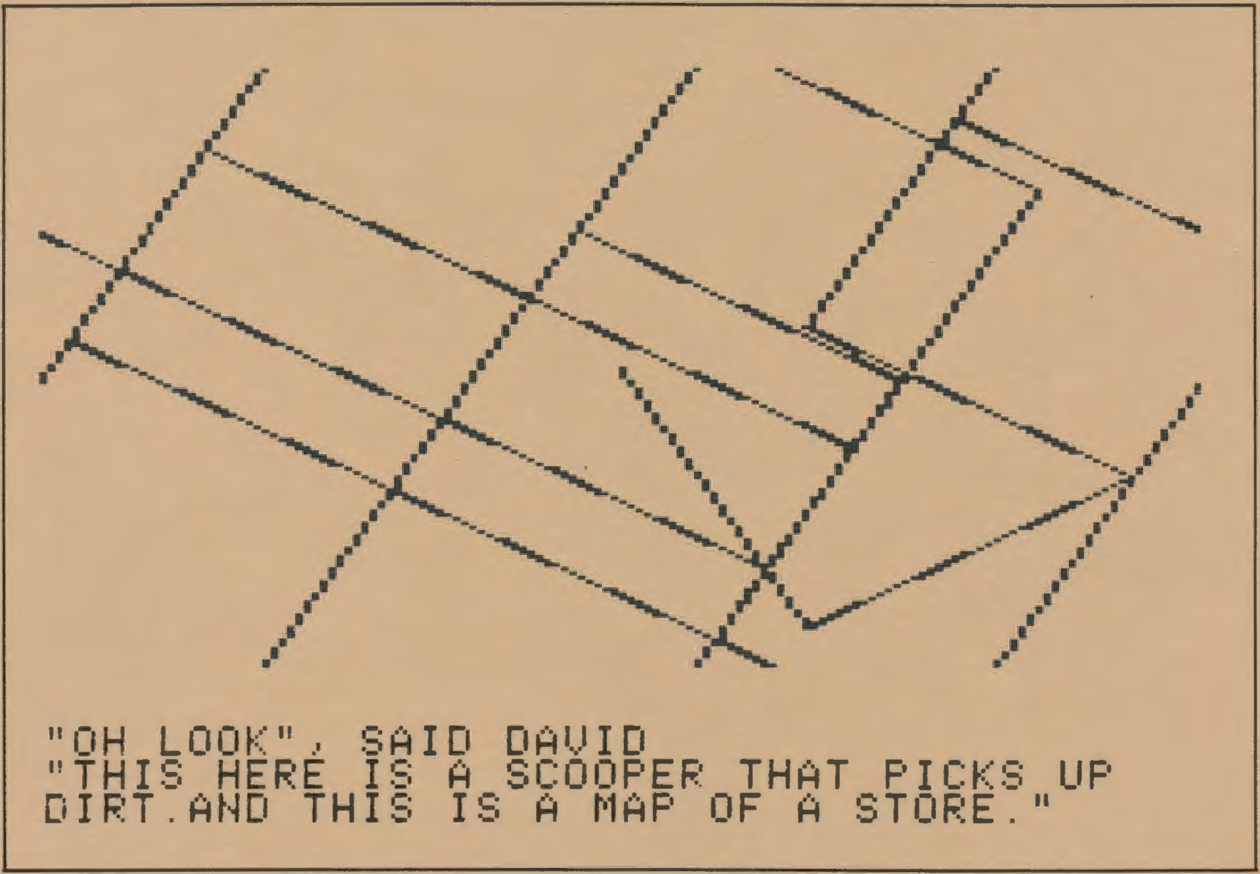
Journal of the ICCE Special Interest Group for Logo-Using Educators

LOGO EXCHANGE

MARCH 1989 VOLUME 7 NUMBER 7



```
"OH LOOK", SAID DAVID
"THIS HERE IS A SCOOPER THAT PICKS UP
DIRT.AND THIS IS A MAP OF A STORE."
```

International Council for Computers in Education

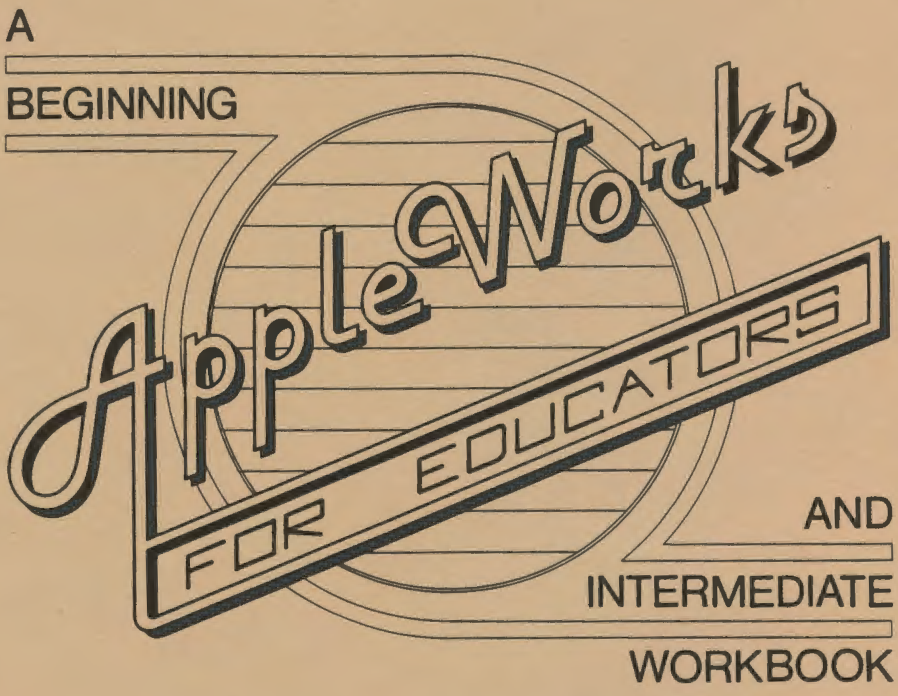
ICCE Publications

GOOD JUST GOT BETTER!

This revision of Linda Rathje's popular workbook improves upon the earlier edition by including a section on the mail merge function, expanded integration activities, a glossary, and *The Computing Teacher's* most current articles examining application software in the classroom. It guides the beginning - and now intermediate - **AppleWorks®** user through word processing, database and spreadsheet management, and printer options. Each section provides step-by-step instructions and practical applications for educators.

The new **AppleWorks for Educators** is laser printed and includes a data disk of working examples to guide you through each application as it appears in the workbook.

With the busy workshop season right around the corner, now is the time to call in your order for the 1988-89 version of **AppleWorks for Educators - A Beginning and Intermediate Workbook.**



ORDER FORM

Name

Address

City/State

Postal Code

Phone

ICCE Member Number
(Members receive a 10% discount)

AppleWorks Prices

# Copies	Price
1-4	$22.95
5-9	$20.65
10-99	$18.35

Shipping Cost

$0-20	Add $2.50
$21-50	Add $3.50
$51-90	Add $4.50
$91 up	Add 5% of total

Call for prices and shipping on large orders: 503/686-4414.

___ copies of AppleWorks
@ _______ each = _______
- 10% for ICCE Member (include ICCE member #) = _______
+ Shipping = _______
+ 5% if mailed to P.O. Box, Alaska, Canada, Hawaii, or outside U.S. = _______
+ $2.50 for billed orders = _______
TOTAL = _______

☐ Payment enclosed. (U.S funds drawn on U.S. bank.)
☐ Bill me. (Add $2.50 for handling).

Mail to: ICCE, University of Oregon, 1787 Agate St., Eugene, OR 97403-9905; ph. 503/686-4414.

LOGO EXCHANGE

Volume 7 Number 4 Journal of the ICCE Special Interest Group for Logo-Using Educators December 1988

Logo Exchange is the journal of the International Council for Computers in Education Special Interest Group for Logo-using Educators (SIGLogo), published monthly September through May by ICCE, University of Oregon, 1787 Agate Street, Eugene, OR 97403-9905, USA.

POSTMASTER: Send address changes to Logo Exchange, UofO, 1787 Agate St., Eugene, OR 97403. Second-class postage paid at Eugene OR. USPS #000-554.

Contents

ICCE Membership (includes *The Computing Teacher*)

	U.S.	Non-U.S.
	28.50	31.50

SIGLogo Membership (includes *The Logo Exchange*)

	U.S.	Non-U.S.
ICCE Member Price	24.95	29.95
Non-ICCE Member Price	29.95	34.95

Send membership dues to ICCE. Add $2.50 for processing if payment does not accompany your dues. VISA and Mastercard accepted. Add $18.00 for airmail shipping.

From the Editor

If It's January Then It Must Be March...

It's a cool January day here on the Oregon coast. From the window, I can see the January rain storms rolling across the Pacific while a toasty warm fire cracles in the fireplace. As I paused to reflect and watch the ever-changing weather, I was struck with the realization that this year of *LX* is drawing to an end. Here I am, putting the final touches on the March issue. (Yes, my Macintosh goes along on weekends at the coast!) Already April articles are arriving and we are beginning to think about how we can make the next year of *Logo Exchange* even better.

While it seems a bit early for reflection, I can't help but note that this has been rather a good year for *LX*. We have gotton onto a reasonable publication schedule so that (hopefully) you are getting your *Logo Exchange* in a timely manner. As always our dedicated columnists have managed to meet the relentless deadlines in the face of holidays and end of term grades. In addition, I have heard from more of you this year with everything from little tidbits to delightful articles.

Now that we have found a stable home at ICCE, it is time to reach out and ask for more of you to participate. During the last two years, we have lost a lot of subscribers — and membership in SIGLogo has not shown any signs of significant increase over the past few months. Clearly, we need to move *LX* in a direction that meets the needs of a larger audience. To do that, we need your help. In hopes that you will mark you calendar now for next year, I am including below the deadlines for submissions to the *Logo Exchange* . Consider contributing —*LX* will be richer for it.

Further, we are going to focus three of next year's issues on a theme. In October, we will have an issue devoted to beginners. We would like articles with suggestions for those teachers just getting started with Logo. Do you have a favorite trick for managing disks? How about the best way to get a class of kids started? What about scheduling computer time in the face of limited equipment? I'd love to hear your ideas. Submit a few lines or a whole article.

In January we will devote an issue to Logo Connections. While Glen Bull's column regularly addresses this topic and other columnists frequently speak to it, I'm sure many of you also have ideas you can share. We will look at connections among versions of Logo, connections between Logo and other software, and connections between Logo and the outside world. Do you have a particularly successful Lego TC logo project? How about driving your favorite robot with Logo? How do you use Logo with other software? Consider sharing your ideas with others.

Finally, in April, we will devote an issue to those with more experience with Logo. Here we'd like to see sophisticated and advanced uses of Logo. Do you have a favorite Logo challange? (Recall Rob's Muir's *InteLXual Challenges?*) Have you used some of the advanced features of Object Logo in an interesting way? Do you use Logo to teach higher mathematics? How about teaching Logo as computer science at the secondary or university level? With this issue we hope to meet the needs of some of our more advanced readers as well as give novices a hint of the power and potential of Logo.

The dates given below are the **final** date that I can accept an article or cover art for a particular issue. Notice that these deadlines are approximately two months before you receive your issue. Also keep in mind that you need to allow a week from the time you drop your submission in the mail before I actually have it in hand.

Month	Final submission date
September	July 3rd
October	July 31st -- Just for Beginners
November	September 4th
December	October 2nd
January	October 30th -- Logo Connections
February	December 4th
March	January 1st
April	January 29th -- Extra for Experts
May	March 5th

So put on your thinking cap, mark your calendar, and consider sharing. Soon the school year will be over and you may find yourself with some time for reflection. Let us hear about your Logo experiences or share with us some of your students' work.

The rain has stopped now and I can see the surf boiling under the grey January sky. It's time for me to put this editorial into PageMaker — and to begin thinking about the April issue. After all, it's almost February so it must be almost April, and before we know it, September deadlines will be upon us!

Sharon Burrowes Yoder
ICCE
1787 Agate Street
Eugene, Oregon 97403
CIS: 73007,1645
BitNet: ICCE@Oregon

Monthly Musings

I ♥ Logo
Part Two: The Beat Goes On
by Tom Lough

Last month's column contained several reasons why people "heart" Logo. In particular, a number of LX readers and students of readers presented their responses to the question, "What is your favorite Logo command and why?" There was enough space only for those who selected the more fundamental commands, such as FORWARD and RIGHT.

In this column, we include additional commands. Thanks again to Sister Mary Grace of Holy Trinity School, Pittsburgh, PA, and to her creative students.

Sarah Petro, Grade 8
Holy Trinity School, Pittsburgh, PA
My favorite Logo command is RANDOM. I think it is fun to use RANDOM because I like to guess what number is going to appear. If you use RANDOM to make a picture, you can keep repeating the program and each time it will be different.

Gary Ratkovich, Grade 6
Holy Trinity School, Pittsburgh, PA
My favorite Logo command is PENUP. I like it best because it moves the turtle without making the extra lines. Instead of having lines not needed, it just moves the turtle where you want it.

Sarah McCloskey, Grade 7
Holy Trinity School, Pittsburgh, PA
My favorite Logo command is PENERASE. I like it because I can erase the mistake I make without using an eraser or an ink eradicator. Many times when I type in my program and command the computer to draw it, if I make an error, I use PENERASE. I think PENERASE is a great Logo command.

Sarah Petro, Grade 8
My favorite Logo command is PENERASE. It is easy to use and is really helpful. All you have to do is type PE and then give commands to the turtle and it will erase your mistakes. I think it is a lot easier to use PENERASE than to clear the whole screen.

Colleen Welsh, Grade 5
Holy Trinity School, Pittsburgh, PA
Of all the Logo commands, my favorite is CLEARSCREEN. To me, it is like magic. I do like to clean the screen when I have too much, and CLEARSCREEN does it so rapidly. How fast I can make something disappear that I do not consider too well done!

Petra Tallo, Grade 5
Holy Trinity School, Pittsburgh, PA.
My favorite Logo command is CLEARSCREEN. I like it because when it erases, it is like a broom sweeping out the dusty garage or like cleaning my bedroom in fast motion. I wish I could clean up my mess as CLEARSCREEN does the computer. [Petra, I wish there was a CLEARSCREEN command I could use to clean up my desk! Maybe I'll try CLEARDESK sometime! T.L.]

Brian Kokkila, Grade 6
Holy Trinity School, Pittsburgh, PA
My favorite Logo command is CLEARSCREEN because it is not just erasing. It gives me a chance to create again, to let my mind use shapes to make many new shapes. It challenges me to think.

Glen Wantje
Holy Trinity School, Pittsburgh, PA
My favorite Logo command is CLEARSCREEN. It makes me feel that I did something right, not wrong. I could try again. I gain confidence. It is a good command for cleaning the unwanted material.

Terry Koller, Grade 8
Holy Trinity School, Pittsburgh, PA
My favorite Logo command is SHOWTURTLE. I like this command because when I am giving this command and I am just starting, I have a challenge before me. I like challenges.

Kevin Helman, Grade 6
Holy Trinity School, Pittsburgh, PA
I am new at Holy Trinity School. I do not know many Logo commands as yet. I have much to catch up with. I do like SHOWTURTLE. It is neat because I can see what I am doing.

Kara Kownacki, Grade 7
Holy Trinity School, Pittsburgh, PA
My favorite Logo command is SETBG. Why? Without this command the computer screen would be boring, just black or green and white. Most people love color. I think this command is cheerful, and brings joy and life into the picture.

John P. Welsh
Holy Trinity School, Pittsburgh, PA
My favorite Logo command is PENCOLOR because by using PENCOLOR we can enhance our program or totally

Monthly Musings -- continued

change it! I feel color is not only an important part of computing, but color is an important part of keeping a different world.

Jackie Jave
Holy Trinity School, Pittsburgh, PA
My favorite Logo command is FULLSCREEN. I like FULLSCREEN because it shows you the entire picture of what you are drawing. Also, you can still type when using FULLSCREEN, but you can't see the words.

Sister Mary Grace
Holy Trinity School, Pittsburgh, PA
My choice Logo command for grades 7 and 8 is LOAD. I feel privileged to LOAD and view the accomplishments of my students and make encouraging comments. In this way, they are motivated to go FORWARD always!

John Java, Grade 5
Holy Trinity School, Pittsburgh, PA
I like PRINT because I can have the drawing on the screen and print it out on paper for others to see my accomplishment.

Heidi Ludwick, Grade 6
Holy Trinity School, Pittsburgh, PA
My favorite Logo command is EDIT. I like EDIT because if I forgot what I typed, I can go in EDIT and see the entire program. I can copy it on paper so that I would have it. I also can enhance my work more efficiently, add to it, make corrections. To me, EDIT is the best.

Missy Krebs
Holy Trinity School, Pittsburgh, PA
My favorite Logo command is EDIT. It lets me correct my mistakes and find errors that I didn't know that I even made! I like it because I want to have nice completed work with no errors.

Mary Upton
Lubbock, TX
I like the BUTFIRST command. When I teach a course and come to BUTFIRST, I always say, "Now we are to the nasty part!" My students love it when I say that!

Ihor Charischak
White Plains, NY
OUTPUT gives me a nice feeling, and allows me to create really powerful elegant complex procedures with recursive calls.

Andy Howard
DeFuniak Springs, FL
I like to work with the MEMBER? or MEMBERP command. I am able to question if something is a MEMBER of a set, comparing it with something else. This command gives me the ability to make comparisons in real life situations and to create models.

Amy Solarczyk, Grade 6
Holy Trinity School, Pittsburgh, PA
All Logo commands are favorites with me. It is difficult for me to choose just one. The reason I like them is because I usually cannot tell people what to do, but when I sit down at the computer and command it what to do, it does it promptly and accurately. It makes me feel in command. How cooperative! [Amy, I have never met you (yet), but I admire your response. I'll bet you are quite cooperative yourself. T.L.]

And there you have it, folks! Have we included a remark about your favorite Logo command? I hope that these comments have encouraged you to muse about the many ways you "heart" Logo.

FD 100!

P.S. I just received information about a new book by Horatio Reggini entitled *Computers: Creativity or Automatism?* in which future roles of the computer are explored. Themes include computers in education, psychology of learning, and artificial intelligence. For more information, write to Emece Ed. S. A., Alsina 2062, 1090 Buenos Aires, Argentina.

Tom Lough
Box 394
Simsbury, CT 06070

Logo Ideas

Part One: Text and LogoWriter: It's Not the Same Old Story

by Eadie Adamson

The world around us can have a strong yet subtle effect upon how we see and express things, often without our being aware of these surrounding influences. Even before computers, neon signs flashed, changed messages, and made moving displays of words and light. Rarely do we think much about this kind of display, although we see it every day in street signs, on top of buildings, on our television screens, and in films. Commercial artists manipulate text displays for newspaper and magazine ads. Electronic transfer of information is proliferating; we're becoming accustomed to seeing words flow across the screen. Artists play with text; sometimes text is their sole focus, as in the recent work of the American artist, Jenny Holzer.

The advent of computer technology has spurred a number of dramatic differences in the way text is used or displayed. Many of these have become so commonplace that we have come to take them for granted. For example, the opening and closing credits of a film or television production are prime examples of changes in text display since the advent of computer technology. Not only is scrolling from the bottom to the top of the screen commonplace, but center or right justification is frequently used. These techniques, also applied in magazine typesetting, are accepted with little notice by the average reader or viewer, yet they reflect the influence of the computer and the versatility it offers in formatting text. Language everywhere is becoming suffused with motion. Occasionally evening newscasts will add text to the screen, echoing the flow of the announcer's words. Our society is becoming increasingly visually-oriented. Even our presentation of printed text appeals to the eye.

Playing with the display of text can be an interesting, informative and thought-provoking exercise. While working with a group of boys in a computer class which focused on writing I decided to ask them to look at the brief stories they had written and to think of how they might control the display of the words on the screen. I wanted them to become more conscious of the ways text is displayed, aware of the new dimensions film, video and computers are adding to our perceptions of words. If the display of the words could be controlled, how would they like to tell their story? Could rapid display of text change or enhance the meaning of an exciting passage? Would a long pause add to the suspense? What happens if a story is displayed only in phrases, mimicking the spoken word? Who would be in control: the writer or the reader or the computer or all three? How do you tell a story when you can control how the text appears on a screen?

Getting Started with a Story

The writing program the students were using, AppleWriter, is ProDOS based and its files are saved in ASCII format. Since LogoWriter 2.0 for the Apple could access text (ASCII) files, it was easy to take the boys' word-processed stories and load them into LogoWriter.

First, we loaded LogoWriter, selected a new page, and moved to the flip side. Then we used the Down key to get to the Command Center. Next we removed the LogoWriter disks and put our data disks with the stories we had written in the drive. In the Command Center we typed:

```
LOADTEXT "name.of.story.file
```

Our stories were loaded onto the flip side of the page. We replaced our LogoWriter disks and named the page STORY. (Caution here: some stories will be too long for the 8K LogoWriter page. You'll need to break those into separate files if you want to load the entire text.)

Now it was time to think of the stories in a new way. We began with a few simple experiments.

Experiment with PRINT and INSERT

PRINT puts text on the screen and moves the cursor down to the next line; INSERT places text on the screen but leaves the cursor at the end of the line. The next piece of text, whether INSERTed or PRINTed joins up with that line.

Try this:

```
INSERT "Hello,
INSERT [how are you?]
```

You will see:

```
Hello,how are you?
```

Notice that there is no space between the "Hello," and the word "how". It is necessary to add a space: INSERT CHAR 32. CHAR 32 is the ASCII value of the space bar. If we are planning to insert spaces often, we need to write a procedure like this:

```
TO SP
INSERT CHAR 32
END
```

Logo Ideas -- continued

When inserting or printing lines, sometimes one also wants to add a carriage return:

```
TO CR
INSERT CHAR 13
END
```

Now try this:

```
INSERT "Hello,
SP
INSERT [how are you?]
CR
```

Now you should see:

```
Hello, how are you?
■
```

Actually, if we had used PRINT for the last part of the line, CR would have been unnecessary. PRINT puts text on the screen and brings the cursor down to the next line, unlike INSERT which leaves the cursor at the end of the inserted text. PRINT [how are you?] has the same effect as INSERT [how are you?] CR.

Taking the first paragraph of the story on the flip side, each boy made it into a procedure and wrote a program so that it would appear on the screen, using PRINT and INSERT first.

Wait!!

So far, this doesn't seem to be a particularly unique approach but, as Marian Rosen says, "wait, there's more!" Text can be paced too. Think about how you speak. This can be modeled by putting text on the screen, adding WAIT and a number can change the tempo of the words appearing on the screen. You can think of it as music:

```
INSERT "Hello,
WAIT 10
SP
INSERT "how
WAIT 5
SP
INSERT "are
WAIT 7
SP
etc.
```

The words still spill out across the screen, but WAIT can slow the text so that it appears at the rate of the spoken word. The words could also be inserted as syllables to slow things down further. Think of the potential when telling a dramatic story!

Spilling Out Words

Now, what about making text appear as if it were being typed or sent by electronic mail, making it spill out on the page letter by letter? This procedure will insert a word one letter at a time:

```
TO IN :WORD
IF EMPTY? :WORD [STOP]
INSERT FIRST :WORD
IN BUTFIRST :WORD
END
```

IN inserts a the first letter of the word, then the second (the FIRST of the BUTFIRST of the word), and then the third, and so forth until there are no more letters.

Since IN works on one word at a time it can be tedious to use if programming a long sentence in this way. I gave my students an additional tool to insert a list of words in the same fashion. INS inserts a list of words one letter at a time. INS uses IN to insert a word. INSERT CHAR 32 puts a space between each word. (You could substitute the SP procedure, above, for INSERT CHAR 32.) INS continues the process until all the words in the list have been inserted.

```
TO INS :LIST
IF EMPTY? :LIST [STOP]
IN FIRST :LIST
INSERT CHAR 32
INS BUTFIRST :LIST
END
```

Inserting WAIT and a number in the IN procedure will slow down the appearance of the letters on the screen. Students can play with the timing until they find one they like. One of my students, Ted, elected to make his text appear as if it were typed, complete with the addition of a sound for each keystroke.

Here are Ted's adaptations for inserting text with sound:

```
TO SOUND
TONE 715 5
END
```

```
TO IN :WORD
IF EMPTY? :WORD [STOP]
INSERT FIRST :WORD
SOUND   <—This makes a sound as a letter is inserted
IN BUTFIRST :WORD
END

TO INS :LIST
IF EMPTY? :LIST [STOP]
IN FIRST :LIST
SOUND    <—This makes a sound as a space is inserted
INSERT CHAR 32
INS BUTFIRST :LIST
END
```

Part of Ted's text procedure looked like this:

```
TAB
INS [Lego Logo, or The Computer Comes
to Life]
PRINT CHAR 13
PRINT CHAR 13
TAB
INS [My dad had a computer for
   his store, and I played games, and
   did my homework on it.  But I
   never could really understand
   computers.]
PRINT CHAR 13
```

Notice that Ted used PRINT CHAR 13. He could have used PRINT [ ] or PRINT " for the same result. Ted also did not need to add a WAIT to slow down the text. The SOUND procedure creates the pause before the next letter appears.

Exploring text in this way is an interesting exercise for student and teacher alike. Students will think of interesting variations on their own without much need for tools other than the ones outlined above. You can expect interesting ideas to emerge!

If you try these ideas that I have outlined above, please send me any interesting variations your students develop. We'll try to include samples in another column. I'd particularly love to hear from some people who experimented with this idea with Terrapin's Logo PLUS.

To be continued.....

Eadie Adamson
Allen Stevenson School
132 East 78th Street
New York, New York 10021

The Logo Institute
June 23 - 30, 1988
The Logo Institute Workshops
June 23-24, 1988

Lesley College, Cambridge Massachusetts
Immediately following NECC (June 20-22)

What are your plans for professional renewal this summer?

Consider The Logo Institute (sixth annual)
- experience a seven-day immersion in a learning environment for Logo-using educators from the US, Canada and other countries
- use the full scope and power of Logo and its educational philosophy to create programming, research, teacher education and curriculum projects
- (optional) receive three graduate credits from Lesley College

Participate with An Outstanding Faculty
- Dan Watt and Molly Watt, Logo Institute founders
- Ricky Carter, Lesley College Faculty
- Scholars in Residence:
  Logo Exchange columnists Eadie Adamson and Gary Stager
- Faculty will offer lectures and demonstrations, lead short courses, provide technical assistance and coaching, and support participants in completing their own creative projects.

Attend The Logo Institute Workshops only
(for those unable to attend the entire Institute)
- Two days of hands-on workshops led by Logo Institute faculty and other leading Logo educators
- Hands on workshops include:
  - Supporting mathematical learning with Logo
  - Supporting language learning with Logo
  - Assessing Logo learning in classrooms
  - Getting a jump start with Lego-Logo
- Workshop participants may register for one optional graduate credit

For further information and registration materials, contact:
Ricky Carter, The Logo Institute
Lesley College, 29 Everett Street
Cambridge, Massachusetts 02138
(617) 868-9600 x370

Stager's Stuff

LEGO® TC logo Ballet: The Baryshnikov of Bricks

by Gary S. Stager

For the past two years I have been working extensively with LEGO TC logo, teaching a wide variety of audiences to use these exciting new materials. LEGO TC logo inspires such an open-ended environment that I am inclined to dream up all kinds of crazy projects. It's been a lot of fun springing my ideas and inventions on my friends at LEGO. I suspect that they cringe when I call or they see me heading towards them at a conference (with some new creation under my arm).

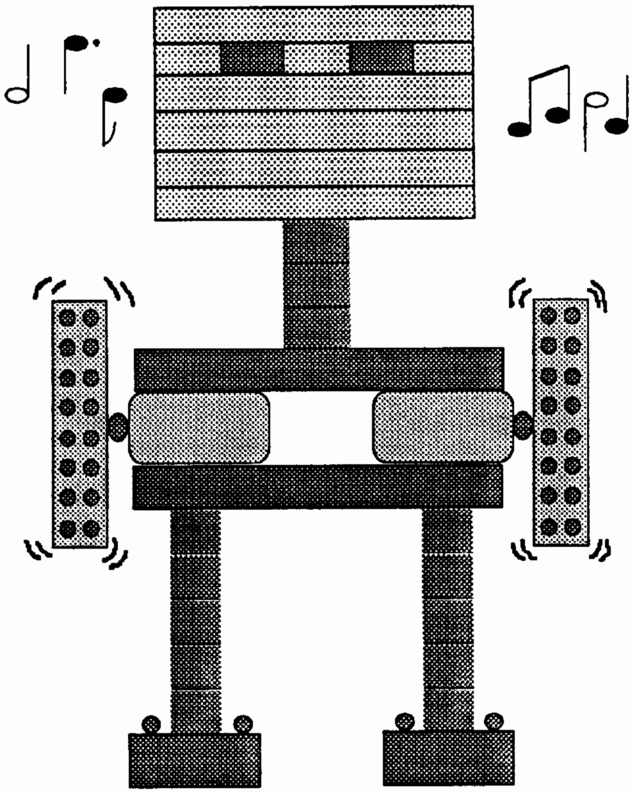
One of Seymour Papert's criticisms of staff development is that too many teachers focus on obtaining a new technique or idea that they can quickly take back to their classrooms rather than immerse themselves in the activity and learn for their own sake. Adults, like children, learn best when they are being challenged and actively engaged in an activity that they find meaningful (fun doesn't hurt either.) Papert makes the point that a teacher who experienced the satisfaction of learning, programming or building something new will be much more valuable to their students than a teacher who is a spectator. For this reason, this month's article will describe a project that was totally egocentric. I had a ball building and programming my LEGO ballerina and I think that the project may **even** have some educational relevance.

The LEGO Ballet project combines some of the most powerful aspects of Logo with the creativity of LEGO building in a microworld for children (big children too). A microworld should be fun, simple, flexible, and contain powerful ideas. The LEGO Ballet project is certainly fun and contains powerful ideas such as: sequencing, dance, music, choreography, programming and robotics. After a good typist enters and saves the procedures for the programming shell the activity is simple and flexible enough to create an infinite number of ballerinas and ballets. The project is syntonic (related to one's interests and experiences) in that it combines LEGO building with Logo programming, music and dance. Perhaps the most unique aspect of this LEGO TC logo activity is that the product is not a truck or machine and therefore the project appeals to girls and dance aficionados.

Build the Bride of Baryshnikov

To use this activity in the classroom, each group of students should build a ballerina. The ballerina can be designed any way their imaginations dictate although it should contain some motors (for motion) and/or lights. My ballerina has light-brick eyes that blink and arms that spin. How about a ballerina that walks, is on roller skates or breakdances?

Remember that the LEGO TC logo interface box has only three motor ports (A, B, C) and six light ports (0-5). Motors and lights may be used in series (plugged into the same port) if they will **always** be turned on and off simultaneously.



The Tool Procedures

The following procedures are necessary for teaching your ballerina to dance. You will probably want to provide them as tools for your students to modify and expand. The CHOREOGRAPH, LEARN, TEACH, NAME.DANCE, REMEMBER, UNDO, and DEFINE procedures form the programming shell which must be used by each group of LEGO/logo choreographers. CHOREOGRAPH is the superprocedure which makes everything else work.

```
To choreograph
make "dance []
cc
learn
choreograph
end
```

```
To learn
teach readchar
learn
end

to teach :key
if number? :key [run (list
   word "dance :key)
   remember (word "dance :key)]
if :key = "a [a]
if :key = "b [b]
if :key = "c [c]
if :key = "d [d]
if :key = "e [e]
if :key = "f [f]
if :key = "g [g]
if :key = "h [h]
if :key = "i [i]
if :key = "j [j]
if :key = "k [k]
if :key = "l [l]
if :key = "m [m]
if :key = "n [n]
if :key = "o [o]
if :key = "p [p]
if :key = "q [q]
if :key = "r [r]
if :key = "s [s]
if :key = "t [t]
if :key = "u [u]
if :key = "v [v]
if :key = "w [w]
if :key = "x [x]
if :key = "y [y]
if :key = "z [z]
if identical? :key "S
   [name.dance make "dance [] stop]
if :key = "* [run :dance stop]
if  (ascii :key) = 27 [undo stop]
remember :key
end
```

Note: You can include as many of these conditionals as you need for your dance.

Note: IDENTICAL? is a LogoWriter and LEGO logo predicate that tells if two words or lists are exactly the same, taking into account upper and lower case characters. This allows me to check for capital S. The last conditional statement checks for the Esc key (ASCII 27).

```
to name.dance
cc
type [What number (0-9) dance is
    this?]
make "key readchar
if not number? :key [name.dance]
define (word "dance :key) :dance
type (sentence char 13 (word
   "dance :key) "DEFINED char 13)
type [Ready to learn a new dance!]
end

to undo
cc
type [I will now perform all but the
    last step you are teaching me.]
run butlast :dance
cc
repeat 2 [tone 800 10]
type [This is the last step I left
    out...]
run (list last :dance)
cc
type [Are you sure you wish to
   forget the last step?]
if readchar = "y [Make "Dance
   butlast :Dance]
end

to remember :step
make "dance lput :step :dance
end

to define :name :stuff
top
print sentence "to :name
print :stuff
print "end
print []
end
```

The Nuts and Bolts

The program for the LEGO Ballet project makes use of some of Logo's most powerful features and highlights a few minor shortcomings of the LEGO logo software. Logo has the ability to treat program as data and therefore can have procedures create and modify other procedures. This powerful feature is employed in the program in such a way that it is transparent to the child who needs only to press keys and concentrate on creating his ballet. The LEGO Ballet program is similar in many ways to the TEACH program included on

Stager's Stuff -- continued

the Apple Logo Toolkit. The TEACH program was a form of INSTANT in which kids could command Logo with a single keystroke. In TEACH the computer keeps track of the keystrokes and assembles them into a procedure.

Each group of choreographers need to create Logo procedures in the Procedure Center (flip side) of the page which are titled with lowercase letters of the alphabet. The procedures need to be named from a - z so they can be dynamically linked to letters on the keyboard (you do not need to use all 26 letters). Each letter-titled procedure should do one thing (turn on a motor, flash a light, play a tone) and should independently cause an action if you typed the procedure's name in the Command Center. For example:

```
to a                  to f
left.arm              right.eye
end                   end

to o                  to m
alloff                tone 200 10
end                   end

to l                  to i
right.arm             repeat 2
rd                        [left.arm rd]
left.arm              end
rd
end
```

Then the TEACH procedure needs to include a conditional line for each letter programmed. (Extra lines can be deleted if they are not used.) For example:

```
if :key = "a [a]
```

Often, programming demands tradeoffs. One such tradeoff caused me to ues a conditional for each key in the TEACH procedure instead of using RUN (LIST :KEY) once. LogoWriter and LEGO logo have no way to catch errors or to check if a procedure is already defined. Therefore, if the user pressed a letter which didn't have a corresponding procedure the program would crash. By having a conditional for each letter procedure you make the TEACH program more crash-proof.

Naturally, procedures like **a**, **f**, and `i` (above) call subprocedures which need to be written. These procedures will use LEGO logo primitives such as: TALKTO, ON, OFF, ALLOFF, FLASH, RD, and TONE. For example:

```
to left.arm           to right.eye
talkto "a             talkto 2
onfor 10              flash 4 2
end                   end
```

The TEACH procedure contains a subprocedure, REMEMBER whose job it is to add the last key pressed to a list of steps in the current dance variable (:DANCE.)

Many versions of Logo have a DEFINE primitive which creates a new Logo procedure. LogoWriter and LEGO logo do not contain such a primitive so I have had to create one. Since LogoWriter and LEGO logo's procedures are text on the flip-side of the page, DEFINE need only print the title and end line of the procedure with the subprocedures printed in the middle.

Teach Your Bricks to Boogie

I have attempted to make the LEGO Ballet program as powerful, flexible, and user-friendly as possible. Included in the program is the ability to review your current dance, undo the last step, name a dance, and assemble a collection of dances and steps into a ballet. Letter keys on the keyboard trigger individual dance steps and numeric keys (0-9) allow you to perform dances you have already choreographed.

Glossary of Terms

Dance Step: an individual movement, sound, or action for your ballerina to perform. Steps are executed by pressing a letter key on the keyboard. There must be a procedure beginning with a corresponding letter of the alphabet for each step. *e.g.*, `if :key = "a [a]`

Dance: a sequence of individual dance steps and/or previously defined dances. Dances are performed when a number key is pressed.

Ballets: collections of LEGO logo dances and are defined in the same way as dances. A ballet may be executed by typing the name of its procedure or by pressing a programmed number key.

Steps for Creating a Logo Ballet

All of the LEGO Ballet project MUST be executed on the procedure side of the page! Press -D to enter the Command Center and -U to enter the Procedure Center.

(1) Create as many dance steps as you wish (with titles a, b, etc...).
(2) Add a conditional line to the teach procedure per dance procedure. E. g., if :key = "a [a]

(3) Type CHOREOGRAPH to begin teaching your ballerina.
(4) Press a letter key to perform a particular dance step. You may wish to print a list of what each procedure/key does.
(4a) Press the ESC key to undo the last step. The program will perform the entire dance and ask if you are sure that you want to erase the last step.
(4b) Press the * (asterix) key to view the dance thus far.
(4c) Press Shift-S to stop teaching Logo a dance step.
(5) After you press Shift-S you will be asked to name the dance. Choose a number 0-9 and be sure that you haven't already named a dance with that number. You will then be told, "Dance# Defined - Ready to Learn a New Dance!"
(6) After you have defined a dance you can perform the dance by pressing the number you used to name it. You are now adding an already choreographed dance to the new dance you are creating. An already defined dance is treated as one step when you use the undo key.
(7) Repeat steps 4-7 as many times as you wish. Press -S to stop the LEGO ballet program.

If you have performed the steps outlined above you should have noticed that a new Logo procedure appeared on the page for each dance you completed (i.e., TO DANCE1, TO DANCE2, etc.)

You may thrill your friends and neighbors by typing one or more of these procedures in the Command Center or by combining these dance procedures in other superprocedures if you want to have more than 10 dances on your page. For example:

```
to Nutcracker
Dance3
Dance1
Dance4
Dance3
end
```

To create new dances and ballets copy all of the procedures from CHOREOGRAPH to the bottom of the Procedure Center to a new page.

I hope you enjoy this LEGO Ballet activity. Remember that any LEGO Technic model (not just ballerinas) can be controlled by these procedures. Please send me descriptions of any offbeat LEGO TC logo projects created by you or your students and I'll share them in the *Logo Exchange*. My future projects include a LEGO model to kick a field goal or shoot a basket. Will anybody beat me to it?

Gary S. Stager is the Vice President of SIGLogo and Director of Training for the Network for Action in Microcomputer Education in New Jersey. A self-proclaimed Logo zealot Gary has been heard to say that "Logo is his best friend." He is a frequent speaker at conferences throughout the U.S. and is available for staff development, consulting, or speaking in your school district. Gary can be reached at:

Gary S. Stager
NAME
Fallon Education Center
51 Clifford Drive
Wayne, New Jersey 07470
CIS: 73306,2446
Applelink: K0331

LogoWriter Home Pack Now Available

Logo Computer Systems, Inc., creators of LogoWriter recently assigned exclusive distribution rights for LogoWriter Home Edition to Soft-Kat. The newest version of LogoWriter is being marketed to parents who are interested in providing interesting, stimulating and fun activities for their children to do on their home computers.

The LogoWriter Home Edition includes the LogoWriter software; a 76-page book, *Introduction Logo Writer* containing easy-to-follow, step-by-step activities which help children become familiar with LogoWriter and a 32 page Quick Reference booklet.

LogoWriter Home Edition is available at major software retailers nationwide. LogoWriter Home Edition can be used with Apple II's (128K) and IBM and compatibles with 256K.

For more information,
call 1-800-321-5646.

Creating Moving Clock Hands

by Diane Miller

I have been teaching Logo for three years at a small (about 40 students) academically-oriented private school. Near the end of last year, the students in the grade 3-5 classroom became bored with procedures that make static pictures and began to ask about creating animation. The unrest quickly spread to the upper-level (grade 6-9) classroom. This seemed an excellent path to explore, so I devoted the next weekly discussion period in each class to the subject. We talked about how an animated picture is made. Even the younger children were familiar with the idea of a rapid sequence of pictures, each slightly different from the last, blurring together into a perception of smooth motion. The children quickly figured out that to animate a Logo picture you have to draw it, hold it for a while, erase it (or part of it), and redraw it in a slightly different place. They also discovered that in order for the movement to look smooth, the drawing/erasing times have to be fast compared to the hold time, and that this limited them to straight lines, and not too many of them. (We were using Apple Logo.)

We discussed ideas for an animation project in class and decided to try to make moving watch hands. Everyone was enthusiastic about the idea, so we decided that each pair of students, working together at a computer, would work on the same task. The idea was an extension of our last project, designing watch faces, in which the students had made hands, usually without much thought to where they were pointing. (I had challenged the older ones to make the positions correspond to the positions real watch hands would have — i.e. to figure out that for any given position of the hour hand there would be only one possible position of the minute hand, or for any given position of the minute hand there would be 12 possible positions of the hour hand.) It was a logical extension of these static watch faces to try to make the hands move.

In figuring out how to simulate the motion we discussed how it is done in the liquid crystal watches that display moving hands instead of flashing numbers. These watches used the same principle by which the digits are displayed, namely by sending a voltage to different pre-set patterns of electrodes coated on the inside of the glass. In the case of pseudo-moving hands the pattern is a sunburst of sixty different hands each pointing at one of the minute hashmarks, and each successive single hand pattern is illuminated in turn. This seemed easy to do with Logo, by drawing a line, then erasing it, and turning the turtle to the direction for the next line.

We decided that a program to run the hands would have the hierarchy:

```
move hour hand
      |
      v
move minute hand
      |
      v
move second hand
```

We therefore decided we should start with the second hand. I firmly believe in thinking through a project and writing out its logic in English first, and then translating it into Logo. So we raced to the blackboard and with the help of a little body English the children came up with a logical sequence of commands for the second hand:

For the second hand:

draw it
hold it about one second
erase it
rotate the turtle 1/60 of 360 degrees
repeat

The Logo commands (using Apple Logo syntax) are

```
TO SECHAND
FORWARD 60
WAIT 60
PENERASE
BACK 60
PENDOWN
RIGHT 6
SECHAND
END
```

The hands look much better if the turtle is hidden, but we knew that we would have higher-level procedures and they would be the place to put the HIDETURTLE command, so we decided to do the HIDETURTLE at the command (top) level at this stage.

Most children accomplished this much pretty easily. Some even timed the resulting hand and adjusted the number after WAIT to give exactly a one-minute sweep.

Now we tried to add the minute hand. We thought it would be simple — an almost identical procedure, making a slightly shorter hand and running 60 times slower. But after a while we realized there were two problems:

(1) We only had one turtle to do two hands which run at the same time, so we could not have two completely independent procedures.

(2) The second hand would "run over" the minute hand and erase it.

This took some head scratching, but we finally figured out that we could solve both problems at once by simply picturing the second hand as pulling the minute hand along behind it, one click for each full 360 degree sweep. In other words we could redraw the minute hand just after the second hand passes and erases it after each sweep. This corresponds nicely to our original hierarchy, with the minute hand procedure calling the second hand procedure. Now we realized we needed SECHAND to repeat 60 times, so we removed the recursive call in SECHAND so that it became:

```
TO SECHAND    (modified version)
FORWARD 60
WAIT 60
PENERASE
BACK 60
PENDOWN
RIGHT 6
END
```

Then we added the procedure:

```
TO SWEEPSECHAND
REPEAT 60 [SECHAND]
END
```

The logic for the minute hand turned out to be even simpler than for the second hand:

For the minute hand:

draw the minute hand (returning turtle to center)
turn turtle 1/60 of 360 degrees
run SWEEPSECHAND

The Logo commands are:

```
TO MINHAND
LINE 50
RIGHT 6
SWEEPSECHAND
END
```

The procedure TO LINE is:

```
TO LINE :SIZE
FORWARD :SIZE
BACK :SIZE
END
```

We decided not to make the minute hand procedure recursive, but to repeat it 60 times, so an hour hand could be added by the same logic.

```
TO  SWEEPMINHAND
REPEAT 60 [MINHAND]
END
```

At this point we sped up the sweep by temporarily changing the WAIT 60 in SECHAND to WAIT 6, to watch a full cycle of the minute hand more easily.

When we did so a large bug loomed before us. At certain positions the minute hand was not fully erased by the second hand. A shadow-like line of dots was left on the screen. It looked as though somehow the turtle's heading must be just a tiny bit off after it had gone around 360 degrees. But how? (Was Logo's ability to do math that bad?) And more importantly, what could we do about it?

The answer came by accident, as usual. I was sure that the previous week someone had hands going around that hadn't left these shadows. We found the file and sure enough no shadows, but they had used the same logic as the groups that had shadows! After much searching we finally found a difference — they had made the minute and second hands the same length. Could this be important? We tried the following exercise:

```
CLEARSCREEN
PENDOWN
RIGHT 18 <-- one of the worst
FORWARD 50   headings for the
BACK 50      shadows
PENERASE
FORWARD 60
BACK 60
```

A ragged line remained as the first line (50 turtle steps) was only partially erased by the 60 turtlestep line. Then we traced over the original line with one of the same length:

```
FORWARD 50
BACK 50
```

and the line was fully erased!

The simplest solution to get a fully erased minute hand was to make it and the second hand the same length. This satisfied many of the children. The more particular ones figured out that they could add a jump 6 degrees left at the end

Creating Moving Clock Hands -- continued

of MINHAND and erase the hand in its old position with a 50 turtle-step length line:

```
TO MINHAND     (modified version)
LINE 50
RIGHT 6
SWEEPSECHAND
LEFT 6
PENERASE
LINE 50
PENDOWN
RIGHT 6
END
```

When we started to add an hour hand, we realized that two hands would sweep over it and erase it; it would have to be redrawn each time the second hand passed over it. We tried staggering the hour hand 3 degrees, so that it was in between the positions of the other two hands, but the turtle's eraser is a little clumsy, and the inner part of it (about 80%) was erased anyway. We decided no one would want to watch the display long enough to see the hour hand move, so we settled for a static hour hand, staggered 3 degrees, that survived the passing of the other two hands as best it could:

```
TO HOURHAND
RIGHT 3
LINE 40
LEFT 3
END
```

The sub-procedure to run the hands is finally reached:

```
TO HANDS
HIDETURTLE
HOURHAND
SWEEPMINHAND
END
```

The final hierarchy is:

$$\begin{tikzcd}
& \text{HANDS} \arrow[dl] \arrow[dr] & \\
\text{HOURHAND} \arrow[d] & & \text{SWEEPMINHAND} \\
\text{MINHAND} \arrow[d] & & \\
\text{SWEEPSECHAND} \arrow[d] & & \\
\text{SECHAND} & &
\end{tikzcd}$$

I was surprised how nice the resulting display looks, and the children were very pleased with themselves.

This project led the children to a number of interesting discoveries:

(1) They had to figure out the basic principle of animation.
(2) The design of these procedures is complex compared to the drawing of simple pictures which had comprised most of the students' Logo experience. Therefore they learned the advantages of thinking through the procedures and writing them out in English, and then translating them to Logo.
(3) The students were led to a hierarchical structure of procedures in constructing the hands.
(4) They discovered the use of the WAIT command to make the hands run at the correct speed.
(5) They learned the trick of speeding up the execution to more easily watch the full cycle of the minute hand.
(6) They had to solve the problem of making one turtle draw two hands which moved at different rates.
(7) They had to find and fix the bug in which the longer second hand in PENERASE mode did not always fully erase the shorter minute hand which it apparently overlaid exactly.

This project led some of the students to the idea of making a watch face with flashing digital numbers, an interesting project which some of them will pursue.

Diane Miller, Computer Teacher
Guadalupe Private School
4614 Old Redwood Highway
Santa Rosa, CA 95401
(707) 546-5399

About the cover

The cover is the work of Davis Trassman of Winkelman School in Glenview Illinois. It was submitted by Gail Estes who noted that this was David's second day with LogoWriter. They were using a Turtle Compass to practice the **SETH** command.

...and you thought only *your* kids liked wrapping!

Little Kids and Logo

Beware the Ide(a)s of March!
by Leslie Thyberg

A terrific survival tactic as a teacher is to provide a variety of opportunities and resources to students to stimulate and tantalize them. When I was in my own classroom I would leave a Logo manual out for the children to browse through for leisure or curiosity. Inevitably someone would come across some fascinating new primitive that I have never used, never heard of and had no notion of how to utilize! My consistent response in such situations was, "Why don't you experiment with it and see what happens"? Most of the time something really great happened and this became the springboard for the rest of the class to try something new and different. To facilitate their experimentation, I provided my students with a shape utility package. What goes into a shape toolbox? Simple, anything that is of use to you or your students.

A simple example of some of the utilities placed in this file would be procedures to draw big and small squares. Every Logo manual I have ever used has a chapter or section on variables. Two obvious examples are SQUARE and TRIANGLE procedures.

```
TO RIGHT.SQUARE.OF.ANY.SIZE :SIDE
REPEAT 4 [FD :SIDE RT 90]
END

TO EQUILATERAL.TRIANGLE.RIGHT. :SIDE
REPEAT 3 [FD :SIDE RT 120]
END
```

So what do you do with these? Draw, of course. What to draw? Why not a classroom carnival of animals?

Ed Emberley's *Drawing Book of Animals* is an excellent example of playful geometric and artistic exploration using common shapes and letters to draw animals. Looking for an integrated curriculum unit? This is it! Have your students listen to and move to Camille Saint Saen's Carnival of the Animals. Creative writing and creative dramatics are just two possible expressive outlets. The real fun begins when Logo shapes are used to create a class circus or carnival. Emberley combines a series of squares and rectangles to make a very convincing gorilla. He makes an elephant out of a square, a triangle, and several half circles. The basic principles for generating such animal art are to remember: you can always change the color or the size, or make one part bigger or smaller. Similarly, shapes of animals can readily be embellished by putting grumpy, sad, happy, emotional faces on the creatures.

One afternoon some time ago I had a youngster who was reading a Logo manual who happened to come across the DOT command. "What does this do?" he queried."Let's see", I responded. While the finer points of understanding the notion of an x and a y coordinate were far beyond this child's grasp, he used DOT very creatively. For example, the following is a simple illustration from Emberley's text that the little boy adopted. What is it? A group of ants headed toward a hole!

. . . . o

Conversely, this is a group of ants coming out of a hole!

o . . . .

By changing the pen color you can follow Emberley's lead. One dot can be a basic brown ant, the next can be a colored dot (an ant wearing a sweater!) The divergent thinking skills and brainstorming possibilities are endless.

What can you do with the STAMP primitive? Lots! Make a "Y" shape and stamp it on the screen. For Emberley, this is a snake's forked tongue. In Logo, Y can be stamped on the screen and a snake figure added using curve utilities. Curves and arcs can be generated following nearly any Logo manual for instructions.

Make a D shape and stamp it on the screen. Emberley uses this as the head of a caterpillar. Stamp a smiley face for the beginning of a happy polliwog - a simple curve or arc can be added to make it complete. Emberley 's book is loaded with examples of how to use semicircles for porcupines, birds and elephants; or squares and rectangles for lions, elephants, and gorillas. Meg Palmer, a teacher at Pittsburgh Urban Christian School, is using Emberley's book as a guideline for her fossils unit in science. The students are generating their own fossils after closely studying the geometric designs of fossils at the local museum and from classroom library resource books. The kind of open-ended problem-solving and divergent thinking skills that can come from such simple, yet creative visualization tasks are only limited by the imagination. Enjoy the parade of animals.

Emberley, Ed. (1970). *Drawing with Animals.* Boston: Little, Brown and Company.

Watt, Daniel. (1983). *Learning with Logo.* New York: Byte Books/McGraw-Hill.

Dr. Leslie F. Thyberg
Chatham College
Woodland Road
Pittsburgh, PA 15206
Applelink: ALS 038

Logo LinX

Turtle Archetypes

by Judi Harris

Ready for some associative thinking? What do you think of when I say "turtle?" Logo, of course...but what else?

Turtles are wonderful "objects to think with," as Dr. Papert says. But they also can be interesting objects to think *about* in cultural, metaphorical, and literary contexts. What patterns emerge when we view the turtle as a cross-cultural symbol? The search for answers to this question can inspire in-depth comparative historical, anthropological, and philosophical inquiry for your students.

The Orient: Turtle as Oracle

Ancient Chinese tradition depicted the tortoise as one of four sacred creatures, along with the dragon, the phoenix, and the ky-lin. Each creature represented one of four metaphoric elements from which physical form springs; the dragon symbolized fire, the phoenix, air, and the ky-lin, earth. The spiritually-endowed turtle embodied the water element; also the winter season, the yin prinicple, the northern region, and the color black, the primordial chaos out of which the earth was born. According to this tradition, the turtle symbolically supported the world, with its four feet placed at the four corners of the earth.

The dragon and tortoise were used as battle symbols, since the Chinese felt that they represented indestructibility; the dragon is unable to crush the turtle, and the turtle is unable to reach the dragon to do it any harm. Therefore, the tortoise was called the Black Warrior, and in that role, it represented strength, endurance, and longevity. Taoist tradition sees the turtle as symbolizing the Great Triad, or the universe, with its dome-shaped back as the sky, its bottom shell as the water, and the body in the middle as man, the mediator between heaven and earth.

Japanese folklore portrays the tortoise as support for the abode of the Immortals and the Cosmic Mountain. Kumpira, god of sailors, used the turtle as his symbol, as did the goddess Benten. Seen in ways similar to those of Chinese tradition, the Japanese turtle represented longevity, support and good luck.

Africa and India: Turtle as Female Creator

Many ancient cultures associate turtles with water. Some concentrate upon the fertility that this connection can suggest. Sumerian mythology considers the sacred tortoise to be Lord of the Great Deep. Ancient Egyptian texts portray the turtle as drought and an enemy of the sun god, Ra. Two tortoises symbolically measure the flood waters of the Nile river, appearing in paintings with the sign of the Great Scales. Nigerian folklore depicts the turtle as procreative and decidedly feminine; according to this tradition, its shape suggests female reproductive and sexual organs, and is an emblem of lubricity.

According to Hindu teachings, the turtle was the first living creature, Kasyapa, or the North Star. This progenitor is an avatar of Vishnu the Preserver, and represents the power of Earth's waters. Paralleling Chinese tradition, Indians see a tortoise's lower shell representing the terrestrial world, and its upper shell as the celestial world. Yet unlike Oriental notions, Hindus see the earth symbolically resting on the back of an elephant, which is, in turn, supported by a tortoise. According to this tradition, the elephant is male and the tortoise female, together representing the two balanced creative powers (much like Chinese yin and yang) that bring about physical form.

Ogden Nash comments upon this attribute as a humorous paradox.

The turtle lives 'twixt plated decks
Which practically conceal its sex.
I think it clever of the turtle
In such a fix to be so fertile.
-Ogden Nash, "The Turtle"

Western Notions

Greek and Roman mythology also portray turtles as representative of the feminine principle and water's fertility. Aphrodite and Venus, who were said to have risen from the sea, embodied this fertile principle in human form. The turtle is mentioned in the Old Testament as the voice of Spring.

Rise up, my love, my fair one, and come away.
For, lo, the winter is past, the rain is over and gone;
The flowers appear on the earth;
The time of the singing of birds is come,
And the voice of the turtle is heard in our land.
-Song of Solomon, 2:10

Early Christian symbolism showed the turtle as illustrative of modesty in marriage. Women were encouraged to live in the home as the turtle lives protected within its shell. Yet the turtle also appeared in early Christian art as a symbol of evil, contrasted with the vigilant cock.

American Turtles

In native American literature, the turtle is sometimes depicted as an earthly coward, braggart, or sensualist. Yet the great Cosmic Tree grows out of its back. The Iroquois, like the Chinese and the Hindus, see the earth as resting on the turtle's shell. Delaware and Algonquian Indians also speak of the turtle as an earth-bearer; the Delaware specifically name the *cistudo carolina*, or box turtle, in this role. Therefore, turtles are also considered to be sacred, beneficent beings.

Yet turtles, according to native American custom, can be dangerous or foolish. Delaware and Shawnee Indians tell a tale about several men who climb aboard the back of a great sea turtle, and all except one are unable to get off before the turtle submerges itself in the ocean. Yet Gluskabe, protagonist for a series of northeastern Algonquain stories, plays practical jokes on the tortoise, who is similarly the butt of other pranks related through tales of other northeastern tribes. Still, many central Algonquain animal stories depict the turtle as racing other animals and winning.

This theme is carried through many modern Amazon mestizo and Guianan stories about the *jabuti*, a small, clever, mischievous land turtle that emerges safe and victorious from every contest with larger, stronger animal rivals. The turtle frequently matches wits with a ferocious but stupid jaguar, who, for example, is killed by the tortoise falling from a tree into a hole in the ground. When the jaguar reaches into the hole and gets hold of the turtle's arms, the clever turtle tells him that he has only grasped the roots of a tree. The jaguar, of course, believes him, and releases the devious creature.

Perhaps one of the most popular native American stories is one that will undoubtedly seem familiar. In it, the turtle challenges the deer to a race. The turtle wins because it has arranged with other turtles to deceive the faster animal.

Cross-Cultural Patterns

As J.E. Cirlot (1962) suggests, in all traditions, the turtle is a symbol for earthly existence. Its slowness suggests natural evolution; its round and square shape represents material substance, as opposed to transcendent forces. The turtle, it seems, embodies the *expression* of cosmic forces as physical form.

It should not surprise us, then, that the Logo turtle is a symbolically powerful, semi-concrete object-to-think-with in a microworld of mathematical ideas. It is easy to befriend a supportive earthly archetype, especially one that makes powerful (even cosmic) ideas manifest in patient physical terms.

References

Circlot, J. E. (1962). A dictionary of symbols. New York: Philosophical Library.

Cooper, J. C. (1978). An illustrated encyclopaedia if traditional symbols. London: Thames and Hudson.

Leach, M., ed. (1950). Funk & Wagnalls standard dictionary of folklore mythology and legend. New York: Funk & Wagnalls Company.

Judi Harris taught students in Philadelphia-area elementary through graduate schools to use computers in teaching and learning for six years. She now does similar work at the University of Virginia, where she is completing her doctoral work in Instructional Technology. She can be reached at

Judi Harris
621F Madison Avenue
Charlottesville, VA 22903
CIS: 75116,1207
BitNet: jbh7c@Virginia

Correction

In Michael Tempel's "Conversations with Logo" article in the January issue of *LX*,there lurks an error. On page 12, column 2, the second "Person" line should read **setxy 60 80.** We hope you didn't get a headachetrying to figure out what was going on! Our thanks to Dorothy Fitch for pointing this out so promptly.

Logo Connections

Is This Procedure Defined?

by Glen L. Bull and Gina L. Bull

Cochran's law states,

If you give a procedure to a friend, the friend's version of Logo will lack one of the commands needed to run the procedure.

Of course, it is generally possible to write a user-defined procedure which serves the same purpose as a built-in (or "primitive") procedure. For example, many early versions of Logo lack the EMPTYP command, but it is possible to write a user-defined procedure which can be used for the same task. However, the corollary to Cochran's law states,

If a procedure is provided to compensate for lack of a Logo primitive, it will be mistyped.

Is that :VARIABLE or "VARIABLE? Does it make a difference whether
(LIST :VARIABLE) or (LIST :VARIABLE)
is typed? Under some circumstances it can and does, and always under the circumstances designed to cause the maximum amount of confusion and trouble.

You say "toh may toh" and I say "toh mah toh". In Terrapin Logo it is EMPTY?, while in LCSI Logo it is EMPTYP, except, of course, in LCSI LogoWriter, in which it is EMPTY? again. Got it? When Paula Cochran was writing this column with us, she swore that a law should be passed preventing any more dialects of Logo. Pointing out that this would prevent any further progress in the development of Logo had no impact on her belief.

Which brings us to the subject of this month's column — transitions from one dialect of Logo to another. This semester we used LogoWriter in the introductory Classroom Computing class for the first time. LogoWriter allowed the students to begin doing interesting things with Logo sooner. We were pleased by the enthusiasm of the response. However, the response of teachers who already knew Logo was not so enthusiastic. As Cochran's law correctly predicted, invariably Logowriter lacked some of the commands needed to convert one of their favorite procedures.

For example, a popular version of INSTANT Logo used by elementary teachers in this area has this form:

```
TO INSTANT
MAKE "COMMAND READCHAR
IF DEFINEDP :COMMAND
   [RUN (LIST :COMMAND)]
INSTANT
END
```

Single-character procedures for this INSTANT are defined in the following way:

```
TO F
FORWARD 10
END

TO R
RIGHT 30
END
etc.
```

The clever aspect of this INSTANT is that it avoids the necessity of a series of tests of this kind:

```
IF :COMMAND = "F [FORWARD 10]
IF :COMMAND = "R [RIGHT 30]
etc.
```

It also allows other users to define new procedures such as

```
TO S
REPEAT 4 [FD 50 RT 90]
END
```

without the necessity of editing the main INSTANT procedure.

Local teachers wanted procedures such as INSTANT converted before they would consider shifting from Logo to LogoWriter, and that in turn, depended upon finding a way to create a DEFINEDP procedure in Logowriter. We began by writing a procedure to find other procedures on the FLIP slide of the LogoWriter page.

```
TO FIND :PROCEDURE
IF FRONT? [FLIP]
TOP
SEARCH
   (WORD "TO (CHAR 32) :PROCEDURE)
END
```

This procedure worked fine (or should we say that it worked "find"?) Only two more lines were needed at the end of the procedure to convert FIND into a DEFINEDP procedure:

```
FLIP
OUTPUT FOUND?
```

Although this worked after a fashion, it had a severe limitation in comparison with the DEFINEDP command in traditional versions of Logo. The LogoWriter search was highly visible as the cursor moved through the defined procedures. In between each key press of "F" as the turtle moved forward, the graphics screen with the turtle was temporarily replaced with the editing screen of procedures on the flip side of the page.

We were stumped. Finally we called the LCSI help line, and asked for a work-around for creating the equivalent of a DEFINEDP command in LogoWriter. We posed the question to the technician, who said, "There isn't any way."

At this point you probably think that the remainder of the column is about the heroic efforts we undertook before finally finding a way to emulate the DEFINEDP command in Logo-Writer. However, we still haven't figured out how to do it. Publicly stating that a certain function is not possible is the surest way to ensure that a reader who has been using Logo for one month will demonstrate a ludicrously simple way of accomplishing same. We will be sitting by our mailbox waiting for this solution, so please write.

In the meantime, you needn't be concerned about the teachers who wanted to convert the INSTANT program shown above to LogoWriter. Our solution to this problem looks like this:

```
TO INSTANT
MAKE "INPUT READCHAR
IF COMMAND?  [RUN (LIST :INPUT)]
INSTANT
END

TO COMMAND?
OUTPUT MEMBER? :INPUT [F B R L]
END
```

The COMMAND? procedure is used in place of the DEFINEDP command. It checks to see if the input (F for Forward, B for Back) is a defined command. An extra step of putting new commands in the COMMAND? procedure is required, so this procedure is not quite as elegant (maintenance free) as its Logo equivalent. However, it meets the need with a minimum of dislocation.

That was easy enough, you say? Ah, well, we have about 1143 other procedures which local teachers would like to see converted from Logo to LogoWriter. Just send us your name and address, and we will see that they are forwarded to you.

Advanced versions of INSTANT sometimes record the student's commands, and allow them to be used as the basis for new commands. For example, the following list might define a square in INSTANT Logo:

```
[F F F F R R R F F F F R R R F F F F
    R R R F F F F R R R]
```

The DEFINE command is typically used to convert this record of the student's keystrokes into a new procedure. Although we were not able to write a satisfactory LogoWriter equivalent of a procedure to tell whether a procedure is defined (DEFINEDP), we were able to write a procedure which defines another procedure (DEFINE). For example, try this procedure.

```
TO DEFINE.SHAPE
IF FRONT?  [FLIP]
BOTTOM
PRINT []
PRINT [TO SQUARE]
PRINT [REPEAT 4 [FORWARD 50 RIGHT 90]
PRINT [END]
FLIP
END
```

It is a short step from this procedure to a more generalized procedure. It takes two inputs which are turned into a procedure, much as the equivalent command in traditional Logo does.

```
TO DEFINE :NAME :COMMANDS
IF FRONT? [FLIP]
BOTTOM
PRINT []
PRINT SENTENCE "TO :NAME
PRINT :COMMANDS
PRINT [END]
FLIP
END
```

Often "Logo Connections" is about connections between Logo and other kinds of software. Finding connections between different dialects of Logo can be an equally challenging task. Cochran's law will continue to apply in full force as new and enhanced versions of Logo are developed. However, if the language were not improved, we would still be programming in FORTRAN. The very first versions of Logo did not have turtle graphics. This feature that was added later as an enhancement to the language. The time for concern will not be when new versions of Logo appear, but rather when they stop appearing. Well, we have to run now. We have just received a beta test copy of a new version of Logo, and the names of all our favorite commands have been changed.

You say "po tay to", I say "po tah to".
Let's call the whole thing off!

Glen and Gina Bull
Curry School of Education, Ruffner Hall
University of Virginia, Charlottesville, VA 22903
Glen's BitNet address is GLB2B@Virginia
Gina's BitNet address is RLBOP@Virginia.

MathWorlds

Edited by A. J. (Sandy) Dawson

A Turtle View on Geometrical Transformations (and Vice Versa)

Uri Leron and Rina Zazkis
Department of Science Education
Technion - Israel Institute of Technology
Haifa 32 000, Israel

Euclid vs. Turtle.

Everybody feels that standard Euclidean geometry and turtle geometry are intimately related. For example, people tend to intuitively identify FORWARD with plane translations and RIGHT with rotations. These intuitions are sensible but, as we shall see, are not entirely correct. It is our purpose in this note to give a more precise articulation of this relationship.

Felix and Kevin

For the discussion, we shall employ two characters, Felix and Kevin, to represent the two views. Felix (after the German mathematician Felix Klein, 1849-1929) is trained to think about geometry in transformational terms, but has never heard of the Logo turtle. Kevin, on the contrary, is a whiz kid about the turtle, but otherwise knows very little geometry and, in particular, knows nothing of geometrical transformations. As our story begins, Felix is watching with amazement the turtle manoeuvres on the computer screen, and is trying hard to explain what he sees in terms of his familiar geometrical transformations. Kevin, in turn, is watching with no less amazement a geometry class, in which a group of kids are practicing geometrical transformations by moving triangles about the table. He, too, is trying hard to make sense of what he sees in terms of his familiar turtle motions.

Felix in turtleland.

Let us start by following Felix in his efforts. The turtle, which he privately refers to as an electronic triangle, reminds him of an isosceles triangle in plane geometry, and the turtle motions remind him of the effect of plane transformations on this triangle. He first identifies these motions as being, roughly, translations and rotations of the electronic triangle in the screen plane. Later he observes that only very specific cases of these transformations actually occur: All translations are in the direction of the triangle's "head" and all rotations are about the mid-base point of the triangle. Furthermore, the translations are the result of the kid-at-the- keyboard typing FORWARD (the size being determined by the number that follows), whereas the rotations are caused by typing RIGHT (again, with a number specifying the amount to turn). Overall, he feels that his geometry is richer than the electronic one, since he can translate and rotate a triangle in the plane anyway he likes. Moreover, he recalls another type of transformation in his stock - reflection in a line - for which he can't even begin to imagine a parallel in the electronic geometry. But perhaps, he thinks, these can be achieved through compositions of the observed transformations? Here we must leave him to his musings and turn to our other hero, Kevin.

Kevin visits Euclid.

In the meantime, Kevin is trying to make sense of the kids' activities of moving triangles, to which he privately refers as paper turtles. One of the activities, as the kids readily explain to him, is studying the effect of plane translations on the triangle. Some of these translations, those shifting the triangle in the direction of its "head", he easily classifies as his familiar FORWARD (or BACK, if the motion is in the reverse direction on the same line). Others, shifting the triangle sideways, he thinks of as crab walks, but is having difficulty explaining in turtle terms. But then an idea occurs to him: A crab walk cannot be described with a single turtle command, but it can with a chain of commands. Since the turtle can only move in the direction it is facing, he reasons, I'll first turn it towards its final destination, then move it in that direction, then turn it back the same amount, so that it returns to its original heading! So, in turtle terms, these crab walks can be described as heading preserving transformations, and expressed by commands of the form: RIGHT *a* FORWARD *b* LEFT *a* (Fig. 1).

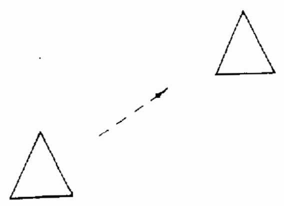


Figure 1(a) A Plane translation

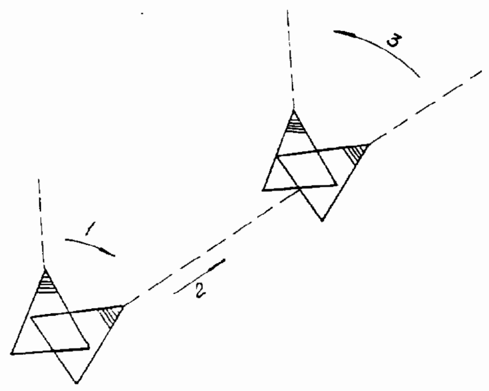


Figure 1(b) Translation as turtle operation

Kevin goes through a similar experience with another activity he observes, which is described by the kids as doing rotations. Again, some of the rotations, those rotating the triangle about itself, are similar to the turtle's RIGHT command, but the rest, in which the center of rotation is outside the triangle, are not so easily interpreted. Eventually, summoning his previous insights, he finds a way to describe even those in turtle terms: Move the turtle in a heading-preserving fashion to the center of rotation, then use RIGHT to rotate it the given amount and, finally, move it back with the reverse heading-preserving motion (Fig. 2).

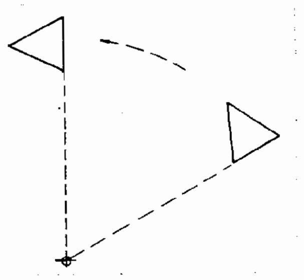


Figure 2(a) A plane rotation

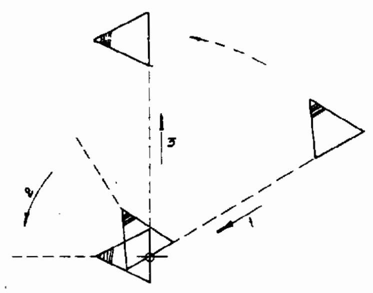


Figure 2(b) Rotation as turtle operation

The final activity, as the kids explain to him, is doing reflections about a line. For this he is at a loss to find an analog in the turtle world, but is determined to continue trying. Here we shall leave Kevin to his thoughts and take this opportunity to summarize what we have seen so far.

Euclid meets the turtle

In this section we describe the foregoing discussion more precisely, introducing appropriate mathematical terminology. First of all, Felix's and Kevin's efforts to interpret one system in terms of the other are described mathematically as building correspondences (or maps) between the two geometries, turtle and Euclidean. In fact, Felix is trying to define a map from the objects of turtle geometry to those of Euclidean geometry, and Kevin is trying to do the same in the opposite direction. The way they go about defining these maps is by identifying the turtle with an isosceles triangle, and turtle operations with plane transformations (represented by their effect on the triangle). Our aim in what follows, is to be more explicit about what precisely these objects and maps are. The objects of the two systems are turtle operations on the one hand, and plane transformations on the other. By turtle operations we mean the operations FORWARD *a* and RIGHT *b* , for all possible inputs *a* and *b* , as well as all sequences thereof. (Note that BACK and LEFT are also implicitly included as FORWARD and RIGHT with negative inputs.) By plane transformations we mean all the standard plane isometries, i.e., translations, rotations and line-reflections, as well as their compositions.

We can now describe Felix's map, which associates with every turtle operation a plane transformation, as follows. Given a turtle operation, start by considering its effect on the turtle in its HOME state. Next replace the turtle with an isosceles triangle and locate a plane transformation that has the same effect on the triangle. ("Same effect" means here that the turtle and the triangle coincide in both their initial and final states.) This transformation is the one corresponding to the given turtle operation. Reproducing Felix's line of reasoning, we find that to FORWARD *a* corresponds the northbound plane translation by a units, and to RIGHT *b* corresponds the clockwise plane rotation about the origin by *b* degrees. A general turtle operation, which is a chain of FORWARDs and RIGHTs maps to the corresponding composition of translations and rotations. In a similar fashion, Kevin's map attempts to associate with each plane transformation a turtle operation. Again, the given transformation and the corresponding turtle operation are to have the same effect on the objects they act upon (triangle and turtle, respectively). Kevin's findings show that, in general, translations and rotations correspond to compound turtle operations.

We shall return to discuss the nature of these compound operations in the next section. Since Felix's and Kevin's maps are inverses of each other, they establish a one-to-one correspondence between turtle operations and direct plane transformations (namely, those that are compositions of translations and rotations, but no reflections). This also solves Felix's problem: Turtle operations are just as rich as the direct plane transformations. Kevin's problem (concerning reflections) remains unanswered, but not for long.

Conjugates.

Going back to the statement that translations and rotations correspond to compound turtle operations, we now add that the resulting turtle operations are all suitable conjugates of FORWARDs and RIGHTs, where by a conjugate of B by A we mean an operation of the form "first A , then B , then the-inverse-of-A " (abbreviated ABA^{-1}). In particular, northbound translations correspond to FORWARD's, but general translations correspond to heading-preserving operations. These are operations of the form RIGHT *a* FORWARD *b* LEFT *a* , thus conjugates of a FORWARD by a RIGHT. Similarly, rotations about the origin correspond to RIGHTs, but general rotations correspond to conjugates of a RIGHT by a heading-preserving operation. We mention in passing that the ubiquity of conjugates can be understood from their intuitive meaning, which is "doing the same thing in a different place" (Leron, 1986). Thus, a translation in any direction (represented by a line through the origin) can be decomposed as follows: First rotate the given line about the origin until it is headed north (A), then do a northbound translation (B), then rotate the line back to its original direction (the inverse of A). Similarly, a rotation about any point can be decomposed as "first a translation to bring the given point to the origin, then a rotation about the origin, then the inverse translation to return to the point to its original position."

Reflecting on reflections

We now come back to Kevin's problem: What in the turtle world corresponds to line-reflections in the Euclidean plane? In our research, this question took a long time to answer, mainly because it required inventing a new turtle operation. However, two tools we've introduced here eventually make the solution quite simple and natural. One tool is representing reflections, as we did with all other transformations, by their effect on a particular isosceles triangle. The second tool is conjugacy, by which all reflections can be reduced to a single, special one: the reflection in the symmetry line of our triangle.

First we note that whatever the turtle operation that corresponds to this symmetry-line reflection, it cannot be obtained as a composition of the existing turtle operations. This can be seen by a physical argument, using Kevin's trick of viewing the triangle as a paper turtle. In its physical realization, the operation of reflection requires lifting the triangle (or the turtle) out of the plane, flipping it face down, then putting it back into the plane. Thus, though mathematically it is a plane transformation, its physical realization must go through the third dimension. Clearly, this "lifting out of the plane" cannot be obtained by any composition of planar translations and rotations. To describe this reflection in turtle terms, therefore, we need a new operation - operation that flips the turtle on its back. This operation is called FLIP, and its main effect is interchanging the turtle's right and left. Moreover, as we have seen in the case of translations, a reflection in any other line is obtained by an appropriate conjugation of this special reflection. Hence we conclude that, with the addition of FLIP, the correspondence between turtle operations and plane isometries is complete, making it possible to translate concepts and statements back and forth between the two geometries. It may be of some interest to note that the idea of the FLIP operation only occurred to us following a prolonged involvement with three dimensional turtles. Apparently our thinking had been bound too strongly to the flatland of the screen turtle.

A technical summary.

With the aim of making this a popular exposition, we have avoided using technical terms beyond those that are common in school geometry and elementary Logo programming.

We now add a brief summary for readers with a more advanced mathematical background. The appropriate conceptual framework for discussing the relationship between turtle geometry and Euclidean geometry is that of group theory, specifically the transformation groups of the two geometries. In order to view turtle operations as transformations, we shift to viewing them as acting on the turtle state, rather than on the turtle itself.

Formally, we define the turtle state to be the triple (x, y, h), where (x,y) are the coordinates of the turtle's position in a Cartesian system, and h is its heading, measured in degrees clockwise from the north. We denote by S the set of all turtle states and call it the turtle plane. Turtle operations are now defined as particular transformations of the turtle plane onto itself. Specifically, given real numbers a and b, RIGHT a and FORWARD b are defined for all (x ,y ,h) in S as follows. RIGHT a: $(x ,y ,h) \longrightarrow (x ,y ,h +a)$ FORWARD b: $(x ,y ,h) \longrightarrow (x + b \sinh , y + b \cosh)$ Finally, the turtle group is defined as a certain group of invertible transformations of the turtle plane, namely, the group generated by the set {FORWARD a , RIGHT b | a ,b real numbers}. The group operation is, as usual, the composition of maps.

In these terms, the main result of the forgoing discussion is that the turtle group and the group of direct isometries of the Euclidean plane are isomorphic. The FLIP operation can be defined similarly by adding a fourth component - the flip-state - to the turtle state. The resulting extended group turns out to be isomorphic to the full group of isometries of the Euclidean plane.

Our discussion of conjugate operations is also based on the notion of conjugate elements in a group. A full account of these issues is given in Zazkis (1989), and will be published elsewhere. (A preprint can be obtained by writing to the authors.)

References

Leron, U. (1986). State transparency and conjugacy. *Micromath* 1(3), 45-47.

Zazkis, Rina (1989). *Turtle geometry and transformational geometry - a group theoretic perspective.* Unpublished doctoral thesis (Hebrew), Technion - Israel Institute of Technology.

A. J. (Sandy) Dawson Compuserve: 76475,1315.
Bitnet: userDaws@SFU.BITNET U

Assessing Logo Learning In Classrooms

VI. Debugging Strategies
by Dan Watt

This is the seventh of nine columns based on a research project which Molly Watt and I have been carrying out with support from the National Science Foundation, "Exploratory Research on Critical Aspects of Logo Learning." In this project, we collaborated with a group of experienced Logo teachers to identify critical aspects of Logo learning and group them under eight headings which were listed in September's column.

This month I will write about each of the subheadings included in the sixth cluster of critical aspects, Debugging Strategies. For a fuller sense of what we mean by critical aspects of Logo learning, and our rationale for this approach to assessing Logo learning, please read the September '88 column in this series.

The Trouble with Kathy's House

In January's column on Mathematical Thinking, we looked at Kathy's House project as an example of visual problem solving. This month we'll focus on one detail, the bug in the turtle's orientation which led to what Kathy called a "lopsided" window, and we'll see how her teacher helped her debug it. As you can see from Figure 1, the turtle looked — to Kathy and to us — as if it was heading straight up the screen. However, the lines it drew when it moved forward were at a slight angle.

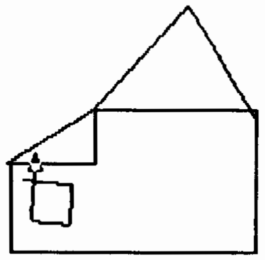


Figure 1

Kathy's approach to drawing Logo pictures relied almost entirely on being able to determine the turtle's position and orientation visually. When its visual image was slightly out of alignment with its actual orientation, she was stumped. The problem occurred because the net result of a series of rotations, very carefully constructed by Kathy through a process of successive approximations (see Figure 2), left the turtle at a heading of 95 degrees, when it looked as though it was at a heading of 90 degrees, just before starting to draw the window.

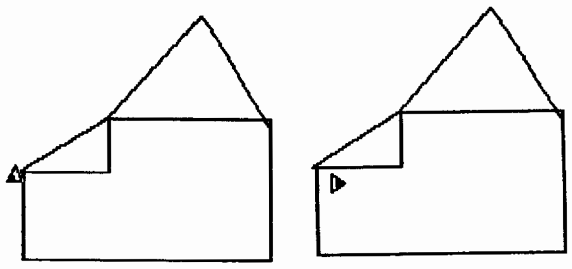


Figure 2

Kathy's teacher showed her how to use the TRACE and PRINT HEADING commands, and suggested that she make the turtle's heading zero, before starting to draw the window. Kathy then found that the turtle's heading was 239 degrees just after it finished drawing the roof. And by trial and error she found that a rotation of RIGHT 121 made the turtle's heading zero. She erased all the commands in her procedure that came after drawing the roof, added RIGHT 121, and completed the rest of her house successfully, using her tried-and-true method of visual problem-solving.

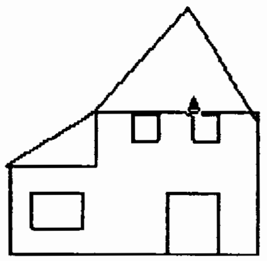


Figure 3

The Central Importance of Debugging

If the impact of computers in classrooms were limited to one idea — and I were allowed to choose what that idea could be — I would choose the idea of a bug, a mistake or misconception in a computer program, and the associated notion of debugging. The Piagetian view of learning, on which Logo's educational philosophy is based, suggests that learning is a process of building internal models of aspects of reality, and refining those models by testing them, gradually revising them when they no longer seem to fit our real-world experiences. This means that the most important types of learning occur through a process of trial and error, and especially, a process of compensating for errors by changing the models guiding our actions. This implies that the errors we make, and our own thinking that leads to those errors, are as important to study as the actual tasks we are trying to accomplish.

Seymour Papert made debugging one of the central themes of *Mindstorms*, and contrasted it with the right-or-wrong ethic that we encounter so often in classrooms (especially in math class!)

School teaches that errors are bad; the last thing one wants to do is pore over them, dwell on them, or think about them. ... The debugging philosophy suggests an opposite attitude. Errors benefit us, because they lead us to study what has happened, to understand what went wrong, and through understanding, to fix it. Experience with computer programming leads children more effectively than any other activity to 'believe in' debugging (Papert, 1980).

Yet even in Logo classrooms we see learners (adults as well as children) attempt to hide and ignore their bugs, clear away the evidence, and start all over without attempting to understand what has just happened. Papert attributed this phenomenon to the cumulative effects of schooling.

The ethic of school has rubbed off too well. What we see as a good program with a small bug, the child sees as "wrong,' "bad," "a mistake." ... The child is glad to take advantage of the computer's ability to erase it all without any trace for anyone to see. ... Contact with the Logo environment gradually undermines long-standing resistance to debugging. ... (ibid)"

I agree that difficulties with debugging are partly a matter of attitude. But I also think that there are specific skills and strategies that make debugging more effective, and encourage people to engage in debugging more often, which in turn helps them shift their attitudes towards errors. Some of these are quite simple and straightforward. Others involve making use of special capabilities of particular versions of Logo. Still others involve taking the time to take note of and think about bugs as interesting, in and of themselves.

Jim MacIsaac was the most experienced Logo programmer in a class of 27 classroom teachers taught by Molly Watt in Vancouver a few years ago. Most of them were learning Logo for the first time and Jim spent a great deal of time helping his colleagues through their programming difficulties — so much so, that several of them were feeling guilty about taking him away from his own work. Sensitive to these feelings, he surprised the class one day by spontaneously thanking them for providing him with such a wonderful problem-solving experience. "The part of Logo I love best is debugging," he explained. "but since I've been programming in Logo for a while now, I have great difficulty coming up with interesting bugs on my own. In this class you've provided me with so many intriguing bugs to work on, really challenging problems to solve, many more than I could possibly have invented in months of trying. It's been lots of fun, and a terrific learning experience for me."

It should be a goal of all Logo teachers to encourage the Jim MacIsaac's in our Logo classes, and to use them as models to help all our students learn to enjoy the process of debugging as one of the most essential, interesting and pleasurable parts of any learning experience.

A general debugging strategy

David Klahr and Sharon Carver, cognitive scientists working at Carnegie Mellon University have developed a general approach to debugging any Logo program, and have reported positive learning gains (that is better debugging, and more success in completing projects) as a result of teaching a specific debugging curriculum to children, as one part of their Logo learning experience (Carver, 1987, Carver and Klahr, 1986). At the risk of oversimplifying, I'd describe their approach as consisting of three distinct steps: identify the bug; locate the bug (that is find out exactly where the bug occurs in a particular set of procedures and subprocedures); and fix the bug. I'd also like to suggest that there are two other aspects of debugging worthy of attention by Logo teachers and students: incorporate the bug, that is, take advantage of a new possibility that came about because your procedure did something you had not intended; and develop new programming strategies that make the possibility of repeating the same bug much less likely in the future.

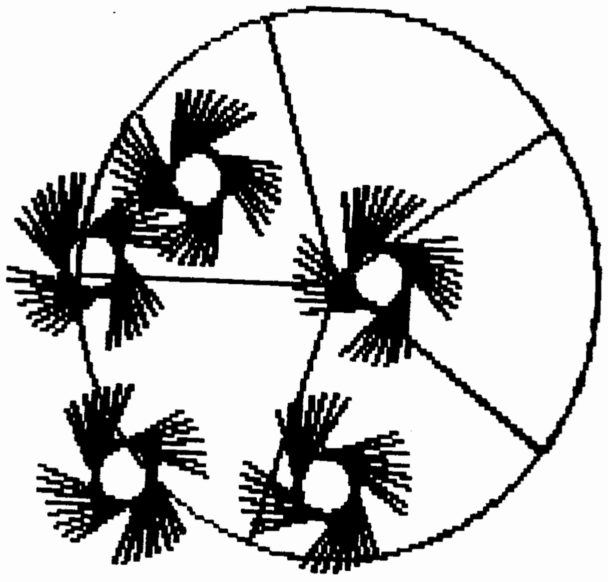
<u>1. Identify the Bug</u>

<u>1a. Make use of Logo's error messages</u>

Although this seems obvious, it is often neglected. How often have you seen Logo students press the return key several times, to clear the screen and remove the evidence of error in the form of a Logo error message? Logo's error messages are designed to help a user identify exactly why a command or procedure could not be carried out, and where the problem occurred. Students should learn to consider an error message as a source of useful information, as a first step in learning to identify, locate and fix those bugs which occur as a result of typing commands which Logo cannot carry out.

<u>1b. Describe the bug — to yourself and someone else</u>

This is exactly what Kathy did when she told her teacher that she was unhappy with the window in her house because it was "lopsided." It was what Douglas did when he wrote "I wish that the design would go in each part of the circle," on a print of his PENTACIRCLE drawing (see Figure 4). In our experience with hundreds of Logo students of all ages, describing a bug to a teacher or classmate is often all that is needed for a programmer to say, "Oh, now I see what happened ... " and be well on the way to resolving the problem (Watt and Watt, 1986).



```
TO PENTACIRCLE
 HT
 PU LT 90 FD 50 RT 90 PD
 REPEAT 360 [FD 1.5 RT 1]
 REPEAT 5 [RT 90 FD 85.1734 BK 85 LT 90!
 REPEAT 72 [FD 1.5 RT 1]]
 PU RT 90 REPEAT 5 [FD 30 LT 90 FD 40 P!
D REPEAT 30 [REPEAT 2 [FD 1.5 RT 1] FD !
15 BK 35 LT 90] PU LT 180 FD 30 RT 180 !
LT 72]
END
```

Figure 4

3. Categorize and name the bug for future reference

This can be extremely helpful in debugging future instances of the same or similar bugs. Both Kathy's and Douglas' bugs could be called "turtle state bugs," and even that general descriptor helps us see where to start debugging. More specifically, Kathy's bug might be called: "The turtle looks like its heading is 0 or 90 degrees, but its actual heading is slightly off." If Kathy had this description written down on a list of common turtle state bugs, she might have known what to do about it on her own.

Naming Douglas' bug is a little more complicated. I think it's actually a combination of two common turtle state bugs: "failure to distinguish between the part of a procedure that moves the turtle, and the part that draws the design," and "failure to have the turtle return to its exact starting place, in drawing a distinct part of a design, before moving to draw another part of the design." In addition, I'd say that Douglas' procedure exhibits two other common bugs in Logo problem-solving strategies: "hiding the turtle to make it draw faster, before having completed a project," and "failure to subdivide a complex project into clearly named subprocedures." I believe that if Douglas had been aware of these four statements as identifying common types of bugs, he would have been much less stumped by the bug in his procedure than he was."

2. Locate the bug

2a. Use Logo debugging aids

Let's not forget the obvious. Logo provides lots of built-in debugging help:error messages, TRACE, SHOWTURTLE, HEADING, PRINTOUT (PO) and so on. If the bug involves a command that Logo could not carry out, Logo's error messages tell what that command is, and in some versions of Logo, the procedure, and even the line in which it occurred. Following error messages, a TRACE primitive (in some versions of Logo) is Logo's most powerful debugging aid. TRACE lets a procedure do its work, one line at a time, so that a user can isolate exactly where something went wrong. For a turtle state bug, using TRACE in conjunction with PRINT HEADING, may give you all the information you need to figure out what went wrong, as well as where. Since only a few command lines can be shown on the screen at once, a print on paper is another crucial debugging aid. (Believe it or not, we've run into many Logo students and even Logo teachers, who have never learned to make hard copy of their pictures or their procedures.)

2b. Divide a long procedure into parts

Sequential, undifferentiated Logo procedures are notoriously difficult to debug. But even a long procedure can be divided into parts, if you learn to look for certain clues. The most common are the use of PENUP and PENDOWN in many Turtle Geometry designs to separate the parts of the drawing that moves the turtle from the part that draws the design. Mark the location of PENUPs and PENDOWNs on your printed copy, and work from there. For designs in different pen colors, commands that set the pen color can be used the same way to differentiate parts of the design and isolate the bug.

A procedure is easier to debug if it's divided into parts, each of which carries out only one task, such as moving the turtle or drawing a particular piece of a design. If a procedure is too complex to understand easily, such as Douglas' PENTA-CIRCLE, rewriting the procedure by dividing it into subprocedures and testing each subprocedure separately, might be an important step in locating the bug.

3. Fix the bug

Sometimes all you need to do is locate the bug, and you'll see exactly what needs to be done to fix it. In other cases, such as Kathy's, for example, there is still more work to be done.

3a. Play turtle

If your bug is a turtle state bug, playing turtle often helps unsnarl what went wrong and helps you see what to do about it.

3b. Use mathematical knowledge

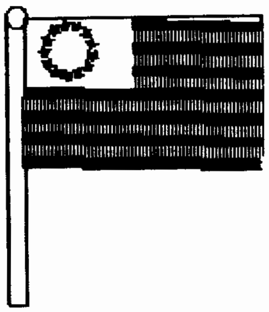
Mathematical knowledge, such as the rule of 360 (total turtle trip), can be invaluable in correcting bugs. If you add up how far the turtle has already turned, you may be able to determine the correct amount to turn next by using the idea that a complete turtle rotation is 360 degrees, or that a symmetrical figure requires a left turn in a certain place, where a right turn was used in a corresponding place somewhere else. This approach might be very helpful to Douglas. He seems to use the total turtle trip idea whenever he can, but he has not applied it correctly to the very last line of his PENTACIRCLE procedure.

3c. Divide a long procedure into subprocedures and test each part separately

Some combination of this approach and the preceding ones will probably be essential to fix Douglas' bug. Unless he identifies, by name, the part of his procedure that draws each of the little whirligig designs, and the part that moves it from place to place, he could use trial and error forever, and have little hope of solving his problem.

Incorporate the bug

Sometimes a bug improves a project because it introduces an element that you might never have planned. Martha and Vinh's FLAG procedure is a case in point. After they completed this procedure entirely on their own, their teacher asked them two questions: "How did you figure out the spacing of the thirteen stars?" and "How did you get the stripes to ripple?" She didn't realize that there was a direct connection between the two processes.



```
TO FLA
 REPEAT 13 [M2 STAR]
END

TO M2
 PU LT 27.6 BK 7 PD
END

TO STAR
 REPEAT 5 [FD 6 [illegible]
END
```

Figure 5

Martha and Vinh explained that they had divided 360 by 13 and used the result, 27.6, to rotate the turtle between stars. (This is clearly shown in three of their subprocedures, FLA, M2 and STAR.) They went on to explain that the ripple of the stripes was an accident! They had expected the stripes to come out straight, but when they came out at a slight angle, they left them that way because it produced a much better looking flag than the one they had planned.

Martha and Vinh never debugged their flag, and neither did their teacher. But even when you have incorporated a serendipitous bug into your final design, it's useful to play detective and see if you can determine how it came about. Experienced turtle geometers will no doubt recognize that the two students were victims of the same bug as Kathy: "The turtle looks like its heading is 0 or 90 degrees, but its actual heading is slightly off." But where did the bug come from? We need to call on the total turtle trip idea, and use PRINT HEADING to complete the detective work. Try this. Start with the turtle at home, and type REPEAT 13 [RIGHT 27.6] PRINT HEADING. You'll find that the turtle's heading is closer to 359 degrees than 360. Q.E.D! Thereby hangs the ripple.

Incorporating a bug can be a cop-out, an excuse for being lazy. And frankly, unless a student has a good reason for fixing a program — such as publishing it for other students to use — there may be little justification for requiring him to fix every bug, especially if he's satisfied with the result. So creating a context in which debugging is appreciated, in which bugs themselves are valued and shared as much as final results, is a critical element of teaching effective debugging strategies.

Is there life after debugging?

What happens after a bug is eliminated or incorporated? Remember Piaget's idea about learning? We only learn by confronting discrepancies between our existing model of how something works, and our experience of what really happens when that model is applied. After debugging and resolving an immediate difficulty, it may be useful - perhaps essential - in maximizing the benefits of learning Logo, that we stop and reflect, and see if we can change the ways we do things, so that in the future we'll be less likely to be plagued with the same bug all over again. Sometimes the reflection is initiated by a student, sometimes by a teacher. For instance, Kathy's teacher insisted that Kathy use subprocedures with separately named MOVE procedures in her next Logo project! Douglas' teacher suggested a similar strategy to him. Martha and Vinh's teacher had already established the use of subprocedures in her students' work. They were less likely to get stumped by bugs they wanted to fix than were Douglas or Kathy.

Although the topic of this column is debugging, I'd like to close by suggesting a few strategies for minimizing bugs—at least of the turtle state variety:

- Take careful notes, draw pictures of what the turtle is doing, write notes on the pictures, use dotted lines to show where the pen is up, and so on.
- Use subprocedures. Name each part of a turtle drawing, and name the moves and picture elements separately.
- Make use of mathematical ideas such as the total turtle trip and right/left symmetry, wherever possible.
- And create state-transparent designs, in which the turtle returns to its starting place after drawing each part, before moving to draw something else. If Douglas had done this, he'd have been home free!

I'm certainly not suggesting that Logo teachers strive for bug-free programming by their students. First of all, it's impossible. Second, it would spoil lots of the fun (remember Jim MacIsaac). And finally, if we managed to avoid all bugs, we'd have stopped learning because we'd only be working with what we already knew!

References

Carver S. (1987). "Learning and Transfer of Debugging Skills: Applying Task Analysis to Curriculum Design and Assessment", to appear in Mayer, R. Ed. *Teaching and Learning Computer Programming: Multiple Research Perspectives*, in press.

Carver S. and Klahr, D. (1986). "Assessing Children's Logo Debugging Skills with a Formal Model," *Journal of Educational Computing Research* (in press).

Papert, S. (1980). *Mindstorms*, Basic Books, New York NY.

Watt, M. and Watt, D. (1986). *Teaching With Logo: Building Blocks for Learning*, Addison Wesley, Menlo Park CA.

The work described here was conducted at Education Development Center (EDC), 55 Chapel Street, Newton Massachusetts, and supported in part by the National Science Foundation under grant # MDR 8651600, Exploratory Research on Critical Aspects of Logo Learning. The ideas and opinions expressed are those of the author and do not necessarily reflect the views of EDC or the National Science Foundation.

Dan Watt
Educational Alternatives
Gregg Lake Road
Antrim, New Hampshire 03440

Long Distance Logo

Educators - You don't have to go to classes to earn graduate credit - let the classes come to you!

Introduction to Logo Using LogoWriter, a graduate level independent study course, allows you to learn at your own pace while corresponding with your instructor by mail. The cost for a four quarter-hour class is $320.00, or less.

Work Individually or With a Group of Colleagues - Take ***Introduction to Logo Using Logo-Writer*** at home, or study with a group of colleagues at school. The course uses a combination of video tapes (ON LOGO) featuring MIT's Seymour Papert, printed materials, textbooks, and diskettes. You view the tapes, read and report on course materials, do projects, design LogoWriter lessons for your students and correspond with your instructor by mail.

Not Just Another Class - Dr. Sharon Burrowes Yoder, editor of the *Logo Exchange* journal, designed ***Introduction to Logo*** to provide staff development and leadership training. The four quarter-hour course meets the standards of the College of Education at U. of Oregon, and carries graduate credit from the Oregon State System of Higher Education.

ON LOGO Video Tapes - School Districts may acquire a license for the use of the ON LOGO package of 8 half-hour videotapes and 240 pages of supporting print for $599.00. For a one-time fee of $1295.00, the package may be obtained with both tape and print duplicating rights, enabling the district to build libraries at multiple sites. If your district owns the tapes, course tuition is lower. Tuition rates are also lower for group enrollment.

Tuition Information, Detailed Course Outlines, and Order Blanks can be obtained from: LONG DISTANCE LEARNING, ICCE, U of O, 1787 Agate St., Eugene, OR 97403-9905. Phone 503/686-4414.

Logo: Search and Research

Planning a Strategy
by Douglas H. Clements

"Read the list [of directions] again, but change all the LEFTs to RIGHTs for this side. "

[Later:] "Not RIGHT 90! We already did RIGHT 20. What's 90 minus 70...20, right? We need, not RIGHT 90, but 70!"

[Later:] "We got to add these three numbers. Then the bottom would be the same length overall as the top."

These elementary students are deciding on a strategy for solving a problem. They're planning. Computer programming appears to demand considerable planning. It requires the careful selection and ordering of instructions to solve a problem. Also, Logo's modular nature allows students to combine procedures in various ways. So it's not surprising that many researchers have studied Logo students' planning and use of problem-solving strategies.

In an early study, Pea and Kurland (1984) hypothesized that the feedback in programming would help students see that planning led to more effective performance. After 20-24 weeks of programming in Logo, middle-school students did not display greater planning skills on a non-computer task than those in a matched group

The problem could be that although some adults see that planning in Logo is helpful, students don't. Do they plan as they work with Logo? Several studies have observed student programmers to find out.

Planning Logo Work
Primary grade students just learning to program in Logo tend to use opportunistic "planning" (planning-in-action only one step at a time). They will plan farther ahead, but only when working with a guiding teacher (Dytman, Peverly, & Wang, 1985). There is a hint that the situation improves. High-knowledge students were more likely to plan well. These students had higher knowledge of computer terminology and procedures for simple geometric figures. They were more likely to use plans that took a general approach to solving a problem. They were more likely to plan in modular units and to provide a reasonable top-down plan for structured tasks. So as students learn more Logo, they may plan more. Regardless of their level of knowledge, Logo students talk about strategies more than do students working with CAI materials (Clements & Nastasi, 1988—the quotations at the beginning were taken from this study).

Along a similar line, Kull (1986) observed quite a bit of pre-planning in first grade Logo programmers. However, it developed slowly and often consisted of planning only a few moves ahead. For example, some students traced lines on the screen with their fingers and recited the commands they would use. Others used PENUP and tried out moves without leaving a path. Students then would often declare an overall strategy, such as "I have to make it go up to here, turn this way, the go over to here and down."

Noss (1984) believes that intermediate grade students in his study did not automatically plan their Logo work because such planning activities were not an explicit part of the teaching. Students did, however, come to see the value of planning and a marked increase in planning activity took place over time. Hillel's (1984) students accepted pre-planning as a part of their ongoing Logo work. However, their teachers had to be very clear that this was expected! In summary, students learn to use planning in Logo. However, especially without clear expectations from their teacher, they develop these skills slowly.

Transfer to Other Planning Tasks
Of course, all these observations were limited to Logo tasks. Do students apply these planning skills on non-Logo tasks? They didn't in the Pea and Kurland (1984) study discussed previously. My own research similarly indicates that the smallest benefit is in the area of planning or strategy selection (Clements, 1986, in press). So does another year-long study by the Bank Street researchers (Kurland, Pea, Clement, & Mawby, 1986). High school students spent 36 weeks, five days per week studying 6 languages. There was no effect on planning tasks. The researchers offered the reason that, despite lengthy and expert teaching, students just had not attained a high level of expertise in programming.

What's Happening Here?
Why do observers of Logo programmers gather evidence of planning—or at least the emergence of planning—but those who give students transfer tasks turn up empty-handed? Planning may be too complex a skill, and the first beginnings we see in Logo may be too fragile for transfer. Or planning and problem-solving strategies used in Logo may be too different from those used to solve other tasks. In either case, there is something that teachers can do. Kurland and his colleagues conclude that teachers need to plan themselves—plan for transfer, that is. They have to provide students with:

•many examples of how to use planning skills,
•links to real-world problem-solving situations,
•subject-matter instruction, and
•universal, or abstract, descriptions of planning skills.

Does this work? It can. There are studies in which Logo students learned to use problem-solving heuristics and to plan (Bamberger, 1985; Billings, 1986; Horton & Ryba, 1986; Lehrer, in press). Each one devoted specific attention to developing these abilities. Some taught the skills directly. Others emphasized question–asking. But they all "planned for planning."

Let's look more closely at one of these studies. Bamberger worked with all the fourth–grade students in one school. She stressed the need to plan a procedure before beginning it and to use such strategies as breaking a large idea into more manageable parts. Some group instruction occurred, but individual or paired instruction was emphasized. The teacher used a guided discovery approach, with lots of questioning. After the Logo experience, Bamberger conducted interviews. Logo students used the strategies of planning and drawing more frequently to solve non-Logo mathematical problems.

What Might We as Teachers Do?

Implications are clear. Logo alone may do little to increase planning skills. If this is your goal, planning for planning is especially important. The following teaching strategies may help.

•Have students discuss their Logo planning strategies, encouraging students to become explicitly aware of their planning processes.
•Discuss other situations in and out of the classroom in which such planning skills would be useful. Elicit general descriptions of what good planning is all about.
•Promote the use of procedurality as a tool (rather than "a way to save pictures") from the beginning of instruction.
•Help students see patterns in the application of different strategies, so that they increase their repertoire of programming templates and approaches to solving problems.
•Discuss alternative approaches to solving Logo and non–Logo problems and consider why and when particular approaches might be useful.
•Use such questions as:
1. What could you do to get started?
2. Can you solve part of the problem?
3. What do you want the turtle to do right now? Next?
4. How will you make the turtle do that?
5. Have you used procedures (plans, strategies) before that could help you solve this problem?
•Remember to have students ask themselves:
1. What plan or strategy was best for me?
2. Why was it the best?
3. What will I do the next time?

In summary, to develop planning skills it may be necessary to structure students' work with Logo so that they predict and plan before programming. It should be remembered, however, that certain students may resist planning their Logo programs. They may enjoying instead the opportunity to work with mathematics in an intuitive style more natural to them (Papert, 1987).

Much work has been done on planning. But space is limited (the editor has threatened to use her procedure DELETE.EVERY.OTHER.WORD :COLUMN). So next month we'll discuss three more studies on planning in Logo, each of which includes specific teaching suggestions for developing this valuable skill.

References

Bamberger, H. J. (1985). The effect of Logo (turtle graphics) on the problem solving strategies used by fourth grade children. *Dissertation Abstracts International, 46*, 918A. (University Microfilms No. DA8512171)

Billings, L. J., Jr. (1986). Development of mathematical task persistence and problem–solving ability in fifth and sixth grade students through the use of Logo and heuristic methodologies. *Dissertation Abstracts International, 47*, 2433A.

Clements, D. H. (1986). Effects of Logo and CAI environments on cognition and creativity. *Journal of Educational Psychology*, 78, 309-318.

Clements, D. H. (in press). Metacomponential development in a Logo programming environment. *Journal of Educational Psychology*.

Clements, D. H., & Nastasi, B. K. (1988). Social and cognitive interactions in educational computer environments. *American Educational Research Journal*, 25, 87-106.

Dytman, J. A., Peverly, S., & Wang, M. C. (1985, April). *An investigation of the role of the learner in Logo learning environments.* Paper presented at the annual meeting of the American Educational Research Association, Chicago, IL.

Hillel, J. (1984). *Mathematical and programming concepts acquired by children, aged 8-9, in a restricted Logo environment.* Unpublished manuscript, Concordia University, Montreal, Quebec.

Horton, J., & Ryba, K. (1986). Assessing learning with Logo: A pilot study. *The Computing Teacher*, 14(1), 24-28.

Kull, J. A. (1986). Learning and Logo. In P. F. Campbell and G. G. Fein (Eds.), *Young children and microcomputers* (pp. 103-130). Englewood Cliffs, NJ: Prentice–Hall.

Search and Research -- continued

Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, 2, 429-458.

Lehrer, R. (in press). Computer–assisted strategic instruction. In M. Pressley, C. B. McCormick, & G. E. Miller (Eds.), *Cognitive strategy research.* New York: Springer–Verlag.

Noss, R. (1984). *Children learning Logo programming. Interim report No. 2 of the Chiltern Logo Project.* Hatfield, U.K.: Advisory Unit for Computer Based Education.

Papert, S. (1987). Computer criticism vs. technocentric thinking. *Educational Researcher*, 16, 22-30.

Pea, R. D., & Kurland, D. M. (1984). *Logo programming and the development of planning skills.* Technical Report No. 16. New York: Bank Street College of Education, Center for Children and Technology.

Douglas H. Clements is an associate professor at State University of New York at Buffalo, Department of Learning and Instruction, 593 Baldy Hall, Buffalo, New York, 14260. He has conducted several research studies on the effects of Logo environments on children's creativity, metacognitive ability, problem-solving skills, and social interaction. Through a National Science Foundation grant, he is currently developing an elementary geometry curriculum based on Logo. His most recent book, *Computers in Elementary Mathematics Education*, was published by Prentice-Hall in September, 1988. An earlier book, *Computers in Early and Primary Education*, was published by Prentice-Hall in Spring, 1985. (E-Mail: CIS: 76136,2027, BitNet: INSDH@UBVMSA)

A Tip for Using Random

by Stan and Linda Burech

Below is a procedure that we use whenever the primitive RANDOM is used. In many versions of Logo, each time you start your computer system, you will get the same sequence of random numbers. By using the procedures we developed, you can cause Logo to start with a different seed number each time. Insert this procedure at the beginning of the first procedure in your program that uses RANDOM.

The program given below generates a different seed or starting point for a sequence of random numbers. The superprocedure, RANDOMSEED waits for a key to be pressed. Each person using the program is likely to take a different amount of time to press a key, thus making the program rely on varying human reaction time. The procedure WAITKEY waits for a key to be pressed using KEYP. If a key is pressed, the program stops. If a key is not pressed, the procedure SEED is called. SEED simply accepts a random number as input and does nothing. Each time SEED is called, it moves further down the sequence of random numbers. Thus, when a key is pressed, Logo will start at that point in the random number sequence the next time RANDOM is called.

```
TO RANDOMSEED
PRINT [PRESS ANY KEY TO BEGIN]
WAITKEY
CLEARTEXT
END

TO WAITKEY
LOCAL "KEY
IF KEYP [MAKE "KEY RC STOP]
SEED RANDOM RANDOM 9999
WAITKEY
END

TO SEED :NUMBER
END
```

Stan and Linda Burech
112 Franklin Street
St. Clairsville, Ohio 43950

Global News

Edited by Dennis Harper
University of the Virgin Islands
St. Thomas, USVI 00802

Logo Exchange Continental Editors

Africa	Asia	Australia	Europe	Latin America
Fatimata Seye Sylla	Jun-ichi Yamanishi	Jeff Richardson	Harry Pinxteren	Jose Valente
Lab Informatique et Ed	Faculty of Education	School of Education	Logo Centrum Nederland	NIED
BP 5036 Dakar	Toyama University	GIAE	P.O. Box 1408	UNICAMP
Senegal, West Africa	3190 Gofuku	Switchback Road	BK Nijmegen 6501	13082 Campinas
	Toyama 930 Japan	Churchill 3842	Netherlands	Sao Paulo, Brazil
		Australia		

This month's Global Logo Comments takes us to three continents. We briefly look at activities in Japan and Bulgaria and finish with a portion of a wonderful series of Logo activities from Argentina.

Our Asian correspondent, Jun-ichi Yamanishi reports that microcomputers are finally appearing in the primary and secondary schools of Japan. Last year the Japanese Ministry of Education presented a plan to introduce a new curriculum whose aim was to help students cope with advances in science and technology as well as being able to process information. The curriculum is expected to be fully implemented by 1993.

The important Center for Educational Computing reported that Logo is the best language to use when introducing computers into the schools. Recently, a great many school teachers have become interested in Logo. This certainly was not the case as recently as two years ago. Japanese Logos are available on a variety of microcomputers in Japan, including a Japanese version of LogoWriter.

Workshops for teachers have begun and some of the early trainees have started to introduce Logo into their schools. At Toyama University, semi-annual workshops have been taking place over the past five years. These courses have been emphasizing LEGO® TC logo during the 1988/89 school year. Watch Global News for updates on Logo teacher training in Japan.

Another country embracing Logo is Bulgaria. The experience of the Research Group on Education (at the Bulgarian Academy of Sciences and Ministry of Education) in teaching informatics with Logo as a programming language has confirmed Bulgaria's conviction that Logo should be the basic language when using computers in the schools.

Recent research in Bulgaria has been devoted to the use of computers in the teaching of mathematics. The good possibilities provided by Logo in this direction are well known. Although the turtle talks in a mathematical language, it has been found that pupils are not always aware of this fact. Logo work in the mathematics classes of Bulgaria has revealed that Logo is useful for elementary levels of mathematics but not quite adequate in more complicated work. In this respect, researchers in Bulgaria have compared Logo to assembly language for work in mathematics, Logo has great potential but all the same ... assembler?

To contend with this problem and make Logo applicable to a wider range of mathematical concepts, Sofia University established the Mathematical Laboratory in Logo. Mathematics educators interested in this work can contact Bojidar Sendov, Faculty of Mathematics and Informatics, Sofia University, 5 A Ivanov bul, 1126 Sofia, Bulgaria for more information.

Dedicated Logoites and long-time readers of this column are familiar with the quantity and quality of Logo activity in Argentina. Led by Horacio Reggini, a fast growing group of educators are developing Logo environments and activities for their students. An series of activities was produced by Rosa Kaufman who is an author of children's texts, a software author, and a lecturer and computer consultant to many of Argentina's primary schools.

Her activities use the graphics commands of the Logo tortoise. The activities are accompanied by Software Logo, a series of Logo programs developed to enhance the elementary mathematics syllabus. The activities emphasize spatial orientation, length and angle measurement, sequences and the creation of algorithms. There are at present more than 50 activities aimed at providing even the youngest students the benefits of tortoise geometry.

Global News -- continued

Following is part of an activity entitled Pirate Logan's Maps. As in all of Rosa Kaufman's activities, Pirate Logan's Maps includes a guide for teachers, objectives (computing, mathematics, and language arts), prerequisites, suggested grade level, class organization hints, and other auxiliary material. What is produced below is part of the students' worksheet.

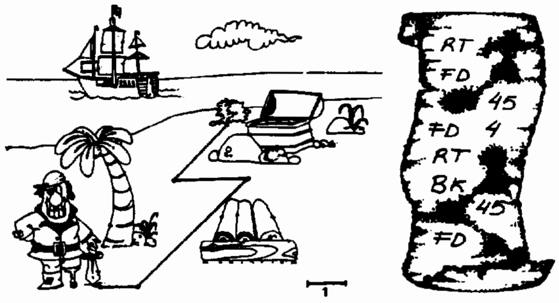
PIRATE LOGAN'S MAPS

Pirate Logan Jr. is very happy because he found some treasure maps in Uncle Logan's house. They are maps from the times when pirates deciphered them and found hidden treasures of jewelry and precious stones. They were brave and audacious, but not very good pirates. Not like the pirates in the Logan family, who never find anything unless they are using the turtle.

The last time Logan Jr.'s uncle looked for a treasure, he ended up finding it in his own ruined brig. He says he is not very good at this. This is why Pirate Logan Jr. decided to look for the coffers in these maps by himself. This is what he decided, but he is a bit inexperienced. Perhaps we can help him.

1. Load the program THE ISLAND in the computer. You have to lead the pirate as the messages appear. You could start using straight angles at first. If you play with other classmates or groups, the team that finds the treasure first, or makes the least mistakes in the commands given, is the winner. GOOD LUCK!!!

2. The itinerary you can see below was made by Logan Jr. based on the map. But he had to discover some data because it seems it was dirty or torn (or both) in some places. What data did he discover? Can you invent a map that will reach the coffer through a different way? Write it and then ask your classmates to represent the itinerary in a graphic way. Let us see if they are good pirates.



Work and Play in the Pacific Northwest this Summer:

Increase Your Leadership and Logo Skills and Enjoy a Week in Scenic Oregon!

Attend these ICCE workshops:

Leadership Development

Topics:
- •Long-range planning for computers in schools
- •Effective inservice and staff development
- •Computer/brain/problem solving
- •The computer coordinator
- •Frontiers and the future
- •Grant writing and other writing for publication

Staff: Dave Moursund and Sharon Burrowes Yoder
Dates: July 9 - July15, 1989
Location: Eugene, Oregon
Cost: $300 before May 1st; $350 after May 1st; covers 2 quarter-hours graduate credit, materials, & instruction.

Receive a 10% discount if you attend both!

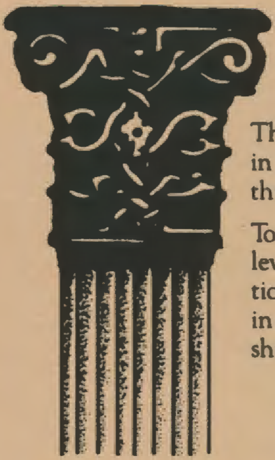
Logo for Leaders: Beyond Turtle Graphics

Topics:
- •Planning the implementation of Logo on a building or district level
- •Integrating Logo into the curriculum
- •Logo and problem solving
- •Comparison of versions of Logo
- •Beyond the beginning: Logo syntax & grammar
- •Hands-on time for development of Logo projects
- •Assessing Logo learning
- •Writing for publication

Staff: Sharon Burrowes Yoder and Dave Moursund
Dates: July 16 - July 22, 1989
Location: Eugene, Oregon
Cost: $300 before May 1st; $350 after May 1st; covers 2 quarter-hours graduate credit, materials, & instruction.

For more information, contact Dave Moursund or Sharon Burrowes Yoder at the International Council for Computers in Education , 1787 Agate Street, Eugene, Oregon 97403 (503) 686-4414.

ICCE



About ICCE

The International Council for Computers in Education was founded by Dr. David Moursund in 1979 as an organization that would foster appropriate instructional use of computers throughout the world.

Today ICCE is the largest professional organization for computer educators at the precollege level. It is nonprofit, supported by 12,000 individual members and more than 50 organizations of computer-using educators worldwide. These organizations are statewide or regionwide in scope, averaging 500 members each. Approximately 84% of ICCE's individual membership is in the United States, 12% is in Canada, and the remainder is scattered around the globe.

About *The Computing Teacher*

ICCE publishes *The Computing Teacher* journal. *The Computing Teacher* provides accurate, responsible, and innovative information for educators, administrators, computer coordinators, and teacher educators. The journal addresses both beginning and advanced computer educators through feature articles, columns, software reviews, and new product and conference listings. Contributors to *The Computing Teacher* are leaders in their fields, consistently providing the latest information in computer education and applications.

Publications, Special Interest Groups

In addition to *The Computing Teacher*, ICCE provides a number of publications to computer-using educators. ICCE's Special Interest Groups provide in-depth information for computer coordinators, teacher educators, computer science educators, and Logo-using educators. *C.A.L.L. Digest* is published nine times per year for ESL teachers. ICCE committees address a variety of ethical and practical issues important to the computer-educating community.

Independent Study Courses

ICCE offers graduate-level independent study courses, designed to provide staff development and leadership. These courses have been approved by the College of Education at the University of Oregon and carry graduate credit from the Oregon State System of Higher Education. Participants correspond with instructors by mail.

Write for information and a free catalog today!

INTERNATIONAL COUNCIL FOR COMPUTERS IN EDUCATION ICCE

ICCE • University of Oregon • 1787 Agate St. • Eugene, OR 97403 • Ph: 503/686-4414