 * :n
right 1
oval :n + 1 :d
end
```

Then type

```
cg
oval -89 90
```

```
to cell :n :s
if or :n = 0 :n = 90 [make "s :s * -
   1]
forward .25 + .04 * :n
right 1
cell :n + :s :s
end
```

Then type

```
cg
pu
setpos t-145 0]
pd
cell 1 1
```

## 5. Trigonometry

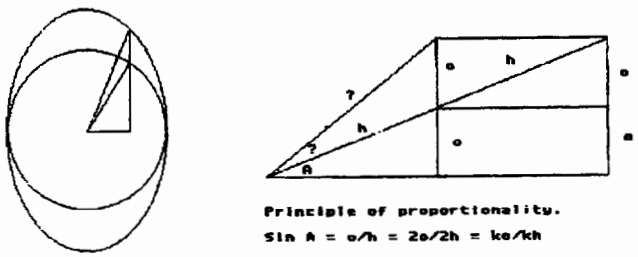
This next method is a modification of the well-known method of producing a circle in coordinate geometry using trigonometry. (Try :x = :y.)

```
to trigell :a :x :y
setpos sentence :x * cos :a :y * sin
   :a
trigell :a + 1 :x :y
end
```

Then type

```
cg
pu
setpos [80 0]
pd
trigell 0 80 40
```

I'm fairly certain that **trigell** produces a true ellipse. But I would be happy to be proved wrong. Consider the following diagram:




## 6. Self-Reference Using Position

In the following example, the turtle uses its own position (**ycor**) to tell it how much to move next.

```
to srpell :n
right .5
forward 1 + abs :n * ycor
right .5
srpell :n
end
```

Then type

```
cg
pu
left 90
forward 100
right 89.7
pd
srpell .013
```

## 7. Self-Reference Using Heading

The turtle uses its heading to tell it how much to move next. **Srhell** obviously doesn't make a true ellipse.

```
to srhell
forward 1 + .02 * remainder (heading
   + 45) 180
right 2
srhell
end
```

Then type

```
cg
pu
setpos [-50 0]
pd
srhell
```

## 8. Random

**Ranell** is very slow, so be prepared to wait. It uses a test condition for points satisfying the conics ratio for producing ellipses.

```
to ranell
pu
setpos sentence -26 + random 102 -20
   + random 40
pd
make "h1 1.2 * distance [-20 0]
make "h2 distance sentence -30 ycor
if (round (10 * :h1)) = round (10 *
   :h2) [setpos pos]
ranell
end
```

Then type

```
cg
ranell
```

## 9. Scanning

**Scanell** is similar to **ranell** above, but is not quite so slow.

```
to scanell :x :y
setpos sentence :x :y
make "h1 1.2 * distance [10 0]
make "h2 distance sentence 0 ycor
if (round (5 * :h1)) = round (5 *
   :h2) [pd setpos pos pu]
if :x = 62 [make "y :y - 1 make "x 2]
if :y < -15 [stop]
scanell :x + 1 :y
end
```

Then type

```
cg
pu
scanell 0 15
```

## 10. Recursion

**Recell** is an example of nested recursion. The variable :l increases in value as it loops down through the levels, and these values are used within the loop to draw the first quadrant of the ellipse. Then after it is stopped, the final line gives a graphical outcome, i.e., the second quadrant as **recell** loops back out with the :l variable, decreasing until the condition :1 = 0 is met. At that point, **recell** starts again.

```
to recell :1
if :1 = 90 [foward .02 * 90 stop]
forward .02 * :1
right 1
recell :1 + 1
right 1
forward .02 * :1
if :1 = 0 [recell :1]
end
```

Then type

```
cg
recell 0
```

## 11. List of Positions

**Posell** takes each position created in the circle-building **lposcirc** and multiplies the x coordinate by some amount. For example, if x = 1.5, the line stretches 50. This has the effect of stretching the circle along its x-axis and producing an ellipse. The command list is designed to show the process step-by-step.

This first procedure provides an empty global variable and clears the screen.

```
to setupp
cg
ct
make "p []
end
```

**Lpos** adds a coordinate to the list **p**.

```
to lpos
make "p lput pos :p
end
```

This next procedure behaves like **forward** but also records the start position of each forward move when creating shapes.

```
to fg :n
lpos forward :n
end
```

**Lposcirc** draws a circle using **fg**.

```
to lposcirc
repeat 72 [fg 5 right 5]
end

to posell :x :p
if :p = [] [stop]
make "q first :p
setpos sentence :x * first :q last
   :q
posell :x butfirst :p
end
```

Then type

```
cg
setupp
setpos [-50 0]
pd
lposcirc
cg
pr :p
ct
pu
setpos [-50 0]
posell 1.5 :p
pd
posell 1.5 :p
```

## 12. Setscrunch

If your version of Logo has a **setscrunch** (or the equivalent) primitive, you can certainly produce an ellipse, but using it is cheating because the way it functions is hidden. This problem is solved here by the use of procedures that duplicate the action of a **setscrunch** primitive and of **forward** and **right**, which are also hidden.

This first procedure is the equivalent of the primitive **setscrunch**.

```
to chscr :s
make "sc :s
end

to ddell
repeat 72 [tr 5 ri 5]
end
```

(This will not work without the **cl**, **ri**, and **tr** procedures below.)

Below are procedures that duplicate **forward** and **right** and are integrated with **chscr** above.

**m2** and **a2** are list-handling functions needed in procedures below. **m2** multiplies each element of a list by a constant, **a**. **a2** adds two lists together.

```
to m2 :a :l :nl
if :l = [] [output :nl]
make "nl sentence :nl :a * first :l
output m2 :a butfirst :l :nl
end

to a2 :l1 :l2 :nl
if :l1 = [] [output :nl]
make "nl sentence :nl (first :l1) +
   first :l2
output a2 butfirst :l1 butfirst :l2
   :nl
end
```

The procedures **ro** and **per** are trigonometric functions needed in **ri** below. **ro** rotates a vector coordinate **v** through angle **a**. **per** outputs a vector coordinate at (clockwise) right angles to the input vector **v**.

```
to ro :v :a
output a2 m2 cos :a :v [] m2 sin :a
   per :v [] []
end

to per :v
output sentence last :v minus first :v
end
```

**tr** duplicates the action of **forward**. Note that one dimension—the x-axis—is stretched by **:sc * item 2**. This allows the **setscrunch** effect to work within the duplicate primitive set.

```
to tr :d
make "np a2 :p m2 :d :h [] []
setpos sentence :sc * item 2 :np
   item 1 :np
make "p :np
end
```

This procedure duplicates the action of **right**.

```
to ri :a
make "h ro :h minus :a
end
```

The next procedure duplicates the action of **cg**. It sets up the global variables needed in the new "primitives" **tr, ri,** and **chscr.**

```
to cl
cg ht
make "p [0 0 ]
make "h [1 0 ]
make "nl []
end
```

Then type

```
cl
chscr 1.5
pu
ri -90
tr 50
ri 90
pd
ddell
```

## 13. 3D Logo

This procedure simply twists a circle in the third dimension, giving an elliptical view of it.

This first procedure loads the procedures for 3D Logo. These include such procedures as **pitch, roll, yaw, for,** and **cls,** which are needed to operate a 3D Logo. Some Logos now have these built in, but the necessary procedures may be modified from the procedures in Item 12 above.

```
to ddd
gettools "toolddd
end
```

Then type

```
ddd
cls
pu
roll 45
yaw -90
for 40
yaw 90
pd
repeat 72 [for 5 yaw 5]
```

## 14. Ballistics

The procedures in this section simulate the effect of gravity. The procedure **belell** doesn't give a true orbit because the gravitational attraction simulated by **g** works toward the x-line and not one point.

```
to balell :y :g :a :b
if or :y = 60 :y = -60 [make "g
   minus :g]
setpos sentence (30 * sin :a) 90 +
   :b
pd
setpos pos
pu
balell :y - :g :g :a + 3 :b + (:y /
   10)
end
```

Then type

```
cg
st
pu
balell 0 2 0 0
```

Note that the spread of dots marks change in velocity of the turtle.

**Beell** has evolved through the following modifications:

```
to bal :y :g :a :b
setpos sentence :a :b
pd
setpos pos
pu
bal :y - :g :g :a + 1 :b + (:y / 10)
end
```

Then type

```
cg
st
pu
bal 50 2 0 0
```

to see a ball falling forever faster.

```
to ballab :y :g :a :b
if or :y = 60 :y = -60 [make "g
   minus :g]
setpos sentence :a 90 + :b
pd
setpos pos
pu
ballab :y - :g :g :a + :g :b + (:y /
   10)
end
```

Then type

```
cg
st
pu
ballab 0 2 0 0
```

to see the effect of **bal** going back on itself, giving an ellipse shape with points.

In **balell,** the reflected ballistic curves are modified by calibrating the x line as sine frequencies instead of regular intervals. Using **(30 * sin :a)** gets rid of the points.

## 15. Celestial Mechanics

The procedures in this section set out to simulate the orbit of a satellite about its planet.

The turtle-space module sets off at the input vector of the command line but is immediately subject to gravitational attraction toward the planet. **Orbell** is a more realistic simulation than **balell** because it has gravity as a point source [40 0]. In the line beginning with **forward 100,** gravity is set to 1000 for scale purposes and falls off as the square of the distance from the center of the planet [40 0]. There is slight precession. Note that gravitational attraction is not reciprocal here.

```
to orbell :d
make "h heading
make "p pos
pd
setpos pos
pu
seth towards [40 0]
forward 1000 / ((distance [40 0]) *
   distance [40 0])
seth :h
forward :d
seth 180 + towards :p
orbell distance :p
end
```

Then type

```
cg
pu
setpos [40 0]
pd
setpos pos
pu
home
pd
right 31
orbell 5.5
```

Also try

```
right -60 6
```

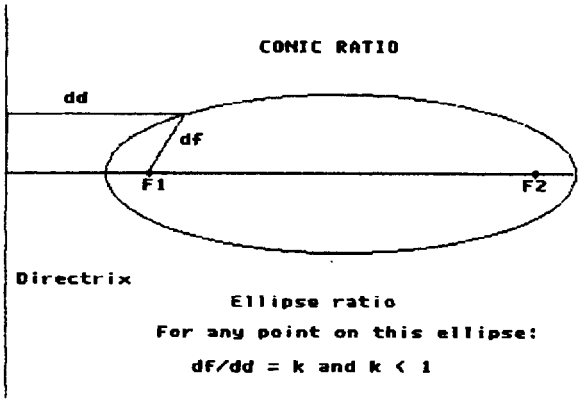
In **orbell2,** the planet and satellite attract each other. The planet has no initial velocity vector. This is more realistic in terms of mechanics but not so good at forming a closed ellipse because of the precession and decay of the orbit.

```
to orbell2 :d
ask 1 [planet]
make "h heading
make "p pos
pd
setpos pos
pu
seth towards ask 1 [pos]
forward 1000 / ((distance ask 1
   [pos]) * distance ask 1 [pos])
seth :h
forward :d
seth 180 + towards :p
orbell2 distance :p
end

to planet
seth towards ask 0 [pos]
forward 10 / ((distance ask 0 [pos])
   * distance ask 0 [pos])
end
```

Note that this gives reciprocal perturbation of planet. Then type

```
rg
tell 1
pu
ht
setpos [60 0]
pd
tell 0
ht
pu
setpos [-100 0]
pd
orbell2 1
```



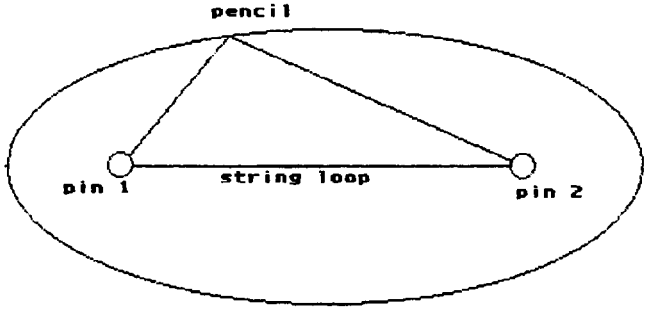
## 16. String Method

```
to strell :d
forward 1
if :d < (distance [-50 0]) + (dis-
   tance [50 0]) [pd setpos pos pu
   setpos [0 0] right 5 forward 40]
strell :d
end
```

Then type

```
cg
pu
forward 40
strell 140
```

This next procedure is a simulation of the string method of making an ellipse. A loop of string is stretched around two pins by a moving pencil. In **strell**, the pin positions are [50 0] and [-50 0]. The string length is distance [50 0] + distance [-50 0] + 100 (the distance between [50 0] and [-50 0], which, because it never changes, can be ignored.) The line **forward 1** repeated gives an outward movement on each new heading, which simulates the outward pull of the pencil. The outward pull is restrained by the length of the string, simulated by the action of the test line. This allows a correct position to be marked when :d is slightly greater than the string length, and moves to the next heading (increment on :a of 5 degrees) for the process to be repeated. In this example the string length is set at 240 and the pins 100 apart. The **forward 40** just speeds things up.




STRING LOOP METHOD OF DRAWING AN ELLIPSE

## 17. Conics (Directrix and Focus Ratio)

The directrix is set at [-140 -+y] and the adjacent focus at [-100 0]. Initially the turtle moves away from the focus until the test line condition is met, when it marks that point. It then returns to the focus and moves outward again on a new heading (old + 5). The condition for the test line is that distance to directrix / distance to focus = 1.2 (that is approximately equal, within ±2). Note that for an ellipse, the ratio must be greater than one.

```
to conell :n
forward 2
if 1.2 * :n > distance sentence -140
   ycor [pd setpos pos pu
setpos [-100 0] make "n 0 right 5]
conell :n + 2
end
```

Then type

```
cg
pu
left 90
setpos [-100 0]
conell 0
```

## 18. Cartesian Formula

This final procedure uses the formula m/M*sqrt($M^x$), where M is the major axis and m is the minor.

```
to formell :mi :ma :x :n
if or :x = :ma :x = minus :ma [make
   "n minus :n]
setpos sentence :x :n * (:mi / :ma)
   * sqrt ((:ma * :ma) - (:x * :x))
formell :mi :ma :x + :n :n
end
```

Then type

```
cg
pu
setpos [0 50]
pd
formell 50 100 0 1
```

Ken Large
84 Shepwell Green
Willenhall
West Midlands, England WV13 2QJ

# Is Programming Obsolete?

by Douglas H. Clements and Julie S. Meredith

Is programming obsolete? Why or why not? Answer before you read on.

This provocative question was the theme of a recent session at the American Educational Research Association meeting. Three speakers presented three different perspectives. We'll provide a brief summary of each.

## Programming as an Extended Literacy

For Andrea diSessa, programming represents the "best ever" representational support for cognitive activities. Programming gives students general, flexible control of dynamic interactive processes. He first takes us on brief historical tour of programming languages.

The early 1980s, he says, were the *Fun Years*. Logo was proposed as a "Math Land," and lots of people jumped on the "train thinking with Logo" bandwagon. In 1984, there was a backlash. Some critics wrote of the terrible consequences of computer use.[1] The next period diSessa termed the *Boring Years*. We wandered about "Application Land," and pigeon-holed computer programming as belonging solely to computer science. *HyperCard* was the one bright star. Finally, with the introduction of powerful machines, emerged *Our Time*, so named because computers such as the recent Macs can run Boxer. Boxer is a programming language that diSessa has been working on for years as a successor to Logo. We shall devote an entire column to it in the future, but, in a phrase, Boxer has boxes inside boxes. Programmers store both data and programs in boxes. In this way, Boxer can control and present "nested complexity."

How do we decide among programming languages? diSessa offers a simple formula:

$$\text{utility quotient} = \frac{\text{value}}{\text{effort}}$$

Unsurprisingly, diSessa believes that Boxer has the highest utility quotient. According to him, Boxer is much more powerful than present programming languages.

diSessa showed some impressive projects with multiple symmetries that students constructed. One student constructed an eight-level adventure game. He used a graduate student's graphing tool, pulling out only the essence of it for his game. Boxer also allows modeling of biological processes. One student produced a biological tool kit.

## New Paradigms for Programming

They had to drag Andrea from the microphone to make room for Mitchell Resnick's talk. Resnick began with *Lego*-Logo. He and his colleague put computers into Lego constructions. They felt that it was a "turn-off" to have "programming" in the name of the activity. Too many people view programming as difficult. Resnick admitted that programming is hard in some ways. The real reason that programming has not "taken off" as some predicted, however, is that people have done to programming what they've done to other subjects. They have disconnected it from:

- other domains of knowledge
- students' interests
- other uses of computers

If this is the way it goes, Resnick claims programming should become obsolete. He told a story about a child in the Hennigan schools who came to his new computer class. When the teacher found out he and many of his peers had done Logo previously, she said, "We'll have to find something else for you to do." She didn't see that because they now knew something, they could really do interesting and important explorations.[2]

Resnick says that we need "Whole Programming"—integrated programming activities that children care about. The dominant early view of programming was "train thinking." This led to a backlash.[3] Another view is to use programming to design things, that is, as an expressive medium.

Resnick cites Brian Harvey's analogy: Consider if teachers viewed other classes as they presently view programming classes. Art instruction would consist of endless practice at drawing straight lines and perspective cubes. Such instruction would chase away any potential artists. Similarly, students of writing would concentrate only upon *Webster's Unabridged Dictionary*.

Instead, students should use programming media such as *Lego*-Logo to express themselves. One group of children, for example, built an entire chocolate factory. Also, students might use Logo as an expressive medium for building video games. In this view, children construct things that they care about and are meaningful to them. They can and do learn mathematical ideas through this type of programming activity.

But present-day tools are limited in certain ways. As the saying goes, "Simple things should be simple and complex things should be possible." Kids intuitively think that programming languages should allow several things to be controlled at the same time. Computer scientists, however, believe that this is a difficult computer topic.

Resnick showed LCSI's new version of Logo. Kids use programming, parallel processing, and paint. They might change the turtle's shape to a bird and paint a cage around it. They create a button and click on it to make a bird move around. They teach the bird to flap its wings. Then they teach it to change direction whenever it hits the side of its cage.

A program created by other students controlled hundreds of objects. The students simulated an ant colony. Later, they wanted to change it to simulate a traffic jam. This was surprisingly accessible. Resnick noted that you can't go out and buy an application that allows you to do this.

## Programming by Direct Manipulation

Segue to David Smith from Apple Computer. Smith discussed a totally different way to approach programming: direct manipulation. He urged that "we need to create programmable video games."

He claimed that programming is not obsolete, but our *approach* to it is. The language is the problem. The solution is to apply good user-interface principles to the process of programming.

Smith discussed three "language-less" approaches to programming. These include programming by

- direction manipulation
- demonstration
- visual rewrite rules

*Programming by direction manipulation* involves directly manipulating pieces of what will be a program. Some programs in *HyperCard*, for example, are created in very much this way. The programmer drags buttons, draws with graphics tools, and so forth.

*Programming by demonstration* involves instructing the computer to "watch what I do." Then one performs the desired actions manually. The computer remembers the actions and can run them later, even on new inputs. People who use powerful macro programs recognize this feature. There are problems, of course, such as generalizing actions to new situations.

*Visual rewrite rules* are a sort of hybrid of these two. A rule is created by

- capturing an area of the screen to the left side
- duplicating it to produce an identical right side
- editing the right side to produce the desired result

A program capturing the edits is built up by demonstration.

For example, you might make a Pacman with an open mouth move and change its shape to a closed mouth (and vice versa). If the Pacman is next to a wall, just close (or open) the mouth by changing the shape. Other rules are similar:

- If the Pacman is next to this object, eat it.
- If there is a monster near and a space is open, move away

And so on.

Smith concludes that people want to program. And until people can program computers, they will have access to only a fraction of the potential power in them. We must, however, change the process of programming before people will be able to program. We can best change this process by applying good-user-interface principles to it.[4]

## Overarching Themes

In general, the presenters agreed on several points:

- Programming has value as a representational mode.
- Programming can be used as a tool throughout the curriculum.
- Types of programming have to be extended, and these activities have to be further extended into the social domain.
- Programming may be hard, but it's worthwhile because it allows us to do interesting things.
- People—all people—can enjoy the creative production and intellectual challenge of programming.
- Programming got a bad name. Possibly we should rename it.[5] We must work practically so people re-see what programming can do.

## Final Comments

- These talks reflected a negative trend. Is this a function of changing knowledge of, or interest in, programming? Maybe. It is more likely that it represents the *inevitable* backlash toward virtually any educational innovation in our country.
- The importance of language in programming and communication must not be overlooked. Cave people could draw. Why did oral and written language develop as the major means of communication?
- Given these criticisms, is it valid to infer that what teachers and students did on 4K or 64K

machines was not educationally interesting or plausible?

## Footnotes

1. It is interesting, but sad, that exaggerated positive claims, though filled with hope, appear so much more amenable to criticism than overzealous negative opinions. The particular issue of a journal diSessa discussed included several authors' opinions that were overstated, later proven incorrect by research, and even ridiculous (using computers would turn us into a "nation of psychopaths").
2. A previous column, "Logo or 'Logo-like': The great debate" [*Logo Exchange, 9*(1), 33-34], makes a related point: "Teachers report having insufficient time to teach even one or two powerful tools. And what students need is a thinking tool, not just a computer tool"; see also our forthcoming article, Clements, D. H., & Meredith, J. S. (in press). My turn: A talk with the Logo turtle. *Arithmetic Teacher.*
3. Don't you just *love* lots of footnotes? As shown in previous columns (and similar to footnote 2), this backlash was exaggerated and ignored positive results that *were* reported.
4. But see an alternate view that emphasizes the importance of a programming *language,* in Clements, D. H. (1991). Logo: Search and research. A new vision of Logo in the secondary school. *Logo Exchange, 9*(6), 24-29.
5. I believe it's a wrong-headed idea to "disguise" programming. We should invite others into our circle, not try to pretend we're in a different one.

Support for the preparation of this material was partially provided by the National Science Foundation under Grants No. MDR-8954664 and MDR-9050210. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Douglas H. Clements, associate professor at the State University of New York at Buffalo, has studied the use of Logo environments in developing children's creative, mathematics, metacognitive, problem-solving, and social abilities. He is currently working with several colleagues on a second NSF-funded project, Investigations in Number, Data, and Space, to develop a full K-6 mathematics curriculum featuring Logo.

Julie S. Meredith is a mathematics education doctoral student at the State University of New York at Buffalo. She has taught secondary mathematics and computer science, gifted math at the middle school level, and mathematics methods courses. Along with Clements, she is currently designing and programming a new version of Logo for the NSF-funded Investigations project.

Douglas H. Clements and Julie Meredith
State University of New York at Buffalo
Department of Learning and Instruction
593 Baldy Hall
Buffalo, NY 14260

CIS: 76136,2027 BITNET:
CLEMENTS@UBVMS.CC.BUFFALO.EDU

# Global Logo Comments

**by Dennis Harper**

## Logo Exchange Continental Editors


| Africa | Asia | Australia | Europe | Latin America |
|---|---|---|---|---|
| Fatimata Seye Sylla | Marie Tada | Anne McDougall | Harry Pinxteren | Jose Valente |
| UNESCO/BREDA | St. Mary's Int. Sch. | Monash Univ. | Logo Cent. Nederland | NIED |
| BP 3311 Dakar | 6-19 Seta 1-Chome | 6 Riverside Dr. | P.O. Box 1408 | UNICAMP |
| Senegal, West Africa | Setagaya-Ku | East Kew | BK Nijmegen 6501 | 13082 Campinas |
| | Tokyo 158, Japan | Victoria 3120 | Netherlands | Sao Paulo, Brazil |
| | | Australia | | |


## Logo at the 10th International Conference on Technology and Education

One would think that an international conference held at M.I.T. would produce numerous papers and presentations on Logo. Well, surprise, it did! Not since the series of Logo conferences in 1984 and 1985 has M.I.T. seen so many Logo papers—and the majority of those were delivered by educators from outside the U.S. In addition, many past and present *Logo Exchange* authors were in attendance, including international editors Anne McDougall, José Valente, and Hilel Weintraub.

This first installment of Global Logo Comments for Volume 12 of *Logo Exchange* will summarize Logo papers contained in the 1,366 pages of proceedings and notes taken at Logo sessions. Logo research outside North America continues at a rapid pace both in quantity and quality.

### Implementing Logo: Cross-Cultural Barriers (Frant & Tornaghi)

The objective of this study was to identify common factors affecting the implementation of Logo in the elementary schools of Brazil and New York. The researchers once again found that teacher preparation is the most important factor in determining how successful a Logo environment becomes. Language and cultural background had no effect.

### Computer Education: An International Perspective (O'Lander)

Richard O'Lander identified the computer languages being used in the high schools of Belgium, Denmark, Ireland, Germany, and France. According to the author, only Germany gives secondary schools the choice of using either BASIC, Pascal, Logo, or Elan. Belgium high school programming courses use BASIC, COBOL, PL/I, and RPG. Denmark uses BASIC and COMAL. Ireland uses BASIC and Pascal, while France discourages programming in secondary schools and emphasizes CAI and using the computer as a learning tool.

### The Instruction Trap (Fleming)

This Australian educator described how teachers at her school were moving from the instruction of students to construction, where the learning is student- and teacher-centered. Her research has shown that the implementation of Logo in schools has changed a teacher-centered environment into a learner-centered environment.

### Understanding Angle Notion in a Logo Context (Mantoan, Barrella, & Prado)

This Brazilian study analyzed how children took advantage of previous knowledge to find and create means to solve problems involving the notion of angle. The researchers found that Logo allowed students to act by themselves in such a way that solutions to problems were not imposed on them. The study characterizes the instructor as an educational designer who adjusts interventions according to the learning of the students, taking into account the route of each individual in the classroom.

### Logo: What Does This Language Mean? (Barrella)

The fundamental hypothesis of this Brazilian study was that natural language supports the acquisition of programming languages. The author conclusions are eloquently stated as follows:

> I believe that language serves communication and also structures and organizes thinking. What frequently we observe in natural lan-

guage is its "external face"—what we hear and speak—its materialization in the gesture form, verbal or written. However, the language has one "internal face" responsible for the organization of our thinking. In other words, I adopt a perspective about language and thinking that, though it established them as different things, conceives them in a continuous relationship: one's functioning influencing the other. The process by which we attribute a meaning to what we hear represents the interfaced between the two processes. Someone can say: "Beautiful city!" with the intention of censuring a confused or dirty city, for example. The meaning isn't only carried by each one of the words that form the enunciation, but also by the situation that is occurring. The complex relationship between what is said and what is intended requires the participation—by the person who interprets—of linguistic and cognitive phenomena. Similarly, a programming written in Logo involves other phenomena besides those we can directly "translate."

## The Logo Environment as a Mediation Tool for Processes of Cognitive Construction in the Context of Preschool (Ripper)

This research relies heavily on the theories of Vygotsky to study the mediational function and the relationships between sign and tool in a Logo-based environment along with its influence on the learning process of preschool children. Results indicated that as four-year-olds began to draw with the turtle, they discovered the syntactic rules of Logo. The reading of the error messages in the beginning is contextually understood as something that has gone wrong:

> The majority of the children showed a low resistance to syntactic mistakes: confronted with an error message, the first reaction was to clean the screen without trying to identify the error. Thus, we see that the global significance of the error is understood, but the discomfort prevents a more constructive action. After a few sessions, often with the mediation of the teacher or another child, the child begins a reflective process: another child points out, "It does not understand without a spacing."

## Construction and Interfacement of Devices With Computers for Educational Purposes (D'Abreu)

This study investigated the use of Logo to control something in the external world (outside the computer). LEGO-Logo was used by a high school physics class in 1991. The students constructed speedometers, cars, and robots to demonstrate physics principles. Materials generated in this LEGO-Logo environment will be disseminated to other Brazilian schools.

## Tridemensional Logo: Art or/and Geometry? (Miskulin)

Tridemensional Logo, proposed and implemented by H. Reggini, is an extension of turtle geometry in which the movements of the turtle are not limited to a plane. This study investigated ways in which tridemensional Logo could be used to explore spatial geometry. The Brazilian researcher found that in spite of limitations imposed by a flat screen, tridemensional Logo did provide a way to practice with a shape's spatial representations.

## A Logo-Based Instruction System for the Development of General Thinking Skills (Liekens, Olivié, De Corte, & Verschaffel)

This University of Leuven project studied alternative Logo classroom environments. The claim that students discover general thinking skills on their own is disputed in this study. A more effective Logo environment was sought. This new method would give more guidance to the pupil rather than relying on self-discovery. The new method involved four main components: the executor, the editor, the planner, and the coach. This system is still being transported to other schools in Europe to test whether it has an impact on the development of general thinking skills of pupils.

## An Environment for Learning Control Theory: Technic Control Software for LEGO-Technic Systems (Krumholtz, Orion, Harel, & Krumholtz)

This research describes how the Israeli Logo Center designed software called TCS—Technic Control Software. The target populations of this study included:

- Children too young to learn programming
- Students learning technology who, in the given limited time, were not able to learn programming
- Special education students who were mentally or physically disabled
- Disadvantaged students with learning difficulties

Intensive observations revealed that almost all students learned to use TCS after approximately half an hour of practice and mastered it after two hours of

practicing. Analyzing the learners' final reports revealed that they properly applied some basic computer control concepts of open-loop control.

## References


Barrella, Fernanda Maria Freire. *Logo: What Does This Language Mean?* Proceedings of the 10th International Conference on Technology and Education, March, 1993, p. 984.

D'Abreu, João Vilhete Viegas. *Construction and Interfacement of Devices With Computers for Educational Purposes.* Proceedings of the 10th International Conference on Technology and Education, March, 1993, p. 1012.

Fleming, Di. *The Instruction Trap.* Proceedings of the 10th International Conference on Technology and Education, March, 1993, p. 516.

Frant, Janete B., & Tornaghi, Alberto. *Implementing Logo: Cross Cultural Barriers.* Proceedings of the 10th International Conference on Technology and Education, March, 1993, p. 241.

Krumholtz, Nira; Orion, Eden; Harel, Henry; & Krumholtz, Eran. *An Environment for Learning Control Theory: Technic Control Software for LEGO-Technic Systems,* Proceedings of the 10th International Conference on Technology and Education, March, 1993, p. 1144.

Liekens, Olivié, De Corte, & Verschaffel. *A Logo-Based Instruction System for the Development of General Thinking Skills.* Proceedings of the 10th International Conference on Technology and Education, March, 1993.

Mantoan, Maria Tereza Eglér; Barrella, Fernanda M. Freire; & Prado, Maria Elisabette B. Brito. *Understanding Angle Notion in a Logo Context.* Proceedings of the 10th International Conference on Technology and Education, March, 1993, p. 981.

Miskulin, Rosana Giaretta Sguerra. *Tridemensional Logo: Art or/and Geometry?* Proceedings of the 10th International Conference on Technology and Education, March, 1993, p. 1016.

O'Lander, Richard. *Computer Education: An International Perspective.* Proceedings of the 10th International Conference on Technology and Education, March, 1993, p. 349.

Ripper, Afira. *The Logo Environment as a Mediation Tool for Processes of Cognitive Construction in the Context of Preschool.* Proceedings of the 10th International Conference on Technology and Education, March, 1993, p. 1010.



Dennis Harper
1113 Legion Way SE
Olympia, WA 98501


## Logo PLUS for the Macintosh Now Available from Terrapin

Terrapin Software, Inc., announces the release of Logo PLUS for the Macintosh, V. 2.0. This major revision of Terrapin Logo for the Macintosh offers 40 new primitives for color, music, serial communication, and text cursor handling, as well as new tools and games, a text mode for graphics windows, rotating shapes, and all-new documentation. Logo PLUS takes full advantage of each computer's color capabilities, with up to 256 colors available at a time.

To explore the new color features, the user can select from 16 or 256 pen and background colors by using the mouse or by using commands. In addition, the user can create new colors.

The new shape editor allows images to be flipped and rotated, and the resulting shapes rotation may be locked at any of four orientations. The turtle's shape may match its pencolor or assume any other color.

New tools and games include a word game, card game, multiple-turtle action game, emulator files for other versions of Logo, and a conversion file for Logo PLUS/Apple. Files created with earlier versions of Terrapin Logo for the Macintosh are fully compatible with Logo PLUS/Macintosh.

The complete package includes a new Logo PLUS Guide that introduces Logo programming through easy-to-understand lessons and activities. The revised Reference Manual explains each primitive and includes hundreds of examples. The updated Quick Reference Guide provides a brief listing of each primitive, and the new QuickStart booklet gets users up and running in minutes with easy instructions and activities using graphics, music, multiple turtles, and shapes.

Logo PLUS runs on Macintosh computers using System 6.0.3 and higher and 1 MB of memory, or System 7 with 2 MB of memory recommended. Upgrades from Terrapin Logo for the Macintosh are available, as well as Logo PLUS/Macintosh Single Versions, Lab Packs, Take-Home Paks, and Site Licenses. Call Terrapin Software at 800/972-8200 for more information.

# "Awesome"
# "Way cool."
# "Slammin'!"
# "Totally there."
# "Swinging."
# "Wicked good."

**(And these are just the teachers' comments.)**

The fact is, whenever we show our three new educational software products to teachers and curriculum coordinators, they get as excited as kids.

And for good reason.

You see, our line of learning software for Macintosh® computers gives teachers of grades 4-8 a unique way of motivating their students.

For one thing, **MicroWorlds**™ products are specifically designed for the classroom. Their flexibility lets students with all different learning styles use what they know to tackle new learning experiences.

What's more, the **MicroWorlds** packages were designed by LCSI, the company known for its award-winning educational products.

Take **MicroWorlds Math Links**™ - it doesn't camouflage math as some space game. Instead, it lets you link math to art, science, and social studies. Students don't just study math, they think mathematically, using math to develop projects ranging from kaleidoscopes to Navaho textile patterns.

With **MicroWorlds Language Art**™ you'll encourage students to explore words and images. Write text in any shape, color or direction. Add effects such as scrolling text, animation. Projects, including Visual Poetry, Ads, Haiku, help you assist students in developing writing skills.

**MicroWorlds Project Builder**™ gives you the tools to develop a problem-solving, creative-thinking, learning culture across the curriculum. And features like text, drawing tools, animation, and music give students the tools to create anything from simple ecosystems to dynamic maps.

Plus there's more: Each of these products is offered under LCSI's well-known site/network license - the most flexible policy available to schools today.

So for information or a **free** demo disk, call us today at **1-800-321-5646**.

We think, like, you'll be blown away.

LCSI®

MicroWorlds

International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Order Desk: 800/336-5191 Fax: 503/346-5890