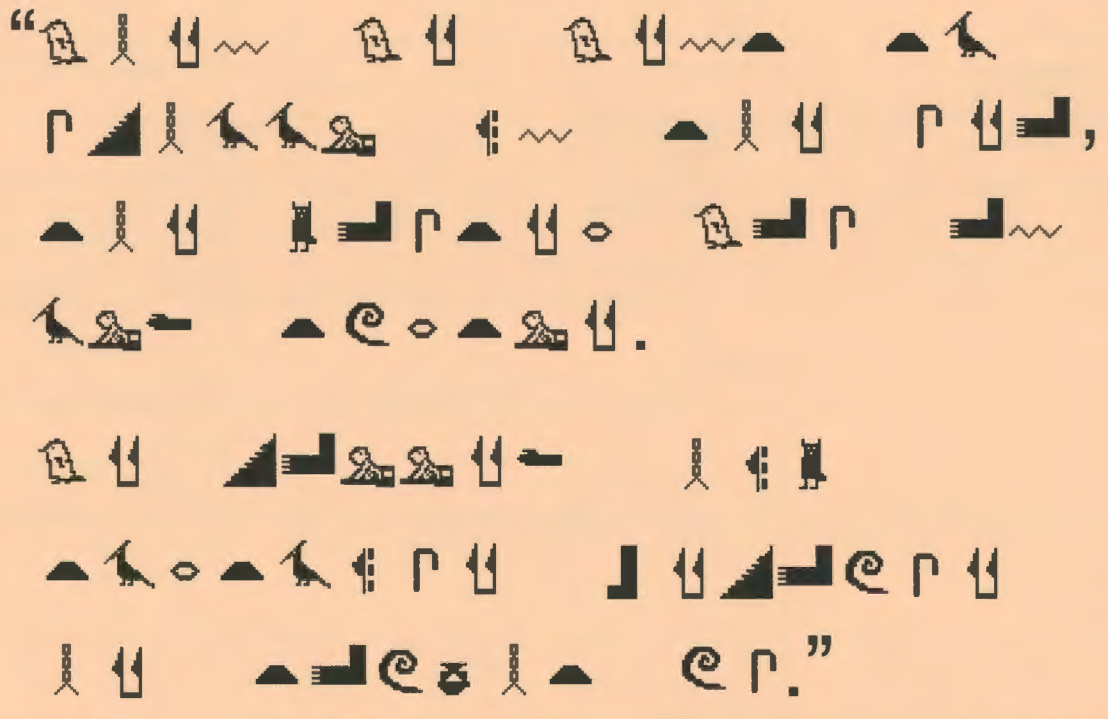
LOGO EXCHANGE

Volume 15 Number 3 Spring 1997

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



— said the Mock Turtle in Lewis Carroll's *Alice's Adventures in Wonderland*

The keys to unlocking this code (and writing your own messages in Logo) are inside...

INTERNATIONAL SOCIETY FOR TECHNOLOGY IN EDUCATION ISTE

In this issue:

ASCII Artwork

Nurturing a Logo Culture

Merging Technologies

Dartboards, Roulette Wheels, and Spinners

OzLogo National Conference Report

Logo Hieroglyphics

Math Fun: Get Rich Quick!

Fun on the Text Screen

Editorial Publisher
International Society for Technology in Education

Editor-in-Chief
Dorothy M. Fitch

Copy Editor
Jennifer Roland

Founding Editor
Tom Lough

International Editor
Dennis Harper

SIGLogo Officers
Chuck Friesen, President
Steve Sesko, Vice-President
Hope Chafiian, Secretary/Treasurer
Dorothy Fitch, Editor

Production
Kerry Lutz

Director of Advertising Services
Judy Stickney

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.

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 may be submitted for review if electronic copies are supplied upon acceptance.

Send surface mail to:

Dorothy M. Fitch
3 Derby Road
Derry, NH 03038

Send electronic mail to:

71760.366@compuserve.com

Deadlines

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

Volume 16, Number 1	Mar. 1, 1997
Volume 16, Number 2	June 1, 1997
Volume 16, Number 3	Oct. 1, 1997
Volume 16, Number 4	Jan. 1, 1998

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®*.

ISTE Members may join SIG/Logo for $24.00. Dues include a subscription to *Logo Exchange*. Non ISTE Member subscription rate is $34.00. Add $10 for mailing outside the USA. Send membership dues to ISTE. Add $4.00 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 director of advertising services 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, 480 Charnelton St., Eugene, OR 97401-2626 USA. Periodicals postage paid at Eugene, OR. USPS# 660-130. ISTE is a nonprofit organization with its main offices housed at the University of Oregon. ISSN# 0888-6970

Logo Exchange

Volume 15 Number 3 Journal of the ISTE Special Interest Group for Logo-Using Educators Spring 1997

Contents

From the Editor's Mailbag .... *Dorothy M. Fitch* 2

Quarterly Quantum
A Matter of Proportion .... *Tom Lough* 3

ASCII Artwork .... *Jim Muller* 5

Dartboards, Roulette Wheels, and Spinners .... *William J. Spezeski* 8

Nurturing a Logo Culture From the Ground Up .... *Nan Youngerman* 12

Merging Technologies: LEGO TC logo and the Color Quick Cam .... *Orlando Mihich* 15

Hieroglyphics + Logo = Fun .... *Robert Macdonald* 18

More Fun on the Text Screen .... *Jim Muller* 25

Math Activity: Get Rich Quick!? .... *Ihor Charischak* 28

Learning in Logo Microworlds: OzLogo National Conference 1996 .... *Jenny Betts* 32

From the Editor's Mailbag

by Dorothy M. Fitch

This is the kind of mail I love to receive. I reprint it here with the author's permission.

Hi Dorothy,

Today when I left school, I was so tired that I promised myself to go straight to bed for a nap. Well, I couldn't. Two of my students, Patrick and Sam, wrote down their work, as I asked them to do, and I was going over and over it thinking that it was really great. When I got home, just thinking about the work, their creativity, dedication, and enthusiasm, kept me from napping, or reading, or just listening to music. Instead, here I am writing to you.

I have many very special students. Those who are doing the Hudson River research studies are special; you may have noticed the long list of names with the article you published. So are Patrick and Sam. They are only 13 years old, but they already act as responsible, mature gentlemen. Patrick has a special inclination for fixing and operating Mac equipment. He was able to rescue three Macs and is taking care of all the hard drive problems. Yesterday, I asked them to write how their Lego model works and what their procedures do. Today, I only suggested to use red text to explain the procedures and to enclose the explanations in two <<...>>. That's it. They did it all by themselves during lunch time, all while I was teaching my fifth-grade class. When I saw the printed procedures and the explanations I asked for, I could not believe it. I must admit I could not have done a better job myself.

I thought that if you are looking for some material for a future *LX*, you may want to consider the work of these two students.

I am enclosing some pictures for you to see of the "machine" and the students.

Thanks,

Orlando Mihich

Such a wonderful example of both creative and resourceful students and a teacher who encourages them and truly appreciates their achievements.

I encourage all of you to take a good look at your students. Ask yourself the following questions:

- What are they doing?
- What are they capable of doing?
- How can I encourage them?

I'll bet they will brighten your day as much as this letter brightened mine.

And while you are at it, have them write up what they are doing with Logo and send it to me. No *LX* author is too young; no article too short. It need not be fancy, and it doesn't have to sound like it was written by an expert.

We can all learn from Logo enthusiasts of all ages. Help your students share what they are doing with the rest of the world!

And you—their teachers, mentors, parents, and adults of any description—let me hear from you as well! ▲

Dorothy M. Fitch
LX Editor
3 Derby Road; Derry, NH 03038
E-mail: 71760.366@compuserve.com
Ph.: 603/425-2010
Fax: 603/425-6487

P.S. I hope you enjoyed the Lewis Carroll quotation on the front cover. If you missed the word play, read it again, out loud!

Quarterly Quantum: A Matter of Proportion

by Tom Lough

The other day I created a procedure just for fun that could draw a square to any scale I wanted. But it had a little wrinkle I liked. Here's what I mean.

First, I had to set up a starting distance.

```
make "distance 100
```

But, after that, I could use my square procedure like this. To draw my first square with sides 100 turtle steps long, I typed:

```
square 1.0
```

Then to draw a square half as big, I used a scale factor of 0.5 and typed:

```
square 0.5
```

But if I again typed

```
square 0.5
```

it drew a square half again as big, instead of drawing a square half as big again.

??? How's that again, you say???

This procedure has a sort of "memory." It keeps track of how big the square is and then applies the scale factor to that size. The only way for you to draw a square the same size a second time (regardless of the size) is to type:

```
square 1.0
```

And if you type something like the following,

```
repeat 6 [square 0.9 rt 60]
```

you get a series of successively smaller squares.

Such a procedure is lots of fun to play with. But it can also provoke some interesting thinking. For example, how can you make your squares increase in size?

One way is to use a scale factor greater than 1, as with:

```
square 1.2
```

But it may be surprisingly difficult to return your square to its original size, especially if you have drawn several squares with different scale factors.

If necessary, you can always return to the original size (100 or any other size you choose) by the brute force method of

```
make "distance 100
```

Sooner or later, someone might suggest a problem such as this:

> I made a square by typing **square 1**. Then I made a square a little smaller by typing **square 0.7**. What must I type next to draw a square of the original size once more?

Let me stop there for now, so that you can either have some fun developing your own square procedure for this or use the procedure furnished at the end of this column to play around with this idea.

You know, none of us is ever too old to play around with ideas. That is why Logo has been such a long-term love of mine. Every time I use it, I learn something new.

My hope for you is that you are continually discovering new and interesting ideas through Logo. Because if you are doing this, it is certain that your students will be doing this also. And it doesn't get much better than that.

FD 100! ▲

Tom Lough
Founding Editor
Box 394
Simsbury, CT 06070

P.S. Oh, yes! I almost forgot! Here is a listing of the procedure I used with this column.

```
to square :scale
make "distance :distance * :scale
repeat 4 [fd :distance rt 90]
end
```

Be sure to give Logo an initial value for the **:distance** variable (such as **make "distance 100**). After that, you're on your own!

ASCII Artwork

by Jim Muller

A question that invariably arises in the life of any Logo fan(atic) is "Why does all the fun occur on the graphics page, screen, or window?" The text screen seems to get left out.

You may have seen the cute little characters added to e-mail and other messages—characters that look like a happy face, (:>), or a sad face, (:>(. There are many of these little additions. It's a way to add expression to what can sometimes be dull online text.

Why not add something like this to your procedures?

ASCII characters

Logo can also display alphabetic and numeric characters using the primitives **char** or **ascii** (American Standard Code for Information Interchange). We'll use these codes, rather than print the characters directly, because some versions of Logo complain if you try to print punctuation marks and other special characters. Try **print** "-, for example.

```
show char 65 displays the letter A.
show char 67 displays the letter C.
show char 49 displays the number 1.
show char 32 displays a space.
```

Char followed by a code number displays the letter, number, or punctuation mark for the specific ASCII code number you type. If you ever want to find out the ASCII code for a character, type:

```
show ascii <character>
```

For example:

```
show ascii "A
65
show ascii "a
97
```

Upper- and lowercase letters each have different codes.

Adding a signature

And, yes, you can add these touches to your procedures! For example, you can display your signature every time you create a picture. This one is written using PC Logo for Windows.

```
to signature
cleartext
type [graphics by <your name>]
     ; add your name
type char 32   ; adds a space
type char 40   ; adds the (
type char 58   ; adds the :
type char 62   ; adds the >
pr char 41      ; adds the )
end
```

When you run this procedure,

```
GRAPHICS BY <your name> (:>)
```

is printed in the PC Logo Listener window.

Some versions of Logo sometimes handle the same primitives a bit differently. For example, try this procedure in PC Logo for Windows.

```
to signature
(type [graphics by jim] char 32 char 40 char
  58 char 62)
pr char 41
end
```

The result is:

```
GRAPHICS BY JIM   ( : >)
```

PC Logo adds some extra spaces. However, MSW Logo prints the same procedure as:

```
GRAPHICS BY JIM (:>)
```

The spaces are gone.

Adding the **print** command is a matter of choice. **Type** and **print** do essentially the same thing, except that **type** does not add a carriage return at the end. **Type** prints the string of characters all on one line until you get to the end. **Print** adds a carriage return, which sends the cursor to the next line.

This comes in handy at times, especially if you want to start printing some fancy ASCII artwork. For example, here is the logo of a popular movie from the 1980s.

```
               0000 00000
              0000000000
             00000000^0
            00 00 0^^^^^
          ^00 *  * 00   ^^^
        ^^^000 0& 0000    ^^^
       ^^^000000000000    ^^^^
      ^^^ 000000000000   ^^^^^^
      ^^^ 0000   00000  ^^^ ^^^
     ^^^  00000   000  ^^^   ^^^
     ^^^   000000000  ^^^    ^^^
    ^^^     000000000^^^      ^^^
    ^^^       000000^^^       ^^^
    ^^^     0000000^^^00000   ^^^
   000^^  0000000^^^000000000^^
  0 0^^^000000000^^^00000000000^0
   0000000000000^^^000000000000000
  0000000000000^^^000000000000000000
     000000000^^^00000000000000
       ^^^   ^^^0000000000^^^
        ^^^ ^^^0000000000^^^
           ^^^^^^^^^^^^^^
              ^^^^^^^^
```

Here are the essential procedures for creating this text art. A few lines have been finished—the rest is up to you!

```
to movie
line1 line2 line3 line4
line5 line6 line7 line8
line9 line10 line11 line12
line13 line14 line15 line16
line17 line18 line19 line20
line21 line23 line24
wait 50 title
end

to tc :x
type char :x
end

to tr :r
pr char :r
end

to line1
ct repeat 15 [tc 32]
repeat 4 [tc 48] tc 32
repeat 4 [tc 48] tr 48
end

to line2
repeat 14 [tc 32]
repeat 9 [tc 48] tr 48
end

to line3
repeat 13 [tc 32]
repeat 8 [tc 48]
sign 1 tr 48
end

to sign :n
repeat :n [tc 94]
end

to line4
repeat 12 [tc 32]
repeat 2 [repeat 2 [tc 48] tc 32]
tc 48 sign 4 tr 94
end
```

Animating words

The only thing missing now is some action.

If your version of Logo does not have a **setcursor** primitive (sometimes called **cursor**), use this:

```
to setcursor :x :y
repeat :x [print "]]
repeat :y [type char 32]
end

to signature
ct ts repeat 10 [tc 32]
tc 71 tc 114 tc 97 tc 112
tc 104 tc 105 tc 99
tc 115 tc 32 tc 66 tc 121
tc 32 tc 74 tc 105 tc 109
wait 100 flasher 1 1
end

to tc :c
type char :c
end
```

```
to flasher :x :y
ct ts setcursor :x :y
tc 71 tc 114 tc 97 tc 112
tc 104 tc 105 tc 99
tc 115 tc 32 tc 66 tc 121
tc 32 tc 74 tc 105 tc 109
make "x :x + 2
make "y :y + 1
if :x > 24 [make "x 1 make "y 1]
wait 3
flasher :x :y
end
```

Put your own name in this procedure and then run it. If you can't find a list of the ASCII codes, why not write a procedure that prints them out for you? For example:

```
to code :asc
if :asc > 127 [stop]
pr (se [ascii code number] :asc [=] char
  :asc)
code :asc + 1
end
```

What else can you do to dress this up? Why not add some color to the display? What about adding sound effects or music?

In this example, the signature moves from upper left to lower right. Why not add some random action to it? For example, make the text randomly move from upper right to lower left, from top to bottom in the center column of the screen, from upper left to lower right.

To make the display truly psychedelic, have the signature appear randomly on the screen, changing the background and text colors with each change.

Maybe the text screen doesn't have to take a back seat all the time. ▲

Jim Muller has had a lifelong interest in translating various technologies into understandable and persuasive programs. In 1981, Muller and his son organized the first Logo users group, the Young Peoples' Logo Association, which eventually grew into a worldwide 6,000 member organization. In 1985, the YPLA merged with CompuServe where it became The Logo Forum. Today, Muller is a computer training and marketing consultant in the Dallas/Fort Worth metroplex.
E-mail: 76703.3005@compuserve.com

N • E • C • C

POTLATCH

@NECC97.seattle.wa.edu

Don't miss your opportunity to experience one of the foremost K–16 educational technology conferences in the country!

Featuring the US' largest national ed tech exhibit hall!

Join your national and regional colleagues this summer at the eighteenth annual National Educational Computing Conference (NECC) for three days of top-notch sessions, networking opportunities, and internationally-known Keynote and Spotlight speakers—plus two days of preconference workshops, June 28 and 29.

Seattle is a breathtaking city, and we hope that in addition to enjoying all that the conference has to offer professionally, you'll take time out for a ferry ride, take in the view from the Space Needle, and dine at some of the finest restaurants and cafés in the Northwest.

For the latest information on attending or exhibiting, check us out on the web or call us toll-free today!

#AD002

June 30–July 2, 1997
Seattle, Washington

Hosted by
Western Washington University

Sponsored by
the National Educational Computing Association (NECA)

http://www.wce.wwu.edu/necc97

1.800.280.6218

Dartboards, Roulette Wheels, and Spinners

by William J. Spezeski

Creating a dartboard

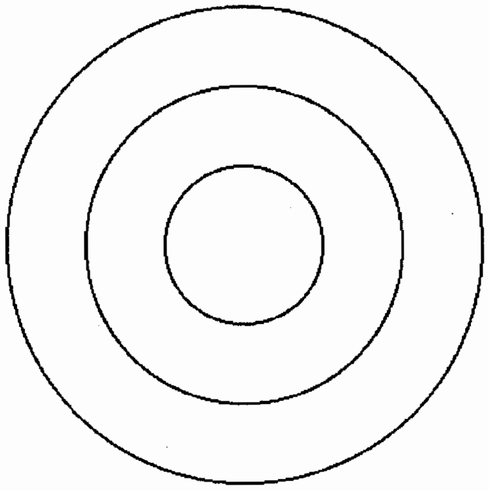
Creating dartboards, roulette wheels, and spinners are all related projects. The workhorse of their construction is a simple, versatile algorithm whose variations create sectors in circles, shade the sectors, aid in displaying numbers, and provide the necessary animation.

Making a dartboard can done using five basic steps: draw concentric circles, draw the sector lines, shade alternate outer sections, shade alternate middle sections, and, finally, display the numbers.

These procedures were developed using PC Logo for Windows.

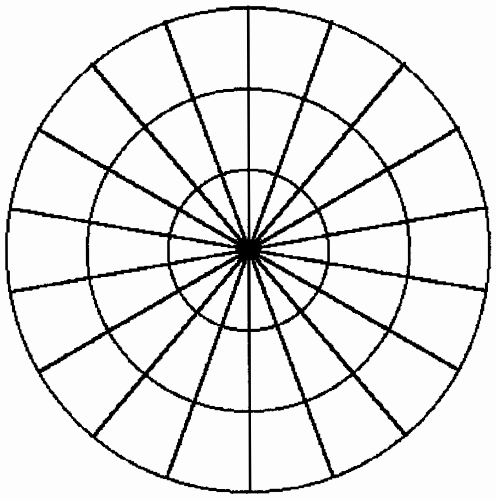
Step 1. Draw three concentric circles using:

```
stampoval 180 180
stampoval 120 120
stampoval  60  60
```



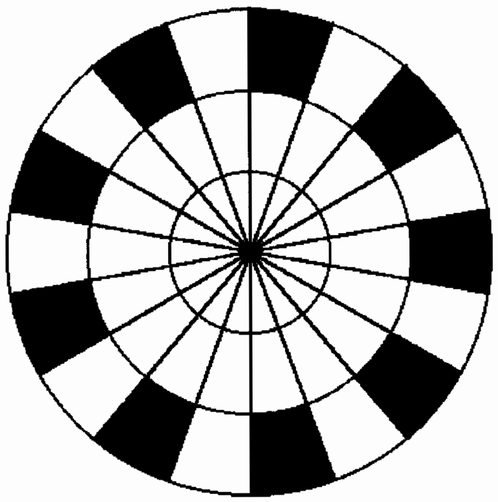
Step 2. Draw the sector lines using:

```
repeat 9 [fd 180 bk 360 fd 180 rt 20]
```



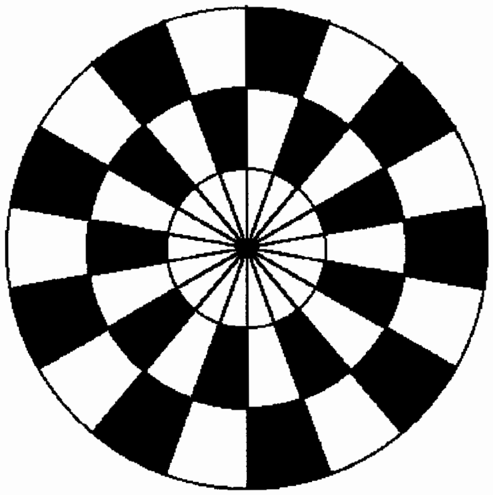
Step 3. Shade the outer sections of alternate sectors. This is a variation of the algorithm used in Step 2.

```
seth 10
repeat 9 [fd 150 pd fill pu bk 150 rt 40]
```



Step 4. Shade the middle sections of alternate sectors. This is also a variation of the algorithm used in Step 2.

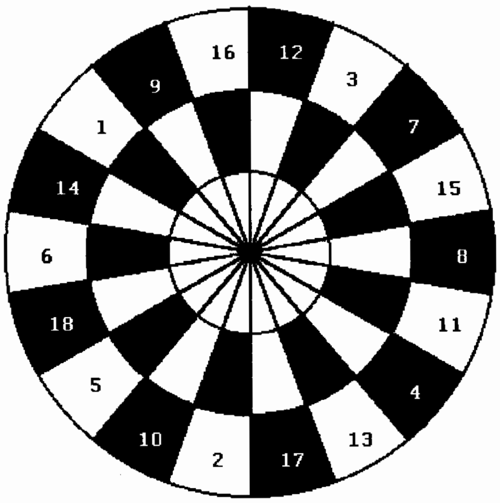
```
seth -10
repeat 9 [fd 80 pd fill pu bk 80 rt 40]
```



Step 5. Display the numbers.

This step is accomplished in two phases. Because we really have no way of knowing the coordinates of the final position of each number, we first approximate them with

```
seth 10 pu
repeat 18 [fd 150 pr list xcor ycor bk 150
  rt 20]
```



The 18 sets of trial coordinates will be printed in the Listener window. The algorithm here is another variation of the algorithm from Step 2. It simply prints out the coordinates of the center position in each outer section.

The second phase of this step is to adjust these approximations accordingly so that the numbers appear centered in their respective sections. Choose a color other than the background color or the shading color to display the numbers. One of many number arrangements can be displayed with the following instructions.

For your version of Logo, you may need to use **setpos** to position the turtle and a different command to stamp the number in the graphics window.

```
pu setxy [  20  155] pd turtletext 12
pu setxy [  70  135] pd turtletext  3
pu setxy [ 115  100] pd turtletext  7
pu setxy [ 135   55] pd turtletext 15
pu setxy [ 150    5] pd turtletext  8
pu setxy [ 135  -45] pd turtletext 11
pu setxy [ 115  -95] pd turtletext  4
pu setxy [  70 -130] pd turtletext 13
pu setxy [  20 -145] pd turtletext 17
pu setxy [ -30 -145] pd turtletext  2
pu setxy [ -85 -130] pd turtletext 10
pu setxy [-120  -90] pd turtletext  5
pu setxy [-150  -45] pd turtletext 18
pu setxy [-155    5] pd turtletext  6
pu setxy [-145   55] pd turtletext 14
pu setxy [-115  100] pd turtletext  1
pu setxy [ -75  130] pd turtletext  9
pu setxy [ -30  155] pd turtletext 16
```

Adding color and shading with patterns are also great enhancements. You might also want to experiment with different fonts and font sizes, and even different symbols, such as letters of the alphabet or symbols for the signs of the zodiac.

Once the dartboard is complete, it is not hard to make slight modifications in order to convert it into a roulette wheel or into a spinner.

Creating a roulette wheel

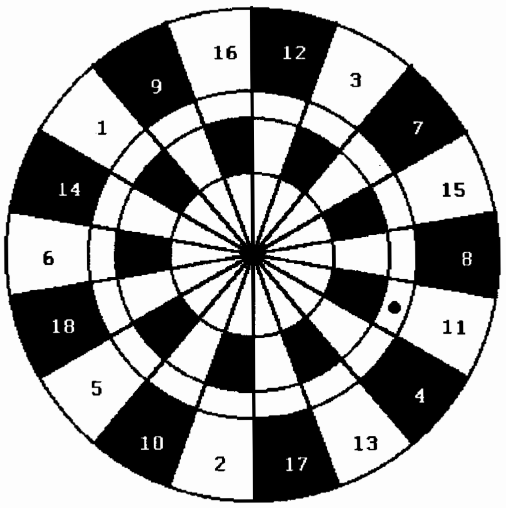
To convert the dartboard into a roulette wheel, we need to add a track for an animated ball to roll around on and then create the animation. That can be accomplished by modifying the five-step process for the dartboard as follows:

Step 1A. Modify Step 1. Draw a fourth circle to create a track for a rolling ball using

```
stampoval 100 100
```

Step 6. Animate a rolling ball. An animated ball can be created by continually drawing and erasing a ball (circle) in the appropriate positions on the screen. Because the motion of the ball is circular, this repeated process turns out to be another variation of the algorithm that was used in Step 2.

```
pu home seth 10
repeat 50 [fd 110 pd (stampoval 5 5 "true)
  wait 10 pe (stampoval 5 5 "true) pu bk 110
  rt 20]
fd 110 pd (stampoval 5 5 "true)
```



In the case of the roulette wheel, many other add-on features are possible. The ability to roll the ball through a random number of sectors is desirable, as is a prompt to allow the player to roll again without having to relaunch the program. The ability to place a bet could be added as well.

Creating a spinner

The dartboard can also be converted into a spinner by first carving out the middle portion of the board and then adding an animated spinning line. To do this, modify the five-step process for the dartboard as follows:

Step 1B. Remove the inner portion of the dartboard by eliminating the innermost circle in Step 1 **(stampoval 60 60)** and then replace Step 2 with

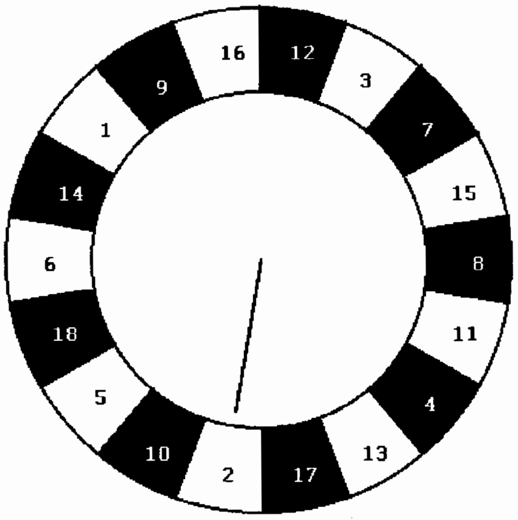
```
repeat 18 [pu fd 120 pd fd 60 pu bk 180 rt
  20]
```

This is just another variation of the algorithm that was originally used in Step 2. It subdivides the remaining annulus (area between the concentric circles). Because only the outer sections of the dartboard remain, only the outer sections require shading. Consequently, Step 4 should be eliminated.

Step 6. Animate a spinning line using

```
pu home seth 10
repeat 50 [pd fd 100 wait 20 pe bk 100 rt
  20]
```

This is similar in spirit to the instructions that animate the rolling ball. And this is yet another variation of the algorithm in Step 2. This is a common thread that is used in every construction step (except Step 1). That is quite a remarkable track record for an algorithm!



The same additional considerations for the roulette wheel also apply to the spinner. Moreover, note that it is possible to form the shape of the spinner from concentric polygons rather than concentric circles.

Once the basic construction of a dartboard is understood, there are many related things to make, such as a simple roulette wheel or a spinner. And, of course, roulette wheels and spinners can be expanded into interactive games. Have fun. Let your imagination run wild! ▲

William J. Spezeski is an associate professor in the Computer Science Department at North Adams State College in North Adams, Massachusetts. The department offers an elective course in problem solving that is built around Logo. The course features a variety of problems for computer solution emphasizing top-down design and modular programming. The text for the course is his book *Logo: Models and Methods for Problem Solving*, recently published by Harvard Associates, Inc.

Computer Science Department
North Adams State College
North Adams, MA 01247
Ph.: 413/662-5591
Fax: 413/662-5010
E-mail: wspezesk@nasc.mass.edu

Editor's Note:

Drawing these designs in Logo PLUS for the Macintosh is an interesting challenge because that version's **fill** command works differently from the one on which this article is based.

In Logo PLUS, the command **fillsh** takes as input a list of graphics commands. Logo draws the design specified by the list and then fills it.

Although you cannot fill areas of a design that is already on the screen, it is still possible to draw the dartboard design. The following procedures draw the basic shape, up to Step 3. The rest is left for you to finish!

```
to dboard
stampcircle 150
stampcircle 100
stampcircle  50
repeat 9 [fd 150 bk 300 fd 150 rt 20]
outersectors 0
pu home pd rt 20
innersectors 0
end

to outersectors :number
if :number > 8 [stop]
setheading :number * 40
fillsh [sector1]
pu home pd
outersectors :number + 1
end

to sector1
pu fd 100 pd
make "there pos
fd 50 rt 90 repeat 52 [fd 1 rt .4]
setheading towardspos [0 0]
fd 49 setheading towardspos :there
setpos :there
end

to innersectors :number
if :number > 8 [stop]
setheading 20 + :number * 40
fillsh [sector2]
pu home pd
innersectors :number + 1
end

to sector2
pu fd 50 pd
make "there pos
fd 50 rt 90 repeat 35 [fd 1 rt .4]
setheading towardspos [0 0]
fd 50 setheading towards "there
setpos :there
end
```

Nurturing a Logo Culture From the Ground Up

by Nan Youngerman

I have been deeply immersed in the teaching and learning of Logo for more than 15 years. During that time, my work has revolved around the students at one school, and a Logo culture has grown around me. This culture valued curiosity, creativity, the sharing of ideas, peer interaction, and problem solving. As it established itself, it supported our efforts and helped both students and teachers grow and change as mathematicians, problem solvers, and collaborative thinkers.

This fall I began a new teaching position at the middle school level. At my new school, some Logo is taught by individual teachers who have considerable knowledge of Logo, mathematics, and children's thinking. No one was using Logo as extensively as I hoped to within my classroom. I was eager to start the first week, but I was puzzled about the best way to build a culture within my classroom that would support imaginative student Logo use, comfort with problem solving, and pride in complex mathematical thinking. I had a classroom computer, a lab, and willing collegial support.

Years of elementary experiences had to be used carefully at the middle school. I thought about a few of my favorite articles by Molly Watt, one in particular titled "Logo Building Blocks" (1984). Because of this article, I knew that I wanted a project that:

- limited turns to 90° angles,
- provided the excitement of open-ended lessons within a supportive structure, and
- allowed each student to work at his or her own level.

I chose an activity I call "Robot Writing" as a first project because it fit all three criteria. Each student was instructed to write his or her initials on the screen and, more importantly, to save his or her thinking in packages or procedures. I wanted to focus on learning the most basic commands, writing procedures, creating superprocedures, keeping useful and accurate journals, and having fun. I expected a range of abilities and needed an assignment that held the possibility of success for each individual, while still leaving lots of room to grow.

Most student work was basic, showing only a small amount of creativity. Students quickly grasped the commands: **rt 90**, **lt 90**, **fd**, **bk**, **pu**, **pd**, and **cg**. A few students played with their initials to add color, repeat them, or spin them. Most accepted the need to think on the LogoWriter "flip side" and put their thinking into procedures. In science lessons, we focused on procedure writing for experiments and had fun writing precise procedures to tie our shoes, sit in a chair, write our names, and so on. This project was an extension of those activities.

The second project, "Flags," (Fitch, 1992), offered both simplicity and challenge. Each student selected a different flag to draw with Logo. I cautioned them to think about their comfort levels with Logo and their problem-solving abilities. We looked at a poster of flags and analyzed the difficulty level of drawing the flag of Chad versus that of South Korea. We discussed 90-degree turns and the new thinking some flags might require. Students selected carefully and, in most cases, quite appropriately. No one knew about the **repeat** command initially. We discussed it and many used it immediately.

We expanded our command knowledge to include **setc** and **fill**. Students began using turns that were not 90 degrees. We strengthened our intuition about various turn sizes. What was a reasonable guess for starters? Ninety degrees became our anchor. We talked about half of 90, half more than 90, slightly more or less than 90, and just plain 90.

This second assignment opened up many opportunities for new knowledge, mistakes, and successes. I found that I needed to establish a calm environment that acknowledged mistakes as the norm. A mistake is OK. A mistake is a starting point. I'll be working on that for a while because I think many students have come to our classroom believing that a mistake is something bad, perhaps something to hide. Students are talking to each other a little. I hope to increase this conversation so that students can share their mistakes, their solutions, and their feelings.

A few students needed to draw circles for the flags of Japan and South Korea. I gave them a formula, **repeat 360 [fd 1 rt 1]**, and wished them luck! I said it would not be the right size circle. We discussed the relationship between the repeat number and the turn number. They understood the connection and made notes in their journals. I added, "The bigger the turn, the smaller the resulting circle."

In fact the students noted how to make half of a circle, a quarter of a circle, and excitedly talked about the yin-yang symbol and how to make it. I marveled at how quickly sixth graders could grasp these ideas and at the thought that these Logo thinkers were in elementary school only a few months ago. The thinking seemed to happen so easily for some students. A University of Wisconsin computer science student and Logo enthusiast came to mentor the circle thinkers. Within a morning they were using **setpos** with understanding and confidence.

Sam, a new student to our classroom and to Logo, chose the Israeli flag. I guessed it would be quite difficult for a first-time Logo thinker to draw the triangles to create a six-pointed Star of David. I wondered if the class would support his effort. Sam made most of the flag without any trouble. He figured out how to make a triangle, but was frustrated getting the triangle to point exactly how he wanted it. Positioning the turtle was the most difficult part of his work. He narrowed down his options using trial and error and one-to-one support until he figured out the exact heading he needed.

I asked students recently how they know if their work is "good."

Someone blurted out, "It's neat!" and then immediately retracted the statement saying "How silly. The computer makes everything look neat."

This provided us with a great opportunity to discuss neatness in our thinking: well-organized and clear step-by-step thinking. Students understand it, though their work may not always reflect it. Away from the computer, they sometimes do neat thinking, but hand in assignments or projects that look sloppy. We discussed the complexity of our thinking or how complicated it might be. Students agreed that it is possible for a beginning Logo thinker to make a design in a simple way. For example, they said that a square measuring 100 turtle steps on each side could be drawn using the instructions:

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

or in a more complicated way using:

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

I suggested an even more complicated way:

```
to square :x
repeat 4 [fd :x rt 90]
end
```

The last method is something the class saw for the first time during this discussion. On the chalkboard I didn't write **:x** but drew an exaggerated zigzag. Someone asked where the zigzag key was! I said it just means, "I'll tell you later." Besides a zigzag, I showed how mathematicians may insert an **:x** or **:n** into a problem. Lots of students wrote notes in their journals.

Two students immediately went to the computer lab and used variables in their projects to make rectangles and circles. Another student printed his "thinking" and asked if it would be OK to edit it to make it more complex. YES! Thinking about neatness in our thinking and trying to express our ideas in increasingly complicated ways is a challenge! Oddly enough, as we increase our understanding and think in more complex ways, our work gets simpler. Students discovered they can do more sophisticated projects in less time. I am quite amazed by how much they have learned since September (it is now November). I am certain that variables will be an area we will explore throughout this year as we increase our comfort level and skill with algebraic principles.

Is our thinking neat? Procedures are longer and more complicated, sometimes like spaghetti. The "Robot Writing" activity called for a clear procedure for each initial and students accepted the idea of keeping relative moves within procedures. The inside workings of flags are not as clearly defined. Many students mixed commands for outlines and colors together. Our third project, creating pictures using polygons, varied greatly from individual to individual. Each student was developing his or her own programming style and repertoire of mistakes! Clearly they could edit their thinking and break it into more doable steps.

What might happen next? It seems important to do a show soon to celebrate our work and perhaps examine each other's thinking. Working with partners may increase the sharing of ideas, math, and Logo knowledge, as well as support for problem solving. We need a bulletin board display in our classroom for quick reference and to show off good work. Perhaps other students would like to know about our work. Projects have been very mathematical in nature. Are we ready to research ancient civilizations and create multimedia projects?

I think a culture has begun to grow. Students' Logo knowledge is growing. They are thinking of new projects, trying each other's ideas, and solving each other's problems. Within our classroom, students are starting to share more freely among themselves and support each other when they are stuck. Error messages are not greeted with frustration. There is a growing determination to find mistakes. Our work is on display within our school. It is part of a local television show featuring technology use within our school. Time will help us gain comfort, take pride in our efforts, share our work with others, and make connections to other work outside of Logo. The culture is emerging. ▲

Nan Youngerman has taught in the Madison, Wisconsin, Metropolitan School District since 1971. She currently teaches sixth grade at Cherokee Middle School. She was the 1994 Wisconsin Presidential Awardee for Excellence in Mathematics and Science Education sponsored by the National Science Foundation. She can be reached during the school year at Cherokee Middle School, 4301 Cherokee Drive, Madison, WI 53711, or anytime by e-mail at neyounge@facstaff.wisc.edu.

References

Fitch, D. (1992). Flags of the world. *Logo Exchange, 11*(2), 7–10.

Watt, M. (1984, January). Logo building blocks. *inCider, 2*(*1*), 110-112.

Merging Technologies: Lego TC logo and the Color QuickCam

by Orlando Mihich

In one corner of our Lab sits the only surviving Apple IIGS from the era of the Apple IIe and the Apple IIGS. They have almost all been replaced by 23 new Macintosh computer. Students love their Macs; I have even witnessed embraces and hand clapping. These Macs are definitely "their machines!"

Still, there are some of us, the previous generation, who love those old machines. We spent hundreds of hours with them, creating so much work. Our diskettes are still here, and they still work. Only six years ago, the first Apple IIGS came into our lab, but one has the feeling that many, many more years went by.

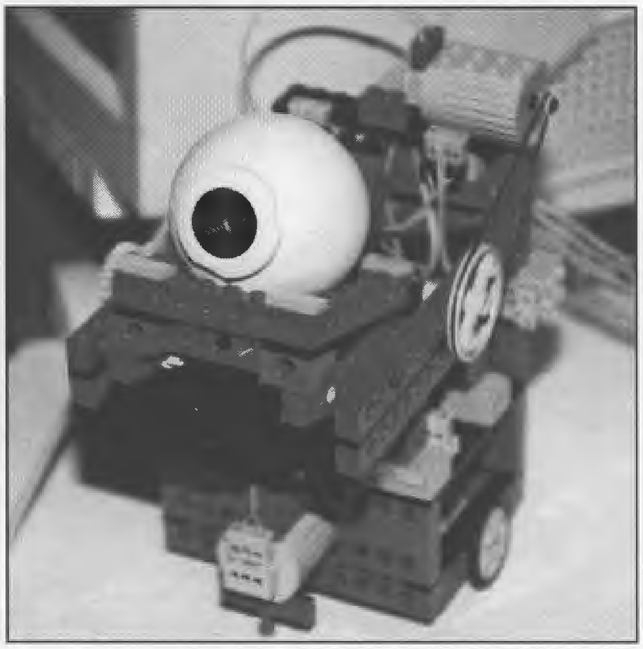
Thanks to a LEGO® TC logo Interface box attached to it, our Apple IIGS is still very active. Each year new students coming into the lab are attracted to the Apple IIGS and get involved in building some device with the numerous Lego blocks tucked away in a large box. Years ago, we stored every Lego piece neatly in the appropriate tray; in time we lost that good habit, and today at least three Lego sets are all in a big box. Still, students always seem to find all the pieces they need.

Off and on all last year, Patrick and Sam worked to assemble a great little "roamer," which went around the floor with flashing lights, piercing sounds, and hiccups, getting the attention of every student in school. The boys were very proud of their creation; they showed it to various visitors and parents.

This year, I purchased a Color QuickCam®, and, of course, every student got his or her 3 MB of fun. I started to use the QuickCam with my sixth-grade classes, and every student recited some lines under a floodlight and in front of the eye of the Color QuickCam. Sixth graders come in all shapes and heights, so one student had to spend the whole period tilting the QuickCam up and down, trying to catch the student being taped and digitized.

Here, Patrick and Sam, now in eighth grade, came up with this great, brilliant idea. Why not mount the QuickCam eyeball on their roamer?

It had survived the summer in a closet and was about to be forgotten. They set immediately to work and in two days came up with a working model of a moveable Color QuickCam controlled by the Apple IIGS.



The device can move in any direction, but, more importantly, it can lower and lift the eye of the Color QuickCam, searching for the student in front of the camera. Now, the whole taping process became very simple, and, needless to say, it looks very "digital studio." A student at the Apple IIGS controls the Color QuickCam and searches for the student sitting in front of the camera looking at the Macintosh screen. When on target, another student at the Mac clicks the "record" button and captures the student reciting his or her lines. Students include their recorded QuickTime movie in a ClarisWorks document.

Yesterday, I asked my two young engineers to write what they did and to explain their procedures. Patrick showed up this morning at 7:30 to check all the procedures and during lunch, with Sam, wrote the following text. As I read it, I became more and more impressed. First, I didn't know they could write something so professional, and second, could I have written something like this when I was 13? I know the answer is no.

Patrick Mahaney and Sam Lazarus

Welcome to LEGO TC logo!

By Patrick Mahaney and Sam Lazarus

LEGO® TC logo is a Logo-based program that is adapted to operate motors, lights, and touch-light sensors. The motors are hooked to a Lego creation and the creation can be made to move through hours of programming. We have utilized this program and our boundless Lego resources to create this vehicle.

The top of the vehicle has a harness that is built to hold the Connectix Color QuickCam. The harness is on top of a bulky frame with two wheels, two motors, and several lights. It is controlled by keyboard movements on our favorite computer, the Apple IIGS.

We have programmed the chassis and harness to control the movements of the QuickCam so that it is possible to film in almost 360°, in all directions. The only drawback is the bundle of wires. They limit the movement of the vehicle so that it has a very small amount of ground that it is able to cover.

Following are the procedures we wrote for the operation of the vehicle:

```
<<This procedure tells the car to go forward
  for a fixed amount of time>>

To tfd :time
talkto [a b]
seteven
onfor :time
end

<<This procedure tells the car to go back-
  ward for a fixed amount of time>>

To tbk :time
talkto [a b]
setodd
onfor :time
end

<<This procedure tells the car to turn right
  for a fixed amount of time>>

To trt :time
talkto "a
setodd
talkto "b
seteven
talkto [a b]
onfor :time
end

<<This procedure tells the car to turn left
  for a fixed amount of time>>

To tlt :time
talkto "a
seteven
talkto "b
setodd
talkto [a b]
onfor :time
end

<<Drive is the procedure that you use to
  move the car. Readchar "key tells the
  computer to use the keyboard to control
  whatever procedure you want it to.>>

To drive
talkto "a
setpower 7
talkto "b
setpower 7
name readchar "key
if :key = "8 [tfd 5]
if :key = "6 [trt 3]
if :key = "4 [tlt 3]
if :key = "2 [tbk 5]
if :key = "7 [tfd 5 tlt 5]
if :key = "9 [tfd 5 trt 5]
if :key = "1 [tbk 5 trt 5]
if :key = "3 [tbk 5 tlt 5]
if :key = "+ [talkto "c setpower 4 on]
if :key = "- [talkto "c setpower 4 off]
if :key = "* [talkto "c rd]
if :key = "x [stopall]
if :key = "= [tfd 50]
if :key = "/ [tbk 50]
drive
end
```

Orlando Mihich is a computer teacher at Booker T. Washington, MS 54, Community School District 3, New York City. He has taught science and Logo for the past 12 years.
E-mail: Omihich@aol.com

Hieroglyphs + Logo = Fun

by Robert Macdonald

The language of the ancient Egyptians

If you were to enter any large museum with displays of ancient scripts, you would probably observe Babylonian clay tablets etched with cuneiforms or Egyptian papyrus manuscripts covered with hieroglyphs. What the Greeks referred to as holy carvings (hieros = holy, glyphe = carving) were to remain "petrified" on the innumerable monuments of this ancient civilization.

How did scholars make any sense of these curious scrawls? How do people learn to decipher a language when they are ignorant of its sounds, the meaning behind the visual signs of its communication, and its grammar? How did the knowledge of Egyptian disappear? It is surely not the oldest language on the face of our planet, but it is one that has fascinated mankind for untold eons.

[*Editor's note:* Robert Macdonald's original article contains a fascinating and thorough history of hieroglyphics that helps answer these many questions. It details the discovery and deciphering of the Rosetta stone and many other related topics of historical interest. Because of space limitations, it is impossible to include the complete text here. However, you can receive an electronic copy, including its extensive bibliography, by sending an e-mail message to logoinfo@harvassoc.com with a subject line of: **request hierogl.txt**]

A microworld for fourth graders

Much of the material in this article was prepared for a fourth-grade class in the late 1980s. At that time one of my students expressed great interest in the language of the ancient Egyptians, rekindling my own curiosity.

Common research sources (*The World Book* and *The Encyclopedia Britannica*) did satisfy our quest for initial information. However, inexpensive Dover Publications provided us with a rich assortment of books that proved invaluable. We were especially indebted to publications by Sir E. A. Wallis Budge (1983), Stéphane Rossini (1989), and Peter der Manuelian (1991). Some time later we discovered a wonderful stamping kit issued by The Metropolitan Museum of Art (Roehrig, 1990). We invested in five of them. For a number of years the kits proved to be a popular sales item at our annual fall book fairs. That kit and the Scott & Scott's *Hieroglyphs for Fun* (1974) helped inspire the title for the present article.

In this microworld based on a long forgotten civilization, we also included, in addition to the language experiences provided in this article, some mathematical encounters: Egyptian multiplication (see Macdonald, 1991) and numerical hieroglyphics (see Harris, 1989).

A few things to consider

Constructing a computer program that will output a hieroglyphic code requires that we make some concessions. The complications of a true hieroglyphic program encompassing the complexities of this ancient language would tax the abilities of even the most able 10 year old. We must insist that each of the 26 letters of the English alphabet be represented by a unique hieroglyph. The reasons for our choices are provided below.

Start by looking at Figure 1, which shows a sample of our hieroglyphic code. Now let us look at the description of the hieroglyph that we have associated with each letter of the English alphabet:

a [forearm], **b** [foot], **c** [hillside], **d** [hand], **e** [two reed leaves joined], **f** [horned viper], **g** [pot stand], **h** [twisted flax], **i** [a single reed leaf with three vertical dots], **j** [cobra], **k** [basket], **l** [lion], **m** [owl], **n** [water], **o** [vulture], **p** [stool], **q** [basket + quail chick], **r** [open mouth], **s** [folded cloth], **t** [loaf of bread], **u** [variant monoliteral

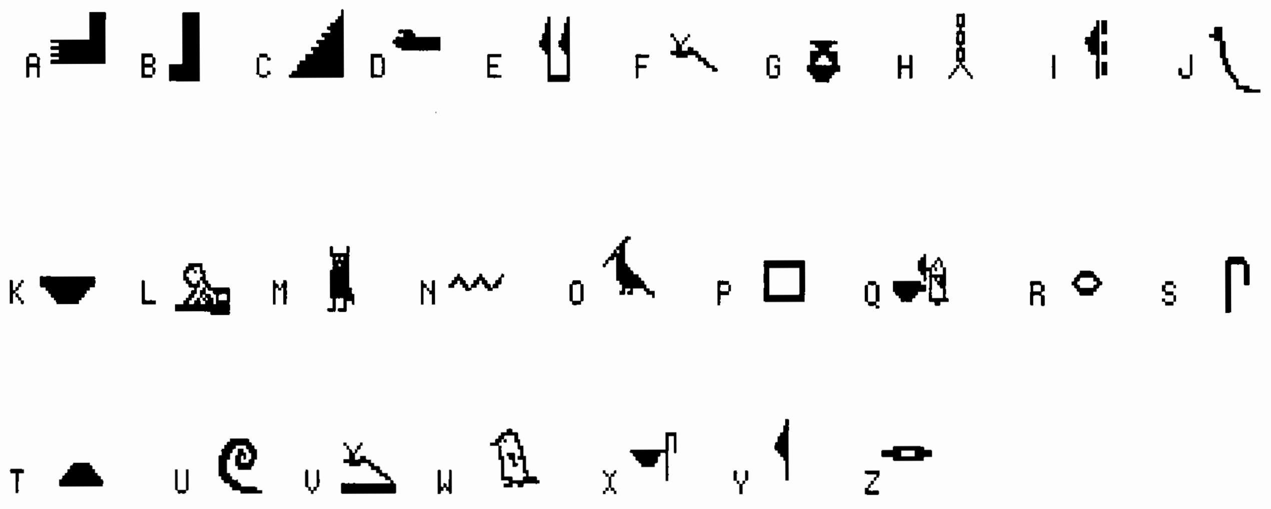


Figure 1

sign borrowed from Rossini, 1989, p. 25], **v** [horned viper rising from ground], **w** [quail chick], **x** [basket + folded cloth], **y** [single reed leaf], **z** [door bolt].

The reader surely deserves the reasons for the hieroglyphs selected above. Phonograms (phonetic renditions) found in hieroglyphs largely provide consonant sounds. They are divided into three categories: a monoliteral sign (one consonant sound—we shall limit our consonant hieroglyphs to this category), a biliteral sign (two consonant sounds, comparable in the broadest sense to English digraphs or consonant blends), and a triliteral sign (three consonant sounds—comparable again in the broadest terms to English consonant blends).

If we considered using only hieroglyphs that are monoliteral, we may easily assign hieroglyphs to the English consonants **b**, **d**, **f**, **g**, **h**, **j**, **k**, **l**, **m**, **n**, **p**, **r**, **s**, **t**, **w**, and **y**. Purists may argue that the **d** not only exists in Egyptian as the English **d**, but also as **dj** (as in fudge), the Egyptian **g** is hard, not soft; that is, it sounds like the **g** in **g**un, not the **g** in **g**in. The Egyptian **h** has four possible sounds, though we will use only the hissed **h**. **K** will represent **k**ing, not **q**ueen. We will use the **s** as in **s**ister, the **t** as in **t**iger, **w** as in **w**ater, and **y** as in **y**es. Note how we differentiate **f** and **v**. We compromise with the consonants **q**, **v**, **x**, and **z** (see Roehrig, 1989).

We fudge with the vowels, because the ancient Egyptians didn't write most vowels (Katan & Mintz, 1981, p. 55). My representations for **e** and **i** are personal adaptations from **y**. We know that in English **y** has a consonant sound as in **y**es and two vowel sounds as exemplified in myster**y**. We will apply a single reed leaf to the consonant sound, then for the long **e** sound, we will adapt double reed leaves joined (an example of a personal adaptation). The short **i** sound will be a reed leaf followed by three vertical dots (another personal adaptation). The **o** sound will be a vulture (see Roehrig, 1990, p. 18). The **a** sound will be a forearm. The **u** we freely adapted from Rossini (1995, p. 25). Remember we neglect to differentiate between long and short vowel sounds. It is our purpose to have a unique hieroglyph for each letter of the English alphabet.

In our representations (phonograms), we ignore a second group of hieroglyphs termed ideograms, which are signs that mean what they show, nor will we complicate our code with a third group of hieroglyphs called determinatives, which cannot be pronounced. They are used in conjunction with hieroglyphs that have more than one meaning for clarification.

We have delivered our apologies. Readers wishing to alter these representations should feel free to do so.

Creating shapes

The computer programs listed in the following sections are written in LogoWriter for the Macintosh. One of the features of the software is that each LogoWriter page includes a Shapes Editor in which up to 90 shapes may be defined. Because we are going to build this program by calling up shapes under numbers corresponding to their ASCII codes, we must construct our hieroglyphic shapes as follows: The hieroglyph representing **a** must be constructed in the space reserved for shape number 65, **b** = shape number 66, **c** = shape number 67, ... **z** = shape number 90.

When we type a character on the keyboard, the computer automatically converts it to a number. That number is part of a standard code called ASCII (American Standard Code for Information Interchange). Later when the computer displays characters, it converts that number back to a character. All of this occurs backstage, so that we, as the audience watching the screen, are unaware of all the complications.

We will be using the ASCII code to program the spacebar, return key, tab key, delete key, and escape key in the subprocedure **odds.and.ends**. Perhaps a quick review of discovering the numerical values of the various keys from the computer itself may be helpful.

If you wish to find the numerical value of **a**, enter the following in the Command Center:

```
show ascii "a
```

If you wish to input a character number to discover its key, use **char**, as in:

```
show char 97
```

Or to quickly discover an unknown ASCII number for a key, enter:

```
show ascii readchar
```

Press the Return key, then touch the key for which you desire the number. The ASCII number will appear in the Command Center.

If you desire to discover a group of ASCII numbers, you might write a simple procedure:

```
to ascii.readout
show ascii readchar
ascii.readout
end
```

Type the command **ascii.readout**, press Return then touch any key on the keyboard. The ASCII number will be revealed. Continue touching as needed. To get out of the program simply click the stop button.

When constructing shapes, make use of the relevant sources listed at the end of this article. Roehrig (1990) and Rossini (1989) are particularly useful. A Copy Me page showing the detailed shapes used in this article is provided on page 24.

The computer programs

Hieroglyphics may read from left to right, right to left, or vertically; the computer programs should do the same. However, because a vertical approach would cause scrolling off the screen without multiple safeguards and for present purposes seems unnecessary to pursue, we will only provide programs reading from left to right and right to left.

Wallis Budge (1983) informs us that the hieroglyphs look toward the direction from which the reading emanates. Therefore, after you create the hieroglyphs reading left to right, you must reverse the direction of the shapes. This is easily done. Copy the shapes from one program and paste them into the other. Then rotate each shape in the shapes editor of the new page. Use the appropriate button function.

The top-down design of the programs is easily followed. Almost all of the work is accomplished in the subprocedure **transliterate** by applying a conditional **ifelse** statement and making use of recursion. You may note that we have not given full directions in our **directions** subprocedure. Pressing the Escape key (char 27), calls the **message** subprocedure, which directs you to use the tab key (char 9) for a printout of the screen. Feel free to adapt.

Writing hieroglyphics—left to right

I named the page of this program **hieroglyphs.l.to.r**. The command of the program is **hieroglyphs** or simply **h**.

```
to hieroglyphs
preparation
directions
transliterate readchar
end

to h
hieroglyphs
end

to preparation
clearpage
right 90
pu
setpos [-213 83]
end

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

to directions
type [Type the message to be transliterated
  into hieroglyphs, press the SPACEBAR to
  create a space, press the RETURN to begin
  a new line, and click on the STOP button
  to stop and escape from the program.]
type char 13
end

to transliterate :letter
odds.and.ends
ifelse (ascii :letter) > 90
 [setsh (ascii :letter) - 32 stamp.it]
 [setsh (ascii :letter) stamp.it]
transliterate readchar
end

to odds.and.ends
if equal? :letter char 32 [fd 32]
if equal? :letter char 13 [setpos list -213
  ycor - 30]
if equal? :letter char 8 [setsh 11 back 16
  pe stamp pu back 16]
if equal? :letter char 27 [message]
if equal? :letter char 9 [printscreen
  stopall]
end

to stamp.it
pd
stamp
pu
forward 22
end
```

```
to message
type char 13
type [If you wish a printscreen of your
  message, touch the TAB key.]
type char 13
end
```

Let's try the program. Input **h**. Now press Return. The screen clears and the directions for using the program appear in the Command Center. We decide to type the following message, pressing the space bar after each word and carefully using the Return key to place the text on the next line when one line of text is complete:

```
Now is the time for all good
computers to come to the aid of
their Egyptologist.
```

The image shown in Figure 2 appears on the screen. Click on the stop button to exit the program.

Writing hieroglyphics—right to left

This program is based on the preceding one. I named its page **hieroglyphs.r.to.l**. There are several significant changes. In the subprocedure **preparation**, be certain to use a **left 90** command and place new inputs in the **setpos** command. In this case, **setpos [212 84]**. Note also an important change in programming the return key (char 13) in the subprocedure **odds.and.ends**: **[setpos list 212 ycor - 30]**. Remember that the writing is emanating from the right side of the page, not the left. Again the command is **hieroglyphs** or simply **h**.

```
to hieroglyphs
preparation
directions
transliterate readchar
end

to h
hieroglyphs
end

to preparation
clearpage
left 90
pu
setpos [212 84]
end
```

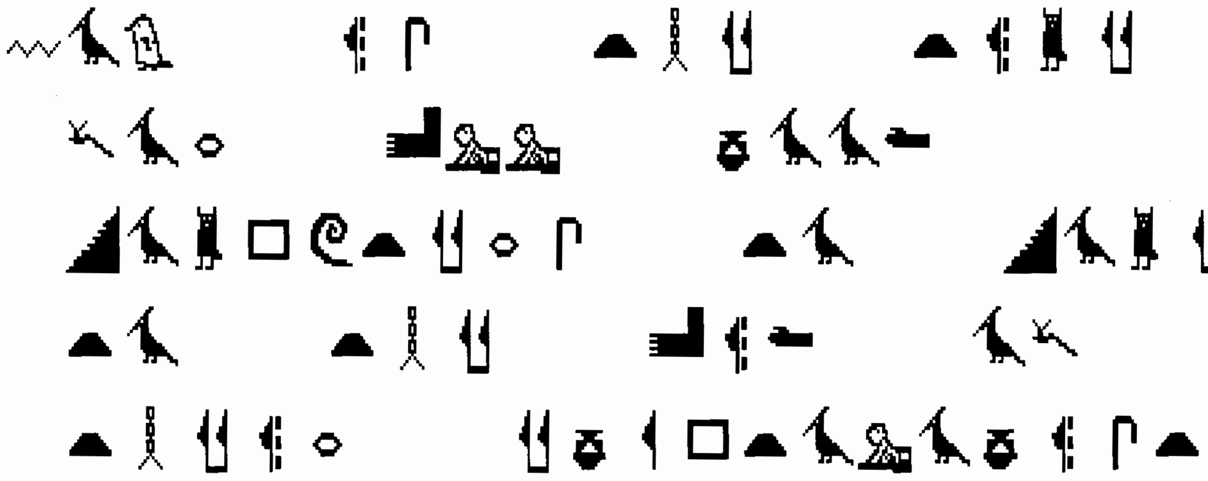


Figure 2

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

to directions
type [Type the message to be transliterated
  into hieroglyphs, press the SPACEBAR to
  create a space, press the RETURN to begin
  a new line, and click on the STOP button
  to stop and escape from the program.]
type char 13
end

to transliterate :letter
odds.and.ends
ifelse (ascii :letter) > 90
 [setsh (ascii :letter) - 32 stamp.it]
 [setsh (ascii :letter) stamp.it]
transliterate readchar
end

to odds.and.ends
if equal? :letter char 32 [fd 32]
if equal? :letter char 13 [setpos list 212
  ycor - 30]
if equal? :letter char 8 [setsh 11 back 16
  pe stamp pu back 16]
if equal? :letter char 27 [message]
if equal? :letter char 9 [printscreen
  stopall]
end
```

```
to stamp.it
pd
stamp
pu
forward 22
end

to message
type char 13
type [If you wish a printscreen of your
  message, touch the TAB key.]
type char 13
end
```

Now let's try the same input we used with our first program:

```
Now is the time for all good
computers to come to the aid of
their Egyptologist.
```

The hieroglyph message of Figure 3 appears. It is the reverse of the previous message. Note that the shapes are reversed to face the direction of reading.

Hieroglyphics and MicroWorlds

There are a number of other encounters we may experiment with in hieroglyphs, but they must await another article, one in which we shall transfer these programs into MicroWorlds. If you have MicroWorlds Project Builder, you

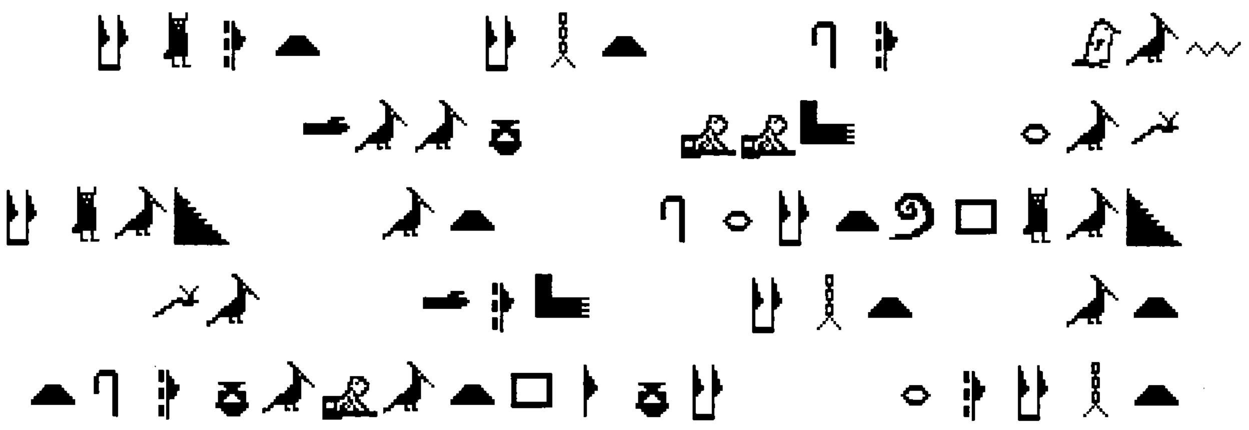


Figure 3

might like to experiment with building shapes representing hieroglyphs. The colors available to you will make the effort worth the time. ▲

Robert Macdonald
Hawthorne Meadows
10225 Nancy's Blvd.
Grosse Ile, Michigan 48138-2128

References

der Manuelian, P. (1991). *Hieroglyphs from A to Z: A rhyming book with ancient Egyptian stencils for kids.* Boston: Museum of Fine Arts.

Harris, J. (1989). Place value–place/hieroglyphics AHA!'s. *The Computing Teacher, 17*(2), 40–42.

Katan, J., & Mintz, B. (1981). *Hieroglyphs. the writing of ancient Egypt.* New York: Atheneum.

Macdonald, R. (1991, Autumn). An old algorithm for multiplication. *CLIME News, 4*(*1*), 12–13.

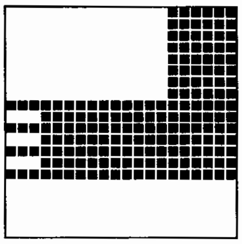
Roehrig, C. (1990). *Fun with hieroglyphs.* New York: The Metropolitan Museum of Art and Viking. (Contains 24 rubber stamps, hieroglyph guide book, and ink pad.)

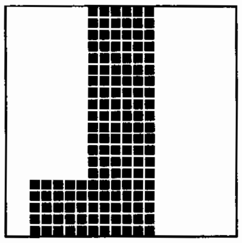
Rossini, S. (1989). *Egyptian hieroglyphics. How to read and write them.* New York: Dover Publications, Inc.

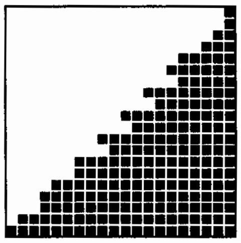
Scott, J., & Scott, L. (1974). *Hieroglyphs for fun—Your own secret code language.* New York: Van Nostrand Reinhold.

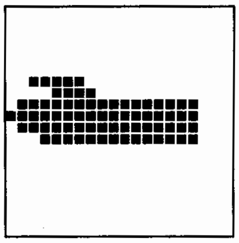
Wallis Budge, Sir E. A. (1983). *Egyptian language. Easy lessons in Egyptian hieroglyphics.* New York: Dover Publications, Inc. (This is a reprint of the eighth edition; the first edition was published in 1889. Note that he has also written books on many other Egyptian topics.)

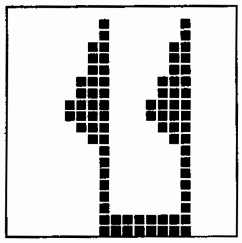
LX Copy Me Page: Hieroglyph Shapes

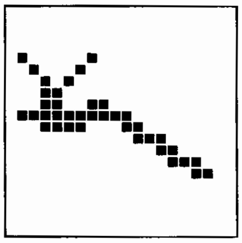

A – forearm

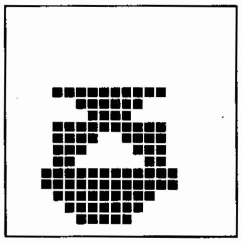

B – foot

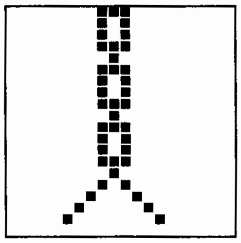

C – hillside

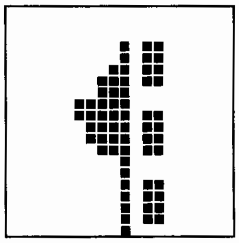

D – hand

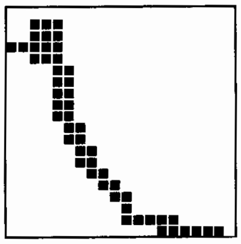

E – two reed leaves

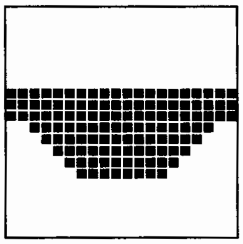

F– horned viper

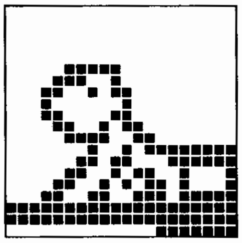

G – pot stand

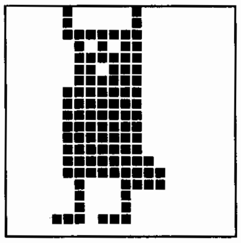

H – twisted flax

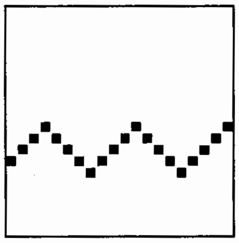

I – one leaf and dots

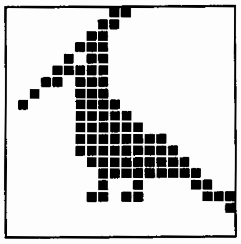

J – cobra

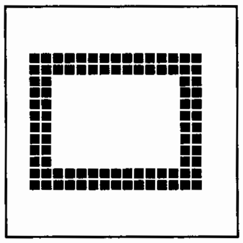

K– basket

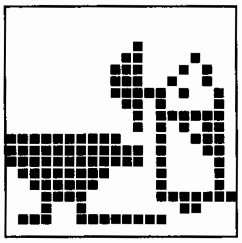

L – lion

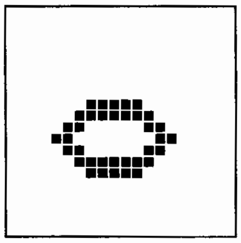

M – owl

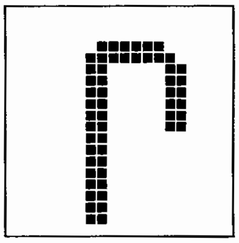

N – water

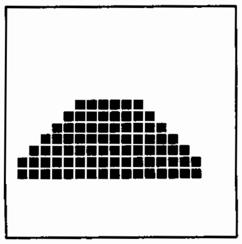

O – vulture

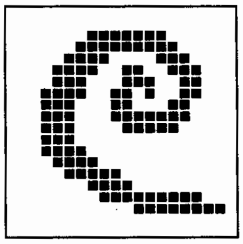

P – stool

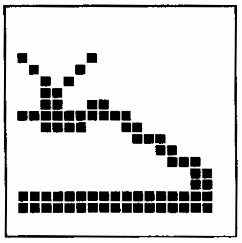

Q – basket+quail

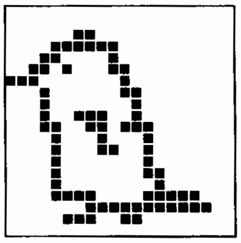

R – open mouth


S – folded cloth


T – loaf of bread


U – symbol

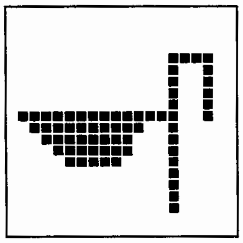

V – rising viper

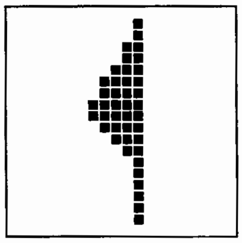

W – quail chick

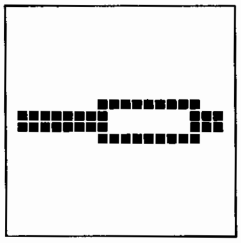
Use these shape designs in your explorations with hieroglyphics.

Or use them as models for your own hieroglyphs!

Be creative!


X – basket + cloth


Y – single reed leaf


Z – door bolt

More Fun on the Text Screen

by Jim Muller

One of the first things every young Logo explorer seems to do when introduced to the language is start creating shapes. First comes the square, then typically a triangle, and so forth.

Eventually, these explorers get around to the fun of list processing. Then it's time to go back to the shapes again.

In the related article "ASCII Artwork" in this issue, young Logo fan(atic)s get the chance to explore designs on the text screen. Typists have long enjoyed creating graphics using the keys of the typewriter.

```
      |  0
      |
|     |   |   |/\/\
\_____/   |   | | |
```

Logo lets you do the same thing using the **char** primitive to display ASCII codes. You can even create word shapes.

Word triangles

Here's a simple one, a word triangle:

```
That's_all_for_now_folks!
That's_all_for_now_folks
That's_all_for_now_folk
That's_all_for_now_fol
That's_all_for_now_fo
That's_all_for_now_f
That's_all_for_now_
That's_all_for_now
That's_all_for_no
That's_all_for_n
That's_all_for_
That's_all_for
That's_all_fo
That's_all_f
That's_all_
That's_all
That's_al
That's_a
That's_
That's
That'
That
Tha
Th
T
```

The procedure for this triangle is equally simple:

```
to wordtriangle :words
if :words = [] [stop]
pr butlast :words
wordtriangle butlast :words
end

wordtriangle "That's_all_for_now_folks!
```

This procedure takes whatever word you select for **:words** and prints everything but the last character. The variable **:words** then becomes everything but the last character of words. It does this repeatedly until **:words** is empty.

More word triangles

This is all well and good, but what about other types of shapes? What about other types of triangles?

The first and most obvious change is to replace **butlast** with **butfirst**.

Although the sequence of letters changes, you still have the same basic shape. What about reversing the shape?

This gets a bit more complicated. Rather than simply dropping a character for each line, you have to add one. What you want is something like this:

```
T
Th
Tha
That
That'
That's
That's_
That's_a
That's_al
That's_all
```

and so forth.

You can start off easily enough:

```
print first :words
```

Then you have to pick up each subsequent character. One way to do that is to make what you print a variable.

```
make "line word (first :words) (first
  butfirst :words)
```

This tells Logo to create a variable called "line" from a word that consists of the first and second characters of **:words**.

First :words is easy to understand. **First butfirst :words** is simply a confusing way to specify the second character. It says to select the first character after **:words** has become everything but the first character.

You start with this:

```
That's_all_for_now_folks!
```

First :words is `"T`. **Butfirst :words** is `"hat's_all_for_now_folks!` So **first butfirst :words** is `"h`.

There is more than one way to continue making this new triangle. A simple method is to define a new procedure to print the rest. The procedure **rest** is called by the last line of the **start** procedure.

```
rest butfirst butfirst :words
```

You should know this one by now. It chops off the first two letters you have already printed. The rest of **rest** is straightforward.

```
to start :words
pr first :words
make "line word (first :words) (first bf
  :words)
pr :line
rest butfirst butfirst :words
end

to rest :words
if (first :words) = " [stop]
make "line word :line (first :words)
pr :line
rest butfirst :words
end
```

Now you can make two types of triangles. What about another one?

```
             T
            THA
           THAT'
          THAT'S_
         THAT'S_AL
        THAT'S_ALL_
       THAT'S_ALL_FO
      THAT'S_ALL_FOR_
     THAT'S_ALL_FOR_NO
    THAT'S_ALL_FOR_NOW,
   THAT'S_ALL_FOR_NOW,_F
  THAT'S_ALL_FOR_NOW,_FOL
THAT'S_ALL_FOR_NOW,_FOLKS
```

```
to start :words
ct ts
tc ((count :words) / 2)
pr first :words
make "line (word (first :words) (first bf
  :words) (first bf bf :words))
tc ((count :words) / 2) - 1
pr :line
rest butfirst butfirst butfirst :words
end

to rest :words
if (first :words) = " [stop]
make "line (word :line (first :words) (first
  bf :words))
tc ((count :words) / 2) - 1
pr :line
rest butfirst butfirst :words
end

to tc :n
if :n < 1 [stop]
repeat :n [type char 32]
end
```

You should be able to trace this easily enough by now. The only thing difference is really the use of **count**. **((Count :words) / 2)** simply tells Logo to count the number of characters in **:words** and divide by two. Then typing **char 32** that number of times puts the cursor at the top of the tree-shaped triangle. For each line that you print, you back up one space: **tc ((count :words) / 2) – 1)**

Other word shapes

Triangles are triangles. What about other shapes?

Make a diamond. Print the tree-shaped triangle previously shown, and then reverse the process so that you print from the longest line to the shortest.

What about a square? A rectangle? That's easy enough.

Some versions of Logo do not have a **cursor** or **setcursor** primitive. You can fake it even though the following example isn't quite the same.

The important difference is that this procedure positions the cursor based on where it is at the time the procedure is called, and not from row 0, column 0, as is the more typical scenario.

```
to setcursor :row :col
repeat :row [pr "]
repeat :col [type char 32]
end
```

Boxword draws a square of text of any size anywhere on the text screen. **Start** sets up the parameters for the square. The **:row** and **:col** variables define the starting point. The **:size** variable defines the width of the square, or the number of characters in each line. The variables **:limitc** and **:limitr** are simply variables to count the number of columns and rows typed. Note that in some versions of Logo you need to use **print1** instead of **type**.

```
to start :row :col :size
ts ct
make "words
  "supercalifragilisticexpialidocious
make "limitc :size + :col
make "limitr :size
setcursor :row :col
boxword
end

to boxword
if :col = :limitc [move]
if :limitr = 0 [stop]
type first :words
make "words lput first :words butfirst
  :words
make "col :col + 1
boxword
end
```

In the middle of **boxword**, there's a critical line that defines the next letter to be typed.

```
make "words lput first :words butfirst
  :words
```

After you type the first letter of **:words**, Logo takes that first letter and "last puts" it at the end of **:words**. For example, if **:words** is **"cat**, Logo takes **first :words** (**"c**) and "last puts" it on **butfirst "cat**, or **"at**. So you end up with **"atc**.

Each time **boxword** types a letter, it checks the position. When **:col** equals the right-hand limit, the **move** procedure is called and the cursor moves to the next line.

```
to move
make "row 1
make "limitr :limitr - 2
make "col :limitc - :size
setcursor :row :col
end
```

Because PC Logo's font is about twice as high as it is wide, a square takes half as many columns as rows to make a square shape. This is why the counter is reduced by two instead of one. The result is:

```
start 1 1 20

SUPERCALIFRAGILISTIC
EXPIALIDOCIOUSSUPERC
ALIFRAGILISTICEXPIAL
IDOCIOUSSUPERCALIFRA
GILISTICEXPIALIDOCIO
USSUPERCALIFRAGILIST
ICEXPIALIDOCIOUSSUPE
RCALIFRAGILISTICEXPI
ALIDOCIOUSSUPERCALIF
RAGILISTICEXPIALIDOC
```

Or you could reduce the counter by 1 instead of 2 and add a space between the letters in each row to make a square with as many rows and columns. Can you figure out how to do that? The revised result would be:

```
start 1 1 5

S U P E R
C A L I F
R A G I L
I S T I C
E X P I A
```

Now it's your turn. Have fun exploring!

Read about ***Jim Muller*** at the end of his other article in this issue, "ASCII Artwork." ▲

Math Activity: Get Rich Quick!?

by Ihor Charischak, reprinted with permission from CLIME Connections, Fall 1992

Here is an activity that will surprise and amaze your students. First, the job offer:

Dear ____________,

Your application for a job with *Fly by Night Widgets* for next summer has been reviewed by our personnel department and we would like to inform you that ***you got the job***! You will be working the entire month of July as a widget entrepreneur. As far as payment is concerned, the wage distributor offers you a choice of payment plans.

<u>Payment Plan #1:</u> You can start with $50 the first day, then you will get an additional $2 each day. That means you will make $50 on day #1, $52 on day #2, $54 on day #3, and so on for the entire month. Now our liberal fringe benefits include holidays on Saturdays and Sundays (at no pay), so that your total number of working days will be 22.

<u>Payment Plan #2:</u> You will be paid 1¢ the first day, 2¢ the second, 4¢ the third, 8¢ the fourth, and so on for the 22 days.

Please <u>circle</u> the plan you wish to have:

Plan #1 **Plan #2**

Welcome aboard! We look forward to seeing you next summer.

Sincerely yours,

Mr. & Mrs. Widget

The fun part of doing this activity is to see the reactions of the students who choose Plan #1 when they discover their very expensive mistake. The inclination is, of course, to go after the "big bucks" immediately (Plan #1), which does work, but only for a while.

One approach is to have students work on this problem with a calculator or a spreadsheet program and fill out a chart (see Table 1). Another way to do the problem is to have students write a Logo program that does the calculations for them. Once the program is complete, the student can "perform" the activity on another student, or even on a faculty member! (Students get a big kick out of "fooling" adults and sharing what they are able to accomplish on a computer.)

Lesson scenario

The teacher hands out the job-offer letter. After reading the problem, the students make their choices and hand in their letters. A tabulation is made of which plans the students chose. Here's an example of a possible conversation.

T (Teacher): Let's see if we can get this computer to help us see which plan is better. Let's start with the first plan. Tell me how much I make each day. The students help the teacher generate this chart on the blackboard.

DAY	AMOUNT
1	50
2	52
3	54
4	56
5	58

T: I think you can see how to continue the pattern. But can you figure out a way to determine how much you would get on the 20th day without listing all the amounts in between?

After some conversation and trial and error, one student noticed the following.

S (Sandra): I see a pattern that might help. If I take 50 away from each number, I notice that I can figure out the new number by doubling the previous Day number.

The other students look puzzled. Sandra continues.

S: Let me show you what I mean. If you take 50 away from each amount, you get the sequence 0, 2, 4, 6, 8. So Day 5 gives you double the previous Day number (4), which is 8. And if you want the actual amount, just add 50!

T: So what is it for the 10th day?

S: The previous day is Day 9, so the amount is 9 times 2 plus 50, which is 68.

T: Does everyone see what Sandra is doing?

After a couple of examples, the other students catch on. The teacher then goes on to help the students develop the rest of the program.

Program for Plan #1

The program to illustrate Plan #1 can be started with a few simple instructions. This first example shows how to computer the wage for Day 10.

```
make "day 10
make "wages1 (:day - 1) * 2 + 50
print :wages1
68
```

Day	Plan 1	Plan 2	Sum 1	Sum 2	Sum 1 – Sum 2
1	50.00	0.01	50.00	0.01	49.99
2	52.00	0.02	102.00	0.03	101.97
3	54.00	0.04	156.00	0.07	155.93
4	56.00	0.08	212.00	0.15	211.85
5	58.00	0.16	270.00	0.31	269.69
6	60.00	0.32	330.00	0.63	329.37
7	62.00	0.64	392.00	1.27	390.73
8	64.00	1.28	456.00	2.55	453.45
9	66.00	2.56	522.00	5.11	516.89
10	68.00	5.12	590.00	10.23	579.77
11	70.00	10.24	660.00	20.47	639.53
12	72.00	20.48	732.00	40.95	691.05
13	74.00	40.96	806.00	81.91	724.09
14	76.00	81.92	882.00	163.83	718.17
15	78.00	163.84	960.00	327.67	632.33
16	80.00	327.68	1040.00	655.35	384.65
17	82.00	655.36	1122.00	1310.71	-188.71
18	84.00	1310.72	1206.00	2621.43	-1415.43
19	86.00	2621.44	1292.00	5242.87	-3950.87
20	88.00	5242.88	1380.00	10485.75	-9105.75
21	90.00	10485.76	1470.00	20971.51	-19501.51
22	92.00	20971.52	1562.00	41943.03	-40381.03

Table 1

The computer displays 68, confirming Sandra's previous calculation. If we want to find out how much one earns on the last day (22), we simply enter:

```
make "day 22
make "wages1 (:day - 1) * 2 + 50
print :wages1
2100
```

The formula is applied and 2100 is printed. (This appears to be an impressive amount.)

Note that these instructions could also be written as a procedure that reports the wage for the input day.

```
to plan1 :day
print (:day - 1) * 2 + 50
end

plan1 10
Result: 68
plan1 22
Result 2100
```

Program for Plan #2

A similar formula can be written for the second payment plan. It should deliver 1¢ for the first day and indicate what is earned on any given day.

```
make "day 5
make "wages2 .01
repeat :day - 1 [make "wages2 :wages2 * 2]
print :wages2
.16
```

The value of **wages2** is multiplied by 2 and this becomes the new value of **wages2**. This is repeated 4 times. (The amount for Day 1 is provided in the **make "wages2** instruction, so only four more calculations need to be made). The value of **wages2**, which started at .01, becomes .02, .04, .08, and finally .16. So on the fifth day, you would earn $.16.

Total earnings for the month

We also need a formula to see what our total earnings are for the month.

For Plan #1:

```
make "total1 0
make "wages1 (:day - 1) * 2 + 50
make "total1 :total1 + :wages1
```

The new **total1** is the old total (0) plus Day 1's earnings (50), which now equals 50.

The same is done with Plan #2. The initial total is set to 0.

```
make "total2 0
```

On Day 1, **wages2** is set to .01.

```
if :days = 1 [make "wages2 .01]
```

On subsequent days the repeat formula is used to determine the wages for a given day.

```
if :days = 1 [make "wages2 1] [make "wages2
  :wages2 * 2]
```

The new **total2** is the old total (0) plus one day's earnings (.01), which now equals .01.

```
make "total2 :total2 + :wages2
```

The result of the day's transaction is printed out for all to see:

```
(print :wages1 :wages2 :total1 :total2)
```

These instructions are then put inside a procedure with a recursive call so the calculations can be performed for a given number of days.

```
to loop :counter :days
if (:counter > :days) [stop]
make "wages1 (:counter - 1) * 2 + 50
make "total1 :total1 + :wages1
if :counter = 1 [make "wages2 .01]
if :counter > 1 [make "wages2 :wages2 * 2]
make "total2 :total2 + :wages2
(print :counter :wages1 :wages2 :total1
  :total2 (:total1 - :total2))
wait 2000 ; for pause between days
loop :counter + 1 :days
end
```

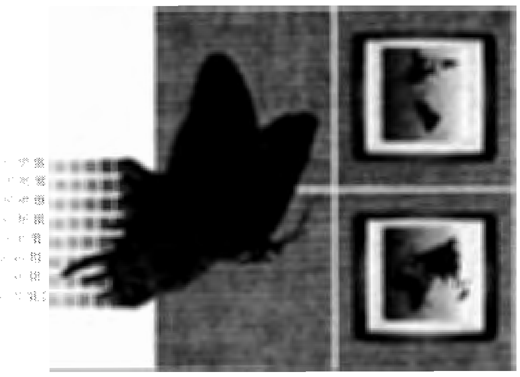
Finally, the superprocedure **report** initializes variables and limits the number of inputs needed to one.

```
to report :days
make "total1 0
make "total2 0
make "wages1 0
make "wages2 0
loop 1 :days
end
```

If you enter **report 22**, you see the entire story, illustrated in the Table 1. The program in this article does not format the table with columns and headers—that's up to you!

Students should be encouraged to make predictions after each day's results are posted. They will notice that Plan #1 is better until Day 14, when the advantage starts to slip. The lesson should end with a "debriefing," during which the teacher asks the students to explain what happened. Why did the second plan start off so slowly, but then do so well? This is a real opportunity for students to share and explore the mathematics behind the problem! ▲

In addition to founding CLIME (Council for Logo & Technology in Math Education) in 1987 and supporting it ever since, ***Ihor Charischak*** is program manager for the Center for Improved Engineering and Science Education (CIESE) at Stevens Institute of Technology. He can be reached at 10 Bogert Avenue, White Plains, NY 10606, or by e-mail at climecon@aol.com



TEL•ED '97

Taking Flight in the Digital Age

Austin, Texas, USA ✈ *Mexico City, Mexico*
November 13-16, 1997

THE MIGRATION BEGINS WITH YOU.

ISTE'S SIXTH INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS AND MULTIMEDIA IN EDUCATION is migrating to Austin, Texas, and Mexico City, Mexico, next November — and as an education stake-holder of the '90s who understands the dramatic, positive changes interactive technologies have made in the classroom, you should be too.

Exponential growth on the Internet. Transformations in the use of the World Wide Web. These developments have made a revolutionary impact in education over the past decade — literally opening up the classroom walls to new collaborations and innovative ways of communication.

ISTE's Tel•Ed has been on the forefront of this educational metamorphosis since 1989, providing practical solutions to better integrate multimedia and telecommunications into our diverse learning environments. ISTE's multisite Tel•Ed conference in 1997 will offer hands-on workshops, classroom-tested sessions, professional networking, a cutting-edge trade show and much more, inspiring you to take your students of today into the learning opportunities of tomorrow.

So don't let the Tel•Ed migration pass overhead. Spread your wings and "Take Flight in the Digital Age."

For more Tel•Ed information (or to receive a free *Call for Participation* or *Advance Program*), please contact ISTE:

Phone: 541/346-2472 • Fax: 541/346-5890
WWW: http://www.iste.org
Internet: laurie_thornley@ccmail.uoregon.edu

Presented By:
International Society for Technology in Education (ISTE)
Texas Computer Educators Association (TCEA)
Instituto Latinoamericano de la Comunicación Educativa (ILCE)

Learning in Logo MicroWorlds: OzLogo National Conference 1996

reported by Jenny Betts

Last October, OzLogo, the Australian Logo Special Interest Group, convened by the Computing in Education Group of Victoria, held a Logo conference in Melbourne specifically to celebrate the 21st anniversary of Logo use in Australia. The title of the conference was "Learning in Logo MicroWorlds: OzLogo National Conference 1996."

The conference provided educators with the opportunity to meet other Logo enthusiasts, discuss theory and philosophy, and explore the many ways in which Logo is used to enhance learning. Below is a brief summary of the papers, workshops, and panels presented by the delegates.

Invited speakers

Gary Stager from Pepperdine University, USA, was invited as the international keynote speaker to give the closing address, in which he would reflect on the many issues that arose during the two days.

Three Australian teacher/researchers were invited to speak of their experiences teaching Logo.

- Pam Gibbons (Australian Catholic University, Sydney) spoke about the way in which her students demonstrated a range of different processes of learning when using Logo.
- Jenny Betts (Brisbane) illustrated her pedagogical approach to using Logo in classrooms, particularly those in which each child has liberal access to a notebook computer.
- Tonia Chapman (Westhall Primary School, Melbourne) spoke about how she used MicroWorlds with a Year 4 class in a highly multicultural suburb in the southeastern part of Melbourne, where 90% of the children within the school speak English as a second language.

Other presentations

Many other presenters covered a wide range of Logo-related topics.

- Martin Boyle presented a fascinating biography of Seymour Papert.
- John Oakley presented the findings of his current research project involving the study of young children as programmers.
- Narelle Best shared the work that she has been doing with her Year 7 students.
- Craig Kerwin addressed the issue of teacher inservices, including how to get teachers started and enthused about using Logo in the classroom.
- Steve Costa detailed the significance of integrating the computer and MicroWorlds into the classroom curriculum.
- Anne McDougall and Jenny Betts outlined the ways in which robotic turtles, LEGO® TC logo, and MicroWorlds are being used to support curriculum throughout a large P–12 technology-saturated school.
- Craig Duncan outlined the benefits of using MicroWorlds with elementary students to explore a number of abstract concepts found in most science curriculum documents.
- Nicola Yelland and Jennifer Masters presented data from two research studies that involved the use of GeoLogo and Turtle Math. The first was a longitudinal study in which they worked with a small group of children during a period of two years. The second investigated the spontaneous learning of 84 children in Year 3.
- Josie Hopkins spoke about her Year 10 students using StarLogo. The main focus was on tracking the students' learning as they challenged the assumptions we make regarding complex systems and emergent phenomena.

Workshops

- John Annable dealt with how to obtain MSWLogo from the Internet. He also tutored participants on creating games using MicroWorlds.
- Craig Duncan presented a MicroWorlds workshop in which participants gained experience incorporating the interaction tools (slider, buttons, etc.) into projects and identifying differences that exist between the primitives of this environment and that of LogoWriter.
- Fran Reedy tutored participants on the concept of creating a TV show in MicroWorlds. She also provided the opportunity for participants to explore the idea of list processing and how this can be used to create projects such as "Old McDonald had a Farm" and "12 Days of Christmas."
- Craig Kerwin gave participants the opportunity to view and play with a myriad of activities that can be incorporated easily into the current curriculum.
- Josie Hopkins discussed and examined the StarLogo project work her students had been doing recently.
- Kevin Maguire mirrored a workshop experience based on one he had attended last summer at the Pepperdine Institute in the United States.

Panel discussions

- "The Internet and Other Resources for Logo Users."
- "MicroWorlds: Enhancing What We Did With Logo."

New resources

Three new resources were made available at the conference.

- *Logo In Australia: Ten Years On*, a book containing many of the papers that were presented during the 1985 Logo Conference.
- *Logo in Australia: Selected Readings*, a book containing a selection of papers written by researchers and teachers during the last 21 years.
- Proceedings of the conference.

These three resources may be ordered at the OzLogo Web site:

`http://powerup.com.au/~jbetts/index.htm`

A special dinner

Naturally, every conference has a dinner, and this Logo conference was no exception. The special guest was Sandra Wills, associate professor at the University of Wollongong, former International Chair of the World Conference for Computers in Education in 1990 in Sydney, Australia's representative on the technical committee for Education in IFIP, and Australia's very first Logo teacher who was the driving force behind introducing Logo into Australia.

Sandra entertained the guests with tales of travelling to Tasmania and introducing Logo into the Tasmanian schools, as well as sharing the developments of what was known as the Tassie Turtle (more formally called the Tasman Turtle).

Later in the evening, Sandra and Anne McDougall, another pioneering Logo user and now an associate professor at Monash University, were invited to cut the celebratory cake, on top of which sat a large green turtle, surrounded by 21 sparklers. This event marked the 21st anniversary of Logo use within Australia, the theme of the conference.

In closing

Gary Stager's lively and thought-provoking closing address left the Australian educators feeling very enthused and positive about using Logo with their students.

It would be fair to say that participating in a conference that focuses specifically on one area is always stimulating, and this conference certainly provided a number of stimulating discussions. Delegates were able to enjoy the saturation of specialised ideas—ideas that will provide a springboard for educators to enhance their existing uses of Logo in exciting, microworld learning environments. ▲

Jenny Betts is the Coordinator for Computing (K–5) at the Elementary School of John Paul College in Australia.
E-mail: jbetts@powerup.com.au

No Hype.
No Hoopla.

Just One Great Deal.

Logo Lovers:
Trade-up to a

MicroWorlds™

Site License For Just $499†

trade-up@lcsi.ca
1.800.321.5646
http://www.lcsi.ca

† All registered owners of commercial Logo-based programs qualify. Price in US funds. Offer valid until July 31, 1997. Not for resale.

ISTE BRINGS THE WORLD OF TECHNOLOGY CLOSER TO YOU.

By drawing from the resources of committed professionals worldwide, ISTE provides support that helps educators like you prepare for the future of education.

As an ISTE member, you benefit from a wide variety of publications, national policy leadership, and our work with Teacher Accreditation.

You also enjoy exciting conferences, global peer networking, and graduate-level Distance Education courses.

So if you're interested in the education of tomorrow, call us today.

International Society for Technology in Education
Customer Service Office
480 Charnelton Street, Eugene, OR 97401-2626 USA
Phone: 800/336-5191 *(US and Canada)*
Phone: 541/302-3777 *(International)*
Fax: 541/302-3778
Internet: cust_svc@ccmail.uoregon.edu
World Wide Web: http://www.iste.org

WE'LL PUT YOU IN TOUCH WITH THE WORLD.