



DR. LOGO

Here's an antidote for
your computer illiteracy

DAVID COLLOPY, SENIOR EDITOR

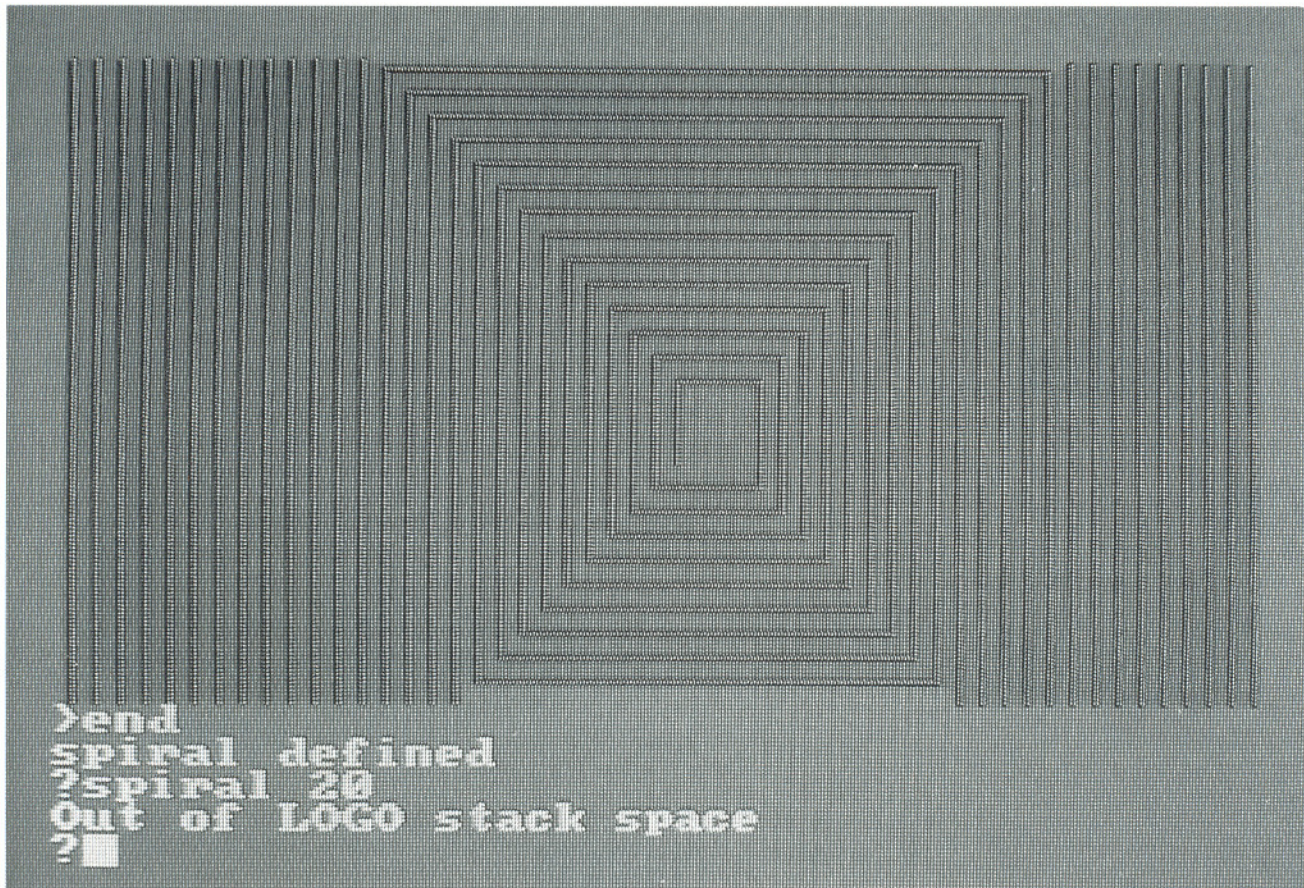


Business Computing
Vol. II, No. 2
L18-01RBC



Digital Research, Inc.
800-227-1617, x400 Nationally
800-772-3545, x400 California

AFTER HOURS



Taking advantage of Dr. Logo's turtle graphics, you'll be creating interesting designs only minutes after beginning to use the language.

DR. LOGO

Here's an antidote for your computer illiteracy.

BY DAVID COLLOPY,
SENIOR EDITOR

Dinner table conversation can sometimes be as challenging as a formal invitation to prepare a speech for the board of directors. Beyond simple amenities like, "How was your day?" (to which you always respond crisply, "Fine"), you're confronted by any number of queries that require the benefit of your vast wisdom.

Someplace between soup and salad, your youngest blurts out, "This big kid Johnny—in the fourth grade—said there ain't no Santa Claus. Is that true?" Worse still, shortly after dessert, your teenager demands, "How old were *you* when *you* started dating?"

But what really gets to you these days is when they want to know about that personal computer you're using at work. "Well, what do you do with it? How does it work? Can you make it talk to the com-

puter at our school? Can you program it to send a rocket to the moon?"

Before they get a chance to ask even more complicated questions, you quickly respond. "Well, today I used a spreadsheet program that helped me get a handle on next year's budget. First I typed in a history of the amount we've been spending the past five years, and then I figured in the rate of inflation for materials as well as the increase in shipping costs. Later I used a word processing program to type the second draft of the Fletcher Foods report. Of course, that was after I updated the files on the customer data base with information that our field reps called in."

You're well aware that the canned speech you're making is being delivered to the wrong audience but, nevertheless, you carry on. After all, you honestly don't know how this machine performs its miracles and you know even less about how its programmers tamed it in

the first place. Besides, reeling off your speech is playing it safe. Even more complicated questions may be lurking behind the salt and pepper shakers. This is especially likely these days since your children probably get more exposure to microcomputers at school than you do at work. (See "Even an adult can do it.")

Maybe it's not only the kid's questions that get to you. Perhaps you're curious about how the computer works its wonders. In either case, you may be well advised to get in touch with Dr. Logo.

A concept is a concept

Dr. Logo is the Digital Research (Pacific Grove, Calif.) version of the increasingly popular Logo computer language. This language was first developed as an easy-to-use introduction to the art of computer programming. The earliest offerings were designed with children (and other first-time computer users) in mind, but the tide seems to have turned. Lately, the Logo language has entered the business environment. One reason for the business community's interest in Dr. Logo is its ease of use.

As the personal computer continues

(continues on page 81)

(After Hours from page 78)

to proliferate in the work place, there is a parallel increase in the number of end users (the term the software industry uses for people who use applications such as spreadsheet analysis and word processing without ever actually developing their own programs).

For the most part, these computer users are satisfied with packaged software and have little need or inclination to develop programs of their own. Besides, the development of software products as sophisticated as most of those commercially available requires more than simply understanding programming languages and concepts. Include an investment called time—usually no fewer than three or four years.

On the other hand, the same user that is satisfied with those products may find an occasional need to develop simple programs, ones outside the realm of those commercially available. Even more likely, this user may simply want to understand the process that makes commercial software tick. The catalyst here is nothing more than curiosity: wanting to learn more about computers and why half the world seems to be programming them.

Knowing how to use Dr. Logo will give you some idea of the conceptualiza-

tion process behind the software products you take for granted. You may never want to tune up your car, but when you understand a little about how the internal combustion engine works, you inevitably develop an entirely new perception of the automobile.

Initiating the illiterate

Essentially, Dr. Logo allows you to become a computer literate non-mechanic who would be comfortable at a cocktail party hosted by the data processing department. As a matter of fact, by the time you've completed the tutorial supplied with the Dr. Logo language, you'll have a clear understanding of the concepts behind programming. Thereafter, it's only a matter of time before you'll be able to take that knowledge and write a practical application.

Dr. Logo encourages you to do top-down structured programming. This entails breaking tasks into little pieces until you have a sufficient number of pieces to do a larger task. By being able to write small procedures that deal with specific functions, you can build on something you've successfully completed.

Beyond simple structuring, Dr. Logo encourages you to use a language with which you're already familiar—English. It allows you to name procedures so that

when you've completed a program, it makes sense. Your program is a string of commands that you've chosen.

And Dr. Logo provides a few features that competitive products do not. It includes a double-precision floating point calculator so that you can incorporate sophisticated mathematic procedures into your programs. There is a split-screen text editor that can be used to make debugging programs simple. And Digital Research integrates lower-case support into its product, whereas the competition uses only capitalization.

In addition, Dr. Logo includes an on-line Help facility—when you're perplexed, you can ask the product for assistance. And finally, Dr. Logo retails for \$99.95, while the IBM version is \$175.

Dr. Logo is by no means the vehicle I'd recommend should you be interested in designing complex, sophisticated software programs. For one thing, Dr. Logo programs operate far less quickly than those developed in other computer languages.

However, if you're interested in understanding more about the mystique beyond the computer keyboard (or you want to reclaim that lost position of authority at the dinner table), Dr. Logo may be just the prescription. ■

*Reprinted with permission from
Business Computing, February 1984*

DR Logo™**PRELIMINARY****DR Logo™ OVERVIEW**

DR Logo is an advanced version of the popular Logo programming language. DR Logo runs under the CP/M® family of operating systems and is designed for the Zilog Z-80® or the Intel 8080 and 8086 family of microprocessors. DR Logo is easy to learn, and yet provides the features to produce sophisticated applications.

DR Logo automatically checks command and statement syntax as you enter each program line. Then, program lines are translated into a condensed, internal format. DR Logo interprets the condensed program lines during execution.

DR Logo is an interactive, procedure-oriented language in which the user can add new words to the language by defining new procedures. Powerful symbolic and list processing capabilities, combined with Turtle Graphics provide a quick and easy method for creating high-quality graphic images. The DR Logo package includes sophisticated programming tools such as a full-screen editor and an interactive debugger. DR Logo also provides informative error messages and a "Help" facility.

TURTLE GRAPHICS PRIMITIVES

DR Logo's Turtle Graphics support high-resolution, point-relative graphic displays. A computer-controlled turtle appears on the screen and responds to commands that make it move FORWARD and BACK and rotate LEFT or RIGHT. The turtle appears as a triangle to indicate its current position and heading. As the turtle moves, it traces its path. DR Logo includes primitives to query and change the turtle's current state.

DR Logo FEATURES

- Turtle Graphics Primitives
- List Processing Primitives
- Recursion Supported
- Floating Point Mathematics Package
- Sophisticated Debugging Facilities
- Workspace Management Primitives
- Comments and Indentation Supported
- Informative Error Messages and Help Facility
- Game Programming Primitives
- Apple Logo Compatible

LIST PROCESSING PRIMITIVES

DR Logo's powerful list processing primitives simplify text and data manipulation. The basic data object in DR Logo is a word. Words can be combined into lists. DR Logo can manipulate lists of objects: numbers, words, symbols, or even other lists. DR Logo's list processing primitives allow you to add, delete, modify, and move list elements.

RECURSION SUPPORTED

DR Logo supports recursive procedures including tail recursion. Sophisticated garbage collection prevents workspace shortages during the execution of recursive procedures.

FLOATING POINT PACKAGE

DR Logo includes a double-precision floating point package that supports fifteen significant digits. This arithmetic accuracy makes DR Logo suitable for business applications and provides additional precision for sophisticated graphics applications.

SOPHISTICATED DEBUGGING FACILITIES

DR Logo can separate interaction with the interpreter, program output, debugging information, and Turtle Graphics into individual frames on the screen. This allows the user to view the program output and commands simultaneously while debugging. When activated, the TRACE facility displays the name and level of each procedure as it is called. The WATCH facility causes a pause after the execution of each statement. During the pause DR Logo allows the user to interact with the interpreter to check or modify variables or expressions.

WORKSPACE MANAGEMENT PRIMITIVES

DR Logo provides additional primitives to manage the expanded memory in 16-bit personal computers. These primitives can change the logical order of procedures in the workspace, cross-reference procedures, and add or remove comments or help information.

COMMENTS AND INDENTATION

DR Logo's features aid both coding and maintenance of programs. The addition of extensive comments and indentation of source code to indicate a program's structure do not affect the size of an executable program. The NOFORMAT primitive removes all comments from the workspace.

INFORMATIVE ERROR MESSAGES AND HELP FACILITY

DR Logo reports program errors with clear, concise diagnostic messages. Each message indicates precisely what is wrong without complicated terminology. A Help facility is built into the POPRIM primitive. It can display a definition and example for each DR Logo primitive, minimizing the need to reference the manual.

GAME PROGRAMMING PRIMITIVES

DR Logo has two unique game-programming primitives: SPIN, which outputs a random integer between 0 and the input number, and SHUFFLE, which organizes elements of a list into random sequence.

APPLE LOGO COMPATIBLE

DR Logo is compatible with the LCSII implementation for Logo for the Apple Personal Computer. Programs developed under the Apple Logo can execute under DR Logo with minimal changes to the source procedures.

SOFTWARE PERFORMANCE REPORT

DR Logo is supported by Digital Research's Software Performance Report (SPR) system. This service provides a prompt response to technical problems associated with DR Logo. Users are provided with SPR forms which serve as a communications device to inform the Digital Research Product Support staff of user-identified problems.

HARDWARE REQUIREMENTS

8-bit System

- An Intel 8080/8085 or Zilog Z-80 microprocessor.
- Operates with any 8-bit Digital Research operating system including: CP/M, MP/M and CP/NET.
- 64K of available memory is required, not including the operating system.

16-bit System

- An Intel 8086/8088 microprocessor.
- Operates with any 16-bit Digital Research operating system including: CP/M-86, Concurrent CP/M-86 and MP/M-86.
- 128K of available memory is required, not including the operating system.

DIGITAL RESEARCH

Digital Research, Pacific Grove, CA., is the leading producer of microcomputer operating systems, languages and utilities for 8- and 16-bit microcomputers. For eight years, Digital Research has been involved with the design, development and support of microcomputer software. Digital Research's operating systems are the industry standard. Digital Research's languages and productivity tools are designed for the professional programmer writing commercial software packages. Digital Research's CP/M graphics products are the standard interface necessary to bring graphics software to market. Together, they form a family of compatible software products. Digital Research users include over 800,000 systems, 700 OEMs and 600 independent software houses.

Digital Research products and logo are either trademarks or registered trademarks of Digital Research.

Z-80 is a registered trademark of Zilog Corporation.
Apple Logo is a trademark of Apple Inc.
Copyright © 1983 by Digital Research. PB 19

NOTE: The information set forth in this Product Brief is descriptive only. For detailed specifications consult the DRI technical manual for the product.

Printed in the United States.

 **DIGITAL RESEARCH™**

P.O. Box 579
Pacific Grove, CA 93950
408-649-5500
TWX 910 360 5001

Digital Research's DR Logo

Gary Kildall, Digital Research, Inc.
David Thornburg, Innovision



BYTE Magazine
June 1983
Vol. 8, No. 6
ART 113



Digital Research, Inc.
P.O. Box 579
160 Central Avenue
Pacific Grove, CA 93950
408-649-5500

Digital Research's DR Logo

A user-friendly language comes of age.

Gary Kildall
Digital Research Inc.
POB 579
Pacific Grove, CA 93950

David Thornburg
Innovision
POB 1317
Los Altos, CA 94022

Logo for personal computers has been heralded by some as the beginning of a revolution in computer languages that promises to be as far reaching as the introduction of the personal computer itself. Yet many people think that Logo is not much more than a graphics language for children. Adding to this confusion is the fact that some commercial implementations of Logo are weak (somewhat akin to a version of English that contained no adjectives). Because of the confusion surrounding Logo itself, the appearance of a sophisticated version of this language on a professional microcomputer such as the IBM Personal Computer might be expected to raise some eyebrows. The development of a powerful Logo for 16-bit computers such as the IBM PC can change our way of thinking about programming.

In this article we will show what makes Logo truly powerful, what it can be used for, and how Digital Research's DR Logo, with its powerful language, large workspace, and complete program-development environment, sets a new benchmark by which to measure the

properties of useful computer languages.

To help you understand the power of Logo, we'll give you some background about the earlier language LISP. LISP, developed more than 20 years ago by John

McCarthy, is overwhelmingly the language of choice for researchers in the field of artificial intelligence. Unlike many other languages, LISP lets users perform operations on several data types, including numbers, words,

and lists. A list can consist of a collection of words, numbers, or lists themselves. Because the names of LISP primitives or procedures are also words, one can write LISP programs that automatically generate other LISP programs. It is the ability to manipulate this type of data that gives LISP its name (*LISt Processing*).

LISP has been used to explore topics as diverse as image processing, the analysis of natural language, the computer solution of certain types of "intelligence" tests, and theorem proving. More mundane programs in LISP (such as word processors) have also been created. Viewed from any angle, it is a powerhouse of a language.

DR Logo incorporates the list-processing capabilities of LISP with a syntax that can be learned by children. And Logo and LISP share other powerful features, too.

Nodes, Bytes, and Bits

The popularity of personal computers has brought new words into our vocabulary, such as bits and bytes (that describe storage capacity). With the introduction of list-processing languages like Logo, yet another term to describe units of memory has been added—the node. In Logo terminology you operate with a workspace that holds a certain number of nodes. Like bits and bytes, you want as many nodes in your Logo workspace as possible because your programs and data values are built from workspace nodes. Because nodes are themselves made up from a fixed number of bytes for any particular Logo interpreter, the exact number of nodes you have depends upon the brand of your computer and its memory size.

When you write procedures, enter data values, and run Logo programs, you use nodes from your workspace. Some of these nodes are used permanently, perhaps to store a portion of your Logo program. Others are used to keep track of a temporary value, then later discarded when not of any use.

When you run out of nodes, Logo automatically searches your workspace for temporary nodes that have been discarded. This "garbage collection" reclaims nodes so that Logo can continue operating. If no nodes are reclaimed, Logo stops and tells you about the situation so that you can erase some of your permanent procedures or data values.

Eight-bit computers generally store fewer nodes than 16-bit computers because the Logo interpreter and workspace must coexist in the same 64K-byte memory area. Logo for the 8-bit Apple II computer gives you about 2800 nodes to work with.

Sixteen-bit computers, however, let you operate with more nodes because the memory size is not limited to 64K bytes, as long as you're willing to invest in more memory boards. An 8086- or 8088-based computer, such as the IBM

PC, can potentially address up to 1 megabyte of main memory, giving you more than 100,000 nodes.

For the first-time Logo user, a 2800-node workspace is large enough to write simple procedures, work with turtle graphics, and learn the basics of list processing. When you become serious about your Logo programming, your requirements will increase because of the complexity of the procedures you write and the amount of data you want active in your workspace.

Now your local computer salesperson has one more term to confuse you. If the computer doesn't have enough bits and bytes for you, he'll throw in some nodes at no extra charge.

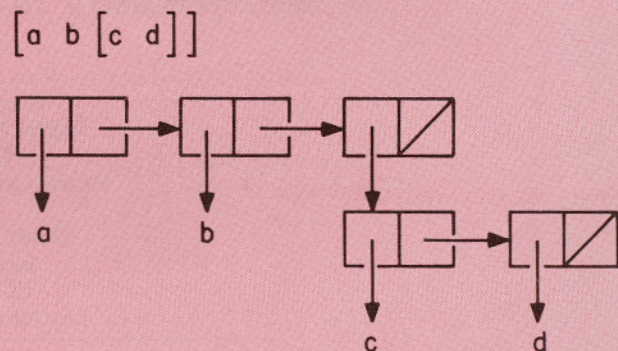


Figure 1: The list containing the two atoms "a" and "b" and the list [c d] is shown above in its node representation. Each node consists of two 2-byte fields that can be used as data or pointers to more data.

DR Logo incorporates the list-processing capabilities of LISP with a syntax that can be learned by children. More than the utility (and beauty and simplicity) of turtle graphics, it is this list-processing capacity that gives it so much power.

Other important characteristics are shared by Logo and LISP. Among these is the ability to extend the language through the creation of procedures that are treated just as if they were part of the language itself. As with some FORTH devotees, many Logo enthusiasts see themselves as not writing programs, but as creating new "words" in Logo tailored to the solution of their particular programming task. While this may appear to be a subtle distinction, it has a tremendous effect on programming style. This style affected the design of Digital Research's Logo in several ways, especially in the debugging and procedure-management tools.

The Power of DR Logo

Before showing what Logo procedures look like, we will list a few of the characteristics of DR Logo. To provide maximum power to the user, we designed the first

implementation of DR Logo for the 16-bit IBM Personal Computer. The use of a 16-bit processor greatly increased the amount of workspace available to the user and also yielded a modest speed improvement over 8-bit versions of the language. A DR Logo user with 192K bytes of RAM (random-access read/write memory) has about 10,000 nodes available for use. (See the text box above.) For comparison, an Apple II user running Apple Logo has only about 2800 free nodes to work with. It goes without saying that sophisticated applications require comparably more workspace than simple ones, and it was important to its designers that DR Logo be able to handle sophisticated applications.

In addition to list processing and turtle graphics primitives, DR Logo can work with integers (30 bits long plus a sign) and both single-precision and double-precision floating-point numbers. A full set of transcendental functions (log, square root, etc.) allows this language to be used for scientific programs as well.

DR Logo is a superset of Apple Logo and more than just a language. A complete programming environment, it includes its own operating system, program editor,

Logo, Turtles, and Kids

Anyone who has watched the personal computer industry for the past few years has probably seen the evolution of certain myths regarding computer languages. Many devotees of BASIC, for example, claim that it is the optimal choice for the home user because of its nearly universal adoption as the default language for personal computers. The fact that BASIC was the only high-level language that was readily available in compact form in the late 1970s is not considered to be relevant by many users. Fortunately, the recent availability of other languages on personal computers (Logo, Pascal, FORTH, and PILOT, to name but a few) has afforded programmers other choices. But some of these languages have myths of their own.

In the case of Logo, the common myth is that it is a turtle graphics language designed to be used exclusively by children. As evidence in support of this myth, one is pointed to Seymour Papert's book *Mindstorms*. It is true that Papert devotes the bulk of his book to the use of turtle graphics as a powerful programming and discovery tool for children, and that he stresses the accessibility of Logo to the young and inexperienced.

The problem with the Logo myth is that it suggests that

Logo is exclusively for children's use. As with many myths, the reality of the situation is quite different. First, it is true that Logo supports turtle graphics. In this regard it is similar to some versions of Pascal, PILOT, and FORTH. Note also that, while turtle graphics is accessible to children, it also has applications of value to advanced programmers as well. Anyone who doubts this would benefit from reading *Turtle Geometry* by Abelson and diSessa or *Discovering Apple Logo* by Thornburg.

The point is that Logo is no more a "kid's" language than is English. Yes, English is the language of "Mary Had a Little Lamb," but it is also the language of Moby Dick and Shakespeare's sonnets.

At its base, Logo is a symbol-manipulation language in the finest sense of the word. Rooted in the artificial-intelligence language LISP, Logo allows the user to extend its vocabulary, to use recursion, and to manipulate various types of data in ways that are nearly impossible with languages like BASIC.

It would be a shame if the myth of Logo kept serious programmers from exploring a language whose foundation goes to the heart of computer science itself.

debugger, and a set of workspace-management tools designed to speed the successful implementation of even the most convoluted artificial-intelligence program.

The graphics system is designed to use either the color monitor alone or to use the color monitor for turtle graphics or mixed text/graphics applications and the monochrome monitor for procedure editing, debugging, and pure text programs. The color display uses the 320-by-200-pixel medium-resolution mode and supports 16 background colors (eight colors that are either bright or dim). It also supports two foreground color sets of four colors each.

A Brief Glimpse at Logo Procedures

Before describing the editor and workspace-management tools, we will examine what a Logo procedure looks like by illustrating the creation and manipulation of a list. A list in Logo is a collection of words, numbers, or lists that are enclosed in square brackets. Each item in the list is separated by a space. For example, [cow horse sheep snake] is a list; so is [1 1 2 3 5 8]. The first list consists of the words cow, horse, sheep, and snake; the second list consists of the first six numbers of the Fibonacci series. A more complex list would be [car [dump truck] airplane [railroad engine]], in which two of the elements are words (car and airplane) and two elements are lists of two words each ([dump truck] and [railroad engine]). Also, a list can have one

word in it ([yellow]) or even be empty ([]).

In common with other computer languages, Logo allows values to be assigned to names. For example, you can assign a list to a name with the make command, e.g.:

```
make "friends [Pam Roy Pat George]
```

The quotation mark is used by Logo to indicate that friends is a word, a variable name in this case, and not a command. If we tell Logo to

```
print :friends
```

we will see

```
Pam Roy Pat George
```

on the screen. The colon in front of friends lets Logo know that we want to see what is bound to the variable rather than the variable name itself. If we had entered

```
print "friends
```

we would have seen

```
friends
```

on the screen instead.

You can take lists apart in Logo with commands such as `first`, `butfirst`, `last`, and `butlast`. For example, if we enter

```
print first :friends
```

the screen will show

```
Pam
```

The command

```
print butfirst :friends
```

prints

```
Roy Pat George
```

Now that we know a little about lists, let's explore Logo's extensibility by creating a new command in the language. Suppose you did a lot of work with lists and you found that you would like to rotate a list by moving its first element to the rear end and pushing everything else up front. We can create a word (e.g., `rotate`) to do this for us. If we had such a procedure, we could make a rotated version of `friends` by entering

```
make "neworder rotate :friends
```

Because Logo doesn't have a primitive called `rotate`, we can create a procedure with this name that looks like the following:

```
to rotate :list
  output sentence butfirst :list first :list
end
```

This procedure accepts a list (denoted by the local variable name `:list`) and makes a new list starting with all but the first word and then appending the first word to the end of the list. The sentence primitive (or native in-

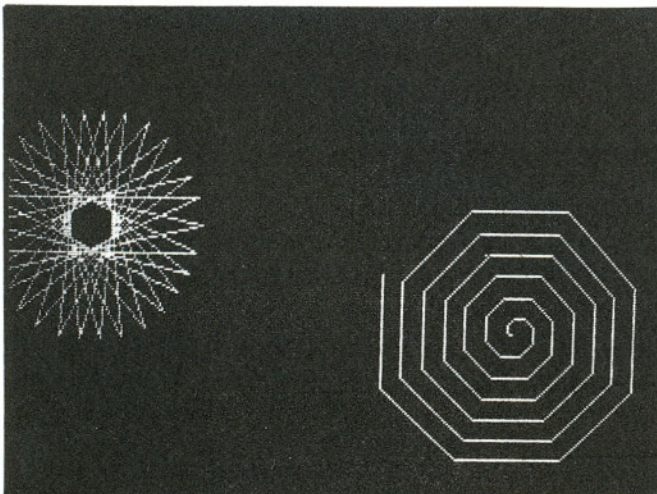


Photo 1: An example of turtle graphics with DR Logo.

struction) is used to assemble a list from two parts. The output command passes the new list back out of the procedure to any procedure that used `rotate`, or to the command level.

Once defined, Logo procedures are treated just as if they were part of the language's native vocabulary. For example, if you were to enter

```
print rotate :friends
```

the list

```
Roy Pat George Pam
```

would appear on the screen.

Logo's ability to manipulate lists by taking them apart, adding to them, examining their contents, and altering their order is central to the use of Logo in the creation of knowledge-based programs. For an excellent introduction to the use of lists in the creation of a knowledge "tree" that "sprouts" new nodes as the program gets "smarter," you should read Harold Abelson's discussion of the program `animals` in his book *Apple Logo*.

In addition to the ability to perform list processing and arithmetic, DR Logo also supports an excellent turtle graphics environment. While much has been written about turtle graphics, especially on its use with children (see the text box on page 214), it is important to understand that turtle graphics is of tremendous value to expert programmers as well. The power of this graphics environment comes through its description of the shape of an object as a series of incremental steps that create it. Once a procedure describing an object has been written, the object can be displayed at any screen location, orientation, and size without having to tamper with the basic description. For example, the procedure

```
to square :size
  repeat 4 [forward :size right 90]
end
```

can be used to create a square at any screen position, angular orientation, or size. To draw a square at a given place, you first instruct the turtle (a cursor that has both position and orientation) to move to a specific *x-y* coordinate and heading (angle). Next you type `square 50`, for instance, to create a square with sides 50 units long. This property of turtle graphics procedures, coupled with Logo's capacity to run recursive programs, has allowed the easy exploration of geometrical shapes and their properties. See photo 1 for an example of turtle graphics.

Programming Tools

DR Logo provides many tools to assist the programmer. While smaller Logo systems can adequately survive with a rudimentary procedure editor, larger Logo environments benefit from some of the extra tools that make program analysis and debugging less tedious. DR Logo's procedure editor allows the use of both uppercase

and lowercase letters for programs and data. Two primitives, uppercase and lowercase, allow the conversion of a word from one case to the other. Also, procedure listings can be indented to make decision branches and nesting easier to see. While not essential to the creation of good programs, such formatted listings are easier to read.

While Logo's syntax generally makes procedures easy to read, it is valuable to have comments appended to certain program lines. This ability is provided in DR Logo, along with the ability to strip these comments from procedures with the `noformat` primitive if more workspace is needed. If the name or syntax of a Logo primitive or editing command is forgotten, online help is available.

Once procedures are created, DR Logo has several primitives that help show how procedures interact with each other. This is especially important for those Logo enthusiasts who experiment with several coexisting versions of procedures before settling on the final choices. Most versions of Logo will print the names of resident procedures on receiving the `pots` command (`print out titles`). If, in DR Logo, you enter `potl`, the workspace will be examined for all top-level procedures (those not called by other procedures) and their names will be displayed on the screen. If you enter `pocall` followed by the name of a procedure, DR Logo will examine the calling structure of the named procedure and print the names of the procedures used by the one mentioned, as well as the names of the procedures used by these secondary procedures, and so on until the calling sequence is complete. This gives a great deal of information on the internal organization of the Logo workspace. If, on the other hand, you enter `porref` followed by a procedure name, all the procedures that reference this name will be found and displayed.

Many Logo programmers create procedures in a haphazard sequence. Because a listing of multiple procedures follows the sequence in which they were entered, large listings can be hard to assimilate. By using the DR Logo follow command, procedures can be resequenced in any order, thus allowing large listings to be more easily scanned.

Once you are ready to try a Logo program, DR Logo provides additional tools to assist in debugging. One of these tools allows the text screen to be split into windows corresponding to the command level, a user I/O (input/output) port, and the debugger (see photo 2). The trace command traces the procedure and displays what is happening and at what level the procedure is relative to the top (command) level. Because a single recursive procedure (that calls a copy of itself) may oscillate through many levels, knowing the level at which an error occurs is helpful when fixing the fault. The command `watch` allows single-step execution of a procedure with the ability to change values and see the effect of each statement. See photo 3 for an example of the watch function.

The use of multiple text windows in debugging is only one application for this powerful tool. The development of good window-management tools can, by itself, in-

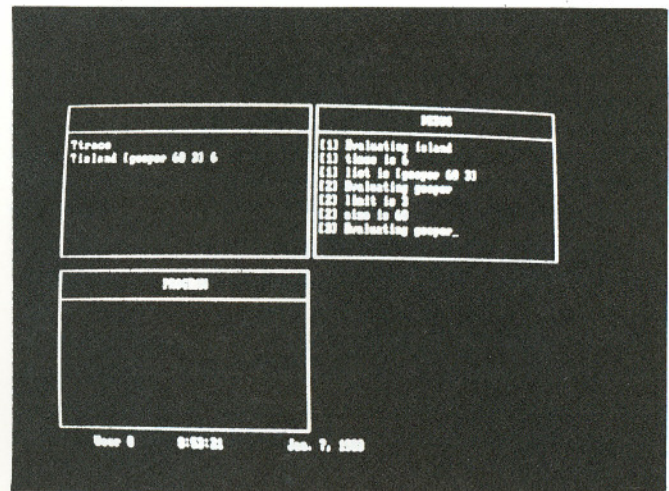


Photo 2: Multiple text windows in the debugging mode, with the trace function turned on. The upper left window is the Logo interpreter where you enter Logo commands. The debug window in the upper right displays information on the current program that is running. Output from the program and input that it requests are handled by the program window at the bottom. The trace function follows the program as it runs, showing the level at which a procedure is called, the name of the procedure, and the values of variables as they are defined. The island procedure being traced has two inputs: a list and a number. This shows up as level 1 in the debug window. The `gospet` subprocedure is called and begins to execute at level 2 with its variables, `size` and `limit`. The `gospet` subprocedure is recursive and calls a copy of itself that begins executing at level 3.

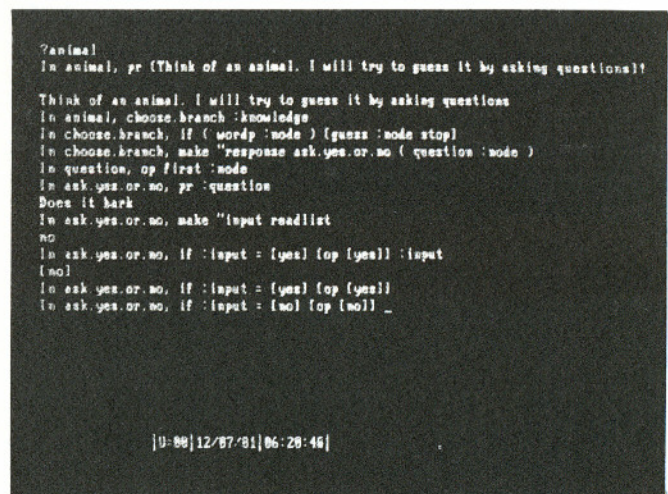


Photo 3: The watch function lets you interact with a program line by line as opposed to the trace function that runs continuously. The animal procedure is being run while the watch function is on. (The question mark on the first line is the Logo prompt.) The name of the current procedure being called is given at the beginning of each line. This is followed by a line from the procedure that is about to be executed. You can hit the return key to execute that line or you can type in a Logo command to display values that the procedure is using. Program input and output occur separately on their own lines.

crease the simplicity, flexibility, and power of this programming environment.

Applying DR Logo in Education

Perhaps because of its historic use as a discovery tool for children (and because of the typically small workspace found with most implementations), Logo is not generally perceived as an applications language. It is anticipated that DR Logo will prove to be an exception in this regard.

The educational applications for Logo have typically focused on the use of turtle graphics. The beauty of turtle graphics is that children simultaneously acquire skills in programming, geometry, and art. Many children who are "turned off" by math have discovered it to be an exciting field through their exploration with turtle graphics. Furthermore, it has been found that once a child uses Logo to discover new ways of thinking about mathematics, this new way of thinking continues to produce beneficial results—even if the child is no longer exposed to Logo.

In the physical sciences, Logo can be used to construct *microworlds* in which bodies obey different natural laws, such as gravitation. By exploring these artificial microworlds, children can develop better intuitions about the properties of their own corner of the universe. (See "Designing Computer-Based Microworlds" by R. W. Lawler on page 138 of the August 1982 BYTE devoted to Logo.)

Given Logo's powerful list-processing capability, one would expect it to be of value in the language arts as well. To pick one simple example, suppose a child created several lists called nouns, verbs, adjectives, articles, etc., and assigned appropriate words to each list. The word order in each list can be randomized with the shuffle command, and a random sentence can be constructed by assembling words from each list in a syntactically valid order. Legitimate nonsense sentences can be automatically generated in this fashion (e.g., No yellow toad smells tall people.) while bringing the child to look at and solve the structure of English.

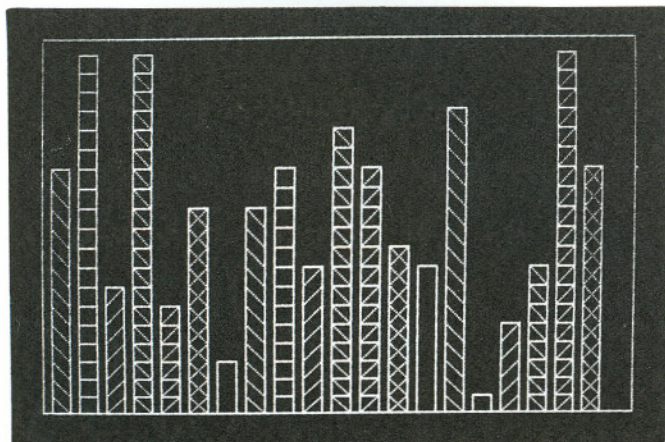


Photo 4: An example of business graphics possible with DR Logo. The program that produces this picture is in listing 1.

The educational value of this program can be seen on several levels. First, if the child creates the lists of words, a misplaced word will show up as a misplaced part of speech. Having a verb appear when a noun is expected results in an obviously invalid sentence structure. The result is a self-reinforcing mechanism for learning the parts of speech. Second, the student can learn to identify valid sentence forms without sample words (sort of the reversal of the traditional parsing process). This helps to cement sentence structure concepts as well. Finally, the student learns some of the challenges awaiting those who want to create natural-language interfaces between people and computers.

DR Logo in Business

While Logo is not usually thought of as a language for business applications, DR Logo has several characteristics that may change this perception. The creation of an interactive illustration generator using an inexpensive graphics tablet is quite easy in DR Logo. Photo 4 shows a possible display of business graphics, and listing 1 is the program that produced it.

In addition to business graphics, the list-processing capability of DR Logo makes it suitable for database management. In fact, one might envision incorporating some of the results of research in natural-language understanding to generate a query system that responds to questions such as: "If we increase our salaries by 10 percent this year and increase our sales by 20 percent next month, what will our profit be in the fourth quarter?"

There is no question that many business applications will be found for DR Logo, but it is premature to set limits on the scope of these applications.

DR Logo in Artificial Intelligence

There has been much talk lately about knowledge-based or "expert" systems. The noble efforts of personal computer software experts notwithstanding, sophisticated microcomputer programs that can adapt to various queries are few and far between. The major reason for this is the inadequacy of most computer languages for dealing with the types of data and operations natural to adaptive systems. Because of DR Logo's close connection with LISP, we expect to see artificial-intelligence techniques appearing in personal computer software rather than being limited to university and large industrial research centers as they have in the past.

This movement is valuable for several reasons. First, it will help to demystify artificial-intelligence research. Second, it will result in the application of advances in artificial intelligence to the development of practical programs. To pick one example, suppose you had a computer program (called car repair) that allowed the following dialogue:

User: I hear noises when I steer the car.

Computer: Do you think the problem is in your steering mechanism?

User: Yes, I think so.

Computer: Do you have power steering?

User: Yes.

Computer: Is the noise loudest when you turn the steering wheel?

User: Yes, but I hear it when the car is idling too.

Computer: You should check the level of your steering fluid before proceeding. Do you know how to do that?

User: Yes.

Computer: Fine. Check the fluid level. If it is low, fill the reservoir and see if the problem is fixed, otherwise we will continue to explore other causes.

Programs that allow this type of interaction can be used for many diagnostic applications and might be far more valuable applications for home computers than checkbook balancers or recipe files.

Domestic applications for artificial intelligence represent a sleeping giant. The list-processing capability and large workspace of DR Logo will allow this giant to be awakened and will enable the creation of a whole new class of applications software.

DR Logo is the first of a new family of languages that promises not only to change our programming style, but to alter the way we think about computing itself. ■

About the Authors

David Thornburg is an author and lecturer who has been actively involved in the development and support of user-friendly programming environments. His most recent book, *Discovering Apple Logo*, shows how Logo can be used to explore the art and patterns of nature.

Gary Kildall is the president of Digital Research Inc. He is active in research and was the developer of CP/M, Digital Research's version of PL/I, and DR Logo.

References

Logo:

1. Abelson, Harold. *Apple Logo*. Hightstown, NJ: BYTE/McGraw-Hill, 1982.
2. Abelson, Harold and Andrea diSessa. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. Cambridge, MA: MIT Press, 1981.
3. BYTE. August 1982.
4. Papert, Seymour. *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books, 1980.
5. Thornburg, David. *Discovering Apple Logo: An Invitation to the Art and Pattern of Nature*. Reading, MA: Addison-Wesley, 1983.

Artificial Intelligence:

1. Bundy, A., ed. *Artificial Intelligence: An Introductory Course*. New York: North Holland, 1978.
2. Winston, Patrick. *Artificial Intelligence*. Reading, MA: Addison-Wesley, 1977.

LISP:

1. BYTE. August 1979.
 2. McCarthy, John et al. *LISP 1.5 Programmers Manual*. Cambridge, MA: MIT Press, 1965.
 3. Winston, Patrick and Berthold Horn. *LISP*. Reading, MA: Addison-Wesley, 1981.
-

Listing 1: A DR Logo business-graphics program. See photo 4 for an example of the screen display. Note the indentation and embedded comments possible with DR Logo.

```
to graphics
; A sample business graphics program for bar graphs
make "screen.height 198
make "yfactor .25
make "zfactor .575
make "zdeg 22.5
make "xmin -139
make "xmax 139
make "ymin -79
make "ymax 119
make "return char 13
get.request
end
```

```

to get.request
(local "reply "h.or.v "s.or.o "2.or.3)
cleartext
make "reply prompt [Horizontal or vertical bars (h or v)] "char
if :reply = "h
    [make "h.or.v "h]
    [make "h.or.v "v]
if :reply = :return
    [stop]
make "reply prompt [Solid or open bars (s or o)] "char
if :reply = "s
    [make "s.or.o "s]
    [make "s.or.o "o]
if :reply = :return
    [stop]
make "reply prompt [2 or 3 dimensional (2 or 3)] "char
if :reply = 2
    [make "2.or.3 2]
    [make "2.or.3 3]
if :reply = :return
    [stop]
make :reply prompt [Values to be graphed] "list
if "reply = []
    [stop]
bar.graph :h.or.v :s.or.o :2.or.3 :reply
get.request
end

to prompt :text :type
local "reply
(type :text " : char 32)
if :type = "char
    [make "reply readchar print :reply output :reply]
    [output readlist]
end

to bar.graph :h.or.v :s.or.o :2.or.3 :values
cleartext
(local "max.value "min.value "origin "width "depth "axis "reply
    "graph.width "graph.height "proc "spacing)
if empty :values
    [stop]
make "max.value 0
make "min.value 999999999
if :h.or.v = "h
    [make "origin list :xmin :ymax make "graph.height :screen.width
    make "graph.width :screen.height make "axis 90]
if :h.or.v = "v
    [make "origin list :xmin :ymin make "graph.height :screen.height
    make "graph.width :screen.width make "axis 0]
if :2.or.3 = 2
    [make "spacing (:graph.width / count :values) * :yfactor]
    [make "spacing (:graph.width / count :values) * :zfactor]
if :2.or.3 = 3
    [make "width (:graph.width / count :values) * (1 - :yfactor)]
    [make "width (:graph.width / count :values) * (1 - :zfactor)]
make "depth :width * :zfactor
minmax :values
make "values scale :values :graph.height * .8 / :max.value

```

```

cleanup
penup setpos :origin pendown
if :h.or.v = "h
    [line [] list :screen.width ycor]
    [line [] list xcor :screen.height]
penup setpos :origin pendown
draw.bars :axis :width :spacing :2.or.3 :values
splitscreen
setcursor [0 23]
type [Return to continue]
make "reply readchar
end

to minmax :list
if empty? :list
    [stop]
if first :list > :max.value
    [make "max.value first :list]
if first :list < :min.value
    [make "min.value first :list]
minmax butfirst :list
end

to scale :list :factor
if empty? :list
    [output []]
output sentence (:factor * first :list) scale butfirst :list :factor
end

to cleanup
hideturtle
setbg 6
penup
home
clean
pendown
end

to draw.bars :axis :width :spacing :2.or.3 :values
if empty? :values
    [stop]
setheading axis
draw.l.bar :s.or.o :2.or.3 first :values :width :depth :zdeg
setheading :axis + 90
forward :spacing + :width
draw.bars :axis :width :spacing :2.or.3 butfirst :values
end

to draw.l.bar :s.or.o :2.or.3 :height :width :depth :zdeg
(local "origin "direction)
make "origin pos
make "direction heading
if :s.or.o = "o
    [make "proc "open.bar]
    [make "proc "solid.bar]
run (list :proc :height :width)
if 2.or.3 = 2
    [stop]
forward :height

```

```
right 90 - :zdeg
forward :depth
right :zdeg
forward :width
right 180 - :zdeg
forward :depth
back :depth
left 90 - :zdeg
forward :height
right 90 - :zdeg
forward :depth
penup setpos :origin pendown
setheading :direction
end
```

```
to open.bar :height :width
repeat 2 [forward :height right 90 forward :width right 90]
end
to line :pos1 :pos2
if not emptyp :pos1
  [penup setpos :pos1 pendown]
make "pos1 pos
setheading towards :pos2
forward sqrt sum
  sq ((first :pos1) - (first :pos2))
  sq ((last :pos1) - (last :pos2))
end
```

```
to sq :num
output :num * :num
end
```

```
to solid.bar :height :width
(local "course "origin)
make "course heading
make "origin pos
repeat :width / 2 [forward :height right 90 forward 1 right 90
  forward :height left 90 penup forward 1 pendown left 90]
if remainder :width 2 = 1
  [forward :height]
penup setpos :origin pendown
setheading :course
end
```


Once he learns to doodle, he's learned to program.

Introducing perhaps the best way yet for you or your kids to learn to use an IBM® PC or PCjr.

Dr. Logo™ Language from Digital Research®. It's the perfect guide for children of the computer age, and for grownups who find themselves in the middle of the computer age.

Family Computing says the Logo language is so easy to grasp, many beginners can learn it in an hour.

And the reason is simple. Dr. Logo is a graphics language. So Dr. Logo programming is literally an extension of what people do naturally, doodle.

The computer keyboard works like a pencil, the monitor like a sketchpad. With the help of a friendly turtle that traces commands on the screen, you see visual results instantly. So Dr. Logo turns problem solving and learning basic logic into an exciting video game.

Dr. Logo dictionary. Both illustrated by award-winning cartoonist Hank Ketcham.

We also offer three challenging learning pacs as part of a growing Dr. Logo library. Dr. Logo Graphics™ and Dr. Logo Games™ unlock the magic of turtle graphics. Dr. Logo Words™ uses words and symbols to create sentences, bar graphs, even pictures.

For more information about Dr. Logo, future Logo learning pacs or the Digital Research retailer nearest you, call 800-222-4000 ext. 400. In California, 800-772-3545, ext. 400. We think you

we speak the same language.

Dr. Logo, Dr. Logo Graphics, Dr. Logo Games and Dr. Logo Words are trademarks and Digital Research is a registered trademark of Digital Research Inc. IBM is a registered trademark of International Business Machines Corporation. Dennis the Menace and the Dennis the Menace characters are trademarks of Hank Ketcham Enterprises. © 1984 Digital Research Inc. All rights reserved.

 **DIGITAL
RESEARCH**
We make computers





LOGO An End-User Language

Alfred Riccomi



Digital Research, Inc.
P.O. Box 579
160 Central Avenue
Pacific Grove, CA 93950
408-649-5500

LOGO AN END-USER LANGUAGE

by
Alfred Riccomi

We have often heard the old adage about history repeating itself. This has been true about computers and computer usage as well as about most other technologies. Our "young" computer industry has just passed the 25th anniversary of the commercial introduction of Fortran; the first higher level programming language. Fortran achieved an order of magnitude increase in the number of people that could be expected to begin to learn how to program computers. That could be compared to what was achieved for the automobile industry by some combination of the electric starter, the synchromesh transmission, and the automatic spark advance (do you remember hand cranking, double clutching, and the spark advance lever on the hub of the steering wheel?) Since the introduction of Fortran, many incremental improvements have appeared: ALGOL with recursion, COBOL for business applications, PASCAL for better quality structured code, many dialects of Fortran culminating in Fortran 77 which incorporates many of the best features of the other languages, plus a myriad collection of other algorithmic languages employing algebraic notation. All have been good at least in some way. None have been to the computer industry what the automatic transmission was to the automobile industry. None have made it possible for all end-users to do their own programming. Hence, we still need professional "chauffeurs" or highly motivated, technically capable, "daredevil drivers" to program our machines.

Why would we want end-users to do their own programming? Why have "automatic transmission" programming languages? What price would have to be paid for such languages? The answers are pretty much the same to similar questions about other technologies. More and more people want to use the new technology, be it the automobile, computer, postal service, telephone, whatever. In each case, projections have indicated the absurd need for every man, woman and child to become a full time chauffeur, programmer, mail-sorter, or switchboard operator. Of course that is ridiculous; so what has happened for the older technologies is the development and acceptance of some end-user participation in the application of the technology: the automatic transmission in the automobile, nine digit zip codes and optical scanners for the mail, replacement of "exchange name" with all digit dialing, area codes, and WATS lines for the telephone. In every case developers had to search for, grope for some approach that end-users could learn easily and quickly, something they could understand. In every case a price had to be paid: greater initial cost, some loss in operational efficiency, and a reduction of flexibility to meet unusual conditions. In every case the price had to be paid, or that industry's growth would stop.

Well, what about computers? Today's half million professional programmers would have to grow to 30 million by 1990 and three billion by 2000 in order to create all the software needed for the computers that the industry can build and that the people will want to buy in those time periods (1). Of course what will happen is either the aforementioned stagnation or the appearance and acceptance of very high level end-user programming languages (plural).

James Martin in his book *Application Development Without Programmers* (1) includes such languages in his view of the future. He expands upon how simple and complex query languages, report generators, graphics languages, application generators, very high-level programming languages, and parameterized application packages will allow end users to bridge the gap between the necessarily limited number of professional programmers and the need for software. The price that has to be paid for these end user tools is: more memory and faster CPU's, less throughput for a given hardware configuration, and difficulty if not impossibility when trying to solve the usual case. Ah, but the unusual case is what will provide continuing job security for today's professional programmers just as it has for chauffeurs, switchboard operators, and postal workers. Actually, the unusual case is what the professional programmers want to work on: operating systems, data-base management, air traffic control, and many more plus, perhaps most important of all, the end-user usable tools themselves.

What about the price, who will pay it? First, the cost of "conventional" software is going up. Figure 1 shows that the fraction of total programming costs being spent on software maintenance and conversion has been growing while the fraction spent for new application development has been declining. Martin projects that maintenance and conversion costs will soon represent 70 percent of the total cost of professional programming. Meanwhile, the cost of computing hardware relative to people time has been coming down. Figure 2 depicts how a one million instruction per second computing system (including memory, disks, etc.) will soon cost less per hour than labor costs, and that by the end of this decade it will be an order of magnitude less. This cornucopia of computing power will render the premium that must be paid for end-user programming insignificant when compared to the advantages.

Ah, but how? Will everyone learn BASIC? Perhaps it will be Pascal as has been tentatively decided by the Texas State Education Agency. Certainly it will not be COBOL or PL-I as those languages have proven to be elusive to many professional programmers. Those languages were intended to be friendly to highly specialized, narrow segments of the population. Hence, Fortran and its interactive derivative, BASIC, are friendly to people familiar with and undaunted by algebraic notation and with a need to solve mathematical problems. Pascal was intended for the subset of that same group that sought to be computer scientists. It has proven also to be an excellent tool for large software projects. COBOL was intended to be and is relatively friendly to professional programmers of business applications. And PL-I was to be all things to all professional programmers. How do they serve the needs of the end-user? They don't!

But wait, BASIC is on all the low cost personal computers! Is it not friendly to end-users? BASIC was chosen to be the first higher level language for microcomputers by Bill Gates and Paul Allen not because it was particularly user friendly, but because they knew it was the only higher level language that had even a ghost of a chance of fitting within the 4K bytes of memory that were standard on the MITS Altair 8800 microcomputer in 1975. BASIC has proven to be a good choice for, unlike Fortran after which it was modeled, it was intended to be interactive and its primary goal was to be used to teach programming to college students. But BASIC is not for everyone.

What is emerging as end-user programmable computing systems is the result of over a decade's efforts to find an approach that end users can learn easily and quickly, something they can understand. Researchers in many places, including the Massachusetts Institute of Technology and Xerox Park, have found that interactive graphics provide at least one such approach, perhaps the best approach. Another adage telling us that "a picture is worth a thousand words" is based upon the nature of human physiology. We all learned to draw (crudely of course) before we learned to write (also crudely). Just this year Apple revealed their LISA computer which is based heavily upon the interactive graphics research done by Alan Kay at Xerox PARC. Late last year Visicorp announced its "VisiOn" software package which is based on the same work as LISA, and will be available on the IBM Personal Computer, the TI Professional Computer, and many other systems. The "Smalltalk" software system was made available by Xerox earlier in '82, while the Xerox Star when introduced in '81 was the first commercial product incorporating Alan Kay's interactive graphics concepts. But all of these examples can best be described as end-user friendly application delivery systems, *not* end-user programmable systems! The first commercially available example of end-user programmable systems has already appeared. It is the LOGO programming language developed by Seymour Papert at MIT, and it was first introduced commercially on a personal computer by Texas Instruments in April, 1981. It was the first such language, but just like Fortran, it will not be the only such language. TI LOGO like IBM Fortran was the first commercially available LOGO for personal computers, but again like IBM Fortran it is not the last LOGO.

LOGO has its origins in artificial intelligence and in that community's favorite language, LISP. Harvey Cragon of TI is a recent convert to LOGO and he has made the analogy that LOGO is to LISP as BASIC is to Fortran. Alan Kay came from the MIT Artificial Intelligence community, but at Xerox PARC he followed a different direction than did Papert at MIT. Kay can be said to have led Smalltalk in the direction of exploiting ever increasing computational power while holding to a fixed cost. Papert led LOGO towards ever decreasing cost while holding to a fixed computational power. Hence, Smalltalk '80 runs well on a machine with 10 times the power of a Motorola 68000 microprocessor (2), and TI LOGO runs well on a \$750 configuration of the TI-99/4A Home Computer. Now the two are not equivalent, nor are they intended for the same end users. Smalltalk is intended to facilitate the development of user friendly applications by professional programmers, while LOGO *is* an end-user programming language. An interesting rumor has it that Alan Kay, now working for Atari, is developing a processor for LOGO around a Smalltalk system (no hints as to what kind of computer power will be needed).

Following the introduction of TI LOGO, two offerings of a LOGO developed by MIT for the Apple II appeared on the market, one from Terrapin, Inc. and the other from a firm named Krell. Having been created by many of the same people at MIT that developed TI LOGO this version had some improvements over TI LOGO, namely floating-point arithmetic. However, it fell short in the area of color and dynamic graphics as the Apple hardware lacked the sprite feature and 16 color display capability of the IT hardware. During the second quarter of '82, Apple corporation made available another version of LOGO licensed from LOGO systems Incorporated, a Canadian firm that hired many of the same developers from MIT. That version included additional features, corrected some shortcomings of MIT Apple LOGO, and promised (but as of this writing in '83 has not yet delivered) the TI sprite feature. Commodore included a reference to LOGO in the announcement of their model 64 in '82, but it has not yet appeared. Tandy started delivering a LOGO on its Color Computer late in '82, but at best this version offers nothing beyond those already on the market. LOGO Systems Inc. is rumored to have licensed their LOGO to Atari but as of this writing it has yet to appear. The most significant recent development of LOGO is one announced in January, 1983 by Digital Research Incorporated (the create and distributors of the CP/M operating system) for both the IBM Personal Computer and the TI Professional Computer. This version contains essentially all of the features of existing LOGO processors (alas, but no sprites) plus many more including the multiple "window" feature of Smalltalk, Xerox Star, VisiOn, and LISA. DR LOGO (for Digital Research), unlike all its predecessors which were implemented for specific machines, is highly machine independent and will be available for all computers supporting either CP/M-80 or 86 and Digital Research's new graphics feature. In my opinion this version of LOGO may prove to become as widespread as was the Fortran IV version of that language. DR LOGO will become the first widespread end-user programming language, and will be used for end-user driven applications programming, not only for achieving computer literacy and providing a good environment for discovery learning in a school environment (3).

Well, what is LOGO? If we examine BASIC we find it has a rather high threshold which must be crossed before one can begin to create a program that one both likes and would be proud to show to peers. Algebraic notation, a language intended primarily for solving numerical problems, arrays, and matrices all serve to frighten mathophobic people (3). And if one's problem is graphic rather than numeric, BASIC serves as little more than a higher level language host for very low level data definitions and primitive operations. In other words, BASIC has a large "gulp factor" (4).

Once mastered, BASIC presents a formidably low ceiling even when compared to other languages of the same genre. Except for some of the most powerful versions, partitioning of programs into separate procedures is awkward, if at all possible. Recursion is not allowed, although it has existed since the introduction of ALGOL over twenty years ago and is included in most languages developed since then. List processing, which is at the core of the power of LISP, is not supported, although a subset of list processing capabilities is typically offered in the form of a character string feature. Before you get the impression that I am unalterably opposed to BASIC, let me say it has a place in the scheme of things. It has been a great boon to computing, and one of my proudest accomplishments was the definition of the computers (the 99/4, 99/2, and CC40 Compact Computer), and to SofTech Microsystem's p-System BASIC. Together, these make up the largest current shipping rate of any dialect of BASIC. That's good but that is not enough; BASIC is not the "automatic transmission" that is needed.

If we examine LOGO we find a very low threshold that needs to be crossed before being able to produce a program one likes and is proud of, will show to one's peers. Immediate success is achieved because the language is designed around interactive graphics. Doodling, sketching, drawing using "Turtle Graphics" (3) are the universal medium to which all humans subscribe. Color, motion, and sound are all part of our world. Music, TV, Entertainment on a CRT are more natural today than any algorithmic notation can ever be. The set of key words is naturally extendable thereby serving as personalized building blocks. Problem partitioning follows easily through the same "teach the computer a new word" approach (3). Error messages are in the user's native language (not numeric codes), and direct one to the offending small procedure into which all problems big and small are naturally partitioned. Hence "bugs" become fixable problems and not an issue of personal failure. Evidence abounds that all people achieve success programming in LOGO (assuming they have at least *begun* to learn to read). LOGO has a low "gulp factor."

LOGO has a high ceiling. Recursion, LISP-like List Processing, the aforementioned problem partitioning procedural programming, and interactive graphics combine to make a powerful programming environment. A new concept called "Dynaturtles" provides dynamic graphics with the physical properties of mass, velocity, momentum, force and quantum mechanics (3) (5). LOGO has even proven to be well suited in the teaching of introductory college physics at MIT.

Now if one's problem is numeric rather than graphic, LOGO at first glance appears to lose its advantage over BASIC. The syntax for performing arithmetic is different from algebra, thereby allaying the fears of the mathophobic. But upon closer observation, solving mathematical problems requires algorithmic semantics and they are there. However, the goal of LOGO was "no threshold" over which one must cross to get started programming (3), not no threshold to cross for whatever one may want to do with a computer. With LOGO the mathophobic user typically has lost whatever fear of the computer may have existed long before the need to solve a mathematical problem appears. Then, within that person's native ability, learning the syntax and semantics needed to solve the problem becomes just a small extension to what is already known.

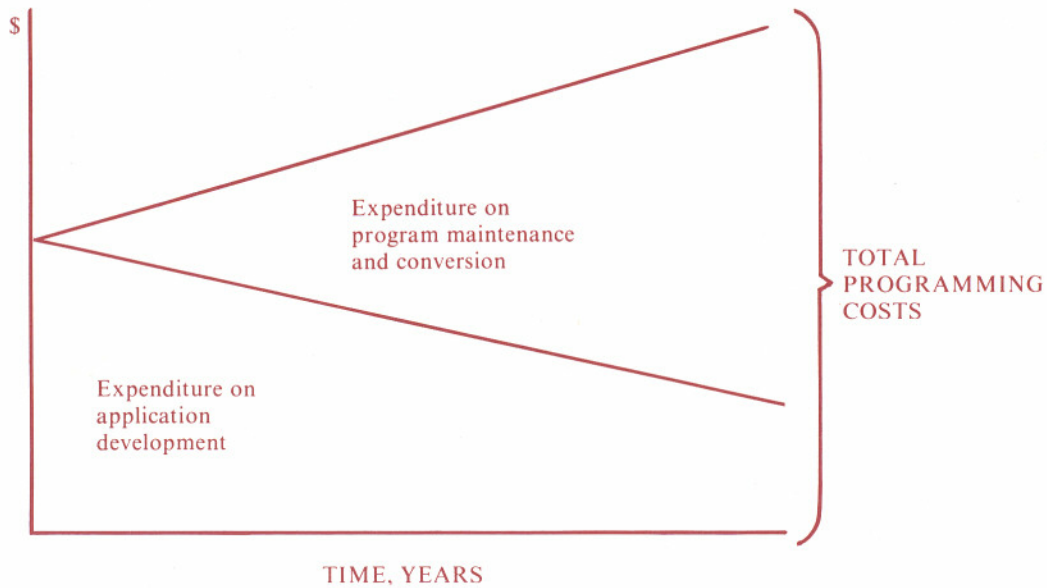


FIGURE 1

New applications often deferred by the rising cost of modifying existing programs.

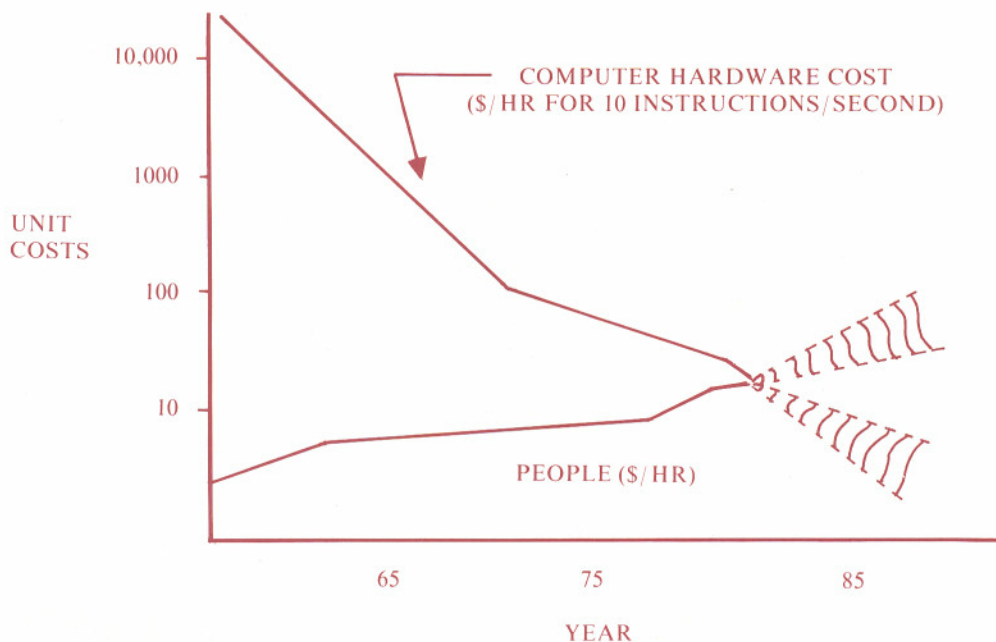


FIGURE 2

A comparison of the estimated cost of computers with the cost of people in US \$ per hour.

Just as "everyone" learns to read and write, "everyone" learns to program using LOGO. What "one" reads and writes depends upon circumstances, personal tastes, and native ability. Likewise what "one" programs with LOGO depends on the same things. Not everyone writes great poetry or solves partial differential equations with a pencil. Neither will everyone write Accounts Receivable Programs or design bridges with LOGO; but it is within the reach of everyone to write great poetry and solve partial differential equations with a computer if they can program.

That is fine but what kind of things would be done through end-user programming? Professional programming today is best typified as "pre-specified" programming (1). Formal requirements are defined, specifications created, and then the END USERS SIGN OFF ON THE SPECS! But in the majority of cases that is not what the end users want. In fact they do not know in detail what they want until they see the result, and then they change it — often! The uncertainty principle of physics applies to computing; the solution to the problem changes the problem (1).

Professional programming at its best is tightly managed with schedules, milestones, checkpoints. But the end user wants the problem solution dashed off in order to see the results, and then dashed off again. Formal documentation that is expensive to produce and maintain, lead times of months and even years, and full time maintenance staffs are what we have, i.e. "a van pool with chauffeurs." Self documentation, quick turnaround of days or at most weeks, and incremental maintenance by the users is what we want, that is "our own car with an automatic transmission." There is a place for pre-specified computing where computationally efficient large programs that must be supported "for all time" are required. But in the vast majority of cases user-driven computationally extravagant small programs that die when their user no longer needs them are sufficient. The days of the 4K byte Altair 8800 and the 32K word IBM 7094 are behind us. Today you can buy a Commodore 64 with over 60 percent as much memory as had the 7094 for under \$400. Microprocessors such as the Intel 80286, the Motorola 68000, and the Texas Instruments 99000 offer the CPU speed of the 7094. We are beginning to use this computational plenty to make computers truly user friendly, including end-user programming languages of which LOGO is the first.

REFERENCES

- (1) James Martin, *Application Development Without Programmers*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.
- (2) Adele Goldbarg, presentation of paper titled *The Influence of an Object Oriented Language on the Programming Environment*, Proceedings of the ACM computer Science Conference, Orlando, Florida, February, 1983.
- (3) Seymour Papert, *Mindstorms*, Basic Books, Inc., New York, 1980.
- (4) John Mashey, *Perspectives on Programming Environments*, Proceedings of the ACM Computer Science Conference, Orlando, Florida, February, 1983.
- (5) Harold Abelson, *LOGO*, BYTE/Mc Graw Hill Publications, Inc., Peterborough, NH, 1982.



A Complete Family of Powerful Programming Languages For the Apple II, II Plus, and IIe Personal Computer.

From the company that brought you industry standard operating systems, languages and graphics products, comes a complete family of languages and programming tools formatted for the Apple II line of computers running CP/M.™

CBASIC	\$150.00
CBASIC Compiler	\$500.00
Pascal/MT+	\$350.00
Speed Programming Package	\$200.00
MicroFocus LEVEL II COBOL™	\$1,600.00
MicroFocus CIS COBOL™	\$850.00
DR Assembler Plus Tools	\$200.00
Access Manager	\$300.00
Display Manager	\$400.00
MicroFocus Animator™	\$800.00
MicroFocus Forms-2™	\$200.00

LEVEL II COBOL, CIS COBOL, ANIMATOR, and FORMS-2 are registered trademarks of MicroFocus, Inc.

The Apple logo and products are either trademarks or registered trademarks of Apple Computer, Inc.

Digital Research Products and logo are either trademarks or registered trademarks of Digital Research, Inc.

© 1983 Digital Research, Inc.

These new CP/M-based products let you enjoy the same quality and performance that have made Digital Research the leader in micro-computer software development.

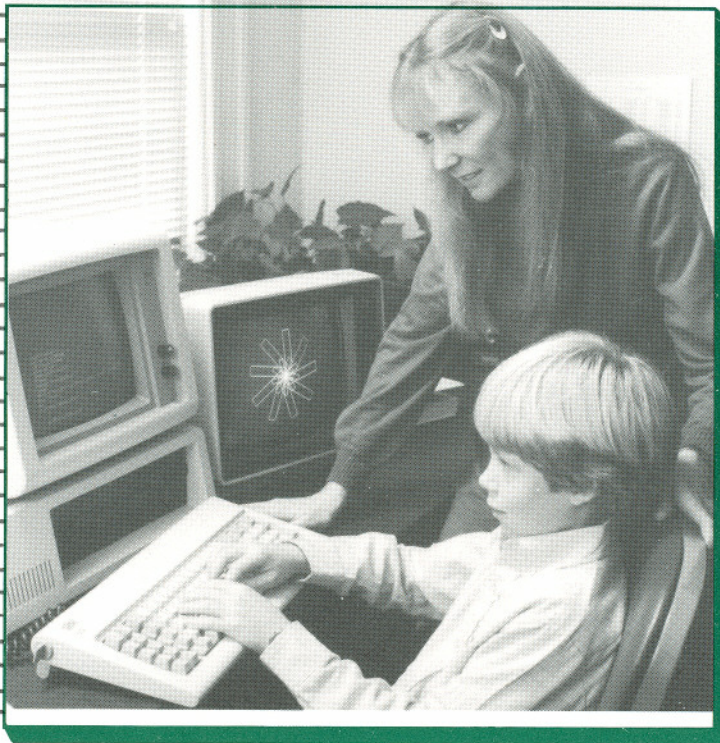
Available December 1, 1983.

And, you can use it with the new CP/M Gold Card™ from Digital Research to give your Apple II:

- maximum memory and speed capabilities
- access to over thousands of CP/M compatible applications
- the powerful CP/M Plus Operating System.

Contact Digital Research for more information.

 **DIGITAL
RESEARCH™**



The Language for Learning™ — DR Logo™

Whether you are a first time user or an experienced programmer, DR Logo is an invaluable tool for learning, programming, and game playing.

Although it's simple to learn, you'll find that DR Logo is a powerful language that grows with your computing needs at school, in business, or at home.

If you have never worked with a computer DR Logo can help you visualize the logic of programming. While you are having fun creating graphic displays, DR Logo will be teaching you the basics of programming. Because you can view the graphics output and commands simultaneously on a split screen, errors are easily detected. A full screen editor makes changing your programs a simple operation.

DR Logo employs turtle graphics to make learning about computer science enjoyable. The turtle is a graphic representation of a location on the screen. The turtle leaves a path of color to show you where it has been. DR Logo contains many commands that allow you to instruct the turtle and control its movements. This approach to programming helps you to discover mathematical concepts painlessly by letting you draw geometric shapes on a computer display. Because it is easy to manipulate, the turtle will have you programming in no time. A cartoon-illustrated tutorial for children and an adult oriented tutorial let you easily learn how to tell the turtle what you want it to do. If you need help remembering commands you can ask for a simple explanation and an example of how each command is used will be displayed on the screen. DR Logo commands, called "primitives," help you create procedures for graphics,

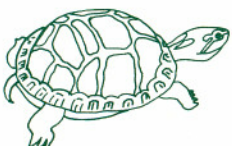
games, and business. Once you have created a set of DR Logo procedures you can save it and use it again. Your personal library of procedures can be changed, discarded or combined with other procedures to form more complex programs. A Reference Manual provides the complete encyclopedia of all available primitives.

If you already know how to program, you'll enjoy this new approach to writing software. Powerful list processing abilities make DR Logo suitable for a wide variety of applications, from simple symbolic mathematics to natural language translation, artificial intelligence, knowledge-based systems and robotics. And if you ever used Apple Logo you'll find DR Logo provides all the features of Apple Logo plus the advanced features you want for writing application software:

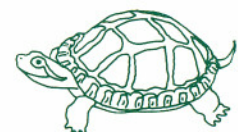
- Large Workspace
- Advanced Debugging Aids
- Workspace Cross Reference Commands
- Split Screen and Text Windowing
- Upper and Lower Case Characters
- Expanded String Processing Primitives
- Additional Game Playing Primitives
- Compatible with Digital Research Graphics

Availability

Look for DR Logo in retail computer stores early this summer. If you have wanted to program but thought it was too hard or if you are a programmer and wonder why it has to be so hard, ask your dealer for a demonstration of DR Logo.



DR Logo provides the challenge of programming without the frustration.
DR Logo and the Digital Research logo are trademarks of Digital Research.



LOOK WHY YOU SHOULD BE PROGRAMMING WITH DR. LOGO LANGUAGE.

Dr. Logo Language will expand your ability to program creatively, with more features for the sophisticated programmer than any other implementation of the logo language currently available.

▶ BUILT-IN OPERATING SYSTEM.

SpeedStart™ is embedded on the same disk with Dr. Logo Language, eliminating the need to use any other software to begin programming.

▶ SPLIT SCREEN DEBUGGING.

Only Dr. Logo Language features this advanced debugging which allows you to view and trace program execution level by level. A special WATCH facility allows you to pause and modify the program during debugging.

▶ LOTS OF PROGRAMMING WORKSPACE.

Dr. Logo Language takes advantage of all the memory you can install on your IBM PC, so you can write more sophisticated applications. You can group procedures and variables in packages to simplify saving, erasing, and displaying objects in your workspace.

▶ POWERFUL PRINTING CAPABILITIES.

Dr. Logo Language supports upper and lower case letters for more readable text, and also supports a printer and color monitor. Monochrome monitor support is also available.

▶ GREATER ACCURACY.

Double precision, floating point math capability with 15-digit accuracy provides the power you need for extended calculations.

▶ ON-LINE HELP.

Command listings and descriptions of what they do are provided by a powerful help facility which displays definitions and required syntax of Dr. Logo Language primitives to help you learn-as-you-go.

▶ AN EASY-TO-FOLLOW TUTORIAL.

Lot of illustrations will help you quickly learn the logical ways that Dr. Logo Language can help you solve programming problems.

THE FIRST LOGO FOR THE IBM PC.

Dr. Logo Language runs on an IBM PC with at least 192K RAM. This memory supports the advanced functions Dr. Logo Language provides for sophisticated programming and to expand your computing creativity. You will also want to use;

▶ AT LEAST ONE FLOPPY DISK DRIVE.


▶ IBM COLOR GRAPHICS DISPLAY ADAPTER AND GRAPHICS MONITOR.

▶ IBM MONOCHROME DISPLAY ADAPTER AND IBM MONOCHROME DISPLAY. (OPTIONAL)

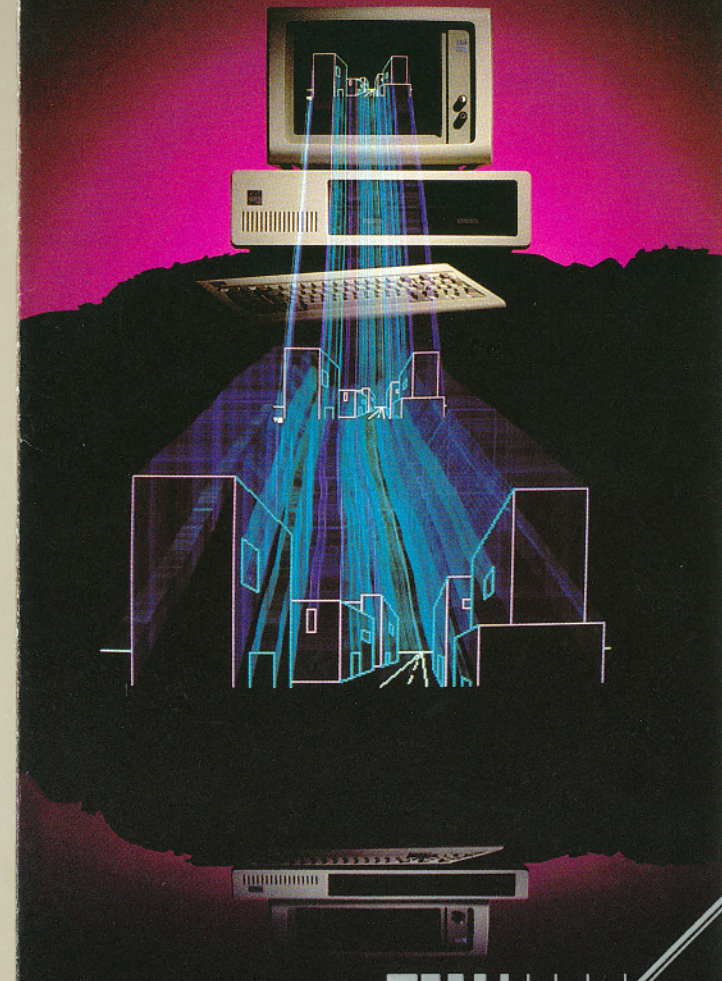
HOW TO GET DR. LOGO LANGUAGE.

Dr. Logo Language is available through authorized Digital Research Dealers. For the name of the dealer nearest you, call 800-227-1617, Extension 400. In California, call 800-772-3545, Extension 400. Large volume orders are available through the Digital Research OEM sales organization. Call 415-856-4343.

Digital Research products and logo are either trademarks or registered trademarks of Digital Research, Inc. IBM is a registered trade name of International Business Machines.

 **Dr. Logo**
Language™

WORKING TO EXPAND YOUR CREATIVE HORIZONS.



 **DIGITAL RESEARCH™**

Now available
for the IBM® PC!

USE DR. LOGO LANGUAGE™ TO PROGRAM INTELLIGENCE INTO YOUR COMPUTER.

This exciting language challenges you to expand your creativity, making it seem as if your computer is helping you by becoming more intelligent. Dr. Logo Language™ is an intuitive language that lets you logically create solutions to programming challenges.

Dr. Logo Language includes turtle graphics, a visual programming package that lets you immediately and easily command your computer to accomplish impressive graphics. Procedures created with Dr. Logo Language can be used as modules which you can combine and build upon to develop solutions to the most complex programming problems.

Dr. Logo Language is a powerful language, especially in helping you to manipulate lists of information. You can choose the most convenient way to handle structures, based on your needs, either character by character, word by word, or list by list. As a result, you can concentrate on creating new solutions to problems, rather than getting distracted by the mechanics of executing specific functions.

IDEAL FOR FIRST-TIME USERS.

Dr. Logo Language is based on the proven educational philosophy that you learn by doing. Through friendly interaction with the computer, first-time users can be writing their own programs in record time. And, Dr. Logo Language makes programming fun!

The versatility of Dr. Logo Language makes it a powerful tool for the home, office, lab, or any place you'll find computers being used to create solutions. And, the language's ability to help you intuitively control the computer makes it equally valuable for the programming professional and the first-time user.



PROVIDING AMAZING VERSATILITY FOR YOUR IBM PC.



```
DEBUG
[19] in while, while :cond :body
[20] in while, if run :cond [run :body] [stop] while :cond :body
[20] in while, while :cond :body
[21] in while, if run :cond [run :body] [stop] while :cond :body
[21] in while, while :cond :body
[22] in while, if run :cond [run :body] [stop] while :cond :body
[22] in while, while :cond :body
[23] in while, if run :cond [run :body] [stop] while :cond :body

PROGRAM
Would you like to try again (Y / N) ?Let's play Blackjack.
Let's play Blackjack.
Please press any key to start.

Let's play Blackjack.
Please press any key to start.
```

▶ ADVANCED DEBUGGING FOR PROGRAMMERS

Dr. Logo Language's sophisticated debugging can pinpoint the exact location of a problem within a program. It allows you to do the debugging on the top screen with the results displayed on the bottom screen.

This allows you to view and trace the program level by level to facilitate faster, easier programming.

```
pprop "Frank "salesman "TRUE
pprop "Frank "quota "met
pprop "Frank "age 47

pprop "Bill "salesman "TRUE
pprop "Bill "quota "not met
pprop "Bill "age 50

pprop "Zelig "salesman "TRUE
pprop "Zelig "quota "met
pprop "Zelig "age 28

display [[quota = met] [age > 30]]

Frank

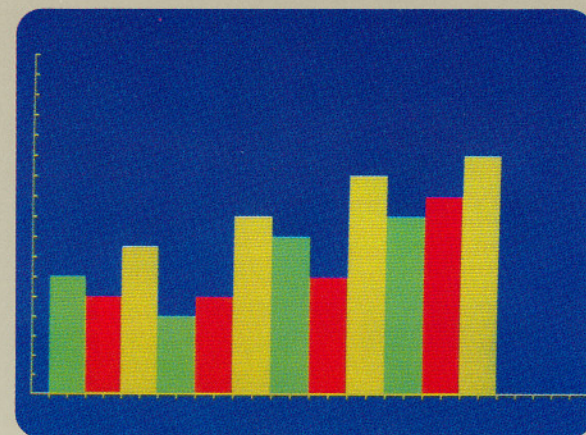
display [quota = met]

Frank Zelig
```

▶ WORK SPACE MANAGEMENT

Dr. Logo Language manages data in modules so that you can manage information more efficiently. For example, in managing a mail list, Dr. Logo Language will search for an entire word instead of a string of characters.

Dr. Logo Language is a powerful tool for creating, cross referencing and managing any large data base.



▶ SOPHISTICATED GRAPHICS

You can easily program Dr. Logo Language to graph data, such as sales results. Sales figures can be entered simply by defining the variables in the construction of a rectangle. Each location can be programmed to respond to changes in sales figures. Pie charts and other business graphics can be quickly and easily developed.



▶ TURTLE GRAPHICS

The exceptionally high resolution of Dr. Logo Language's graphics provides more programmer control by allowing you to pinpoint the movement of the turtle. Graphics to suit almost any need can be easily developed and manipulated.

You can program an exceptional variety of colors with Dr. Logo Language, even changing background colors.