ST·LOG™

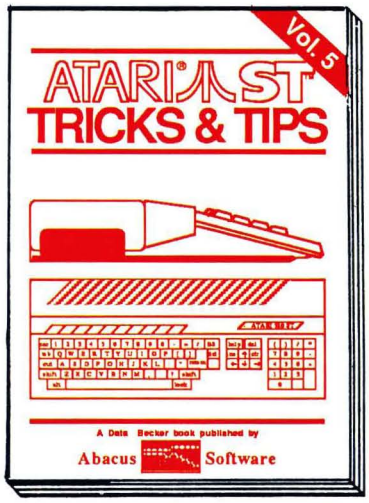
THE ATARI ST OPERATOR'S MAGAZINE

MAY 1986 ISSUE 2

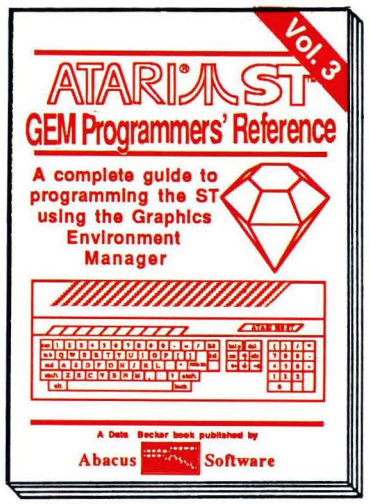
ST Sound Waves
C-manship
Word Processing on the Atari

ATARI® ST™

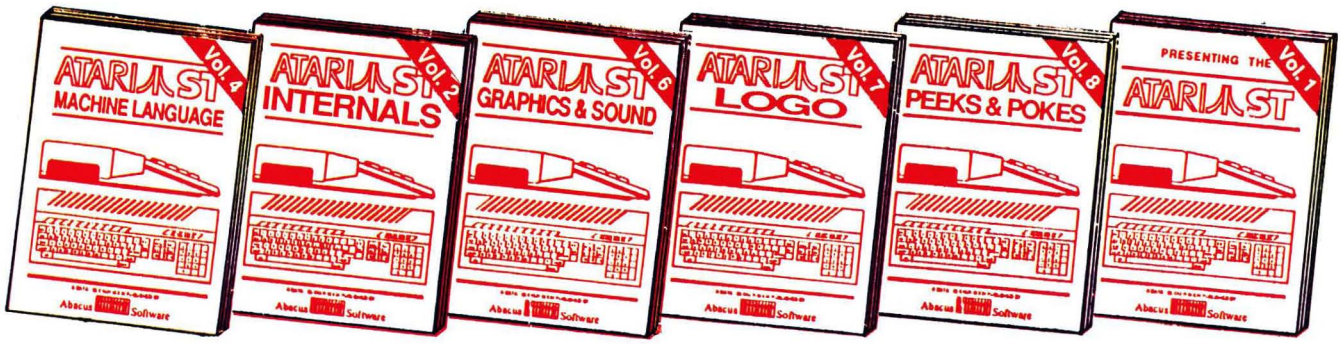
REQUIRED READING



A fantastic collection of useful programs and information for the ST. Complete programs include: super-fast RAM disk; time-saving printer spooler; color print hardcopy; plotter output hardcopy. Explains BASIC commands to access GEM using VDISYS and GEMSYS and describes resource files with examples. Manipulate text output (size, rotation, bold, etc.) Change line types (thickness, end-points, etc.) Mixing machine language with BASIC or C programs. Save software dollars with these tricks and tips. 200 pages $19.95



For the serious programmer in need of detailed information on the GEM operating system. Written especially for the Atari ST with an easy-to-understand format that even beginners will be able to follow. All GEM routines and examples are written in C and 68000 assembly language. Covers working with the mouse, icons, Virtual Device Interface (VDI), Application Environment Services (AES) and the Graphics Device Operating System. Required reading for the serious programmer intrested in understanding the ST. 450 pages. $19.95



MACHINE LANGUAGE
Program in the fastest language for your Atari ST. Learn the 68000 assembly language, its numbering system, use of registers, the structure & important details of the instruction set, and use of the internal system routines. 280pp $19.95

ST INTERNALS
Essential guide to learning the inside information on the ST. Detailed descriptions of the sound & graphic chips, internal hardware, the I/O ports, system addresses, more. Fully documented BIOS assembly listing. An indispensable guide. $19.95

GRAPHICS & SOUND
A comprehensive handbook showing you how to create fascinating graphics and suprising music and sound from the ATARI ST. See and hear what sights and sounds that you're capable of producing from your ATARI ST. $19.95

LOGO
Take control of your ATARI ST by learning LOGO–the easy-to-use, yet powerful language. Topics covered include structured programming, graphic movement, file handling and more. An excellent book for kids as well as adults. $19.95

PEEKS & POKES
Enhance your programs with the examples found within this book. Explores using the different languages BASIC, C, LOGO and machine language, using various interfaces, memory usage, reading and saving from and to disk, more. $19.95

PRESENTING THE ST
Gives you an in-depth look at this sensational new computer. Discusses the architecture of the ST, working with GEM, the mouse, operating system, all the various interfaces, the 68000 chip and its instructions, LOGO. $16.95

Abacus Software

P.O. Box 7219 Grand Rapids, MI 49510 - Telex 709-101 - Phone (616) 241-5510

Optional diskettes are available for all book titles at **$14.95**
Call **now** for the name of your nearest dealer. Or order directly from ABACUS with your MasterCard, VISA, or Amex card. Add $4.00 per order for postage and handling. Foreign add $10.00 per book. **Other software and books coming soon**. Call or write for **free** catalog. Dealer inquiries welcome–over 1400 dealers nationwide.

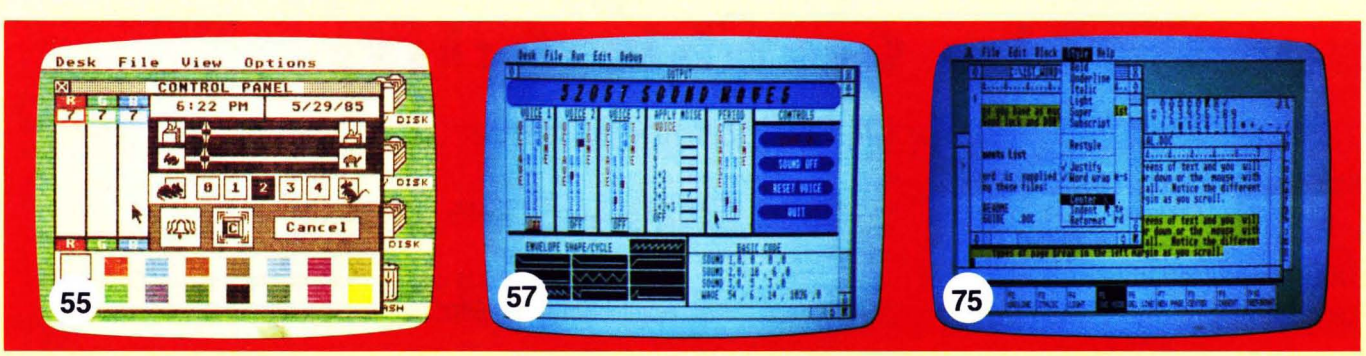
ST·LOG
THE ATARI ST OPERATOR'S MAGAZINE

ISSUE 2
MAY 1986

FEATURES

REVIEWS

COLUMNS



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

The Graphic Artist is a trademark of Progressive Computer Applications, Inc. 520ST is a trademark of Atari Corp.

CIRCLE #124 ON READER SERVICE CARD

MODULA-2

TDI SOFTWARE INC.
10410 Markison Road
Dallas, TX 75238
(214) 340-4942
520ST $79.95

by Sol Guber

A good computer needs good languages in order to utilize all of its capabilities. The Atari 520ST is one of the newest and best computers on the market today, and TDI, has brought out a fine example of what a good language *should* be, **Modula-2**.

First, let me explain what **Modula-2** is, for those who haven't heard of it yet. Niklaus Wirth, the inventor of Pascal, created **Modula-2**. The language is an enhancement and extension of Pascal and has five added features: (1) the module concept, in particular, the facility to split a module into its definition and implementation portions; (2) a more systematic syntax, which facilitates the learning process—every structure starting with a keyword also ends with one (i.e., is properly bracketed); (3) the concept of process as the key to multiprogramming facilities; (4) so-called low-level facilities, which make it possible to breach the rigid type consistency rules and allow you to map data with **Modula-2** structure onto a store with inherent structure; and (5) the procedure type, which allows procedures to be dynamically assigned to variables.

TDI's **Modula-2** compiler implements the full language as described by Dr. Wirth. It includes separate compilation, opaque types, co-routines (pseudo-concurrent processes) and floating-point routines. It is integrated into the GEM environment and will support all the GEM routines. It also promises to generate compact code.

As in all high-level languages, the **Modula-2** package comes in three parts: an editor, a compiler and a linker in a two-disk package. There's no copy protection on any of the disks, and you can arrange the system to fit your needs.

Also included is a nice demonstration of the power of **Modula-2**. Since most programmers don't spend much time with either the compiler or the linker, the editor is the most important part of the package. This editor is very, very nice and quite powerful.

Modula-2's is a full-screen editor which uses both the mouse and function keys. You can point the mouse to the spot where you want to work and click it. This will be the new spot where writing begins. Using the function keys, you can move one word right or left, page up and down, and move a line up and down.

The arrow keys are also used to move around the screen. The deletion function works like the movement functions, with the ability to delete a character right or left, a word right or left, or a line right or left. You can undelete with the undo function.

For large insertions or deletions, you can mark, then cut or paste text into the proper spot. These capabilities are usable both with the mouse and drop-down menus, as are the function keys.

There's one more function unique to this editor, one that I've never seen on another editor. After a program is written, it is compiled. Most of the time, there are errors in the compilation. With **Modula-2**'s compiler, the errors are written to a file on the disk. When you return to the editing mode, the error file and your original file are combined, so there are little @s where the errors occurred.

All errors in the program are examined. There's even a special function key to look for the next @. When the cursor is moved to this spot, a message will be shown at the bottom of the screen, with the error message. The message is further explained in the table of the **Modula-2** book. There are four pages of possible errors, and they're defined well enough that it's easy to correct them and continue.

This listing of the messages in a file that's merged into your own is one sign of the time and effort invested in this fine package. The editor, of course, was written in **Modula-2**.

Figure 1 shows a sample program in **Modula-2**. You can see the various differences between it and Pascal. The first is that there are two parts to the program—a definition module and an implementation module. There's a difference between what a procedure does and what the outside world sees of the program. You must define what will come out of the module. In this case, a cardinal number will be "exported" from the module.

A second difference can be seen in the third-to-last line. The language uses very strong typing. To return a cardinal number from a function that uses a division between "longcards," you must specify that the cardinal number is returned. All procedures and parameters must be typed, so the program will know what kind of variable it's handling.

There are also several subtle points in the example module shown in Figure 1. One that isn't seen is what can be done

with this module. It can be compiled and, when needed by any other program, it can be "imported" into that program. **Modula-2**, unlike Pascal, can link together compiled code via the linker. This means that a library can be put together, then used just as any other function would be.

A second subtle point is the interface between parts of the programs. The definition module explains exactly what another part of the program will "see" of the implementation module. The only thing visible to the outside environment is the procedure "random," since it's the only thing exported out. Variables, variable types and other procedures can be exported out, too. Objects declared in other modules can be referenced in module M, if they're explicitly made to be known in M (i.e., if they're "imported" into M).

The major strength of **Modula-2** is that many of its parts are hidden from the other portions. It's very easy to write and debug parts of programs, since you know exactly how they'll fit together; this has to be specified in the definition section of the module.

Also, the fine details of a system need not be known at any higher level. The same program can be written for two different machines, and each will have specific I/O that will be imported when needed.

In terms of I/O, you may wish to have them available, but don't need to know—or, rather, don't want to bother to learn—how this works in detail. In many cases, you may even wish to hide parts away from access, to guarantee that they'll work correctly.

There are several extensions to **Modula-2** that were proposed by N. Wirth. They include a change in the case syntax, a new variable type called "string," and two new variables for 32-bit machines called "longcard" and "longint." There is optimization for Boolean constants. The "set" type is supported by several new features, and you're allowed to have open array parameters.

The most exciting part of the language is its having both high-level and low-level implementation. For the low-level portion, there's a procedure "code," which allows for the insertion of machine language into the object code. There's "setreg," to put values into one of the 68000 processor's registers, as well as "register," to return values.

On the high-level end, there's the "type process," as well as procedure "newprocess." "Iotransfer" moves to different peripheral devices as needed, and procedure "listen" services the interrupts. The "newprocess" is used to have concurrent programming.

I've started using this language and have found no bugs at all. There are some typos in the documentation, but none of these are significant. I've typed in several of the programs from N. Wirth's book *Programming in Modula-2*, and they've worked perfectly.

The major weakness of the package is in the documentation of the GEM routines. You're expected to have read the GEM manuals. Without them, many GEM routines cannot be used properly. Except for this flaw, I have no qualms recommending this language to anyone. It's a good way to move away from DRI's C.

Sol Guber is a Chemical Engineer with a large petroleum servicing company in the Midwest. He has a seven-year-old daughter who corrects both his articles and games—now that her writing skills have improved. He's been programming the Atari 800 for four years and now has an 520 ST, which his daughter lets him use once in a while.

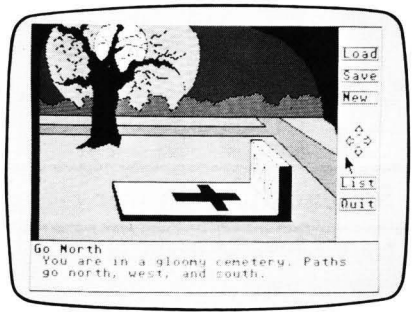
Figure 1.

```
DEFINITION MODULE RandomNumbers ;

(*------------------------------------------------------------------------*)
(*  (c) Copyright 1985 1985 TDI Ltd.   All Rights Reserved                *)
(*------------------------------------------------------------------------*)

EXPORT Random ;

PROCEDURE Random(MaxValue : LONGCARD) : CARDINAL ;

END RandomNumbers.


IMPLEMENTATION MODULE RandomNumbers;

CONST
    M = 10000000 ;
    m1 = 10000 ;
    b = 31415821 ;

VAR seed : LONGCARD;

PROCEDURE Random(MaxValue : LONGCARD) : CARDINAL ;

    PROCEDURE Multiply(p, q : LONGCARD) : LONGCARD ;

    VAR p0, q0, q1, p1 : LONGCARD ;

    BEGIN
         p1 := p DIV m1 ;
         p0 := p MOD m1 ;
         q1 := q DIV m1 ;
         q0 := q MOD m1 ;
         RETURN (((p0*q1 + p1 * q0) MOD m1) * m11 + p0*q0) MOD M;
    END Multiply;

BEGIN
    seed := (Multiply (seed,b) + 1) MOD M ;
    RETURN CARDINAL (((seed DIV m1) * MaxValue) DIV m1) ;
END Random ;

BEGIN (*  MODULE  *)
   seed := 349887 ;
END RandomNumbers.
```

ST NEWS!

POLARWARE

Penguin Software adds to their Comprehend Interactive Novel series with **The Coveted Mirror, Frank and Ernest** and **Oo-Topos**. Pictured below is **Transylvania**, an earlier series offering.



Transylvania.

In **Mirror**, the evil King Voar has taken control of Starbury and its residents, and seeks the last piece of the magical, broken mirror which would make him invincible. It's up to you to stop him. The second new title is a cartoon novel based on the **Frank and Ernest** comic strip, while **Oo-Topos** is a science fiction novel.

Another Penguin game, **Sword of Kadash**, is a fantasy adventure with over 200 rooms, each having its own puzzles.

These adventures retail for $39.95 each, from Polarware/Penguin Software, 2600 Keslinger Road, P.O. Box 311, Geneva, IL 60134 — (312) 232-1984.

CIRCLE #172 ON READER SERVICE CARD

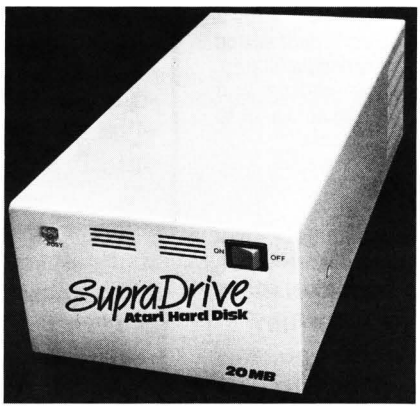
SUPRADRIVE

The **SupraDrive** hard disk for the 520ST and 1040ST is available in 10-, 20-, 30- and 60-megabyte formats, ranging in price from $799 to $1995. The drives are said to be compatible with TOS and all applications software.

The drive comes ready for connection to the ST and improves disk transfer rates 300% to 1000% over the stock drives. The ST is also capable of booting directly from the **SupraDrive** upon initialization. The hard disk comes complete with a utilities disk containing backup, formatting and partitioning programs.

For additional information, please contact Supra Corporation, 1133 Commercial Way, Albany, NY 97321 — (503) 967-9075.

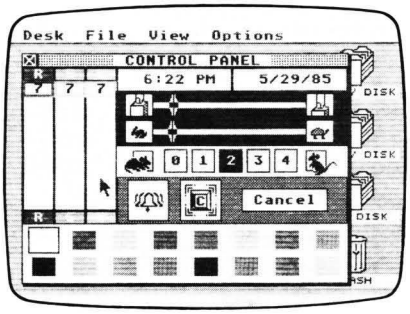
CIRCLE #173 ON READER SERVICE CARD



GRAPHIC COLOR PRINTER

Shanner is marketing the **SPC-700CI**, a 7-color graphic printer. Some of the printer's features include: color mixing without smudging, pin- or friction-feed, parallel interface and low noise. The printer is capable of printing at 38 or 50 characters/second, on normal paper. Black, yellow, magenta and cyan are combined on a special ribbon. When intermingled, this produces up to seven colors.

The **SPC-700CI** retails for $299.95. Contact Shanner International Corp., 453 Ravendale Drive, Mountain View, CA 94043.



CIRCLE #129 ON READER SERVICE CARD

OTHER NEWS

Batteries Included will shortly be releasing **I*S TALK**, a telecommunications program with plenty of power and ease of use. A 50,000-word spelling checker and three levels of macros are also featured.

Also due is **I*S TIME**, a time management and billing system for professionals. From Batteries Included, 30 Mural Street, Richmond Hill, Ontario, Canada L4B 1B5 — (416) 881-9941. CIRCLE #175 ON READER SERVICE CARD

■ ■ ■

Lamar Micro offers a **1-megabyte 520ST upgrade** for $300, giving the user over 750K or RAM. In conjunction with their **RAM Overdrive**, the upgrade lets you set aside the upper 512K of RAM as a ramdisk. **RAM Overdrive** retails for $34.95, from Lamar Micro, 2107 Artesia Boulevard, Redondo Beach, CA 90278. CIRCLE #176 ON READER SERVICE CARD

■ ■ ■

Commnet Systems specializes in telecommunications software and currently offers **ST-Term**. Their newest release is **FoReM ST**, an ST BBS. Some of the many highlights include: a feature-packed message and file system, E-Mail capability and Sysop commands. For additional information, contact Commnet Systems, 7348 Green Oak Terrace, Lanham, MD 20706 — (301) 552-2517.

CIRCLE #132 ON READER SERVICE CARD

■ ■ ■

Infocom is now a subsidiary of Activision. While the consumer probably won't notice any changes in the near future, one thing to look for may be the release of **Cornerstone**, the powerful IBM productivity package that never quite made it in the marketplace. The original $495 price may dwindle down to $100-$150 if ported to the ST.

BUSINESS & STATISTICAL SOFTWARE

Lionheart has advanced software ST packages, including: **Business Statistics, Sales and Market Forecast, Experimental Statistics, Multivariate Analysis, Quality Control & Industrial Experiments, Pert and Critical Path Technologies, Forecasting and Time-Series, Decision Analysis Techniques, Linear & Non-linear Programming** and **Optimization**. The programs are in machine language, to run at optimum speed. Easy to use, with menus at every stage, they're sophisticated and professional.

Example data assists learning. The manuals are touted as full books, with illustrations, indexes and practical examples (the **Experimental Statistics** book is over 200 pages).

Lionheart, P.O. Box 379, Alburg, VT 05440 — (514) 933-4918. CIRCLE #174 ON READER SERVICE CARD.

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

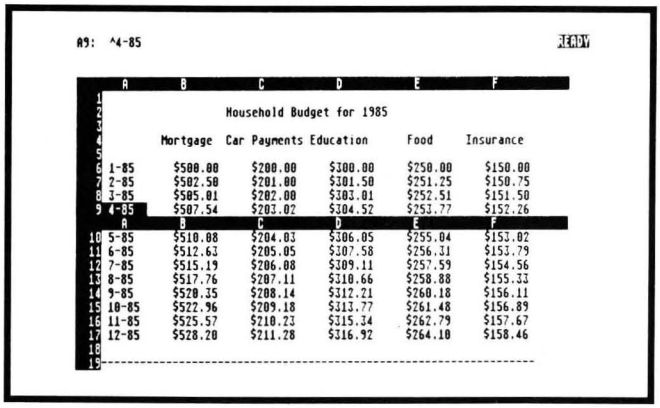
Professional also includes sophisticated macro programming commands. With several special macro commands, the user can actually *program* Professional to be dedicated to a specific task such as accounting.

Just Minutes to Learn

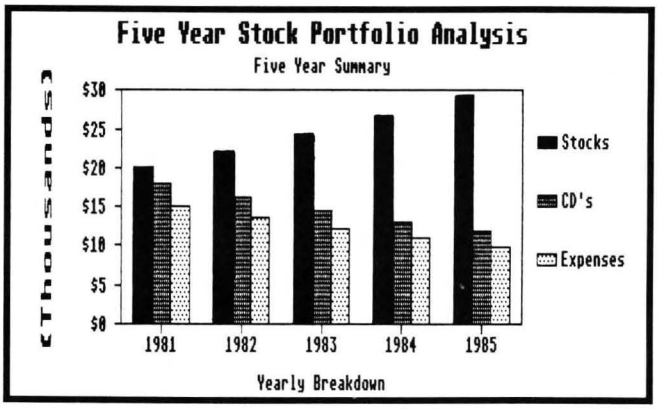
Professional is as easy to use as it is powerful. It comes with a user-sensitive tutorial for the newcomer. And help is built right into the program. With the handy tutorial, you will be able to create professional worksheets in just minutes.

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!



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.

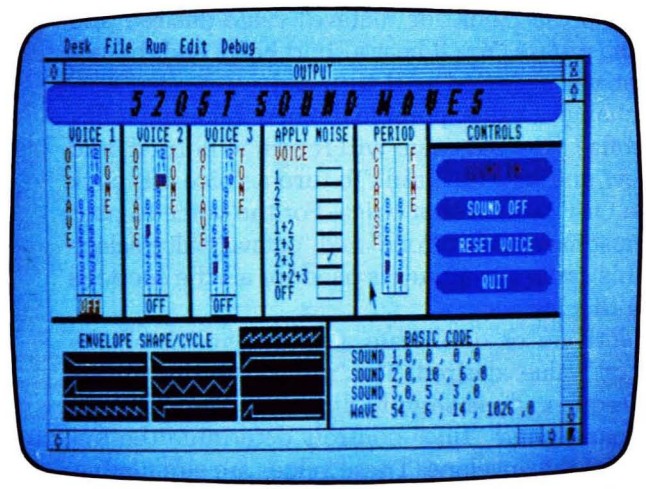
VIP TECHNOLOGIES

132 Aero Camino
Santa Barbara
California 93117

(805) 968-9567

SYSTEM REQUIREMENTS: Amiga with 512K; One disk drive; Monochrome or color monitor; Works with printers supported by the Workbench.

ST Sound Waves



by James Luczak

The one thing that stands out most about demos and programs available for the Atari 520ST is the silence surrounding the machine. I've owned an Atari since 1979 and have watched sound routines progress, from simple one-voice melodies to the truly amazing sound productions now possible on the Atari 8-bit line of computers.

It may seem hard to believe, but the 520ST is capable of producing sound as good as, if not better than the 8-bit computers. However, there's one problem—a lack of information concerning the 520ST's sound capabilities. As you may have noticed, there are two commands in ST BASIC that can be used to produce sounds. These are "sound" and "wave." The sound command is well documented and more or less self-explanatory. The wave command is another sotry. There's almost no useful information in the *ST BASIC Sourcebook* on how to use the wave command.

ST Sound Waves has two purposes: to demonstrate how to use the sound capabilities of the 520ST and to let you get a feel for programming in ST BASIC.

About the program.

Sound Waves is written in a modular fashion. This means that everything, from drawing different parts of the display to producing a sound, is contained in its own module. Each module is accessed via GOSUB as it's needed. I chose this method because it's very easy to follow what each part of the program is doing.

VDISYS() and GEMSYS().

The VDISYS() and GEMSYS() commands are very powerful tools in ST BASIC. The VDISYS() command gives you full access to the GEM VDI (Virtual Display Interface) library. This, in effect, gives you an additional 120+ functions available in the VDI library. The GEMSYS() command gives you full access to GEM's AES (Application Environment Services) library and an additional 60+ functions. Unfortunately, an explanation of how to use the VDISYS() and GEMSYS() commands is beyond the scope of this article. However, once you get the program running, you can bypass one of the modules and see what shows up missing. Using this method, you can determine what the various VDISYS() or GEMSYS() calls are doing.

Using Sound Waves.

Sound Waves is meant to be run in medium resolution. By applying a modifier to all vertical references (the variable *MODY*), I tried to make provisions to run the program in high resolution, but, since I don't have a monochrome monitor, I haven't been able to test the program. If you're going to use high resolution, you may have to make some adjustments to get the display to show up correctly.

Before loading **Sound Waves**, Listing 1, turn buffered graphics off. Now you're ready to use the program.

The display screen is divided into eight sections: one for each of the three "voices," and sections for

noise, period, controls, wave shape, and BASIC code display. To activate a voice, simply move the mouse pointer to the "tone" or "octave" value desired and click the mouse. You'll hear a short beep, and the value that you selected will turn red.

To change a value, point to the new value and click the mouse. The old value will return to normal, and the new one will turn red. To turn a voice off, click on the OFF box for that voice. For the voice to play, you must have both a "tone" and "octave" value selected. This same procedure is used for all the sections of the display.

To activate or deactivate a section, click on the "box" or "value" desired. The "period" section ("envelope frequency") is slightly different. The values in the "coarse" and "fine" controls determine the frequency of the envelope. These values are cumulative.

For example, if you click on coarse 1, then on fine 7, both values will turn red. The values will add together to produce a longer period. To deactivate a period value, click on the value that is active (red). The value will return to normal. You can have all (or any combination) of the values in the coarse and fine controls active at once.

As you select different values, the BASIC code required to produce the tone is displayed in the BASIC section of the display. This section is automatically updated each time you make a change. If you create a sound you want to reproduce in your own program, simply jot down the code that appears in the BASIC code section.

Using the mouse.

Since this program is written entirely in BASIC, the mouse click is probably slightly slower than you're used to. Hold the mouse button (left) a little longer than usual. After you click the mouse and hear the beep, move the mouse pointer away from the area that you clicked on.

The wave command.

Table 1 breaks down the "wave" command and its parameters. The first parameter ("enable") can have

a value from 1 to 63, depending on what combination of voices and noise you select. The values in the chart are typical. The enable parameter is a 6-bit word that enables the "tone" and "noise" channels. Table 2 shows a breakdown of the "enable word." The envelope, shape, period and delay parameters shown in Listing 1 are the actual values to use.

First impressions of ST BASIC.

One of the reasons I wrote this program was to get a feel for programming in ST BASIC. The language itself has a nice variety of powerful and versatile commands from which to choose.

The label option (the ability to name a line) is quite useful for making program listings easier to read. As I previously described, the VDISYS() and GEMSYS() commands give you full access to the VDI and AES libraries. These two commands more than double the number of functions available from ST BASIC—if you know how to use them. And, as far as pure number crunching is concerned, ST BASIC is very fast.

It does have some serious drawbacks that will keep it from being a very useful language. The biggest problem is with its windows. Windows have no place in a computer language. They make it cumbersome, at best, to write a program of any length (for me, it's impossible). They also slow any output to the screen to such a degree that the 68000's speed advantages are totally defeated.

The editor is a disgrace. Once you have the lines you want to work on in the edit window, you can move the cursor to the area you want to edit. Then, as soon as you press a key, the line you're editing turns a very light shade of grey! If this isn't bad enough, the text in the line being edited becomes slightly skewed. On top of all of this, the symbols in the line being edited change. For instance, the plus sign turns into an arrow that points to the left. All of this makes it next to impossible to read the line you're trying to edit.

Fortunately, there's a way around this mess. Apparently, the team that wrote ST BASIC was aware of these problems. ST BASIC programs are written to disk as text files. This being the case, you can use any text or word processor that produces standard text files to write your program. That's the method I used to write this program. Text processors such as **MicroEMACS**, **Mince** or **Letter Express** work just fine. (**ST Writer** will not work; the files it produces are nonstandard files.)

When ST BASIC loads a saved program, it checks each line for syntax errors. If it finds one, it prints the number of the line containing the error and a

Table 1.

THE WAVE COMMAND

SYNTAX: WAVE enable,envelope,shape,period,delay

ENABLE — Enables VOICE or NOISE, or VOICE with NOISE

CHANNEL	TONE VALUE	NOISE VALUE	NOISE with TONE VALUE
1	1	8	9
2	2	16	18
3	4	32	36
1+2	3	24	27
1+3	5	40	45
2+3	6	48	54
1+2+3	7	56	63

ENVELOPE — Enables ENVELOPE for VOICES

VOICE	VALUE
1	1
2	2
3	4
1+2	3
1+3	5
2+3	6
1+2+3	7

SHAPE — SHAPE/CYCLE control

VALUE	DESCRIPTION
1-3	alternate,hold
4-7	attack,alternate,hold
8	continue
9	continue,hold
10	continue,alternate
11	continue,alternate,hold
12	continue,attack
13	continue,attack,hold
14	continue,attack,alternate)
15	continue,attack,alternate,hold

PERIOD — Controls frequency of the ENVELOPE

VALUE	DESCRIPTION
001-128	FINE tune value
256-32768	COARSE tune value

DELAY — Sets time in 1/50-second increments before BASIC resumes execution

Use any desired value. If your WAVE command doesn't seem to be working the way you think it should, try increasing this value.

HOW TO USE THE WAVE COMMAND.

First, use the SOUND command to select the voice or voices desired, and set the NOTE and OCTAVE values. Set the VOLUME and DURATION to 0. When using the WAVE command, the VOLUME and DURATION values in the SOUND command are disabled. Next, set the values for the WAVE command. As your program runs, the SOUND command will take on the parameters set in the WAVE command.

Table 2.

THE ENABLE PARAMETER

BIT	FUNCTION		DECIMAL VALUE
0	Tone	Channel 1	1
1	Tone	Channel 2	2
3	Tone	Channel 3	4
4	Noise	Channel 1	8
5	Noise	Channel 2	16
6	Noise	Channel 3	32

description of the error, then continues to load the rest of the program.

This undocumented feature makes it even easier to use an external text processor to write programs. All you have to do is write down the line numbers and use ST BASIC's editor to correct the errors. This reduces your exposure to ST BASIC's editor.

Like any programming language, ST BASIC has its share of bugs. Most of those I encountered were minor and could be worked around. However, there's one that's very serious.

After your program grows larger than about 10K, this program killer shows up. When you try to save your program (SAVE AS / REPLACE), ST BASIC will erase any previous copy with the same name of your program on disk, then lock up.

There's no way out. You have to press the RESET key. If you haven't taken the precaution of saving your program under a different name, or on another disk, you'll lose your entire program.

There's a way to avoid this nasty little bug: before saving your program, type *CLEARW 2* (Clear window 2), then SAVE your program. This seems to eliminate the bug, but I like to stay on the safe side when writing large programs, and always keep a copy of the program on another disk—just in case.

Jim Luczak maintains and operates electronic telephone switching and processing equipment. He's been writing computer programs since 1979, having bought his first Atari in 1980, and has written in BASIC, C, LOGO, FORTH, Action!, and 6502 assembly. He enjoys writing dedicated database programs.

Listing 1.
ST BASIC listing.

```
100 '520ST SOUND WAVES  Revision 1.0
110 '****** by JIM LUCZAK ******
120 sy=peek(systab)
130 if sy=1 then 150
140 if sy=2 then 170
150 filcol=1:filcol1=1:mody=2
160 goto 240
170 filcol=2:filcol1=3:mody=1
180 poke contrl,26:poke contrl+2,0:pok
e contrl+6,2:poke intin,3
190 poke intin+2,1:vdisys(1)
200 ri=peek(intout+2):gi=peek(intout+4
):bi=peek(intout+6)
210 poke contrl,14:poke contrl+2,0:pok
e contrl+6,4:poke intin,3
220 poke intin+2,0:poke intin+4,0:poke
 intin+6,1000:vdisys(1)
230 ' --- PROGRAM INITIALIZATION ---
240 coorx=605:coory=40*mody:coorx1=3:c
oory1=22*mody:cellh=8
250 a#=gb:gintout=peek(a#+12):gintin=p
eek(a#+8)
260 dim vv1(13),vp1(8),hp1(2),Mh1(26),
Mv1(58),vld(26),vc(3),vn(3),vo(3)
270 dim nv1(8),pc(8),pc1(8),pf(8),pf1(
8),pvo1(6),pvo2(8),pvn1(6),pvn2(12)
280 dim cv1(4),nc(3),en1(24),ev1(3),ev
2(3),ev3(3)
290 for x=1 to 26:read Mh1(x):next x:M
v1(1)=132:Mv1(2)=123
300 for x=1 to 26:read vld(x):next x
310 read nh1:for x=1 to 8:read nv1(x):
next x
320 for x=0 to 23:read en1(x):next x
330 for x=1 to 8:read pc(x),pf(x):next
 x
340 for x=1 to 4:read cv1(x):next x
350 ' -------- DRAW SCREEN ----------
360 fullw 2:clearw 2
370 gosub DOHEADER:wm=2:gosub WRITEHEA
DER
380 x1=50:x2=80:x3=2:x4=1:gosub DOVBAR
S
390 wm=4:gosub SETWRITEMODE:gosub DOOF
F:wm=2:gosub SETWRITEMODE
400 x2=6:gosub DOVBARTEXT
410 x3=6:x4=1:mv=3:gosub DOVBARNUM
420 gosub DONOISE
430 gosub DOPERIOD
440 gosub DOCONTROLS
450 gosub DOSHAPE
460 color 1,1,1:gotoxy 47,13*mody:?"BA
SIC CODE";
470 gosub DOLINES
480 wm=1:gosub SETWRITEMODE
490 ' ------ MAIN PROGRAM LOOP ------
500 poke gintin,257:gemsys(78)
505 poke systab+24,1
510 while mkey <> 2:gemsys(79)
520 mx=peek(gintout+2):my=peek(gintout
+4):mkey=peek(gintout+6)
530 if mkey=1 then gosub CHECKKEY
540 wend
545 poke systab+24,0
550 poke gintin,256:gemsys(78):clearw
2
560 poke contrl,14:poke contrl+2,0:pok
e contrl+6,4:poke intin,3
570 poke intin+2,ri:poke intin+4,gi:po
ke intin+6,bi:vdisys(1)
580 clear:end
600 CHECKKEY:
610 gosub CHECKMOUSEHLOC
620 if hc1=14 then return
630 if hc1=10 and hc2=1 then 650
640 sound 1,0,5,7,0:wave 1,1,1,256,0
650 gosub DOPARAMETERS
660 gosub DOCHANGE
670 gosub DOBASICCODE
680 return
700 DOHEADER:
710 poke contrl,23:poke contrl+2,0:pok
e contrl+6,1:poke intin,2:vdisys(1)
720 poke contrl,24:poke contrl+2,0:pok
e contrl+6,1:poke intin,5:vdisys(1)
730 poke contrl,25:poke contrl+2,0:pok
e contrl+6,1:poke intin,filcol1:vdisys
(1)
740 poke contrl,11:poke contrl+2,2:pok
e contrl+6,0:poke contrl+10,9
750 poke ptsin,coorx:poke ptsin+2,coor
y:poke ptsin+4,coorx1
760 poke ptsin+6,coory1:vdisys(1):retu
rn
780 WRITEHEADER:
790 gosub SETWRITEMODE
800 poke contrl,12:poke contrl+2,1:pok
e contrl+6,0
810 poke ptsin,0:poke ptsin+2,10:vdisy
s(1)
820 poke contrl,106:poke contrl+2,0:po
ke contrl+6,1:poke intin,5:vdisys(1)
```

```
830 color 1,1,1
840 read tnum,x,y:poke contrl,8:poke c
ontrl+2,1:poke contrl+6,tnum
850 for tx=0 to tnum-1:read ascii:poke
 intin+(tx*2),ascii:next tx
860 poke ptsin,x:poke ptsin+2,y:vdisys
(1)
870 poke contrl,12:poke contrl+2,1:pok
e contrl+6,0
880 poke ptsin,0:poke ptsin+2,6:vdisys
(1)
890 poke contrl,106:poke contrl+2,0:po
ke contrl+6,1:poke intin,0:vdisys(1)
900 return
920 DOVBAR5:
930 poke contrl,23:poke contrl+2,0:pok
e contrl+6,1:poke intin,0:vdisys(1)
940 bary=122*mody:bary1=50*mody
950 for x=x1 to (x2*x3)+x1 step x2
960 poke contrl,11:poke contrl+2,2:pok
e contrl+6,0:poke contrl+10,1
970 poke ptsin,x:poke ptsin+2,bary:pok
e ptsin+4,x-15:poke ptsin+6,bary1
980 vdisys(1)
990 poke ptsin,x+15:poke ptsin+2,bary:
poke ptsin+4,x:poke ptsin+6,bary1
1000 vdisys(1):if x4=1 then gosub OFFB
OX
1010 next x:return
1020 OFFBOX:
1030 poke ptsin,x+15:poke ptsin+2,bary
+(10*mody)
1040 poke ptsin+4,x-15:poke ptsin+6,ba
ry
1050 vdisys(1)
1060 return
1080 DOVBARTEXT:
1090 gotoxy 3,2*mody:?"VOICE 1":gotoxy
 12,2*mody:?"VOICE 2"
1100 gotoxy 21,2*mody:?"VOICE 3"
1110 DOVTEXT:
1120 poke contrl,8:poke contrl+2,1:pok
e contrl+6,1
1130 color filcol,1,1:for tx=1 to x2
1140 read tnum:read x:for x1=1 to tnum
:read ascii
1150 poke intin,ascii:poke ptsin,x:pok
e ptsin+2,bary1+(cellh*x1)
1160 vdisys(1):next x1,tx:color 1,1,1:
return
1180 DOVBARNUM:color filcol1,1,1
1190 poke contrl,12:poke contrl+2,1:po
ke contrl+6,2:poke ptsin,0
1200 poke ptsin+2,4:vdisys(1):cellh=pe
ek(ptsout+6)
1210 poke contrl,8:poke contrl+2,1
1220 for x2=1 to x3:tnum=1:read x1,x:g
osub 5AVEHLOC
1230 for tx=0 to x1-1:read ascii:if tx
>=9 then tnum=2
1240 poke contrl+6,tnum:poke intin,asc
ii
1250 if tnum=2 then read ascii:poke in
tin+2,ascii
1260 poke ptsin,x:poke ptsin+2,bary-(c
ellh*tx)
1270 vdisys(1):if x2=2 then gosub 5AVE
VLOC
1280 next tx,x2
1290 poke contrl,12:poke contrl+2,1:po
ke contrl+6,0
1300 poke ptsin,0:poke ptsin+2,6:vdisy
s(1)
1310 cellh=8:color 1,1,1:return
1330 DONOISE:mod1=10:x4=2
1340 gotoxy 30,2*mody:?"APPLY NOISE":c
olor filcol,1,1
1350 gotoxy 30,3*mody:?"VOICE":color 1
,1,1
1360 read x:for x1=1 to 8:poke contrl,
8
1370 poke contrl+2,1:read tnum:poke co
ntrl+6,tnum
1380 for tx=0 to tnum-1:read ascii:pok
e intin+(tx*2),ascii:next tx
1390 poke ptsin,x:poke ptsin+2,bary1+(
cellh*x1)+mod1:vdisys(1)
1400 poke contrl,11:poke contrl+2,2:po
ke contrl+6,0:poke contrl+10,1
1410 poke ptsin,x+80:poke ptsin+2,bary
1+(cellh*x1)+mod1
1420 gosub 5AVEVLOC
1430 poke ptsin+4,x+50:poke ptsin+6,(b
ary1+(cellh*x1))+(mod1-7):vdisys(1)
1440 next x1:x=nh1:np1=8:ascii=8:tnum=
1
1450 y=nv1(np1):gosub MAKECHANGE:retur
n
1470 DOLINE5:
1480 poke contrl,16:poke contrl+2,1:po
ke contrl+6,0:poke ptsin,6
1490 poke ptsin+2,0:vdisys(1)
1500 linef coorx1,112,coorx,112:linef
450,29,605,29
1510 poke ptsin,3:vdisys(1)
1520 linef 90,coory1-3,90,112:linef 17
0,coory1-3,170,112
1530 linef 255,coory1-3,255,112:linef
365,coory1-3,365,112
1540 linef 450,coory1-3,450,112:linef
332,112,332,190
1550 linef 332,126,605,126
1560 poke ptsin,1:vdisys(1):return
1580 DOPERIOD:
1590 gotoxy 43,2*mody:?"PERIOD"
1600 x1=412:x2=1:x3=0:x4=0:gosub DOVBA
R5
1610 x2=2:gosub DOVTEXT
1620 x3=2:x4=3:gosub DOVBARNUM
1630 return
1650 DOCONTROL5:
1660 gotoxy 55,2*mody:?"CONTROL5"
1670 poke contrl,23:poke contrl+2,0:po
ke contrl+6,1:poke intin,2:vdisys(1)
1680 poke contrl,24:poke contrl+2,0:po
ke contrl+6,1:poke intin,4:vdisys(1)
1690 poke contrl,25:poke contrl+2,0:po
ke contrl+6,1:poke intin,filcol1
1700 vdisys(1)
1710 poke contrl,114:poke contrl+2,2:p
oke contrl+6,0:poke ptsin,605
1720 poke ptsin+2,132*mody:poke ptsin+
4,453:poke ptsin+6,52*mody:vdisys(1)
1730 poke contrl,23:poke contrl+2,0:po
ke contrl+6,1:poke intin,1:vdisys(1)
1740 poke contrl,11:poke contrl+2,2:po
ke contrl+6,0:poke contrl+10,9
1750 x1=68*mody:x2=0:x3=18*mody:x4=4:x
5=1
1760 for y1=x1 to x1+(x3*3) step x3:x2
=x2+1
1770 poke ptsin,595:poke ptsin+2,y1:po
ke ptsin+4,460:poke ptsin+6,y1-10
1780 vdisys(1):gosub 5AVEVLOC:next y1
1790 a$(1)="SOUND ON":a$(2)="SOUND OFF
":a$(3)="RESET VOICE":a$(4)="QUIT"
1800 color 0,1,1:x1=55:for x=0 to 3:if
 x=3 then x1=57
1810 if x=2 then x1=54
1820 gotoxy x1,(4*mody)+(x*2):?a$(x+1)
:next x:color 1,1,1:return
1840 DO5HAPE:
1850 x4=5:gotoxy 4,13*mody:?"ENVELOPE
5HAPE/CYCLE"
1860 poke contrl,11:poke contrl+2,2:po
ke contrl+6,0:poke contrl+10,1
```

```
1870 read x:for tx=1 to x:read x1,y1:x
2=x1-100:y2=y1-10*mody:y1=y1*mody
1880 poke ptsin,x1:poke ptsin+2,y1:pok
e ptsin+4,x2:poke ptsin+6,y2
1890 vdisys(1):if tx>6 then gosub SAVE
VLOC
1900 next tx
1910 poke contrl,6:poke contrl+6,0
1920 color 1,1,0:read tnum:for tx1=1 t
o tnum
1930 gosub MAKESHAPE
1940 next tx1:return
1960 DOBASICCODE:
1970 gotoxy 40,14*mody:?"SOUND 1,0,"vn
(1)","vo(1)",0    "
1980 gotoxy 40,15*mody:?"SOUND 2,0,"vn
(2)","vo(2)",0    "
1990 gotoxy 40,16*mody:?"SOUND 3,0,"vn
(3)","vo(3)",0    "
2000 gotoxy 40,17*mody:?"WAVE "en","ev
","sh","pd",0     ";
2010 return
2030 CHECKMOUSEHLOC:
2040 mc=0:hc=1:hc1=1:hct=20
2050 if my>137 then hc=21:hc1=11:hct=2
6
2060 while mc=0
2070 if mx >= mh1(hc) and mx <= mh1(hc
+1) then mc=1
2080 if mc=0 then hc1=hc1+1
2090 hc=hc+2:if hc>hct then mc=1
2100 wend:if hc1>10 and hct=20 then hc
1=14
2110 if hc1 <=6 then hc2=0 else hc2=1
2130 CHECKMOUSEVLOC:
2140 if hc1=14 then hc2=-1:return
2150 mc=0:mc1=0:hc=vld((hc1*2)-1):hct=
vld(hc1*2)
2160 while mc=0
2170 if my <=mv1(hc) and  my >= mv1(hc
+1) then mc=1:mc1=1
2180 if mc=0 then hc2=hc2+1
2190 hc=hc+2:if hc>hct then mc=1
2200 wend:if mc1=0 then hc1=14:hc2=-1
2210 return
2230 DOPARAMETERS:
2240 if hc1 < 7 then gosub SETSOUND
2250 if hc1=7 then gosub SETNOISE
2260 if hc1=8 or hc1=9 then gosub SETP
ERIOD
2270 if hc1=10 then gosub SETCONTROLS
2280 if hc1>10 then gosub SETSHAPE
2290 return
2310 DOCHANGE:
2315 poke gintin,256:gemsys(78)
2320 if hc1<7 then gosub CHANGEVOICE
2330 if hc1=7 then gosub CHANGENOISE
2340 if hc1=8 or hc1=9 then gosub CHAN
GEPERIOD
2350 if hc1=10 then gosub CHANGECONTRO
LS
2360 if hc1>10 then gosub CHANGESHAPE
2365 poke gintin,257:gemsys(78)
2370 return
2390 SETSOUND:
2400 on hc1 goto 2410,2440,2470,2500,2
530,2560
2410 if hc2=0 then vc(1)=0:vn(1)=0:vo(
1)=0:ev1(1)=0:goto 2580
2420 vc(1)=1:vo(1)=hc2:ev1(1)=1:if vn(
1)=0 then vn(1)=1
2430 goto 2580
2440 if hc2=0 then vc(1)=0:vn(1)=0:vo(
1)=0:ev1(1)=0:goto 2580
2450 vc(1)=1:vn(1)=hc2:ev1(1)=1:if vo(
1)=0 then vo(1)=1
2460 goto 2580
2470 if hc2=0 then vc(2)=0:vn(2)=0:vo(
2)=0:ev1(2)=0:goto 2580
2480 vc(2)=2:vo(2)=hc2:ev1(2)=1:if vn(
2)=0 then vn(2)=1
2490 goto 2580
2500 if hc2=0 then vc(2)=0:vn(2)=0:vo(
2)=0:ev1(2)=0:goto 2580
2510 vc(2)=2:vn(2)=hc2:ev1(2)=1:if vo(
2)=0 then vo(2)=1
2520 goto 2580
2530 if hc2=0 then vc(3)=0:vn(3)=0:vo(
3)=0:ev1(3)=0:goto 2580
2540 vc(3)=4:vo(3)=hc2:ev1(3)=1:if vn(
3)=0 then vn(3)=1
2550 goto 2580
2560 if hc2=0 then vc(3)=0:vn(3)=0:vo(
3)=0:ev1(3)=0:goto 2580
2570 vc(3)=4:vn(3)=hc2:ev1(3)=1:if vo(
3)=0 then vo(3)=1
2580 en=vc(1)+vc(2)+vc(3)+nc(1)+nc(2)+
nc(3)
2590 gosub ENABLEENV:return
2610 SETNOISE:
2620 e=(hc2-1)*3:for x=0 to 2
2630 nc(x+1)=en1(x+e):if nc(x+1)>0 the
n ev2(x+1)=1 else ev2(x+1)=0
2640 next x:en=vc(1)+vc(2)+vc(3)+nc(1)
+nc(2)+nc(3)
2650 gosub ENABLEENV:return
2670 SETPERIOD:
2680 on (hc1-7) goto 2690,2720
2690 if pc1(hc2)=0 then pd=pd+pc(hc2):
pc1(hc2)=1:goto 2710
2700 if pc1(hc2)=1 then pd=pd-pc(hc2):
pc1(hc2)=0
2710 goto 2740
2720 if pf1(hc2)=0 then pd=pd+pf(hc2):
pf1(hc2)=1:goto 2740
2730 if pf1(hc2)=1 then pd=pd-pf(hc2):
pf1(hc2)=0
2740 if pd<0 then pd=0
2750 return
2770 SETCONTROLS:
2780 on hc2 goto 2790,2840,2860,2980
2790 sound 1,0,vn(1),vo(1),0
2800 sound 2,0,vn(2),vo(2),0
2810 sound 3,0,vn(3),vo(3),0
2820 wave en,ev,sh,pd,0
2830 goto 2990
2840 for sx=1 to 3:sound sx,0,0,0,0:ne
xt sx
2850 wave 0,0,0,0,0:goto 2990
2860 poke gintin,256:gemsys(78)
2865 gosub CHANGECONTROLS
2870 hc2=0:for hc1=2 to 6 step 2:gosub
 CHANGEVOICE:next hc1
2880 hc2=8:gosub CHANGENOISE
2890 restore 5220:x3=2:x4=3:gosub DOVB
ARNUM
2900 sh3=1:gosub CHANGESHAPE
2910 for x=1 to 3:vc(x)=0:vn(x)=0:vo(x
)=0:ev1(x)=0
2920 ev2(x)=0:ev3(x)=0:nc(x)=0:next x
2930 for x=1 to 8:pc1(x)=0:pf1(x)=0:pv
o2(x)=0:next x
2940 for x=1 to 6:pvo1(x)=0:pvn1(x)=0:
next x
2950 for x=1 to 12:pvn2(x)=0:next x
2960 en=0:ev=0:pd=0:sh=0:sh1=0:sh2=0:s
h3=0:np1=0
2965 poke gintin,257:gemsys(78)
2970 goto 2990
2980 mkey=2
2990 return
3010 SETSHAPE:
3020 on (hc1-10) goto 3030,3050,3060
3030 sh1=hc2:if hc2=1 then sh=1 else i
```

```
f hc2=2 then sh=4 else sh=8
3040 goto 3070
3050 sh=hc2+8:sh1=hc2+3:goto 3070
3060 sh=hc2+11:sh1=hc2+6
3070 return
3090 ENABLEENV:
3100 if (ev1(1)+ev2(1))>0 then ev3(1)=
1 else ev3(1)=0
3110 if (ev1(2)+ev2(2))>0 then ev3(2)=
2 else ev3(2)=0
3120 if (ev1(3)+ev2(3))>0 then ev3(3)=
4 else ev3(3)=0
3130 ev=ev3(1)+ev3(2)+ev3(3):return
3150 CHANGEVOICE:px=0
3160 on hc1 goto 3170,3250,3340,3420,3
510,3590
3170 if pvo2(1)=0 then wm=1:c1=1
3180 sx=38:gosub SETOFF:if pvo2(1)=0 t
hen goto 3230
3190 if px=1 then px=2:hc1=hc1-1
3200 x=hvl(pvo1(1)):y=vvl(pvo2(1)):asc
ii=pvo2(1)+48:tnum=1:c1=filcol1:wm=1
3210 gosub SETMODE:gosub MAKECHANGE
3220 if px=2 then px=0:hc1=hc1+1:goto
3670
3230 pvo1(1)=hc1:pvo2(1)=hc2:if hc2=0
then px=1:goto 3270
3240 goto 3670
3250 if pvn2(1)=0 then wm=1:c1=1
3260 sx=38:gosub SETOFF:if pvn2(1)=0 t
hen goto 3320
3270 if px=1 then px=2:hc1=hc1+1
3280 x=hvl(pvn1(1)):y=vvl(pvn2(1)):asc
ii=pvn2(1)+48:tnum=1:c1=filcol1:wm=1
3290 if pvn2(1)>9 then tnum=2:ascii=49
:ascii1=38+pvn2(1)
3300 gosub SETMODE:gosub MAKECHANGE
3310 if px=2 then px=0:hc1=hc1-1:goto
3670
3320 pvn1(1)=hc1:pvn2(1)=hc2:if hc2=0
then px=1:goto 3190
3330 goto 3670
3340 if pvo2(2)=0 then wm=1:c1=1
3350 sx=118:gosub SETOFF:if pvo2(2)=0
then goto 3400
3360 if px=1 then px=2:hc1=hc1-1
3370 x=hvl(pvo1(2)):y=vvl(pvo2(2)):asc
ii=pvo2(2)+48:tnum=1:c1=filcol1:wm=1
3380 gosub SETMODE:gosub MAKECHANGE
3390 if px=2 then px=0:hc1=hc1+1:goto
3670
3400 pvo1(2)=hc1:pvo2(2)=hc2:if hc2=0
then px=1:goto 3440
3410 goto 3670
3420 if pvn2(2)=0 then wm=1:c1=1
3430 sx=118:gosub SETOFF:if pvn2(2)=0
then goto 3490
3440 if px=1 then px=2:hc1=hc1+1
3450 x=hvl(pvn1(2)):y=vvl(pvn2(2)):asc
ii=pvn2(2)+48:tnum=1:c1=filcol1:wm=1
3460 if pvn2(2)>9 then tnum=2:ascii=49
:ascii1=38+pvn2(2)
3470 gosub SETMODE:gosub MAKECHANGE
3480 if px=2 then px=0:hc1=hc1-1:goto
3670
3490 pvn1(2)=hc1:pvn2(2)=hc2:if hc2=0
then px=1:goto 3360
3500 goto 3670
3510 if pvo2(3)=0 then wm=1:c1=1
3520 sx=198:gosub SETOFF:if pvo2(3)=0
then goto 3570
3530 if px=1 then px=2:hc1=hc1-1
3540 x=hvl(pvo1(3)):y=vvl(pvo2(3)):asc
ii=pvo2(3)+48:tnum=1:c1=filcol1:wm=1
3550 gosub SETMODE:gosub MAKECHANGE
3560 if px=2 then px=0:hc1=hc1+1:goto
3670
```

```
3570 pvo1(3)=hc1:pvo2(3)=hc2:if hc2=0
then px=1:goto 3610
3580 goto 3670
3590 if pvn2(3)=0 then wm=1:c1=1
3600 sx=198:gosub SETOFF:if pvn2(3)=0
then goto 3660
3610 if px=1 then px=2:hc1=hc1+1
3620 x=hv1(pvn1(3)):y=vv1(pvn2(3)):asc
ii=pvn2(3)+48:tnum=1:c1=filcol1:wm=1
3630 if pvn2(3)>9 then tnum=2:ascii=49
:ascii=38+pvn2(3)
3640 gosub SETMODE:gosub MAKECHANGE
3650 if px=2 then px=0:hc1=hc1-1:goto
3670
3660 pvn1(3)=hc1:pvn2(3)=hc2:if hc2=0
then px=1:goto 3530
3670 if hc2=0 then wm=4:c1=filcol:gosu
b SETOFF:goto 3710
3680 x=hv1(hc1):y=vv1(hc2):ascii=hc2+4
8:tnum=1:c1=filcol:wm=4
3690 if hc2>9 then tnum=2:ascii=49:asc
ii1=38+hc2
3700 gosub SETMODE:gosub MAKECHANGE
3710 c1=1:wm=1:gosub SETMODE:return
3730 SETOFF:
3740 gosub SETWRITEMODE:color c1,1,1
3750 poke contrl,8:poke contrl+2,1:pok
e contrl+6,3
3760 poke intin,79:poke intin+2,70:pok
e intin+4,70
3770 poke ptsin,sx:poke ptsin+2,bary+(
10*mody)-2
3780 vdisys(1):return
3800 MAKECHANGE:
3810 poke contrl,12:poke contrl+2,1:po
ke contrl+6,2:poke ptsin,0
3820 poke ptsin+2,4:vdisys(1):poke con
trl,8:poke contrl+2,1
3830 poke contrl+6,tnum:poke intin,asc
ii:if tnum=2 then poke intin+2,ascii1
3840 poke ptsin,x:poke ptsin+2,y:vdisy
s(1)
3850 poke contrl,12:poke contrl+2,1:po
ke contrl+6,0
3860 poke ptsin,0:poke ptsin+2,6:vdisy
s(1):return
3880 CHANGENOISE:
3890 wm=1:c1=1:x=nh1:tnum=1:ascii=0:go
sub SETMODE
3900 y=nv1(np1):gosub MAKECHANGE
3910 ascii=8:y=nv1(hc2):gosub MAKECHAN
GE
3920 np1=hc2:return
3940 CHANGEPERIOD:
3950 on (hc1-7) goto 3960,3980
3960 if pc1(hc2)=0 then wm=1:c1=filcol
1 else wm=4:c1=filcol
3970 goto 3990
3980 if pf1(hc2)=0 then wm=1:c1=filcol
1 else wm=4:c1=filcol
3990 ascii=hc2+48:tnum=1:x=hp1(hc1-7):
y=vp1(hc2)
4000 gosub SETMODE:gosub MAKECHANGE:c1
=1:wm=1:gosub SETMODE:return
4020 CHANGECONTROLS:
4030 wm=2:gosub SETWRITEMODE
4040 if cp1=0 then 4060
4050 color 0,1,1:gotoxy cv1(cp1),(4*mo
dy)+((cp1-1)*2):?a$(cp1)
4060 color filcol,1,1:gotoxy cv1(hc2),
(4*mody)+((hc2-1)*2):?a$(hc2)
4070 cp1=hc2:wm=1:c1=1:gosub SETMODE:r
eturn
4090 CHANGESHAPE:
4100 wm=2:gosub SETWRITEMODE
4110 c1=0:if sh2=0 then 4250
4120 on sh2 goto 4130,4140,4150,4160,4
170,4180,4190,4200,4210,4220
4130 restore 5310:goto 4230
4140 restore 5330:goto 4230
4150 restore 5350:goto 4230
4160 restore 5400:goto 4230
4170 restore 5420:goto 4230
4180 restore 5450:goto 4230
4190 restore 5470:goto 4230
4200 restore 5520:goto 4230
4210 restore 5540:goto 4230
4220 restore 5570
4230 color 1,1,c1:gosub MAKESHAPE
4240 if sh3=1 then sh3=0:goto 4260
4250 sh2=sh1:sh3=1:c1=filcol:goto 4120
4260 color 1,1,1:c1=1:wm=1:gosub SETMO
DE:return
4280 MAKESHAPE:
4290 poke contrl,6:poke contrl+6,0
4300 read x:poke contrl+2,x*2
4310 for tx=0 to (x*4)-1 step 2:read x
1,y1:y1=y1*mody
4320 poke ptsin+(tx*2),x1:poke ptsin+(
(tx*2)+2),y1:next tx
4330 vdisys(1):return
4350 SETWRITEMODE:
4360 poke contrl,32:poke contrl+2,0:po
ke contrl+6,1:poke intin,wm:vdisys(1)
4370 return
4390 DOOFF:
4400 color filcol,1,1
4410 read tnum:poke contrl,8:poke cont
rl+2,1:poke contrl+6,tnum
4420 for tx=0 to tnum-1:read ascii
4430 poke intin+(tx*2),ascii:next tx
4440 for tx=1 to tnum:read x
4450 poke ptsin,x:poke ptsin+2,bary+(1
0*mody)-2
4460 vdisys(1):next tx:color 1,1,1:ret
urn
4480 SAVEHLOC:
4490 on x4 goto 4500,4520,4510,4520,45
20
4500 hv1(x2)=x:goto 4520
4510 hp1(x2)=x:goto 4520
4520 return
4540 SAVEVLOC:
4550 on x4 goto 4560,4580,4600,4610,46
20
4560 vv1(tx+1)=bary-(cellh*tx):mv1(mv)
=vv1(tx+1):mv1(mv+1)=vv1(tx+1)-5
4570 mv=mv+2:goto 4630
4580 mv1(mv+1)=bary1+(cellh*x1)+(mod1-
7):mv1(mv)=bary1+(cellh*x1)+mod1
4590 mv=mv+2:goto 4630
4600 vp1(tx+1)=bary-(cellh*tx):goto 46
30
4610 mv1(mv+1)=y1-10:mv1(mv)=y1:mv=mv+
2:goto 4630
4620 mv1(mv+1)=y2:mv1(mv)=y1:mv=mv+2
4630 return
4650 SETMODE:
4660 color 1,1,1:gosub SETWRITEMODE:co
lor c1,1,1:return
4680 WORKDATA:
4700 data 35,50,51,65,115,130,131,145,
195,210,211,225
4710 data 320,350
4720 data 397,412,413,427
4730 data 460,595
4740 data 18,118,123,223,228,328
4750 data 1,18,1,26,1,18,1,26,1,18,1,2
6,27,42,3,18,3,18,43,50
4760 data 53,58,53,58,51,58
4770 data 333,66,74,82,90,98,106,114,1
22
4790 data 8,0,0,0,16,0,0,0,32,8,16,0,8
,0,32,0,16,32,8,16,32,0,0,0
4800 data 256,1,512,2,1024,4,2048,8,40
96,16,8192,32,16384,64,32768,128
4810 data 55,55,54,57
4830 data 31,100,35,53,32,50,32,79,32,
83,32,84,32,32,83,32,79
4840 data 32,85,32,78,32,68,32,32,87,3
2,65,32,86,32,69,32,83
4860 data 3,79,70,70,38,118,198
4880 data 6,23,79,67,84,65,86,69
4900 data 4,70,84,79,78,69
4920 data 6,104,79,67,84,65,86,69
4940 data 4,149,84,79,78,69
```

```
4960 data 6,184,79,67,84,65,86,69
4980 data 4,229,84,79,78,69
5000 data 8,39,49,50,51,52,53,54,55,56
5020 data 12,52,49,50,51,52,53,54,55,5
6,57,49,48,49,49,49,50
5040 data 8,119,49,50,51,52,53,54,55,5
6
5060 data 12,132,49,50,51,52,53,54,55,
56,57,49,48,49,49,49,50
5080 data 8,199,49,50,51,52,53,54,55,5
6
5100 data 12,212,49,50,51,52,53,54,55,
56,57,49,48,49,49,49,50
5120 data 270,1,49,1,50,1,51,3,49,43,5
0,3,49,43,51,3,50,43,51
5140 data 5,49,43,50,43,51
5160 data 3,79,70,70
5180 data 6,387,67,79,65,82,83,69
5200 data 4,430,70,73,78,69
5220 data 8,401,49,50,51,52,53,54,55,5
6
5240 data 8,416,49,50,51,52,53,54,55,5
6
5260 data 10,118,160,118,172,118,184
5270 data 223,160,223,172,223,184
5280 data 328,148,328,160,328,172,328,
184
5300 data 10
5310 data 2,20,152,32,156,32,156,114,1
56
5330 data 3,20,170,32,164,32,164,32,17
0,32,170,114,170
5350 data 15,20,177,32,181,32,181,32,1
77,32,177,44,181,44,181,44,177
5360 data 44,177,56,181,56,181,56,177,
56,177,68,181,68,181,68,177,68,177
5370 data 80,181,80,181,80,177,80,177,
92,181,92,181,92,177,92,177,104,181
5380 data 104,181,104,177,104,177,114,
181
5400 data 2,125,152,137,156,137,156,21
9,156
5420 data 8,125,164,137,170,137,170,14
9,164,149,164,161,170,161,170,173,164
5430 data 173,164,185,170,185,170,197,
164,197,164,209,170,209,170,219,164
5450 data 3,125,177,137,182,137,182,13
7,177,137,177,219,177
5470 data 15,231,145,241,141,241,141,2
41,145,241,145,253,141,253,141,253,145
5480 data 253,145,265,141,265,141,265,
145,265,145,277,141,277,141,277,145
5490 data 277,145,289,141,289,141,289,
145,289,145,301,141,301,141,301,145
5500 data 301,145,313,141,313,141,313,
145,313,145,323,141
5520 data 2,231,158,241,153,241,153,32
3,153
5540 data 8,231,170,241,164,241,164,25
3,170,253,170,265,164,265,164,277,170
5550 data 277,170,289,164,289,164,301,
170,301,170,313,164,313,164,323,170
5570 data 3,231,182,241,176,241,176,24
1,182,241,182,323,182
```

●

ST-CHECKSUM DATA.

(see page 58ST)

```
100 data 279,481,559,359,369,731,405,7
39,226,600,4748
200 data 698,208,863,531,781,925,304,3
92,421,953,6076
300 data 363,642,287,105,365,872,518,7
17,18,913,4800
400 data 377,482,331,395,841,315,803,3
43,744,191,4822
500 data 858,543,872,452,243,64,552,99
9,229,976,5788
580 data 615,613,338,89,693,936,995,36
1,766,364,5770
700 data 606,553,560,457,701,261,502,1
920 data 565,561,14,992,695,510,621,51
0,47,382,4897
1020 data 459,127,57,699,443,102,425,9
63,656,444,4375
1130 data 430,231,454,575,235,270,523,
336,432,812,4298
1240 data 269,228,33,391,866,432,316,8
77,60,901,4373
1350 data 174,932,424,395,15,709,923,4
40,960,190,5162
1450 data 261,590,296,648,904,413,433,
748,928,759,5980
1560 data 504,812,903,959,85,908,458,8
6,320,641,5676
1680 data 648,141,715,593,506,638,746,
89,280,991,5347
1780 data 881,403,427,439,977,612,333,
727,177,706,5682
1890 data 526,477,365,620,618,708,239,
348,358,368,4627
2000 data 503,440,679,126,475,692,778,
110,834,4,4641
2110 data 222,710,866,676,695,503,116,
837,445,446,5516
2230 data 390,418,107,203,649,105,454,
740,856,442,4364
2330 data 468,564,10,466,864,455,859,7
37,878,234,5535
2430 data 582,881,241,585,891,250,588,
887,250,584,5739
2530 data 897,260,587,900,267,368,634,
826,277,412,5428
2640 data 872,633,939,269,970,711,583,
990,735,575,7277
2750 data 465,345,844,166,165,171,821,
600,225,220,4022
2860 data 866,430,521,519,522,488,160,
426,533,585,5050
2950 data 396,976,882,607,314,475,774,
715,53,569,5761
3050 data 790,723,448,881,846,856,867,
195,656,757,7019
3170 data 750,355,891,418,710,27,695,5
81,748,354,5529
3270 data 888,417,923,712,33,695,583,7
56,399,894,6300
3370 data 437,720,37,699,584,754,407,8
91,429,931,5889
3470 data 722,43,706,586,762,430,897,4
49,723,40,5358
3570 data 710,594,767,431,894,441,939,
725,46,710,6257
3670 data 125,956,834,724,785,470,746,
475,347,99,5561
3780 data 734,980,287,626,852,564,450,
433,201,512,5639
3900 data 268,252,529,307,308,610,612,
618,984,36,4524
4020 data 683,817,542,385,825,746,144,
817,258,732,5949
4130 data 164,169,174,166,171,178,183,
168,173,52,1598
4230 data 221,922,768,169,875,358,604,
699,92,719,5427
4350 data 378,846,459,353,512,294,217,
510,580,87,4236
4460 data 272,807,133,899,888,460,834,
151,962,628,6034
4580 data 724,630,97,60,519,464,608,23
1,799,361,4493
4710 data 821,592,855,321,47,807,25,44
1,67,205,4181
4830 data 987,88,234,464,830,439,843,4
67,843,994,6189
5020 data 53,974,29,2,27,39,747,152,44
9,801,3273
5220 data 973,983,649,349,248,273,24,4
07,869,356,5131
5370 data 357,186,356,544,377,48,567,3
68,353,693,3849
5520 data 332,553,329,985,2199
```

●

STylish Software

No question about it, the new Atari 520 ST™ is a remarkable computer. And nothing complements a great computer better than great software and great peripherals.

HabaWriter™. A full-function word processor, featuring windows for simultaneous multiple document editing as well as pull-down menus for fast access to program commands. Advantageous use of the mouse means never having to memorize cryptic commands again. **HabaWriter** is the word processor your 520 ST has been waiting for. If you do any writing at all, take a look at **HabaWriter**. Suggested Retail: $74.95

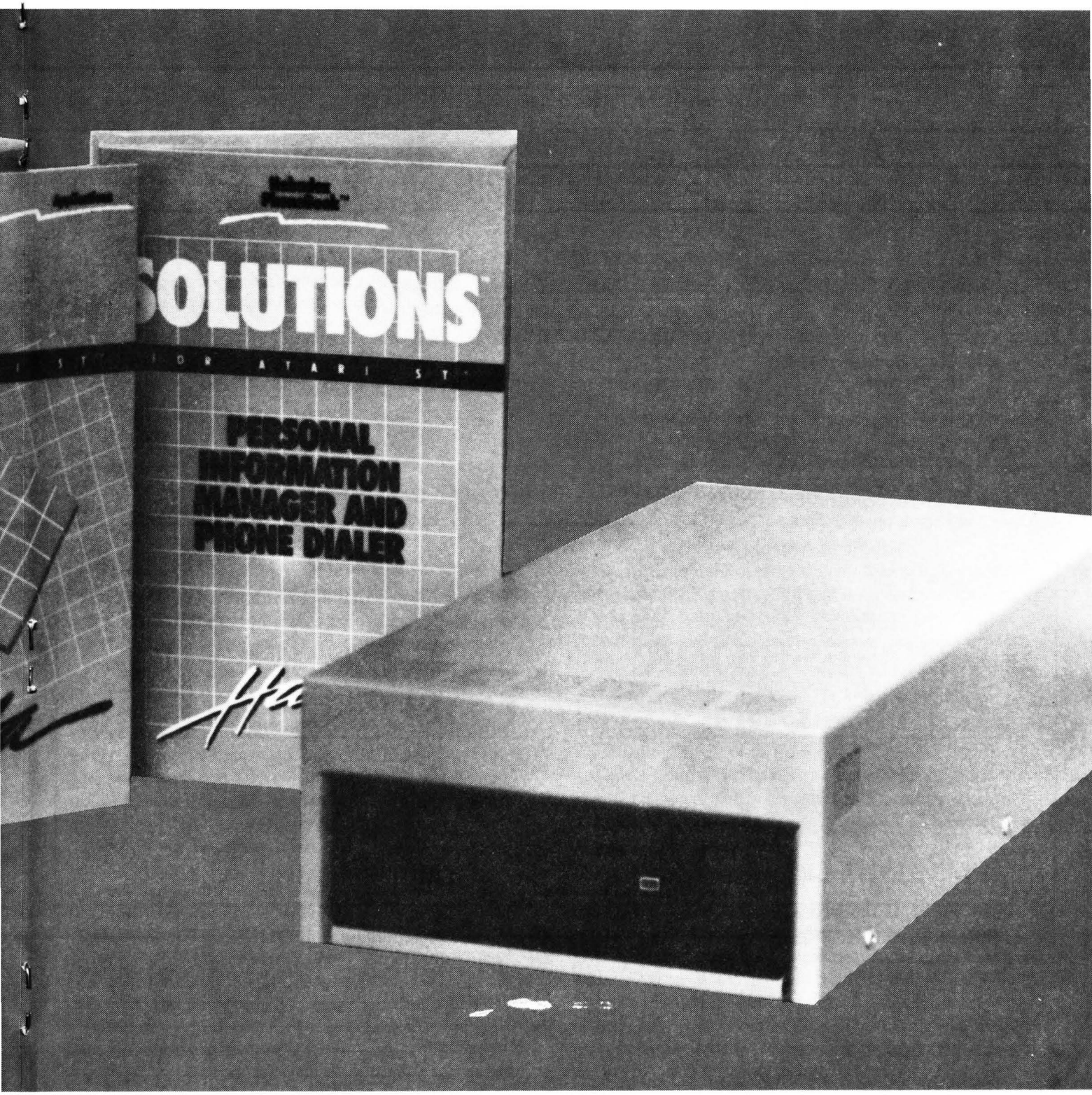
Habadex PhoneBook™ is the elegant way to store phone numbers. And it not only stores numbers, but it can dial them as well. It works and looks just like the flip-up phone book that you're used to. Long distance services like MCI and Sprint can be automatically dialed so you don't have to. The **PhoneBook** can sort on any field, is versatile enough to handle other types of information and can even print mailing labels. (Automatic dialing requires either a **HabaModem**™ or any Hayes™ compatible modem.) Suggested Retail: $49.95

The new **HabaDisk™ 10 Megabyte** hard disk for the 520 ST is a Winchester plug-in hard disk that is capable of storing the equivalent of more than 12 dual-sided 800K diskettes and retrieves information in seconds (3 msec. track-to-track access time). It is self-powered and completely Atari ST compatible (including Atari Desktop and GEM™ DOS). Suggested Retail: $699.95

Also available for the 520 ST:

Haba Checkminder™—Suggested Retail: $74.95
Haba Mail Room™—Suggested Retail: $74.95
HabaMerge™—Suggested Retail: $39.95
Solutions: Wills™—Suggested Retail: $49.95
Solutions: Business Letters™—Suggested Retail: $49.95

STupendous Storage



Haba

6711 Valjean Avenue
Van Nuys, CA 91406
(818) 989-5822 • (800) HOT-HABA (USA) • (800) FOR-HABA (CA)

HOW TO GET THE MOST OUT OF YOUR ATARI.

The powerful new Atari ST is capable of extraordinary color graphics. Right now, there is only one full size, commercially available color printer that can screen dump in over 120 colors off the Atari ST, and we've got it. The Shanner SPC 700CI color printer.

Other features include:

- Fast and quiet operation
- Uses standard paper—inexpensive to operate
- Centronics interface—compatible with many other computers

- Million characters per color ribbon cassette
- Made by Seikosha

VIP Professional is our powerful new software for the Atari ST. Its program is identical in features and commands to Lotus 1-2-3 with Macros, 256 x 8092 columns, Data Base capabilities with Sort and Query Fields.

VIP Professional is a trademark of VIP Technologies. Lotus 1-2-3 is a trademark of Lotus Development Corporation. Atari ST is a trademark of Atari Corporation.

VIP PROFESSIONAL SPREADSHEET

$99.90

SPC-700CI COLOR PRINTER

$299.95

Full-Blown Original Version

GUMBALL EXPRESS ORDER FORM • For FAST delivery use this order form or call **TOLL FREE 800/423-9442**

Product Description	Price	P&L	TOTAL
VIP Professional for Atari ST	$ 99.90	$3.50	**$103.40**
Shanner SPC-700CI	$299.95	$7.50	**$307.45**

☐ Check here and add $14.95 to $399.95

All products will be shipped prepaid UPS ground.

☐ Check enclosed. (NOTE— order will be shipped when check clears).

Make check payable to:
Gumball Express
707 S.W. Washington Street Suite 200
Portland, Oregon 97205

☐ VISA ☐ MASTERCARD ☐ INTERBANK (MasterCard only)

Name on card _____

Account # _____

Expiration Date _____

Signature _____

SHIP TO:

Name _____

Address _____

City _____ State _____ Zip _____

C.O.D.'s and purchase orders will not be accepted by Gumball Express. Outside the USA add $10. and make payment by bank draft, payable in U.S. dollars drawn on a U.S. bank.

CIRCLE #129 ON READER SERVICE CARD

C-MANSHIP

Part 4.

by Clayton Walnum

Before we get started, I'd like to thank everyone who's sent me their comments on **C-manship**. When I first started this column, I was a little worried there might not be much interest. I'm pleased to report that all the feedback has been positive, so I feel this column has a secure future.

If you were thinking of writing, but haven't gotten around to it, please do. I want to hear from you. The only way I have of knowing that I'm on the right track is from the comments I receive from you, the readers.

While we're on this subject, I got one letter of particular interest from Donald Howes, who writes:

> *Don't use scanf()!...To make a long story short, scanf() can develop a life of its own. It will often present you with results completely at variance with what you might have expected. This is a problem for novice programmers especially. They might be a little unsure of what they're doing and could spend needless hours trying to debug a simple program, when the problem is inherent in scanf(). It's a much better idea to develop your own input routines...*

So we've been warned. Keep that in mind during your experiments with C. As I said last month, we'll eventually construct our own input routines. But for now, we'll treat scanf() with suspicion.

Our current project.

Okay, fun's over. Let's get back to work. Listing 1 is this month's program. Type it in and compile it. If you need help, see the sidebar accompanying last issue's **C-manship** article.

Feeling lucky? Good. Get out all that green stuff that's been cluttering up your wallet and give Lady Luck a wink. This month, we're all going to learn how to play craps. (I know that was top priority on your things-I've-got-to-do-today list.)

Now I admit, this isn't the most stunning version that'll cross your eyeballs, but it's a good programming exercise and demonstrates a lot of new techniques.

If you already know the rules (that's where you've been all those late nights, huh?), skip ahead to the next section. For those who've led sheltered lives, craps is a dice game which has the dubious reputation for making and breaking many a fortune. In our case, we'll try to leave your savings intact—only the rules remain the same.

Step one is to roll the dice. If, on your first roll, you get a seven or an eleven, you win. A two, three or twelve, on the other hand, leaves you the loser. If you manage to avoid all lucky and unlucky combinations, you must roll again. . .and continue to do so, in an effort to attain one of two outcomes: if you reroll your original number, you win; if you get a seven or eleven first, you lose.

The game's afoot (without toes).

Now that you know how to play, take a moment to try the program out. Have a little fun and get a general idea of what's going on.

Now let's take a look at the listing. You might want to number each line, so you can refer to them more easily as we go through the program. Remember, I don't count blank lines.

I don't think it's necessary to go through every line as we have in the past. I think you've had most of the basics pounded into your heads, right? Just notice that we've included a couple extra files (osbind.h and portab.h) this time around. You'll see why later on.

Let's skip ahead to Line 9. You've probably noticed that I usually use *ch* as a character variable. This time, I have it declared as an "int." Does that mean that I've abandoned our poor friend *ch* to a new and unknown fate? No, we're still going to use it to hold character information, because it just so happens that the only difference between a character variable and an integer is the number of bytes taken up in memory.

If you remember from a few months ago, a character is stored in 1 byte and an integer is in 2. For our purposes, the two are really interchangeable. What you should be aware of is that, in C, character variables are converted to integers for processing, then truncated back to a single byte.

By declaring them as integers in the first place, you'll always be reminded of what's going on in your machine's innards. And you may come across a time in your illustrious programming career where the difference will be critical.

Now skip ahead to Line 11. This is the beginning of the main game loop. You remember the "while" loop, right? The variable we're testing, *play*, was initialized to *1* (or true) in Line 10. As long as it retains this value, the game loop will repeat.

Notice that we aren't using the statement *while (play = = 1)*. Any nonzero value is evaluated to true, therefore *play = =1* and *play* are really the same expression. The way to test for a false condition (0) is with the not operator: *while (!play)*.

The game loop is another example of structured programming. Each major task of the program has been allotted to a function. First we roll the dice, then we check to see if the player won, lost, or has to roll again.

If the call to check_roll() leaves the variable *win* in its zero state, then the second while loop is executed. The dice are rolled until *win* changes to *1* (win) or *-1* (lose).

The variable *win* is then tested in an "if" statement, and the appropriate message is relayed to the player. The percentage of games won is calculated, and the player is asked if he wishes to play again. If he answers with a *Y*, then play remains true and the game loop repeats. Otherwise, play becomes false, and the program returns you to the desktop.

Now the details. Start with Line 11. Here we initiate the main loop. As long as the expression in parentheses is true, the loop will repeat. Since we initialized *play* to *1*, we enter the loop.

The first thing we have to do in the loop is initialize a couple more variables. This is important, since the values of *first* and *roll* are passed to the function that "rolls" our dice.

The variable *first* is used as a flag to indicate if it's the player's first roll. What roll we're on is important. For example, a seven on the first roll is a winner. A seven on the second roll is a loser. The variable *roll* will hold the value of the current roll (except the first one). Line 14 is to call the function roll_dice() and places the value returned in *first_roll*.

Line 15 calls check_roll() and places its return into *win*. In order to evaluate the player's roll, this function needs some information. We're passing the information by giving it the values of first, first_roll and roll.

Line 16 changes the flag *first* to its false condition. If the player neither won nor lost with his first roll, then the value of *win* will still be *0*, and the second while loop, which begins on Line 17, will be performed.

See the *win = =0*? Why didn't I use the *while (!win)* construction as mentioned previously? There's really no reason, as far as the program goes. I used the former construction to make the program more readable. Using *!win* might make someone looking at the source code think that if *win* was *0* the player lost. This isn't true. A value of *0* means that the player hasn't won *and* he hasn't lost. It's a neutral state. If you want to use *!win*, go right ahead. It'll work just fine.

Line 19 calls roll_dice() a second time. This time, it assigns the value returned to the variable *roll*. We need this second variable, since we need to compare the first roll with all subsequent rolls.

Line 20 calls check_roll() again. If the value of *win* remains *0*—meaning the player still hasn't either won or lost his turn—the loop repeats. Once the player has managed to make his roll—or has blown it, with a seven or eleven—we exit the loop.

Line 22 will increment the game counter, *num_games*. We'll use this value to calculate the percentage of games won.

Lines 23 through 29 make up the body of an "if" statement. It uses the value contained in *win* to print the appropriate message to the player, as well as keep track of the number of wins.

If *win* is *-1*, the player has lost, and the program prints *You lose*—deep, huh? If *win* equals *1*, the player has won the game, and a statement of equal profundity is printed (sigh).

Also, the counter *num_win* is incremented, keeping track of the number of games our lucky player has managed to be victorious in. We're also calling a new library function here, *puts()*. This function is going to print the string argument contained in the parentheses. The main difference between the two, puts() and printf(), is that the former has no formatting options.

Last month, we just touched on the format of the "if" statement. This month, we're going to look at some much more complex examples. The statement we're looking at now is a slight variation of the one we saw in the last installment. The difference is the addition of the *else* portion.

Thinking back, you'll remember that the body of an "if" statement is performed only when the expression in the parentheses is true. When you add the *else*, the rules change just a bit. You now have a kind of "either/or" condition. If the expression being tested is true, the statements following the *if* and preceding the *else* will be performed. If the expression tested is false, the statements associated with the *else* are performed.

The syntax rules for the *else* are the same as for the *if*. If the body of the *else* portion consists of more than one statement, you must enclose them in brackets, and—remember—each statement must end with a semicolon.

Line 30 calls the function percent()—which prints out the percentage of games won.

Line 31 calls the function play_again()—to find out if the player wishes to continue or quit.

Digging deeper.

Now that we've taken a look at the general scheme of things, we can get into the details of each function.

The function roll_dice() does exactly as its name implies. The first thing you should take note of is the way this function is declared. There's something extra here. See what it is? Up till now, our functions have been declared simply by the function name.

Now the key word *int* has been added in front of the name. This specifies that the value to be returned by the function will be an integer. In this case, we could have left it off, since the default is always an integer.

But, if we wanted to return some other data type from a function, we must declare the type in the function definition *and* in the calling function. For instance, if we wanted to return a character from a function named ret_char(), we would first declare the function type in the calling function like this:

```
main()
{
   char ret_char();
```

Then the function declaration might look like this:

```
char ret_char(l, b)
int l, b;
```

The variables *l* and *b* are the values being passed to the function, and are included here only to differentiate between the two examples.

Lines 36 through 39 declare some local variables, print a prompt, and wait for a key press.

Line 40 gets a random number and places it in *d1*. Random() is a function specific to the ST and is an extension of the BIOS (Basic Input/Output System). It returns a 24-bit random number.

In our case, we need an integer. Take a good look at Line 40. See the *int* in parentheses? This is a "cast operator." What we're doing is forcing the return of Random() into a 16-bit integer, rather than doing it implicitly through automatic conversion (just leaving the cast operator out). In this particular case, the statement would've worked either way, but sometimes the difference can be critical.

Look at these two code segments:

```
int i;
i = 3.4 + 7.8

int i;
i = (int) 3.4 + (int) 7.8
```

In the first example, the addition is performed, yielding a result of *11.2*. Then, since the variable *i* is defined as an integer, the conversion from *float* to *int* is done automatically by trucation, making *i* equal to *11*.

In the second example, *3.4* and *7.8* are converted to integers before the addition is performed. This yields a result of *10*. Not quite the same answer.

Line 41 takes the value in *d1* and converts it to a positive number between 1 and 6, using modulo arithmetic and the absolute value function.

The abs() function is defined in the stdio.h file. It

looks and works exactly the way you've grown accustomed to in BASIC, returning the absolute value of a single argument.

The percent sign is the modulus operator. It is used only in integer arithmetic and yields the remainder when the number on the left is divided by the number on the right. For example, the expression *6 % 4* gives a result of *2*.

So, in Line 41, we're taking the absolute value of *d1* (in case we got a negative number from Random()), dividing it by six, then adding one to the remainder. Using six in the modulo math assures us we always get a remainder less than six (zero through five, to be exact). Adding one gives us our roll of the die (one through six).

Lines 42 and 43 get a value for the second die in the same manner.

The function then prints out the value of each die, as well as the total. The total *(t)* is then passed back to main().

Line 50 declares the function check_roll() as returning an integer. Three values are being passed to the function. Notice that the variables being passed (Line 15) and the variables that are accepting the values, have the same names. This is purely for reasons of clarity. They're still completely separate identities.

Now look at the body of the function. This is surely the most complex piece of code we've tackled yet. Basically, the whole thing is an "if" statement, but with layer upon layer. This function will give you great insight into the problems inherent in nested "if" statements.

Before we get too far into this function, I should introduce you to the *else if* construction. I mentioned previously that, with the "if...else" statement, we have an either/or situation. The *else if* takes this one step further, and allows us to add a test to the *else* portion of the statement. Look at this example:

```
if (exp1)
    statement1;
else if (exp2)
    statement2;
else
    statement3;
```

If exp1 is true, statement1 will be executed and the *elses* ignored. If exp1 is false, then exp2 is tested. If we get a true result, statement2 is executed and the final else is ignored. Finally if both exp1 and exp2 are false, statement3 is executed.

In check_roll(), we're using the flag variable *first* to decide which set of "rules" apply to the player's roll. If it's his first roll, *first* will be equal to *1*, and we'll go ahead and evaluate the second "if" statement, which checks to see if the roll was a seven or an eleven. If it was, the player wins. The flag *wn* is set to *1*, and program execution continues at Line 66.

See those vertical bars in the middle of Line 57? That's the *or* operator. Line 57 reads: *if first_roll equals seven or first_roll equals eleven*. The *or* operator yields a true result if one or more of the expressions are true. Here are a couple of examples. . . If we assume that *a* equals *1*, *b* equals *2*, and *c* equals *3*, then following expressions evaluate as shown:

```
a==1 || b==6              TRUE
a==4 || b==2              TRUE
a==1 || b==2              TRUE
a==2 || b==5              FALSE
a==3 || b==3 || c==3      TRUE
a==1 || b==5 || c==3      TRUE
a==2 || b==3 || c==4      FALSE
```

Continuing with check_roll(), if the roll wasn't a seven or eleven, we evaluate the *else if* portion of the statement. Here we check for a *2*, *3* or *12*. If we find one of these values, the player loses.

The flag *wn* is set to *-1*, and, as in the first case, program execution continues at Line 66. If neither of the previous conditions are true, *wn* retains its initialized value of *0* (Line 54), and, once again, the program continues at Line 66—which returns the value of the flag to main().

Whew! All that's only if the player's on his first roll. If *first* is *0*, program execution jumps to the "else if" statement on Line 62.

Before we continue, I'd like to see if I can help you avoid a good deal of teeth-knashing and hair-pulling in your future C programming. Look at those brackets on Lines 56 and 61. They're absolutely essential with nested "if" statements containing else constructions.

Without those brackets, the compiler has no way of knowing that the last two "else if" statements go with the outer "if" and not the inner. Keep in mind that the indenting is only cosmetic; it means absolutely nothing to the compiler. This is an easy trap to fall into, since the indenting makes everything so clear to the programmer.

Now let's take the second possible path in this function. If *first* is *0*, all the stuff between the brackets is skipped, and we continue at Line 62. This line checks to see if the player's roll was equal to his first. If it was, *wn* is set to *1* (win), and its value is returned to main() at Line 66.

If the first condition isn't true, we drop down to test the second. Line 64 checks for a roll of seven or eleven. If it evaluates to true, *wn* is set to *-1* (lose) and its value is returned at Line 66.

If none of the above conditions are met, the only thing that happens in this function is *wn* is set to *0* (Line 54) and its value is returned to main() (Line 66). The player has neither won nor lost, and must roll again. This process repeats until *wn*—and, subsequently, *win*—gets a nonzero value.

Moving on, Line 68 defines the function percent(). The word *VOID* in front of the function name indicates to the programmer that the function doesn't return a value. Like the *int*, it could've been left off.

VOID is defined in portab.h, which we included at the beginning of the program, and is really just an empty comment. In other words, even though we've declared percent() as *VOID*, it's still capable of returning an integer value. We're declaring it this way for the sake of clarity only.

This function does nothing more than calculate the percentage of games won and print the result out to the player. A few things should be said about Line 72, though.

First of all, in case it isn't obvious, the "/" (not to be confused with "\") is the division operator. The value on the left of the operator is divided by the value on the right.

You'll notice that the integer variables *num_win* and *num_games* are being cast to floating point. This is critical in this calculation. When you divide integers in C, you get an integer result; the decimal portion is truncated. If we allow this to happen with our percent calculation, we'll get two possible results, only one of which will be accurate. If we've won every game, then *num_win/num_games* will give us *1*, which multiplied times *100* equals *100%*. Fine and dandy.

But what happens if we've only won one game out of two? In integer division, *num_win/num_games* will give a result less than *1*. When the decimal portion is truncated, we'll end up with *0*. And what's *0* times *100*? It's certainly not *50%*, the result we want.

Okay, we're almost done. Just one more function to take a look at. The function play_again() is responsible for finding out if the player wants to play another game. There's really nothing very new here. Something that we had a brief encounter with was the way we're using getchar() in Line 80. We could rewrite this line as follows:

```
ch = getchar();
if (ch == 'Y' || ch == 'y')
```

One of the neat things about C is the way you can cram a lot of stuff on one line. Here, getchar() is called and its value is compared to the character *Y*. The variable *ch* now contains the value returned by getchar(), and it's compared to the character *y*.

If either of these compares finds a match, then the flag *p* is set to its true condition and returned into play, to be evaluated at Line 11. This way, the game repeats until the call to play_again() results in a *0*.

Breathing time.

That's it—class dismissed. If any of the program is still fuzzy to you, study up on it, especially the function check_roll(). When you feel you've got it all down pat, try your hand at writing a simple game.

How about that classic guess-the-number game? It should be fairly easy. Have the computer pick a random number between 1 and 100. As the player tries to guess the number, have the computer tell him whether he's too high or too low.

As for me, C you later. (Sorry about that.)

(Listing begins on next page)

```
#include <stdio.h>
#include <osbind.h>
#include <portab.h>

main()
{
   int first_roll, win, roll, play, first;
   int num_win = 0;
   int num_games = 0;
   int ch;

   play = 1;  win = 0;
   while (play)
      {
      first = 1;  roll = 0;
      first_roll = roll_dice();
      win = check_roll (first, first_roll, roll);
      first = 0;
      while (win == 0)
         {
         roll = roll_dice();
         win = check_roll (first, first_roll, roll);
         }
      ++num_games;
      if (win == -1)
         puts("You lose.  ");
      else
         {
         ++num_win;
         puts("You win!  ");
         }
      percent(num_games, num_win);
      play = play_again();
      }
}
int roll_dice()
{
   int d1,d2,t;
   int ch;

   puts ("Press space bar to roll\n");
   ch = getchar();
   d1 = (int) Random();
   d1 = abs(d1) % 6 + 1;
   d2 = (int) Random();
   d2 = abs(d2) % 6 + 1;
   printf ("Die #1: %d      ", d1);
   printf ("Die #2: %d\n\n", d2);
   t = d1 + d2;
   printf ("Your roll: %d\n\n", t);
   return (t);
}

int check_roll(first, first_roll, roll)
int first, first_roll, roll;
{
   int wn;

   wn = 0;
   if (first == 1)
      {
      if (first_roll == 7 || first_roll == 11)
         wn = 1;
      else if (first_roll == 2 || first_roll == 3 || first_roll == 12)
         wn = -1;
      }
   else if (first_roll == roll)
         wn = 1;
   else if (roll == 7 || roll == 11)
         wn = -1;
   return (wn);
}

VOID percent (num_games, num_win)
int num_games, num_win;
{
   float pc;
   pc = ((float) num_win / (float) num_games) * 100.0;
   printf ("You've won %d %% of the games\n", (int) pc);
}

int play_again ()
{
   int p;
   int ch;

   puts ("Play again? ");
   if ((ch = getchar()) == 'Y' || ch == 'y')
      p = 1;
   else
      p = 0;
   puts ("\n\n");
   return(p);
}
```

Word Processing on the Atari 520ST

by Arthur Leyenberger

The 16-bit Atari ST has been available to the public for nine months now, and there's no denying its success. The reason for its popularity is largely due to the amount of software available for the machine.

There's no category of software more widely used than word processing programs, regardless of which computer we talk about. This applies to the Atari ST, as well. Fortunately, there are plenty of programs to choose from, as you'll soon discover.

The business of processing words is relatively straightforward on most computers. Sure, some programs offer a few more features here, or an easier-to-use command set there, but, for the most part, if you've seen one word processor, you've seen them all.

However, with the introduction of computers like the Apple Macintosh—and now the Atari ST with its GEM interface—the business of processing words has, at least potentially, been made easier.

There are five word processing programs for the ST, which can be categorized as either "text-based" or "GEM-based." Text-based programs function like about any other word processor on any computer; they display only text on-screen. They typically require you to use commands or respond to menus, in order to edit, format and print your documents.

The GEM-based programs use the natural interface of the GEM desktop, with its icons, drop-down menus, windows and mouse control. The mouse may seem awkward at first. Some people claim that, since using a mouse requires that your hand leave the keyboard, it takes more time to use the mouse than it saves.

However, with a well-designed set of commands, a word processor that allows use of a mouse can be *easier* than a text-based program. After a couple hours of mousing around, most users find that they're accustomed to manipulating the mouse—it eventually becomes quite natural.

Text-based word processors.

ST-Writer — In the beginning, Atari said, "Let there be an ST." And there was, and it was good. Then the users said, "Let there be a word processor for the ST." And there wasn't, and it was bad. Then Atari scrambled; **ST-Writer** was the result—the first word processor for the ST computer.

ST-Writer is an all-text word processor, based primarily on the highly-rated **AtariWriter** for the 8-bit computers. **AtariWriter** has been around for a couple years now and has proven itself with many 8-bit owners. Anyone familiar with the program will find **ST-Writer** easy to slip into, comfortable as an old pair of slippers.

Instead of using the START, SELECT and OPTION keys of its predecessor, **ST-Writer** takes advantage of the ten function keys located across the top of the keyboard. The ESC, CTRL and ALTERNATE keys are also used, either individually or in conjunction with one of the function keys. Of course, the major difference is the eighty columns of text across your RGB or monochrome monitor screen.

On a color monitor, the control codes (used for special printer codes, end of paragraph carriage returns and option settings) are easily found, displayed in red. At the top of the **ST-Writer** screen is the command line, with red letters and white numbers. The screen is clear, too, with text in white on black—or the reverse, if you like.

On a monochrome monitor, text can also be displayed in either white on black or black on white. And, when using **ST-Writer** with a monochrome screen, you can choose to display either 20 or 40 lines on-screen at once.

There are a number of improvements Atari's made over the original. From the menu screen, you can now: format a

NEW

NEW

FOR THE ATARI ST

* **LogiKhron Clock Card** * Are you tired of resetting the date and time when the Atari ST is turned on? Then LogiKhron is for you! It automatically enters the date and time into the Atari ST each time the computer boots. Through the use of an internal battery, LogiKhron maintains the precise time and date even when the computer is off. Installation is easy, just insert LogiKhron into the left cartridge slot and enjoy the benefits. — Only $49.95

---

* **Electro Solitaire & 21** * Transform your computer into the perfect card playing friend. Why hassle with searching for a deck of cards, shuffling, and dealing if your computer can do it for you? Electro Solitaire & 21 adds a new dimension to the ever popular games of Solitaire and Blackjack. Just think, point, and play; it's as easy as that? — Only $19.95.

---

* **The Study Guide** * The Study Guide is a carefully designed educational tool. You enter the information which you want to learn and the computer helps you learn it. Questions are then organized into formats such as multiple choice, true/false, essay, or fill in the blank. You can then compare your answers against the computer's to evaluate your success. For ages 11 to Adult. —$39.95

---

* **Electro Calendar** * Electro Calendar is an enhanced version of your wall calendar. You can store and retrieve important messages quickly and easily. For instance, it can remind you of upcoming meetings, birthdays, anniversaries, or any important event. If your wall calendar becomes outdated, let Electro Calendar print one for you. It will display or print calendars between the year 0001 and 9999. — Only $39.95

For more information or to place an order, call (314) 894-8608. Dealer inquiries invited.

Sorry, no Visa or MC accepted.

SL Soft Logik Corp.™

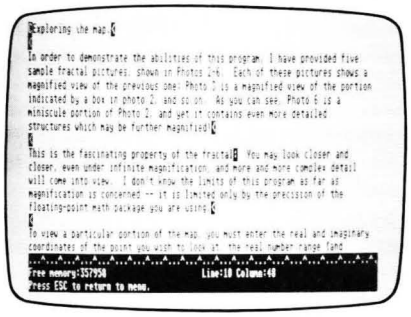
4129 OLD BAUMGARTNER • ST. LOUIS, MO 63129 • (314) 894-8608

CIRCLE #131 ON READER SERVICE CARD

disk; print to either the screen, a disk file or a printer; and receive files directly transmitted from an 8-bit Atari.

In the edit screen, the number of free memory is always displayed—over 150-thousand bytes free with RAM TOS, or over 350-thousand free bytes with TOS on ROM. Further, you can mark blocks of text and save them to disk.

There are other ST-specific features worthy of note. The program allows you to use files located in subdirectories, on any disk, and it automatically keeps track of which directory you're looking at when doing a file save or a disk index.



ST-Writer.

ST-Writer is a what-you-see-is-what-you-get word processor. You can, therefore, see on-screen how single- or double-column formatting will look before you print it out. . . a very useful feature.

Documentation consists of several files as freely available as is the program itself. There are tutorial, reference, quick reference and function key label files available.

In addition, a couple of other files are needed before the program can be effectively used. There's a configuration program that uses a specific configuration file to create the "XYZZX" printer driver file that **ST-Writer** looks for when told to print. If you're familiar with ASCII printer control codes, you can modify the source driver file so that **ST-Writer** works with your non-Epson-compatible printer.

ST-Writer contains just about every feature most users would need. There are a few commands not offered—like ability to move the cursor from word to word or paragraph to paragraph, the ability to delete a word or paragraph at a time, or the capability to use multiple windows for text.

And there are a few problems with **ST-Writer**. For example, cursor movement is slower as your file gets bigger. There's a lengthy conversion time required to import a non-**ST-Writer** file. For a seemingly small file, it can take a couple minutes for the screen to return to normal.

There are also a few bugs that mysteriously lock up the computer at times—and, just as mysteriously, have no effect at other times.

One of the bugs rears its ugly head when you try to print double-column output. It seems, when a carriage return occurs at the end of a paragraph in the left column, the right column margins get out of sync. Another bug prevents you from blocking the headers flush against the right margin. In fact, I've had some problems trying to print headers in general.

ST-Writer appears a reasonable choice for a word processor, considering its price and availability. Unfortunately, Atari's limited resources will probably prevent them from correcting bugs or adding features regularly. So, although it's free, you must decide if you want to trust your important word processing work to a program which Atari has no stake in maintaining or improving.

Regent Word.

While various versions of **ST-Writer** were making the rounds, **Regent Word** appeared on the scene, an easy-to-use but powerful word processor. **Regent Word** is from the same folks who developed **AtariWriter** and **AtariWriter Plus**—Regent Software.

The program is billed as the first full-function word processor for the ST computer. It was surely one of the first and is clearly full of features.

Regent Word is menu driven and will work on either a monochrome or RGB monitor. One of its key features is the built-in "help" function. By pressing HELP on the keyboard, the user can display one of five menu screens at any time. Pressing the SPACE BAR cycles through the menus, and the UNDO key exits the menus.

The multiple menus in **Regent Word** aren't menus *per se*. Rather, they're help screens that tell you what the commands are. Since most commands are invoked by a single keystroke, the complexity of help menus and commands is not burdensome.

Commands are issued by a variety of two-key presses with the ALTERNATE, CTRL and ESC keys. Also, function keys across the top of the keyboard have been defined for block operations.

Like other ST word processors, **Regent Word** offers 80-column editing and the usual. The RETURN key is used only at the end of a paragraph; word wrap occurs at the end of each line; and DELETE and BACKSPACE work as expected.

The cursor can be positioned throughout text by a character or screen at a time. In addition, you can move directly to the top or bottom of the text with one command.

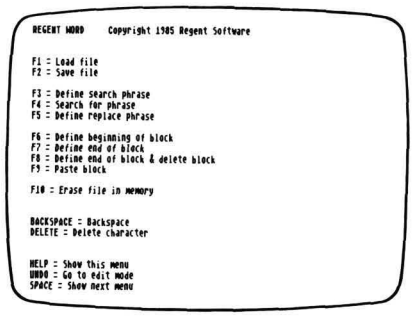
The print preview function shows how your underlined, elongated and bold print will look before you actually print it. Super- and subscripts, headers, footers, justification and page numbering are also supported.

Much like **AtariWriter** word processors, **Regent Word** allows you to access all the features of your printer control codes and options. Using CTRL and various letters, such attributes as line spacing, print style, centering, headers/footers, margins and justification can be set. In addition, specific ASCII printer

codes can be sent directly to the printer from anywhere in the document.

A number of features in **Regent Word** are especially useful and well implemented. When the program's first run, you're asked to enter the date and time. Given the lack of battery backup and the unreliable clock in the ST, this feature ensures that any files created or edited with **Regent Word** will have the correct date and time stamp. During editing, the time is continually displayed at the top of the screen.

When loading a file, all files on the current disk are displayed on-screen, with a letter next to each. Pressing the letter corresponding to the file you want will load that file into memory. In addition, the directory also shows total storage available on the disk, plus creation dates and sizes of individual files.



Regent Word.

Another useful **Regent Word** feature is its ability to give you a word count of the file currently in memory at any time. The program also has a built-in communication feature, allowing documents to be sent or received via the communications port on the ST.

Currently, **Regent Word** is designed for Epson dot-matrix printers (or compatibles) and the Juki letter-quality printer. The program will use however much memory is available (an Atari 520ST with TOS in ROM yields almost 400K free for text files). Also, the size of the document does not affect cursor speed.

Copy protected **Regent Word** sells for $50. Fortunately, a backup disk can be ordered in either single- or double-sided versions for only $5.00. Still, there's no way to put this word processor on your hard disk. Overall, **Regent Word** is a powerful, easy-to-use, text-based word processor, just as advertised.

Regent Software also has a product called **Regent Spell**, a 30,000-word spelling checker that can be used with **Regent Word** or other word processors. As carefully designed and thought out as **Regent Word** is, though, **Regent Spell** seems the opposite. Perhaps it was rushed to market.

It's not that the program has bugs. On the contrary, it's fast and seems bug-free. The problem lies with the number of small irritations and feature omissions.

For example, there's no way to quit a spelling check before the program has reached the end of the file. And, since there's no word count or visual indication of progress, you have no idea how long the check will take.

Further, there's no display of the filename and no method to return to the last word checked. **Regent Spell** is a one-way program. There's no mass dictionary update, to let you take a file you know is good and have it added to the program's list of words.

Finally, although you can use some features of the program from the mouse, you must leave the mouse and press RETURN after clicking on INSERT or DELETE.

Regent Spell sells for $40 (from Regent Software, 7131 Owensmouth, Suite 45A, Canoga Park, CA 91303 — (818) 883-0951) and is copy protected. As it stands now, the program works, but the number of irritations makes it difficult to use. In discussing these problems with Regent Software, I was assured that most (if not all) of my complaints would be incorporated into the next version of the program.

I have no reason to doubt their word. If these and a few other minor changes are made, **Regent Spell** could well be the *de facto* spelling checker for the ST.

Final Word.

It's somewhat difficult to describe Mark of the Unicorn's **Final Word**. Frankly, I don't know where to start. The program's a sophisticated word processor that'll probably require a substantial hunk of time to master. But, having digested the tome-length documentation, you'll find yourself amazed by the power at your word processing fingertips.

Final Word comes with almost 500 pages of documentation accompanying the two single-sided program disks. Part of this is a 150-page tutorial that must be followed chapter by chapter if you expect to get anywhere past turning on your ST.

In addition, a terse, 4-page installation guide tells you how to get this program up and running. The only specific reference to the Atari ST occurs in this little guide. As with the PC version (the original), the whole kit and caboodle comes in an attractive black slipcase.

One of the unique aspects of **Final Word** is that it's crashproof. Unlike other word processors, where you *could* lose your work, **Final Word** automatically saves your work into a buffer file whenever you stop typing for a period of ten seconds. If your computer loses power, all you've lost is ten seconds' work.

Another feature is split-screen editing. With a single command, you can split the screen into two parts. And, since **Final Word** can edit up to twelve documents at once, you could display parts of various files in one window, while editing your document in the other. Or you could keep a rough draft in one window as you polish off the final version in the other. Text can easily be copied or moved between windows.

Final Word is really two word processors in one. Either may be chosen, based upon the needs of the task at hand. The on-screen Editor is a what-you-see-is-what-you-get word processing editor. You type and edit the text on-screen, and it appears exactly as it will on the final, printed copy.

The other editor, called the Advanced Formatter, allows you to specify how your finished documents will look, by including formatting commands in the body of the document.

These commands are inserted into the document as you type and edit, but you'll only see how it looks when the document is printed. With the Advanced Formatter, you can produce serious writing, including table of contents, index and footnotes.

Final Word contains a series of menus, grouped into a main menu and ten function menus. The main menu lists all submenus. When displayed, it partially obscures your text. After you're familiar with the program, you can turn the menu feature off.

After a brief exposure to **Final Word**, you'll get used to seeing the message *swapping*, which occasionally flashes at the bottom of the screen. When this message is displayed, the program is copying the text from memory into a disk file called the "swap file."

Swapping is automatic whenever you stop typing for ten seconds. You can change the time interval to suit your needs. It's this swap file that contains your text—when the power fails, the

computer crashes, or you forget to save a file. Should you need to retrieve lost text, a recover program is run to re-establish where you left off.

Final Word is truly a powerful program. For the power writer, or someone who needs sophisticated word processing features, this one should be sufficient.

Some of the program's heavy duty features include: precise control of headers and footers; the ability to place footnotes at the bottom of each page, at the end of the document or embedded in the same line as their reference; automatic chapter, section and "new page" formatting controls; the ability to create lists, numbered or not, within the body of the text; and the ability to create a table of contents and index.

The **Final Word** tutorial is what makes this complex program manageable. Although a little chatty at times, it uses good training techniques—beginning each section with a list of objectives, providing plenty of examples during the tutorial, then summarizing what was learned at the end of the section. The reference manual is complete, too.

There are a few flaws that you should be aware of. The program is copy protected. You can run it from a copy, but one of the two original disks (used as a "key disk") must be in drive A whenever you start up. Fortunately, the ability to change just about every parameter and option of the program allows you to put the multitude of necessary programs on a double-sided disk or, better yet, a hard disk.

Another problem: for all its power, **Final Word** does not allow double-column printing. Newsletter editors should look elsewhere if that's a major requirement. Further, the manual assumes you're working on a CP/M or MS-DOS system. For the uninitiated, this could be a little confusing.

Probably the most serious weakness in the program is the extremely tedious setup—and the sheer number of program files. To get to the point where you can just type one word from within the program requires a seemingly endless checklist. And heaven forbid you should need to recover a file. The first few times you try to run "recover," you'll be quivering in your boots.

Final Word sells for $125 and works with a variety of dot-matrix and letter-quality printers. Although it can be used with a single disk drive, the less patient Atari ST user will opt for two drives or a hard disk.

If you need the word processing power to go along with your "Power without the Price," **Final Word** is the only current alternative. It's available from Mark of the Unicorn, 222 Third Street, Cambridge, Ma 02142 — (617) 576-2760.

GEM-based word processors.

HabaWriter is the first GEM-based word processing program for the Atari ST. All the features of the GEM interface—like drop-down menus and mouse control of the cursor—are available.

However, the first release of the program, version 1.0, was somewhat "bug

gy." Files were occasionally lost, the screen froze, and the cursor tended to disappear. Fatal bugs have been eliminated in version 1.1 and a number of enhancements made to the program. This review is based on version 1.1.

HabaWriter is copy protected, but you can copy the program files to another disk. As long as the original disk is in drive A when you start the program, it will start. . .then you can remove the original "key disk."

When the program begins, a familiar GEM window appears on-screen, with a menu bar across the top. Desktop accessories are available under the "desk menu," assuming the .ACC files were on your disk when you first booted up your ST.

The menu bar contains **HabaWriter** commands, and each menu has related commands. The menus labeled File, Edit, Search, Format, Style and Print are all descriptive of commands they contain, and everything is logically laid out. The function keys on the ST keyboard can also be used to select commands.

The **HabaWriter** document window is a regular Atari ST window. You can change its size, move it around the desktop and close it by clicking on the close box.

You can also use the vertical and horizontal slider bars to move throughout the document currently in the window. Up to six windows can be open at a time, and there's a clipboard for cutting and pasting parts of (or entire) documents.

To load a previously created file, the user selects the "open" command from the File menu. A GEM file selector box appears, and a file can be chosen, or the drive and directory can be changed.

As the file loads, a small, square box much like a meter appears on-screen, showing the progress of reading the file.

HabaWriter lets you work with many documents at once. The procedure for opening another existing document is as described above, or you can create a new document.

The multiple window technique is very handy for taking notes while you're writing. As you think of ideas you want to save, you can click on your other window, type your thoughts and return to the spot where you left off.

You can copy or cut and paste text between or within documents. Once the text is highlighted, by dragging the mouse from start to end, the block can be moved directly to another location or document, or to the clipboard. An entire file can also be pasted into a document.

The document format can be set, changed and even saved with the file. Each time the document's loaded in the future, settings will be loaded along with the text.

Margins, indentation, justification, type styles, tabs and preferences can all be specified. **HabaWriter** supports ASCII or "word processing mode" files, and sub- and superscripts.

There are a few features **HabaWriter** version 1.1 lacks. Currently, there's no way to change the line spacing to anything other than single-space mode. Headers and footers aren't supported. There are no fonts at this time—just underline, bold and normal text.

The program has trouble reformatting paragraphs or documents in ASCII mode. In fact, the feature only works properly in "HabaWriter" mode on text entered with the program. The Search and Replace function can't handle ASCII characters like carriage returns (CTRL-M).

The DELETE key on the ST keyboard does not work (frustrating!), and, finally, there's no index in the 46-page manual.

On the positive side, Haba is making an effort to produce quality software. Version 1.1 of the program corrected earlier bugs and added features, such as the ability to create your own printer configuration file, the ability to send ASCII printer codes to the printer, and sub- and superscripts.

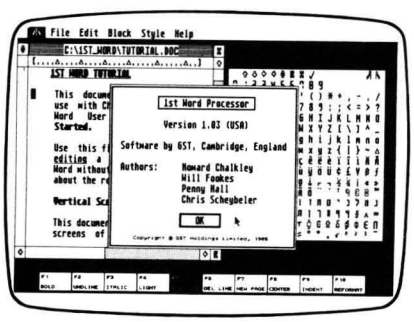
I'm told that version 1.2 of **HabaWriter** will have variable line spacing, headers and footers, and multiple printer configuration files on the distribution disk.

I'd also like to mention Haba's on-line support on CompuServe. Gerry Humphrey from the product development staff is very active on the 16-bit SIG (pcs 58).

Not only does he answer all questions courteously, but he's interested in feedback on this and other Haba products, so they can be enhanced to give you the best products possible. If you have any questions, suggestions or complaints on **HabaWriter**, leave Gerry a message. He'll get back to you quickly.

The on-line customer support provided by Haba is another indication that Haba wants to be your word processing company. I don't know of any other company currently marketing a word processor for the Atari ST which provides this level of support. Given the current version of the program and the planned changes, **HabaWriter** could easily become *the* ST word processor for a lot of people.

HabaWriter works in either high or medium resolution on the ST. The program lists for $80, and Version 1.2 should be available by the time you read this. Existing owners can receive an upgrade by sending $7.50 and their original disk to Haba Systems, 6711 Valjean Avenue, Van Nuys, CA 91406 — (818) 901-8828, attention Technical Support.



1st Word.

1st Word is a GEM-based word processor developed by GST Holdings, Ltd., an English company. The program was originally provided free by Atari for ST purchasers during the Christmas 1985 season. It's now sold separately for $40 and is still being marketed by Atari Corp., 1196 Borregas Avenue, Sunnyvale, CA 94086 — (408) 745-2021.

Like **HabaWriter**, **1st Word** takes full advantage of user-oriented GEM features like drop-down menus, icons and windows. Editing tasks, such as cut and paste or changes in document layout and style, can be performed with the mouse. The documentation consists of a 42-page file on the program disk.

1st Word isn't copy protected, so it can easily be placed on the higher capacity, double-sided disks or a hard disk. The program works in low, medium and high resolution, but only medium- and high-resolution screens will provide you with 80-column text format.

A number of files come on the distribution disk. In addition to the **1st Word** program and resource file, there's a program called 1st Print, to allow you to print while using **1st Word**. You can't print a file while there's an open window on-screen, though. Trying to run

the printer program from the GEM desktop will not work.

There's a user guide file and short "tutorial file" that's nothing more than a short example. Three printer drivers—Epson, Qume and plain ASCII—are included on the disk.

In addition, an install program's available, to let you create the necessary printer driver file after you've edited the hex source file. This program creates either a .DOT or a .DSY file, for use with dot-matrix or daisy-wheel printers, respectively.

The operation of **1st Word** is quite similar to that of **HabaWriter**, with a couple notable exceptions. In addition to the menu bar at the top of the screen, which contains various commands in separate drop-down menus, there's a set of function key icons across the bottom, to indicate the word processing functions assigned to each function key.

Keys F1 through F5 indicate toggle settings of bold, underlined, italic, light and insert mode. The other five keys perform actions when editing: line deletion, new page, line centering, and paragraph indenting and formatting.

A particularly useful feature of **1st Word** is that these function keys can be operated either from the keyboard or by clicking the mouse on the appropriate key icon.

Another significant difference between **HabaWriter** and **1st Word** is the font table on-screen, partially obscured by the primary GEM window. This table holds the ST character font, which may vary from country to country. Since not all the ST's 256 characters are available from the keyboard, this table allows you to select any character you want, simply by clicking the mouse on the correct position.

1st Word will copy that character to your window at the current cursor position. This font table is only available in medium and high resolution.

There are a few things I *don't* like about **1st Word**. . . Like **HabaWriter**, it only lets you single-space documents. The program also has problems reformatting text from an ASCII file. And there's a silly page eject each time you print a document, which can't be surpressed . There are no extra fonts (like Macintosh uses) available for the program.

On the favorable side, **1st Word** allows the use of headers and footers, although they can only be one liners. The program supports page numbering, conditional page breaks, the use of position markers, and cut and paste operations. As mentioned before, it's not copy protected, so it can easily be backed up and installed on a hard disk.

1st Word is an easy-to-use and capable word processing program for the Atari ST. I'm sure we'll see more sophisticated GEM-based word processors become available, but, until then, **1st Word** will let you do a lot of writing.

GEM Write is a word processor for the ST, which has been promised since the ST was first announced. Atari said that several GEM programs would be available when the computer hit the streets in the summer of 1985. Similar to the Apple Macintosh, which comes with a decent word processor and drawing program, the ST was to arrive with some software. But, somewhere between there and here, the GEM programs got waylaid.

For the last several months, I've been repeatedly asking *both* Atari and Digital Research when we could expect **GEM Write** and **GEM Draw**. Both have been available for the IBM PC market for quite a while.

The response I get from either party is the same—it's the other guy who's holding things up.

If you believe the scuttlebut, Atari wants to pay a lower price than they first agreed to (seems believable). And DRI wants Atari to pay them for some other stuff, before they turn over the programs (also believable).

The debate won't be settled here. Having used **GEM Write** on a PC, and assuming that it'll appear for the ST *sometime*, I thought it would be appropriate to give it a brief mention here.

GEM Write looks like **HabaWriter** and **1st Word**, but isn't as functional. Sure, it uses a mouse and windowing environment like the others, but it allows only one open window at a time. You can't do something as rudimentary as mixing formats to produce, say, a single-spaced paragraph in a double-spaced document. Also, there's no way to pass control codes to your print or delete files from within the program.

GEM Write allows you to use either drop-down menus or CTRL-key sequences for issuing commands. The latter are very similar to those used by **Wordstar**, a popular word processor for the PC. **GEM Write** supports pagination, justification, single-line headers and

footers, block editing, and search and replace functions.

Probably its strongest feature is the ability to merge a picture into the body of a document. **GEM Paint** files can be brought into the program and placed where you like. Then the entire document can be saved, or printed as one entity. This is how it works on the PC. The availability of **GEM Paint** for the ST is as questionable as is that of **GEM Write** itself.

If you need the ability to merge text with graphics, then you may be inclined to wait around for the ST **GEM Write**. Otherwise, existing GEM-based word processors, such as **HabaWriter** and **1st Word** easily outdistance **GEM Write**'s simple-minded features.

Which is for you?

This comparison review was based on many, many hours of intimate, personal, first-hand use (and, in some instances, abuse) with the currently available word processors for the Atari ST. Having had the chance to get familiar with the ST over the last nine months, and having used the various word processing programs over the last several, has allowed me try each program and form opinions based on empirical data.

I began this project with no favorite programs or axes to grind. I've used dozens of word processing programs on various computers over the last few years. In addition, I spend my daily working hours evaluating hardware and software from the user's viewpoint.

I feel strongly that, for an application program to be the best it can be, it must be designed with a particular computer in mind from the start. A word processing program is limited by the hardware it runs on. If it doesn't take advantage of all the hardware has to offer, it's not an optimum program.

With regard to word processing on the Atari ST, only a GEM-based program can offer the user the most.

My choice, of the programs compared in this article, is **1st Word**. With the exception of a few flaws (which, unless resolved, limit its usefulness), **1st Word** provides the ease of use, power and flexibility that an ST word processing program should.

If you were lucky enough to receive **1st Word** during the 1985 Christmas season for free, all the better. But, even at its list price of $40, it represents what I think is the best ST word processor.

1st Word should satisfy the word processing needs of the majority of ST owners. For that matter, **HabaWriter** would probably be okay, too (especially version 1.2). However, it's slightly more expensive and is copy protected.

If you simply *must* have a text-based word processor, then you can't beat the price of **ST-Writer**. It's freely available on many bulletin boards and on several information services.

Regent Word works as advertised, seems bug-free and is straightforward to use. Multiple menus, which can be accessed in an instant, take you through the assorted commands. There's no need to be a cartographer to navigate your way around this program.

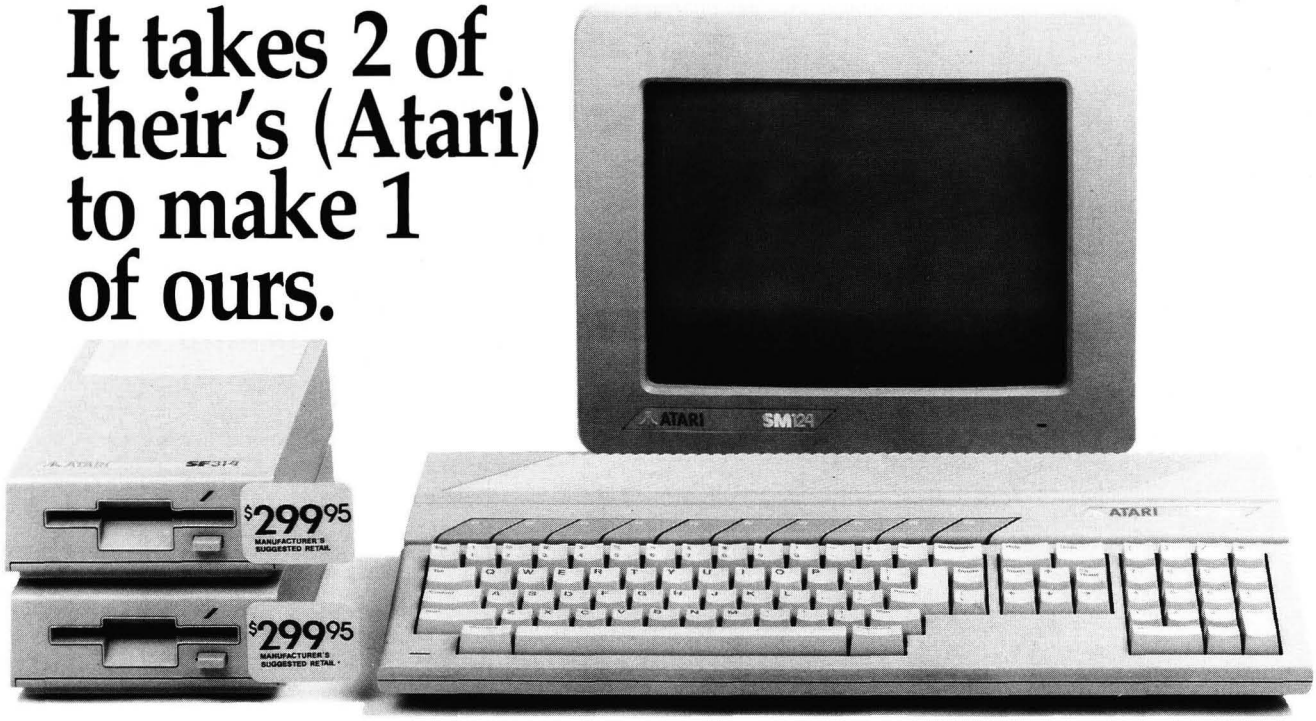
Final Word, from Mark of the Unicorn, is *the* most, of all of these programs. It is the most expensive, the most sophisticated, the most time-consuming to learn. It also weighs the most.

Kidding aside, if you need this much power and are willing to spend some time learning the intricacies of it, **Final Word** may be just right for you.

That's my opinion, based upon hours of use. Of course, the final decision rests with you, since you'll be the one who either blesses or curses the day you brought a particular word processor home to your ST.

ST INDEX TO ADVERTISERS

It takes 2 of their's (Atari) to make 1 of ours.



Why pay more — FOR LESS!

THE SHANNER SD-2000
Available Now!

Why buy 2 Atari 1 Mb. Drives, separately packed, encased in large plastic housings for $599.90, when you can purchase the SD-2000 Drive System for $399.95. The SHANNER SD-2000 is comprised of two 3.5″, 1 Mb. double sided, double density drives packaged in an attractive, coordinating grey METAL housing and is fully compatible with the Atari ST.*

You can pay Atari or you can SAVE $200.00.

You be the judge.

$399.95

*Atari and Atari ST are trademarks of Atari Corporation.

GUMBALL EXPRESS ORDER FORM • For FAST delivery use this order form or call **TOLL FREE 800/423-9442**

Product Description	Price	P&L	TOTAL
Shanner SD-2000	$399.95	$7.50	**$407.45**

If you currently own either an Atari SF354 or SF314, you can use your existing power supply. If not, please order Model No. SD-2000PS for $14.95.

☐ Check here and add $14.95 to $399.95

All products will be shipped prepaid UPS ground.

☐ Check enclosed. (NOTE— order will be shipped when check clears).

Make check payable to:
Gumball Express
707 S.W. Washington Street Suite 200
Portland, Oregon 97205

☐ VISA ☐ MASTERCARD ☐ INTERBANK (MasterCard only)

Name on card _____

Account # _____

Expiration Date _____

Signature _____

SHIP TO:

Name _____

Address _____

City _____ State _____ Zip _____

C.O.D.'s and purchase orders will not be accepted by Gumball Express. Outside the USA add $10. and make payment by bank draft, payable in U.S. dollars drawn on a U.S. bank.