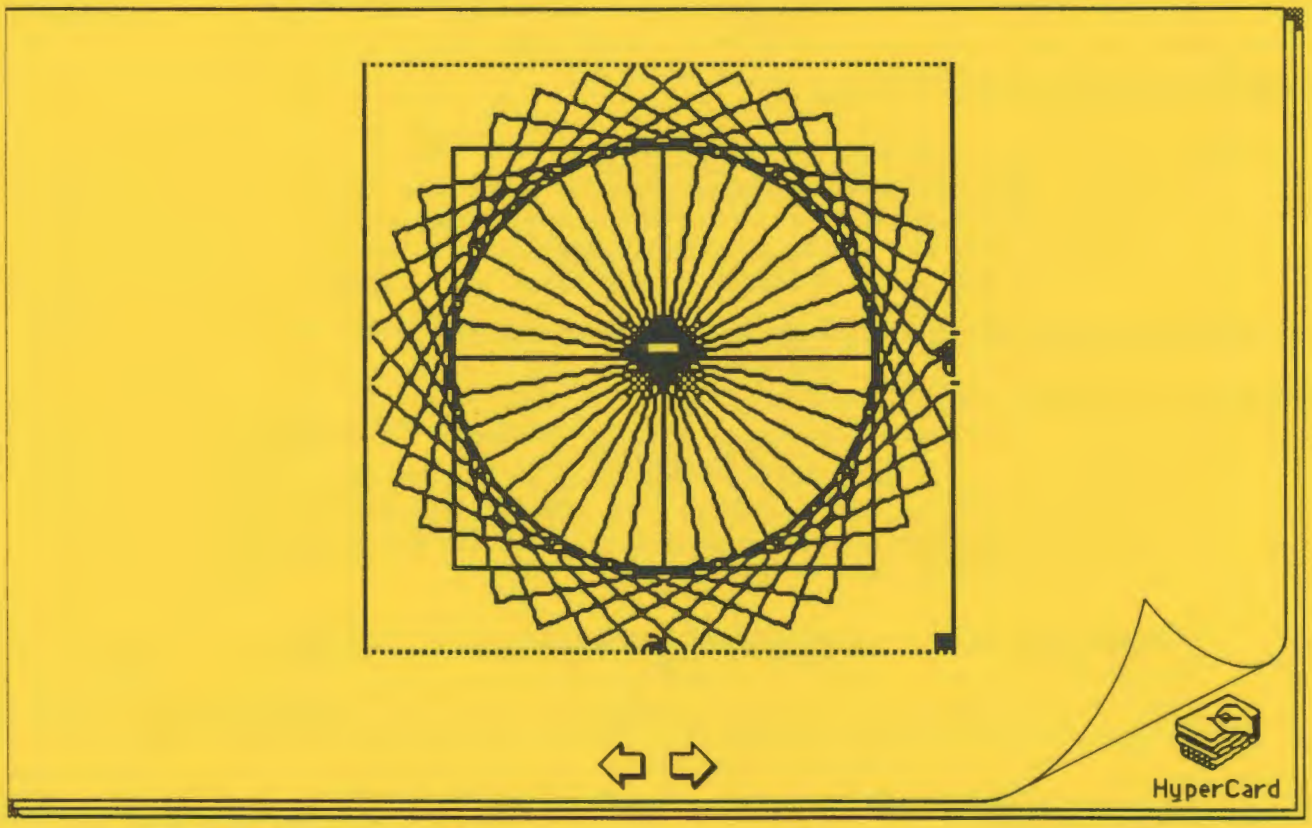
Journal of the ISTE Special Interest Group for Logo-Using Educators

LOGO EXCHANGE

May 1990 Volume 8 Number 9



International Society for Technology in Education

ISTE Publications

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 For Educators, a graduate level independent study course, allows you to learn at your own pace while corresponding with your instructor by mail.

WORK INDIVIDUALLY OR WITH A GROUP

Take ***Introduction to Logo For Educators*** at home, or study with a group of colleagues. The course uses video tapes (ON LOGO) with MIT's Seymour Papert, printed materials, textbooks, and disks. View the tapes, read and report on course materials, do projects, design Logo lessons for students, and correspond with instructor by mail.

NOT JUST ANOTHER CLASS

Dr. Sharon (Burrowes) Yoder, editor of the *Logo Exchange* journal, designed ***Introduction to Logo For Educators*** to provide staff development and leadership training. The four quarter-hour course meets the standards of the College of Education at University 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 districts to build libraries at multiple sites.

Group Enrollment. A tuition of $260 per participant is available to institutions that enroll a group of six or more educators. This special price does not include the ON LOGO videotapes. Your group must acquire the tapes or have access to them. Once acquired, the library of tapes and materials may be used with a new groups enrolling for the same reduced fee.

Individual Enrollment. Educators with access to the tapes may enroll indiviidually for $290. Tuition including tape rental is $320. A materials fee fo $60 per enrollee is charged for texts and a packet of articles. Enrollees who already have the texts do not neet to order them.

Tuition Information, Detailed Course Outlines, and Order Blanks can be obtained from:

LONG DISTANCE LEARNING, ISTE, University of Oregon,
1787 Agate St., Eugene, OR 97403-9905.
Phone 503/346-4414

LOGO EXCHANGE

Volume 8 Number 9 | Journal of the ISTE Special Interest Group for Logo-Using Educators | May 1990

Logo Exchange is the journal of the International Society for Technology in Education Special Interest Group for Logo-using Educators (SIGLogo), published monthly September through May by ISTE, University of Oregon, 1787 Agate Street, Eugene, OR 97403-9905, USA; 503/346-4414. This publication was produced using Aldus *PageMaker®*.

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

Contents

ISTE Membership

	U.S.	Non-U.S.
	28.50	36.00

SIGLogo Membership (includes *The Logo Exchange*)

	U.S.	Non-U.S.
ISTE Member Price	25.00	30.00
Non-ISTE Member Price	30.00	35.00

Send membership dues to ISTE. 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

The Logo Exchange ...or the *Journal of Learner Based Tools?*

It's a blustery day on the Oregon coast. The clouds and the sun have been playing tag over an ever changing ocean as I have prepared the columns and articles for this year's final issue of *LX* . Only 10 years ago, my typewriter would have been surrounded by piles of paper, and my desk would have been littered with eraser dust. But it's 1990. Now my Macintosh desktop shows a well-organized group of word processed documents ready for the production department. Those neat rows of documents are deceptive, though. Much as the soaring of the gulls belie the strength of the ocean winds, those Macintosh icons mask the potential turmoil that is hidden within several of my documents.

Trouble in Logo-land?

The material is this issue of *LX* runs the range from immediately useful ideas for classroom teachers to deep philosophical discussions; from Logo as a tool to Logo as a programming language. This May issue clearly highlights the range of points of view in the Logo community, a range that makes it increasingly difficult to meet the needs of every reader in one journal.

Over the past couple of years, I have seen an increasing division occurring in the Logo community. One group feels strongly that *LX* should contain only teacher-centered articles that provide activities and ideas that can be used in the classroom "tomorrow morning." Nothing but Logo should even be mentioned in these pages. Others feel just as strongly that it is the philosophy of learning that Logo represents that is most central and whatever fits that philosophy belongs in the pages of *LX* .

This current issue clearly highlights these two points of view. On the one hand we have the classroom-oriented work of Eadie Adamson and Dorothy Fitch. These two columns provide ideas that you can use in your classroom right away. On the other hand we have the "Logo Connections" column that gives us turtle graphics in *HyperCard*; Sandy Dawson's philosophical article about Logo, math, and technology; and the Harris, Bull, and Bull article, "The Gears of Childhood." These authors have chosen not to provide Logo activities for the classroom. In particular, "The Gears of Childhood" provides a thoughtful insight into ideas that are growing in some segments of the Logo community and beyond.

Where Are We Going?

Do the contents of this month's *LX* mean that the *Logo Exchange* as we know it will to cease to exist? Will it suddenly be filled with *HyperCard* articles and philosophical discussions? Certainly not! As we all know, *LX* was founded to provide a source of ideas for Logo-using teachers. That mission will continue. We fully expect to fill many pages next year with practical material such as that which Eadie, Dorothy and a number of our readers provide.

But what of the idea, expressed in a number of places this month, that *LX* should become the *Journal of Learner Based Tools?* It is certain that the name of *LX* is not going to change next year, but it is also certain that the contents will continue to reflect the changes that are taking place in the world of computer education. There will still to be occasional articles about Logo-like environments and the column "Logo Connections" will highlight the relationship between Logo and the "outside" world.

Rather than changing the name of the Logo *Exchange* it may be more accurate to think of *LX* as being one sub-part of a journal that does not yet exist. The *Journal of Learner Based Tools* may be an "umbrella" that encompasses part of the newsletters of the SIG's for telecommunications and hypermedia as well as parts of the *Logo Exchange*. Perhaps, as Judi, Gina, and Glen suggest, they are all part of a SIGTools. Perhaps some day that journal or SIG may actually exist. For now, however, we'll have to content ourselves with three separate publications to touch on the various Logo-like ideas. In this ever-growing and ever-changing field, it is hard to know what will happen in the years to come. Just as I no longer use a red pen and a typewriter, the *Logo Exchange* must continue to grow and change to reflect the world around us.

But you can read this issue and draw conclusions for yourself. There is certainly food for thought in these pages.

And What of Next Year?

There will be some changes in *LX* next year. You will be receiving eight 36-page issues instead of nine 32-page issues. You will still receive *LX* from September through May, but the December and January issues will be combined into one.

We will be welcoming back most of the familiar columnists for the coming year, so you will continue to receive a rich variety of ideas each month. We have planned for two "theme issues" next year. In October we will again have a "Just for Beginners" issue; in April, a "Logo Connections" issue. Instead of an "Experts" issue, next year we will be adding a column called "Extra for Experts." This column will serve as

a forum for the more advanced articles that we publish in *LX*. The column will be edited by Mark Horney. Mark is the SIG coordinator for ISTE and will be completing his PhD here at the University of Oregon in the area of computers in education in the near future.

What Can *YOU* Do?

The *Logo Exchange* is what you make it. We depend on your contributions to fill its pages. While the columnists provide the backbone of each issue, without articles and ideas from those of you who read *LX* ,we could not survive. So, as the school year is winding down and summer begins, take some time to write to me about your work with Logo. Even if you've never published an article before, I'll be glad to work with you to develop your ideas. Or perhaps you or your students have some art that we can use on the cover? Maybe you just have a neat idea that works with kids. **Any and all ideas are welcome.** (Send them to me at the address given below.) The deadlines for each issue next year are given below.

Issue	Due Date
September	June 25
October (Just for Beginners)	July 30
November	August 27
December/January	October 1
February	October 29
March	December 15
April	January 28
May	March 4

So, plan to spend time this summer relaxing. Refresh yourself in whatever way works best for you. But add several things to your "to do" list this summer. Reflect on the issues raised in this month's *LX*. Consider reading a good book on Logo (several are mentioned in this issue.) Drop me a line about your work. Send me some of your student's work. Share with others your excitement about Logo.

In any case, have a great summer! I certainly hope we'll see you in the fall.

Sharon Yoder
SIGLogo/ISTE
1787 Agate Street
Eugene, Oregon 97403

CIS: 73007,1645 BITNET: Yoder@Oregon

Monthly Musing

Qwerty Revisited
by Tom Lough

When I got to the bottom of page 32 during my very first reading of Seymour Papert's *Mindstorms*, I stopped. That is where I encountered the term "QWERTY phenomenon" for the first time in print. It certainly gave me something to think about.

Like most of you, I was familiar with the idea. How something is done for the first time seems to have an incredible effect on how the same thing (or a related variation) is done forever more. This forms the basis for important values such as tradition and heritage.

However, things can be carried too far. For example, when innovative methods or new ideas are discarded without consideration because a policy or routine must be followed rigidly, then the sense of balance may be a little out of kilter.

I appreciated Papert's application of the QWERTY phenomenon to the state of educational computing as it existed in the early 1980's and welcomed Logo to the scene. But, more than the language itself, I welcomed the teaching and learning philosophy which Logo facilitated. To me, this resonance was the critical feature.

Since the founding of this publication, I have sensed this resonance several times with the manner in which certain other software packages could be used. What could it called? The term "Logo philosophy" seemed much too narrow. In discussions with Glen Bull, Steve Tipps and others, a name for this essence proved elusive.

Named or not, the idea was never far from mind. During the planning for the East Coast Logo Conference (held April 2-4, 1987), we felt it would be important to include many additional events which were not directly related to Logo, yet which possessed the essential elements of this resonance. As many of you may recall, the program included juggling, singing, dancing, music teaching, a talking word processor, and several other activities. In fact, the members of the conference evaluation team commented on the unusual but delightful diversity of the conference program. When HyperCard was released, I felt this same sense of resonance. (See From the Editor, *LX*, Feb - Apr 1988)

At about that same time, I began to hear the term "learner-based tools" being used with increasing frequency. Glen Bull and I came to feel that this term seemed to embody the main ideas which we felt were most important about the "Logo philosophy." We discussed what impact this idea might have on the *Logo Exchange*. Was the magazine primarily about Logo or was it about how Logo could be used? Would it be appropriate to extend this to cover how other products could be used in a "Logolike" way?

The article by Glen and Gina Bull and Judi Harris on page 11 of this issue is an extension of this discussion, brought forward several years into the context of the present. The heart of the matter seems to be the service mission of the magazine. Should it be focused on the needs of those using Logo? After all, it is the official publication for SIGLogo. Or, should it focus on the application of a Logolike philosophy and seek to serve a greater proportion of educational computing? What implications would this have for SIGLogo?

I, for one, would welcome the transition of this publication into an exchange to serve teachers using technology and learner-based tools in a "Logolike" manner. To be sure, the needs of Logo teachers should certainly continue to have high priority. (For me, Logo is a near-ultimate learner-based tool.) However, I believe that the needs of a larger group could be served by the enrichment of teaching ideas and applications from a wider variety of perspectives. To do otherwise may be the same thing as applying the QWERTY phenomenon to this publication.

I expect that the NECC Preconference Symposium on Logo (see details on page 21) will be rather spirited as this issue is discussed. If you are coming to NECC, I hope that you will attend this important meeting.

As ever, FD 100!

Tom Lough
Founding Editor
PO Box 394
Simsbury, CT 06070

About the Cover

This month's cover art was produced by a *HyperCard* turtle. See the article by Glen and Gina Bull on page 23 for details.

Logo Ideas

Thinking Non-mathematically About Mathematical Physics Problems

by Eadie Adamson

One of the important characteristics of Logo is that you can easily play around with procedures: changing inputs, directions and even the order of events. A student can pose many "what-if" questions without directly confronting the underlying mathematics. This "mathworld" is akin to the immersion we all go through as we learn language. A Logo "mathworld" can provide a playground to encourage mathematical thinking very indirectly as procedures are manipulated and the resulting visual display is closely observed. I tend not to think of myself as a mathematician, although I recall that in school I was an excellent math student...until I got to statistics. However, I do like to play around intuitively with ideas that are often strongly related to mathematics. I don't apply any lengthy theorems or rules that I have learned. I take a guess, look at the result, compare it with my mental picture of what I want to see. (My mental model may not always be accurate, but nonetheless....) Then I make further changes. Similarly, I make no claim to much knowledge about physics, but....

Recently in class we talked about the idea of simulating a bouncing ball with Logo. Here was an opportunity to explore a little physics and a little mathematics. I began with a bit of experimenting from my own point of view. Now, I knew about the parabolic curve which a bounce might follow, but I also knew that the screen was too narrow for that to be a good visual simulation. (I had already experimented with adapting some ideas about programming a bouncing ball that I found in James Hurley's *Logo Physics*.) A bounce which traveled in a parabolic curve could easily wrap the screen several times before a ball rolled to a stop.

The first idea my students had was a simple bounce. The turtle was changed to the ball shape with its heading at 0:

```
to ball1
pu
setpos [-125 -80]
setsh 12
forward 10
ball1
end
```

If you try this procedure you will find that it makes a never-ending bounce. There's no elasticity in the ball at all. The height of the bounce never changes and it bounces in exactly the same spot each time. Most of us know from simple observation that this is not an accurate representation of a bouncing ball. My students decided to try an adjustment to make the ball move. We assumed it would travel across the screen from left to right, bouncing along the white bar at the bottom of the LogoWriter page.

Ball2 requires an x-coordinate as an input for a starting point. This gave us a little more flexibility in just where a bounce might begin:

```
to ball2 :xcor
pu
setpos list :xcor -80
setsh 12
forward 10
wait 2
ball2 :xcor + 5
end
```

Using an input of -120, **ball2** begins near the left side of the screen. As we watched the simulated ball moving across the screen, someone observed that a real ball's bounce gets smaller the longer it bounces. The force is diminished with each bounce. We tried it with a real ball. Could we change our procedure to simulate that?

We decided we needed an input for **bounce**. The input, which we named **bounce**, would diminish each time. Also, the boys wanted the ball to move more slowly. The adjustments looked like this:

```
to ball3 :xcor :bounce
pu setpos list :xcor -80
setsh 12
repeat :bounce / 2 [ forward 2 ]
wait 2
ball3 :xcor + 5 :bounce - 1
end
```

Better, but, whoa! Now we get an error message:

```
repeat doesn't like -0.5 as input in
  ball3
```

What was happening? The value of bounce was getting below zero. When the value of **bounce** became zero, the boys thought, the ball should stop. Instead, since the amount of **bounce** was an input to repeat, Logo was balking at a negative input for repeat. In earlier work with polygons, the boys had learned to write stop rules for recursive procedures. They realized that something similar needed to be applied here. We came up with a rule like this:

```
if :bounce = 0 [stop]
```

Where should it go? The boys recalled drawing "**growsquare**" which increases in size with each recursive call and checks the size before drawing the next square:

```
to growsquare :size
if :size > 85 [stop]
repeat 4 [forward :size right 90]
growsquare :size + 5
end
```

The bouncing ball procedure should also check first if the bounce was zero. If so, the procedure (and the bouncing) should stop. The new procedure looked like this:

```
to ball3 :xcor :bounce
if :bounce = 0 [stop]
pu
setpos list :xcor -80
setsh 12
repeat :bounce / 2 [forward 2]
wait 2
ball3 :xcor + 5 :bounce - 1
end
```

Well, that does stop the bouncing, but we compared what happened to a real bouncing ball. A ball tends to roll a bit after it stops bouncing. How to add that? And when? We watched a bouncing ball for a while and thought about what happened "procedurally." The bounce needed to stop before the roll began. We decided first to try writing a simple roll procedure:

```
to roll
setx xcor + 1
wait 1
end
```

Typing roll and pressing Return did not get the ball very far. Someone suggested using a **repeat**:

```
repeat 10 [roll]
```

That looked a little better. Now we had a real problem: how could we connect it to the bounce procedure? Someone thought of putting the word roll at the end of the bounce, right before the word end. Some people said, "But that will look just like it did when we just typed roll and pressed Return. We should use a repeat also." They decided to try it and see what would happen.

```
to ball4 :xcor :bounce
if :bounce = 0 [stop]
pu
setpos list :xcor -80
setsh 12
repeat :bounce / 2 [forward 2]
wait 2
ball4 :xcor + 5 :bounce - 1
repeat 10 [roll]
end
```

The roll seemed remarkably long! What was happening here? Time to think about recursion! Although the boys had used recursion previously, they had no way to relate to the complication they had just injected into the situation. Rather than try to give a long verbal explanation of what was happening, I decided that having Logo report on its progress would help explain what was happening. I suggested that we add a line to the end of the roll procedure, so that roll now looked like this:

```
to roll
setx xcor + 1
wait 1
show "FINISHED!
end
```

To the boys' surprise, when we tried the bounce procedure again, we got a long string of FINISHED! Now I could explain a little more about recursion. In fact, we could now simulate it with a little recursion-play to make the process clearer. Here another command was added after the recursive call and before the end statement. This kind of recursion is called embedded recursion, as opposed to the tail recursion used in earlier procedures. Everyone began to see that embedded recursion had a delightful but unexpected result as each "Little Person" finished its task (to use Brian Harvey's analogy from his *Computer Science Logo Style*).

The boys finally concluded that the **repeat** command for **roll** was unnecessary, for in fact, it made the roll far too long. Now they could understand a little better why that was so. We eliminated the **repeat** so that our **bounce** now looked like this:

```
to ball4 :xcor :bounce
if :bounce = 0 [stop]
pu setpos list :xcor -80
setsh 12
repeat :bounce / 2 [forward 2]
wait 2
ball4 :xcor + 5 :bounce - 1
roll
end
```

Once satisfied with this partial "cure," we eliminated the **show** at the end of the roll procedure.

Notice how many times the students have made real-life observations, made adjustments in procedures, and compared

the simulation with the real-life action. Also notice that the conversations are not so much involved with the formal mathematics or the physics as with the changes that will make the simulation more real for them. They made their own mistakes and then, when they saw the result, did the correcting themselves. The rules they followed were those needed to write procedures using what Logo grammar they knew, but they were using this skill to get a visual result, and then they reformulated their ideas based on the results.

We have not yet carried this experiment much further, but in what we have created thus far we have a beginning model, not yet "correct." The students have learned about observation, adjustment of theories, about the difference between a simulation and a real-life action. They know that the "bounce" is still not perfect, but now they can begin to think more about the nature of a bouncing ball from a different perspective. They may even pay more attention when a ball bounces now, recalling the experience of trying to program a simulation. They have also gotten a good look at a new way to use recursion, one which they have not encountered before. Soon someone will undoubtedly want to make an animation which lets the ball shape "squish" when it hits the ground and then expand as it rises. When the students examine the procedure closely, they will discover that the ball moves forward after it bounces. Their own observations will be enough to tell them that this is not what happens, that instead the ball moves forward while in the air. They will need to find a way to express this in Logo.

The kind of learning these students have experienced, in which we did not attempt to construct any "laws" of physics, nonetheless has been powerful learning. They have been experimenting with very sophisticated ideas. It has placed them, as fifth graders, in a position to be remarkably more receptive to some ideas about physics in the world about them. In the context of their explorations they have drawn on their own knowledge, made observations, and attempted to express them in a Logo procedure designed to simulate the real world action in two dimensions. In addition to their growth in understanding of the world about them, their sophistication in working with recursion has undoubtedly been advanced a bit.

References

Harvey, Brian (1985). *Computer science Logo style: Volume I.* Cambridge: MIT Press.

Hurley, James P. (1985). *Logo physics.* New York: Holt, Rinehart and Winston.

Eadie Adamson
1199 Park Avenue, Apt 3A
New York, NY 10128

The Beginner's Corner

A Summer Program of Words and Lists

by Dorothy Fitch

When Logo users who are comfortable with turtle graphics first encounter words and lists, their reaction is often something like "Words and lists are too complicated," or "What in the world would I use them for?"

Words and lists really aren't all that complicated, but it may be helpful for you to see some examples where you know in advance what the programs are going to do. The goal of this column is not necessarily to teach you lots of things about words and lists, but to familiarize you with what word and list commands will allow you to do and why you might want to use them.

Here's a scenario that will help introduce you to words and lists:

You have volunteered (or been volunteered) to print up song sheets for sing-along time at summer camp this summer. As you sing the songs to yourself, you wonder if there isn't some better way to print out the lyrics than to sit down at the typewriter (or word processor) and type every word of every verse of every song. That's the right idea! If you can find repeated patterns, then you can get Logo to help you out.

These particular songs have been selected not because they are the campers' favorites, but because they represent different song forms that lend themselves to being programmed, especially in Logo. With song lyrics as a basis, you can learn about musical form and words and lists—they go together perfectly.

One of my favorite college courses explored folklore. We studied folk songs, stories, hop-scotch patterns, and counting-out rhymes. They are universal–timeless–and found in fascinating variations throughout the world. As I later learned Logo, thinking about folk song forms helped me understand words and lists. Maybe it will help you!

Let's see how words and lists can be used to print song lyrics of various types using Logo. In these examples, any text following a semi-colon contains explanations and comments to you. You do not need to enter them into your program (although you can if you want).

Circular Form

The first text is not really a song, but a rhyme that originated in Wisconsin, home of many Scandinavians. It goes something like this:

My name is Yon Yohnson,
I come from Visconsin,
I vork in the lumberyards there.
Oh, the people I meet,
As I valk down the street,
They say "Hello"
I say "Hello"
They say "What's your name?"
I say,
My name is Yon Yohnson,
I come from Visconsin, (etc.)

Other example of songs in circular form are the perennial favorites "There's a Hole in the Bucket" and "Found a Peanut."

Using Logo

Printing the lyrics to this rhyme is easy. Each line of text is printed to the screen using the PRINT command. Then to keep the rhyme going forever, use tail recursion, which calls the same procedure again. In the following example, the last instruction calls itself. Since there is nothing to stop the program, it will continue forever. Type the procedure and then type GREET to watch it run. To pause the text as it is printed to the screen, press Control-W; continue it by pressing any key. Stop the program with Control-G.

```
TO GREET
PRINT [My name is Yon Yohnson,]
PRINT [I come from Visconsin,]

 continue in the same way with rest of rhyme...

PRINT [They say, "What's your name]
PRINT [I say,]
GREET          ; this recursive call makes the song circular
END
```

Verse and Refrain

A common song form is that of verse and refrain (or stanza and chorus). One familiar example is Yankee Doodle. Here's a program that prints a couple of verses and refrains. Although you have to type the words for each verse, you only have to enter the refrain once! You can add other verses in the same manner.

Using Logo:

```
TO YANKEE
 VERSE1
 REFRAIN
 VERSE2
 REFRAIN
END

TO VERSE1
 PRINT [Verse 1:]
 PRINT [Father and I went down to camp]
 PRINT [Along with Captain Gooding,]
 PRINT [And there we saw the men and
   boys]
 PRINT [As thick as hasty pudding.]
 PRINT []   ; this prints a blank line
END

TO REFRAIN
 PRINT [' ' Yankee Doodle keep it up,]
   ; In MIT Logo dialects, you
 PRINT [' ' Yankee Doodle dandy,]
   ; can include spaces inside
 PRINT [' ' Mind the music and the step
   and]   ; apostrophes
 PRINT [' ' With the girls be handy!]
 PRINT []
END

TO VERSE2
 PRINT [Verse 2:]
 PRINT [And there was Captain
   Washington]
 PRINT [Upon a slapping stallion]
 PRINT [A-giving orders to his men,]
 PRINT [I guess there was a million.]
 PRINT []
END
```

Progressive Chain

In a progressive chain song, there is a verse about each item in a sequence, such as numbers, days of the week, months of the year, etc. In this familiar counting song, the numbers decrease to zero, at which time the song ends.

Using Logo:

Here is an efficient way to print all the words to 100 Bottles of Beer (or pop, or whatever) On the Wall using just one procedure:

```
TO BOTTLES :NUMBER
 IF :NUMBER = 0 THEN STOP
 (PRINT :NUMBER [bottles of beer on the
   wall,])
 (PRINT :NUMBER [bottles of beer,])
 PRINT [You take one down and pass it
   around,]
 (PRINT :NUMBER - 1 [bottles of beer on
   the wall.])
 PRINT []      ; prints a blank line
 BOTTLES :NUMBER - 1
END
```

Run the program by typing BOTTLES 100. This starts running the BOTTLES procedure with the value of :NUMBER at 100. Every reference to :NUMBER will be replaced by 100 when the program runs. The parentheses around the entire PRINT statement keeps the number and the rest of the sentence together.

At the end of the procedure is a recursive call to the same BOTTLES procedure, but the new value of :NUMBER will be the current value of :NUMBER (100) minus 1. Thus, the next verse begins "99 bottles of beer on the wall."

The conditional statement at the beginning (IF :NUMBER = 0 THEN STOP) keeps track of the value of :NUMBER and stops the program as soon as it becomes 0. If you leave that line out, you'll eventually be singing "-1 bottles of beer on the wall."

If you don't like the fact that it prints "1 bottles of beer", you can always make it print "bottle(s)." (Yes, it is possible to print "bottle" and "bottles" at appropriate times, but remember, this is a beginner's column; we'll leave fine tuning to the experts for now!)

Cumulative Song

In a cumulative song, there is a verse about each item in a sequence, just as in the progressive chain. However, in a cumulative song, none of the items ever gets thrown away. Each verse contains the current item, plus all that has come before. Examples of cumulative songs are "There's a Hole in the Bottom of the Sea," "Old MacDonald Had a Farm," and the seasonal favorite used in the example below. (If we sing it all summer, maybe we'll remember the words next December!)

Using Logo:

This song contains some constant information: the list of gifts and the ordinal number associated with each day (first, second, third, etc.). We could make these global variables (using a MAKE statement), but these really aren't variables; they're constants. They don't change during the running of the program. So, the program below includes procedures that output a list for the GIFTS and DAYS. As you become a more

experienced programmer, you will write more and more procedures that output information (also called operations or reporters). They are very powerful types of procedures and help make your programs much easier to write and read.

This program "sings" The Twelve Days of Christmas. Type 12DAYS to begin, after entering all the procedures.

```
TO 12DAYS; this is the main program, which simply calls
 SING 1   ; the SING procedure with a starting value of 1
END

TO GIFTS  ; this is the procedure that stores the list of gifts
 OUTPUT [[and a partridge in a pear
   tree.] [2 turtle doves,] [3 French
   hens,] [4 calling birds,] [5 golden
   rings,] [6 geese a-laying,] [7 swans
   a-swimming,] [8 maids a-milking,] [9
   ladies dancing,] [10 lords a-leap
   ing,] [11 pipers piping,] [12 drum
   mers drumming,]]
END

TO DAYS   ; this is the procedure that stores ordinal numbers
 OUTPUT [first second third fourth
   fifth sixth seventh eighth ninth
   tenth eleventh twelfth]
END

TO SING :DAYNUMBER
    ; this is the main part of each verse
 IF :DAYNUMBER > 12 THEN STOP
    ; stops when number 13 is reached
    ; the next line looks up the current number in DAYS (first,
          second, etc.)
 ( PRINT [On the] ITEM :DAYNUMBER DAYS
   [day of Christmas] )
 PRINT [my true love gave to me:]
 ; this next line calls GET.GIFT to show the current list of gifts
 ; remember :DAYNUMBER is the current day number
 GET.GIFT :DAYNUMBER
 PRINT []        ; prints a blank line
 ; sings the verse with the next number
 SING :DAYNUMBER + 1
END

TO GET.GIFT :NUMBER
    ; This procedure prints the accumulated gifts for each verse
    ; The next line stops when there are no more gifts
 IF :NUMBER = 0 STOP
    ; The next line prints a space, then looks up and prints the gift for the
      current :NUMBER
 ( PRINT "\ ' ITEM :NUMBER GIFTS)
 ; The next line calls a copy of GET.GIFT with the previous
    day's number
 GET.GIFT :NUMBER - 1
END
```

The primitive ITEM (used in the SING and GET.GIFT procedures above) reports information from a list. It takes two inputs: a number and a list. The list used in the instruction in the SING procedure is the one returned by the DAYS procedure. The list used in GET.GIFT is the one output by the GIFTS procedure. If you type PRINT ITEM 3 GIFTS, for example, Logo responds with the word "three French hens." The GET.GIFT procedure starts with the current day and prints the gift for that day, the gift for the day before, the gift for the day before that and so on until it runs out of days. Then it goes on to the next verse. This program is rather complicated, but knowing what it is supposed to do makes it easier to understand.

I hope that these ideas will get you interested in learning more about words and lists in Logo. Over the summer, you may want to read one of the following books for further information and instruction:

Birch, Alison (1986). *The Logo project book: Exploring words and lists.* Portland, ME: Terrapin Software, Inc.

Goldenberg, E. Paul and Feurzeig, Wallace (1987). *Exploring language with Logo.* Cambridge, MA: The MIT Press

Have a great summer. See you next year!

Dorothy M. Fitch,
Director of Product Development
Terrapin Software, Inc.
400 Riverside Street
Portland, ME 04103
(207) 878-8200

The Gears of Childhood

by Glen Bull, Gina Bull, and Judi Harris

In his foreword to *Mindstorms*, Seymour Papert describes his childhood fascination with gears. At an early age he became intrigued by the movement ratios produced by rotating gear combinations of different sizes. Gear ratios became the method which he used to conceptualize the multiplication tables, and later differential gears became the model for building equations with two variables, such as $3x + 4y = 10$.

In Papert's terminology, gears served as a transitional object. They provided a concrete object that he understood which could be used as the basis for building active experiential understanding of a new abstraction: mathematical equations. Papert proposed that computers could be used similarly to provide inexpensive, flexible transitional objects. In essence, computers could be used to create "electronic gears." An electronic turtle on the computer screen could serve as a transitional object because children could relate the movement of the turtle on the screen to the movement of their own bodies in space. This might provide an accessible and motivating method for exploration of geometry and other mathematical concepts.

At the time that *Mindstorms* was written, this was a rather radical way of thinking about using the computer in education. Previously, the chief tendency had been to think of the computer as a mechanized replacement for the teacher, rather than as a tool that could be used by the teacher or learner as a bridge to other concepts.

Over the years, Logo and turtle geometry provided a rather durable transitional object. When Logo first experienced tremendous popularity, there arose a debate as to whether Logo was the only valid transitional object that could exist on the computer. This might be termed the phase of *Logo as religion.*

Enter: the Heretics

At about the same time, Bob Tinker was evolving the concept of microcomputer-based laboratories (MBL) at the Technical Education Research Centers (TERC). The concept of MBL is that rather than using the computer to *simulate* scientific experiments, probes and sensors attached to the computer could be used to collect actual data to conduct real scientific experiments. Many of these ideas were described in a column with the wonderful title of "Tinker's Toys" which he wrote for TERC's *Hands On!* newsletter. In one of these columns Bob noted that for some MBL applications Logo lacks the speed required to acquire some types of data in real time.

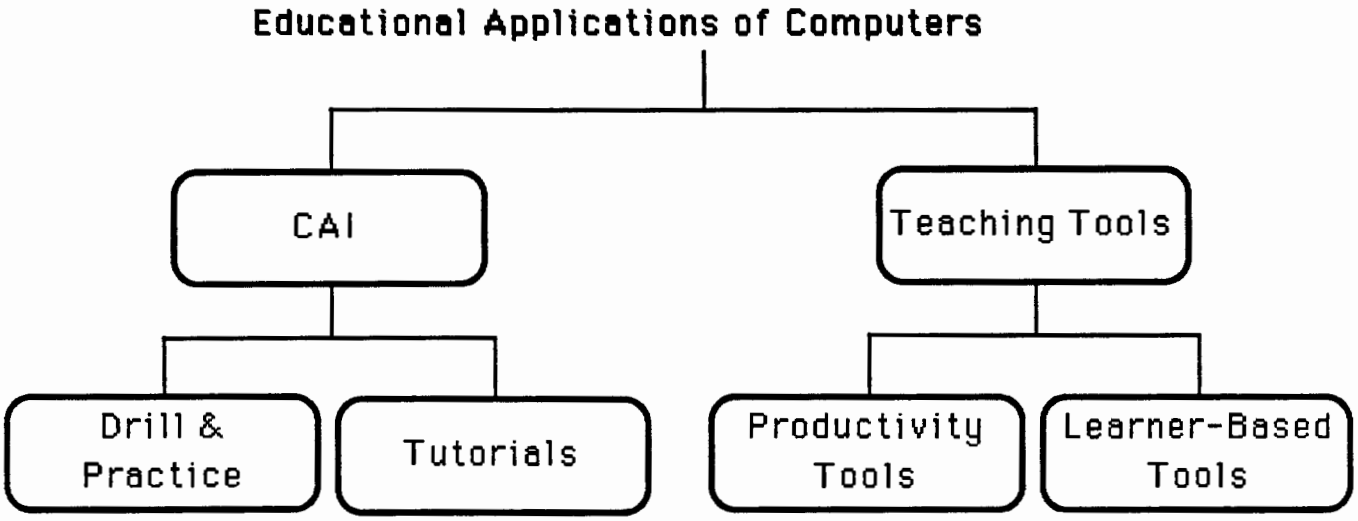
This was hardly a startling observation, since Logo is an interpreted language with moderate though not blinding speed when implemented on an eight-bit microprocessor. However, at the planning meeting for the first national Logo conference, this revelation was treated as pure heresy by some. In the midst of an active discussion, Hal Abelson (of M.I.T.) finally said, "Bob Tinker's a pretty good guy. If he's making these observations, maybe we should be listening to what he's saying."

In response to the discussion that followed, Abelson said, "Logo is not the only way." When challenged to list other approaches, Abelson thought for a moment, and then said, "Well, Boxer is a possible model." Boxer is a programming language inspired by Logo that uses windows (or boxes, hence the name "Boxer") as containers for procedures. At present Boxer requires the power of a Sun workstation with several megabytes of memory to run. But then again, when Logo was first developed, microcomputers had not even been conceived, and hence Logo was not generally accessible in the public schools either. Readers interested in the characteristics of Boxer can refer to the citation listed at the end of this article.

Hal Abelson probably did not expect casual comments made over coffee to be remembered nearly a decade later, but his remarks get at the essence of an important issue. "Is Logo the only valid transitional object, or are there other approaches?" Before considering this question, we would like to turn the clock forward a few years to a discussion held with Tim Riordon while walking on an Atlantic beach. At that time, we were considering the phenomenon of *Logo as dogma.*

Dogma in this case refers to blind adherence to an edict without a sensitivity to reasons underlying what may have been intended as a guideline rather than a stricture. Recursion is a good case in point. The capability for recursion is one of the more desirable characteristics of Logo. Thousands of words have been written about recursion in Logo. Some have interpreted this to mean that recursion should always be used and that loops should never be used. In point of fact, there are instances in which a loop is more efficient than a recursive procedure. For example, a loop may be appropriate in a condition in which an external switch or sensor attached to the computer is polled.

As we discussed this, we were bemused by the vehemence with which some at that time attacked the use of any non-recursive procedure, and spontaneously invented the concept of the *Logo police.* The Logo police, we decided, will monitor your behavior and break down your door at night and



carry your computer away if you are caught using a non-recursive procedure, or engaging in any other form of un-Logolike behavior. Although the Logo police are a myth (as far as we know), at times it does seem as though the rigid application of dogma stifled the very creativity and innovation that Logo was designed to express.

CAI and Learner-Based Tools

An important aid to our thinking about this issue soon emerged from the Technical Education Research Centers. This aid was the notion of *learner-centered software* (Mokros and Russell, 1986; Russell, 1986). At that time Susan Jo Russell, a researcher in special education, was concerned about the almost exclusive use of Computer Assisted Instruction (CAI) applications in the field of special education. She and her colleagues completed a national survey and found that at that time almost all uses of the computer in special education classrooms consisted of CAI applications such as drill-and-practice activities. They found that although teachers expressed interest in more open-ended uses of the computer, such as Logo, they found it difficult to measure their effects, and therefore difficult to justify their use. Russell and company suggested ways in which teachers could extend their own notions of instructional objectives to include goals for learning which were less content-specific, more process-oriented, yet just as documentable as those for CAI. Russell (1986) took the opportunity to point out that computers had the potential to cause teachers to reconsider not only how they were teaching, but also what they were teaching.

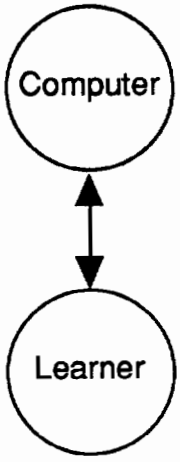
Russell and other researches at TERC described learner-centered software as an alternative to CAI which "has particular pedagogical characteristics which place more cognitive control in the hands of the learner" (Mokros and Russell, 1986, p. 185). For several years we have been using a modification of their term: *learner-based tools*. By this we mean software that is open-ended (having a variety of flexible outcomes) and learner-centered. Generally such software is distinguished from "utilities" or "productivity tools" by the potential for exploration or discovery. Logo is an example of a learner-based tool. Other early examples, we realize in retrospect, were attractive to us because they seemed Logo-like when they first came to our attention. They include Bob Tinker's micro-based labs as well as the talking word processing software developed by Teresa Rosegrant (Rosegrant and Cooper, 1985;1987.)

Interactive Structures

Good CAI programs can be very effective teaching devices. Unfortunately, good CAI can be expensive to produce. It has been estimated that it may take 200 hours of development to produce one hour of effective CAI. At that rate, one year of work is required to produce just ten hours of CAI. Another problem can be that the program may not present the information as the teacher would have preferred.

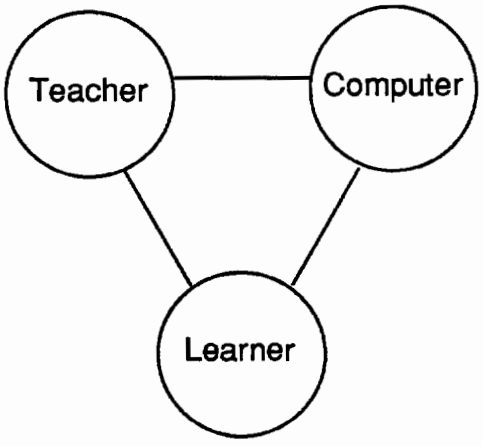
In traditional CAI, the computer replaces the teacher to some extent. In a drill-and-practice application, the computer drills the student on facts that have been taught by the teacher. In a tutorial program, the computer actually teaches the fact as well as directs and evaluates the student's practice efforts. There is a two-way interaction that occurs between learners and computers when students use CAI.

Two-Way Interaction Computer-Assisted Instruction



In contrast, effective use of a learner-based tool requires a three-way interaction between the teacher, the computer, and the learner (Bull, Cochran, and Snell, 1988; Bull, Lough, and Cochran, 1987; Cochran and Bull, 1985). Logo is a good example of this type of interaction. The turtle is used by the teacher to illustrate the relationship of a new concept to existing knowledge.

Three-Way Interaction Teaching Tools



Once the learner understands the relationship, the student can use the computer environment to explore the terrain. However, it is necessary for the teacher to provide "nudges" from time to time that edge the student into arenas that are apt to lead to productive discovery. One of the frequent misconceptions about this type of learning is that it is a "hands-off" process in which the teacher allows the student to discover concepts independently and rather randomly. It is to be hoped that discoveries of these types will occur, but facilitation by the teacher is a necessary and integral part of the process.

There are several ways to distinguish CAI from learner-based tools in the classroom. One is to ask whether a two-way or three-way interaction is occurring. Another important way is to determine whether the application is extensible. In some respects, a good CAI program is like a cleverly designed maze. If the program has been well-designed, the user will never be aware that there are boundaries beyond which it is not possible to go. However, CAI programs are of necessity finite with fixed boundaries. (It is possible that the area of artificial intelligence (AI) will change this, but AI applications are unlikely to affect public education in this century.)

In contrast, learner-based tools are always *extensible*. This means that the user can create uses and applications of the tool that were not envisioned by the developer. The extensible quality of these tools shifts the locus of control to the learner, which accounts for the derivation of the term *learner-based* tool. Logo is one such tool. Are there others?

Building a Learner-Based Toolkit

We suggest that *Boxer*, and certain uses of programs such as *Talking TextWriter*, as they were envisioned by Teresa Rosegrant, qualify as learner-based tools. Moreover, many hypermedia applications, as they have evolved over the last decade, could meet the criteria for this type of tool, depending on their use. In a hypermedia system, learners can move within a universe of knowledge, creating trailmarks and links as they go. Because the user directs the direction of travel, the locus of control is with the learner. It is true that the body of knowledge is finite. However, the 650 megabytes of information that can be placed on a CD-ROM (for example) constitute a rather large conceptual universe. For purposes of comparison, it can be noted that everything that Shakespeare wrote will fit in 7 to 8 megabytes of space.

HyperCard, written by Bill Atkinson, popularized the current interest in hypermedia programs, and now numerous hypermedia programs are available on almost all brands of computers. Bill Atkinson acknowledges that Logo was one of the inspirations that he drew upon as he was developing *HyperCard*. *HyperCard* extends the concept of "object-oriented programming" (OOP) which is found to a lesser extent in Logo. Turtles and sprites are examples of objects that can be programmed in Logo. Once the turtle is assigned characteristics (heading, pen color, etc.) it maintains those characteristics until they are changed. Some types of sprites even can be assigned a velocity so that they stay in motion until told to stop. *HyperCard* has many more objects that can be

programmed in this way, serving as actors in a "script" created by the user. Certainly it is possible to do many of the same kinds of exploratory, extensible, learner-centered activities in *HyperCard* as in Logo. Hence, we believe it qualifies for the term "learner-based tool."

These types of similarities aside, we have found that teachers who use Logo can quickly develop similar teaching tools with *HyperCard*. Elsewhere in this issue of *Logo Exchange*, an article even describes a means of creating of a *HyperCard* turtle, illustrating the range and flexibility of this environment. Naturally Logo and *HyperCard* each have certain strengths that are not present in the other; they are not clones. However, we believe that the uses to which these programs can be put are similar enough that they should be placed in the same phylum.

Shall We Concentrate Upon the Tools or Their Use?

Learner-based hypermedia applications share another important characteristic with Logo. Although the learner may have almost unlimited directions to travel, it is likely that the journey will be facilitated by the presence of a guide who has previously covered the terrain.

The issue of interest here is not so much the tools themselves but their actual use. It is possible to write a drill-and-practice or tutorial program with both Logo and hypermedia programs. In fact, many teachers' initial explorations of Logo often involve development of tutorial programs. Some mathematics books for elementary grades contain examples of drill-and-practice programs written in Logo. Therefore it might be more accurate to say that Logo can be used as a learner-based tool than to say that Logo itself is such a tool.

The question we raise is whether the issues discussed in the Logo Exchange should have to do only with the programming language Logo, or a philosophy of teaching. We believe that the *Logo Exchange* is more about a pedagogic philosophy than about programming methods in a specific language. If Papert had focused on the specific transitional object of gears rather than the larger educational issues, he might have spent a lifetime refining more and more sophisticated gear systems that could be brought into the classroom as educational toys. This could have led to classics such as *Gear Storms: Children, Cogs, and Powerful Machines.*

In the early part of this century many carriage manufacturers defined their task as production of horse buggies and subsequently went out of business. Others defined their business as transportation and survived. This is why every car made by General Motors has "body by Fisher" (a large carriage manufacturer that survived) embossed on a plate on the door frame. At this juncture the issue of whether we are more interested in Logo as a programming language or Logo as an approach to teaching with computers is a crucial one.

Logo's Evolutionary Pattern

During the 1970s an almost unnoticed revolution occurred in the field of paleontology. This revolution, described by Stephen Jay Gould in his book, *Wonderful Life: The Burgess Shale and the Nature of History,* resulted in a reexamination of the history of life, including our own evolution. Gould describes the shift in thinking that occurred in the following way,

> ... in an error that I call 'life's little joke' (Gould, 1987), we are virtually compelled to the stunning mistake of citing unsuccessful lineages as classic 'textbook cases' of 'evolution.' We do this because we try to extract a single line of advance from the true topology of copious branching. In this misguided effort, we are inevitably drawn to bushes so near the brink of total annihilation that they retain only one surviving twig. We then view this twig as the acme of upward achievement, rather than the probable last gasp of a richer ancestry. (Gould, 1989, p. 35)

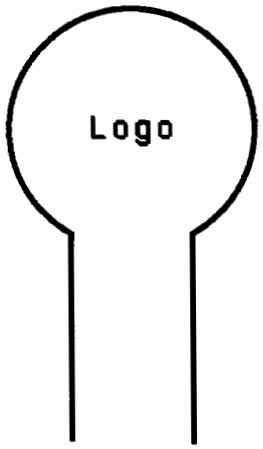
Gould notes that the view of evolution as an inexorable march of progress, culminating with the person telling the story at the peak of the evolutionary ladder, is an appealing one. However, he observes that the real success stories of mammalian evolution–such as bats, antelopes, and rodents–are the ones which present us "with thousands of twigs on a vigorous bush." Would it be more productive to consider Logo as the terminating event in the evolution of educational software, or should it be considered as one branch in a broader lineage?

One perspective is that Logo lies at the peak of a software evolution, unrelated to any other lineages. Rather than thinking of Logo as the end of an evolutionary line, we find it more productive to think of it as one of many examples of a thriving lineage of learner-based tools. Since Logo was one of the first of these educational tools, we will always have a strong interest in its use. However, there are now many interesting companion tools to explore as well. Thus we move from the *Logo as Religion* through *Logo as Dogma*, to the *Logo as Exemplar* phase.

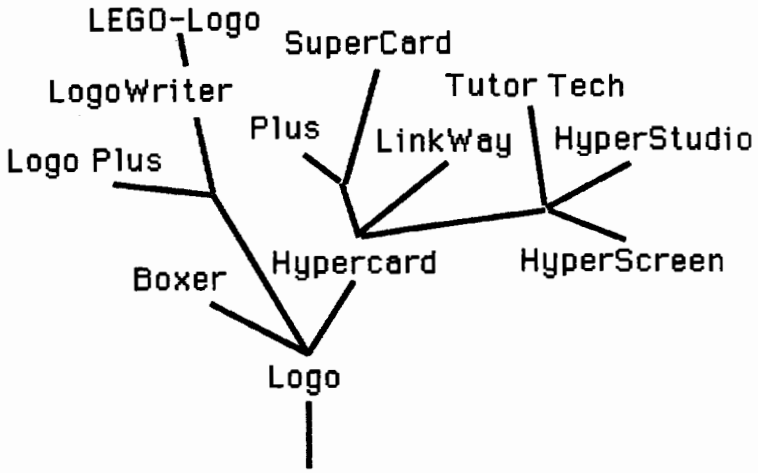
A Proposal: Logo as Exemplar

To signal the extension of a welcome to discussing all types of Logo-like tools, we propose a change in title of *Logo*

Electronic Cul de Sac?



Or Evolving Tools?



Exchange magazine to *Technology and Teaching Tools* (or *Journal of Learner-Based Tools*, or a similar, open-ended title.) When Tom Lough founded the *Logo Exchange*, Logo was the only example of this new instructional use of the microcomputer. Now that it has been joined by many other companions, we think it desirable to welcome them into the fold as well. Is Logo a transitional object leading to new and ever more interesting educational applications ranging from *Boxer* to *HyperCard*, or is it an electronic cul de sac? At one time it was not uncommon for many teachers to employ Logo as the sole tool for all applications from word processing to computer art. Now teachers have a much wider instructional computing tool kit. Rather than ignoring these small mammals (which have been busily eating dinosaur eggs) in the instructional computing community, we would like to extend a welcome.

In the seventies Logo was in its infancy. In the eighties it entered its adolescence. In the nineties it approaches adulthood. Papert used the gears of his childhood as a transitional object, which was used to found a new philosophy of educational computation. For many of us, Logo itself has served as a transitional object that has helped us comprehend a new way of using computers instructionally. Although we have not yet seen any journals devoted to Boxer, there are many journals and newsletters devoted to hypermedia applications. However, a careful examination reveals that the majority of these articles describe methods for production of CAI with these new systems. There should also be a place for exchanging ideas about use of these new systems for construction of the learner-based tools for which they are so admirably suited.

Over the years a number of people who were formerly active appear to have dropped out of the Logo community: Tim Riordon, Steve Tipps, Paula Cochran, etc. (We mention these names because we have coauthored Logo books with each of them, but we are sure that you could mention many others.) Rumors that these individuals have been abducted by the Logo police are completely untrue! In many cases, they are still actively using learner-based tools, but are employing a range of many different tools rather than Logo alone. We would like the *Logo Exchange* to follow their good example.

If there were any truth to the rumors of the Logo police, we would have certainly have heard from them by now as a result of writing this article. We can assure you that we will all be back next fall, writing our columns as we have in years past, although the content of the articles will continue to evolve. Er, one moment ... What's that?!!? They're coming in the front door! Glen, you go out the back. Gina, you take the manuscript. Run, Judi, RUN!!

References

Bull, G.L., Cochran, P.S., & Snell, M.E. (1988). Beyond CAI: Computers, language, and persons with mental retardation. *Topics in Language Disorders*, 8 (4), 55-76.

Bull, G. L., Lough, T. , & Cochran, P. (1987). Logo and exceptional individuals. In J. D. Lindsey (Ed.),

Computers and exceptional individuals (pp. 169-187). Columbus, OH: Merrill.

Cochran, P. S., & Bull, G. L. (1985, November). *Creating a shared context: Using a computer in language therapy.* Paper presented at the annual convention of the American Speech-Language-Hearing Association, Washington, D.C.

diSessa, A.A., & Abelson, H. (1986). Boxer: A reconstructable computational medium. *Communications of the Association for Computing Machinery*, 29 (9), 859-868.

Gould, S.J. (1989). *Wonderful life: The Burgess shale and the nature of history.* New York: W.W. Norton & Company.

Mokros, J.R., & Russell, S.J. (1986). Learner-centered software: A survey of microcomputer use with special needs students. *Journal of Learning Disabilities*, 19 (3), 185-190.

Rosegrant, T. J., & Cooper, W. (1985). *Listen to Learn* [computer program]. Boca Raton: IBM.

Papert, S.P. (1980). *Mindstorms: Children, computers, and powerful ideas.* New York: Basic Books, Inc.

Paper, T.Q. (1980). *Gear Storms: Children, cogs, and powerful Machines.* Nuevo York: Alternate Universe Press.

Rosegrant, T. J., & Cooper, W. (1987). *Talking text writer* [computer program and teacher's manual]. New York: Scholastic.

Russell, S. J. (1986). But what are they learning? The dilemma of using microcomputers in special education. *Learning Disability Quarterly, 9*, 100-104.

About the Authors

Glen Bull has written a column for the *Logo Exchange* since it was founded. Gina Bull was originally a fine arts librarian, but changed careers as a result of exposure to Logo. She subsequently acquired a graduate degree in computer science and a position as a system administrator in a department of computer science. Glen and Gina Bull currently write a column, "Logo and Company," which addresses the emerging array of "Logo-like" tools. Judi Harris is a columnist for the *Logo Exchange*, and writes a Logo column for *The Computing Teacher* as well. Collectively the authors have written more than a hundred chapters, books, articles, and columns about Logo.

***Editor's note:** This manuscript was slipped over the transom and was lying on the floor when we arrived at the office one morning. Despite the fact that we have not yet been able to contact Bull, Bull, & Harris, we would like to reassure you that the conclusion to this article is a prank, resulting from what we can only describe as a distinctly odd (not to say sophomoric) sense of humor. However, if anyone should see any of these individuals, we would very much like for them to contact us.*

Pacman Penpals

by Jandy Bird

While there are many ways of having students work together using *LogoWriter* within a single school or class, it is also possible to work cooperatively at a distance. My third and fourth grade gifted/talented resource room group in Colts Neck, New Jersey, had made contact with a fourth grade class in State College, Pennsylvania. This contact had been established through acquaintances made at the ECCO conference in Cleveland in the spring of 1988. We had begun the exchange, not via telecommunications, but simply by mailing letters and printouts of our *LogoWriter* work to each other. After exchanging news about our respective programs and *LogoWriter* interests, the Pennsylvania group sent us a challenge. The challenge was to improve on a pacman game they had begun, using *LogoWriter*.

It is worth confessing here that using *LogoWriter* to recreate video games was not an idea that thrilled me at first. However, I couldn't have been more mistaken, as the briefest introduction of the idea led to an avalanche of ideas from my students. The challenge spurred them into more advanced areas of *LogoWriter* than they would have experienced, and it enabled them to work as a group, in small groups, and individually, as they wished.

The Pennsylvania group had, of course, first created a pacman shape on the shapes page. In presenting their problem to my group, I duplicated their shape and entered their program on the computer. My group saw the program, and spent a full hour brainstorming improvements. The pacman was shape 2. The program the Pennsylvania group sent was as follows:

```
TO PACMAN
setsh 2
pu
setpos [-131 1]
tell 1
setsh 12
pu
setpos [111 1]
st
tell 0
right 90
repeat 115[st forward 1 wait 3 ht
   forward 1]
end
```

They had used two turtles, turtle 0 for the pacman and turtle 1 one for the circular goal. The pacman begins on the left with the goal on the right, and pacman moves across the page to the goal. The HT in the REPEAT command gives a slight flashing effect as the turtle moves.

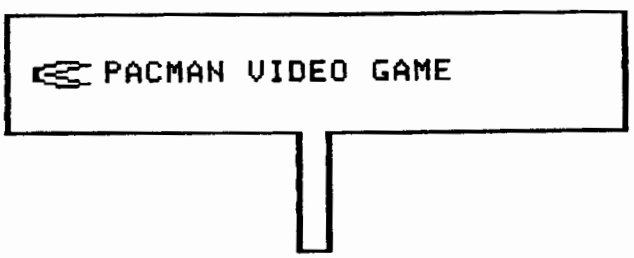
During the brainstorming session, weentered ideas on a large monitor so the whole group could see easily. The first thought was to add color, so we put in a SETC for the pacman and for the circle. The group felt that the pacman should eat the circle instead of disappearing, so the next problem was to make the pacman look as if it were eating, with mouth opening and closing. To do this, we made a second pacman shape with mouth closed, (shape 3). Then, by alternating the shape of turtle 0 between the two shapes, it appeared as if the pacman was eating its way across the screen. Making the wait shorter (from wait 3 to wait 1) also moved pacman more quickly. To make the pacman "eat" the goal or food, SETC turned turtle black. Finally, we used TONE to make pacman "burp" after eating and INSERT to make the words print on the middle of the screen. Our amended version was:

```
TO PACMAN
setsh 2
setc 4
pu
setpos [-131 1]
tell 1
pu
setc 3
setsh 12
setpos [111 1]
st
tell 0
setc 4
rt 90
repeat 115[setsh 2 st forward 1
   wait 1 ht setsh 3 st forward 1]
st
tell 1
setc 0
tone 50 25
repeat 12 [insert char 32]
insert [YUMMY IN MY TUMMY !!!]
wait 30
ct
end
```

After this group session, the students had several avenues they were anxious to pursue, and so they worked in small groups on different problems. One group was to add ghost figures, as in the real video game. This pair of third graders

spent time on the shapes page. Another group was to have different foods of different colors, and to have different sounding burps depending on the food. These fourth grade girls experimented with sound, with IF and with COLORUNDER. Another student wanted to have pacman travel randomly around the screen, and to have something happen if/when he ran into a ghost. He also worked with COLORUNDER and with RANDOM. After some time developing different ideas, a single student was in charge of combining and coordinating the different parts into a single version.

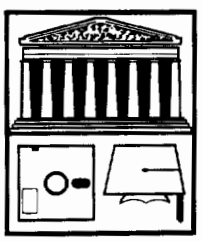
The New Jersey students were not the only ones excited by this process. The Pennsylvania group sent us another version of their game, before they had received our response! They had created an introductory screen, with flashing sign "Pacman Video Game", and they, too, added sound in the form of a musical introduction and music when pacman reached the goal.



With another exchange, the school year was over. The pacman fund of ideas was not, though. In the most advanced version, the introductory screen flashed the sign, the pacman appeared on the left of the screen with a dotted line to the colored food on the right, the pacman "ate" the line across the screen to the colored food and became "energized". When this happened, he jumped around the screen, where there were ghosts stamped at different random spots. If pacman hit a ghost, all the ghosts disappeared and "No more ghoulies!" appeared on the screen. Finally, the Pennsylvania group sent a video about themselves while our group made one about the sequence of pacman, showing the different versions.

All the versions of each group are not shown here for two reasons. The first is that the final versions are quite lengthy. Secondly, if you have a group interested in developing the pacman idea, it would be better for them to develop their own approach. The beginning procedures are shown here just to get you started. The process was fun and exciting and the pacman vehicle was familiar and appealing to the students. The exchange challenge gave the students lots of incentive to learn more *LogoWriter*. They explored how to move randomly, how to make sound, how to check using COLORUNDER, how to integrate text and graphics, and much more. I was reminded of an exercise most writing teachers know well. In this exercise you begin with a sentence such as "The cat ran" and have the group add adjectives, adverbs, and phrases to develop a better sentence. You might end up with "The old scruffy one-eyed cat ran crookedly but swiftly through the hole in the fence chasing the escaped white mouse." The pacman exchange involved a similar elaboration and showed another way in which *LogoWriter* can encourage critical thinking and creative problem solving through a cooperative exchange of ideas.

Jan J. Bird, Ed.D.
Conover Road School
80 Conover Road
Colts Neck, New Jersey 07722
201-946-8590



SIGLogo Business Meeting

Plan now to attend the SIGLogo Business Meeting at NECC '90
June 25-27
Opryland Hotel
Nashville, Tennessee
Meeting: Tuesday, June 26
12:00 - 2:00 pm
Place to be announced at NECC

MathWorlds

Confluence: Logo, Educational Technology, and Mathematics

edited by A. J. (Sandy) Dawson

Three experiences in the past year have led to the writing of this column, the final one for Volume 8 of *Logo Exchange*. Each experience in its own way caused me to reflect on the confluence of mathematics, computer technology, and Logo. In my mind I picture three mountain streams cascading down narrow gorges, flowing across broad plains, joining together (the confluence) to form a river of great breadth, depth and power. In the case of mathematics, computer technology, and Logo, the concern I have is whether or not we will see a confluence which has great breadth, depth, and power, or if we will witness a fragmentation and isolation of these with each creating its own river of thought and action, unconcerned about the paths taken by the other streams.

Let me deal with the experiences in the order in which they happened but reversed from order noted above. First to the Logo experience.

Last May at the Canadian Mathematics Education Study Group (CMESG) meeting in St. Catharines, Ontario, Canada, I was a co-organizer, with Benoît Côté of the Université du Québec Montréal, of a three-day, twelve-hour study session on the topic of "Using computers for investigative work with elementary teachers." The three four-hour sessions were discussions centered around a Logo-based software called "Les deux tortues" created by Benoît Côté, and the set of mathematical activities that this software allows. Although the working group had been planned to focus on using computers in the context of elementary teacher training activities, we ended up spending most of our time looking at the mathematical activities, and discussing the role of computers in mathematical learning. The system presented is the result of an effort to build a bridge between computer activities with the Logo turtle and middle school mathematics curriculum.

The final session of the three days concluded with a general discussion on a question raised by me throughout the workshop: do mathematical activities defined around computers induce a reduced view of mathematics? Much debate throughout the three sessions focused on the supposed neutrality of the computer.

This is where the second experience comes into play. Last fall I taught a graduate level course called Selected Topics in Educational Technology, the focus of which was C. A. Bowers' recent book *The Cultural Dimensions of Educational Computing*.

The question that was of central concern to Bowers, and hence to the students in my course, was whether or not educational technology and in particular educational computing is neutral, in terms of accurately representing, at the level of the computer program, the domains of the real world in which people live. If the answer to this question is that it is not neutral, the critically important question that follows is how the technology alters the learning process (Bowers, p. 24).

In particular, Bowers contends that "computers foster a digital, dichotomous, context-less, ultra rational form of world view, which though extremely productive in many ways, is also at the foundation of many misunderstandings about the world." To paraphrase Gregory Bateson, if we separate an object from its context we are likely to misunderstand it.

Computer educators perpetuate the view that the computer is culturally neutral, that it is simply a 'dumb' machine. But this overlooks the fact that "...the classroom strengthens certain cultural orientations by communicating them to the young and weakens others by not communicating them" (Bowers, p.6).

Much debate throughout the three sessions focused on the supposed neutrality of the computer and of Logo.

Bowers' conclusion, noted below, was hotly debated:

> Thus the machine that the student interacts with cuts out of the communication process (the reduction phenomenon) tacit-heuristic forms of knowledge that underlie commonsense experience. While the technology amplifies the sense of objectivity, it reduces the awareness that the data represent an interpretation influenced by the conceptual categories and perspective of the person who "collected" the data or information. The technology also reduces the recognition that language, and thus the foundations of thought itself, is metaphorical in nature. The binary logic that so strongly amplifies the sense of objective facts and data-based thinking serves, at the same time, to reduce the importance of meaning, ambiguity, and perspective. Finally, the sense of history, as well as the cultural relativism of both the student's and the software writer's interpretative frameworks, is also out of focus. As a symbol-processing technology, the computer selects and

amplifies certain aspects of language...." (Bowers, pp. 33-34)

With these discussions and the thoughts generated by them rolling about in my mind, I embarked on the third experience, one that I am still in as I prepare this column. Since January I have been working with a group of15 secondary school mathematics teachers who are enrolled in a specially designed (for them) masters program at Simon Fraser University. During the current semester they are taking a course with me called the Teaching and Learning of Mathematics. This course has looked at, most closely, the constructivist orientation to the teaching/learning process in mathematics education, and the debates that occurred in recent years regarding the constructivist position.

Discussion in the class around the constructivist position has brought to light a basic debate in mathematics itself, namely, is mathematics created or is it discovered, a topic addressed by such recent writers as Hawking in his *A Brief History of Time*, and Penrose in his *The Emperor's New Mind*. The question here is whether there exists an external mathematical reality independent of all observers that is discoverable, or whether mathematics is purely and simply the creation of the human mind. Penrose, for example, seems to waffle in his answer to this question.

"But does it matter?" you might well ask. I think it does. That is why the confluence issue has become important to me.

It would seem that if one believes

1. in an external reality independent of any human mind,
2. that mathematics is discovered and not created,
3. that computer technology is neutral and value-free, and
4. that language is a conduit through which information simply flows,

then this is a position incompatible with the philosophical orientation of Logo pedagogy. Let me expand a bit on this notion.

The above four points imply, I would argue, that there is a body of knowledge external to all knowers that would form the basis of a mathematics education. That being the case, then the most effective means of having learners come to that knowledge is to present it to them in efficient manner, such as by using a computer technology that is value-free and neutral, and by using instructional methods that establish a clear conduit between the body of knowledge and the student's mind. There would be no need for the learner to explore, to create possible solutions, to guess and test, or to do any of the things that constructivists advocate, because these would only slow down and impede the acquisition of the required mathematical knowledge.

Indeed, it is just such an orientation that has permeated mathematics education since public schools were invented.

But, of course, that is not the pedagogical foundations of Logo, and this may in part explain why so few have adopted a Logo orientation, and why so many reject any changes to the mathematics curriculum, particularly changes that might incorporate a Logo philosophy of teaching and learning.

Logo, in my view, is constructivist in orientation, which means that

1. even if there is a reality external to the human mind, the meaning of that reality still must be created by each individual (this is the weak constructivist position),
2. mathematics is created, not discovered,
3. computer technology is not neutral and value-free, and
4. language is culturally biased (the only meaning that words have, as Alice said, is that which we create for them!)

To adopt this stance necessitates looking at the world and our knowledge of it in very different ways than most of us are accustomed to. This in turn causes a struggle so well documented in the literature on change with respect to why individuals do not change their world view and outlook on life. The confluence of these four factors are the basis of the Logo philosophy, in my view, and only when these streams merge will we achieve a river of thought and action that has breadth, depth, and power in an educational sense.

But if Logo and its many cousins (see Sharon Yoder's editorial in the March 1990 issue of *Logo Exchange*) are to be embraced by the educational community, then those of us who are already of the constructivist perspective will have to assist others in coming to see the world in different ways. In so doing, we will no doubt educate ourselves as others educate themselves. But what educator could ask for more?

References

Bowers, C. A. (1988) *The cultural dimensions of educational computing*. New York: Teachers College Press.

Hawking, Stephen W. (1988) *A brief history of time: From the big bang to black holes*. New York: Bantam Books.

Penrose, Roger (1989) *The emperor's new mind: Concerning computers, minds, and the laws of physics*. Oxford: Oxford University Press.

A. J. (Sandy) Dawson is a member of the Faculty of Education at Simon Fraser University in Vancouver, Canada. He can be reached through Bitnet as userDaws@SFU.BITNET U

ISTE Preconference Activities at NECC '90

ISTE is planning a number of preconference activities at NECC '90 in Nashville Tennessee. Advance programs will be be mailed to ISTE members in the spring.

Workshop for ISTE Organization Affiliates
Saturday, June 23, 1990 9 a.m. - 4 p.m.

Grant Writing and Fund Raising Workshop
Alan November, MassCUE, 1988 Christa McAuliffe Educator
Saturday, June 23, 1990 9 a.m. - 4 p.m.
This workshop, for both the novice and seasoned grant writer, will present an overview of how to develop a grant writing culture within your school district.

Educational Software Designers Forum
Sponsored by ISTE's Private Sector Council
Saturday, June 23, 1990 1:00 p.m. - 4:00 p.m.
Due to the success of the 1989 NECC Educational Software Designer's Forum, it will be held again this year in a preconference expanded form. The forum will focus on current design issues, designing for new technologies, and the needs and expectations of the software design community. Contact coordinator Mary Cron at 213/375-7557.

ISTE Council Meeting
"A New Beginning"
Saturday, June 23, 1990 7:00 p.m. - 9:00 p.m.
Representatives from ISTE Organization Affiliates are invited to attend this important organizational meeting to shape the future for ISTE OA council activities. Active participation will be encouraged as ideas are brainstormed and breakout groups are planned.

SIGLogo Institute
Peeking Into the Crystal Ball:
How Shall We Proceed?
Sunday, June 24, 1990 9:00 a.m. - 4:00 p.m.
Join us for presentations by, and small group discussions with, nationally known Logo leaders including Glen Bull, Judi Harris, Sharon Burrowes Yoder, and Dorothy Fitch. All grade levels apply. Some experience with using or teaching Logo will be assumed on the part of participants. The institute will be aimed at teachers, teacher trainers, instructional computing coordinators and education school professors.

ISTE SIGs to Meet at NECC
ISTE Special Interest Groups will hold their annual meeting at NECC. These SIGs are SIGTE (teacher educators), SIGCC (computer coordinators), SIGCS (secondary computer science educators), SIGLogo (Logo using educators), SIGTel (telecommunications in education), and the new Hyper/MultiMediaSIG (hypermedia in education).

Computer Coordinator's Workshop
Facilitating Change:
Strategies for Implementing Technologies in Schools
Linda Boyse of MECC
Sunday, June 24, 1990 Time: TBA
SIGCC plans and organizes the NECC preconference workshop for computer coordinators.

For information on NECC '90, write or call ISTE, University of Oregon, 1787 Agate St., Eugene, OR 97403-9905; 503/346-4414.

Letter to the Editor

by Brian Harvey

The "Language Chauvinist" editorial attacks exaggerated positions and thereby unfairly slights more moderate, and I think justified, forms of language chauvinism.

Anyone who says that Logo is the only good language ever invented would indeed be absurd. You're right to say that languages must be evaluated in their historical context; the advance from machine language to FORTRAN was probably a bigger step than any single programming language development since then. Still, precisely because FORTRAN was the first high-level language, it is full of design flaws that were unavoidable 30 years ago but well-understood today. There is still a lot of old FORTRAN-based software out there, but it would be foolish to begin a new programming project in FORTRAN.

You're right in saying that Logo enthusiasts shouldn't want programming language development to be frozen at the level of 1980 Logo. But neither should we make the opposite mistake and declare all languages equal, forgetting the ideas that made Logo such a good idea back then. I think the most important of those ideas was the use of procedures as the central control mechanism. Under the name "functional programming" the same general idea is an important part of the methodology of parallel computation. It didn't begin with Logo; Lisp and APL were there first. Most modern languages, including Pascal and C, permit this functional programming style, even though other styles are more popular in those languages. The versions of BASIC available to most personal computer users, however, do not allow the use of procedures at all. More advanced versions of BASIC do include procedures; we can take this as a victory for Logo's approach to programming, while remembering that most BASIC programming (especially in schools) is still done in the bad old versions.

Even BASIC was used for a good reason in its historical context. The reason was that the first personal computers had very limited memories; 8000 bytes was considered enormous. It happens that BASIC is an easy language to implement, and a BASIC interpreter could be written that would fit in the small machines. In those days, it was BASIC or nothing. BASIC made the personal computer revolution possible. We can respect it for that, just as we should respect the historical importance of FORTRAN.

In the present, though, we have plenty of memory in our computers. For us as educators, the central point isn't what language we use, but what ideas we teach. I think we should teach functional programming. This can be done in Logo, or Pascal, or Scheme, or C, or some advanced versions of BASIC. It can't be done in the versions of BASIC commonly used in schools. A school that uses those primitive forms of BASIC is locked out of teaching some great ideas. I don't think I'm being mindlessly chauvinistic when I deplore that restriction.

(I'm aware that I haven't explained what functional programming is all about, or why it's a good thing. That's a long discussion in itself, and I've made the argument elsewhere. For now, just substitute your own favorite Logo idea. If you don't have any Logo ideas that can't be done in some other language, then indeed you might as well use another language!)

The editorial is right that we run the risk of sounding silly if we make the choice of programming language a litmus test for political correctness in educational computing. Ideas are central, not languages. Still, languages embody ideas! If we go to the opposite extreme and accept all languages uncritically, we'll end up by teaching the least common denominator, the ideas that are so old-fashioned that every language can handle them.

(A specific case about which I'm particularly upset is the Advanced Placement curriculum in Computer Science. To object that the test is given in Pascal seems silly. But it's not silly to notice that the ideas in the test include things like strong variable typing and don't include things like list processing. Crucial ideas like recursion that are possible but downplayed in Pascal programming are, therefore, present but downplayed in the AP curriculum. Pascal isn't a bad language in any absolute sense, but the AP curriculum is a bad curriculum, and it's bad because of its historical ties to the style of programming that goes along with Pascal.)

P.S. Do people really object to the new features in LogoWriter and Logo PLUS? I've always liked the new features, especially the page metaphor instead of workspace files. What I don't like is the elimination of old features, specifically CATCH, THROW, DEFINE, TEXT, and property lists. (I'm comparing LogoWriter with earlier LCSI versions; I'm not as familiar with the Terrapin story.) These missing features are compatible with LogoWriter's new ideas; they were left out just to make more room on tiny Apples. Now that we have new versions for larger machines, let's get those features back in! Then I'll be happy to use LogoWriter.

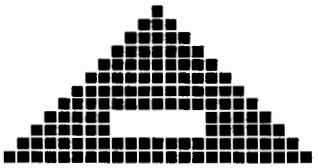
Brian Harvey, Computer Science Division
University of California, Berkeley

Logo and Company

Turtle Graphics for HyperCard

by Glen L. Bull and Gina L. Bull

In last month's column we described how to build a turtle in *HyperCard* and provided procedures for FORWARD, BACK, LEFT, and RIGHT. These commands can be used to move the turtle around the *HyperCard* screen in traditional Logo fashion, but PENUP and PENDOWN commands must be added to allow the turtle to draw. In this column, the last of the year, we will provide turtle graphics for the turtle created last month.



Adding Pen Commands

PENUP and PENDOWN commands provide a way to control the turtle's pen. A global variable called **penState** is used to record the state of the pen. When the **penState** is "down," the turtle will draw. When the **penState** is "up", the turtle will move across the screen without drawing. Add the following procedures to the turtle commands already entered in the script of the Stack Editor last month.

```
on PENUP
  global penState
  put "up" into penState
end PENUP

on PENDOWN
  global penState
  put "down" into penState
end PENDOWN
```

Because these commands are frequently used, we will create abbreviations for them. You may want to use the same approach to create abbreviations of FD for FORWARD and BK for BACK.

```
on PU
  PENUP
end PU

on PD
  PENDOWN
end PD
```

In last month's column we demonstrated how the turtle heading can be initialized when the stack is first opened. The **penState** should also be added to this initialization procedure. We have chosen to place the pen in the "up" position when the turtle first appears, but you can follow your own preference. Please note that, as always, the "¬" symbol indicates that a line is continued in *HyperCard* and is obtained by holding down the option key as the Return key is pressed.

```
on openStack
  global heading,penstate
  put 0 into heading
  put "up" into penState
  set icon of card button ¬
    turtle to "T" & heading
end openStack
```

Drawing a Line

Drawing in *HyperCard* is accomplished by dragging the paint brush from one set of Cartesian coordinates to another set of coordinates. First choose the paint brush by typing the following in the Message Box, and pressing the return key:

```
choose brush tool
```

This command should result in selection of the paint brush in the tools palette.
After the brush tool is selected, draw a line by entering the following in the Message Box of *HyperCard*:

```
drag from 50,50 to 100,200
```

This command will draw a line across the *HyperCard* screen between the two coordinates specified. The drag command will be used to create a **drawLine** procedure which can be used to draw a line from one position to another if the **penState** is "down".

```
on drawLine
  global penstate,pos,newPos
  if penstate is down then
  if the tool is not "brush tool" ¬
    then choose brush tool
    drag from pos to newPos
  end if
end drawLine
```

In this instance, we would like to draw a line from the old turtle position to the new turtle position. This can be accomplished by adding the drawLine procedure to the end of the script for FORWARD that was developed last month.

```
on FORWARD length
  global heading,pos,newPos
  put the loc of card button ¬
    turtle into pos
  put item 1 of pos + length * ¬
    cos ((90 - heading) * PI / 180)¬
    into x
  put item 2 of pos - length * ¬
    sin ((90 - heading) * PI / ¬
      180) into y
  put round (x) & "," & round (y) ¬
    into newPos
  set the loc of card button ¬
    turtle to newPos
  drawLine
end FORWARD
```

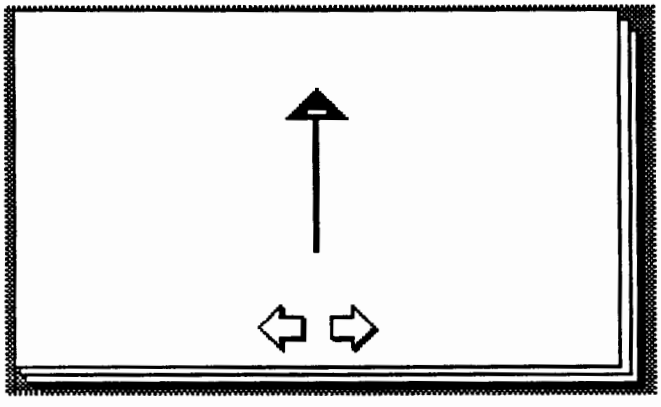
To use new drawing capabilities of the turtle, first put the pen down by entering the following in the Message Box and then pressing return:

```
PENDOWN
```

Then type the following:

```
FORWARD 100
```

Did the turtle draw a line 100 turtle steps long?



Emulating the Repeat Command

In Logo it is possible to enter more than one command on the same line. In *HyperCard* this will generate an error message. For example, try typing something like "FORWARD 50 RIGHT 90" on the same line in the Message Box. The chances are that some computer jargon like "Comma Expected Between Arguments" will be generated when you press Return.

At times it may be convenient to enter more than one command at a time in the Message Box. Logo has a RUN command, not often used, which allows the user to run a list of commands. The command looks like this in Logo:

```
RUN [FORWARD 50 RIGHT 90]
```

Actually "RUN" is the same as "REPEAT 1" in Logo, because the two commands have the same effect.

```
REPEAT 1 [FORWARD 50 RIGHT 90]
```

Creating a RUN command in *HyperCard* will allow us to enter more than one command at a time in the Message Box. The syntax for the *HyperCard* version of RUN will be slightly different than the Logo version. The list of commands are enclosed in quotation marks rather than Logo brackets, and it will be necessary to place a comma between each command. The *HyperCard* version looks like this:

```
RUN "FORWARD 50, RIGHT 90"
```

The REPEAT command is used more often than RUN in Logo. The word "REPEAT" is already used for another purpose in the HyperTalk scripting language, so REPEAT can not be used on the command line in the same way that it is in Logo. However, the RUN procedure that we are about to create can be made to serve double duty by adding a provision for specification of the number of times the commands will be run. The form of the command might look like this:

```
RUN 4, "FORWARD 50, RIGHT 90"
```

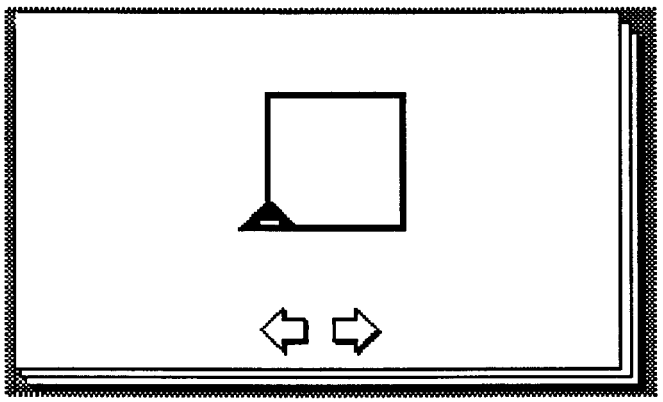
There must be a comma between the number and the list of commands. In Logo, inputs to procedures are separated by spaces, but in *HyperCard* they are separated by commas. In this instance there are two inputs: (1) the number of times the commands should be run, and (2) the list of of commands to be run. If the number of times to run the list of commands is omitted, it is assumed that the default should be one time. The procedure to create a RUN command in *HyperCard* looks like this:

```
on RUN times,command
  if command is empty then
    put times into command
    put 1 into times
  end if
  put number of items in command
  into max
  repeat times
    repeat with n = 1 to max
      send item n of command
    end repeat
  end repeat
end RUN
```

After you have entered the script for RUN into the Stack editor of *HyperCard*, try typing the following in the Message Box:

```
RUN 4, "FORWARD 50, RIGHT 90"
```

HyperCard should produce the familiar Logo square.



There is one significant limitation to the *HyperCard* version of RUN. Since it uses commas as the delimiters that tell it where one command ends and the next one begins, commands with two inputs such as "RECTANGLE 51,40" cannot be included in the list of commands to be run.

Drawing Embellishments

There are several commands that make turtle graphics more useful. One of the more important is the CLEAR command, which clears the graphics screen. The CLEAR command can be written in the following way in *HyperCard*.

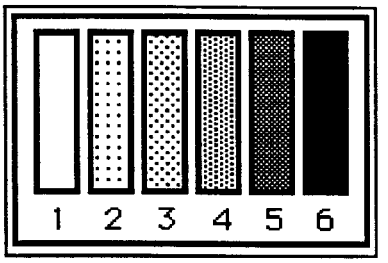
```
on CLEAR
  put the tool into oldTool
  choose select tool
  doMenu "Select All"
  doMenu "Clear Picture"
  choose oldTool
end CLEAR
```

Some features can be included in the *HyperCard* version of turtle graphics that may not be available in some other versions of Logo. For example, it is possible to set the pen width to any of six different sizes.

This is a procedure that will alter the size of the line drawn by the *HyperCard* turtle.

```
on penWidth size
  if size < 2 then set brush to 28
  if size = 2 then set brush to 32
  if size = 3 then set brush to 8
  if size = 4 then set brush to 7
  if size = 5 then set brush to 6
  if size > 5 then set brush to 5
  set lineSize to size
end penWidth
```

The current version of *HyperCard* does not support color, but several different shades of gray can be used for the pen color.



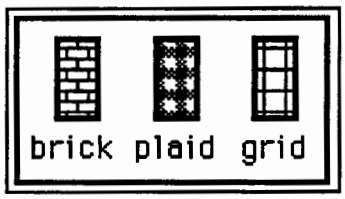
When the pen shade is set to "white," it will erase any lines that are drawn in black.

```
PENSHADE "white"
```

The commands "PENSHADE white" and "PENSHADE 1" have the same effect in this instance. The script for the PENSHADE command can be written in this way.

```
on penShade color
  if color = "white" then set pattern
  to 1
  if color < 2 then set pattern to 1
  if color = 2 then set pattern to 2
  if color = 3 then set pattern to 3
  if color = 4 then set pattern to 13
  if color = 5 then set pattern to 22
  if color > 5 then set pattern to 12
  if color = "black" then set ¬
    pattern to 12
  if color = "brick" then set ¬
    pattern to 36
  if color = "plaid" then set ¬
    pattern to 40
  if color = "grid" then set pattern ¬
    to 34
end penShade
```

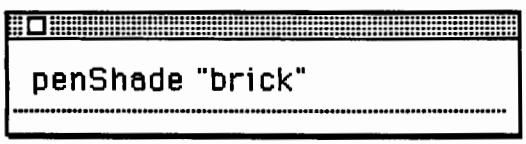
In addition to shades of gray, different patterns such as brick and plaid can also be selected through the **penShade** command.



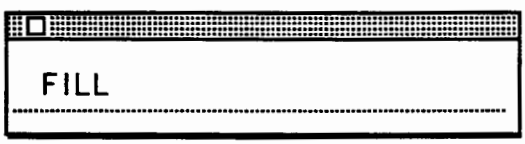
The pen shade not only affects the pattern of the line that the turtle draws; it also affects the pattern used by the FILL command. The FILL command is written in the following way:

```
on FILL
  put the tool into oldTool
  choose bucket tool
  click at the loc of card button
  turtle
  choose oldTool
end FILL
```

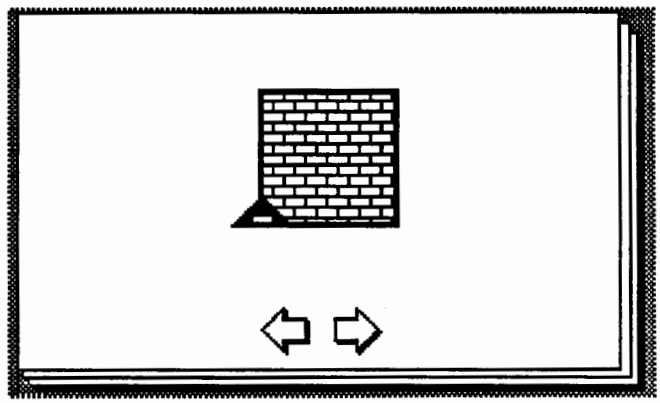
After you enter the FILL procedure into the script of the Stack editor, try the following. First, draw a square using the turtle graphics commands that you have added to *HyperCard*. Pick the pen up and put the turtle in the center of the square. Then set the pen shade to the brick pattern by typing the following in the Message Box and press return:



After you have set the pen shade to the brick pattern, type "FILL" in the Message Box and press return.



The square should be filled with a brick pattern. Don't forget to set the pen shade back to black (unless you want the turtle to draw in a brick pattern).



The PENSHADE script is based on the menu of patterns in *HyperCard*, so if you want, you can experiment by adding your own favorite patterns to the PENSHADE command.

Creating Procedures with Your Turtle Graphics

Procedures are the heart of Logo, and of *HyperCard* as well. They make it possible to break larger problems into "mind-sized" chunks. Now that you have added the turtle graphics commands to the script of the Stack editor, you are ready to begin using them to write procedures. You don't want your students to intermingle their procedures with the basic procedures that provide the new turtle graphics capability. Fortunately, each card in *HyperCard* also has a script editor.

The basic structure of LogoWriter is composed of pages. Each page in LogoWriter has a procedure editor in which procedures for that page can be edited. In the same way, each card in *HyperCard* also has an editor. If you place the turtle graphics procedures you write on the Card editor, they will not become intermingled with the basic turtle graphics commands that you placed in the Stack editor.

To access the Card editor, go to the *Objects* menu, but this time select the "Card Info" option rather than "Stack Info" as you have previously. (*Note*: to highlight the critical features, we have edited some of the menus so that only the most salient aspects are shown.)

Once you are in the "Card Info" dialog box, select the "Script" button, which will take you to the script editor for the card. When you are in the script editor for the card, enter the following procedure, which will be familiar to most Logo users.

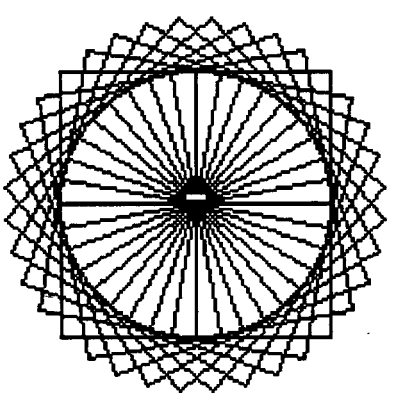
```
on square size
  repeat 4
    fd size
    rt 90
  end repeat
end square
```

There are a couple of differences between the *HyperCard* version of SQUARE and the Logo version. In a *HyperCard* script, the commands to be repeated must be placed between a "REPEAT" statement on the first line and an "END REPEAT" statement on the last line. Further, only one command is permitted on each line.

Once these minor differences are noted, the overall format will seem very familiar to most Logo users. You will be able to run your favorite turtle graphics procedures just as you can in Logo. For example, here is a procedure for **spinSquare.**

```
on spinSquare
  repeat 36
    square 50
    right 10
  end repeat
end spinSquare
```

After you enter this procedure in the Card editor, you will obtain the following result when you type "**spinSquare**" in the *HyperCard* Message Box:



Summary

In the previous three months, we showed you how to do the following:

- Create a *HyperCard* stack
- Write HyperTalk procedures
- Build a Logo turtle in *HyperCard*
- Create FORWARD, BACK, LEFT, and RIGHT commands

In this column we added turtle graphics to the basic turtle.

We have written a column for the *Logo Exchange* each issue since the founding of the journal, for a total of 71 columns over eight years. (The total would have been 72 columns, but we skipped one month when our son, Stephen, was born.) There have been a number of amazing changes in technology over the past decade. The first issues of *Logo Exchange* were hammered out on a dot matrix printer in Tom Lough's basement. Now they are generated on PostScript laser printers.

In a recent editorial, Sharon Yoder, the current *Logo Exchange* editor, noted the trend toward use of Logo as an application rather than as a programming language. She asked,

> "Should we be disturbed that many people are treating Logo as an application? Does it affect the fundamental philosophy behind Logo and Logo-like learning? I think not."

We agree. When microcomputers were first introduced to the schools, Logo was one of the few interesting applications available. Now there are a multitude of interesting applications, which range from hypertext and hypermedia to projects such as the National Geographic Kid's Network and MIX. The International Society for Technology in Education (it seems strange to no longer call it the "International Council for Computers in Education") not only has SIGLogo, but SIGHyper, SIGTelecommunications, and many others as well.

In the 1960s some thought that the most effective use of computers would be found in their role as teaching machines through Computer Assisted Instruction (CAI). In the 1990's it is clear that their role as a teaching tool is one of the most powerful instructional uses. SIGLogo, SIGTel, and SIGHyper are all part of a larger interest group of special interest, *SIGTools*. When a student in Charlottesville, Virginia uses a modem to send a Logo procedure to a class in Alaska or

Moscow, is Logo or telecommunications the most important element? The question is meaningless, because the tool-using attitude is the foundation that underlies the most important lesson that they are learning. Logo was the early forerunner of the tool-using concept, but it has now been joined by many other applications, with more arriving every year.

Possibly a title such as *Technology and Teaching Tools* rather than *Logo Exchange* would be more reflective of the broad array of applications that will become available in the coming decade. In any event, we will continue to discuss links between Logo and Logo-like tools in the next year through the vehicle of "Logo and Company." Have a good summer!

Glen Bull is a member of the instructional technology faculty in the Curry School of Education at the University of Virginia. Gina Bull is a programmer analyst for the University of Virginia Department of Computer Science. By day she works in a Unix environment; by night, in a Logo environment.

Glen and Gina Bull
Curry School of Education
Ruffner Hall
University of Virginia
Charlottesville, VA 22903

BITNET addresses:
Glen: GBULL@ VIRGINIA.bitnet
Gina: GINA@VIRGINIA.bitnet

ef•fec•tive\i-'fek-tiv\adj (14c)

1 a : producing a decided, decisive, or desired effect b : IMPRESSIVE, STRIKING
2 : ready for service or action

Computer-Integrated Instruction: Effective Inservice

Dave Moursund's comprehensive series on inservice training for computer using educators has grown. ***Effective Inservice for Secondary School Mathematics Teachers*** and ***Elementary School Teachers*** are joined by texts for ***Secondary School Science Teachers*** and ***Secondary School Social Studies Teachers.***

Based on a National Science Foundation project, these volumes bring you the latest research on effective training. Each work contains specific activities and background readings that enable you to hold inservices that result in positive, durable change at the classroom level.

If you design or run computer-oriented inservices, ***Effective Inservice for Integrating Computer-As-Tool into the Curriculum*** will help you develop a sound program through theory and practice. Sample forms for needs assessment and formative and summative evaluations are included.

Each of the five volumes comes in a three ring binder that includes both hard copy and a Macintosh disk of the printed materials. Individual Math, Science, Social Studies, and Elementary School volumes are $40 each ($3.95 shipping per copy). Computer-As-Tool is $25 ($3.95 shipping per copy). The complete set of five is available for the discounted price of $150 ($7.50 shipping per set).

ISTE, University of Oregon, 1787 Agate St., Eugene, OR 97403-9905; ph. 503/346-4414.

Logo: Search and Research

Programming with Style
by Douglas H. Clements

In the previous column we saw that successful high-school teachers explicitly encouraged their students to master each "link in the chain," from learning language features to learning to design programs to solving problems. Even this may not be sufficient, however, especially in helping students become "expert" programmers.

Programming style: A way of thinking

How does one impart to students beneficial ways *of thinking* about code that will allow them to write truly good programs (Joni & Soloway, 1986)? That is, programs that are:

- correct,
- user-friendly,
- concise, efficient,
- debuggable, extensible, and maintainable.

Which of these goals should we emphasize? How should we achieve them? A common way is to emphasize efficiency. However, Joni and Soloway point out real problems with this criterion. First, "efficiency" is rarely meaningful in the context of the relatively small programs students write. Second, efficiency is not really understood until late in the development of programming ability. Third, the message we are sending is "efficient programs are best." But even in sophisticated programs, efficiency must be balanced with other characteristics, such as being debuggable.

Therefore, Joni and Soloway take as their criterion the development of *readable* programs. Over 90% of computer programs they examined—and these were written by college students—violated the principles of readability. So there is a deficit. More important, focus on readability helps develop good programming practices and is a familiar activity to students.

Program readability

How does one define readability? The authors' answer is based on theory and on research with both novice and expert programmers. Experts have two types of knowledge of programming: Programming plans, which we called "templates" previously, and rules of programming discourse. These rules specify the conventions in programming, for example, meaningful names for variables. They set up expectation in the minds of the programmers about what the program does. So they are analogous to discourse rules in everyday conversation.

Joni and Soloway's research indicates that experts' programs are built of templates modified to meet the needs of a particular problem. The modification and combination of these templates are guided by the rules of programming discourse. Programmers find it significantly more difficult to comprehend programs in which these rules are not followed. For example, in a program that finds the minimum number among its inputs, changing only the name of the variable, and storing the largest value from MIN to MAX results in a 100% increase in reading time.

So, programmers expect other programmers to follow the rules. Readable programs have clear and evident plan structures and present no surprises. Joni and Soloway examined the programs of 57 college students. These students wrote a program to solve a tax problem. From this, they formulated a basic list of rules of two types. *Maxims* are generalized statements of good programming style. *Rules of programming discourse* are specific statements regarding each maxim. While Joni and Soloway's students used Pascal, research with Logo indicates that the same or similar rules apply using Logo (Lee & Lehrer, 1988).

Information roles and variables

A basic programming process is to choose and name variables. This leads to:

> *Maxim 1:* Think carefully about the roles information will play. Use one and only one variable for each role. Give the variable a name that reflects this information role.

One way students violated this rule solving the tax problem was to have one variable (STATUS) hold information both about marital status and about whether to end the program or to continue. This lead to:

> *Discourse rule:* Avoid Double-Duty Variables. Do not have one of your variables serving two different information roles. Instead, use two separate variables, and name the variables in a way that makes their roles clear (p. 107).

Other students had several variables at different points in the program that held the same information. That is, both variables played the same information role. So:

> *Discourse rule:* Avoid Half-Duty Variables. Do not have more than one variable for any information role in each program module. Use one variable, and have its name reflect its information role (p. 110).

Good construction

Other programs are poorly constructed. Students violated:

> *Maxim 2:*
> 1. "Be sure that all portions of your code are *useful to the task at hand.*
> 2. Choose language constructs that *help to make your program actions clear"* (p. 110).

An analogy is that in writing travel directions, one would want (1) all instructions to be useful in getting the person to the goal, (2) expressed in such a way that the actions they had to take were clear. Program readers look at code and first ask, why is this here? What purpose does it serve? Certain kinds of code violates readers' expectations.

A common error is initializing a variable more than once. Often students initialize a variable by directly assigning it a value such as 0 and then re-initialize it through a different means. For example, the second assignment might result from some computation or through input from the user. Some students set the variable SALARY to 0, then asked the user to type the salary, re-initializing SALARY with that value. The first initialization leads a reader to expect that SALARY might be an accumulation variable, but it is not. When the reader sees SALARY re-initialized, an expectation failure occurs, and rereading is often necessary. So:

> *Discourse rule:* Initialize each variable precisely once.

There were several instances of choosing language constructs that obstructed readability. Some of these instances were more specific to the Pascal programming language. One generally applicable rule was placing code that was not affected by an IF test in both the TRUE and FALSE branches. One student's work, translated to Logo, was:

```
IF :salary < 10000 THEN MAKE "tax
  :salary * :rate1 MAKE "net :salary
  - :tax ELSE MAKE "tax :salary *
  :rate2 MAKE "net :salary - :tax
```

Note that the command MAKE "net :salary - :tax is repeated, even though it is invariant with respect to the IF instruction. This leads to a:

> *Discourse rule:* DO NOT put code that is invariant with respect to the outcome of a conditional IF text inside the scope of the IF construct. Instead, put this code immediately preceding or following the IF construct (p. 120).

Merged goals

Finally, students sometimes wrote segments of programs that achieved more than one goal. In contrast, the authors suggest:

> *Maxim 3: Minimize* having merged goals in your program. Try to use a separate plan to realize each program goal (p. 120).

Some students, for example, combined three goals of the tax problem: (a) getting and validating data from the user, (b) calculating, and (c) printing the results. This led to extensive reader confusion. This could have been avoided by:

> *Discourse rule:* Don't merge validation of input with any other program goal.

In sum, emphasizing the readability of programs is meaningful and accessible to students. Teachers might make maxims and discourse rules explicit through discussions. Students should revise programs for readability, not just edit them so they "work." Such activities will probably lead to more readable programs. More important, they help students develop deeper knowledge of programming. The specific rules used are not as important as the approach: Discussing and developing programming style and understanding.

References

Joni, S. A., & Soloway, E. (1986). But my program runs! Discourse rules for novice programmers. *Journal of Educational Computing Research, 2*, 95-125.

Lee, O., & Lehrer, R. (1988). Conjectures concerning the origins of misconceptions in Logo. *Journal of Educational Computing Research, 4*, 87-105.

Douglas H. Clements
State University of New York at Buffalo
Department of Learning and Instruction
593 Baldy Hall
Buffalo, NY 14260.
CIS: 76136,2027 BITNET: INSDHC@UBVMS

Global Logo Comments

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

This month's Global Comments come to us from our Latin American and Asian correspondents. Marie Tada gives us a recent overview on the use of Logo in Japan while José Valente highlights some Logo activities in Uruguay and Brazil.

Japan
by Marie Tada

For me, trips to the States in the summer include courses or conferences that help me develop my Logo learning and contacts with Logo enterprises such as LCSI, Terrapin, andLego TC Logo to see what new ideas and products are available. The rest of the year I rely on the *Logo Exchange*! I have wondered, however, about the kind of support that Logo users in Japan could find, and have been delighted to see that a number of sources are available. In this article I will talk about Logo Japan and its efforts to promote Logo learning in the Japanese schools.

Three years ago I made my first contacts with Logo Japan in order to get a site license for LogoWriter. At that time the Tokyo office was fairly small, and I have been amazed to see the growth that has taken place in the past couple of years and the inroads that the Japanese version of LogoWriter has been making in Japan in the process.

Logo Japan is a private company that has been working in conjunction with LCSI since 1987. Seymour Papert of MIT is a scientific advisor and comes to Japan regularly to promote Logo and exchange ideas.

Mr. Tsuru, the president of Logo Japan, sees that the schools of the coming decades will witness widespread growth in the use of computers in education. As the Education Ministry of Japan is revising guidelines to include computers as a central part of the educational experience of children, Mr. Tsuru predicts that Logo Japan will play a large role in determining how and what will be learned. Logo Japan has taken the Logo language and LogoWriter to Japan and worked on development of materials and support for users of the Japanese version. Now that LogoWriter is being used more extensively in Japanese schools, it is Logo Japan's goal to work on development of the software's capacity and promote exchange of data and standardization of software to facilitate file transfers among schools.

The philosophy of Logo Japan is to use the resources of the technological age to foster a creative learning environment. The Japanese version of LogoWriter is intended to be a kernel from which the new technology of the age can be developed while children engage in educationally sound learning projects. Logo Japan is in the process of developing quality educational materials and reference manuals, and is also facilitating the development of leaders who will help in introducing LogoWriter to teachers in the schools.

Logo Japan recently came to our computer club meeting to write up our activities in their *Logo World* publication. We received many issues of this Japanese language Logo journal and I was delighted with its high level of articles and activities. The magazine attempts to provide helpful information, programs, information about computer education, overseas news, and reader exchanges. We have explored some fractal activities that were suggested with successful results. At the end of our meeting I felt that this must surely be what Dr. Papert had in mind when he stated that Logo was a vehicle for "playing" with mathematics as well as a medium for sharing our thinking on a world-wide scale.

Aside from this journal, there is also a telecommunications exchange called Logo-Net. At present the network is used primarily by teachers and other enthusiasts to exchange problems, information, and suggestions. It is hoped that this network will will become an "electronic conference room" of Logo exchanges among the schools in the near future. The

host computer is in Tokyo with 64 access points throughout Japan.

Logo Japan has provided me with a list of universities and other educational institutes that have purchased LogoWriter. The list is long and impressive, suggesting that the level of interest in Logo as a part of the educational scene is definitely on the rise. I have had the opportunity to see LogoWriter in action in the Japanese schools and will write more on this in future Global News columns.

Uruguay
from Jose Valente

The Instituto Yavne is dedicated to teaching pre-university students (16-18 years old) who have an interest in attending engineering schools and students who are going to be math teachers at secondary and high schools.

The objective of this research is to show that Logo is not only for kids, but it can be a useful tool to prepare students for more advanced degrees. Logo is used not only to teach computer programming, but as a didactic tool to help students learn how to think, to learn, to solve problems, to debug, and to train people how to teach.

These students are developing the following activities:

Study of real functions

For a given function, the students can study domain, limits, derivative, variations, maximum and minimum, inflection points, etc. This study is done through a project the students develop using Logo. One project, for example, is to plot the function. In doing this they have to study the function's behavior, and adjust scales according to the range of the abscissa, maximum points, and the screen size. Another project could be to identify problems in real life, describe them through specific functions, and study them.

Use of geometric transformations

This activity has two parts. First, the student has to apply several transformations on a particular point (x,y) on the screen. For example, send the turtle to its homotetic point with respect to the (0,0) coordinate and a given ratio k (that is x' = kx and y' = ky). The second activity is to define a procedure that draws a particular figure and to apply to this figure several transformations. With this activity the student can try, investigate, and analyze different properties of the figure and of the transformation.

Discover the number π

Using Logo, the students calculate the area of polygons inscribed in a circumference as function of the circumference radius and the number of sides of the polygon. If the student repeats this procedure for different polygons, he or she obtains pairs of numbers that converge monotonically to π, if the circumference radius is 1. By doing this the students can study the properties of polygons, trigonometry, convergent series, limits, etc. Also, they can come to understand much better the Logo procedure that draws a circumference.

Develop educational software (tutorials)

As a project, the student has to develop a procedure that can teach a particular subject through the computer. For example, a 16-year-old developed a program to teach the Pythagorean Theorem. In order to calculate the angle the turtle has to turn, the student had to implement the arctangent function. To do this he used Taylor-MacLaurin series and had to deal with errors and the calculation of errors.

Exercising problem solving

This activity showed that in Logo, physics and mathematics can help each other in solving problems. One student implemented a microworld in which one can play and study the behavior of light according to a law implemented through a student-defined procedure. One activity is to measure the angles of incidence and refraction in several circumstances until one can discover the law that is guiding the behavior of light.

This work has shown that through the use of Logo it is possible to awaken in the students the adventurous spirit of investigation while they experience several different learning situations. The most important thing is that they have to correct their own mistakes so they can solve their own problems. These activities will, certainly, be very important in helping them to become professionals of tomorrow.

For more information please contact:

Alicia Villar
Avda. Rivera 5760
Montevideo 11400 URUGUAY

Their inquiring minds want to know.

And you'll help them discover how to find the answers. By using ***Teaching Thinking Skills with Databases*** in your classroom, you'll challenge students to develop a mind of their own.

Designed for Grades 4-8, this step-by-step guide by Jim Watson gives you the opportunity to impact your students' cognitive development through the use of databases.

Teaching Thinking Skills with Databases contains 14 data files and 46 worksheet and transparency masters. Teach with databases in any subject using *AppleWorks®* or *FrEdBase*.

Use ***Teaching Thinking Skills with Databases***... because they want to know.

School site license **$30.00** +$3.95 shipping

ISTE, University of Oregon, 1787 Agate St., Eugene, OR 97403-9905; ph. 503/346-4414.

We've polished up a proven favorite!

AppleWorks for Educators by Linda Rathje really shines. The new edition has been expanded to include sections for:

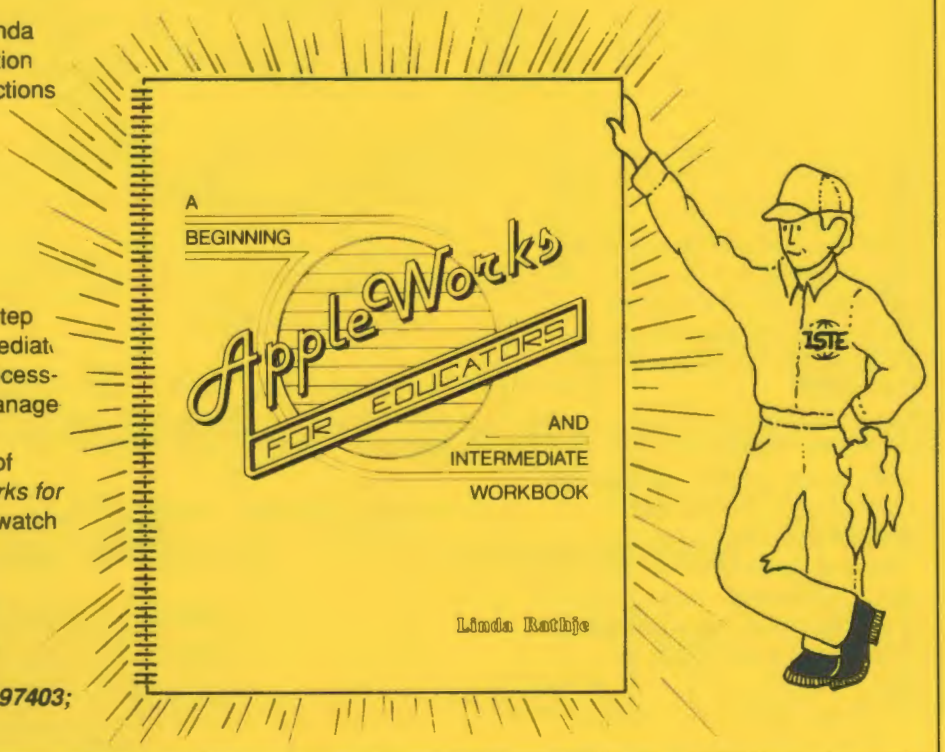
- mail merge
- integration activities
- a glossary, and
- software applications.

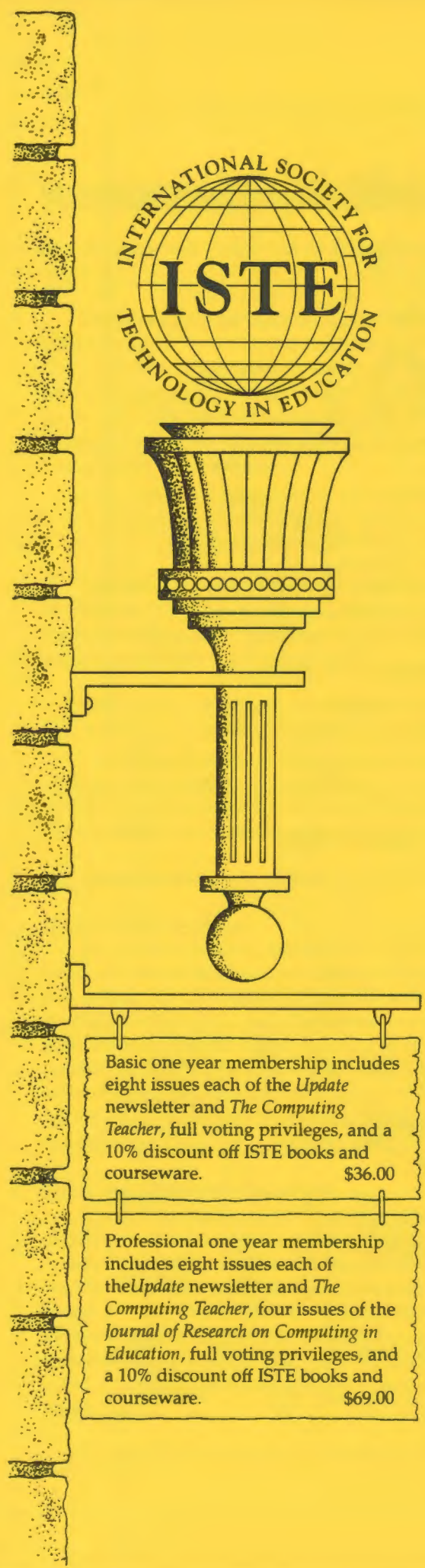
Each section provides step-by-step instructions. Beginning and intermediat AppleWorks® users learn word processing, database and spreadsheet manage ment, and printer options.

Your copy includes a data disk of working examples. Add *AppleWorks for Educators* to your classroom and watch your students shine.

Single copy price: **$22.95**
please add $3.95 shipping

ISTE, 1787 Agate St., Eugene, OR 97403; ph. 503/346-4414.





The ***International Society for Technology in Education*** touches all corners of the world. As the largest international non-profit professional organization serving computer using educators, we are dedicated to the improvement of education through the use and integration of technology.

Drawing from the resources of committed professionals worldwide, ISTE provides information that is always up-to-date, compelling, and relevant to your educational responsibilities.

Periodicals, books and courseware, *Special Interest Groups, Independent Study* courses, professional committees, and the Private Sector Council all strive to help enhance the quality of information you receive.

Rely on ISTE support:

- *The Computing Teacher* draws on active and creative K-12 educators to provide feature articles and carefully selected columns.
- The *Update* newsletter reaches members with information on the activities of ISTE and its affiliates.
- The *Journal of Research on Computing in Education* comes out with articles on original research project descriptions and evaluations, the state of the art, and theoretical essays that define and extend the field of educational computing.
- Books and courseware enhance teaching materials for K-12 and higher education.
- Professional Committees develop and monitor policy statements on software use, ethics, preview centers, and legislative action.
- The Private Sector Council promotes cooperation between educational technology professionals, manufacturers, publishers, and other private sector organizations.

It's a big world, but with the joint efforts of educators like yourself, ISTE brings it closer. Be a part of the international sharing of educational ideas and technology. Join ISTE.

Join today, and discover how ISTE puts you in touch with the world.

Basic one year membership includes eight issues each of the *Update* newsletter and *The Computing Teacher*, full voting privileges, and a 10% discount off ISTE books and courseware. $36.00

Professional one year membership includes eight issues each of the*Update* newsletter and *The Computing Teacher*, four issues of the *Journal of Research on Computing in Education*, full voting privileges, and a 10% discount off ISTE books and courseware. $69.00

ISTE, University of Oregon,
1787 Agate St., Eugene, OR 97403-9905.
ph. 503/346-4414