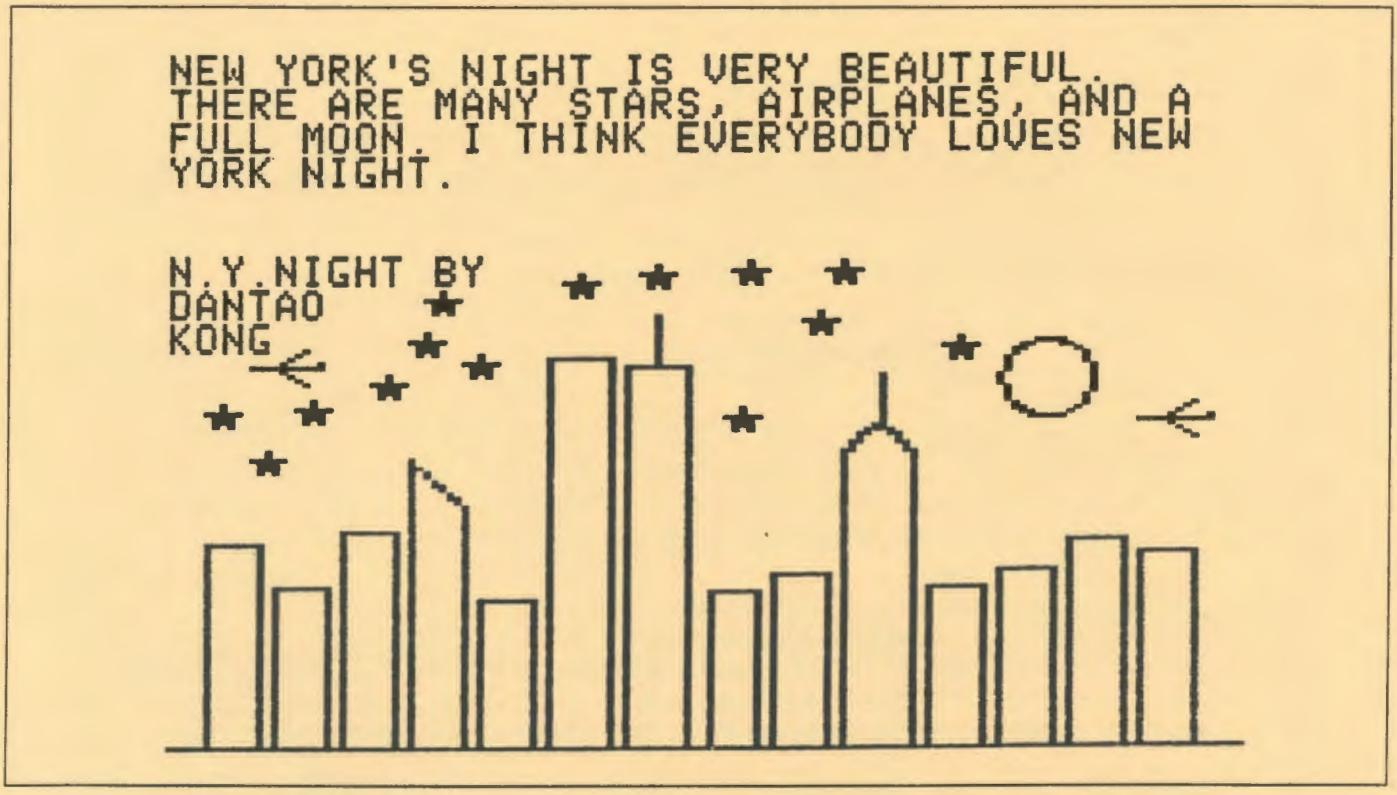
Journal of the ISTE Special Interest Group for Logo-Using Educators

LOGO EXCHANGE

November 1989 Volume 8 Number 3



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

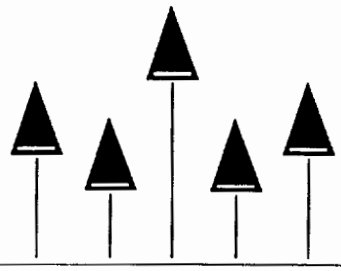
School Districts may acquire a license for the use of the ON LOGO package of 8 half-hour videotapes and 240 pages of supporting print for $599.00. For a one-time fee of $1295.00, the package may be obtained with both tape and print duplicating rights, enabling districts to build libraries at multiple 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 503/686-4414.



LOGO EXCHANGE

Volume 8 Number 3 Journal of the ISTE Special Interest Group for Logo-Using Educators November 1989

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.

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

Contents

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

Logo: Programming Language or Application

I recently returned from the New Zealand Computer Educators Conference held in New Plymouth, New Zealand. Aside from meeting some delightful people and enjoying some wonderful scenery, I came away from the conference with a number of new ideas. (Isn't that the reason we attend those conferences and workshops?) Most people I talked with at the conference who were interested in Logo knew of *LogoWriter* but weren't using it and hadn't yet heard about *Logo PLUS*. At least one person commented to me that *LogoWriter* and *Logo PLUS* (which I demonstrated during a conference session) seemed more like an application than a programming language.

That comment struck a resonant chord with me. I've been doing a good bit of thinking lately about programming languages and their role in education today. Only 10 years ago, we thought that every teacher and every student should learn to program – in BASIC, since that was almost the only thing we could do on our 4K or 16K micros. A couple of years later, as Logo became available on machines used in schools, many of us chose Logo over BASIC as the programming language of choice. We were on the crest of a wave. Students were enrolling in programming classes in droves. Everyone was sure that programming was the way to go: students would learn logical thinking, mathematics and become good problem solvers — just from learning to program. Those who knew how to program would leave high school to find lucrative jobs in business and industry.

Ah, how times do change. Today, the trend towards learning to program has reversed itself. No longer is programming "in." Research has not born out the claims that programming in and of itself – in Logo or any other language – leads to better problem solving skills or greater understanding of mathematics. Today's students are learning to use word processors, databases, spreadsheets, and graphics packages.

From the time Logo arrived on the educational scene, most of us have thought of Logo as a programming language. Certainly in early versions you couldn't do much that was interesting without writing procedures. And if you *did* do something interesting in the immediate mode, then you often couldn't save it . In many cases you couldn't even print it. Working with Logo meant working with procedures, learning syntax and grammar, understanding control and data structures and learning to debug. Quite simply, it meant programming.

Today we have *LogoWriter* and *Logo PLUS*. These versions allow the user to create and save both graphics and text and to print the product without writing a procedure. It's not uncommon today to encounter a "Logo teacher" who has never written a procedure and really doesn't realize the power and extensibility of Logo. To that teacher, Logo is not a programming language; rather, it is an application.

As an application package, Logo is still a powerful tool. The ability to mix text and graphics on the same screen makes it a more flexible product than almost any equivalently priced word processor or graphics package on the market. But those of us who have been using Logo for many years are a bit unnerved by the idea of teaching Logo without teaching programming.

However, the idea of integrating a programming language and an application is becoming increasingly common in today's software market. My telecommunications program allows me to write "macros" to log in to a particular telecommunications network. The macro is merely a small program. HyperCard allows the user to easily create hypertext documents without programming, but more sophisticated users quickly find the need to "script." Scripting is, quite simply, writing little programs to accomplish specific tasks. Have you used a sophisticated database system like FoxBase? Behind applications such as this are "command languages." In FoxBase, the "command language" is DBASE — clearly a programming language. And what about spreadsheets? Anyone out there use Lotus?

There is little question in my mind that the boundary between programming languages and applications is blurring. Perhaps the new versions of Logo are simply following a prevailing trend in the software industry, albeit moving from programming language to application rather than the more common move from application towards programming language.

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 can still encourage exploration, discovery, problem solving, thinking, and creativity whether we write code or use Logo as a tool to create products in math or social studies classes. And, like the sophisticated applications mentioned earlier, soon those who view Logo as an application find themselves wanting to do more and thus needing to grow in the direction of programming in Logo's extensible environment.

Sharon Yoder
SIGLogo/ISTE, 1787 Agate Street, Eugene, Oregon 97403
CIS: 73007,1645 BITNET: YODER@OREGON

Monthly Musing

The RT Stuff
by Tom Lough

Back in 1982, when I was pasting up the dot matrix printed *NLX* articles by hand, I often ended up with a few inches of space right at the end of the columns. Rather than leave the space blank, I usually found a right small paragraph or one-line filler to insert.

I happened to be perusing the October 1982 issue of the newsletter recently. I noticed one of these fillers right at the bottom of page 4. It was right much like rediscovering an old friend, as this was one of my favorites. I remembered it was right late at night when I pasted up that particular issue, and I desperately wanted a filler for that space right then. Finally the right words came.

> In Logo, as in life, the direction you are heading is often much more important than how far you go.

The words came in from left field, but as soon as I wrote them down I knew they were exactly right. Not a word was left over. I was left with a sense of accomplishment as I pasted them into the right hand column. I left the finished newsletter on the table and went right to sleep.

By now, I'm sure you have noticed the excessive RTs and LTs scattered throughout the above paragraphs. I just used them to make the point. But, whenever I use the RT and LT commands, I think about that little filler sentence.

It is amazing how many incidents in daily life affect the direction of events, just as using the RT and LT commands affect the direction of the line the turtle draws. If you don't believe me, take a look at an old friend.

```
TO SQUARE
REPEAT 4 [ FORWARD 100 RIGHT 90]
END
```

If you change the 100 to another number, you still get a square. But if you change the 90, the effect is quite different. I suggest that events in life can be classified similarly.

The "directional" incidents produce a change in attitude or spirit, and do not represent just a little more plodding progress toward some well defined but very official curriculum objective.

"I wonder why it did that?"

"Hey, that looks interesting!"

"Could you help me please?"

"You'll never guess what my program can do!"

I suspect that effective teachers might have identified incidents such as these long ago, but may not have realized the significance until later.

"Remember when you wondered why my turtle didn't stop? That got me interested in recursion and..."

"One day, you told me my project was interesting. I never forgot how good that made me feel. That is one of the reasons I got interested in teaching..."

"Do you remember the day I finally asked for help? I was hopelessly stumped and you helped me without embarrassing me. That is why I decided to write this book – so I could help others..."

"I had finally gotten my Logo program to do exactly what I wanted, and I couldn't wait to show you. You watched, understood and appreciated; that made me feel good. Now that I am a professional computer programmer, I get that feeling often, and I think of you each time..."

I believe that Logo and Logo teachers have the RT stuff to make a significant difference in the lives of students. I believe that it is our RT and duty to do so.

Perhaps the RT and LT commands can serve to sensitize us to the importance of affecting attitudes as well as facilitating cognitive growth in our students. But, don't forget: Once the appropriate direction is set, then it's

FD 100!

Tom Lough
Founding Editor
PO Box 394
Simsbury, CT 06070

Logo Ideas

The Subject Is Poetry
Teacher Focus: Robert MacDonald
by Eadie Adamson

There are many extraordinary teachers who have found wonderful ways of incorporating the use of Logo into their classroom teaching. To do this seamlessly so that the use of the computer becomes a integral part of the learning process, not simply an added flourish or something thought of as quite separate from the normal course of study, is one of the problems many of us are wrestling with today, whether we teach in a laboratory or a classroom situation. We do not want Logo to be viewed simply as a programming language, but rather as a useful tool which can be readily appropriated by our students as part of their materials (much like to paint, glue, cardboard, and so on) as they work on a project assignment. At the same time we do not want to neglect the teaching of programming, but rather to incorporate those skills in a context meaningful to our students.

During the past year, Sharon Yoder called my attention to the work of one teacher who has been very successfully and inventively making this integration of Logo an important part of his curriculum in many areas. Robert MacDonald teaches a fourth grade class in Grosse Isle, Michigan. Robert has been a student in the Distance Learning course in Logo offered through ISTE (For more information on these courses, see the advertisement in this issue). As instructor for the Logo course, Sharon came to know Robert's work. Sharon forwarded to me a number of remarkable samples of Robert's extremely inventive methods of integrating *LogoWriter* into his curriculum. I, in turn, have corresponded with Robert, who has gladly shared materials with me as well. It seems appropriate to feature some of his work in this column, since my focus for much of the past two years has been on classroom projects using *LogoWriter*.

This month's column will trace an idea Robert has used successfully with his fourth grade classes; an idea which could be readily adapted for older students as well. As you read through this description, notice how well Robert has incorporated the use of *LogoWriter* into the learning process.

Poetry, poetry, poetry

Here is how Robert MacDonald describes his work with his class on what he calls "verbal recursion," a form actually known as accumulative rhymes, a device used also in prose (Harvey, 1961). The form is also sometimes defined as "incremental repetition" (Shaw, 1972). The recursion Robert refers to is actually in the recursive procedure used to generate the poem (see the listing below). The major portion of this project is done without the computer, but as you follow though his steps, note how interestingly the work culminates in the use of Logo and the computer to produce something which might have been quite difficult and time-consuming for children to produce as a standard wordprocessing task.

> On the first day — a Monday — I introduced several recursive poems — "This Is the House That Jack Built" and the wonderful "Drummer Hoff," adapted by Barbara Emberley and illustrated by Ed Emberley (Emberly, 1967). I asked the children to think about how they might go about writing something in this style... On the following day I suggested that we construct a verbal recursion of our own. I began with the first sentence: "This is the computer that runs all night." We finished it within a very short period of time...Most children had no problems with seeing how the sentences had to run into each other. The next day I asked how many of them would like to construct their own poem. The class agreed that it would make a good writing assignment.

Robert wisely has allowed the children to make the choice of assignment, or at least has constructed the situation so that the children could feel a part of the process, making the decision to work on this project rather than being *assigned* the project. This creates an important sense of ownership for the student, something which I find strongly desirable for a Logo environment. Notice also that the work is still within the normal classroom structure. The computer has not made its appearance yet in this project.

Robert continues:

> I gave them until the end of the week to turn them in to me. Most of the children submitted their work within two days. I did a minimum amount of editing (largely spelling) and had them ready to hand back for our computer lab that Friday.

The handwritten work was then ready for the next step, using the computer.

> On Friday I presented the group with a computer copy (printout) of "This is the computer that runs all night." I asked them to go to the last stanza. I requested them to type this stanza — going backwards — into the procedure **collect**...that each of them had on their scrapbook disks. All they had to do was remove lines of poetry already there and substitute the last stanza of the computer poem. It

was an easy task to explain about the use of brackets. I merely had to watch over the work of some five students. The task before them now was to put in their own poems and see if they could get them to output successfully. They were very interested in accomplishing this.

Robert had placed a copy of a program, original by David Thornburg (Thornburg, 1986), on each child's scrapbook disk. The program takes the lists added in **collect** and prints on the screen the stanzas of the poem, adding a line each time. (See the end of this article for a revised version I wrote of the same program.) Thus, with a minimum of typing, the students can produce a computer version of a very long piece of poetry, a task which would be beyond tolerance of teacher or student if done with a standard word processor. Robert not only had some work prepared for the children to use, but he created an opportunity to teach them about using Logo lists, an important concept that here is elevated to a higher level than the children had previously encountered through learning about the use of **repeat.**

Robert took the children's disks and checked over spelling, making an interesting insight that might merit further research:

> I only had four children who had to make some minor corrections. It has always amazed me that children rarely make silly errors while typing on a computer, but cannot prevent them from occurring all over a sheet of paper while writing with a pencil.
>
> On Monday and Tuesday of Thanksgiving week the children were required to get two printouts of the poem. One they had to mount on a large colored sheet of paper — title it and illustrate the contents on the margins — the other they handed in to me.

Robert goes on to describe the attention a bulletin board outside his classroom attracted after he displayed this poetry assignment. Other students have become interested; his colleagues have asked to do the same assignment. "I think we have a movement afoot," he comments. Robert again has found an interesting way to spread the use of *LogoWriter*. In addition to agreeing to supply other teachers with the program, he has volunteered not himself but *his students* to be teachers for the other classes if they need help. What a wonderful piece of sharing this represents! The students become more secure in their own knowledge as they share it with others; teachers begin to see the value of cooperative learning and, very importantly, discover that there are things they can have their classes do with *LogoWriter* even if they know little or nothing about it themselves.

What Did They Learn?

One might ask, indeed, what these children might have learned from this exercise. They were not required to do any original programming, but they were asked to think about and to work out a process. When they entered their work into the appropriate procedure in the computer, they were learning about using the editor, they had to understand and apply the rules about using brackets to create lists, including the need for complete sets of matched brackets. They had a chance to experience a very complex recursive procedure in operation, one which they were free to examine if they wished. The exercise actually encouraged them to tinker a bit with the program, since they were customizing the **collect** procedure to reflect their own writing. Lastly, they learned about printing their work, since the assignment required them, not the teacher, to produce those two copies of the completed poem.

How Does This Fit?

In a holiday setting, one might try applying this to a holiday theme. Robert found an interesting new version of a traditional holiday song, "The Twelve Days of Christmas" (Trivas, 1988). A teacher might use the original "Twelve Days..." as a starting point and ask the children to dream up something of their own. For a short exercise, each child might be assigned a line to add. In addition to adding the line, a Logo illustration might be required. A challenge for you, the teacher: a fully "computerized" version might then, as a line is added, move to a page illustrating that line, then back to the main page to generate the next verse. Can you work it out?

The Poetry Program

Students merely need to add the lines to the **collect** procedure below. Each line goes in its own set of brackets. To see the poem, simply type **poetry**. The first line of **poetry**, **if not front? [flip]**, is a good safety device to know about. Any time you use **ct** (cleartext), precede it by this command. If by chance the procedure is typed in the Command Center on the Flip side, the procedure will detect that fact and flip to the front before proceeding. (**front?** is a reporter which reports **"true** if on the front of the page, **"false** if on the flip side; **flip** merely turns the page as if one had pressed the flip keys.)

```
to poetry
if not front? [flip]
ct
collect
make "num count :lines
poem :num
end
```

Logo Ideas, continued

```
to poem :num
if equal? :num 0 [stop]
poem :num - 1
print [This is]
printlines  :num
end

to printlines :num
if equal? :num 0 [print [] stop]
print item :num :lines
printlines :num - 1
end

to collect
make "lines [ [The house that Jack
   built.] [The malt that lay in]
   [The cat that ate up] ]
end
```

This exercise gives children a chance to create a long poem with very little typing, but at the same time it gives you an opportunity to teach about making lists of Logo objects. It may be helpful to describe a list as a "package" which is created by enclosing in the square brackets. A collection of these "packages" will hold the various lines of the poem. Most children will not have much trouble with the brackets, but if you are in doubt, try changing **collect** so that it reads like this:

```
to collect
make "lines [
   [The house that Jack built.]
   [The malt that lay in]
   [The cat that ate up]

                                  ]
end
```

Instruct your students to delete only the the brackets which contains words, and to leave the rest as they were. Unlike other versions of Logo, *LogoWriter* allows you to format your procedures and does not change them when you leave the editor.

Here's a challenge! Can you write a procedure which lets students type a word (like **add**), followed by a list of lists, which will then create a new list in the **collect** procedure?

Here's a starter which should delete the old lists from your collect procedure:

```
top
search "to\ collect
unselect          <—the \ and a space make
                     it one word (search
cf                   takes a single input)
search "\[
unselect select cf
eol
cb
cut
```

From here you need to **insert** the list collected when the student has invoked **add**. If you want to try this but can't solve the problem, write to me!!

References:

Emberley, Barbara, adaptor and Emberley, Ed, illustrator (1967). *Drummer Hoff.* New York: Prentice Hall, Inc.

Harvey, Sir Paul, ed. (1967). *The Oxford Companion to English Literature.* Oxford: The Clarendon Press. See the discussion of "The House That Jack Built," p. 400. See also Baring-Gould, William S. and Baring-Gould, Ceil (1961). *The Annotated Mother Goose.* New York: Bramhall House. P. 43, note 65.

Shaw, Harry (1972). *Dictionary of Literary Terms.* New York: McGraw-Hill. See the entry, **incremental repetition**, p. 200.

Thornburg, David D. (1986). *Beyond Turtle Graphics.* Menlo Park, CA: Addison-Wesley Publishing Company. See pages 97 - 102.

Trivas, Irene. (1988). A delightful new version of "The Twelve Days of Christmas", *Emma's Christmas, an old song re-sung & pictured.* New York and London: Orchard Books.

Eadie Adamson is a Computer Coordinator at The Allen-Stevenson School, an independent school for boys in New York City, where *LogoWriter* is used intensively both in scheduled computer classes and in French language classes. She is also a consultant for Logo Computer Systems, Inc., giving teacher workshops in New York City, and an instructor at Teachers College, Columbia University, where she teaches a course in Logo for educators, "Teaching Computing to Children Using LOGO."

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

Customizing Logo with a Startup File

by Diane Miller

(Note: Some of the information discussed here will not apply to all versions of Logo. It generally applies to Apple II Logo and LCSI Logo II, and to a lesser extent to *LogoWriter*, as indicated.)

Although there are many different versions of Logo, each with a slightly different set of primitives, every implementation appears to have some limitations in the primitives available — i.e. no primitives for certain commonly desirable operations. For example, in most (if not all) versions there are no commands to draw circles and arcs of a specified size, which add immensely to the power and fun of Logo graphics. In addition, many of the primitives have built-in abbreviations — such as FD for FORWARD — which make Logo much easier to use for young children, special-education users, and Ph D's who cannot type. But some of the primitives do not have abbreviations and are tedious to type, such as .SETSCRUNCH, DRIBBLE "PRINTER, and so forth. Many users might find abbreviations for some of these primitives useful.

There are good reasons for a programming language such as Logo to have a fairly minimal set of built-in primitives. For instance, it must not occupy too much memory space. And since it is an extensible language, users may use existing primitives to define their own new customized tool procedures.

You can customize Logo to your own needs by making procedures to draw circles and arcs, define abbreviations or otherwise rename commands. For instance you can make a simple procedure to create an abbreviation for printing.

```
TO PPIC                  TO PS
PRINTPIC "PRINTER        PRINTSCREEN
END                      END
```

With one of the above procedures in memory, the user need only type PPIC instead of PRINTPIC "PRINTER in LCSI Logo II or PS instead of PRINTSCREEN in *LogoWriter*.

But adding such tool procedures to Logo would seem to entail some work. Do you have to type them each time you use Logo? Mercifully, no. You *can* create tool procedures once, save them in a file, then load the file each time you need it. But wait! There is an even easier way.

Creating a Startup File

If these tool procedures are saved in a file with the special name STARTUP, they are loaded automatically whenever you start Logo. When Logo starts, an introductory screen is presented (I will resist calling it a logo), along with the message "PRESS RETURN." When you press Return, the disk is accessed again for some seemingly mysterious purpose, and Logo is ready to go. What is happening is that Logo is looking for a file named STARTUP. If it finds this file, it reads it into memory just as it would load any other file. (If it does not find it, Logo starts with only its built-in primitives available.)

Since the message "PRESS RETURN" does not prompt the user to switch disks, it is convenient to have the STARTUP file on the Logo language disk itself. You cannot put it on the original distribution disk because it is permanently write-protected. But if you can make a working backup copy of the disk, you can add a STARTUP file to it.

Keep in mind that the only sensible use of an original master disk, for any software, is to make a backup copy which you actually use to run the program. Keep the master disk in a safe place for the day when the backup gets zapped or lost and you need to make another. Some versions of Logo are copy-protected, so the disk-copying utility on the DOS disk will not work. Often in these cases a backup can be made with a special copying utility such as Copy II Plus (Central Point Software). The making of a single backup by the registered owner for daily use so that the master may be kept on file is completely legal.

If you don't want to put your STARTUP file on the language master disk, you can also put it on a files disk. You must then make sure this files disk which has the STARTUP file on it is in Drive 1 before you press Return. All of Logo has been loaded when the "PRESS RETURN" message appears. All that remains is to look for the STARTUP file. You can have as many different STARTUP files as you want to be used for different circumstances and users. However, each STARTUP file must be located on a separate file disk.

If you are using *LogoWriter*, you create a page called STARTUP. When you start *LogoWriter*, that STARTUP page is automatically loaded into memory so that you begin on the STARTUP page instead of the Contents page.

Startup Variables and Burying Procedures

There is a feature which may be used with any Logo file, called a startup variable, which gives a list of commands which will be executed whenever the file is loaded. A startup variable is created with the MAKE command, as is any other global variable, but is given the special name STARTUP. All

Customizing Logo with a Startup File--continued

current global variables are saved, along with procedures, when a SAVE command is given. (If you are using *LogoWriter*, you create a procedure named STARTUP since variables are not saved with the page.)

In addition, you can use the startup variable to cause the procedures in the STARTUP file to be hidden (buried) after they are loaded, so they appear to be built-in Logo primitives. This means they do not erase with ERALL, do not clutter up the list of procedure titles in memory, and do not SAVE with the user's procedures. The only way in which you might notice these additional procedures is that they take up some memory, and of course you cannot give a user-defined procedure the same name as one of the tool procedures. (In *LogoWriter*, you bury procedures by using LOADTOOLS to get procedures from one page to use on another.)

There is a potential problem with using buried procedures, as pointed out by Burrowes. If your procedures use buried circle and arc procedures which got into memory via a STARTUP file, they will not work if Logo has been started without that STARTUP file (or without one which contains the needed procedures). This is because the buried procedures did not save with the other procedures in your file. So, it is best for non-sophisticated users to keep the STARTUP file as simple as possible and just use one version. I keep the same STARTUP file on the language disk and each files disk, so it gets loaded no matter what disk is in when that "PRESS RETURN" message comes up.

One caution: If you already have a STARTUP file but want to experiment with others, you may wish to keep a copy of the old one in case it contains procedures you may want later. The simplest thing to do is to rename it with the command: RENAME "STARTUP "OLDSTARTUP.

How to Make a Startup File

1. First check your procedure titles to make sure there are no stray procedures in memory which you do not want included in your STARTUP file. Erase any stray procedures with the ERASE command.

2. Check for stray memory variables in the same way, using the command PONS ("print out names"). They may be erased with the ERNAME command.

3. Get the desired procedures into memory by loading them from another file or typing them. You might try typing some of those given below.

4. Create the memory variable to bury the procedures by typing:

```
MAKE "STARTUP [BURYALL]
```

5. Then to save the file type: SAVE "STARTUP

How to Modify a Buried Startup File

1. Start Logo and let the STARTUP file load.

2. Type: `UNBURYALL`

3. Edit, add, or remove procedures as you wish.

4. Type: `SAVE "STARTUP`

How to Copy a Buried Startup File from One Disk to Another

1. Start Logo and let the STARTUP file load.

2. Type: `UNBURYALL`

3. Put the destination disk into the drive. (A backup of the Logo disk or a Logo file disk.) Type:

```
SAVE "STARTUP
```

Once you have a blank file disk with the appropriate STARTUP file on it, you can make extra copies of the disk with the Apple II System Utilities Disk, for Pro-DOS users, or the DOS System Master Disk for DOS 3.3 Users. (You can also copy the STARTUP file using the DOS file-copying utility.)

Some Suggested Procedures for a Startup File

Circles and Arcs

These procedures give the user a very good set of tools which will draw circles and arcs of a given radius, to fit other components of a drawing. To draw a circle of a given radius, starting from the turtle's present position and heading, with turns to the right:

```
TO RCIR :RADIUS
MAKE "X :RADIUS * (3.14159 / 18)
REPEAT 36 [RIGHT 5  FORWARD :X
   RIGHT 5]
ERN "X
END
```

To draw a circle of a given radius, starting from the turtle's present position and heading, with turns to the left:

```
TO LCIR :RADIUS
MAKE "X :RADIUS * (3.14159 / 18)
REPEAT 36 [LEFT 5  FORWARD :X
   LEFT 5]
ERN "X
END
```

To draw a circle of a given radius centered around the turtle:

```
TO CCIR :RADIUS
MAKE "X :RADIUS * (3.14159 / 18)
PENUP
FORWARD :RADIUS
RIGHT 90
PENDOWN
REPEAT 36 [RIGHT 5  FORWARD :X
   RIGHT 5]
LEFT 90
PENUP
BACK :RADIUS
PEDOWN
ERN "X
END
```

(Note: This procedure will draw a circle even if the pen was up or in erase or reverse mode, because it contains an explicit PENDOWN. For this reason, a CCIR cannot be erased by issuing a PENERASE and redrawing the circle. It can be erased by changing the pen color to the background color and redrawing the circle. It can also be erased by moving the turtle to the perimeter — an easy task since you know the radius — and drawing an RCIR or LCIR in PENERASE mode. The procedure as written will also leave the pen down when it finishes.

A more sophisticated CCIR procedure can be written which will check the pen's and turtle's state prior to running and return them to the same state on finishing:

```
TO CCIR :RADIUS
MAKE "PEN? PEN
MAKE "SHOWN? SHOWNP
MAKE "X :RADIUS * (3.14159 / 18)
HIDETURTLE
PENUP
FORWARD :RADIUS
RIGHT 90
PENDOWN
REPEAT 36 [RIGHT 5  FORWARD :X
   RIGHT 5]
LEFT 90
PENUP
BACK :RADIUS
IF :PEN? = "PENDOWN [PENDOWN]
IF :PEN? = "PENERASE [PENERASE]
IF :PEN? = "PENREVERSE [PENREVERSE]
IF :SHOWN? [SHOWTURTLE]
ERN "PEN?
ERN "SHOWN?
ERN "X
END
```

To draw a 90-degree arc of a given radius, starting from the turtle's present position and heading, with turns to the right:

```
TO RARC :RADIUS
MAKE "X :RADIUS * (3.14159 / 18)
REPEAT 9 [RIGHT 5  FORWARD :X
   RIGHT 5]
ERN "X
END
```

To draw a 90-degree arc of a given radius, starting from the turtle's present position and heading, with turns to the left:

```
TO LARC :RADIUS
MAKE "X :RADIUS * (3.14159 / 18)
REPEAT 9 [LEFT 5  FORWARD :X  LEFT 5]
ERN "X
END
```

There are two more useful procedures for variable-length arcs, given by Abelson, page 30. To draw an arc of a given radius and sweeping a given number of degrees, starting from the turtle's present position and heading, with turns to the right:

```
TO ARCR :RADIUS :DEGREES
ARCR1 0.174532 * :RADIUS
   :DEGREES / 10
IF 0 = REMAINDER :DEGREES 10 [STOP]
FORWARD 0.174532 *
   :RADIUS / 20 / REMAINDER
   :DEGREES 10
RIGHT REMAINDER :DEGREES 10
END

TO ARCR1 :STEP :TIMES
REPEAT :TIMES [RIGHT 5  FORWARD :STEP
   RIGHT 5]
END
```

Customizing Logo with a Startup File--continued

To draw an arc of a given radius and sweeping a given number of degrees, starting from the turtle's present position and heading, with turns to the left:

```
TO ARCL :RADIUS :DEGREES
ARCL1 0.174532 * :RADIUS
   :DEGREES / 10
IF 0 = REMAINDER :DEGREES 10 [STOP]
FORWARD 0.174532 *
   :RADIUS / 20 / REMAINDER
   :DEGREES 10
LEFT REMAINDER :DEGREES 10
END

TO ARCL1 :STEP :TIMES
REPEAT :TIMES [LEFT 5  FORWARD :STEP
   LEFT 5]
END
```

Abbreviations

To send a picture on the screen to the printer:

```
TO PPIC
PRINTPIC "PRINTER
END
```

To turn on the printer to echo the text on the command line — useful for POTS or POPS, or just to record commands typed:

```
TO PRON
DRIBBLE "PRINTER
END
```

To turn off the printer after PRON:

```
TO PROFF
NODRIBBLE
END
```

To abbreviate .SETSCRUNCH:

```
TO SS :N
.SETSCRUNCH :N
END
```

SETSCRUNCH changes the aspect ratio of the screen by stretching or shrinking turtle steps in the vertical dimension. It is useful for adjusting the picture on the screen if you cannot achieve enough adjustment with the controls on the monitor. It is also useful for situations in which a square on the screen comes out a rectangle on the printer, because it allows you to distort the picture on the screen so it prints true. .SETSCRUNCH also allows you to draw ellipses with the circle commands, or make squares into rectangles. Unfortunately, this command is not available in LogoWriter.

Using the pattern of these last three procedures, any Logo command can be renamed/abbreviated.

Miscellaneous

If you are working with buried procedures in your workspace and issue the command UNBURYALL, you will unbury the STARTUP procedures as well. You can make a procedure to selectively rebury the procedures in your STARTUP file. Note that any time you change the procedures in your STARTUP file, you must change their names in the list in this procedure.

```
TO REBURY.SU
BURY [RCIR LCIR CCIR RARC LARC ARCR
   ARCR1 ARCL ARCL1]
BURY [PPIC PRON PROFF SS REBURY.SU]
END
```

(Note that the procedure REBURY.SU can bury itself!)

References

Abelson, H. (1982). *Apple Logo*. New York, New York: BYTE Publications, McGraw-Hill.

Burrowes, S. (1985). Circles, Arcs, and Headaches. *The National Logo Exchange*, Volume 3, Number 5, p. 1.

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

About the Cover

This drawing was done by Dantao Kong who is in the 9th grade. Dantao has been in the U.S. only two years. This drawing was done in Fran Rothkin's computer class at JHS 217 Queens in New York City.

Beginner's Corner

A Logo Melody
by Dorothy Fitch

If you're like me, you need a ruler to draw a straight line! You love to draw intricate designs in Logo because it is so easy to make an impressive picture. But Logo can be more than a tool for making drawings. It can also be a tool for writing music. Even if you don't consider yourself a musician, you can compose original music or recreate melodies that you might not be able to play by yourself on another instrument.

For this column, I chose a traditional round that you can enjoy throughout the upcoming holiday season. You and your students don't have to know much about music to enjoy this project, since it is a fairly simple song. And there are many possibilities for using it to explore much more. (See the ideas listed at the end of this column.)

With only a little introduction to the language of music and some clues, you and your students can figure out everything you need to know to take this song written in musical notation and turn it into a program written in Logo. You can use the Copy-Me page at the end of this column, use the program listing given below as a guide, or you can study the documentation that comes with your version of Logo. You'll find that this is almost more of a mathematics lesson than a music lesson. You'll be exploring fractions, number relationships, and even critical thinking as you collect the information you need to write the song in Logo.

The Music

The main program in this piece is called DONA, which directs Logo to the procedure called I, then the procedure called II, and finally the procedure called III. Each of these procedures is divided into two parts, A and B. Each part is a musical phrase, which corresponds to a sentence in English—an idea with a complete thought.

Dividing the song in this way makes musical sense, and it also helps keep each of the procedures a manageable length. Imagine the problem of finding a wrong note if the entire piece were written in one procedure! (Keep this strategy in mind when you create graphics programs too!)

The procedure for each phrase is the one that actually contains instructions to play the notes. The command NOTE takes two inputs—the pitch of the note (how high or low it is) and its duration (how long it lasts). Enter the procedures below or use the Copy Me page to figure out the procedures on your own. In the Logo notation used in this column, note names that include a ' are in a higher octave and ! is a symbol for flat. Consult a friendly music teacher if your curiosity extends beyond what can be included here!

You can change the tempo (speed) of the music by altering the number values for duration. The lower the number, the shorter the duration.

Program Listing for Logo PLUS:

```
TO DONA
I
II
III
END

TO I
IA
IB
END

TO II
IIA
IIB
END

TO III
IIIA
IIIB
END

TO IA
; 1
NOTE F 30
NOTE C 30
NOTE A 120
; 2
NOTE G 30
NOTE C 30
NOTE B! 120
; 3
NOTE A 60
NOTE G 60
NOTE F 60
; 4
NOTE F 60
NOTE E 120
END
```

Beginner's Corner--continued

```
TO IB
; 5
NOTE D' 60
NOTE C' 30
NOTE B! 30
NOTE A 30
NOTE G 30
; 6
NOTE C' 90
NOTE B! 30
NOTE A 60
; 7
NOTE A 30
NOTE G 30
NOTE F 60
NOTE E 60
; 8
NOTE F 180
END

TO IIA
; 9
NOTE C' 180
; 10
NOTE C' 180
; 11
NOTE C' 60
NOTE B! 60
NOTE A 60
; 12
NOTE A 60
NOTE G 120
END
```

```
TO IIB
; 13
NOTE D' 60
NOTE D' 120
; 14
NOTE C' 60
NOTE C' 120
; 15
NOTE C' 30
NOTE B! 30
NOTE A 60
NOTE G 60
; 16
NOTE F 180
END

TO IIIA
; 17
NOTE F 180
; 18
NOTE E 180
; 19
NOTE F 90
NOTE G 30
NOTE A 30
NOTE B! 30
; 20
NOTE C' 60
NOTE C 120
END
```

```
TO IIIB
; 21
NOTE B! 60
NOTE B! 120
; 22
NOTE A 60
NOTE A 120
; 23
NOTE E 30
NOTE G 30
NOTE C' 60
NOTE C 60
; 24
NOTE F 180
END

TO C
OUTPUT 523.26
END

TO E
OUTPUT 659.26
END

TO F
OUTPUT 698.46
END

TO G
OUTPUT 784
END
```

```
TO A
OUTPUT 880
END

TO B!
OUTPUT 932.3
END

TO C'
OUTPUT 1046.52
END

TO D'
OUTPUT 1174.64
END
```

Changes for Terrapin Logo for the Apple and Commodore Logo:

To use music in Terrapin Logo, you first need to load the music file from your Logo Utilities Disk. Type READ "MUSIC and press Return. After the music procedures are loaded into your Logo workspace, you'll need to add the following procedure. To enter the editor, type TO and press Return.

Type the following lines, making sure to use the correct punctuation and spacing. Press Return at the end of each line. Press Control-C when you are through to define the new procedure.

```
TO NOTE :PITCH :DURATION
.CALL.2 :TONE :PITCH - :FUDGE
   :DURATION * :BASE.PERIOD / :PITCH
END
```

You also need to assign different numbers to the pitches. Use the following numbers instead of those that appear in the procedures for the note names above.

```
C  = 67.1866    F  = 50.3481
A  = 39.9711    C' = 33.6175
E  = 53.339     G  = 44.8607
B! = 37.73      D' = 29.9535
```

Changes for PC Logo:

Use the command TONES instead of NOTE. Divide each of the pitch frequencies in half. For example, the procedure for C could be written:

```
TO C
OUTPUT 523.26 / 2
END
```

Use 6 for the quarter note duration instead of 60. Adjust the other values accordingly.

Changes for *LogoWriter*:

Use the command TONE instead of NOTE. Use 40 for the quarter note duration instead of 60. Adjust the other values accordingly. You may have to adjust some of the note numbers to make them more in tune. The higher the number, the higher the pitch of the note. Consult your documentation for the exact note numbers to use.

Other Ideas for Dona Nobis Pacem

Once your computer can play this song, you and your students can try some of these ideas:

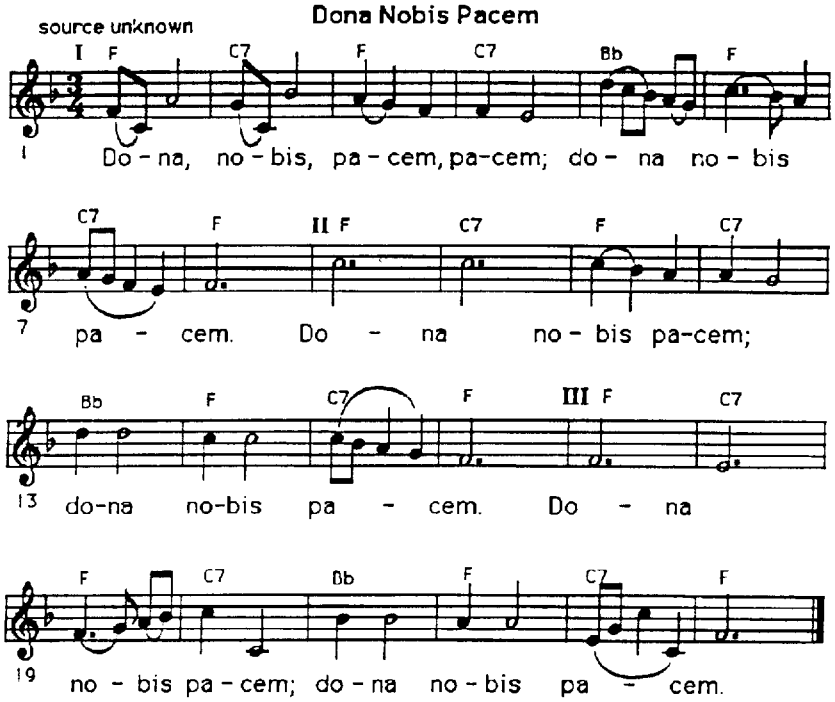
- Get two or three computers playing it as a round (each part of the round begins when the previous one gets to the beginning of the next section, marked I, II and III)
- sing along with the computer to help learn the song
- Play an instrument (flute, recorder, violin, etc.) in a round with the computer
- Practice playing an autoharp or guitar using the chord symbols F, C7 and Bb
- Think of as many English words as you can that have the same root as 'Dona' and 'Pacem'
- Find the word for 'Peace' in as many foreign languages as you can
- Include this song in your school's holiday program!

Dorothy Fitch has been the Director of Product Development at Terrapin, Inc. She first become involved in educational technology in 1981 when the school where she taught music received its first computer. Since that time, as a consultant she has provided schools with inservice training, curriculum development and software customization; taught a number of college courses; and directed a computer classroom for teachers and students. She has also coauthored *Kinderlogo*, a single keystroke Logo curriculum for young learners, and created the *Logo Data Toolkit*. Through her work at Terrapin, she has presented at many local, regional and national conferences, edited many of Terrapin's curriculum materials, and brought *Logo PLUS* to life.

She can be contacted at Terrapin's new address:

Terrapin Logo. Inc.

has moved to a new address.
They can now be reached at
400 Riverside
Portland, Maine 04103
207-878-8299



Nobody knows who wrote this song. It is a three-part round and the Latin words mean "Grant us peace."

With a bit of detective work, you can write this song in Logo. To play a note, you need to know two things—the pitch of the note (how high or low it is) and its duration (how long it lasts). The clues on this page will help you figure out this information. When you know these two things about each note, you are ready to use the NOTE command and combine them to make the whole song.

Clue #1: There are 3 beats in every measure. (A measure is the group of notes between the vertical bar lines. For example, the first measuregoes with the word 'Do-na' and has three notes.)

Clue #2: This chart shows the 5 different types of notes in this piece and how long the sound lasts for each note. Part of this chart is already filled in for you. See if you can fill in the rest of the durations by looking at the music and seeing how the notes fit together in a measure.

Clue #3: Here is a chart that tells you the names of the notes you will need for this piece. The ' at the end of two note names indicates that it is a high sounding note. The ! after the B indicates that it is a flat note (a black key on a keyboard).

When you know the pitch and duration of each note, use the NOTE command like this:

NOTE *pitch duration* Example: NOTE C 30 (for a quarter note C)

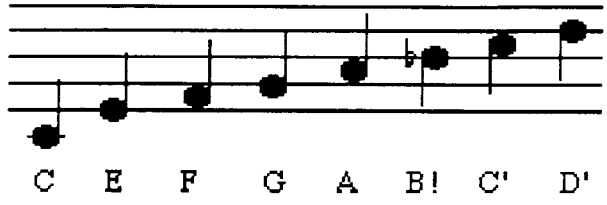
Ask your teacher for help in combining the Logo commands into a Logo procedure. Only put 4 measures of notes in the same procedure. That will make it much easier to fix your program if you find a bug! Have fun!

Clue #2

Note	Name	Duration
	Eighth note (two can be connected)	____
	Quarter note	60
	Dotted quarter note	____
	Half note	____
	Dotted half note	____

Clue #3

Notes needed for Dona Nobis Pacem



Logo LinX

Alphabet Adventures
by Judi Harris

Some classroom activities are methodological staples, like the quart of milk, loaf of bread and laundry detergent that my aunt Esther always asked us to bring her when we went to the grocery store. These lesson formats survive, while educational fads come and go, because they are simple, philosophically non-partisan, and intrinsically engaging.

As the November and/or December school holidays approach, perhaps many of you will decide to carry on the pedagogical tradition by giving your students hidden word matrices, lotto games, crossword puzzles, or similar classic learning activities to do. One such popular pursuit is what has been called an *alphabit*; a word from which smaller words can be formed by rearranging component letters.

Perhaps you remember being asked to do this with the words THANKSGIVING or CHRISTMAS when you were a student. The goal is usually to form as many subwords as possible from the letters in the alphabit, making sure that the words constructed do appear in the dictionary. As a teacher, you may now notice that this is an effective way to encourage spelling practice, vocabulary exploration, and collaborative student efforts. Some simple Logo tools may assist this time-tested educational activity, so that your students can concentrate fully upon forming and researching words, instead of dividing their attention between such exploration and the mechanics of playing this sort of educational game.

Choosing Challenges

One easily-amended Logo tool procedure can store alphabits with which to work.

```
TO WORDS
OUTPUT [JURISDICTION QUALIFICATION
   HEMOGLOBIN ENIGMATIC NEVERTHELESS
   SEDENTARY HORSERADISH UNEMPLOYMENT
   SATISFACTION ADVENTURE COMPLEXITY
   DANDELIONS]
END
```

Another tool, used in tandem with WORDS, can direct the computer to choose an alphabit with which to challenge the student. The well-known PICK procedure will do this nicely.

```
TO PICK :OBJECT
OUTPUT ITEM ( 1 + RANDOM COUNT
   :OBJECT ) :OBJECT
END
```

These two tools can be combined in a number of ways, such as

```
PRINT PICK WORDS
```

to which the computer may respond

```
SATISFACTION,
```

or

```
MAKE "ALPHABIT PICK WORDS
```

if the choice should be retained as the value of the global variable ALPHABIT.

Lexicographic Requests

To facilitate experimentation with the letters of an alphabit, the computer can check to see that subwords that a user forms are indeed only comprised of letters contained in the alphabit itself, and keep a record of all such subwords formed.

The superprocedure BEGIN randomly chooses an alphabit challenge. (Please note that all procedures are written in *LogoWriter* 2.0, but can easily be adapted to function with any full-featured Logo).

```
TO BEGIN
CC
MAKE "ALPHABIT PICK WORDS
MAKE "SUBWORDS [ ]
SOLUTIONS
HT
CT
PRINT SENTENCE [Please spell a word
   using some of the letters in:]
   :ALPHABIT
CD
MAKE.WORDS.FROM :ALPHABIT
COMPARE.WITH.LIST.FOR :ALPHABIT
END
```

The subprocedures MAKE.WORDS.FROM and TEST.LETTERS.IN prompt the alphabetic explorer to form words, automatically checking them against the alphabit's component letters before committing user-generated subwords to a list of successes.

```
TO MAKE.WORDS.FROM :WORD
TYPE [YOUR WORD?]
MAKE "SMALLWORD FIRST READLISTCC
IF :SMALLWORD = "q [STOP]
TEST.LETTERS.IN :SMALLWORD :WORD
 MAKE.WORDS.FROM :WORD
END
```

Logo Linx--continued

```
TO TEST.LETTERS.IN :SUBWORD :WORD
IF EMPTY? :SUBWORD [MAKE "SUBWORDS
   SENTENCE :SUBWORDS :SMALLWORD
   STOP]
IFELSE MEMBER? (FIRST :SUBWORD) :WORD
   [MAKE "WORD REMOVE (FIRST :SUB-
   WORD) :WORD]  [TRY.AGAIN (FIRST
   :SUBWORD ) STOP]
TEST.LETTERS.IN (BUTFIRST :SUBWORD)
   :WORD
END
```

The Letter of the Lexicographic Law

As you can see, both of these procedures are written using tail-recursive structures, one nested within the other. MAKE.WORDS.FROM prompts the user repeatedly for subwords; TEST.LETTERS.IN checks their entries against the letters from the original alphabit. Allison Birch's REMOVE tool is especially helpful in this application for making sure that students use only the letters contained in the alphabit.

```
TO REMOVE :ITEM :OBJECT
IF EMPTY? :OBJECT [OUTPUT :OBJECT]
IF :ITEM = (FIRST :OBJECT ) [OUTPUT
   BUTFIRST :OBJECT]
IF LIST? :OBJECT [OUTPUT FPUT (FIRST
   :OBJECT ) REMOVE :ITEM BUTFIRST
   :OBJECT]
OUTPUT WORD (FIRST :OBJECT ) REMOVE
   :ITEM BUTFIRST :OBJECT
END
```

If an error of this type is made, TRY.AGAIN gives the user specific feedback on the nature of his/her lexicographic transgression.

```
TO TRY.AGAIN :LETTER
PRINT (SENTENCE  [Sorry! There aren't
   enough] WORD :LETTER "'s [in]
   :ALPHABIT [to spell] :SMALLWORD )
CD
END
```

Monitored Musing

Once the user decides that they would like to stop entering subwords, they can type 'q', which is recognized in MAKE.WORDS.FROM as the cue to return to the last line of BEGIN, and execute COMPARE.WITH.LIST.FOR :WORD, which provides feedback on how many subwords were correctly formed during the session.

```
TO COMPARE.WITH.LIST.FOR :WORD
CT
PRINT (SENTENCE [You have formed]
   COUNT :SUBWORDS [words from] WORD
   :ALPHABIT ". )
CD
PRINT (SENTENCE [There are] THING
   :ALPHABIT [words with four or more
   letters that can be made from the
   word] WORD :ALPHABIT ". )
CD
PRINT SENTENCE [To see a list of
   these subwords, type CT LOADTEXT]
   WORD "" :ALPHABIT
CD
PRINT [To see a list of your words,
   type CT PRINT WORD :SUBWORDS]
END
```

The words themselves can act as global variable names, each storing the current number of component words that have been correctly formed from alphabit letters. In this case, these values are assigned in a SOLUTIONS procedure (called in BEGIN), and refer only to subwords of four or more letters.

```
TO SOLUTIONS
MAKE "JURISDICTION 78
MAKE "QUALIFICATION 81
MAKE "HEMOGLOBIN 60
MAKE "ENIGMATIC 48
MAKE "NEVERTHELESS 73
MAKE "SEDENTARY 80
MAKE "HORSERADISH 79
MAKE "UNEMPLOYMENT 79
MAKE "SATISFACTION 57
MAKE "ADVENTURE 76
MAKE "COMPLEXITY 61
MAKE "DANDELIONS 79
END
```

The number of user-generated subwords can therefore be compared with current across-user totals. If the student would like to see a list of their words, or a screen of possible subwords, they can access these by following on-screen directions printed with lines 7 and 9 of COMPARE.WITH.LIST.FOR :WORD. To conserve memory space and expedite tool execution time, lists of possible subwords are stored in separate text files formed with a word processor and accessed with the LOADTEXT command.

Alphabits in Action

It is a testament to the power of a good idea and the versatility of Logo text primitives that rich spelling and vocabulary exploration such as this can be structured and supported with just 9 tool procedures. The explorative environment that they can create for users is reflected in the following sample session.

```
----------ALPHABITS----------
Please spell a word using some of the
letters in:  QUALIFICATION
```

```
Type BEGIN to start game; Q to quit.
YOUR WORD? aqua
YOUR WORD? coin
YOUR WORD? facial
YOUR WORD? flaunt
YOUR WORD? tail
YOUR WORD? uncoil
```

```
----------ALPHABITS----------
Please spell a word using some of the
letters in:  QUALIFICATION
Sorry!  There aren't enough
l's in QUALIFICATION to spell quill
```

```
Type BEGIN to start game; Q to quit.
YOUR WORD? quill
YOUR WORD? q
```

```
----------ALPHABITS----------
You have formed 3 words from
QUALIFICATION.
There are 81 words with four or more
letters that can be made from the word
QUALIFICATION.
To see a list of these subwords, type
CT LOADTEXT "QUALIFICATION
To see a list of your words, type CT
PRINT :SUBWORDS
```

```
CT
LOADTEXT "QUALIFICATION
```

```
----------ALPHABITS----------
            QUALIFICATION
acquaint    coil        finical     natal
acquit      coin        flat        nautical
alit        cola        flaunt      quail
aloft       colt        flint       quaint
anal        count       flit        quilt
anil        cult        foal        quint
antic       facial      font        quintal
aqua        fact        foul        quit
atonal      faction     fount       quoit
aunt        factional   incult      quota
caftan      factual     into        tail
canal       fail        laic        talc
cant        fain        licit       talon
cation      faint       lift        toil
cilia       fatal       lint        tonal
clan        fault       lion        tonic
clout       fiat        loci        tuna
coal        fiction     loft        tunic
coat        fictional   loin        uncial
coati       finial      nail        uncoil
unfatal
```

The 12 text files and tool procedures that were used to develop this language arts Logo application can be obtained in diskette form from the author. If this is your preference, please send her a self-addressed, sufficiently stamped diskette mailer with a 5.25" diskette inside.

References

Birch, A. (1986). *The Logo project book: Exploring words and lists.* Cambridge, MA: Terrapin, Inc.

Manchester, R. B. (1978). *The 2nd mammoth book of word games.* New York: Hart Publishing Company.

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

Survey and Graphing Tools

by Bill Craig

I have been very interested in the survey and graphing tools that have appeared recently in the Logo literature. Stager (1988) and Upton (1988) have described procedures for surveying and graphing written especially for *LogoWriter*. I had interested a teacher at my school in using Upton's Survey procedures to conduct a poll of her class and then graph the results using the Bar, Pie, and Line graph tool procedures. As a whole group activity, the class wrote a set of questions which were entered into the computer by the teacher. I assumed that the procedures would allow the questions to be saved on disk and then answered at a later time. I then discovered a unique weakness of the *LogoWriter* version of Logo. The contents of defined variables are not saved on a page in the same way that those contents are saved in a Terrapin Logo file. The result was that the questions and answers that the class had designed for the survey were erased as soon as the computer was turned off. The survey procedures are great tools but they lost some of their luster for me when I could not enter questions one day and have students answer them the next. What follows is my solution to the problem. It's not pretty, but it works.

The main procedure is START which initializes question and answer lists, asks the survey maker to name the survey, enter questions which are saved under the variable :QUESTION.LIST and answers which are saved under :ANSWER.LIST. As I have written the procedures, the survey is limited to a total of 26 answers with no limit on questions. The user should also know that no more than 5 answers will fit on the screen with the question displayed.

```
TO START
SET.UP
INIT ALPHABET
INIT.VARS
INIT.SURVEY
GET.QUESTIONS :N 1
SAVE
END

TO SET.UP
RG
CT
HT
END

TO INIT.VARS
MAKE "QUESTION.LIST []
MAKE "ANSWER.LIST []
MAKE "NUMBER.LIST []
END

TO INIT.SURVEY
PRINT [WHAT IS THE NAME OF YOUR
   SURVEY?]
MAKE "TITLE  READLIST
PRINT [HOW MANY QUESTIONS WILL YOU
   HAVE?]
MAKE "N FIRST READLIST
MAKE "NUMBER.LIST FPUT :N
   :NUMBER.LIST
END

TO INIT :LIST
IF :LIST =[] [STOP]
MAKE FIRST :LIST 0
INIT BUTFIRST :LIST
END

TO ALPHABET
OUTPUT [A B C D E F G H I J K L M N O
   P Q R S T U V W X Y Z]
END

TO GET.QUESTIONS :N :COUNT
SET.UP
IF :N = 0[STOP]
(PRINT [WHAT IS QUESTION #] :COUNT
   [?])
MAKE "Q READLIST
MAKE "QUESTION.LIST LPUT :Q
   :QUESTION.LIST
PRINT [HOW MANY ANSWERS?]
MAKE "NA FIRST READLIST
MAKE "NUMBER.LIST LPUT :NA
   :NUMBER.LIST
CT
PRINT :Q ANSWER.LIST :NA 1
GET.QUESTIONS :N-1 :COUNT+1
END

TO ANSWER.LIST :N :COUNT
IF :N= 0 [STOP]
(PRINT [WHAT IS ANSWER#] :COUNT[?])
MAKE "A READLIST
MAKE "ANSWER.LIST LPUT :A
   :ANSWER.LIST
ANSWER.LIST :N-1 :COUNT+1
END
```

Once all questions and answers have been entered, SAVE is called. This creates a new page named by the title of the survey, flips to the back of the page, and prints a procedure QUESTIONS which contains the contents of :QUESTION.LIST and a second procedure ANSWERS which contains the contents of :ANSWER.LIST.

```
TO SAVE
NP FIRST :TITLE
FLIP
BOTTOM
PRINT [TO QUESTIONS]
(PRINT [OUTPUT] CHAR 91
   :QUESTION.LIST CHAR 93 )
PRINT [END]
PRINT [TO ANSWERS]
(PRINT [OUTPUT] CHAR 91 :ANSWER.LIST
   CHAR 93)
PRINT [END]
PRINT [TO NUMBERS]
(PRINT [OUTPUT] CHAR 91 :NUMBER.LIST
   CHAR 93 )
PRINT [END]
END
```

The newly created page can now be used to answer the survey questions. To start, simply enter BEGIN:

```
TO BEGIN
INIT
ALPHABET
RESULTS
END
```

VOTE sets all the totals to 0 and repeats the survey questions to as many voters as there are to be polled. When all voters have been surveyed, the totals are displayed through the RESULTS procedure.

```
TO VOTE
SET.UP
INIT.VOTE
PRINT []
PRINT []
REPEAT :N
   [CT MAKE "AI :AI+1
   ASK.QUESTIONS
   MAKE "QCOUNT :QCOUNT +1
   WAIT 15  ]
GO.ON
END
```

```
TO INIT.VOTE
MAKE "QCOUNT 1
MAKE "ACOUNT 1
MAKE "N FIRST NUMBERS
MAKE "AI 1 MAKE "AN 0
END

TO GO.ON
CC
TYPE [PRESS C TO CONTINUE]
MAKE "KEY READCHAR
IFELSE :KEY = "C
   [CC BEGIN]
   [RESULTS]
END

TO ASK.QUESTIONS
PRINT []
MAKE "IN 1
PRINT ITEM :QCOUNT QUESTIONS
MAKE "NA ITEM :AI NUMBERS
REPEAT :NA
   [PRINT [] PRINT []
   (PRINT ( :IN[-] ITEM :ACOUNT
   ANSWERS)
   MAKE "ACOUNT :ACOUNT +1
   MAKE "IN :IN +1]
GET.ANSWER
MAKE "AN :AN + :NA
END

TO GET.ANSWER
CC
TYPE [TYPE THE NUMBER OF YOUR CHOICE
   AND PRESS RETURN.]
MAKE "ANS FIRST READLIST
MAKE "ANS.COUNT THING  ITEM (:ANS +
   :AN )
ALPHABET
MAKE "ANS.COUNT :ANS.COUNT +1
MAKE  ITEM (:ANS + :AN ) ALPHABET
   :ANS.COUNT
END

TO RESULTS
SET.UP
MAKE "TCOUNT 1
MAKE "ACOUNT 2
PRINT []
MAKE "IN 1
MAKE "QCOUNT 1
REPEAT FIRST NUMBERS [TOTALS]
END
```

Survey and Graphing Tools--continued

```
TO TOTALS
CT
PRINT ITEM :QCOUNT QUESTIONS
PRINT []
REPEAT ITEM :ACOUNT NUMBERS
   [(PRINT ITEM :TCOUNT
   ANSWERS [-] THING ITEM :TCOUNT
   ALPHABET)
MAKE "TCOUNT :TCOUNT +1]
MAKE "QCOUNT :QCOUNT +1
MAKE "ACOUNT :ACOUNT +1
GO.ON
END
```

These procedures are good examples of what is called "winning ugly." In other words, I do not think they will be used as examples in any future computer science textbooks. But the procedures work and serve the instructional purpose for which they were designed. Students are now able to take the survey results and use the graphing tools to experiment with how best to display the information they have collected. And that's good enough for me.

References

Stager, Gary. (1988). Another Bunch of Election Stuff. *Logo Exchange*, *7* (1).

Upton, Mary. (1988). Public Domain Graphing Tools. *The Computing Teacher*, *16* (2).

Bill Craig
4111 Forest Hill Ave.
Richmond, VA 23225

Call for Presentations for the

Great Lakes/East Coast Logo Conference

Sponsored by: The Educational Computer Consortium of Ohio (ECCO)
Where: Cleveland, Ohio
When: May 4 - 5th, 1990.
(Preconference workshops on May 3.)

Categories of presentations:

Poster Presentations:

These sessions will offer you an opportunity to share a particular idea. Three presentors will be scheduled each hour. Presentations should cover specific teaching ideas, lesson plans, or classroom activities. An Apple IIc or IIe computer and appropriate monitor will be made available during the session.

Session Presentations:

These sessions will be approximately an hour long. Presentations should cover classroom ideas, research project reports, innovative uses of Logo, and Logo connections. An Apple IIc or IIe, overhead projector, and appropriate display screen will be made available.

PreConference Workshops:

These workshops will be six hours in length and will be held on May 3rd. Workshop topics should go beyond beginning turtle graphics and offer participants new ideas and challenges. Either IBM or Apple equipment will be available.

Conference Workshops:

These workshops will be held during the conference and will be approximately 3 hours in length. Workshops can cover either beginning or advanced ideas, but should be limited to material that can be meaningfully presented in a hands-on format in the allotted time. Either IBM or Apple equipment available.

Presentation Proposals are due November 15th, 1989

Contact ECCO for forms to use to submit presentation ideas.

ECCO
1123 S.O.M. Center Road
Cleveland, OH 44124
216-461-0800

Logo & Company

Hypermedia Castle

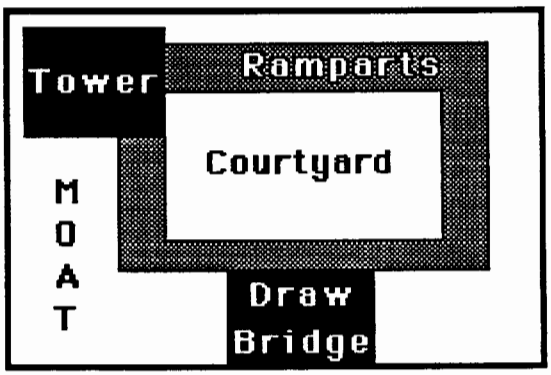
by Glen L. Bull and Gina L. Bull

Hypermedia is a term which is appearing with increasing frequency in educational computing. *Hypermedia* refers to the ability to go directly from any point in a medium to any other place in the medium. If this column were a hypermedia application, you would be able to touch the word "hypermedia" to see an expanded definition and more examples.

Hypercard is the best-known example of a hypermedia program, but it is possible to develop certain types of hypermedia applications in Logo. In previous issues of the *Logo Exchange*, Eadie Adamson has described hypertext applications that she has developed for *LogoWriter*. *Hypertext* is one type of hypermedia application that involves the ability to go directly from one place in a body of *text* to another location.

Hypermedia applications can involve graphics as well as text. For example, in a hypermedia application it might be possible to touch Texas in order to see an expanded view of that state. How is this done? In Hypercard this could be done with the electronic equivalent of two index cards. A map of the United States might be on one card, and a map of Texas on the other card. Touching Texas on map of the United States would cause the computer to shift from the United States card to the Texas card.

The GETPAGE command in *LogoWriter* can be used in much the same way. The terminology is different. Hypercard uses the descriptor *card* while *LogoWriter* uses the term *page*, but the effect is similar. In the following example, a Logo castle will be used to develop a sample hypermedia application. In this example, the turtle can travel from place to place in the castle. Typing the proper command will cause the turtle to travel to an expanded view of its current location. Longtime readers of the *Logo Exchange* will recognize this castle from an earlier column dating from the days in which the magazine was printed on a dot matrix printer.



The turtle needs some means of identifying its location in the castle. In Hypercard this could be accomplished by placing an invisible box around each area of the castle. In *LogoWriter* we can accomplish the same result by placing an invisible grid across the entire screen.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	29
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

This grid will be invisible in the completed design. Drawing the grid on the screen during creation of the castle will facilitate the design process. The following STARTUP procedure automates the process of drawing a grid. The version shown is for *LogoWriter* on the Apple II. If you are using *LogoWriter* on an IBM, substitute 320 for **ScreenWidth** and 190 for **ScreenHeight** in STARTUP.

Logo & Company--continued

```
TO STARTUP
MAKE "ScreenWidth 280
MAKE "ScreenHeight 180
MAKE "Rows 10
MAKE "Cols 10
MAKE "BlockWidth
    :ScreenWidth / :Cols
MAKE "BlockHeight
    :ScreenHeight / :Rows
MAKE "HorizontalOffset
    (:ScreenWidth / 2) - 1
MAKE "VerticalOffset
    (:ScreenHeight / 2) - 1
END
```

The **DrawGrid** procedure uses the utility programs **PositionTurtle, Line,** and **Over**. Type **DrawGrid** to see a grid drawn across the screen. You can use the lines drawn as boundaries for your design. This grid program draws 100 blocks on the screen, numbered 0 through 99, as shown in the diagram above.

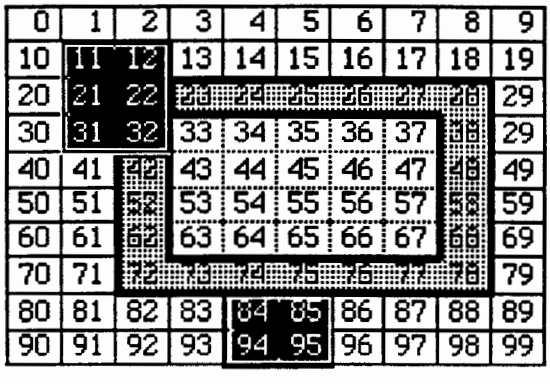
```
TO DrawGrid
STARTUP
PositionTurtle
REPEAT :Cols [Line :ScreenHeight
    Over :BlockWidth]
PositionTurtle
RIGHT 90
REPEAT :Rows [Line :ScreenWidth
    Over :BlockHeight]
LEFT 90
END

TO PositionTurtle
PU
SETPOS LIST :HorizontalOffset * -1
   :VerticalOffset * -1
PD
END

TO Line :Length
FORWARD :Length
BACK :Length
END

TO Over :Distance
PU
RIGHT 90
FORWARD :Distance
PD
LEFT 90
END
```

The **Block?** procedure allows us to see the location of the turtle on the screen. This procedure works whether or not the lines are actually drawn on the screen. Try moving the turtle around the screen. Type **SHOW Block?** to see which block the turtle is in.

```
TO Block?
MAKE "X INT (XCOR +
    :HorizontalOffset) / :BlockWidth
MAKE "Y INT (YCOR +
    :VerticalOffset) / :BlockHeight
MAKE "Y (:Rows - 1) - :Y
OUTPUT WORD :Y :X
END
```

Once we are able to tell the block on the grid on which the turtle is located, it will be possible to determine whether the turtle is in the tower or the drawbridge or the moat. First create procedures which list the blocks associated with each location. For example, by examining the diagram carefully, you will see that Blocks 11, 21, 31, 12, 22, and 32 are located in the tower of the castle. Create a procedure that looks like this.

```
TO Tower
OUTPUT [11 21 31 12 22 32]
END
```



Then create similar procedures for the courtyard, the ramparts, the moat, and the drawbridge. For example, the drawbridge procedure would look like this.

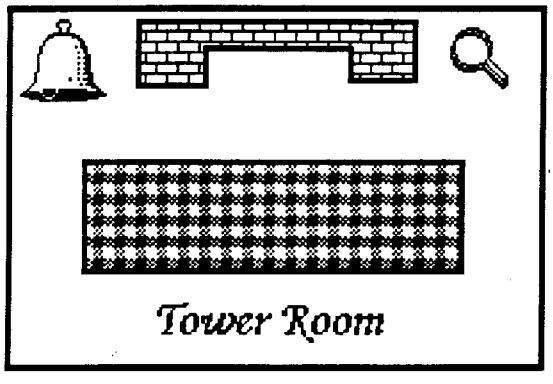
```
TO Drawbridge
OUTPUT [84 85 94 95]
END
```

These procedures will tell where the turtle is, through use of the **Where?** procedure. To find where the turtle is in the

castle, type **SHOW Where?** They will tell where tell where the turtle is even if it is hidden. You may want to determine how accurately you can navigate from one location to another with the turtle hidden. You can use the Where? command to check your progress as you move forward.

```
TO Where?
MAKE "Location Block?
IF MEMBER? :Location Tower
   [OUTPUT "Tower]
IF MEMBER? :Location Drawbridge
   [OUTPUT "Bridge]
IF MEMBER? :Location Courtyard
   [OUTPUT "Yard]
IF MEMBER? :Location Ramparts
   [OUTPUT "Ramparts]
IF MEMBER? :Location Moat
   [OUTPUT "Moat]
OUTPUT "Nowhere
END
```

The **Where?** procedure is the heart of the hypermedia component of the program. In this example, using **Zoom** procedure given below will cause the turtle to go to another page in *LogoWriter* which shows an expanded view of the room or location. To try this concept, first create another *LogoWriter* page with an expanded drawing of one of the castle structures. For example, the second *LogoWriter* page might show what is in the tower of the castle. It is possible that the tower might contain a rug and a fireplace, with a bell on one side of the fireplace and a magnifying glass on the other.



The following procedure will take us from the castle to the tower room. This presumes, of course, that you have created a page in *LogoWriter* named "Tower", and that the turtle is in the tower area on the castle page.

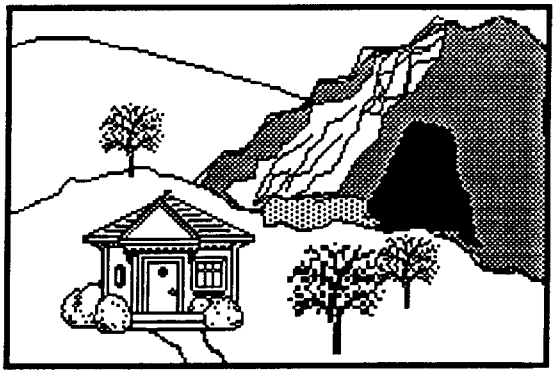
```
TO Zoom
IF NOT MEMBER? Where? PAGELIST [SHOW
   SENTENCE [There is no expanded
   drawing of] Where? STOP]
GETPAGE Where?
END
```

You will need a procedure to return to the castle from the tower room. The following program on the procedure side of the tower room page will ensure that you can return by typing "Castle". You will need an identical **Castle** procedure in each of the other *LogoWriter* pages (Yard, Bridge, Moat, etc.) that you create.

```
TO Castle
GETPAGE "Castle
END
```

The use of the **Where?** procedure is not limited to the expanded drawings of each of the Castle areas. The **Zoom** procedure is given as a model that can be used as the basis for similar procedures. For example, in addition to the expanded drawings that show additional details of what is inside each area, we also created other *LogoWriter* pages that show the view that can be seen from those areas.

For instance, the tower view below shows what can be seen looking out from a window in the Tower. There is a small cottage, mountains, and a cave in the distance. Since the name "**Tower**" was already used for a *LogoWriter* page, we used the name "**TowerV**" (the "V" stands for "View") for the *LogoWriter* page showing the view from the tower window.



The **View** procedure that shifts to a view from the area where the turtle is placed is very similar to the **Zoom** procedure. The only difference is that a "V" (for "View") is added

Logo & Company--continued

to the end of the page name. This concept can be used as a model for hypermedia procedures such as an X-Ray procedure that looks for hidden objects, and many similar variants of this idea.

```
TO View
IF NOT MEMBER? Where? PAGELIST [SHOW
   SENTENCE [There is no view from
   the] Where? STOP]
GETPAGE WORD Where? "V
END
```

The images created can be drawn with Logo, or developed with external paint programs and brought into *LogoWriter* with the **Loadpic** command. An increasing library of ready made clip art is also becoming available for this type of application.

This type of *LogoWriter* hypermedia application can be utilized in a variety of ways. Instead of a castle, the floor plan of a building or an enchanted garden could be used. In science class the solar system could be depicted, with individual pages for each of the planets. This approach could also be used to depict scenes from a story in language arts class.

Hypermedia refers to the ability to go directly from one point to another location in a medium. Multimedia refers to use of more than one medium (such as computers and video) at the same time. The hypermedia example described above can be developed with no more than a standard Apple II or IBM computer. We have also used this type of application in conjunction with a videodisc player. In this type of multimedia application, the View command controls a videodisc player. When the View command is used, an image from the videodisc player is displayed on a separate video monitor beside the computer monitor. In this instance, the view from the tower of the castle shows a flowing mountain stream rather than a computer illustration. The hypermedia examples described above can be useful tools in themselves, and also provide a good starting point for work with multimedia applications.

A Note about Versions and Dialects

If you are using Version 1 of *LogoWriter*, you should consider upgrading to Version 2. Version 2 has many powerful commands which are not available in Version 1. However, if you are currently using Version 1, you can still develop the hypermedia examples. Substitute FIRST in place of the INT function in the procedure BLOCK?. You will also need to create the following procedures for XCOR and YCOR.

```
TO XCor
OUTPUT FIRST POS
END

TO YCor
OUTPUT LAST POS
END
```

The concept of pages in *LogoWriter* is a good parallel to the concept of cards in Hypercard. Therefore *LogoWriter* provides a good basis for discussion of the examples listed above. However, you can also develop similar applications in other versions of Logo, allowing for slight dialectical differences. For example, the function MEMBER? is written as MEMBERP in some dialects of Logo such as LCSI Logo II. Check your Logo manual to determine the form used in your dialect of Logo.

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

Letter to the Editor

I received the following comments from Brian Harvey regarding two articles in the September 1989 issue of *LX*. His comments address specific programming issues in each of the articles. —Editor

Although the code in this article was intended to be used to "match" the physical model of the Russian dolls, I think that the version that uses

```
MAKE "WORD BUTLAST :WORD
PRINT.WORD :WORD
```

instead of saying

```
PRINT.WORD BUTLAST :WORD
```

encourages the all-too-common confusion between the name of an input (established in the title line of a procedure definition) and the value of the input (established in an instruction that invokes the procedure). The article makes it look as if the expression that provides the argument value has to be the same (:WORD) as the name in the title line. Also, the MAKE encourages the learner to think about changing the value of the existing WORD variable, whereas we really want to focus attention on the fact that there are several variables with the same name. Even using the version without the MAKE, it's common for students to misunderstand by thinking that the recursive call "assigns a new value to the variable." The MAKE makes that misunderstanding more likely. (My version produces different results from the one in the article. Mine is symmetrical, with DREAMS as the last line, equal to the first line printed. I think this is better, too, because it makes the point that on return from the subprocedure, the caller's variables are unchanged.)

In addition "the MAKE statement had changed the value of the global variable :WORD" is wrong. It's not a global variable; if it were, the procedure wouldn't work at all.

The Crume and Maddux article on page 26 should make it clear that their versions of TEST etc. are not quite like the primitives in other LCSI Logos, for two reasons. First, the real TEST primitive does not take an expression list as input, but rather takes the word TRUE or FALSE. That is, instead of

```
TEST [:A = 5]
```

as in the article, in a regular LCSI Logo and in Terrapin Logo you say

```
TEST :A = 5
```

They could make their tool work like the real primitive by removing the RUN from the definition of TEST on page 27. (The RUNs in the definitions of IFTRUE and IFFALSE should not be removed.) Second, in the real primitive version of TEST, the result of the test is local to the procedure in which TEST was invoked. In their version, the test result is global. That is, suppose you have a procedure

```
TO FOO
TEST such-and-such
IFTRUE [BAZ]
IFFALSE [GARPLY]
END
```

Now suppose that FOO is invoked and the test condition is true. Suppose further that BAZ (which is invoked by the IFTRUE) does its own TEST and IFTRUE/IFFALSE instructions. Then, on return to FOO, the IFFALSE instruction will have the wrong effect (invoking GARPLY) if BAZ's test came out false. This can't be fixed without making TEST a real primitive.

Brian Harvey can be reached at
bh%anarres.Berkeley.EDU@berkeley.edu

MathWorlds

Two Turtles in a Hot Tub: Part Three
Parallels between Logo primitives and mathematical definitions.

Through a series of email messages initially between Tom Kieren and myself, but later joined by David Pimm, I suggested to Tom and David that they should sit down one day and tape record their conversation about the relationship between mathematics and Logo. This they did one evening whilst reclining in Tom's hot tub in Edmonton. They mailed the tape to me, and I had it transcribed. I did some editing on it, and then emailed the result to David in Britain and Tom in Edmonton. Within hours David returned a corrected copy to me by email. Tom's revisions came a few days later. Because of the limitations of space for any one column in *LX*, we have divided the conversation between David and Tom into three sections, the first of which appeared in the September 1989 Issue of *LX* and the second of which appeared in October *LX* and the third of which appears below.

(DP) For a new starting point, let's work on the parallel between primitives in Logo and definitions in mathematics.

(TK) Yes, you were arguing that, or pointing out the problem of packing a lot of mathematical information into a primitive. In mathematics as well there seems to be a trend toward packing a lot of power into the definitions such that the consequences flowing from the definition become trivial. We would like to look at the opposite side of that same coin and try to think about powerful primitives: what are their positive consequences and what happens when you try to unpack them?

For example, one of the first primitives I had kids playing around with was something we found in *Byte* magazine. It was called "Squiral." It made interesting spiral shapes. Readers are probably familiar with it. The turtle simply went forward a bit and turned an angle, went forward a bit further and turned the same angle, went forward a bit further, and etc. Although this seemed like a fairly harmless activity, the primitive carried with it three parameters which kids could play with. Because it had a lot of power, the kids came to see that there was much, much more to shape then they might of thought of previously—shape in a controlled sense, not shape in some random, non-replicable drawing sense. I think that is one of the nice things of having super powerful primitives. The kids really like the fact they could get star shapes and spirals and shapes that seemed to be partly three-sided and partly four-sided, depending on how you looked at them: all kinds of interesting nuances you probably would not get by drawing things by hand. So again, the primitive provided some interesting mathematical power to the students.

(DP) One of the additional factors is that all these different shapes came from the same primitive, thereby suggesting that the shapes actually have something in common whereas you may otherwise have classified them as totally different, coming across them at just the visual level. The fact the shapes are all generated by the same primitive can suggest that you look for the commonalities across a wide range of different phenomena.

(TK) Again, we have the "order thing," don't we? If we think of the primitive as a definition, the definition comes first, and the consequences of the definition come afterwards. This is opposed to our normal play in mathematics which is to do a bunch of things and then see if we can write a definition for it. If you're given one of these powerful primitives, the definition comes first. That's an interesting sequential thing, and it may have some positive consequences.

Again, if you think of the primitive as a definition one can start to do things with a primitive; you can go to edit mode and look and see what the primitive is saying...

(DP) ...which is a nice reversal and suggests the possibility of perhaps not seeing primitives as definitions, but seeing definitions as primitives. Nonetheless, you can look at a definition and try to reconstruct to see how the various elements of it actually came about.

(TK) I think that is a nice point, but let me just explore the distinctions between a definition as it is normally written, let us say, in a mathematical textbook, and a primitive as it might be written in Logo. Let me take the instance of the 11 and 12 year olds I worked with on Squiral as a way of exploring this. The interesting thing seems to me was that some of the students, not all, were interested in looking to see how the "primitive" worked. They were very bothered by the fact that the turtle always ended up the Squiral out some place on the screen in a non-predictable location. They were wishing to use some of these objects in their drawings, and could never figure out where the turtle was going to be. So their first attempt at it was to try to get the turtle to end up in the center instead of the turtle ending up on the edge. They seemed to think, "Perhaps we can find out how the darn thing works." In that sense, I think, they were looking at the primitive as a definition.

The interesting alternative effect was that they then changed some of the lines in this "definition." Unlike an ordinary definition they could now see what happened. They could, in fact, have the computer execute the definition, if you

will, and see the consequences. This is probably not very different from what a mathematician might do, but probably very different from what a child might do. So maybe Papert is on to something: there is mathematics buried in Logo, in the Logo activity, regardless of whatever the objects you are looking at and working with are mathematical or not.

(DP) I'm also remembering an earlier article in *Logo Exchange* written by Uri Leron called "On the Mathematical Nature of Turtle Programing" (Leron, 1987). There he draws some similarities between the mathematical definitions of a rectangle and the Logo prescription of a rectangle, and talks about one of the differences in terms of it being an *active* — describing what you have to *do* in order to generate one — rather then a *passive* definition, which a lot of mathematical definitions tend to be. And that remains me of a "history of math discussion": that a lot of early definitions, particularly in geometry, have been described as definitions by genesis; which is, you say what you need to do in order to be able to generate the curve. Some of the definitions, for example, as I think of a circle in Euclid, are by genesis. This style of defining mathematical objects is very useful for doing — for having a strong sense of the object — but is less useful if you actually want to prove the results. The alternative conception which seem to come about, particularly with Appolonius' work on conics, has been called definitions by "property," where you try to come up with some characteristic property, particularly in the context of curves, and you specify the mathematical curve by the fact it satisfies this property. In fact one of the earliest examples, in seventeenth century, of differential equations are set up precisely not as a curve that has this property, but as a curve whose tangents have this particular property, which is one stage further removed. So it seems to me that the primitive, to the extent that Logo is associated with action, will be emphasizing a definition by genesis approach to mathematical objects. At the same time, I can see how a mathematician working in Logo will be able to implement the properties that they know — rectangles or whatever mathematical shapes satisfy — in order to provide a more active definition of it.

(TK) There are two things in your comment that strike me. One is the general notion that mathematics in its primitive state is a sort of engineering report, a "how to do it" report rather than anything else. I think in that case, Logo provides a fairly ready language for making quite nice mathematical engineering reports. We were talking yesterday about growth or movement between levels, and it seems to me that there is a kind of parallel in the growth and use of Logo in the movement from procedures to families of procedures, and then to recognizing that procedures themselves are useful objects of study. Then we go to something that might be called the structured procedure, or top down procedure, which is much closer to definition by property as opposed to definition by action. If you were to look at some more advanced Logo procedures, they are no longer in any way tied to action. You can't see the particular action by looking at a particular line, or even a particular sub-procedure in the Logo procedure. So I think that there is a parallel again between the move towards this definition by property away from definition by prescription. I think that's an interesting thing.

Minsky talks about using language as a controlling factor (Minsky, 1985). As one moves from these procedures, which define action by a very controlled, orientated use of language, to definition by structure, it is less obvious what the control is. The control is certainly not directly of the turtle; it's clearly control of your thoughts, as opposed to the control of the turtle.

(DP) Right, the turtle then loses the focus of your attention as being the thing that you're controlling. That's always been something I have difficulty sorting out in my own mind when I come to work with Logo; namely, to what extent am I automating mathematical concepts and understanding that I have had in an dynamic active way, as opposed to using Logo to explore mathematical ideas that I haven't actually either had or gained control of, so that in someways I am building something new.

Hence, it seems to me that there are these two different ways you can use Logo in connection with mathematics. As young children work through Logo they're are obviously learning mathematics outside the Logo context as well, and I'm quite interested in the interrelation between these two, sort of parallel, developments that make links at certain points.

(TK) I think that that is a very interesting kind of point and again I think it reflects on our previous discussion about powerful primitives. One of the virtues of powerful primitives is that it puts you into a mathematical environment; then you can start to explore areas that you wouldn't have even anticipated existed. The Squiral thing was a very, very good example of that. Two of the 11 and 12 year olds that I was working with came to look at Squirals where the angle was around 180, because they had made these beautiful wing shaped objects, and then they were trying to make wing shaped objects with two wings, and they found that if they went just over 180, and just under 180, they got the things that they wanted. But they also came up with the theorem that if you have 180 plus some amount, and 180 minus some amount, you get the same figure, except the opposite, so that the primitive and the use of number gave them a kind of a different handle on shape and I think gave them, pushed them if you will, to start looking at the properties rather than the result of

the properties. Clearly, the result of the properties is what peaked their interest; that is, they got these beautiful wing shapes that they were looking for. But then they were really struck by "does this work even if we aren't making wings," and I think that that's an interesting kind of thing with powerful primitives; it does lead you into mathematics.

(DP) That's right, and I think that's a nice example of trying to distinguish between mathematical thought and, if you like, any other kind of thought, including programming thought or computing thought. In mathematics, if you're attending to the processes directly, as opposed to the objects you're focusing on, I think that's one of the characteristics of explicitly mathematical thought. So for me they might be using what Vergnaud is calling, "theorems in action" when they are concentrating on implementing what they want to do on the screen (Vergnaud, 1981). But the move that you describe so well when the pulling away from the particular implementation, as saying "is this more generally true, is the complimentary of over 180 matched by a under 180," then I think they're moving much more directly into straight mathematical exploration and away from the screen. Another thing that struck me in what you were saying was how rare it is in a mathematics class for pupils to know what they want. You gave a very clear description of how they wanted to do something on the screen, then went about pulling particular tools and knowledge towards achieving that end. In a traditional mathematics class if you ask pupils what do they want, or make a developmental, what's the phase. There's a notion of developmental reading where you come into the classroom and one of the things you immediately ask the pupils is, "what do you want to get better at." And if children got the notion of developmental mathematics, I suspect if you went into a class and ask, "what to do you want to get better at," you wouldn't get particularly strong or coherent answers. It seems to be one of the strengths of Logo, related to this implementability of action, is that pupils can either generate for themselves, or see on other pupils screens, or be promoted by a teacher offering something, to want something; and therefore provide some goal to which they are willing to work drawing whatever resources they can.

(TK) I suspect that this is the more general point made by Noss and Hoyle when they talk about their UDGS model at least the "use- discrimination-generalizing sequence," seems to be three levels of "what do you want." One is, what do you want now; the second is, can I make this different, doing pretty much what I'm doing now; third, is this really part of a much larger pattern that one might do in general. You have both the language and the consequence of the language rather closely tied, at least at one level of Logo use.

References

Ainley, J. & Goldstein, R. (1988). *Making Logo Work.* Oxford: Basil Blackwell.

Balacheff, N. (1988). Aspects of proof processes in pupils' practice of school mathematics. In D. Pimm (ed.) *Mathematics, teachers and children.* London: Hodden & Stoughton .

Hillel, J. (1987). Multiple perspectives and task-analyses of a Logo activity. In J. Hillel (ed.) *Proceedings of the Third International Conference for Logo and Mathematics Educatio*n. Montreal: Concordia University.

Kieran, C., Hillel, J. & Erlwanger, S. (1986). Perceptual and analytical schemas in solving structured turtle-geometry tasks. In C. Hoyles & R. Noss (eds.) *Proceedings of the Second International Conference for Logo and Mathematics Education.* London: University of London School of Education.

Kieran, C., Hillel, J. & Gurtner, J. L. (1987). Qualitative strategies in Logo centering tasks. In J. Hillel (ed.) *Proceedings of the Third International Conference for Logo and Mathematics Education.* Montreal: Concordia University.

Leron, U. (1987). On the mathematical nature of turtle programming. *Logo Exchange, 5* (9).

Ludwig, S. (1986). *A Logo/motion geometry curriculum environment.* Masters Thesis. Edmonton: University of Alberta.

Minsky, M. (1985). *The society of mind.* New York: Simon & Schuster.

Pimm, D. (1987). *Speaking mathematically.* London: Routledge and Kegan Paul.

Spivak, M. (1965). *Calculus on manifolds.* London: Benjamin.

Stubbs, M. (1974)."Organizing classroom talk. Occasional Paper !9, *Centre for Research in Educational Sciences*, University of Edinburgh.

Vergnaud, G. (1981). Quelques orientations theoriques et methodologiques des recherches francaises en didactique de mathematique. *Proceedings of PME V.* Glenoble.

About the discussants

David Pimm is a lecturer in Mathematics Education at the Open University in Britain. Tom Kieren is professor of Mathematics Education, and Distinguished Research Professor, at the University of Alberta, Canada.

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

Logo: Search and Research

To err is human...

by Douglas H. Clements

How pervasive are errors in Logo learning? How persistent? Heller (in Fay & Mayer, 1988) found that fourth-grade students were 48% correct on a Logo syntax test after 3 weeks of Logo experience. After 12 weeks, performance rose only to 65% correct. How about older students? In one study, students from sixth to eleventh grade also showed errors on essentially every Logo construct. Over 70% had major difficulties. No student had no difficulties (Kuspa & Sleeman, 1985).

Even when students gain control over the turtle's world (see last month's column), they may face substantial difficulties. What are these difficulties? Why do they occur? The next several columns will explore these critical questions. As you read, try to imagine students of yours who show similar misconceptions.

General Types of Errors

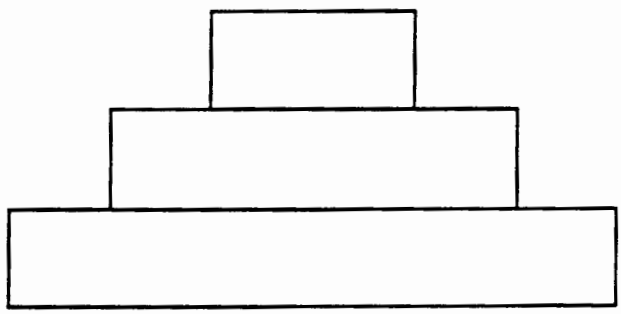
Students make several types of errors. For example:

1. Typing errors
2. Syntactic errors: Incorrect Logo statements that would lead to an error message.
3. Semantic errors: Logo statements that lead to inaccurate or contradictory actions, although each of these statements is syntactically correct.
4. Stylistic errors: Logo code that does not follow standard programming conventions, making the program inefficient or difficult to comprehend and debug.

In this column, we will emphasize syntactic and semantic errors that go beyond the basic (semantic) turtle graphics errors that were discussed in last month's column. Stylistic errors will be discussed in a future column.

Creating and Using Procedures

Why do so many students resist using procedures, especially as subprocedures? Often, young children do not understand the relationship between the instructions they give the turtle and the graphics it creates. Rather, they seem to believe that the turtle remembers a visual image of the graphic picture and calls up that image (Vaidya & Mckeeby, 1984-1985). Even older students identify programming with turtle graphics as "drawing with the turtle." Of course, this is also a strength—children can use their well-established intuitions about drawing to ease their introduction to Logo. The disadvantage, however, is that this hampers procedural analysis of tasks. Writing a program is seen as "tracing over all the lines" rather than putting together "building blocks" (some of which may not yet be defined) (Hillel, 1985). This may lead to the common approach of constructing "stacked rectangles" as described by a fourth grader: "I drew the outside and then filled in the lines." (The previous column "Planning for Planning" contains teaching ideas pertinent to this problem.)



Even when the power of subprocedures is illustrated, students often believe that—as one explained—"It's easier to do it the hard way." This often occurs because students relate a turtle geometry procedure with a specific output (including its initial position and orientation). They view it as a fixed product rather than a dynamic and flexible process. For example, one boy constructed a rectangle procedure (beginning at a corner), then decided he wanted it to be centered on the screen. He believed that he would need to redefine the procedure rather than merely change the initial turtle state (Hillel & Samurçay, 1985). So students need to be guided to look closely at turtle state changes and at interfaces.

Students also have problems using and interpreting procedures (43% in Kuspa & Sleeman, 1985). These students were given the following procedures and procedure call and were asked to execute them by hand.

```
TO BOX
RT 90 FD 10 RT 90 FD 10 RT 90 FD 10
END

TO SIDES
FD 10 RT 90 FD 30
END

TO SHAPE
SIDES
BOX
SIDES
END

SHAPE
```

Logo: Search & Research--continued

Several students did not know when to execute the procedures. Some executed them as they were encountered (within the procedure's own definition; e.g., BOX SIDES SHAPE). Others executed them as they were encountered and then again when they were called (leading to: BOX SIDES SHAPE SIDES BOX SIDES SHAPE).

Showing another class of errors, some students misunderstood the role of the *name* of the procedures. For example, some executed only "meaningful" figures. These students would not execute SIDES or SHAPE because neither "sides" nor "shape" specifies a complete, specific figure. Others ignored procedures whose name did not match its actions. BOX did not execute because it did not literally draw a box. Finally, the name of procedure often dictated what it would draw. For example, another procedure CIRCLE which called BOX was interpreted as drawing a circle.

MAKE and Variables

Problems with assignment statements (e.g., MAKE) occurred in up to 71% of students (Kuspa & Sleeman, 1985). Many were confused by statements such as MAKE "A :B. Some would ignore the statement (A's value was left unchanged). Others transferred the literal character ("B") instead of B's value. Some thought that MAKE "A 3 would print A three times.

Programming experience in another language may hurt more than it helps. Students who previously learned BASIC prefer the MAKE command to other devices such as parameter passing, often leading to inefficient procedures (Lee & Lehrer, 1988).

In a previous column, we saw that students frequently misapply analogies. For example, students are often taught to think of variables as a box. However, many students then believe that, like boxes, variables can hold more than one value (du Boulay, 1986). Similarly, Logo encourages meaning names for variables. However, using meaningful names sometimes leads students to believe the names are meaningful to the computer. So, in naming both procedures and variables, it may be wise to begin with meaningful names, use nonsense names as an exercise, and finally return to meaningful or abstract naming schemes.

REPEAT Statement

REPEAT statements are another source of difficulties for many (57% of students in Kuspa & Sleeman, 1985). Errors include:

1. Ignoring a turn command when it is the last command in the REPEAT statement.
2. Adding an extra turn command (e.g., RT 90) to the end of a REPEAT statement to make it draw something "meaningful" (e.g., a square).
3. Believing that commands adjacent to the bracket should be executed during the first pass of REPEAT but not at any subsequent pass. Given REPEAT 2 [A1 A2] A3, they would execute the commands as A1 A2 A3 A1 A2.
4. Similar, but executing the adjacent command with each pass, yielding: A1 A2 A3 A1 A2 A3.
5. Executing commands sequentially, forward then backward.
6. Allowing the name of a procedure which the REPEAT statement referenced to interfere with the execution of the command. Given

```
TO TRIANGLE
FD 2 RT 90 FD 3
END
```

and the procedure call

```
REPEAT 2 [TRIANGLE]
```

the students drew a triangle.

IF/THEN Statements and Logical Operators

The IF statement also causes problems (in 21% of students in Kuspa & Sleeman, 1985). Examples:

1. When the IF predicate was true, executing *both* lists (LCSI syntax); when false, executing neither.
2. Ignoring the second list.
3. Not knowing what to do if the predicate was false.
4. Omitting ELSE. Because they could omit ELSE if it were not needed, many adults learning Terrapin Logo tended to omit it even when it would be helpful (Lee & Lehrer, 1988). Such findings may support the use of IF *and* IFELSE as separate statements in recent versions of Logo, or the insistence on using both lists (LCSI syntax) in older versions whether or not they are absolutely needed.
5. Having problems with logical operators, from avoiding their use (Lee & Lehrer, 1988) to making misguided analogies to natural language use; for example, "and" and "then" often mean "what next" in English (du Boulay, 1986).

Does Changing Instruction Help?

Struggling with many of these ideas is probably unavoidable and may even be beneficial. Nevertheless, might not instruction make this struggles less frustrating and more profitable? Yes. For instance, students in one study couldn't transfer from infix notation used in turtle graphics (e.g., :side + 5) to the prefix notation list processing (e.g., BUTFIRST

:LIST1). When they used prefix notation for everything from the beginning (e.g., sum :side 5), this problem disappeared (Lee & Lehrer, 1988). Similarly, the use of TEST, IFTRUE, and IFFALSE, rather than the traditional IF, reduced working memory demands and led to increased performance.

It is also important to provide students with the right level of detail about what's going on inside the machine. The detailed level of logic gates is not helpful (du Boulay, 1986). What may be useful is a less sophisticated explanation of Logo's internal operations, such as is given by "match box" computers.

Finally, it may help us as teachers to recognize that even "simple" Logo problems are not at all simple. Programming statements and their combination lead to a need for precision and a grasp of complexity not often demanded in other intellectual tasks. We need to develop sound mental models of the content (e.g., turtle graphics) and the control structures (e.g., REPEAT or IF) of Logo programming right from the beginning. We also need to understand what underlies students' errors, a topic to which we shall return next month.

References

du Boulay, B. (1986). Part II: Logo confessions. In R. Lawler, B. d. Boulay, M. Hughes, & H. Macleod (Ed.), *Cognition and computers: Studies in learning* (pp. 81-178). Chichester, England: Ellis Horwood Limited.

Fay, A. L., & Mayer, R. E. (1988). Learning LOGO: A cognitive analysis. In R. Mayer (Ed.), *Teaching and learning computer programming: Multiple research perspectives* (pp. 55-74). Hillsdale, NJ: Erlbaum.

Hillel, J. (1985). On Logo squares, triangles, and house. *For the Learning of Mathematics, 5*, 38-45.

Hillel, J., & Samurçay, R. (1985). *Analysis of a Logo environment for learning the concept of procedures with variable.* Unpublished manuscript, Concordia University, Montreal.

Kuspa, L., & Sleeman, D. (1985). *Novice Logo errors.* Unpublished manuscript, Stanford University, Stanford, CA.

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

Vaidya, S., & Mckeeby, J. (1984-1985). Conceptual problems encountered by children while learning Logo. *Journal of Educational technology Systems, 13*, 33-39.

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

Eurologo '89 Report

by Ken Johnson

I spent Tuesday to Friday, 29 August to 1 September, at the European Logo Conference at the State University, Gent, Brussels.

This conference was bigger than the last conference (1987, in Dublin) with 110 delegates, and very well financed since one of the main Belgian banks (ASLK) had agreed to sponsor it. There were delegates from all over Europe: not just the EEC, but also from Bulgaria and Hungary. There had been enquiries from the Soviet Union but apparently no potential delegates from the USSR had been able to obtain an exit visa. There was also a small knot of American visitors from LCSI and from Lego Inc, and a group of Canadians.

There were a number of theoretical presentations from the EEC countries other than Britain, with the British-led sessions having a more practical bias. This, I think, is due to the sudden need for training in the Logo language which arises out of the National Curriculum in mathematics and in design technology. (Unfortunately, the schools do not have the resources to pay for the training they need.)

A couple of new software products were on show. LCSI has pushed the boat out on *Logo Writer* and held a number of sessions demonstrating (American) English, French, and Flemish versions; the Dutch Logo Centre in Nijmegen brought an improved version of LCN Logo which has a structure editor and an unusual implementation of procedures. There was also a Dutch product (in English) called *Logonaut*, but I never found anyone free to demonstrate it. I have been promised a review copy though. So there are still people around developing Logo software despite the scarcity of funds in education everywhere.

I didn't see any new hardware at all. No new computers nor floor turtles!

Other things: The Bulgarians have adopted Logo for their own National Curriculum wholesale and weredisplaying and discussing their set of school mathematics textbooks. Unfortunately these are in Bulgarian and of value only to real enthusiasts! Richard Noss and Celia Hoyles gave an account of recent work at the University of London Institute of Education.

Edinburgh's 'Nimbus Logo' has clearly attracted quite a bit of attention since the Dublin conference when nobody had heard of it, and I spent quite a bit of time trying to sort out misconceptions and programming problems of Nimbus users.

Gent is an excellent conference site, though it must be a fairly expensive place to live in. The next European Logo Conference will probably be in Parma, Italy (near Venice) in 1991.

Ken can be reached through BITNET at
KEN@AIAI.ED.AC.UK

How to increase your Logo Power

Whether you're a Logo teacher, trainer, or enthusiast, you know that this powerful computer language has the potential to have a significant impact on how teachers teach and how students learn. ISTE's Special Interest Group for Logo Educators (SIGLogo) offers you a forum for the exchange of ideas, concepts, and techniques.

What is SIGLogo? SIGLogo is a professional organization that helps Logo Educators get ahead. We sponsor workshops, providing a support community for Logo-using educators. Novice or expert, you will find helpful information in each issue of our journal, *Logo Exchange*.

Satisfaction Guaranteed. Whether you teach Logo or use Logo to teach, SIGLogo and *Logo Exchange* bring you a wealth of ideas from top Logo educators throughout the world, providing you with current information on Logo research, resources, and methods. We're your personal window on professional Logo activities.

Join SIGLogo Today! As a member of SIGLogo, you will receive the *Logo Exchange* journal nine times per year. SIGLogo members are invited to participate in local, regional, and national meetings and to contribute to the flow of ideas through *Logo Exchange*. *Logo Exchange* is published monthly except for June, July, and August. SIGLogo membership is $25 for ISTE members, $30 for non-members. Add an additional $5 for non-U.S. SIGLogo membership.

The International Society for Technology in Education (ISTE) is the leading U.S. and international professional organization for computer educators. It is non-profit, supported by more than 60 organizations of computer using educators worldwide.

ISTE/SIGLogo, University of Oregon
1787 Agate Street, Eugene, OR 97403-9905
503/686-4414