ST-LOG

THE ATARI ST OPERATOR'S MAGAZINE

AUGUST 1986 ISSUE 5

THIS ISSUE:
Intro to Logo

A NEW COLUMN:
Ian's Quest

A BONUS FOR DISK SUBSCRIBERS:
Solid States for the 520ST

ST-LOG

THE ATARI ST OPERATOR'S MAGAZINE

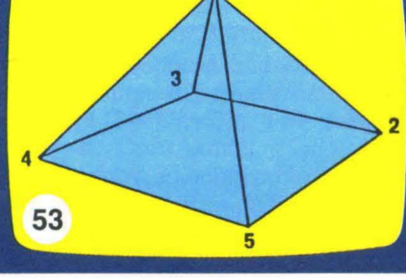
ISSUE 5
AUGUST 1986

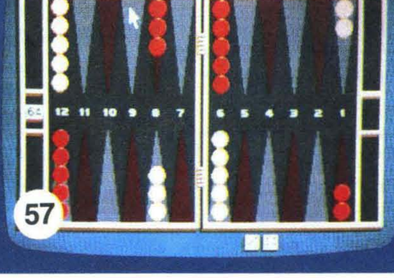
FEATURES

SPECIAL BONUS FOR DISK SUBSCRIBERS

REVIEWS

COLUMNS





ST-Log is normally printed as a center section in **ANALOG Computing** (ISSN 0744-9917), published monthly for $28 ($36 in Canada, $39 foreign) per year by ANALOG 400/800 Corp., 565 Main St., Cherry Valley, MA 01611. No portion of this magazine may be reproduced in any form without written permission of the publisher.

VIP Professional™

Finally – A Business Program that Brings Lotus 1-2-3® Functionality to Your *Atari ST*™!

VIP Professional is a state-of-the-art, integrated spreadsheet program which brings together a spreadsheet, a database and graphing capabilities. Professional was modeled after the powerful and best-selling Lotus 1-2-3® program which dominates the business world

Worksheet Magic

Nothing is left out of the workings of the worksheet. Ranges of cells can be named for convenience; column widths are variable; the screen can be split into two windows; titles can be frozen; contents of cells may be copied or moved; the worksheet may be altered as a whole or only partially; the list goes on and on. Perhaps most important, Professional can use and save Lotus 1-2-3 files for transfer between computers.

The worksheet includes over 45 special functions to simplify commonly used formulas, including powerful financial functions for the internal rate of return, present value, and future value. Of course Professional also has all mathematical, trigonometric, table, conditional and logical functions.

Database Power

The built-in database can handle up to 8192 records, with a possibility of up to 256 fields. The records can be searched, sorted and analyzed to find your best salesperson or your rarest stamp. Sorts can be done using multiple criteria, in ascending and descending order. And database functions can be used to do up to seven different kinds of statistical analyses of your database.

Graphs

The graphing capabilities of Professional are astounding. Not only are there six completely different types of graphs available, there are tens of ways to manipulate the data, titles, grids, colors, legends, keys, and scaling of the size of the graph.

Macros

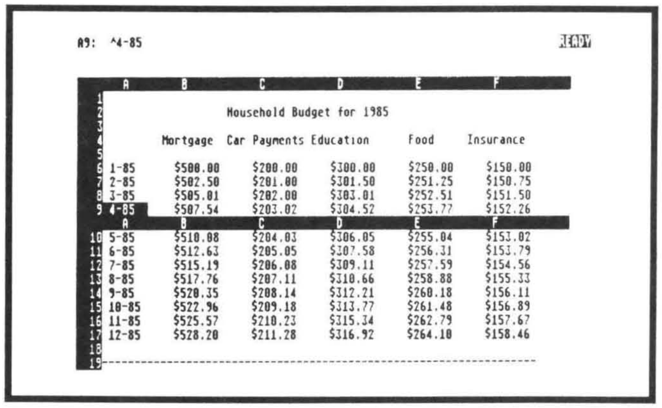
Professional also includes sophisticated macro programming commands. With several special macro commands, the user can actually *program* Professional to be dedicated to a specific task such as accounting.

Just Minutes to Learn

Professional is as easy to use as it is powerful. It comes with a user-sensitive tutorial for the newcomer. And help is built right into the program. With the handy tutorial, you will be able to create professional worksheets in just minutes.

Introducing Professional LITE™

For those of you who do not need the full power of Professional, we offer Professional LITE™. Though without the macros and the database features, and having a smaller sheet size (256 columns by 2048 rows, LITE still packs a powerful punch for only $99.95!

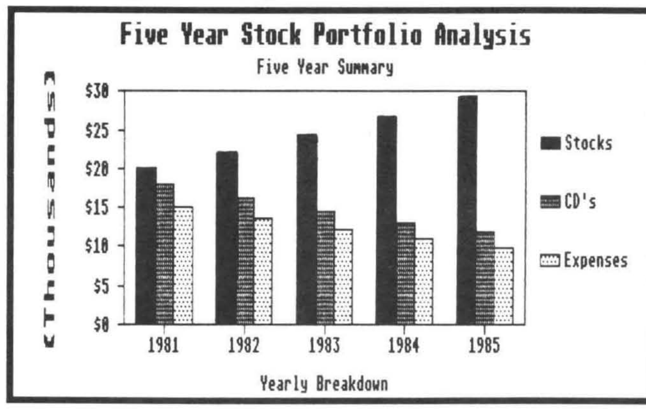


A9: ^4-85 READY

Household Budget for 1985

	Mortgage	Car Payments	Education	Food	Insurance
1-85	$500.00	$200.00	$300.00	$250.00	$150.00
2-85	$502.50	$201.00	$301.50	$251.25	$150.75
3-85	$505.01	$202.00	$303.01	$252.51	$151.50
4-85	$507.54	$203.02	$304.52	$253.77	$152.26
5-85	$510.08	$204.03	$306.05	$255.04	$153.02
6-85	$512.63	$205.05	$307.58	$256.31	$153.79
7-85	$515.19	$206.08	$309.11	$257.59	$154.56
8-85	$517.76	$207.11	$310.66	$258.88	$155.33
9-85	$520.35	$208.14	$312.21	$260.18	$156.11
10-85	$522.96	$209.18	$313.77	$261.48	$156.89
11-85	$525.57	$210.23	$315.34	$262.79	$157.67
12-85	$528.20	$211.28	$316.92	$264.10	$158.46

Integrated Spreadsheet Power



Easy-to-Use Graphs

The Power of Professional
Only $179.95
Or the Power of LITE
Only $99.95

If your dealer is out of stock, order direct. Send your check or money order to the address below, together with $3 for shipping and handling. In California add 6% sales tax. COD's and purchase orders not accepted. Personal checks will be held for three weeks to clear. All prices are subject to change without notice.

VIP TECHNOLOGIES

132 Aero Camino
Santa Barbara
California 93117

(805) 968-9567

SYSTEM REQUIREMENTS: Atari 520K; One disk drive; Monochrome or color monitor; Works with printers supported by GEM.

SYSTEM REQUIREMENTS: Atari 520K; One disk drive; Monochrome or color monitor; Works with printers supported by GEM.

Solid States ST

A bonus for ST disk subscribers

by James Luczak

This article is published as a bonus for our ST disk subscribers. The listings for it, and complete documentation, are found on the ST disk version of this issue, and the article here is simply a brief explanation to go with those programs.

Solid States is a 3-D object plotting program that appeared in issue 16 of **ANALOG Computing**. It was soon followed by **Solid States Revisited** in issue 19, and **Son of Solid States** in 22.

Each of the latter articles added enhancements and image files to the original. Now, you can use **Solid States** on the 520ST.

Solid States for the 520ST (for brevity, **SS-ST**) has the same features as the original. You can load, store, display and edit image files. And there's a plus—this version can automatically rotate an image along the X-, Y- or Z-axis.

The keyboard entry mode has been modified, to make it easier to use the keypad on the ST to enter or edit files. Entering object coordinates has been made quick and easy, by using the mouse to input coordinate data.

SS-ST will run in medium- and high-resolution modes. The image files (files containing object parameters) use a slightly different format from the 8-bit version. If you port image files from the 8-bit version to the **SS-ST**, they won't work. Don't panic!

By modifying a couple program lines, you can load and run ported image files. When you save the ported files, they'll be saved with the ST image file format.

Solid States recapped.

Solid States allows you to display three-dimensional objects on-screen, in a wire frame configuration. You are able to view the object from any angle, retaining true perspective.

There are two steps in drawing a 3-D object. First, you must define the location of each "point" of the object in X-, Y-, Z-coordinates. Figure 1 shows a pyramid. Each number corresponds to a point of the pyramid.

Second, you must define the "endpoints" of each line to be drawn. In other words, you must tell the computer how to draw the object. For example, to draw the pyramid, you must instruct the computer to draw a line from point 1 to point 2, then from point 1 to point 3, etc.

To find the X-, Y-, Z-coordinates of each point of the pyramid, start with a graph like the one shown in Figure 2. Next, draw the top view and number each point where two or more lines intersect. A top view of the pyramid is shown in Figure 3, with five points labeled.

Now, draw a side view of the object, as shown in Figure 4. Note that the vertical axis is labeled as the Z-axis. This view will give the object's height. Again, number each point where two or more lines intersect. Notice that, in Figure 4, points 3 and 4, and points 2 and 5 are at the same location. This is because they have the same X- and Z-coordinates.

Write down each point's coordinates. To find the X-coordinate of point 1, look at Figure 3 or 4. In either case, the coordinate for point 1 is *0*. For the Y-coordinate, look at Figure 3. The Y-coordinate for point 1 is *0*.

Finally, to find the Z-coordinate, look at Figure 4. The Z-coordinate for point 1 is 5. Now that you have the coordinates for point 1, continue until all the points have been defined. Figure 5 shows the coordinates for the five points of the pyramid.

Figure 6 shows the sequence in which to draw the pyramid's lines. These are the endpoints for each of the eight lines that make up the pyramid.

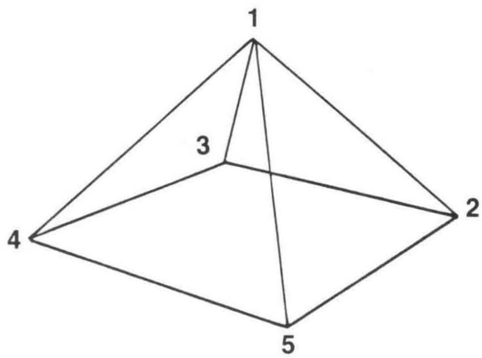


Figure 1.

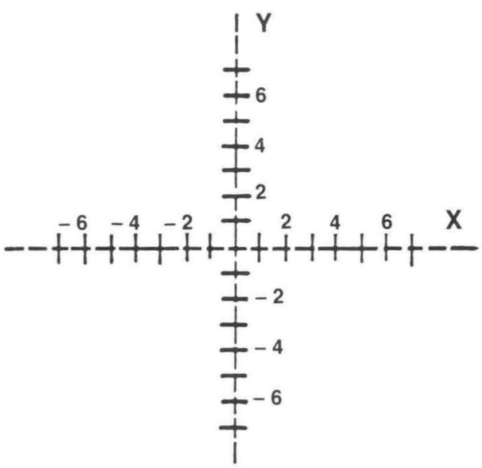


Figure 2.

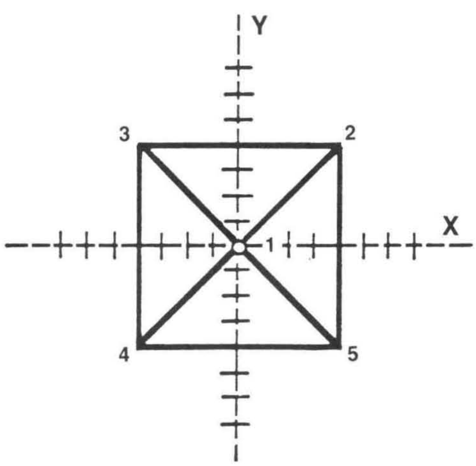


Figure 3.

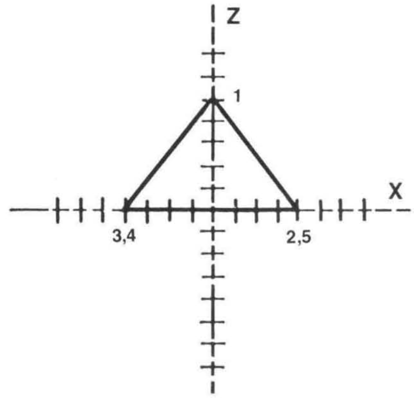


Figure 4.

POINT #	X	Y	Z
1	0	0	5
2	4	4	0
3	−4	4	0
4	−4	−4	0
5	4	−4	0

Figure 5.

LINE #	FROM POINT	TO POINT
1	1	2
2	1	3
3	1	4
4	1	5
5	2	3
6	3	4
7	4	5
8	5	2

Figure 6.

Solid States for the 520ST has some great potential. With the 68000's speed, the ST's fast line-drawing capability and some assembly language programming, it should be possible to have fluid real-time rotation—showing just how powerful the ST really is.

James Luczak bought his first Atari in 1980 and has, since 1979, written programs in BASIC, C, LOGO, FORTH and Action!, plus 6502 assembly. He enjoys writing dedicated database programs.

PrintMaster

UNISON WORLD INC.
2150 Shattuck Ave., Suite 902
Berkeley, CA 94704
(415) 848-6666
$39.95

PRINTMASTER ART GALLERY I
$29.95

THE CREATIVE PRINTMASTER
Book $13.95

by Arthur Leyenberger

PrintMaster is a graphics program that allows you to use your Atari ST computer and printer to design personalized greeting cards, signs, stationery, calendars and banners.

It has a series of multiple menus to guide you through the creation and printing processes and is very easy to use. If a mistake is made, you can easily back up through the series of selections, change an item, then scoot right back down to where you left off—and continue with the program.

If you're familiar with **The Print Shop**, an 8-bit graphics program from Broderbund, then you have a good idea of how **PrintMaster** works and what it can do. However, **PrintMaster** has a number of features that surpass those of **The Print Shop**. Before I tell you why it's better, let me briefly describe the program.

PrintMaster was first developed for the IBM PC and compatibles; its ST implementation is basically the same. Once the program is run, you select what you want, to create from a list in the main menu.

In addition to the items mentioned above, you can also create your own picture with the graphic editor, and initialize the program to use the particular printer you own.

Since the program's menu driven, selecting an item like "Sign" leads to another menu. Next, you must choose the type of border you want. The choices are listed, as well as shown graphically, and one choice allows you to design your own border (border designs include thick and thin lines, cars, footprints, ants, and more).

Next, you select a graphic, either from the more than 100 provided with the program, or from an optional Art Gallery Disk (another 100, sold separately). Again, a list of graphic names is displayed, and the graphic is actually shown as the cursor's moved over the name. Once the graphic has been chosen, you select size, position and the font you want to use.

Eight different fonts are available, and each can be used in one of three sizes. In addition, solid, outline or "3-D" styles can be picked.

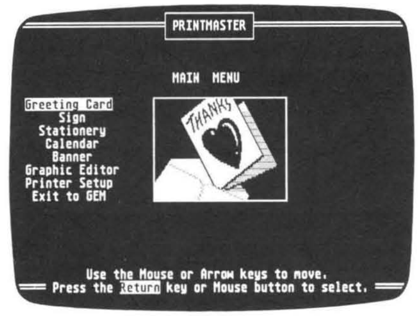
Enter the desired message, and the entire sign is displayed on-screen for your perusal. If all is to your liking, you give the signal—the print jumps to life. This example showed how a sign may be created, but all the possibilities are just as easy to work up.

If you want to design your own graphic, or if you'd like to modify an existing one, the Graphic Editor is used. With this editor, you can draw and erase lines, or invert and flip whole pictures. All available commands are presented on-screen, next to the drawing area.

Compared to **The Print Shop**, **PrintMaster** for the ST has a number of advantages. First, it runs on the ST; **The Print Shop** doesn't, at least not yet.

More important: with **PrintMaster**, you can see the entire design on the screen, before it's printed. Also, unlike **The Print Shop**, once you've created your own unique sign, banner, or whatever, you can save that design to a disk file.

Finally, **Printmaster** has better quality graphics output on the printer (at least, it does on my Epson FX-80). And **PrintMaster** lets you create and print calendars, which **The Print Shop** does not.



PrintMaster.

There's a book available from Unison World, to help you get the most out of the program. Called *The Creative PrintMaster*, this 200-page work gives tips and ideas on how to select and use the various typestyles in your designs. It helps you in laying out your work, selecting borders, using color, picking the proper paper, and creating ads and newsletters.

The majority of the book is generic information, so if you're a user of **The Print Shop**, **The Newsroom**, or another, similar graphic design program, this information will be of help to you.

Interestingly, Broderbund is attempting to sue Unison World, saying the program is a knock-off of **The Print Shop**. Well, it's somewhat similar, but has more features and is better implemented.

All in all, **Printmaster** is an excellent program that will give you hours of rewarding fun.

WHAT IS ST-CHECK?

Most program listings in **ST-Log** are followed by a table of numbers appearing as DATA statements, called "ST CHECKSUM DATA." These numbers are to be used in conjunction with **ST-Check** (which appeared in **ANALOG Computing/ST-Log** issue 41).

ST-Check (written by Clayton Walnum) is designed to find and correct typing errors when readers are entering programs from the magazine. For those readers who would like copies of the article, you may send for back issue 41 ($4.00).

ANALOG Computing/ST-Log
P.O. Box 625, Holmes, PA 19045

Hippo Backgammon

HIPPOPOTAMUS SOFTWARE, INC.
985 University, Suite 12
Los Gatos, CA 95030
(408) 395-3190
$39.95 Color or B/W

by Clayton Walnum

Back when I was a tot, there were two activities that filled the greater portion of my days. The first of these, an innate and mysterious ability discovered shortly after birth, was surely the most controversial. Simply known among us rugsters as "discovering the limits," it was, to my delight, a skill that could evoke some of the most amazing sounds from the throats of my parents. Unfortunately, this pastime had serious side effects, not the least of which were watering eyes, a running nose and a well pinked fanny.

But once the eyes and nose had dried, and the glare from my glowing posterior had ebbed (that portion of my anatomy was so often in its luminous state I feared it radioactive), I resorted to my second favorite activity: playing games.

The best games were, of course, those easily learned, yet able to be played at whatever strategic level best suited the player. Checkers was one such game. And on the back of every checkerboard was the other game, the one only a few kids knew how to play: backgammon.

Now, from the viewpoint of a parent, backgammon is a wholly different experience. Just finding the game usually means placing each child beneath bright lights and interrogating them ruthlessly. Once the box has been located, the pieces belonging in it must be gotten together—and pried apart (we don't use superglue in our house; lollipops do the trick quite nicely, thank you). Sometimes you have to sit back and wonder if playing the game is worth the effort.

Ah, but there *is* hope for the child-infested household. If you have a 520ST, you can get a copy of **Hippo Backgammon**. Let the young people do what they will with the old-fashioned plastic and cardboard version. You'll never lose the pieces to this one.

The first thing you'll notice when you load this game is the stunning graphics.

Hippo Backgammon.

They've done a marvelous job of presenting a "lifelike" playing board. Just watching the game set itself up is interesting. Each piece floats up from its resting place, turns about and places itself at its starting position, all with a 3-D effect.

The game play is as simple as one could expect from a computer game. Pieces are picked up and moved about the board with the mouse as freely as with your hand. The dice are rolled automatically by the computer, and when your turn is finished, clicking on the dice with the mouse passes them to your opponent.

Your opponent is one of two "robots" supplied. You may choose to play either robot A or robot B. If you're feeling particularly lazy, you can sit back, pit the robots against each other and watch them battle it out. Unfortunately, there's no option allowing you to play against another human. Strange, since this would be an easy thing to implement.

The robots may be set up as expert, medium, or novice (the default is expert) from files supplied on the disk, or you may customize each robot's abilities by editing the "cortex" or "neurons" for the robot you wish to change.

The cortex supplies the robot's overall strategy, and the neurons contain data determining the importance of each point on the board (a backgammon board is made up of 24 points, triangular spaces upon which the pieces move). Once you've modified the robot to your liking—no easy feat, since the strategies are quite complex—you may save your creation to disk for later use.

The top of the screen sports a series of drop-down menus to allow access to the game's options. Utilizing these menus, you may save a game in progress, load a previously saved game, load or save robots, edit robots, choose your opponent, change the screen colors, change sides, clear the score, view the manual, and quit to the desktop. There's also an option to let you set up the board in any configuration you wish.

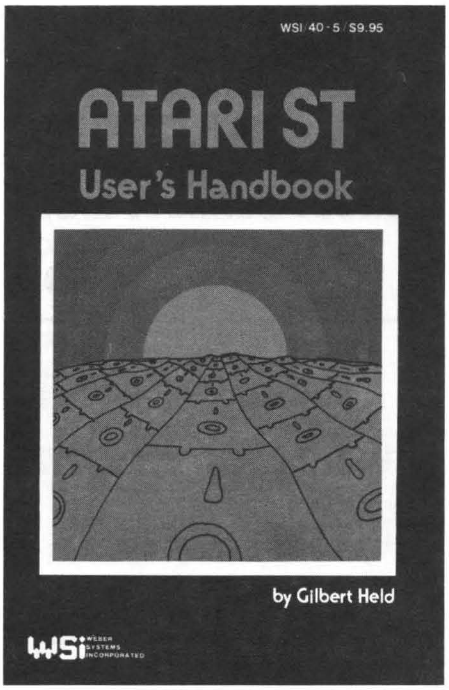
The program's not perfect. One annoying oversight: you can pass the dice to your opponent, whether or not you've completed your move. This stems from the fact that, according to the rules of backgammon, a piece on the bar must be placed back in the game before other pieces may be moved.

Depending on your roll, this can leave you unable to move. Rather than check for this specific circumstance, **Hippo Backgammon** lets you pass the dice at any time, legal or not.

Hippo Backgammon is a playable and interesting version of this popular game. A word to the wise: be sure you get the latest version. Earlier releases had a lot of bugs, not the least of which was an inability to set up the board manually.

Insist on testing the game before you lay out your cash. Select the "game board" option from the edit menu, then try to set up the board. If it works, you have the latest version. This may seem like a lot of bother, but it'll be worth it. After all, there's only one alternative: find a bottle of sulphuric acid and hope *that* can eat its way through the lollipop gunk.

ST NEWS!



USER'S HANDBOOK

Weber Systems Incorporated has just published the ***Atari ST User's Handbook***, by Gilbert Held.

The book covers a lot of ground, with chapters on: installation of the ST computer, information on disks and disk drives, using the operating system, Logo programming, keyboard and mouse usage and communications.

To ease the first-time user into understanding the ST, there are examples used throughout the book, along with diagrams, screen dumps for many situations you'll encounter and a few photos.

Chapters other than those mentioned above cover the use of the VT52 emulator provided by Atari and give you a fairly complete rundown on the fundamentals of the ST. The book even supplies some "history" on Atari's development of the new line.

The ***Atari ST User's Handbook*** is 159 pages. Give the bookstore ISBN 0-9388-62-40-5 to order your copy.

The ***Handbook*** is selling for $9.95. For more information, contact Weber Systems, Incorporated, 8437 Mayfield Road, Chesterland, OH 44026.

CIRCLE #183 ON READER SERVICE CARD

LET YOUR FINGERS DO THE TALKING

First Byte, Inc. has announced the development of its unlimited, software-only speech synthesis technology for the Atari ST series.

Requiring no hardware, this new version, **Smoothtalker**, accepts plain English text from either the keyboard or a text file. Pronunciation can be switched between an adult male and a female voice, in clear, smooth sound. Potential applications consist of education, accounting, scientific, tutorial and data entry proofreading, among others.

For additional information, contact First Byte, Inc., 2845 Temple Avenue, Long Beach, CA 90806.

CIRCLE #182 ON READER SERVICE CARD

PROFESSIONAL DATA MANAGEMENT

Talisman is a full relational database system designed to utilize the ST's power, huge RAM and crisp graphics. Touted as easy to use, its data input routines are described as being simple and hassle-free.

The **Talisman** is also virtually free of constraints, such as field lengths, number of records, or number of files. Its special design pushes these limitations beyond the point where most users would encounter them. This database is integrated with GEM and is available in three language versions.

For more information, contact Talent Computer Systems, Curran Building, 101 St. James Road, Glasgow G40NS, Scotland, U.K. — (041) 552-2128.

CIRCLE #181 ON READER SERVICE CARD

GEM DATABASE

Regent Base is a full-function relational GEM database written specifically for the ST. Regent Software describes the program as being ideal for home or small business, with modules available for checkbook balancing, general ledger, accounts receivable and more.

Regent Base was written in machine language, making it very fast and efficient. Another plus for it is an ability to merge data with **Regent Word II**, Regent's word processor, allowing mail-merge functions.

A sorting program permits up to three fields to be processed, recognizing information in **Regent Base** tables. The database also recognizes memory upgrades, making sorting and query functions even faster. Reports generation is available, using custom formats. These may include multiple type styles, subscripting, superscripting, underlining, bold and elongated type. Over fifteen printers are compatible.

Price is $99.95, from Regent Software, 7131 Owensmouth, Suite 45A, Canoga Park, CA 91303 — (818) 883-0951.

CIRCLE #184 ON READER SERVICE CARD

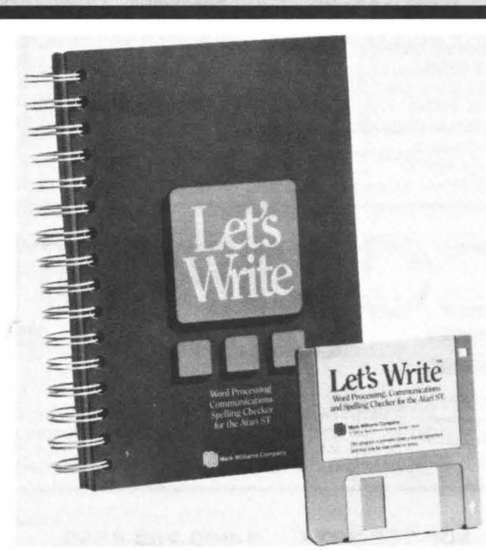
LET'S WRITE

The Mark Williams Company has begun shipping a new professional text processor that goes by the name of **Let's Write**.

This program features a spelling checker and a communications utility. The latter allows your ST to be employed as a remote terminal to larger systems, with the ability to transfer text and binary files over the phone lines.

Additional features of the program include a full-screen editor with its own formatting language and the capability to display up to eleven windows. Users can also design their own keyboard macros for frequently used phrases.

Let's Write is available for $79.95. Contact Mark Williams Company, 1430 W. Wrightwood Ave., Chicago, IL 60614 — (312) 472-6659.

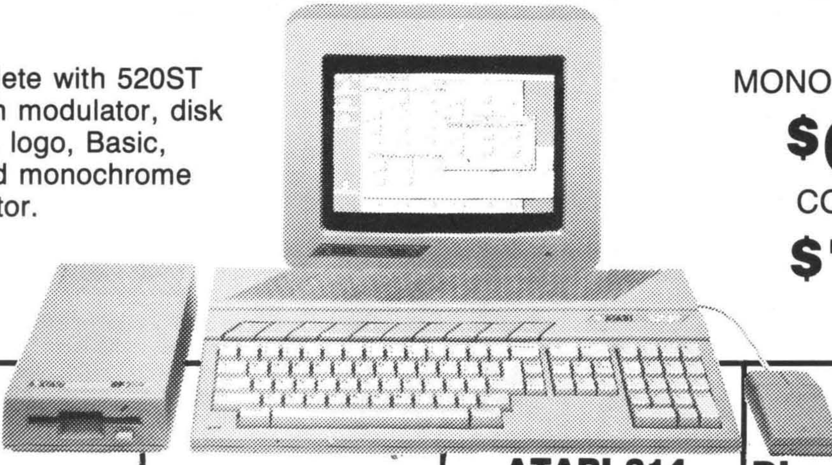


CIRCLE #180 ON READER SERVICE CARD

ATARI SPECIALISTS

ATARI 520ST SYSTEM PACKAGE

Comes complete with 520ST computer with modulator, disk drive, mouse, logo, Basic, 1st Word, and monochrome or color monitor.



MONOCHROME SYSTEM
$649.00
COLOR SYSTEM
$769.00

SUPRA DISK
20 Meg HARD DRIVE
$829.00

SUPRA MODEM
MODEL 1200ST $159.00

ATARI 314
1 Meg Double Sided DISK DRIVE
$219.00

ANCHOR 520
Direct Connect MODEM
300/1200 Baud
$149.00

— PRINTERS —

CITIZEN
MSP-10 (80 col.) $279.00
MSP-15 (132 col.) $389.00
MSP-20 (80 col.) $349.00
MSP-25 (132 col.) $509.00

C.ITOH
Prowriter 1550P $349.00
Starwriter 10-30 ..... $399.00

EPSON
Homewriter 10, LX80 ..... CALL
FX85, FX286, RX100 ..... CALL
SQ2000, HI80, HS80, AP80 ..... CALL
LQ800, LQ1000 ..... CALL

JUKI
6000 Letter Quality ..... CALL
6100 Letter Quality ..... CALL
6200 Letter Quality ..... CALL
6300 Letter Quality ..... CALL
6500 Letter Quality ..... CALL
5510 Dot Matrix ..... CALL
5510 Color ..... CALL

OKIDATA
182, 183, 192, 193, 2410, 84 ..... CALL

LEGEND
808 Dot Matrix 100 cps $179.00
1080 Dot Matrix 100 cps $259.00
1380 Dot Matrix 130 cps $289.00
1385 Dot matrix 165 cps $339.00

NEC
3000 Series $779.00
8000 Series $1099.00
ELF 360 ..... $399.00
Pinwriter 560, 660, 700 ..... CALL

PANASONIC
KX1091 ..... $259.00
KX1092 ..... $389.00
KX1592 ..... $469.00

TOSHIBA
P321 (80 column) $489.00
P341 (132 column) $799.00
P351 (132 column) $1049.00

STAR
NX/SB/SD/SR Series ..... CALL
Powertype Letter Quality ..... CALL

SUBLOGIC
Flight Simulator II
$37.99

SOFTWARE

ACTIVISION
Hacker $29.99
Borrowed Time $32.99

BATTERIES INCLUDED
D.E.G.A.S. $27.99

HABA/ARRAYS
Hippo-C $44.99
Business Letters $29.99
Printmaster $29.99
Write Your Own Will $29.99
Haba Writer $44.99
Habadex Phonebook $29.99
Habamerge $12.99
Checkminder $39.99
Mail Room Manager $39.99

INFOCOM
Cutthroats $29.99
Deadline $34.99
Enchanter $29.99
Hitchhiker's Guide $29.99
Infidel $34.99
Planetfall $29.99
Sea Stalker $29.99
Sorcerer $34.99
Starcross $34.99
Suspect $29.99
Suspended $34.99
Wishbringer $29.99
Witness $29.99
Zork I $29.99
Zork II $29.99
Zork III $29.99

MINDSCAPE
Deja Vu $37.99

MIRAGE
Express $34.99

MUSE SOFTWARE
Final Word $99.99
Hex $27.99
PC Intercom $79.99

O.S.S.
Personal Pascal $49.99

PENQUIN SOFTWARE
Crimson Crown $29.99

SIERRA-ON-LINE
Ultima II $39.99
King's Quest $37.99

V.I.P
Professional $129.00

ATARI 520ST Hardware/Peripherals

Atari 520 CPU $369.00
Atari 124 Monochrome Monitor $189.00
Atari 1424 14" Color Monitor $269.00
Atari 354 Single Sided Drive $179.00
Haba 10 Mb Hard Drive $669.00

BLANK DISKETTES & ACCESSORIES

AMARAY
Disk Tub 3½" $9.99

ELEPHANT
(10) 3½" SS $24.99

MAXELL
(10) 3½" SS $18.99
(10) 3½" DS $29.99
(5) 3½" SS w/Flip'n File $9.99

CURTIS Surge Suppressors
Emerald $39.99
Ruby $59.99
Diamond $29.99
Safety Strip $19.99
Sapphire $49.99

COMPUTER MAIL ORDER

CALL TOLL-FREE 1-800-233-8950

TELEX: 5106017898

477 East Third Street, **Dept. B708,** Williamsport, PA 17701

POLICY: Add 3% (Minimum $7.00) shipping and handling. Larger shipments may require additional charges. Personal and company checks require 3 weeks to clear. For faster delivery use your credit card or send cashier's check or bank money order. Pennsylvania residents add 6% sales tax. All prices are subject to change and all items are subject to availability. Defective software will be replaced with the same item only. Hardware will be repaired or replaced at our discretion within the the terms and limits of the manufacturer's warranty. We cannot guarantee compatibility. All sales are final and returned shipments are subject to a restocking fee.

— CANADIAN ORDERS —

1-800-268-3974 Ontario/Quebec
1-800-268-4559 Other Provinces
All prices shown are for U.S.A. orders. Call the Canadian Office for Can. prices.
1-416-828-0866 In Toronto
Telex:06-218960
2505 Dunwin Drive,
Mississauga, Ontario
Canada L5L1T1

CIRCLE #129 ON READER SERVICE CARD

AN INTRODUCTION TO Logo FOR THE ST

by James Luczak

As all of you are probably aware by now, the Atari ST comes equipped with two programming languages, ST BASIC and Digital Research's Logo for the ST. While both have been around for quite a while, BASIC is by far the more popular of the two. Since the ST comes with both BASIC and Logo, eventually people who've never programmed with Logo are going to give it a try.

What kind of language is it? Well, Digital Research's Logo can be used to draw with "turtle graphics" and to process words and lists. The first, drawing, is the language's primary function.

What is turtle graphics? The phrase refers to a method that will draw a line in a specified direction (in degrees), for a specified number of units.

The coordinate system used in Logo is a little different from the coordinate system of BASIC. Instead of coordinates *0 0* being at the upper left-hand corner of the screen, coordinates *0 0* in Logo put you in the very center of the screen. Coordinates below the screen's center and to the left of center are represented by negative numbers. Coordinates above center and to the right are represented by positive numbers.

How do you program in Logo? As in BASIC, you can use the "immediate" or "program" mode. In the immediate mode, instructions are carried out as soon as you press the RETURN key. For example, if you type *FORWARD 20 RIGHT 90 FORWARD 20*, the turtle will move forward 20 units, make a 90-degree turn, then move forward another 20 units, as soon as you press RETURN.

In the program mode, unlike BASIC, Logo does not use line numbers. Instead you must define what's called a "procedure." Here's an example of a procedure that will clear the screen and draw a square.

```
TO SQUARE

CLEARSCREEN
FORWARD 20 RIGHT 90
FORWARD 20 RIGHT 90
FORWARD 20 RIGHT 90
FORWARD 20

END
```

Now, if you type the word *SQUARE*, the screen will be cleared and a square will be drawn. Of course, you could use the procedure SQUARE within other procedures.

When learning a new programming language, I've found the easiest way is to determine how to perform commonly used simple functions, such as *A=A+1, PRINT "HELLO"*, etc. It's also a good idea to type in known, working programs using the new language. Having even a few program listings in the new language is a great help, because they demonstrate how it performs various functions.

Table 1 (BASIC vs Logo) shows a number of simple functions in BASIC and illustrates how the same functions can be performed in Logo.

Table 2 (Working with Logo) shows how to perform various housekeeping tasks with Logo. By "housekeeping," I mean things like clearing old programs from memory, getting your program to list to the printer, and so forth.

Table 3 (Helpful hints for programming Logo) points out a number of items the Sourcebook for Atari Logo doesn't make quite clear.

Listing 1 is a numbers demo program. To run the demo, set the screen to low resolution. After you've entered the program, type *NUMBERS*. The program will display a series of ten numbers, displayed five times in a variety of positions, colors and line widths.

Listing 2 gives you a designs demonstration. Again, set the screen to low resolution. After you've entered the program, type *DESIGN*. The program will display the same design in eleven different positions in different colors, then randomly switch colors.

Before running the demostration programs, set the graphics display window to full screen. To do this, move the mouse so the arrow is on the shaded area where *GRAPHICS DISPLAY* is written, then click the mouse. Next, move the mouse to the full box icon and click it. (The full box icon can be found in the upper right-hand corner of the screen.)

Like any programming language, Logo has lets you do most things in any of several different ways. The methods I've described may not be the best or most efficient ways of programming in Logo, but I hope that they'll make starting off in Logo a little easier for you.

(Figures and listings on next page)

Table 1. — BASIC vs Logo.

BASIC
```
A = A + 1
B = B - 1
X = INT (Y)
X = INT (Y * Z)
X = ABS (Y)
X = ABS (Y / Z)
```
LOGO
```
MAKE "A :A + 1
MAKE "B :B - 1
MAKE "X INT :Y
MAKE "X INT (:Y * :Z)
MAKE "X ABS :Y
MAKE "X ABS (:Y / :Z)
```

BASIC
```
PRINT "HELLO"
PRINT "ONE";"TWO"
```
LOGO
```
PRINT [HELLO]
TYPE [ONE] PRINT [TWO]
```
NOTE: In Logo, the "PRINT" command inserts a carriage return at the end of the list being printed. The "TYPE" command prints the list without inserting a carriage return.

BASIC
```
FOR X = 1 TO 10
PRINT "HELLO"
NEXT X
```
LOGO
```
REPEAT 10 [PRINT [HELLO]]
```

BASIC
```
IF X = 1 THEN Y = 7
```
LOGO
```
IF :X = 1 [MAKE "Y 7]
```

BASIC
```
10 IF X = 10 THEN GOTO 30
20 PRINT "X <> 10" : GOTO 40
30 PRINT "X = 10"
40 PRINT "ALL DONE"
```
LOGO
```
IF :X = 10 [GO "L1]
PRINT [X <> 10] GO "L2
LABEL "L1 PRINT [X = 10]
LABEL "L2  PR [ALL DONE]
```

BASIC
```
DIM A(3)
A(1) = 10
A(2) = 20
A(3) = 30
FOR X = 1 TO 3
PRINT A(X)
NEXT X
```
LOGO
```
MAKE "A [] MAKE "CNT 1
MAKE "A LPUT 10 :A
MAKE "A LPUT 20 :A
MAKE "A LPUT 30 :A
```
NOTE: The "LPUT" command puts the incicated value at the end of the list. The *L* in *LPUT* stands for "Last." The "FPUT" command puts the indicated value at the beginning of the list. The *F* in *FPUT* stands for "first."

Table 2. — Working with Logo.

TO:	Display a list of all procedures in memory.
TYPE:	*PROCLIST*
TO:	Display a list of all procedures and their definitions.
TYPE:	*POALL*
TO:	Display a single procedure and its definition.
TYPE:	*PO "procedure_name*
TO:	Edit a single procedure.
TYPE:	*ED "procedure_name*
TO:	Edit all procedures in memory.
TYPE:	*EDALL*
TO:	Erase a single procedure from memory.
TYPE:	*ER "procedure_name*
TO:	Erase all procedures from memory.
TYPE:	*ERALL*
TO:	Print text to a printer.
TYPE:	*COPYON*
TO:	Stop print to printer.
TYPE:	*COPYOFF*
TO:	Display a list of all Logo files on a disk.
TYPE:	*DIR*
TO:	Clear the graphics screen.
TYPE:	*CS*
TO:	Clear the text screen.
TYPE:	*CT*

The "CS" command erases the screen and places the turtle at position *0 0*, heading *0* with pen down. The "HOME" command places the turtle at position *0 0*, heading *0* without erasing the screen.

Table 3. — Helpful hints for programming Logo.

Variables in Logo can be either *numeric* or *alphabetical*. For example:
```
MAKE "A 10
MAKE "A [HELLO]
```
A file loaded into memory is "appended" to memory. If, for example, you've defined procedures named *ONE* and *TWO*, and then load a file containing procedures named *THREE* and *FOUR*, you'll have procedures *ONE*, *TWO*, *THREE* and *FOUR* in memory. If you want to clear memory before loading another file, use the "ERALL" command.

To employ variables to set parameters within brackets in a command, use the following methods.
```
COMMAND____SETPOS [-100 50]
```
To use variables for the values of *−100* and *50*, use the following form:
```
MAKE "X -100 MAKE "Y 50
SETPOS SE :X :Y
```
```
COMMAND____SETPAL 3 [500 600 700]
```
To use variables for the values of *500*, *600* and *700*, use the following form:
```
MAKE "A 500 MAKE "B 600 MAKE "C 700
SETPAL 3 (SE :A :B :C)
```
In the command *SETPAL 3 [500 600 700]*, the numbers within the brackets *(500, 600* and *700)* are referred to as the "RGB list." The first number represents the color red. The second represents green, and the third, blue. Any value between *0* and *1000* can be used for each color. However, only the "most significant digit" of the number is used to set the intensity of color. Figure 1 is a color intensity chart for the RGB list.

Jim Luczak has, since 1979, written programs in BASIC, C, Logo, FORTH and Action!, plus 6502 assembly. He enjoys writing dedicated database programs.

Figure 1.
Color intensity chart for RGB list.

0 = LOW INTENSITY   8 = HIGH INTENSITY

Value	Intensity
000 - 099	0
100 - 199	0
200 - 299	0
300 - 399	1
400 - 499	2
500 - 599	3
600 - 699	4
700 - 799	5
800 - 899	6
900 - 999	7
1000	8

As an example, in the command "SETPAL 3 [500 600 700]," the first number *(500)* indicates that the color red is at an intensity of *3*. Any value between *500* and *599* could be used *(545, 539, or 567)*, and the intensity of red would remain *3*. Remember that the M.S.D. determines the intensity. To increase the red's intensity to, let's say, *7*, we would have to change the first number of the RGB list from *500* to a value between *900* and *999*.

The *3* which appears before the brackets is the "color index," which designates the color register whose red, green and blue constituents are being set by the command.

In the command "SETLINE [1 2 3]," the three numbers within the brackets *(1, 2* and *3)* represent, respectively, line style, line width and line color. Line style can be any number between *1* and *7*. Figure 2 shows the line style chart.

Figure 2. — Line style chart.

STYLE NUMBER	DESCRIPTION
1	Solid line
2	Long dash line
3	Dot line
4	Dot short dash line
5	Short dash line
6	Dash half-dash dot line
7	User defined line

Line width can be any value between *1* and *39*, if the line style is style 1. The higher the line width value, the wider the line will be. To use line styles 2 through 7, the line width must not exceed *2*. If the line width value exceeds *2*, the line style value will default to *1* (solid).

To activate the "FILL" function, include the following code in your procedure:

```
MAKE "GFILL "TRUE
```

Like this:

```
TO DRAWCIRCLE
CIRCLE [0 0 20]
MAKE "GFILL "TRUE
FILL
END
```

The above procedure will draw a circle at coordinates *0 0* with a radius of *20* and fill the circle. By making GFILL false, the procedure will draw the circle, but will not fill it, even though the "FILL" command is in the procedure.

***Attention:* In these listings, the exclamation points at the end of program lines shouldn't be typed in. They are there to indicate that the statement wraps around to the next line.**

Listing 1.
Logo listing.

```
TO ZERO :SIZE
R PU BACK :SIZE PD
RT 45 FD :RO
RT 45 FD :SIZE
RT 45 FD :RO
RT 45 FD :SIZE * 3
RT 45 FD :RO
RT 45 FD :SIZE
RT 45 FD :RO
RT 45 FD :SIZE * 3
PU FD :SIZE PD
END

TO ONE :SIZE
R PU BACK :SIZE RT 90
FD :SIZE LT 45 PD
FD :RO
RT 135 FD :SIZE * 5
RT 90 FD :SIZE
BACK :SIZE * 2 PU
RT 90 FD :SIZE * 5
LT 90 FD :SIZE * 3
RT 90 PD
END

TO TWO :SIZE
R PU BACK :SIZE PD
REPEAT 2 [RT 45 FD :RO RT 45 FD :SIZE!
]
RT 45 FD :RO * 3
LT 135 FD :SIZE * 3
PU LT 90 FD :SIZE * 5
LT 90 FD :SIZE * 3
RT 90 PD
END

TO THREE :SIZE
R PD
RT 90 FD :SIZE * 3
RT 135 FD :RO * 2
LT 135 FD :SIZE
REPEAT 2 [RT 45 FD :RO RT 45 FD :SIZE!
]
RT 45 FD :ROOT
PU RT 45 FD :SIZE * 4 PD
END

TO FOUR :SIZE
PD RT 180 FD :SIZE * 3
LT 90 FD :SIZE * 3
BACK :SIZE
LT 90 FD :SIZE * 2
BACK :SIZE * 4
PU FD :SIZE * 5 LT 90
FD :SIZE * 2 RT 90 PD
END

TO FIVE :SIZE
R PD
RT 90 FD :SIZE * 3
BACK :SIZE * 3
RT 90 FD :SIZE * 2
LT 90 FD :SIZE * 2
REPEAT 2 [RT 45 FD :RO RT 45 FD :SIZE!
]
RT 45 FD :RO
PU RT 45 FD :SIZE * 4 PD
END

TO SIX :SIZE
R PU
RT 90 FD :SIZE * 3
RT 90 FD :SIZE
RT 135 PD FD :RO
```

```
LT 45 FD :SIZE
LT 45 FD :RO
LT 45 FD :SIZE * 3
REPEAT 3 [LT 45 FD :RO LT 45 FD :SIZE!
]
LT 45 FD :RO
PU RT 135 FD :SIZE * 3 PD
END

TO SEVEN :SIZE
R PD
RT 90 FD :SIZE * 3
RT 135 FD :RO * 2
LT 45 FD :SIZE * 3 PU
RT 180 FD :SIZE * 5
LT 90 FD :SIZE RT 90 PD
END

TO EIGHT :SIZE
R PU RT 90 FD :SIZE PD
FD :SIZE
RT 45 FD :RO
RT 90 FD :RO
RT 45 FD :SIZE
REPEAT 4 [LT 45 FD :RO LT 45 FD :SIZE!
]
RT 45 FD :RO
RT 90 FD :RO
PU LT 135 FD :SIZE RT 90 PD
END

TO NINE :SIZE
R PU
RT 90 FD :SIZE * 3
RT 90 FD :SIZE RT 135 PD
FD :RO
REPEAT 3 [LT 45 FD :SIZE LT 45 FD :RO!
]
LT 45 FD :SIZE
RT 180 FD :SIZE * 3
RT 45 FD :RO
RT 45 FD :SIZE
RT 45 FD :RO
PU RT 45 FD :SIZE * 4 PD
END

TO NUMB :LIST :SIZE
IF :LIST = [] [STOP]
RUN SE FIRST :LIST :SIZE
PU RT 90 FD :SIZE * 4 LT 90 PD
MAKE "COL RANDOM 15 IF :COL = 0 [MAKE!
 "COL 1]
SETPC :COL
NUMB BF :LIST :SIZE
END

TO DONUMBERS
MAKE "SIZE RANDOM 10
IF :SIZE < 5 [MAKE "SIZE 5]
MAKE "T1 1
MAKE "T2 :SIZE - 1
MAKE "T3 1
SETLINE (SE :T1 :T2 :T3)
HEADEG
NUMB SHUFFLE [ONE TWO THREE FOUR FIVE!
 SIX SEVEN EIGHT NINE] :SIZE
END

TO HEADEG
PU HOME
MAKE "HED RANDOM 360 SETH :HED
IF :HED >= 270 [MAKE "X1 RANDOM -150 !
MAKE "Y1 RANDOM -100 GO "L1]
IF :HED >= 180 [MAKE "X1 RANDOM 150 M!
AKE "Y1 RANDOM -100 GO "L1]
IF :HED >= 90 [MAKE "X1 RANDOM 150 MA!
KE "Y1 RANDOM 100 GO "L1]
MAKE "X1 RANDOM -150 MAKE "Y1 RANDOM !
100
LABEL "L1 PU SETPOS (SE :X1 :Y1) PD
END

TO R
MAKE "RO :SIZE * 1.41421
END

TO PALETTE
SETPAL (3 + RANDOM 13) (SE RANDOM 100!
0 RANDOM 1000 RANDOM 1000)
END

TO NUMBERS
CS HT
REPEAT 5 [DONUMBERS]
LABEL "L3 PALETTE GO "L3
END

MAKE "GFILL "FALSE
MAKE "COL 13
MAKE "ST3 4
MAKE "ST2 7
MAKE "Y 29
MAKE "ST1 1
MAKE "X 7
MAKE "HED 213
MAKE "ROOT 8.485257
MAKE "B -23
MAKE "A -100
MAKE "SIZE 5
MAKE "SZ 32
MAKE "RO 7.071048
MAKE "Y1 -65
MAKE "X1 12
MAKE "T3 1
MAKE "T2 4
MAKE "T1 1
```

●

Listing 2.
Logo listing.

```
TO BASE
FD :S
IF REMAINDER :K :F = 0 [RT :A2 GO "L1!
]
RT :A
LABEL "L1 MAKE "K :K + 1
END

TO SETDESIGN
MAKE "SP [-250 90 -100 90 50 90 200 9!
0 -250 -150 -100 -150 50 -150 200 -15!
0 -175 -25 -25 -25 125 -25]
MAKE "SF [-200 110 -50 110 100 110 25!
0 110 -200 -130 -50 -130 100 -130 250!
 -130 -125 -5 25 -5 175 -5]
MAKE "X ITEM :CNT :SP MAKE "Y ITEM (:!
CNT + 1) :SP
MAKE "X1 ITEM :CNT :SF MAKE "Y1 ITEM !
(:CNT + 1) :SF MAKE "CNT :CNT + 2
PU SETPOS (SE :X :Y) PD
REPEAT 40 [BASE]
PU SETPOS (SE :X1 :Y1) PD FILL
MAKE "SL RANDOM 15 MAKE "SL1 RANDOM 1!
5
LABEL "K2 IF :SL = :BG1 [MAKE "SL RAN!
DOM 15 GO "K2]
LABEL "K3 IF :SL1 = :BG1 [MAKE "SL1 R!
ANDOM 15 GO "K3]
SETLINE (SE :SX :SX :SL) SETFILL (SE !
:SX :SX :SL1)
END

TO PALETTE
```

```
SETPAL (3 + RANDOM 13) (SE RANDOM 100!
0 RANDOM 1000 RANDOM 1000)
END

TO DESIGN
MAKE "S 25 MAKE "A 45 MAKE "A2 -45 MA!
KE "F 5 MAKE "K 0 MAKE "P 2
MAKE "CNT 1 MAKE "GFILL "TRUE MAKE "S!
X 1 MAKE "BG1 RANDOM 15
SETFILL [1 1 3]
SETBG :BG1 SETPC 1 SETZOOM 0.5 HT CS
REPEAT 11 [SETDESIGN]
LABEL "K1 PALETTE GO "K1
END

MAKE "A2 -45
MAKE "GFILL "TRUE
MAKE "CNT 23
MAKE "YP2 42
MAKE "YP1 42
MAKE "XP2 108
MAKE "Y -25
MAKE "XP1 -31
MAKE "X 125
MAKE "S 25
MAKE "P 2
MAKE "SL1 2
MAKE "K 440
MAKE "F 5
MAKE "A 45
MAKE "BG1 3
MAKE "SX 1
MAKE "YP 10
MAKE "XP [-250 150 -100 150]
MAKE "SP [-250 90 -100 90 50 90 200 9!
0 -250 -150 -100 -150 50 -150 200 -15!
0 -175 -25 -25 -25 125 -25]
MAKE "SL 7
MAKE "SF [-200 110 -50 110 100 110 25!
0 110 -200 -130 -50 -130 100 -130 250!
 -130 -125 -5 25 -5 175 -5]
MAKE "PC 3
MAKE "Y1 -5
MAKE "X1 175
```

●

C-MANSHIP

by Clayton Walnum

Good news. . . Megamax C is now available. Though I haven't had much time with it yet, it seems to be everything promised—and then some.

The compiler's about seven times faster than the DRI system. For instance, using the DRI compiler, a 28K source file took about fourteen minutes to compile and link. The same file compiled and linked in slightly over two minutes with Megamax.

The program comes with a hefty loose-leaf manual, including all operating instructions, as well as good descriptions of the GEM and BIOS routines. The package contains not only the compiler and linker, but an editor, a resource construction program, a disassembler, a "code improver," and a shell program to neatly tie the whole thing together, allowing easy access to all of Megamax's features.

If you've been holding off on getting your C development system because the DRI offering was too expensive and you didn't want to compromise on quality, here's your chance. For $195, you can make both yourself and your ST very happy. Watch for a complete review of this product in a future issue of **ST-Log**.

You can contact Megamax, Inc. at Box 851521, Richardson, TX 75085, or phone them at (214) 987-4931.

Mailbag.

The postman dropped an interesting comment on my desk this month. Earl Davis writes:

> At present, I have an 800XL system, but am anticipating (anxiously) getting a 520ST (or 1040ST). . . On our 8-bits, no full-fledged C seems to exist (or so I was told!), and, as a result, I took up Action! (from OSS) to get accustomed to a "C-type" language. Do you think I made the right decision?

First of all, learning a new language is never a *wrong* decision. Most programming techniques you're exposed to will apply to many other languages. With each new language, you gain not only new insight to old ideas, but skills which will make subsequent studies that much easier.

Action! is really a combination of C and Pascal. Learning to program in Action! will certainly help you, once you get that ST, in your experiments with C. Just like C, Action! uses functions as program building blocks. The lessons you'll learn from structuring your code and passing arguments between functions can be applied directly to C programming.

One of the major differences between Action! and C is that the former tends to be more readable, incorporating English words where C uses more cryptic symbols. Action!'s enhanced readability is due to its combination with Pascal. Okay, enough stalling. Back to business.

What's the point?

This month, we'll take a look at pointers. They're a decisive factor in separating the real programmers from the dabblers. Actually, though handling pointers can be confusing at times, the basic concept is quite simple. In fact, we've been using them all along, whether you're aware of it or not, whenever we referred to an array name.

What exactly is a pointer? Simply put, it's a variable containing the address of a data item we wish to access. For example, look at this line of code:

```
pointer = &var;
```


LEADER BOARD™
"Pro Golf Simulator"
By Bruce & Roger Carver
Access welcomes you to the exciting world of professional golf with LEADER BOARD. With amazingly realistic 3-D animation, golfers will face multiple 18-hole golf courses that demand strategy, total concentration and control to come in under par.
LEADER BOARD is the most realistic golf game available because it is actually a simulator that gives you a true perspective view of your golf game.
There are three levels of play so you can compete from amateur to touring professional. You'll get all the fun, aggravation and stimulation of true golf competition!
• Suggested Retail Price $39.95
• 1 to 4 Players
• Choice of Club, Distance, Type of Shot (hook, slicer, putting) and More
• Realistic Sound
• Joystick Controlled
• Multiple 18-Hole Golf Courses
• Automatic Scoring
• Disk Commodore 64/Atari 800 Atari 520ST/Amiga
• Practice Driving Range
Enhanced Graphics on the Atari St and Amiga version!
RAID OVER MOSCOW
ACCESS
BEACH-HEAD II
THE DICTATOR STRIKES BACK
BEACH-HEAD
ACCESS™ Software Incorporated
For mail orders, enclose check or money order, plus $1.00 shipping and handling, and specify machine version desired. Order by phone on VISA, MasterCharge or C.O.D. by calling (801) 298-9077.
ACCESS SOFTWARE, INC. 2561 South 1560 West Woods Cross, UT 84087
CIRCLE #132 ON READER SERVICE CARD
Also available for the Atari 800 Home Computer Series.
Suggested retail price: $34.95

After this line has been executed, pointer contains the address of var, or, to say it another way, *points* to that section of memory where the value of var is stored.

So, what's all the hoo-ha? Why not use the "&" operator and be done with all this nonsense? Because there's a subtle difference between pointer and &var. The first is a variable; the second is a constant.

Still not impressed? Okay, let me ask you a question: what makes variables so handy? Give up? We can perform mathematical procedures on variables; not so with constants.

Another advantage to pointers is that, when declared properly, they're much "smarter" than constants or run-of-the-mill variables. We'll see why in a minute.

A point of declaration.

To get full service from a pointer, the compiler needs some information, namely the type of data the pointer points to. We pass this information with the pointer's declaration.

```
int *p1;
char *p2;
float *p3;
```

The first example tells the compiler we want a pointer to an integer value. The second sets up a pointer to character data. The third points our way to floating point information.

Each of these data types (as well as others) is stored in a special way in memory. A pointer to integer won't function as we expect if we try to use it on character data. The * before the name identifies the variable as a pointer and requests "special handling" from the compiler. Don't confuse it with the multiplication operator.

Once we've declared our pointer, we have to assign it a value. We want it to point *to* something, don't we? You assign an address to a pointer in exactly the same way as you would to any other variable. Take a look at this segment of code:

```
int var, array[10];
int *p1, *p2, *p3;

p1 = &var;
p2 = array;
p3 = &array[5];
```

First, we've declared an integer variable and an integer array. Following that are the declarations for three pointers to integer. After declaration, these pointers are still useless to us. We have to assign them values—that is, addresses to point to.

In the first case, p1 is assigned the address of var (or &var). Don't forget the ampersand; without it, you'd be assigning the value of var, not its address.

In the second assignment, p2 gets the address of the first element of the array array[]. What? No ampersand? Don't tell me you've forgotten already! An array name *is* an address.

Ah, but what about the third assignment in our example? There's that address operator. No mistake here. Once you add the brackets to the array name, you're referring to the contents of an element of the array, not its address.

Just remember: the only time you don't need the address operator is when you're doing your assignment with an array name. The following two lines do exactly the same thing:

```
p1 = array;
p1 = &array[0];
```

Putting them to work.

Okay, now we've got our pointers declared and assigned addresses. Now what?

There are several operations we can perform with pointers, including: assignment, getting the address, getting a value, and incrementing or decrementing.

The first, assignment, we've already learned about. The second, getting the address, is nothing new, either. To get the address of a pointer—the place in memory where the pointer itself is stored—place the address operator in front of the pointer name:

```
adrp1 = &p1;
```

A more useful operation is getting the value the pointer is pointing to:

```
var = 5;
p1 = &var;
z = *p1;
```

In the above example, z becomes equal to var. How? Our pointer, p1, is assigned the address of the variable var. The third line is read "z gets the value of the contents of the address pointed to by p1." The asterisk is referred to as an "indirection operator," since it allows us to access data indirectly.

Of course, this is a pretty silly example. It would've been more efficient to directly assign the value of var to z (z = *var;*). But there are times when we can't get at variables in the conventional way. Remember, C passes arguments between functions by value, not address.

Take, for instance, our bubble sort program a couple of months ago. What if, instead of using an array, we had three integer variables we wanted to sort, then pass back to the calling function? The following lines show a function call and the first two lines of the function. Will it work?

```
sort(x, y, z);

sort(a, b, c)
int a, b, c;
```

Think about it for a minute. The three arguments passed to the sort function are placed in the three automatic variables a, b and c. No problem there, so we go ahead and sort the three values (code not shown), putting y into x, and z into y, and x into z—or whatever's necessary to complete the sort. Hurray! We did it.

Wrong.

We forgot one tiny detail. We now have to pass all three values back to the calling function. Any suggestions? The return() statement will allow only one argument. Looks like we're stuck.

What did we do wrong? Why is C being so obstinate? Shall we forget the whole thing and go back to BASIC? Why am I asking you all these questions, instead of just providing the answers?

The answer is (drumroll, please): pointers.

Let's change our function call to this:

```
sort(&x, &y, &z);
```

We're still passing our arguments to automatic variables, but now those variables will contain the addresses of the original three. And, to make things as efficient and elegant as possible, we're going to make those automatic variables pointers. The first two lines of our function will look like this:

```
sort(p1, p2, p3)
int *p1, *p2, *p3;
```

Now we have access to the variables from the calling function. We can switch them around any way we want, using code similar to this:

```
save = *p1;
*p1 = *p2;
*p2 = *p3;
*p3 = save;
```

In English, the above reads: "save gets the contents of the address pointed to by p1; the contents of the address pointed to by p1 get the contents of the address pointed to by p2; and so on. What we're actually doing is this:

```
save = x;
x = y;
y = z;
z = save;
```

Once we've got the variables the way we want them, we exit the function. We don't have to return any values now; we've done all our work on the variables themselves.

Incrementing and decrementing.

I stated earlier that pointers were much smarter than conventional variables. One reason is that they're mathematical whizzes. When we perform addition or subtraction on a pointer, the compiler does a lot of the work for us, taking into account the data type it's pointing to and the way that data is stored in memory.

For instance, if we add 1 to an integer pointer, we don't end up with an address 1 byte higher in memory; we actually move forward 2 bytes. The compiler knows that integers are 2-byte animals, and, if we're going to end up with a usable address, it had better be pointing to the beginning of the next integer.

Quiz time.

A character array has a beginning address of *73455*. A pointer to character, p1, has been initialized to the starting address of the array. What address will we get if we increment the pointer?

Answer: *73456*. Character data requires 1 byte of storage for each element in the array. Adding 1 to the pointer yields the address of the next element in the array. In this case, the next element is 1 byte higher in memory.

The proof.

Now's a good time to dig into Listing 1. Type it in, compile and run it. The output should look something like this:

```
+0  p1 = 71926     &p1 = 72910     *p1 = 65

+0  p1 = 71926     &p1 = 72910     *p1 = 65
+1  p1 = 71927     &p1 = 72910     *p1 = 66
+2  p1 = 71928     &p1 = 72910     *p1 = 67

+0  p2 = 71930     &p2 = 72914     *p2 = 10
+1  p2 = 71932     &p2 = 72914     *p2 = 11
+2  p2 = 71934     &p2 = 72914     *p2 = 12

Press space bar
```

The table this program prints sums up everything we've discussed about pointers. Take a look at the first line of the table. Using what you've just learned, what's the address of the letter *A* (65) in the character array array1[]? If you answered 71926, then you probably have a good basic understanding of how pointers work.

For those of you who are still confused, don't fret. It'll sink in as you get accustomed to using them. Let's go through the program and see what's going on.

Lines 2 and 3 declare the arrays and pointers, as well as initializing the arrays. Line 7 puts the address of the first element of array1[] into the character pointer p1.

Line 8 prints out the four values in our table: the amount added to the pointer, the contents of the pointer, the address of the pointer, and the contents of the address the pointer's pointing to. The first line of the table will be printed again when we get into the loop at Line 9. The reason for this is to show you that setting the pointer with the array name is equivalent to setting it with the address operator preceding an array element. In this case, we're comparing array1 with &array1[0].

Lines 9 through 12 move through array1[], using the address operator. Each pass through the loop prints one line of our table.

Lines 14 through 17 accomplish the same thing, only now we're cycling through an array of integers, incrementing the pointer with the incrementation operator instead of with the address operator.

A glimpse of macros.

Notice that, in Listing 1, we've used printf() three times, in almost exactly the same way. In fact, the only difference between them is the name of the pointer we're working with. If the programmer's voice within you is screaming that it's stupid to code the same thing three times, then listen to it. It's right. C provides us with a handy technique to avoid this type of redundant code. The technique involves the use of macros.

As with pointers, you've already been exposed to macros—though you were probably unaware of it. Every time you use the #define statement, you're setting up a macro. We've done this dozens of times over the last few months, but only in the simplest fashion. Macros can be quite complex and are powerful programming aids.

Listing 2 is a modification of Listing 1. Here, each occurrence of the printf() call has been replaced with a macro call. The macro itself is defined in Line 2. Any legal variable name can be used as a macro name.

See the parentheses? This macro contains an argument that will be passed when the macro's expanded (the substitution of the replacement string). In our example, the argument will be the pointer name we want printed in the table.

Of course, just placing the argument in the macro name isn't enough. We've got to tell the macro where we want the argument used in the expansion. In our example, every Z in the replacement string will be replaced by the argument supplied when the macro is called.

Lines 9, 13 and 17 show the macro calls. In Lines 9 and 13, p1 will be substituted into the replacement string, and the code generated by the compiler preprocessor will look identical to Line 8 in Listing 1. In Line 18, p2 will be substituted.

Changes for Megamax.

There's little doubt that Megamax C will be a popular development system. For that reason, I'm going to try to keep this column compatible with both the DRI and Megamax compilers—or, at least, note the differences when they pop up.

At this time, I don't know if that'll be practical in the long run, or not. Once we get more involved with GEM, the differences between the two compilers may force us to make a choice. I can't promise anything for the future. We'll just have to see what happens.

Some differences I noted this month involved the function getchar() and the use of macro arguments within strings.

Megamax C has its own interpretation of the way getchar() should work. You'll find, rather than grabbing one character and continuing on, it'll wait for a carriage return. Ironic, when you consider that the DRI version of getchar() doesn't recognize carriage returns at all.

Also, Megamax C doesn't allow macro arguments within strings. Listing 2, when compiled from Megamax, will print the Zs in the printf() control string literally, rather than substituting the argument passed to it. However, the values in the table will be correct.

Till next month, happy compiling. ■

Listing 1.
C listing.

```
#include <stdio.h>

char *p1, array1[] = "ABC";
int *p2, array2[] = {10, 11, 12};

main()
{
   int x, ch;

   p1 = array1;
   printf("+0  p1 = %ld    &p1 = %ld    *p1 = %d\n\n", p1, &p1, *p1);
   for (x = 0; x < 3; ++x) {
      p1 = &array1[x];
      printf("+%d  p1 = %ld    &p1 = %ld    *p1 = %d\n", x, p1, &p1, *p1);
      }
   printf("\n");
   for (x = 0, p2 = array2; x < 3; ++x) {
      printf("+%d  p2 = %ld    &p2 = %ld    *p2 = %d\n", x, p2, &p2, *p2);
      ++p2;
      }
   printf("\nPress space bar");
   ch = getchar();
}
```

●

Listing 2.
C listing.

```
#include <stdio.h>
#define PRINT(Z) printf("+%d  Z = %ld   &Z = %ld   *Z = %d\n", x, Z, &Z, *Z)

char *p1, array1[] = "ABC";
int *p2, array2[] = {10, 11, 12};

main()
{
   int x = 0, ch;

   p1 = array1;
   PRINT(p1);
   printf("\n");
   for (x = 0; x < 3; ++x) {
      p1 = &array1[x];
      PRINT(p1);
      }
   printf("\n");
   for (x = 0, p2 = array2; x < 3; ++x) {
      PRINT(p2);
      ++p2;
      }
   printf("\nPress space bar");
   ch = getchar();
}
```

●

IAN's QUEST

ST news and information

by Ian Chadwick

It's not easy trying to think up something witty with which to begin a new column. So I'll save the wit for a later issue. Let's talk about languages, instead.

ST developers and users are high on C these days, citing portability between machines, modular structure, etc., as reasons for enthroning it as language of choice. I'm not so sure it fits. We heard the same gushing praise for Pascal when it arrived, and even for FORTH. (Argh! Try as I might, I could never get my head bent enough to learn to program backwards in FORTH style. . .)

I'm in favor of BASIC, myself. Hands up, everyone who agrees. See? A lot of us out there don't want a career as a programmer, but want to tinker, hack and write little programs to amuse ourselves and our children, or to put up on CompuServe as monuments to our abilities. Good ol' BASIC suits that need, or would if there were a decent version on the ST.

C is a superior developer's language, and I agree about all of its benefits—except ease of learning. It takes dedication and effort, which equates with time. Time is something a lot of us have little of today.

BASIC is *easy* to learn. It lets you fumble around, write sloppy code, experiment and see the results right away. It's immediate. And lots of fun. I've never heard anyone describe C as fun. And *fun* was one

// Ian's Quest *continued*

of the reasons I bought a computer, back in the Pleozoic.

ST BASIC is awful, simply put. It's slow, clumsy, does awkward things with windows and doesn't like TOS in ROM. Wonderful. One sure way of moving everyone to C is to give them a terrible BASIC. Atari managed to do that. The ever-churning rumor mill has it that they have a *new* BASIC due out very soon now, which fixes everything. Sure. After Halley's Comet, I gave up on wonders.

BASIC is the people's language. It may not be the best, but I don't know any computer owner who doesn't know *some* BASIC, or who hasn't tinkered with programming in it. BASIC programmers often graduate to assembly language, if they move upward at all.

To me, assembler is superior to *any* high-level language. Hey, you're speaking the *machine's* language, not through some interface. You and your computer can have a real conversation in assembler, once you learn to translate into hex in your head.

My first experience with my brand new ST was to load TOS (from disk, of course), load BASIC (ditto; no one had considered the advantages of putting it in ROM, or at least on a cartridge, like the 8-bit line), to find I had 5K free.

The mind boggles. I had started in about 1979 with 16K, at which point, people were shouting about all the memory they had to work in (machines had gone from 4K to 16K in the preceding year). I crawled up to 48K, 64K, 128K—and now, on my half-meg machine, I was reduced to 5K! The benefits of an advancing technology . . . It tends to color one's approach to BASIC, to say the least.

At the Atlanta COMDEX, I was given a copy of Softworks BASIC, a compiled BASIC with a *lot* of commands, plus complete access to AES, VDI, BIOS and the rest of the GEM functions (more than some versions of C offer). It should be great; it has oodles of features and compiles your code, removing the *numero uno* complaint about BASIC—slowness. But there are strings attached, as always.

Softworks BASIC doesn't have an interpreter. Being able to write your code and test it as you work is BASIC's main advantage for the non-professional. If C had an interpreter, it would probably take over a lot of BASIC users. The process for this BASIC is much like that for C; you write the code using a word processor (they provide **1st Word** on the disk, but it's not my favorite choice), then compile it.

You don't see the results of your efforts until the very end, so if you find bugs, it's back to the text file again. But, before you run the program, enter their runtime system. What? Yes, that's right. . .you don't create stand-alone code, only an intermediate code. It can't be run by anyone else, unless they also have the runtime program. This suggests either that they expect everyone will buy a copy, or that you can distribute the runtime system freely. Nothing in the manual suggests otherwise; it lacks even the basic copyright information (those lines about: "distribute this program, and we break your legs"). Big mistake. All BASICs need an interpreter. Carve that one in stone, please.

Second mistake: compilers should produce stand-alone code. Hand me the chisel again, for law number two. This intermediate stuff harks back to the good old days of the Apple II+ and TRS-80 Model 1 (and the brontosaur). We've come a tad further than mandatory runtime programs (especially if those aren't for free distribution).

Mistake number three (this is like Monty Hall, isn't it?): the manual. Manuals are a subject dear to my heart, having written so many in my time. To call this one terse is to lean toward understatement. Curt is a better adjective. It's a modest reference document, if you're already familiar with BASIC; otherwise, it's rather like Linear B. Several important (and powerful) commands are given a few threadbare lines of description, rather than the lengthy explanations they deserve. Functional examples are so rare as to be an endangered species.

Many of the commands are not found in the Atari 8-bit universe (they come from the IBM or TRS worlds, I gather), so a lot of ST owners won't be familiar with them. You can probably figure out half of the basic command list from your background BASIC use, but the rest are baffling.

The GEM information is spartan, and the error message description cryptic. This keeps book publishers happy; they gleefully churn out books to decode the abysmal manuals that accompany most software.

Bottom line: looks very good, but needs serious work. This could be the dynamite BASIC we've all been waiting for, once these three elements are patched up. (It's a major overhaul, but think of the fame, the riches, the glory!) Until then, it has a limited audience, unfortunately.

Lest we forget, there are two other languages vying for space on CompuServe's ATARI16 SIG: OSS's Personal Pascal and TDI's Modula-2.

I've played minimally with Pascal on other machines and know of its popularity, but I haven't worked with the OSS version. All I can say is that it has a good reputation (as does the company), and the text editor that accompanies it is superb. I know programmers who bought it for its non-GEM text editor, to write assembly code!

I bought Modula-2 and tinkered a bit, although I haven't seen the latest release (mine's version 1, and I've been lazy about returning my disks for an upgrade). Word has it that it's pretty good. The interesting thing is that Pascal and Modula-2 were both written by Niklaus Wirth, Modula-2 being the later—and some say more advanced—version of Pascal.

The greatest difficulty is finding entry-level teaching books on it. It's relatively new and unpopular, although it offers some very potent programming features for the HLL (that's high-level language, folks) crowd, who want to go beyond BASIC. Who knows, I might even get around to serious tinkering when I finally get version 2.

Finally, a serious question. Do you really need to learn to program? It's a lot like cars: you can learn to drive, go shopping, pick the kids up, race around the streets and even parallel park downtown—without ever once learning to change a spark plug.

Programming is auto mechanics for the computer. You don't need to program to use a computer. If your main use is word processing (like 94% of us, according to one survey I read) or games, then why bother?

I don't need to learn C to enjoy playing **Time Bandit** on Ethel. (Jerry Pournelle names *his* computers, so I call my 130XE Fred, and my ST is Ethel. I toyed with Bert and Ernie, Roy and Dale, and even Ford Prefect and Arthur Dent.)

Anyway, aside from the writing aspects, I use my computer to play games. When I finally get tired of other people's games, I'll sink my teeth into one of those HLL's and write my own. ∎

Ian Chadwick is the author of Mapping the Atari *(Compute! Books, 1983, 1985) and has been employed as a writer, editor, journalist, cook, salesman, house painter and itinerant fruit picker, among other things. He lives in Toronto with his wife, six cats and one beleaguered dog, whom he's trying to teach to play chess.*

ST INDEX TO ADVERTISERS

ANALOG Computing on Delphi... more of a good thing!

The **ANALOG Computing** Telecommunications System, or TCS, has been overwhelmingly successful. Because of this popularity, we've now integrated with Delphi, an on-line, full service communication and information network. Delphi offers news and sports from the Associated Press, weather reports, movie reviews, shopping services, travel information, and much more.

But now, Delphi offers the Atari Users' Group, operated by the same people who bring you the #1 Atari magazine, **ANALOG Computing**. You can access Delphi for as little as 10 cents a minute from most cities in the U.S. There are no additional telephone charges, and there's no extra charge for access at 1200 or 2400 bps.

On Delphi, we'll give you a variety of services, including a Forum, where you can send and receive messages from Atari users worldwide. The Atari database consists of hundreds (soon to be thousands) of programs you can easily download and use right away. . .even those from the pages of **ANALOG Computing**. You can also upload your own programs for others.

A conference feature allows you to meet electronically with other Atari users. From time to time, **ANALOG Computing** will arrange electronic gatherings with some of the big names in the Atari world. These will give you a chance to ask those tough questions and offer your opinions in discussions.

ANALOG Computing has set up some of the most knowledgeable people in Atari-dom. Matthew Ratcliff will handle your 8-bit questions, while Arthur Leyenberger will keep you posted on what's hot (and what's not).

Specials for current TCS and *ANALOG Computing* subscribers.

Your TCS membership (before April 15, 1986) entitles you to join Delphi and the Atari Users' Group absolutely free. If you were a TCS subscriber before the date above, you've probably already received our letter about the switch. You should receive a free lifetime Delphi membership, a Delphi Command Card and $10 of line time.

ANALOG Computing magazine subscribers may join Delphi free of charge, too. All subscribers will receive the lifetime Delphi membership and a $5 line-time credit. If you purchase the Delphi Handbook for $29.95, you'll get $20 worth of line-time. And you can subscribe to **ANALOG Computing** directly, while you're on-line, to be eligible for these bonuses.

It's easy to join us on Delphi. First, determine which data communications network you'll use: Uninet, Tymnet or DataPac (in Canada). If you're in the Boston area, you can dial direct (617-576-0862). For local numbers, call Uninet (800-821-5340, in Missouri, 800-892-5915), or Tymnet (800-336-0149). If you have difficulty, call Delphi at 800-544-4005 (in Massachusetts, 617-491-3393).

To reach the sign-up stage, current **ANALOG Computing** subscribers should type *JOINATARI* when asked for user name, then type *ANALOG* when asked for a password. Those who wish to start subscribing on-line should type *SUBSCRIBE*. Once you're on Delphi, you'll find the Atari Users' Group on the "Groups & Clubs" menu. Just type *GR ATARI* from the main menu prompt. It's that easy!

DELPHI™

THE #1 MAGAZINE FOR ATARI® COMPUTER OWNERS

ANALOG COMPUTING

P.O. BOX 23, WORCESTER, MA 01603

The change will do you good!

Graphic Arts

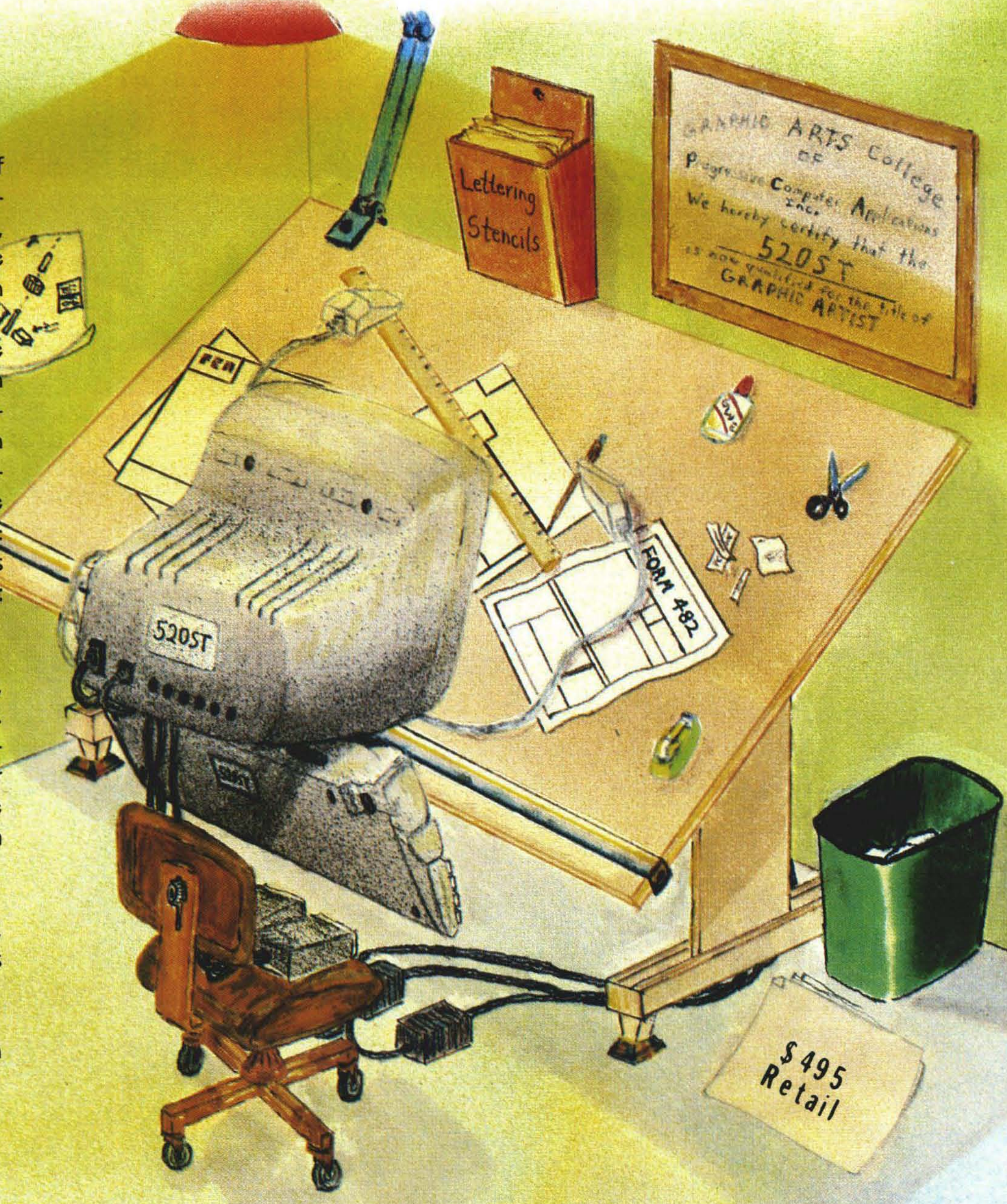
The Next Step in the Evolution of Software

Imagine combining the functions of Computer Aided Design (CAD), Business Graphics, free-hand drawing, and typesetting programs into one package. Include functions of a simple spreadsheet for data manipulation. Add abilities of a simple word processor for text annotation in various fonts, sizes, and rotations. Mix all these features into a single package instead of "integrating" separate packages. The result is a new breed of software: **Graphic Arts.** The only graphic arts program available: **The Graphic Artist.**

The Graphic Artist is menu, macro, command, and language driven. You read correctly. Menus for beginners, commands and macros for experts. And an optional language interpreter for creating custom applications.

Support for dot matrix printers, color plotters, and laser printers is standard, of course.

Impressed? Wait until you see a demo at a dealer near you!



The Graphic Artist

Graphic Arts has finally arrived.

PCA PROGRESSIVE COMPUTER APPLICATIONS
2002 McAuliffe Drive
Rockville, Maryland 20851
(301) 340-8398

★Language $245 additional

CIRCLE #137 ON READER SERVICE CARD