ST-LOG™

THE ATARI ST OPERATOR'S MAGAZINE

APRIL 1986

PREMIER ISSUE

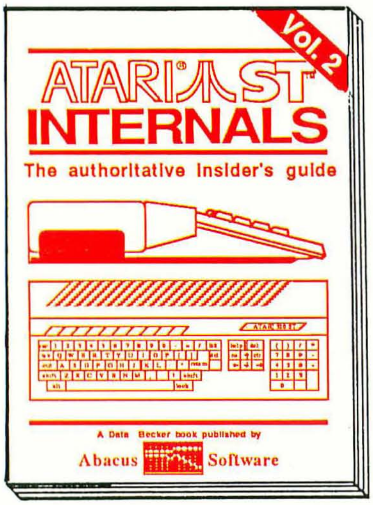
Mr. Scratch

GEMSYS()

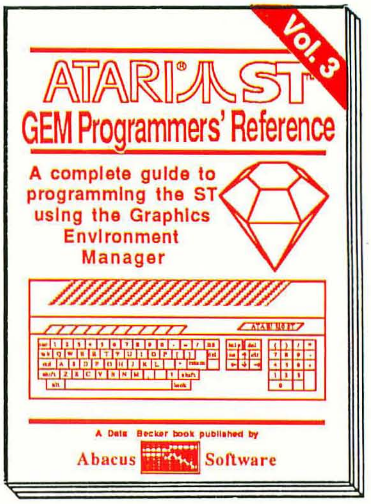
ST-Check

ATARI® ST™

REFERENCE LIBRARY


Vol. 2
ATARI® ST
INTERNALS
The authoritative insider's guide
A Data Becker book published by
Abacus Software

Essential guide to learning the inside information on the ATARI ST. Written for the user who wants thorough and complete descriptions of the inner workings of the ST. Detailed descriptions of the sound and graphics chips, the internal hardware, the Centronics and RS-232 ports, GEM, important system addresses and plenty more. Also included is a complete documented BIOS assembly listing. This indispensible reference is a required addition to your ATARI ST library. 450 pages. $19.95


Vol. 3
ATARI® ST
GEM Programmers' Reference
A complete guide to programming the ST using the Graphics Environment Manager
A Data Becker book published by
Abacus Software

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

TRICKS & TIPS
Treasure trove of fascinating tips and tricks allows you to make full use of your ATARI ST. Fantastic graphics, refining programs in BASIC, assembler, and C. Includes program listings for RAM disk, printer spooler and more. $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 1200 dealers nationwide.

CIRCLE #118 ON READER SERVICE CARD

ST-LOG
THE ATARI ST OPERATOR'S MAGAZINE

PREMIER EDITION
APRIL 1986

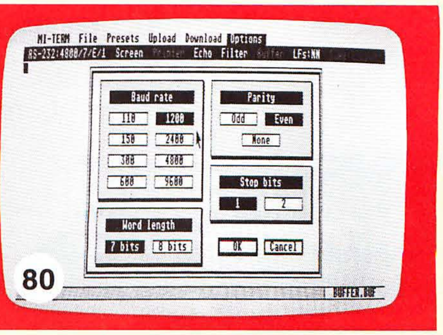
FEATURES

REVIEWS

COLUMNS

57

76


MI-TERM File Presets Upload Download Options
Screen Printer Echo Filter Buffer
OK Cancel

RS-232:4800/7/E/1 LFs:NN

Baud rate: 110 1200 150 2400 300 4800 600 9600

Parity: Odd Even None

Stop bits: 1 2

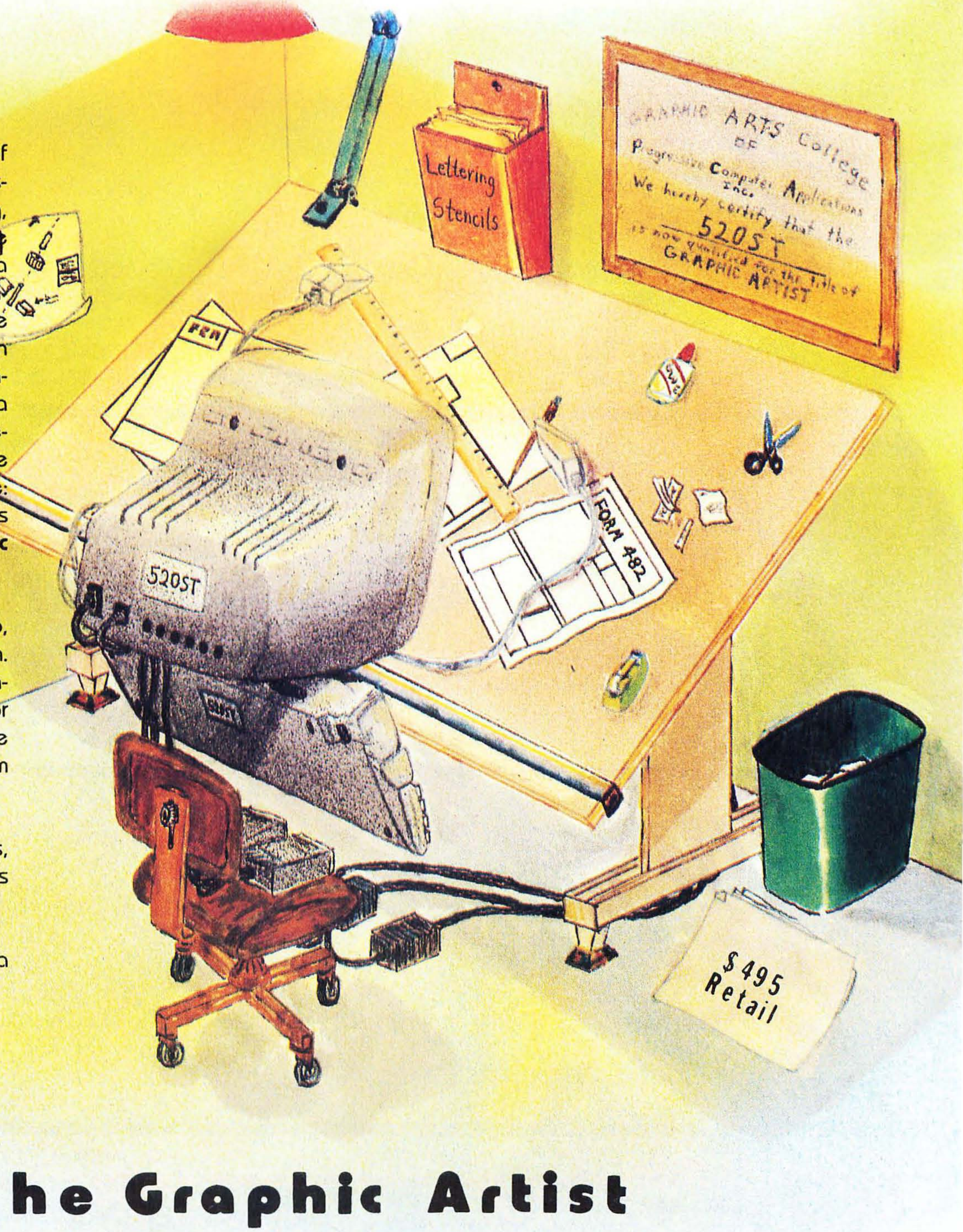
Word length: 7 bits 8 bits

BUFFER.BUF

80

Graphic Arts

The Next Step in the Evolution of Software


Lettering Stencils
GRAPHIC ARTS College OF Progressive Computer Applications Inc.
We hereby certify that the 520ST is now qualified for the title of GRAPHIC ARTIST
PCA
520ST
FORM 482
$495 Retail

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.

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.

ST-Check

A checksum program for the 520ST

by Clayton Walnum

This issue marks the first appearance, within **ANALOG Computing**'s pages, of a BASIC program for the 520ST. Those of you who spend a good deal of your free time typing in the listings from the magazine have surely grown accustomed to seeing (and, we hope, using) the checksum data that follows each BASIC listing. One of the first projects I undertook when ST BASIC made its appearance was to make sure that this tradition continued. The result is **ST-CHECK**—a typing validator for the 520ST computer.

What good is it?

Typing in a program listing can be a frustrating and time-consuming task. Just one mistyped character will frequently render a program completely unusable. To insure that your program will run correctly, the entire listing must be checked character by character against the original. This can take many hours. To make matters worse, you can't trust your own eyes. Do you know how easy it is to overlook an *O* where a *0* is supposed to be?

Typing checkers like **Unicheck** (latest publication in issue 39) and **ST-Check** take over the arduous task of proofreading your program files. Using these programs can cut down your debugging time by a huge factor. When the checker's output matches that published with the listing, you can be sure your typing is accurate.

Getting started.

Load your copy of ST BASIC, then type in the listing that accompanies this article. When you've finished, save a copy to your disk.

Now, the bad news. There's no foolproof way **ST-Check** can find typos within itself. An error in the program will make all data suspect. So why have I included the checksum data with the program listing? Because, though you can't get much help *finding* your errors, you *can* be sure when none exist.

Introspection.

When you run **ST-Check** against itself, you will get one of several results. The program may just give up and crash. In that case, go through the listing character by character until you find your mistake.

A second possibility is that the program will run okay, but will create all bad data. This may indicate an error somewhere between Lines 80 and 420. Find the typo and correct it. A last possibility is that the checksum data will have only a few bad values. In this case, use the normal method detailed below to locate your errors.

Warning: until you get your checksum data for **ST-Check** to match the data following the listing, you can't trust it to proofread other programs.

Using ST-Check.

When you finish typing an ST BASIC program listing from the magazine, save a copy to your disk, then run **ST-Check**.

// ST-Check *continued*

The program will first ask for a filename. Type in the name for the program you wish checked (the one you just saved to the disk), then press RETURN. You'll be asked for a "bug" name. Enter a name for the checksum file (this can be any name not already on the disk), followed by RETURN. Hint: if you include a *.BAS* extension on your bug filename, you'll be able to view the generated data without leaving BASIC.

ST-Check will now proofread the program. When the checking process is complete, you'll have a file on your disk (saved under your bug name) which contains the checksum data for the program checked.

If you added the *.BAS* extension, you may now load this file and view it. If you didn't use the *.BAS* extension, you must return to the desktop, double-click the bug file, then click the "show" command.

Check the last value of each line. If it matches the value in the published checksum data, then go on to the next. If it doesn't match, you've got a typo.

To find the error, look at the line number of the data statement in which the bad value occurred. This number is equivalent to the first program line the data evaluates. Let's call this "Line X." Count the entries in the data line until you get to the bad value. We'll call this count "Y." Now look at the program you typed in. Starting with, and including, Line X, count down Y lines. The line you end up on will be the one containing the typo.

Correct the error, then rerun **ST-Check**. When you get all the checksum data to match that in the magazine, your new program is ready to run.

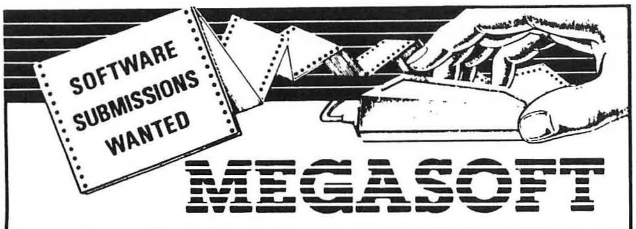
Passing the buck.

Okay, friends. Here's where the truth comes to the fore. I can take only minimal credit for **ST-Check**, as it's virtually a direct translation from **D:CHECK2** (issue 16) by Istvan Mohos and Tom Hudson. All accolades and tribute should be directed to those two fine gentlemen. I'm sure they'll divvy it up fairly, and perhaps pass a small share on to me. Thanks, guys!

You may now type in this month's ST BASIC program, secure in the knowledge that the searching eye of **ST-Check** is primed and ready. ▣

Listing 1.
BASIC listing.

```
10     'ST CHECK typing validator by Cl
ayton Walnum
20     'based on a program by Istvan Mo
hos and Tom Hudson
30     if peek(systab)=1 then cl=17 els
e cl=32
40     fullw 2:clearw 2:gotoxy cl,0:? "
ST CHECK":ex=0:sp=0:x=0
50     input "Enter filename: ",f$:inpu
t "Enter BUG name: ",f1$
60     on error goto 590:open "O",#1,f1
$:open "I",#2,f$:close #2
70     open "I",#2,f$:on x goto 140,220
80     color 2:?:? "Counting lines":lin
ecount=0:color 1
90     on error goto 570
100    line input#2,i$:linecount=lineco
unt+1
110    ? ".";:goto 100
120    close #2:q=int(linecount/10):dim
 c(linecount),r(q)
130    x=1:goto 70
140    range=0:lyne=0:color 2:?:?:? "Fi
lling array":color 1
150    ? ".";:count=0
160    line input#2,i$:count=count+1
170    lyne=val(i$):r(range)=lyne:range
=range+1
180    on error goto 580
190    line input#2,i$:count=count+1:if
 count=10 then 150
200    goto 190
210    close #2:x=2:goto 70
220    color 2:?:?:? "Calculating check
sums":color 1
240    for i=1 to linecount:checksum=0:
line input #2,i$
250    for z=1 to len(i$):number=asc(mi
d$(i$,z,1))
```



```
260    if number=asc(" ") and ex=0 and
sp=1 then goto 320
270    if number<>asc(" ") then sp=0 el
se sp=1
280    if number<>34 then 300
290    if ex=1 then ex=0 else ex=1
300    if ex=0 and number>=asc("a") and
 number<=asc("z") then number=number-3
2
310    product=x*number:checksum=checks
um+product:x=x+1:if x=4 then x=1
320    next z:? ".";
330    checksum=checksum-1000*int(check
sum/1000):c(i)=checksum:x=2:next i
340    close #2:lyne=r(0):item=0
350    color 2:?:?:? "Creating BUG file
":color 1
360    count=10:total=0:if linecount<10
 then count=linecount
370    i$=str$(lyne):i$=i$+" data "
380    for i=1 to count:datum=c(10*item
+i)
390    i$=i$+str$(datum):i$=i$+",":tota
l=total+datum:next i
400    i$=i$+str$(total):print #1,i$:?
".";
410    item=item+1:linecount=linecount-
10:if linecount<1 then 430
420    lyne=r(item):goto 360
430    close #1:clearw 2:?:gotoxy 0,1
440    ? "To check BUG data against the
 checksum data found in the magazine,"
450    ? "return to the GEM desktop and
 double click your BUG file.  You may"
460    ? "then SHOW the data on your sc
reen or PRINT the data to your printer
.":?
470    ? "The line number of each data
statement coincides with the first lin
e"
480    ? "of the user program the data
statement evaluates.  Numbers within"
490    ? "each data statement represent
 consecutive lines of the user program
."
500    ? "The last number is the total.
":?
510    ? "Check the last number of each
 statement against the version in the"
520    ? "magazine.  Only when there's
a discrepancy need you check each numb
er"
530    ? "in the data statement.":?
540    ? "Take note of the lines contai
ning typos, then make corrections.  Wh
en"
550    ? "all corrections have been mad
e, rerun this program to double check.
"
560    ? "Press <RETURN>":input i$:clos
e #1:close #2:end
570    if err=62 then resume 120
580    if err=62 then resume 210
590    if err=53 then ? chr$(7);"FILE N
OT FOUND!":close:resume 50
600    ? "ERROR #";err;" at LINE ";erl:
end
```

●

ST CHECKSUM DATA.

```
 10 data  447, 129, 203, 518, 661, 160
, 942, 482, 640, 556, 4738
 110 data  25, 905, 797, 52, 79, 349,
852, 644, 9, 402, 4114
 210 data  883, 479, 621, 744, 498, 25
5, 165, 826, 410, 337, 5218
 320 data  1, 166, 578, 136, 801, 898,
 937, 271, 769, 363, 4920
 420 data  99, 155, 889, 243, 764, 168
, 192, 906, 156, 757, 4329
 520 data  251, 146, 509, 146, 916, 53
9, 541, 733, 845, 4626
```

●

ST NEWS!

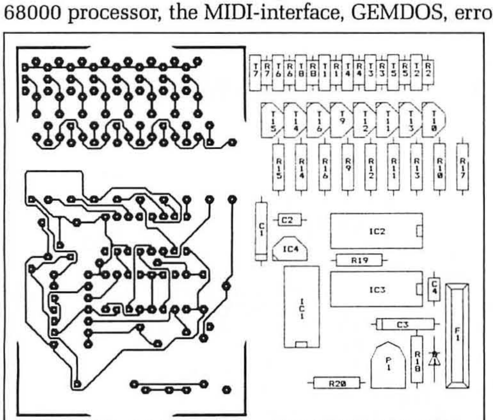
INSIDE GUIDES FOR THE ST USER

Abacus Software is offering three new books in their ST line. ***Atari ST Internals*** covers the 68000 processor, the MIDI-interface, GEMDOS, error codes, custom chips, disk controller and much more.

The complete guide to programming the ST using GEM is explained in ***Atari ST—GEM Programmer's Reference***. This 414-page book looks at GEM, including VDI, AES, GDOS and GIOS. Also examined is the development system, programming the Virtual Device Interface, using the editor, C-compiler, assembler and linker.

Atari ST—Machine Language looks at logical operations and bit manipulation, program development, the operating system and programs and the 68000 register structure.

The books retail for $19.95 each. A disk is available for each guide, giving the programs within, at $16.95 per book.



Actual printout from an Epson printer suitable for photoetching

Actual printout from an Epson printer suitable for silkscreening

Abacus has also just announced an interactive, computer-aided program designed to automate printed circuit board layouts. The user "places" the components on the screen, then specifies the connection. The ST then proceeds to automatically route the traces on-screen. At any time you can change components or locations and have the traces redrawn. Abacus says **PC Board Designer** is friendly to use and features drop-down menus. Screen dumps are produced on Epson and compatible printers. Suggested retail cost of **PC Board Designer** is $395.00. Abacus Software, P.O. Box 7211, Grand Rapids, MI 49510 — (616) 241-5510.

CIRCLE #118 ON READER SERVICE CARD

A MULTI-TASKING ENVIRONMENT

Beckemeyer Development Tools has announced their **MT C-Shell**, described as a fully multi-tasking, Unix-like environment for the ST line.

It should be noted by readers that this is not a replacement operating system for the ST, but an extension to GEMDOS. It allows for multiple ST applications to be used at the same time.

For instance, while the ST is compiling a program, it can also print out hardcopy—as you're editing, telecommunicating or whatever. While all of this is occurring, the ST is said to slow down minutely.

Beckemeyer also offers a Unix-compatible C library and several utilities. The **MT C-Shell** is expected to retail for $79.95 or slightly higher. From David Beckemeyer Development Tools, 592 Jean Street #304, Oakland, CA 94610 — (415) 658-5318.

CIRCLE #128 ON READER SERVICE CARD

C TOOLBOXES

InSoft offers several ST toolboxes with full documentation. The **Math Tool Box** includes programs covering vector arithmetic, statistical functions, curve fitting, matrix arithmetic and more. The **Searching and Sorting Tool Box** consists of several utilities, including a quicksort, file merge and string/array search. The **Graphics Tool Box** will handle curve drawing in 2-D, shapes in 3-D, object rotation and zooming. These toolboxes retail for $59.00 each.

They'll be followed by the **Graphic Work Station**, for 2-D/3-D construction and display. InSoft also offers an ST disk magazine and newsletter. Contact: InSoft, Corp., 1834 Beacon St., Suite 1, Brookline, MA 02146 — (617) 739-9012. CIRCLE #171 ON READER SERVICE CARD

SUPER 3D PLOTTER II

This program lets you display full-screen, high-resolution, 3-D images and gives you the ability to rotate and move the images at up to six times per second.

Features include hidden line removal and interactive graphic editing. The 56-page manual covers major functions: hardcopy printout, rotation control, data editor, etc. Also offered are routines to convert **Solid States** (from our issue 16) for **Super 3D Plotter**.

Retails for $39.95, from Elfin Magic, 23 Brook Place, East Islip, NY 11730.

CIRCLE #168 ON READER SERVICE CARD

SOLADISK

This ramdisk sets up an area of RAM to be used as another "disk drive." This assembly program transfers data at the astounding rate of over 10 million bytes per second, with the least memory-consuming directory of any ramdisk for the ST.

In stores $15.00; $11.00 from Solar Powered Software, 1807 N. Evergreen, Chandler, AZ 85224. CIRCLE #169 ON READER SERVICE CARD

REGENT SPELL

Regent Software's new **Regent Spell** is a 30,000-word spelling checker for the 520ST, compatible with **Regent Word** and most ST word processors. Misspelled words are highlighted, with ten suggested spellings shown. Choose a spelling or type in the correct one. The dictionary expands to 60,000 words.

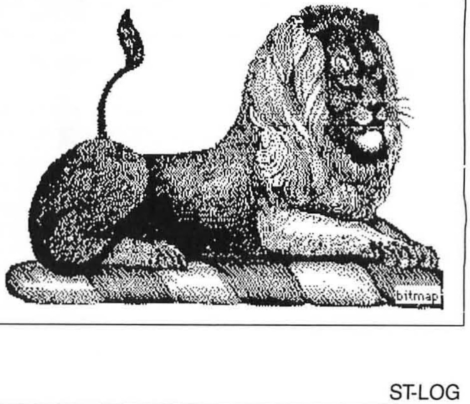
It's $49.95. Regent Software, 7131 Owensmouth, Ste. 45A, Canoga Park, CA 91303 — (818) 883-0951. CIRCLE #125 ON READER SERVICE

BITMAP COLORING BOOK

Created from design and style books issued early in the 20th century by leading typographers and engravers, Bitmap's images are suited to use as a "coloring book," or for editing or other graphic changes.

Bitmap will also be producing an architectural drawing package, an "Electronic Woodcut" set and a special font package.

Bitmap Coloring Book is $18.95. Bitmap will digitize images at $25.00 (see **End User**, issue 40). Bitmap, Inc., Box 237, Westwego, LA 70094. CIRCLE #170 ON READER SERVICE CARD



Mr. Scratch

by Clayton Walnum

"You want to what?"

"You heard me," replied Scratch, eyeing his assistant with annoyance. His tail twitched, and the barb struck the floor with a loud thwack. This schmuck was a perfect example of his current dilemma. He needed good people down here — not these muddle-brained losers, lacking in vision and ambition.

"Advertise!" Scratch continued. "That's how all the successful companies on the surface get their trade. Why should Hell be any different?"

The assistant shook his head in disgust. A pillar of flame crackled into existence behind him, and he had to leap away to avoid getting scorched. Scratch grinned.

"Think about it, sir!" pleaded the assistant. "Every time you come up with one of these ideas, it backfires on you. How about that Daniels guy? Remember that stupid fiddle contest?"

The assistant dodged another blast of fire. Scratch was losing patience. He'd warned this idiot once; he didn't want to hear any more. Sulphur and Brimstone! He still couldn't show his face in Georgia.

But the assistant wasn't taking the hint. "And then there was that fiasco with the little girl. What was her name. . . Regan?" The assistant chuckled. "Boy, that priest sure put a crimp in your pitchfork! For

NEW

FOR THE ATARI ST

NEW

* **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

Heaven's. . .uh. . .for Hades' sake, they didn't even pay you for the film rights."

There was a bright flash, a choked-off scream, then silence. Scratch glared at the smudge of ash, the sole remains of his assistant. He stroked his beard and began to write.

IMAGINE! Anything you desire. . .

The game.

Well, it looks like there's trouble brewing. Old Scratch has had a whole slew of advertisements printed. He's mailed them out to a select list of citizens, and he's snapping up souls so fast that the furnace stokers had to go on double shifts just to keep up.

You, of course, are an aware and duty-conscious community member. You've decided free enterprise should not extend to the nether realms—especially since the infamous ad has popped up in *your* mailbox. What are you going to do about it? Is that a challenge? You bet your sweet asbestos suit it is.

The first challenge.

Type in the program exactly as in Listing 1. I know some of the lines look a little weird. All text in the program has been encrypted, so that you won't learn the game's secrets as you type it in. Yeah, I sympathize. It doesn't make the typing any easier, but there really isn't a better way. Your perseverance will be rewarded. Trust me.

Once you've got it all typed, save a copy to disk, then use **ST-Check** (see page 53ST) to be sure you've made no typos. This is especially important with an adventure game because errors won't necessarily affect the game in an obvious way. You could end up with a game that's impossible to win. So check that typing!

When running the game, be sure you have no desktop accessories loaded, and that the "buf graphics" are turned off.

Playing Mr. Scratch.

As in most text adventures, you communicate with **Mr. Scratch** by two-word commands. These should be in a normal verb/noun format (i.e., *GET BOOK, GO DOOR*). There are a few exceptions. All directions should be abbreviated to a single letter (*N, S, E, W, U, D*).

There are also a few special commands you should be aware of. These are: SAVE GAME, LOAD GAME, HELP and QUIT. Use the save command to store your progress on disk. The load command will restore the last position saved. Type *HELP* any time you wish to have one of the encrypted hints translated. Finally, to end the game, type the command *QUIT*. Be sure you save your progress before quitting.

Mr. Scratch won't understand everything you type. To help you find the right commands, the program will give you short messages. The message *Don't understand that verb* or *Don't understand that noun* indicates that the verb or noun you used isn't in the program's vocabulary. When you see *You can't do that!* it means that you haven't met the conditions required for the requested action, or that the command is beyond the scope of the game.

Novice's corner.

If you've never played a text adventure before, you may find **Mr. Scratch** a bit confusing at first. You'll see the message *You can't do that!* at times when it seems completely illogical. For instance, why can't you *OPEN BAG?* It's right there in plain sight!

It's important to realize that the game will respond only to those commands it's been programmed to accept. There's no computer in the galaxy big enough to hold all the possible replies to all the possible commands (and you sure wouldn't want to *type* a program that big). Sometimes, rewording your command will yield a result. How about *GET BAG* instead?

Draw a map. That's the only way you can keep track of your location. The most common mapping technique for adventures is to represent each room (every location is a room, even if it's outside) by a small box. You then write the room's name, as well as any item found, inside the box.

Each possible exit is indicated on your map by a small line leading toward the next room. When you enter a new room, be sure to take note of all exits. It's imperative that you try each one. Otherwise, you're likely to miss something important.

To start your adventure, try each available exit and note any items found. When you can go no farther, stop and think about everything you've discovered. What should you do with the letter? Is the red pen significant in some way? How about the wallet? Is it important? When you solve a puzzle, repeat the process—moving from room to room, gather items and information until you get stuck again. Eventually, you'll find your way to the game's solution.

Before signing off, I'd like to thank our new ST man, Doug Weir, for the machine language subroutines that allow the mouse to be turned off and on. You BASIC programmers will find these routines useful in your own work, I'm sure.

Mr. Scratch hints.

To use the following hints, type the command *HELP* at any time during play. Find the question that relates to your problem, then type in the first encrypted hint beneath it. Each line is a separate hint, and some questions have several hints. After you decode the first, try to solve the puzzle on your own. If you're still stuck, then decode the next hint.

Above all, don't even *glance* at the hints until you really need help. The questions aren't encrypted, and could give away many of the game's surprises.

How do I open the jewelry box?

```
UIFSF!JT!OP!LFZ
ZPV!OFFE!B!UPPM
PQFO!UIF!UPPM!CPY
VTF!UIF!TDSFXESJWFS
VOTDSFX!UIF!IJOHFT
```

How can I ride the bus?

```
FOUFS!JU
ZPV!IBWF!UP!QBZ
UBML!UP!UIF!ESJWFS
HJWF!SJHIU!OVNCFS!PG!UPLFOT
```

How do I use the terminal?

```
FYBNJOF!JU
QVTI!UIF!CVUUPO
ZPV!OFFE!UIF!CPPL
FOUFS!UIF!DPEF!GSPN!UIF!CPPL
```

How do I get into the house?

```
HP!EPPS
IPX!EJE!ZPV!MJLF!KBJM@
USZ!LOPDLJOH
```

How do I get the jar?

```
UBML!UP!UIF!MBEZ
MPPL!JO!UIF!CBH
HJWF!IFS!UIF!DPPLJFT
```

How do I get the bicycle?

```
NBLF!UIF!EPH!IBQQZ
HJWF!IJN!TPNFUIJOH!UP!FBU
GJOE!UIF!IBNCVSHFS
JO!UIF!HBSCBHF!DBO
```

How do I go somewhere on the bike?

```
SJEF!JU
XIFO!JO!B!DFSUBJO!QMBDF
PO!EPXOJOHWJMMF!TUSFFU
```

What about the wino?

```
IF!IBT!TPNFUIJOH
HFU!UIF!CPUUMF
```

How can I keep the bottle?

```
NBLF!UIF!XJOP!VOBXBSFT
HJWF!IJN!B!TMFFQJOH!QJMM
QVU!JU!JO!UIF!CPUUMF
HJWF!UIF!CPUUMF!CBDL
```

What about the church?

```
MPPL!BU!UIF!GPOU
HFU!TPNF!XBUFS
JO!UIF!CPUUMF
```

What about the priest?

```
UBML!UP!IJN
NBLF!B!EPOBUJPO
UBML!UP!IJN!BHBJO
HJWF!IJN!UIF!CPUUMF!BOE!XBUFS
```

How can I find Scratch?

```
SFBE!UIF!MFUUFS
VTF!UIF!TUSFFUOBNF
GJOE!UIF!DPEF!JO!UIF!CPPL
FOUFS!DPEF!JOUP!UIF!UFSNJOBM
```

How can I defeat Scratch?

```
ZPV!NVTU!IBWF!XBUFS
JU!IBT!UP!CF!IPMZ!XBUFS
UIF!QSJFTU!XJMM!CMFTT!JU
IBWF!UP!HFU!SJE!PG!UIF!XJOF
```

Listing 1.
ST BASIC listing.

```
10 START:fullw 2:clearw 2:nr=24:ni=31:
nv=27:option base 1:goto INITIALIZE
30 CASE:'change from lower to upper ca
se
40 for x=1 to len(cm$):b$=mid$(cm$,x,1
)
50 if b$>="a" and b$<="z" then mid$(cm
$,x,1)=chr$(asc(b$)-32)
60 next:return
70 TRANS:'print translated text
80 gotoxy 5,7:color 2
90 for x=1 to len(a$):if mid$(a$,x,1)=
"=" then mid$(a$,x,1)="!":goto 100
```

```
95 mid$(a$,x,1)=chr$(asc(mid$(a$,x,1))
-1)
100 next:? a$:a$="":return
110 RENEW:'update screen
120 color 1:gosub DESCRIPTION:gosub VE
CTORS:gosub ITEMS:gosub INVENTORY:retu
rn
130 DESCRIPTION:'print room name
140 gotoxy 9,2:? space$(25):gotoxy 9,2
:a$=room$(room-4):gosub 90:return
150 VECTORS:'display exits
160 gotoxy 9,4:? space$(15)
170 for x=0 to 5:vector(x+1)=vecs((roo
m-4)*6-5+x):next
180 dr=0:gotoxy 9,4
190 for x=1 to 6:if vector(x)>0 then ?
 mid$(singles$,x,1);" ";:dr=1
200 next:if dr=0 then ? "None";
210 return
220 ITEMS:'display visible items
230 color 1:for x=11 to 15:gotoxy 4,x:
? space$(14):next
240 it=0:y=11:for z=1 to ni
250 if abs(iloc(z))=room then gotoxy 4
,y:a$=item$(z):gosub 90:it=1:y=y+1
260 next:if it=0 then gotoxy 4,11:? "N
othing"
270 return
280 INVENTORY:'display inventory items
290 for x=11 to 15:gotoxy 19,x:? space
$(14):next
300 i=0:y=11:for z=1 to 5
310 if inv(z)<>0 then gotoxy 19,y:a$=i
tem$(inv(z)):gosub 90:i=1:y=y+1
320 next:if i=0 then gotoxy 19,11:? "N
othing"
330 return
340 PARSER:'get command
350 if len(a$)>0 then gosub TRANS
360 on error goto 2870
370 if drgcnt>0 and room<>24 then drgc
nt=drgcnt+1:if drgcnt>10 then iloc(13)
=0:iloc(28)=-24:drgcnt=0
380 color 1:? chr$(7):gotoxy 2,8:input
 cm$:gosub CASE
390 gotoxy 4,8:? space$(31):gotoxy 5,6
:? space$(30):gotoxy 5,6:? cm$
400 gotoxy 5,7:? space$(30):gotoxy 5,7
:color 2
405 if cm$="QUIT" then gosub LIVEMOUSE
:color 1:end
410 if cm$="HELP" then a$="Uzqf!jo!uif
!fodszqufe!ijou":h=1:goto PARSER
420 if h=1 then h=0:a$=cm$:goto PARSER
430 if len(cm$)=1 then 500
440 x=instr(cm$," "):if x<3 then ? "HU
H?":goto PARSER
450 verb$=left$(cm$,3):noun$=mid$(cm$,
x+1,3)
460 v=instr(vtab$,verb$):if v=79 then
v=6
465 if v>0 then v=tr(int(v/3)+1):goto
480
470 ? "Don't understand that verb!":go
to PARSER
480 n=instr(ntab$,noun$):if n>0 then n
=int(n/3)+1:goto 500
490 if room<>14 or v<>8 then ? "Don't
understand that noun!":goto PARSER
500 turn=turn+1:if int(turn/10)<>turn/
10 then 510
505 iloc(9)=0*(iloc(9)<0)+11*(iloc(9)=
0):if room=11 then gosub ITEMS
506 if iloc(9)=-11 then bus=bus+1:if b
us>2 then bus=1
510 if room=15 then scr=scr+1:if scr=3
 then a$="IF!UISPXT!IJT!QJUDIGPSL=":go
to DEAD
520 if room<>15 then scr=0
530 color 2:gotoxy 5,7
540 if bturn>0 then bturn=bturn+1:if b
turn=5 then bturn=0:room=11:iloc(9)=0:
gosub RENEW:a$="Zpv!hfu!upttfe!pgg!uif
!cvt":goto PARSER
550 if len(cm$)=1 then goto ONELETTER
560 on v goto 650,800,970,1200,1250,14
30,1460,1510,1610,1640,1720,1740,1780,
1840,1880,1960,2050
570 ONELETTER:'single letter commands
580 v=instr(singles$,cm$):if v=0 then
? "WHAT?":goto PARSER
590 if room=24 and iloc(13)=-24 and il
oc(14)=-1 then a$="UIF!XJOP!BUUBDLT!ZP
V=":goto DEAD
610 if vector(v)=0 then ? "You can't g
o that way!":goto PARSER
620 ? "Okay":room=vector(v):gosub RENE
W:goto PARSER
650 color 2
660 if room=25 and wt=0 and iloc(14)=-
1 and n=39 then wt=1:a$="Zpv!gjmm!uif!
cpuumf":goto PARSER
670 if n=8 and op3=1 and (iloc(4)=0 or
 iloc(4)=-1) then 720
680 if n=7 and op1=1 and iloc(n)=0 and
 room=12 then 720
685 if n=2 and op=1 and iloc(n)=0 and
iloc(1)=-1 then 720
690 if n=14 and iloc(14)=0 and room=24
 then 720
695 if n=26 and iloc(n)=0 and room=24
then 720
697 if n=27 and iloc(n)=0 and iloc(5)=
-1 then 720
700 if n=29 and iloc(19)=-1 then 720
705 if (iloc(30)=room or iloc(30)=-1)
and n=23 and iloc(n)=0 then 720
707 if n>ni then goto CANT
710 if iloc(n)<-4 then ? "You can't ge
t that!":goto PARSER
715 if abs(iloc(n))<>room then ? "It's
 not here!":goto PARSER
720 if iloc(n)=-1 then ? "You already
have it!":goto PARSER
730 i=0:for x=1 to 5:if inv(x)=0 then
i=x
740 next:if i=0 then ? "You can't carr
y anymore!":goto PARSER
750 if room=27 and n=19 and iloc(18)=-
room then a$="TIF!TBX!ZPV=!KBJM!UJNF="
:goto DEAD
760 if room=28 and iloc(20)=-room then
 a$="UIF!EPH!DIFXT!ZPV!VQ=":goto DEAD
770 ? "Okay":iloc(n)=-1:inv(i)=n:color
 1
780 gosub ITEMS:gosub INVENTORY:goto P
ARSER
800 color 2
810 if n<33 or n>34 then 850
820 pay=n-32:if pay>tok then a$="Zpv!e
po(u!ibwf!fopvhi":goto PARSER
830 tok=tok-pay:if tok>0 then 835
831 for x=1 to 5:if inv(x)=8 then inv(
x)=0
832 next:iloc(8)=0:gosub ITEMS:gosub I
NVENTORY
835 if pay<bus then bturn=4:goto 540
840 a$="Plbz-!uif!cvt!mfbwft":bxit=bus
(bus):bturn=0:goto PARSER
850 if iloc(n)<>-1 then ? "You don't h
ave it!":goto PARSER
860 for x=1 to 5:if inv(x)=n then i=x
870 next:if room=27 and n=23 then b$="
(Pi=!J(mm!hfu!b!qmbuf=(":iloc(23)=0:il
oc(18)=0:inv(i)=0:goto 950
880 if room=28 and n=26 then b$="Uif!e
```

```
ph!hpccmft!ju!vq":iloc(n)=-4:iloc(24)=
-room:iloc(20)=0:inv(i)=0:goto 950
890 if n=29 and iloc(14)=-1 then drg=1
:b$="///jo!uif!cpuumf":iloc(n)=0:inv(i
)=0:goto 950
900 if room=24 and iloc(13)=-24 and dr
g=1 and n=14 then drgcnt=1:iloc(14)=0:
inv(i)=0:color 1:b$="Uif!xjop!ublft!ju
!cbdl":goto 950
910 if room=25 and n=27 then iloc(n)=-
4:inv(i)=0:b$="///jo!uif!ejti":goto 95
0
920 if room=25 and bl=0 and n=14 and w
t=1 and wn=1 then a$="Uif!qsjftu!cmftt
ft!uif!xbufs":bl=1:goto PARSER
925 cnt=0:for x=1 to ni:if abs(iloc(x)
)=room then cnt=cnt+1
926 next:if cnt=5 then ? "No more room
 here!":goto PARSER
930 inv(i)=0:iloc(n)=room:? "Okay":col
or 1
940 gosub ITEMS:gosub INVENTORY:goto P
ARSER
950 color 1:gosub ITEMS:gosub INVENTOR
Y:a$=b$:gotoxy 5,7:goto PARSER
970 if n>ni then 1180
990 if n=1 and iloc(n)=-1 and op=1 and
 iloc(2)=0 then a$="Uifsf(t!b!mfuufs!j
otjef":goto PARSER
1000 if room=7 and n=4 and op2=0 then
a$="Uif!mje!jt!mpdlfe-!ijohfe":goto PA
RSER
1010 if room=12 and n=6 and op1=1 and
iloc(7)=0 then a$="Uifsf(t!b!tdsfxesjw
fs":goto PARSER
1020 if n=4 and (iloc(n)=-1 or iloc(n)
=room) and op3=1 and iloc(8)=0 then a$
="Uifsf(sf!uplfot!jo!ju":goto PARSER
1030 if n<>9 or iloc(n)<>-room then 10
65
1040 a$="Ju(t!hpjoh!up!":on bus goto 1
050,1060
1050 a$=a$+"EPXOVOEFS":goto PARSER
1060 a$=a$+"EPXOJOHWJMMF":goto PARSER
1065 if n=8 and iloc(8)=-1 then ? "You
 have ";tok;" of them":goto PARSER
1070 if room=14 and n=11 then a$="Uifs
f(t!b!cvuupo!po!ju":goto PARSER
1080 if room=26 and n=17 then a$="CFXB
SF!PG!EPH":goto PARSER
1090 if room=24 and n=25 and iloc(26)=
0 then a$="Uifsf(t!bo!pme!ibncvshfs!jo
!ju":goto PARSER
1100 if n=5 and iloc(n)=-1 and iloc(27
)=0 then a$="Uifsf(t!b!epmmbs!jo!ju":g
oto PARSER
1110 if room=25 and n=16 then a$="If!i
bt!b!dpmmfdujpo!ejti":goto PARSER
1120 if room=25 and n=15 then a$="Uifs
f(t!xbufs!jo!ju":goto PARSER
1130 if room=24 and (n=13 or n=28) and
 iloc(14)=0 then a$="If(t!hpu!b!cpuumf
":goto PARSER
1140 if n=19 and iloc(n)=-1 then a$="U
ifsf(t!tmffqjoh!qjmmt!jo!ju":goto PARS
ER
1150 if n=14 and iloc(n)=-1 and wn=0 t
hen a$="Uifsf(t!xjof!jo!ju":goto PARSE
R
1160 if n=30 and iloc(n)=-1 and iloc(2
3)=0 then a$="Uifsf(t!dppljft!jo!ju":g
oto PARSER
1170 if n=3 and iloc(n)=-1 then a$="Ui
f!ujumf!jt;!TUSFFU!DPEFT":goto parser
1172 if room=15 and n=12 then a$="XPX=
!Ipsot!boe!fwfszuijoh=":goto PARSER
1180 ? "You see nothing special":goto
PARSER
1200 if n=1 and iloc(n)=-1 and op=0 th
en a$="Zpv!sjq!ju!pqfo":op=1:goto PARS
ER
1210 if room=12 and n=6 and op1=0 then
 ? "Okay":op1=1:goto PARSER
1220 if n=4 and (iloc(n)=-1 or iloc(n)
=room) and op2=1 then a$="Pqfojoh///":
op3=1:goto PARSER
1230 if n=4 and (iloc(n)=-1 or iloc(n)
=room) then a$="Zpv!epo(u!ibwf!uif!lfz
":goto PARSER
1235 goto CANT
1250 if n<>2 or iloc(n)<>-1 then 1400
1260 clearw 2:gotoxy 17,0:a$="777!Gjsf
gbmm!Dpvsu":gosub 90
1270 gotoxy 17,1:a$="Epxovoefs-!OK!435
76":gosub 90
1280 gotoxy 17,2:a$="Kvof!23-!2:97":go
sub 90
1290 a$="Efbs!ofjhicps-":gosub 90:?:a$
="JNBHJOF=!Bozuijoh!zpv!eftjsf!dbo!cf"
:gosub 90
1300 a$="zpvst!xjuijo!35!ipvst=!Uibu(t
!sjhiu=":gosub 90
1310 a$="POMZ!35!IPVST=!Npofz-!Mpwf-!b
!ofx":gosub 90
1320 a$="dbsffs/!!Bozuijoh!bu!bmm/!Dbo
!zpv":gosub 90
1330 a$="usvtu!vt@!Ifz-!xf(wf!cffo!nbl
joh":gosub 90
1340 a$="hsfbu!efbmt!gps!dfouvsjft/!Up
!ublf":gosub 90
1350 a$="bewboubhf!pg!uijt!gboubtujd!p
ggfs-":gosub 90
1360 a$="kvtu!tjho!po!uif!epuufe!mjof!
)SFE!jol":gosub 90
1370 a$="pomz-!qmfbtf*/":gosub 90:?:a$
="Ns/!Tdsbudi":gosub 90
1380 color 1:gotoxy 10,17:? "Press RET
URN";:input a$:gosub SCREEN:gosub RENE
W
1390 goto PARSER
1400 if room=26 and n=17 then a$="CFXB
SF!PG!EPH":goto PARSER
1410 if n=3 and iloc(n)=-1 then a$="Vt
f;!GJOE!tusffu!obnf":goto PARSER
1430 if n=2 and iloc(n)=-1 and iloc(31
)=-1 then a$="Zpv(wf!usbefe!bxbz!zpvs!
tpvm=":goto DEAD
1440 goto CANT
1460 if (iloc(4)<>-1 and iloc(4)<>room
) or iloc(7)<>-1 or n<>32 or op2=1 the
n goto CANT
1470 a$="///Vtjoh!uif!tdsfxesjwfs":op2
=1:goto PARSER
1510 if room=11 and n=9 and iloc(9)=-r
oom then room=13:bturn=1:gosub RENEW:a
$="Plbz":goto PARSER
1520 if room<>14 or but=0 then 1580
1530 but=0:if n=36 then room=15:goto 1
570
1540 if n=37 then room=16:goto 1570
1550 if n=38 then room=9:goto 1570
1555 if n=45 then room=14:goto 1570
1560 a$="Opuijoh!ibqqfot":goto PARSER
1570 gosub RENEW:a$="Qpppggggg=":goto
PARSER
1580 if room=26 and n=22 then a$="ZPV(
SF!UISPXO!JO!KBJM=":goto DEAD
1590 goto CANT
1610 if room=13 and n=9 then room=bxit
:? "Okay":gosub RENEW:goto PARSER
1620 goto CANT
1640 if n<>10 or room<>13 then 1680
1650 a$="(Gbsf!jt!":on bus goto 1660,1
670
```

```
1660 a$=a$+"2!uplfo(":goto PARSER
1670 a$=a$+"3!uplfot(":goto PARSER
1680 if room=24 and n=13 and iloc(13)=
-24 then a$="If!cvsqt!boe!tnjmft":goto
 PARSER
1690 if room=27 and n=18 and iloc(n)=-
27 then a$="(Ipx!bcpvu!b!tobdl@(":goto
 PARSER
1695 if room=25 and n=16 and iloc(27)<
>-4 then a$="(Dibsjuz!jt!hppe!gps!uif!
tpvm(":goto PARSER
1697 if room=25 and n=16 then a$="(Ipx
!nbz!J!ifmq!zpv@(":goto PARSER
1700 goto CANT
1720 if room=14 and n=35 then a$="B!wp
jdf!tbzt;!(Foufs!Dpef(":but=1:goto PAR
SER
1730 goto CANT
1740 if room=26 and n=22 and iloc(18)=
0 then room=27:gosub RENEW:a$="Epps!jt
!pqfo/!Zpv!tufq!jotjef":gotoxy 5,7:got
o PARSER
1750 if room=26 and n=22 then room=27:
gosub RENEW:a$="B!mbez!mfut!zpv!jo":go
toxy 5,7:goto PARSER
1760 goto CANT
1780 if n<>13 or iloc(14)<>-1 or wn=1
then 1810
1790 if room=24 and iloc(13)=-24 then
a$="UIF!XJOP!BUUBDLT!ZPV=":goto DEAD
1800 wn=1:wt=0:a$="Zpv!qpvs!pvu!uif!xj
of":goto PARSER
1810 if room=15 and iloc(14)=-1 and bl
=1 and n=39 then a$="IF!NFMUT!BXBZ!UP!
OPUIJOH":goto WINNER
1820 if room=15 and iloc(14)=-1 and n=
39 then a$="IF!UISPXT!B!QJUDIGPSL=":go
to DEAD
1830 goto CANT
1840 if room<16 or room>24 and n=21 an
d iloc(n)=-1 then a$="Xiffffffff=":got
o PARSER
1850 if room>15 and room<25 and n=21 a
nd iloc(n)=-1 then room=9:gosub RENEW:
a$="Pgg!zpv!hp=":goto PARSER
1860 goto CANT
1880 if n<>40 then goto CANT
1890 ? "Saving...":open "O",#1,"SCRATC
H.DAT"
1900 for x=1 to ni:write #1,iloc(x):ne
xt
1910 for x=1 to 5:write #1,inv(x):next
1920 write #1,room,op,op1,op2,op3,turn
,bturn,bus,tok,bxit,but
1930 write #1,drg,drgcnt,wat,bl,wn,scr
:goto 2020
1960 if n<>40 then goto CANT
1970 on error goto 2030:? "Loading..."
:open "I",#1,"SCRATCH.DAT"
1980 for x=1 to ni:input #1,iloc(x):ne
xt
1990 for x=1 to 5:input #1,inv(x):next
2000 input #1,room,op,op1,op2,op3,turn
,bturn,bus,tok,bxit,but
2010 input #1,drg,drgcnt,wat,bl,wn,scr
:gosub RENEW
2020 close:gotoxy 5,7:color 2:? "Done!
      ":goto PARSER
2030 a$="OP!HBNF!TBWFE=":resume PARSER
2050 if n<41 or n>44 or iloc(3)<>-1 th
en goto CANT
2060 a$="Uif!dpef!jt;!":if n=41 then a
$=a$+"GBM"
2070 if n=42 then a$=a$+"BJO"
2080 if n=43 then a$=a$+"WJM"
2085 if n=44 then a$=a$+"UPO"
2090 goto PARSER
```

```
2100 CANT:? "You can't do that!":goto
PARSER
2110 DEAD:'Player blew it!
2120 clearw 2:gotoxy 18-len(a$)/2,5:go
sub 90
2130 gotoxy 6,8:? "This adventure is o
ver!"
2140 gotoxy 6,12:? "You lasted ";turn;
" turns"
2150 color 1:gotoxy 8,17:? "Play again
";:input a$
2160 if left$(a$,1)="Y" or left$(a$,1)
="y" then gosub LIVEMOUSE:goto 2240
2170 if left$(a$,1)="N" or left$(a$,1)
="n" then gosub LIVEMOUSE:end
2180 goto 2150
2190 WINNER:'Mission completed!
2200 clearw 2:gotoxy 18-len(a$)/2,5:go
sub 90
2210 gotoxy 12,8:? "YOU WIN!!":gotoxy
6,12:? "It took you ";turn;" turns":go
to 2150
2220 INITIALIZE:'set up game
2230 if peek(systab)=1 then gotoxy 10,
10:? "You must have a color monitor!":
for x=1 to 5000:next:end
2235 dim vector(6),vecs(nr*6),item$(ni
),inv(5),room$(nr),iloc(ni+12),tr(nv)
2236 dim v%(4),ml%(50):av=varptr(v%(1)
):strt=varptr(ml%(1))
2237 dim bus(2): bus(1)=14:bus(2)=16
2240 poke contrl,32:poke contrl+2,0:po
ke contrl+6,1:poke intin,2:vdisys(1)
2250 restore 2360:fullw 2:clearw 2:if
peek(systab)=2 then linef 302,0,302,16
8
2260 color 2,1:fill 150,80
2270 ef=16:gosub TEXTEFFECT:gotoxy 5,5
:? "M I S T E R  S C R A T C H"
2280 ef=4:gosub TEXTEFFECT:color 3:got
oxy 9,7:? " A Devilish Tale"
2290 ef=1:gosub TEXTEFFECT:color 4:if
peek(systab)=2 then color 2
2300 gotoxy 15,9:? "by"
2310 gotoxy 10,10:? "Clayton Walnum":e
f=0:gosub TEXTEFFECT
2340 for x=1 to 5:inv(x)=0:next:inv(1)
=5:inv(2)=31
2350 for x=1 to nr*6:read vecs(x):next
2360 data 6,7,8,0,0,12,0,5,0,0,0,0,5,0
,0,0,0,0,9,10,0,5,0,0
2370 data 0,8,0,0,0,0,8,11,0,0,0,0,10,
0,0,0,0,0,0,0,0,0,5,0
2380 data 0,0,0,0,0,0,14,14,14,14,14,1
4,0,0,0,0,0,0,22,17,22,16,0,0
2390 data 16,18,24,0,0,0,17,19,25,0,0,
0,18,20,26,0,0,0,19,21,0,0,0,0
2400 data 20,22,22,21,0,0,23,21,21,22,
0,0,22,23,23,24,0,0,0,0,23,17,0,0
2410 data 0,0,0,18,0,0,0,0,0,19,0,0,0,
0,28,26,0,0,0,0,0,27,0,0
2420 for x=1 to ni:read item$(x),iloc(
x):next
2430 data Fowfmpqf,5,Mfuufs,0,Cppl,6,K
fxfmsz!cpy,7,Xbmmfu,-1,Uppm!lju,-12
2440 data Tdsfxesjwfs,0,Uplfot,0,Cvt,-
11,Esjwfs,-13,Ufsnjobm,-14
2450 data Ns/Tdsbudi,-15,Xjop,-24,Cpuu
mf,0,Gpou,-25,Qsjftu,-25
2460 data Tjho,-26,Mbez,-27,Kbs!pg!qjm
mt,27,Tobsmjoh!eph,-28,Cjdzdmf,28
2470 data Gspou!epps,-26,Dppljft,0,Ibq
qz!eph,0,Hbscbhf!dbo,-24,Ibncvshfs,0
2480 data Epmmbs,0,Tmffqjoh!xjop,0,Qjm
m,0,Tipqqjoh!cbh,5,Sfe!qfo,-1
2490 vtab$="GETTAKDROGIVLOOEXAOPEREASI
GUNSREMGO ENTEXILEATALSPEPRE"
```

// Mr. Scratch *continued*

```
2500 vtab$=vtab$+"PUSKNOPOUEMPRIDSAVLO
AFINPAY"
2510 for x=1 to nv:read tr(x):next
2520 data 1,1,2,2,3,3,4,5,6,7,7,8,8,9,
9,10,10,11,11,12,13,13,14,15,16,17,18
2530 for x=1 to 11:read room$(x):next
2540 for x=12 to 19:room$(x)="Po!Epxoj
ohwjmmf!Tu/":next
2550 for x=20 to nr:read room$(x):next
2560 data Jo!zpvs!mjwjoh!sppn,Jo!zpvs!
efo,Jo!zpvs!cfesppn,Po!Nbjo!Tusffu
2570 data Po!Nbjo!Tusffu,Po!Nbjo!Tusff
u,Bu!uif!cvt!tupq,Jo!uif!dfmmbs
2580 data Po!b!cvt,Po!Ipuufsupo!Tusffu
,Jo!b!sfe!pggjdf
2590 data Jo!bo!bmmfz,Jo!b!divsdi,Jo!t
pnfpof(t!gspou!zbse
2600 data Jo!b!mjwjoh!sppn,Jo!tpnfpof(
t!cbdl!zbse
2610 ntab$="ENVLETBOOBOXWALKITSCRTOKBU
SDRITERMR.WINBOTFONPRISIGLADJARDOGBIC"
2620 ntab$=ntab$+"DOOCOOHAPGARHAMDOLSL
EPILBAGPEN"
2630 ntab$=ntab$+"HINONETWOBUTFALVILAI
NWATGAMFIRMAIDOWHOTTON"
2640 singles$="NSEWUD"
2645 for i=0 to 96 step 2:read c:poke
strt+i,c:next
2646 data &h3f3c,&h22,&h4e4e,&h548f,&h
2a6f,&h6,&h2a55,&h2040,&h41e8,&h10
2647 data &h2a90,&h2f3c,0,&h34,&h2f3c,
0,&h36,&h3f3c,0,&h3f3c,0,&h4e4e
2648 data &hdffc,0,&hc,&h4e75,&h4e75,0
,0,0,0,&h2a6f,&h6,&h2a55,&h2f15
2649 data &h2f3c,0,&h5e,&h3f3c,&h1,&h3
f3c,0,&h4e4e,&hdffc,0,&hc,&h4e75,&h3,0
2650 room=5:op=0:op1=0:op2=0:op3=0:tur
n=0:bturn=0:bus=1:tok=4:bxit=11:but=0
2660 drg=0:drgcnt=0:wat=0:bl=0:wn=0:sc
r=0
2680 gosub SCREEN:gosub DEADMOUSE:gosu
b RENEW:goto PARSER
2690 SCREEN:'draw display
2700 poke contrl,32:poke contrl+2,0:po
ke contrl+6,1:poke intin,2:vdisys(1)
2705 restore 2730:clearw 2:color 2,4,2
2710 read a,b,c,d:if a=-1 then 2790
2720 linef a,b,c,d:goto 2710
2730 data 0,0,303,0,0,0,0,166,303,0,30
3,166,0,166,303,166
2740 data 92,0,92,10,218,0,218,10,10,1
0,293,10
2750 data 10,10,10,156,293,10,293,156,
10,156,293,156
2760 data 15,14,288,14,15,15,15,151,28
8,15,288,151,15,151,288,151
2770 data 15,32,288,32,15,50,288,50,15
,86,288,86,151,86,151,151
2780 data -1,-1,-1,-1
2790 fill 150,161:color 1,8,6,1,1:fill
 150,5:fill 150,12
2795 if peek(systab)=2 then color 2
2800 gotoxy 11,0:? "MISTER SCRATCH":co
lor 1
2810 gotoxy 2,2:? "PLACE:":gotoxy 2,4:
? "EXITS:"
2820 gotoxy 2,10:? "YOU SEE:":gotoxy 1
7,10:? "YOU HAVE:"
2830 return
2840 TEXTEFFECT:'set effects for text
2850 poke contrl,106:poke contrl+2,0:p
oke contrl+6,1
2860 poke intin,ef:vdisys(1):return
2870 gotoxy 5,7:? "ERROR ";err;" AT LI
NE ";erl:resume PARSER
2880 DEADMOUSE:'get rid of the critter
2890 call strt(av):return
2900 LIVEMOUSE:'rodent reincarnation
2910 strt1=strt+62:call strt1(av):retu
rn
```

●

ST CHECKSUM DATA.

(see page 53ST)

```
10 data 923,456,515,720,32,373,655,181
,502,227,4584
110 data 148,119,332,850,790,101,582,3
45,924,54,4245
210 data 335,436,851,361,166,136,353,5
53,839,914,4944
310 data 484,198,343,121,14,739,561,85
5,36,605,3956
405 data 613,429,661,964,726,255,305,1
9,402,257,4631
490 data 748,166,257,695,868,326,745,4
91,706,786,5788
570 data 142,562,110,872,258,344,132,9
30,576,674,4600
690 data 671,472,674,223,84,432,251,39
5,333,216,3751
740 data 194,970,567,257,115,333,228,8
56,291,4,3815
832 data 336,737,378,133,522,411,664,4
54,63,848,4546
920 data 996,690,223,329,107,354,515,6
,159,403,3782
1020 data 24,722,793,316,784,664,881,3
13,844,449,5790
1110 data 50,420,125,633,170,465,143,6
01,58,265,2930
1210 data 216,917,140,799,214,934,640,
425,723,778,5786
1310 data 256,35,180,820,753,717,492,2
79,75,317,3924
1410 data 469,809,796,487,2,589,30,51,
236,139,3608
1555 data 244,637,306,392,804,421,800,
918,129,205,4856
1670 data 522,281,989,242,201,801,987,
804,832,196,5855
1760 data 807,312,219,848,793,146,807,
108,570,810,5420
1880 data 332,527,91,959,73,980,333,90
8,85,981,5269
2000 data 76,632,28,919,26,85,178,239,
254,68,2505
2100 data 120,161,503,760,636,617,931,
171,568,191,4658
2200 data 504,195,658,106,322,243,120,
627,432,913,4120
2270 data 147,296,505,788,322,204,899,
729,698,449,5037
2390 data 624,913,996,702,497,229,893,
871,458,664,6847
2490 data 540,212,420,332,626,818,894,
983,249,193,5267
2590 data 369,468,844,442,401,810,478,
741,425,411,5389
2649 data 309,674,278,928,329,638,685,
953,252,624,5670
2740 data 584,231,550,358,92,325,288,4
86,884,864,4662
2830 data 466,901,645,709,666,46,392,8
79,423,5127
```

●

C-MANSHIP

Part 3.

by Clayton Walnum

I hope you've been keeping up with your studying, because this month we're going to get down to some serious business. Looping structures are on our agenda, as well as a bit more about functions. And, just so we end up with something practical, the program I've chosen incorporates a function that should prove useful in the future—a sort routine.

First, I want to tie up some loose ends from last month. You may have been wondering how you can input strings of more than one word. The scanf() function is pretty useless for this purpose, since, as soon as you try to put a space between characters, scanf() grabs whatever you typed and assigns it to the first argument on its list.

We need a function that will ignore white space characters, one that will accept every character we enter until we tell it we're done. Of course, there is just such a beast.

The gets() function allows the input of strings containing white space characters. It terminates only when it sees a newline. The format for gets() is: *gets(str)*.

As you see, gets() requires one argument (in this case, *str*), the address where the string is to be stored. This will usually be a previously declared character array, so supplying the function with the array name passes the address (remember, an array name holds the address of the first byte of the array).

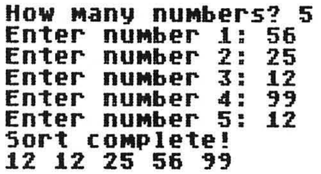
Why haven't we been using this neat little trick all along? Think about the RETURN key on the ST. What does it do for us? It provides a return character, right? And what does gets() need to terminate input? All of you mumbling "newline" get a gold star for the day. The only way that I've found to get a newline character out of the ST keyboard is with a CTRL-J. Kind of a clumsy way to end input, don't you agree?

Later on, we'll design our own input routine, so we won't be at the mercy of scanf() or gets(). But first, we need to take a look at a couple of new ideas.

Onward.

It's typing time again. Type in Listing 1 and compile it. If you have trouble, see the sidebar accompanying this article.

When you run the program, you'll be asked how many numbers you wish to sort. Enter a number between 1 and 10, then press the SPACE BAR to terminate your input. You'll be asked to enter each of the numbers. When you're done, the numbers will be sorted in ascending order and printed out. For those of you who don't have your compilers yet, a program run looks something like this:



```
How many numbers? 5
Enter number 1: 56
Enter number 2: 25
Enter number 3: 12
Enter number 4: 99
Enter number 5: 12
Sort complete!
12 12 25 56 99
```

Digging in.

Now let's take a good look at the program's innards. Since this one's much longer than any of the others we've done, you might *want* to number each line in your listing so you can follow the explanation more easily. I don't include blank lines when numbering; skip over them.

Line 1 instructs the compiler to add the contents of the stdio.h file to our program.

Line 2 defines the symbolic name *MAX* as 10. This is the maximum number of values to sort. Take a quick look at the listing. MAX is referenced in three places. If we didn't use the define statement, we'd have to substitute the number *10* for each occurrence of MAX. When we wanted a different maximum, we'd have a lot of changes to do. The #define allows a modification by simply changing the value assigned to MAX at the start of the program. See how handy this is? Imagine how much time it would save you if you were working on a thousand-line program.

Line 3 is a function name.

Line 4 marks the beginning of the function.

Line 5 declares the variable *num* as type integer.

Line 6 declares *val* as an array of type integer. Because we used the symbolic name *MAX* to dimension its size, this array will contain 10 elements, 0 through 9.

Line 7 declares the variable *ch* as type character.

Line 8 gives us something new to discuss. Here we're calling the function how_many(), which starts at Line 14, and assigning the value it returns to the variable *num*. This will be the number of items we want to sort (not to be confused with MAX, which is the *maximum* items). Notice that this function call has the same format as another that we've used quite frequently—ch = getchar(). Function calls work exactly the same, whether you're calling a library routine like getchar() or a function of your own.

Line 9 calls another of our functions, get_nums(). Since this function doesn't return a value, we aren't assigning its return to a variable. We simply call it by name, just like printf().

We do, however, have to pass arguments *to* the function—num (the number of values we wish to sort) and val (the array address where we're going to store the values).

Line 10 calls the function that does the sort. It doesn't return a value either, but still must be passed the same arguments as get_nums().

Line 11 calls the function that prints the sorted numbers to the screen. It requires the same arguments as the two previous functions.

Line 12 waits for you to press the BACKSPACE key. This statement probably looks pretty alien to you. I'm going to ask you to take it on faith for now. We'll talk about "while" loops later on in this article.

Line 13 marks the end of the function.

The Golden Moment.

We've now stumbled upon the perfect time to discuss structured programming techniques.

Our function *main()* is constructed so that anyone can easily see what's going on. Each function call performs a logical step in the sequence of actions that must be completed to utilize the sort.

This type of construction matches the way people think. When you're going to make a lunch of beans and hot dogs, you don't consciously dwell over all the details in each step. Your thoughts would run like this: "First heat the beans, then boil the hot dogs and put them in the buns."

But you have to remember details: what about taking the pans out of the drawer and placing them on the stove? Don't forget, you've got to open the can before you can get to the beans. And where did the hot dogs come from? Did you open the refrigerator? Who turned on the stove?

We don't worry about these minor details, because, if we did, we'd get so confused we'd starve. A programmer should think in this same structured way. Projects that seem impossible when you're mired in details become a snap when viewed from a more general viewpoint. It's the old bit about the forest and the trees.

It's this form of thinking that's the essence of structured programming. To get our sort routine working, all we have to do is find out how many items there *will be, get the items,* sort them, then print them out. At this point, we're not concerned about *how* we're going to do each of these steps. One thing at a time, slow and easy.

When we have the general logic worked out, *then* we can get into the details, taking each step and writing a function to accomplish it. In large programs, this process becomes even more important. Using structured techniques will make your job much easier and will result in very readable code.

Back to the program.

Line 14 is a function name. This is the function called from Line 8.

Line 15 marks the start of the function.

Line 16 declares the variable *n* as type integer.

Line 17 sets *n* equal to the value of MAX+1, or, in this case, *11*.

Line 18 is the start of a "while" loop.

This type of loop will repeatedly perform a statement or series of statements, as long as the expression within the parentheses is true. Here are some other examples:

```
while (x = 1)
while (z > 2 && ch != 'e')
```

The second line is read: while z is greater than 2 and *ch* doesn't equal the letter *e*. C uses some unusual character combinations for operators. The double ampersand is the equivalent to BASIC's *AND*. The *!=* is the symbol for "not equal to." It's the opposite of another operator we learned a while back, = =. Remember the difference between = = and =?

We're using a while loop here to insure the input of a value no larger than MAX. Looking back, Line 17 initializes the variable we're using in the conditional expression to a value greater than MAX. If we didn't do this, we might not get a chance to enter our number. Whatever was in *n* would be used to evaluate the conditional expression.

If it was less than MAX, the loop would be skipped and whatever value *n* happened to contain would be passed to the program. If you don't initialize your variables, they'll contain whatever value happened to be in the address they were assigned.

Line 19 marks the beginning of the statements within the while loop. Whenever a loop will contain more than one statement, the start and end are marked with the left and right brace, just like a function. The braces are not necessary if a loop contains only one statement. Here's an example of a single statement while loop:

```
while (x < 5)
    x = x + 1;
```

Line 20 prints a prompt.

Line 21 accepts a number from the keyboard and assigns it to *n*.

Line 22 prints a blank line.

Line 23 marks the end of the loop. At this point, the value of *n* is checked, and, if it's greater than MAX, the loop repeats. This will continue until the user enters a number less than MAX.

Notice the indenting of the statements that make up the loop. This isn't required, but makes your programs much more readable, by clearly delineating the body of the loop.

Line 24 introduces you to the "return" statement. Whenever a return is encountered, control is passed back to the calling function, along with the value in parentheses. The return may be anywhere within the function. If you don't *want* to pass a value, delete the parentheses. In this case, we're sending the value *n* back to main(), where it will be stored in the variable *num*.

The variable *n* in how__many() is a local variable. It's created when the function is called and destroyed when control is passed back to the calling function. It has no relationship with other variables in the program (except maybe *num*, which will get only its value). You could even have another *n*, without conflict, elsewhere in your program.

Arguments in C are passed "by value" rather than "by reference." This means only the value contained in the variable is passed, not its address. The original values are safe from change. If you want access to a variable that's been passed to a function, you must pass the address with a "pointer." We'll get into pointers a little later on.

Line 25 marks the end of the function.

Line 26 is a function name. This function is called by Line 9. Notice something a little different here? There're two variables enclosed in the parentheses, which means two arguments are being passed from the calling function. The argument's values will be stored in *n* and *v*, and are passed between the functions in the same order in which they appear in the function call. That is, *n* receives the value of *num*, and *v* receives the value of *val*.

Line 27 tells get__nums() how it should interpret the data in *n*, an integer. All arguments within the function name's parentheses must be defined, and you must do so *before* the beginning brace.

Line 28 tells the *function* that *v* is an integer array. We're not dimensioning the size of *v*, since it's really the same array we dimensioned in Line 6 (val[]). How can that be? Aren't arguments in C are passed by value, not address? So how can *v[]* be the same array as *val[]*? Why am I asking all these silly questions?

I'll tell you why. Because I'll bet you forgot that an array name *is* an address. The contents of *val* are being passed as I described previously, but its value is the address of the array's first byte. What does this mean to us? It means that we're very definitely going to be monkeying with the contents of the original array. It's not safely protected from our clumsy fingers like *num* is.

Line 29 marks the start of the function.

Line 30 declares some local variables. These variables exist only in the function. They're forgotten the second we exit.

Line 31 gives you a look at a new looping technique. The "for" loop in C is very similar to the "FOR. . .NEXT" loop in BASIC. Its syntax is the word *for* followed by three expressions, within parentheses, which define the limits of the loop. The three expressions are separated by semicolons.

The first expression initializes the loop variable. Here, we're setting X to 0. The second expression is the condition that controls the loop. As long as the condition yields a true result, the loop will continue executing. The third expression is the loop's step value or reinitialization. Line 31 in BASIC would look like this:

```
FOR X=0 TO N-1 STEP 1
```

Of course, in BASIC we don't need the *STEP 1*, since it's assumed. I just included it for purposes of clarity.

What do you think of that + +x in Line 31? Got any ideas? This is essentially the same as BASIC'S X=X+1. As a matter of fact, you can use the latter construction in C, as well. The + + is an increment operator. There is also a decrement operator, --. These operators may be placed before or after the variable; however, there's a subtle difference. The expression + +x increments x before the value is used. The expression x+ + increments x after the value is used. For example, let's say that x starts with a value of 1. Then, z = + +x will yield a result of 2, whereas z = x+ + yields a result of 1.

Line 32 marks the start of the loop.

Line 33 asks for the input of a number. The prompt uses the value of x to tell us the number of the value we're entering.

Line 34 gets the number and stores it in the variable *num*. Note that this variable has nothing whatever to do with the variable *num* declared in main().

Line 35 places the number into the array's next element. In C, arrays are indexed as in BASIC. In our first pass through the loop, x has a value of 0. Therefore, the first element of the array (in the context of our function, the first element is *v[0]*, but this is really our original array, *val[0]*) gets the first number input. As x gets incremented, each consecutive element of the array is filled with its appropriate value.

Line 36 moves the cursor to the next line.

Line 37 marks the end of the loop.

At this point, x is incremented, and the control statement is evaluated. If the result is true, then another iteration of the loop is performed. This continues until the loop's condition evaluates to false.

Line 38 passes control back to main(). There are no parentheses in the return statement because we aren't sending a value back.

Line 39 marks the end of the function.

Line 40 is a function name. This function is called from Line 10. The same arguments are being passed as in the previous function.

Line 41 defines the first argument as integer.

Line 42 defines the second argument as an integer array.

Line 43 marks the beginning of the function.

Line 44 defines some variables of type integer.

Line 45 initializes the variable used to evaluate the conditional expression in the while loop. This makes sure we enter the loop properly.

Line 46 starts the while loop.

Another break in the proceedings.

Before we get too far into this function, I should give you a little background on the sort.

We're going to use a "bubble" sort, one of the simplest (and slowest). It works by comparing two values and switching them if they're in the wrong order. The next two values are then compared and, if necessary, switched. This continues until the last value has been compared. Then, if there were any switches, the loop is repeated. Once the process finishes without a switch, the sort is complete.

The sort gets its name by the way the highest values "bubble" up to the top.

Back to it.

Line 47 marks the beginning of the loop.

Line 48 turns off the switch flag. If this variable retains the value of *0* through the loop that follows, then the sort is complete.

Line 49 sets up a "for" loop that will move through the array, element by element.

Line 50 should be strangely familiar. This is C's version of the IF. . .THEN statement. Its construction is very similar to its BASIC counterpart. There are two differences.

First, the expression that follows the *if* is always within parentheses. Second, don't include the word *then*. The "if" statement body follows the same rules as loops do. If you have more than one statement, the entire block must be enclosed in braces. A single statement may be placed after the if statement with no braces.

Our if statement compares an element of the array with the next element up. If the first is larger than the second, the statements contained in the braces are executed (this is the switch). If they're already in the proper order, the switching is skipped. The next iteration of the for loop is then initiated.

Line 52 is the first step of the switch. The value in v[x] is placed in "temp."

Line 53 places array element v[x+1] into v[x].

Line 54 places temp (originally v[x]) into v[x+1], and the switch is complete.

Line 55 sets the switch flag to its true condition, so the loop will be performed again.

Line 56 marks the end of the if statement.

Line 57 marks the end of the while loop.

Line 58 returns control to main().

Line 59 marks the end of the function.

Line 60 is a function name.

Line 61 declares the first argument.

Line 62 declares the second argument.

Line 63 marks the beginning of the function.

Line 64 declares a variable.

Line 65 prints a message.

Line 66 initiates a loop to print the sorted array values.

Line 67 prints the array values using the loop variable as an index.

Line 68 prints a blank line.

Line 69 returns control to main().

Line 70 marks the end of the function.

Take a breath.

Boy, we covered a whole hunk of material this time around. If you're still with me, pat yourself on the back. You've learned most of the information you need to write usable C programs. Next month, we'll get a few new tidbits and have some fun.

(Listing starts on next page)

Listing 1.
C listing.

```
#include <stdio.h>
#define MAX 10

main()
{
    int num;
    int val[MAX];
    char ch;

    num = how_many();
    get_nums(num, val);
    sort(num, val);
    output(num, val);
    while ((ch = getchar()) != '\b') ;
}

how_many()
{
    int n;

    n = MAX +1;
    while (n > MAX)
    {
        printf("How many numbers? ");
        scanf("%d", &n);
        printf("\n\n");
    }
    return(n);
}

get_nums(n, v)
int n;
int v[];
{
    int x, num;

    for (x = 0; x < n; ++x)
    {
        printf("Enter number %d: ", x+1);
        scanf("%d", &num);
        v[x] = num;
        printf("\n");
    }
    return;
}

sort(n,v)
int n;
int v[];
{
    int swtch, x, temp;

    swtch = 1;
    while (swtch == 1)
   {
        swtch = 0;
        for (x = 0; x < n - 1; ++x)
            if ( v[x] > v[x+1] )
            {
                temp = v[x];
                v[x] = v[x+1];
                v[x+1] = temp;
                swtch = 1;
            }
    }
    return;
}

output(n, v)
int n;
int v[];
{
    int x;

    printf("Sort complete!\n\n");
    for (x = 0; x <= n - 1; ++x)
        printf( "%d ", v[x] );
    printf("\n\n");
    return;
}
```

All the program listings in **C-manship** were written using the ST Developers Kit from Atari. Many of you who've recently received this package may be a little confused as to how to compile and run the programs (I know I was). For those nodding their heads in agreement, I've put together this quickie tutorial.

The first thing you must do is create the proper batch files for both the compiler and linker. LOAD your text editor and type the following exactly as it appears here:

```
cp68 %1.c %1.i
c068 %1.i %1.1 %1.2 %1.3 -f
rm %1.i
c168 %1.1 %1.2 %1.s
rm %1.1
rm %1.2
as68 -f -l -u %1.s
rm %1.s
wait.prg
```

When you're sure you've typed it correctly, SAVE it to your compiler disk under the name CC.BAT.

Now clear the previous text from memory and type in this batch file:

```
link68 [u] %1.68k=gemstart,%1,gemlib,libf,osbind
relmod %1 %1.tos
rm %1.68K
wait
```

Check your typing well, then SAVE it to your linker disk under the name LINK.BAT.

Now you're ready to compile any of the listings from **C-manship**. We'll use Listing 1 from this installment as an example.

Single-drive compilation.

(1) Use your text editor to type in Listing 1, then SAVE a copy under the name LIST1.C to both your compiler disk and a backup disk.

(2) Place the compiler disk in your drive and double click the drive A icon.

(3) Double click the BATCH.TTP program, and enter *CC LIST1* into the parameter window, concluding the entry by pressing RETURN.

(4) After the compiler has finished, there should be a file named LIST1.0 on your compiler disk. Copy this file to your linker disk.

(5) Place the linker disk in your drive and double click the drive A icon.

(6) Double click the file BATCH.TTP, and enter *LINK LIST1* into the parameter window.

(7) When the linker has finished, the file LIST1.TOS should be on the disk. This is the executable version of the program. To RUN it, simply give it a double click.

Two-drive compilation.

(1) Use your text editor to type in Listing 1, then SAVE it to disk under the name LIST1.C.

(2) Place your compiler disk in drive A and your source disk (the one you saved the program to) in drive B.

(3) Double click the drive A icon.

(4) Double click the BATCH.TTP program, and enter *CC B:LIST1* into the parameter window, concluding the entry by pressing RETURN.

(5) When the compiler has finished, replace the disk in drive A with your linker disk.

(6) Double click the drive A icon.

(7) Double click the BATCH.TTP program and enter *LINK B:LIST1* into the parameter window.

(8) When the linking is complete, your source disk will contain the file LIST1.TOS. This is the executable version of the program. RUN it by giving it a double click.

The above instructions will work with all the C program listings presented thus far in **C-manship**. Only the filenames you use must be changed.

GEMSYS()

A tutorial on the ST BASIC command and AES functions

by James Luczak

Atari's new ST BASIC provides you with a very powerful command that gives you access to the entire AES (Application Environment Services) library. We're talking about GEMSYS(). It allows you to use an additional sixty-plus functions. There is a rather large problem, however: the *ST BASIC Sourcebook* doesn't tell you how to access any of the AES functions.

This article will provide you with the necessary information to use a handful of the AES functions. In the listing that follows is the BASIC code required and a description of the parameters used by the functions.

We'll only be scratching the surface of the AES library of functions. All the functions described here can be used in any order in a BASIC program. Many AES functions require a specific sequence of function calls to create the desired end result. Those described here can be used independently of each other. They don't have to be used in any specific order.

GEM's AES.

Every time you boot up your computer (after the color show), you end up on what's called the "desktop." The desktop is created with functions from the AES library. The disk icons, menu bar, drop-down menus, trash can icon—all were created using various AES capabilities. Here's a partial list of what the AES functions are responsible for:

Monitoring the mouse buttons;
Monitoring the mouse location;
Setting or retrieving the double click speed;
Providing a timer;
Creating the menu bar;
Producing drop-down menus;
Creating alert boxes;
Creating dialog boxes;
Producing shrinking boxes;
Producing growing boxes;
Dragging boxes;
Creating rubber boxes;
Displaying different mouse forms;
Monitoring boxes;
Sliding boxes;
Moving boxes;
Displaying file selector boxes;
Creating windows;
Providing window controls; and
Updating windows.

BASIC and AES Coordinates.

When using the GEMSYS() command, there's one important thing to remember: all references to X- and Y-coordinates made by the AES library are relative to the screen, while all X- and Y-coordinates referred to by BASIC commands (such as the LINEF command) are relative to the output window.

Assume, for example, that the output window occupies the full screen. The X-coordinate 0 would be

// GEMSYS() *continued*

at the extreme left-hand side of the screen for both BASIC and the AES function. The X-coordinate *619* would be to the extreme right-hand side of the screen for both BASIC and AES. The Y-coordinate 0 for BASIC is at the bottom of the information line (the bar that runs along the top of the screen with the word *OUTPUT* in the center). The Y-coordinate *0* for AES is at the very top of the screen (above the menu bar).

The reason X-coordinates are the same for BASIC and for AES is because the output window doesn't use any horizontal space to draw the border of the window. If, on the other hand, you size the output

AES Graphics Library.

RUBBERBOX	Draws a "rubberbox." The upper left corner of the box is fixed. By holding down the left mouse button and moving the mouse, you can draw boxes of varying sizes.			
	BASIC CODE	DESCRIPTION	BASIC CODE	DESCRIPTION
	1 a#=gb		6 poke gintin+4,xw	xw=Minimum width of box in pixels
	2 gintout=peek(a#+12)	Define integer output	7 poke gintin+6,yh	yh=Minimum height of box in pixels
	3 gintin=peek(a#+8)	Define integer input	8 gemsys(70)	OPCODE
	4 poke gintin,x	x=Coordinate of box (upper left corner)	9 Bxw=peek(gintout+2)	bxw=Width of box when mouse button is released
			10 Byh=peek(gintout+4)	byh=Height of box when mouse button is released
	5 poke gintin+2,y	y=Coordinate of box (upper left corner)		
MOVEBOX	Draws a box outline, moving from one position to another.			
	BASIC CODE	DESCRIPTION	BASIC	DESCRIPTION
	1 a#=gb		6 poke gintin+6,y	y=Coordinate of box (initial position)
	2 gintin=peek(a#+8)	Define integer input	7 poke gintin+8,x1	x1=Coordinate of box (final position)
	3 poke gintin,xw	xw=Width of box in pixels	8 poke gintin+10,y1	y1=Coordinate of box (final position)
	4 poke gintin+2,yh	yh=Height of box in pixels	9 gemsys(72)	OPCODE
	5 poke gintin+4,x	x=Coordinate of box initial position)		
GROWBOX	Draws an expanding box outline.			
	BASIC CODE	DESCRIPTION	BASIC CODE	DESCRIPTION
	1 a#=gb		7 poke gintin+8,x1	x1=Coordinate of box (final size)
	2 gintin=peek(a#+8)	Define integer input	8 poke gintin+10,y1	y1=Coordinate of box (final size)
	3 poke gintin,x	x=Coordinate of box (initial size)	9 poke gintin+12,xw1	xw1=Final width of box in pixels
			10 poke gintin+14,yh1	yh1=Final height of box in pixels
			11 gemsys(73)	OPCODE
	4 poke gintin+2,y	y=Coordinate of box (initial size)		
	5 poke gintin+4,xw	xw=Initial width of box in pixels		
	6 poke gintin+6,yh	yh=Initial height of box in pixels		
SHRINKBOX	Draws a shrinking box outline.			
	BASIC CODE	DESCRIPTION	BASIC CODE	DESCRIPTION
	1 a#=gb		7 poke gintin+8,x	x=Coordinate of box (initial size)
	2 gintin=peek(a#+8)	Define integer input	8 poke gintin+10,y	y=Coordinate of box (initial size)
	3 poke gintin,x1	x1=Coordinate of box (final size)	9 poke gintin+12,xw	xw=Initial width of box in pixels
			10 poke gintin+14,yh	yh=Initial height of box in pixels
			11 gemsys(74)	OPCODE
	4 poke gintin+2,y1	y1=Coordinate of box (final size)		
	5 poke gintin+4,xw1	xw1=Final width of box in pixels		
	6 poke gintin+6,yh1	yh1=Final height of box in pixels		
MOUSE	Changes the mouse form to one of a predefined set.			
	BASIC CODE	DESCRIPTION	BASIC CODE	DESCRIPTION
	1 a#=gb		*3 poke gintin,x*	x=5 Thin cross hairs 6 Thick cross hairs 7 Outline cross hairs 256 Hide mouse form 257 Show mouse form
	2 gintin=peek(a#+8)	Define integer input		
	3 poke gintin,x	x=0 Arrow 1 Vertical bar 2 Bee 3 Hand with pointing finger 4 Flat hand with extended fingers	4 gemsys(78)	OPCODE
MKSTATE	Returns the current mouse location, mouse button state and keyboard state.			
	BASIC CODE	DESCRIPTION	BASIC CODE	DESCRIPTION
	1 a#=gb		7 kb=peek(gintout+8)	kb= Current keyboard state 0 No key pressed 1 Right shift key pressed 2 Left shift key pressed 4 Control key pressed 8 Alternate key pressed
	2 *gintout=peek*(a#+12)	Define integer output		
	3 gemsys(79)	OPCODE		
	4 mx=peek(gintout+2)	mx=Coordinate of mouse's current location		
	5 my=peek(gintout+4)	my=Coordinate of mouse's current location		
	6 mb=peek(gintout+6)	mb=Current mouse button state. 0 No button pressed 1 Left button pressed 2 Right button pressed		

window to occupy only half the screen, BASIC commands will function within it. They will not, for instance, draw a line outside of the window. An AES function will operate anywhere on the screen, regardless of the size of the window.

Y-coordinates are different for BASIC and AES via similar logic. The BASIC output window uses up vertical space drawing the menu bar and information line. This causes the Y-coordinate in BASIC to start

(continued on next page)

AES Event Library.

KEYBOARD Waits for any kind of keyboard input.

BASIC CODE	DESCRIPTION
1 a#=gb	
2 gintout=peek(a#+12)	Define integer output
3 gemsys(20)	OPCODE
4 kb=peek(gintout)	kb=Standard keyboard code

Standard Keyboard Code

A=7745	B=12354	C=11843	D=8260	E=4677	F=8518	G=8775
H=9032	I=5961	J=9290	K=9547	L=9804	M=12877	N=12622
O=6223	P=6480	Q=4177	R=4946	S=8019	T=5204	U=5717
V=12118	W=4439	X=11608	Y=5465	Z=11354		
a= 7777	b=12386	c=11875	d=8292	e=4709	f=8550	g=8807
h=9064	i=5993	j=9322	k=9579	l=9836	m=12909	n=12654
o=6255	p=6512	q=4209	r=4978	s=8051	t=5236	u=5749
v=12150	w=4471	x=11640	y=5497	z=11386		
1=561	2=818	3=1075	4=1332	5=1589	6=1846	7=2103
8=2360	9=2617	0=2864				
Space=14624	Return=7181	Tab=3849	Esc=283	Backspace=3592		

BUTTON Waits for a particular mouse button state.

BASIC CODE	DESCRIPTION
1 a#=gb	
2 openw 2	Open output window
3 gintin=peek(a#+8)	Define integer input
4 gintout=peek(a#+12)	Define integer output
5 poke gintin,bc	bc=Number of mouse clicks to wait for
6 poke gintin+2,mb	mb=Mouse button to look for 1 Left 2 Right
7 poke gintin+4,bs	bs =Mouse button to look for 0 Left up 1 Left down 0 Right up 2 Right down
8 gemsys(21)	OPCODE
9 n=peek(gintout)	n=Number of times the mouse button entered the desired state
10 mx=peek(gintout+2)	mx=Coordinate where event occurred
11 my=peek(gintout+4)	my=Coordinate where event occurred
12 ms=peek(gintout+6)	ms=Mouse button state when event occurred 0 Left/right up 1 Left down 2 Right down
13 kb=peek(gintout+8)	kb=Current keyboard state 0 No key pressed 1 Right shift key pressed 2 Left shift key pressed 4 Control key pressed 8 Alternate key pressed

MOUSE Waits for mouse to enter or leave a specified rectangle.

BASIC CODE	DESCRIPTION
1 a#=gb	
2 gintin=peek(a#+8)	Define integer input
3 gintout=peek(a#+12)	Define integer output
4 poke gintin,fl	fl=Flag for call 0 Return on entry 1 Return on exit
5 poke gintin+2,x	x=Coordinate of rectangle (upper left corner)
6 poke gintin+4,y	y=Coordinate of rectangle (upper left corner)
7 poke gintin+6,xw	xw=Width of rectangle in pixels
8 poke gintin+8,yh	yh=Height of rectangle in pixels
9 gemsys(22)	OPCODE
10 mx=peek(gintout+2)	mx=Coordinate of mouse pointer
11 my=peek(gintout+4)	my=Coordinate of mouse pointer
12 ms=peek(gintout+6)	ms=State of mouse button 1 Left button 2 Right button
13 kb=peek(gintout+8)	kb=Current keyboard state 0 No key pressed 1 Right shift key pressed 2 Left shift key pressed 4 Control key pressed 8 Alternate key pressed

TIMER Waits for a specified amount of time to pass.

BASIC CODE	DESCRIPTION
1 a#=gb	
2 gintin=peek(a#+8)	Define integer input
3 poke gintin,lo	lo= Low word of long value 1 1 Millisecond 1000 1 Second 60000 1 Minute
4 poke gintin+2,hi	hi=High word of long value (0)
5 gemsys(24)	OPCODE

DCLICK Gets or sets the mouse double click speed.

BASIC CODE	DESCRIPTION
1 a#=gb	
2 gintin=peek(#+8)	Define integer input
3 gintout=peek(a#+12)	Define integer output
4 poke gintin,cv	cv=Double click speed 0 Slow 1 2 3 4 Fast
5 poke gintin+2,fl	fl=Flag to set or get double click speed 0 Get speed 1 Set speed
6 gemsys(26)	OPCODE
7 cs=peek(gintout)	cs=Double click speed

Note: If the value in Line 5 (fl) is *0* (get speed), the value in Line 4 is ignored.

22 pixels from the top of the screen (in low and medium resolution).

The GEMSYS() demo program (medium resolution).

This is a simple program. It draws an expanding box which ends in a rectangle in the middle of the screen. As you move the mouse in and out of the box, it exhibits all the predefined forms the mouse pointer can have. As you enter or exit the rectangle, the X- and Y-coordinates are displayed at the top of the screen. To exit the program, hold the left mouse button down while entering or exiting the rectangle.

Conclusion.

The GEMSYS() command is a little more involved than its companion command, VDISYS(). Once you have some of the information needed, it's not too hard to program using the GEMSYS() command.

Although I've only presented a handful of the AES functions in this article, along with the demo program, you'll find that it's not very difficult to use the AES library from BASIC. ▀

Jim Luczak maintains and operates electronic telephone switching and processing equipment. He's been writing computer programs since 1979. He got 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.
BASIC listing.

```
100   '************************ GEMSYS
() DEMO PROGRAM **********************
***
110   '***************************** by
 JIM LUCZAK ****************************
***
120   '
130   ' --------------------------- INI
TIALIZE PROGRAM ----------------------
---
140   fullw 2:clearw 2:flag=0:bs=0:off
set=22
150   a#=gb
160   gintin=peek(a#+8):'
  Defint Integer Input
170   gintout=peek(a#+12):'
  Define Integer Output
180   a$="Hold LEFT mouse button down
when entering or exiting box to EXIT d
emo"
190   '------------------------- DRAW
GROWING BOX --------------------------
---
200   poke gintin,320:'
  X coordinate initial size
210   poke gintin+2,75+offset:'
  Y coordinate initial size
220   poke gintin+4,2:'
  Initial width
230   poke gintin+6,1:'
  Initial height
240   poke gintin+8,270:'
  X coordinate final size
250   poke gintin+10,25+offset:'
  Y coordinate final size
```

```
260    poke gintin+12,100:'
  Final width
270    poke gintin+14,100:'
  Final height
280    gemsys(73):'
  Graf growbox
290    '--------------------------- DRA
W BOX OUTLINE -----------------------
---
300    linef 270,25,370,25
310    linef 370,25,370,125
320    linef 370,125,270,125
330    linef 270,125,270,25
340    gotoxy 3,14:?a$
350    '----------------------- FIND PO
INTER LOCATION ----------------------
---
360    gemsys(79):'
  Graf mkstate
370    mx=peek(gintout+2):'
  Current X coordinate of pointer
380    my=peek(gintout+4):'
  Current Y coordinate of pointer
390    flag=0
400    if (mx>270 and mx<370) and (my>2
5+offset and my<125+offset) then flag=
1
410    '------------------- SET UP PARA
METERS FOR MOUSE EVENT --------------
---
420    poke gintin+2,270:'
  X coordinate of rectangle
430    poke gintin+4,25+offset:'
  Y coordinate of rectangle
440    poke gintin+6,100:'
  Width of rectangle
450    poke gintin+8,100:'
  Height of rectangle
460    '------------------- MAKE SURE M
OUSE POINTER IS VISIBLE -------------
---
470    poke gintin,257:'
  Show mouse form
480    gemsys(78):'
  Graf mouse
490    '--------------------------- MAI
N PROGRAM LOOP ----------------------
---
500    while bs=0
510    poke gintin,index:'
  Mouse form
520    gemsys(78):'
  Graf mouse
530    poke gintin,flag:'
  Flag for event mouse call
540    gemsys(22):'
  Event mouse
550    mx=peek(gintout+2):'
  X coordinate of mouse
560    my=peek(gintout+4):'
  Y coordinate of mouse
570    bs=peek(gintout+6):'
  State of mouse button
580    if flag=0 then flag=1:b$="Mouse
ENTERED" else flag=0:b$="Mouse EXITED"
590    gotoxy 9,1:?b$" box at these Coo
rdinates.  X = "mx"  Y = "my"   "
600    index=index+1:if index>7 then in
dex=0
610    wend
620    '----------------------- DRAW SH
RINKING BOX -------------------------
--
630    poke gintin,320:'
  X coordinate final size
640    poke gintin+2,75+offset:'
  Y coordinate final size
650    poke gintin+4,2:'
  Final width
660    poke gintin+6,1:'
  Final height
670    poke gintin+8,270:'
  X coordinate initial size
680    poke gintin+10,25+offset:'
  Y coordinate initial size
690    poke gintin+12,100:'
  Initial width
700    poke gintin+14,100:'
  Initial height
710    gemsys(74):'
  Graf shrinkbox
720    '----------------- MAKE SURE MOU
SE POINTER IS AN ARROW --------------
---
730    poke gintin,0:'
  Make mouse form an ARROW
740    gemsys(78):'
  Graf mouse
750    '--------------------------- CLEA
N UP AND END ------------------------
---
760    poke gintin,256
770    gemsys(78)
780    clearw 2:end
```

●

ST CHECKSUM DATA.

(see page 53ST)

```
 100 data  564, 146, 477, 876, 310, 0,
 830, 136, 361, 827, 4527
 200 data  210, 260, 640, 668, 103, 41
, 493, 588, 718, 809, 4530
 300 data  485, 533, 654, 554, 209, 11
2, 688, 674, 682, 171, 4762
 400 data  188, 395, 320, 215, 426, 49
0, 623, 937, 328, 846, 4768
 500 data  749, 552, 312, 131, 440, 47
, 56, 156, 813, 443, 3699
 600 data  310, 57, 789, 836, 989, 222
, 319, 408, 449, 945, 5324
 700 data  943, 892, 556, 867, 322, 49
9, 395, 737, 939, 6150
```

●

BRATACCAS
PSYGNOSIS LIMITED
1st Fl., Port of Liverpool Bldg.
Pier Head, Liverpool L3 1BY
England
520ST $39.95
(Medium or high resolution)

by Clayton Walnum

Over the years, I've played oodles of adventure games (as have we all). One of the hazards of this type of dedicated usage is that, even though the games are still well done and challenging, one tends to become a bit jaded. After all, each Infocom game looks much the same as the next. This repetition in design is also evident in games like the **Ultima** series. As a matter of fact, virtually all adventure games bear a resemblance to those which have gone before.

Sooner or later, it's bound to happen —boredom sets in. There are no surprises anymore. When was the last time you booted up an adventure game and said, "Wow!" when it came up on the screen?

So, when I say I was stunned by this new import from England, it should be a clue to you that something significant is going on here.

The story goes like this. . .You're Kyne, a genetic scientist who's discovered a process for the creation of a superbeing. The government decides that, rather than use these beings for peaceful causes, it would be a great idea to set up an army trained to kill. Kyne, being of a nonviolent bent, refuses to pass his research on to the authorities and goes into hiding.

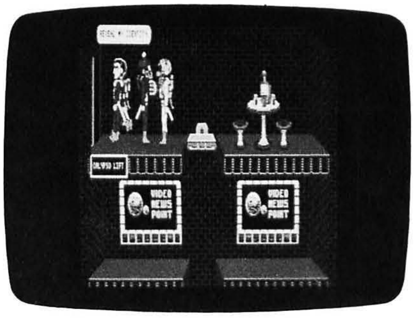
The government doesn't find this to be an adequate solution to their differences. They immediately place a warrant on Kyne's head, accusing him of selling his studies to the underworld. Of course, the underworld *does* have an interest (an understatement) in Kyne's research. They figure that, "Hey, he's on the run. Maybe we *can* get him to work for us."

And so, Kyne finds himself pursued by both extremes of the law (or perhaps they're really quite similar).

Escaping from Earth, Kyne makes his way to a small mining asteroid where he's heard that evidence attesting to his innocence exists. The name of the asteroid? **Brataccas**.

Playing **Brataccas** is like stepping right into a comic book, getting that chance every kid dreams of—to become the hero. Each character is detailed and lifelike in movement.

As the citizens of **Brataccas** make their way about the asteroid, they exchange pleasantries (or nasty remarks, depending on who's doing the talking). In the comic book tradition, word bubbles appear over the characters' heads when they speak. These bubbles follow them as they stride onto or off of the screen, allowing plenty of time to read their contents.



Brataccas.

Quite honestly, this game looks so great you don't even have to play. Just slap it into movie mode, sit back and watch. . .Saturday morning cartoons!

When you get ready to play, you may control Kyne in one of three ways. The default control mode is with the mouse (natch). The two other possibilities are joystick or keyboard. Should you choose to send Kyne on his way from your keyboard, the program allows you to define the keys you wish to use, a nice feature.

Be forewarned. Due to the large number of possible movements, manipulating Kyne can be a bit clumsy at first. Be patient. With a little practice, you'll soon be running and jumping with the best of them. I found that, of the three control methods, the joystick worked best for me.

The gameplay consists of moving between rooms (or on the surface of the asteroid), gathering clues, bribing the inhabitants for information and generally trying to stay out of trouble. You must keep a low profile while you search frantically for your salvation.

Should you run into serious difficulty, you've no choice but to draw your sword and battle it out. When fighting, there are various thrusts and parries available to you. The action is quite lifelike; when you become skilled with your weapon, the battles can be surprisingly exciting.

Beware: most of the swordsmen in this game know their stuff. If you're not careful, you'll find a word bubble over your head with the exclamation "Arrrrgggggg!"—which means it's back to the start of the game for you.

The safest way to deal with people on **Brataccas** is with your sword undrawn. If you have a money bag or a bottle of the asteroid's best, you can get a lot of information from the Snitches. These guys hang around the bars waiting to trade their knowledge for a little of that green stuff or perhaps a good stiff drink.

Brataccas is a neat place. Scattered throughout the rooms are all sorts of gadgets, such as rotating cameras (Big Brother is watching you. . .) and video screens where a game of **Space Invaders** is frequently interrupted for important news flashes. On Tannoys (speakers) you can listen to police broadcasts. There are switches to turn various items on and off, not to mention Electro Bombs, money bags, bottles of booze, scrolls, IDs and, of course, the evidence itself.

There are about sixteen different characters on **Brataccas**, each with their own distinctive appearance and personality. One of my favorites is Commander Stopp, the chief of police, who lost his legs in a laser fight and now moves about in a jet-propelled hover dish.

Other characters consist of the aforementioned Snitches, the ubiquitous police, several bar owners, guard droids, assassins and the evil Kol Worpt, **Brataccas**' arch-villain in residence. They all move about freely, and you never know where or when you're going to bump into someone significant.

The manual is attractive, sporting a cover by one of my favorite artists, Roger Dean (all you Yes fans will immediately recognize his distinctive style). The text, printed on slick paper, is well written and, many times, downright funny. These people definitely have a sense of humor. Wait until you see the hint sheet on page 27.

Besides all the funny business, the game's functions are accurately described, and the story background is a quick, fun read. As an added convenience, the rear of the manual contains a pocket for storage of the disk, which makes the package easy to keep together on a book shelf.

Also included in the package is a poster of the cover art. It's a nice little plus that goes well beyond the call of duty.

Psygnosis should be congratulated on a fine effort. I can't wait to see what products they'll be bringing to the marketplace in the future. If this game is any indication of what's in store for ST adventurers, then there are exciting times ahead. What can I say? Buy it.

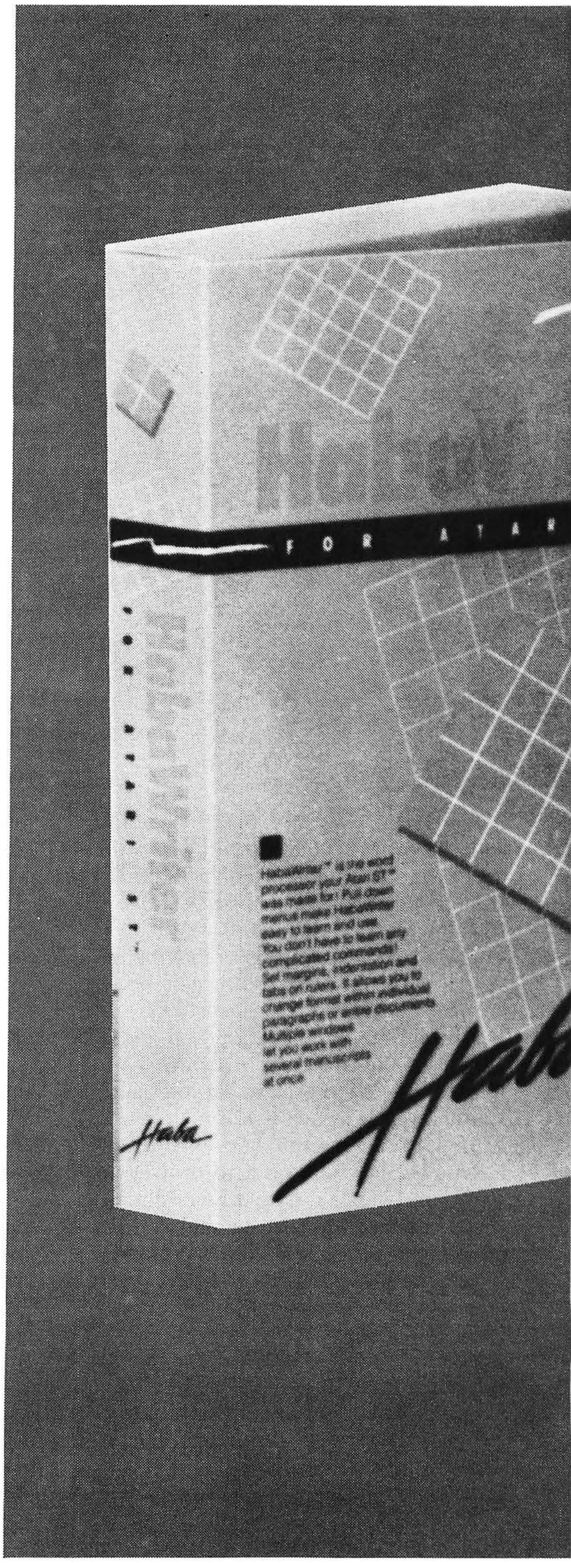
STylish Software

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

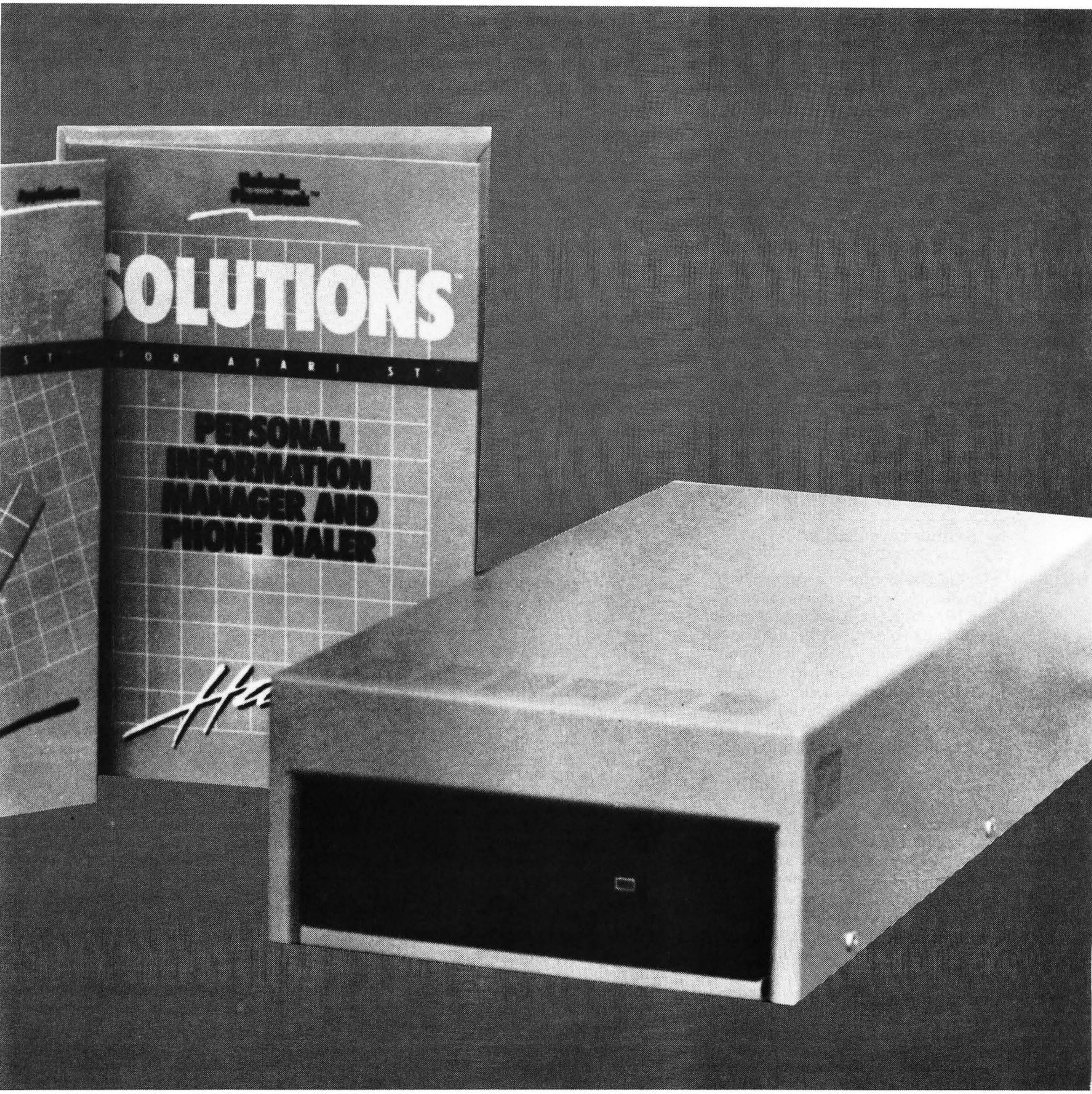
The new **HabaDisk**™ **10 Megabyte** hard disk for the 520 ST is a Winchester plug-in hard disk that is capable of storing the equivalent of more than 12 dual-sided 800K diskettes and retrieves information in seconds (3 msec. track-to-track access time). It is self-powered and completely Atari ST compatible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

520 ST is a trademark of Atari Corp. HabaWriter, Habadex PhoneBook, HabaModem, HabaDisk, Haba Checkminder, Haba Mail Room , HabaMerge, Solutions: Wills, and Solutions: Business Letters are trademarks of Haba Systems, Inc. Hayes is a trademark of Hayes Microcomputer Corp. GEM is a trademark of Digital Research, Inc.

STupendous Storage



Haba

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

MI-TERM
MICHTRON
576 S. Telegraph
Pontiac, MI 48053
(313) 334-5700
520ST $50.00

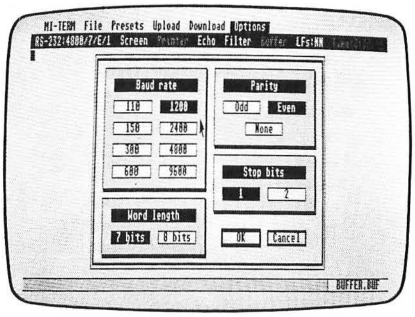
by Arthur Leyenberger

In the six months the Atari 520ST computer has been available, one company has stood out as having the most complete line of software for it. It's one thing to have an extensive line of software, but if the software isn't up to par, then the distinction becomes moot. MichTron, a small Michigan software company, holds this honor—and well it should, since all of its products are quality efforts.

MichTron entered the ST software sweepstakes with the first arcade game, **Mudpies**, and has since concentrated mostly on utility software. I've been using **M-Disk** (ramdisk) and **M-Utilities** (sector and disk copier) for several months. Now MichTron has entered the application market with their new telecommunications program, **Mi-Term**.

The first thing you notice about **Mi-Term** is that it uses the familiar GEM features, like drop-down menus and dialog boxes. Naturally, selecting options is performed by pointing and clicking with the mouse. In addition, any previously loaded desktop accessories—a calculator, the control panel or printer driver—are available from within the program, as they should be in a properly designed GEM application.

Mi-Term is truly a full-featured telecommunications program. In addition to providing simple two-way communication capability between a variety of computer systems, **Mi-Term** allows you to automate your log-on procedure, as well as your most frequently used commands and ASCII uploads, to conserve valuable connect time. Any number of custom configurations may be saved as individual files and loaded whenever you want them. This avoids repeated setups and allows an expert user to design a system that a beginner can easily follow.



Mi-Term.

Mi-Term supports two different error-checking protocols for flawless file transfer: DFT and XMODEM. Eight different operating speeds (up to 9600 baud) may be used, and an automatic capture buffer is provided. The buffer file may be changed at any time. Its current name is always displayed on the **Mi-Term** menu bar, and the contents may be viewed whenever you wish.

One useful aspect of this feature is the visual indicator that shows how many characters have been saved into the buffer. There's a bar along the bottom of the screen, much like a GEM slider bar, which shows the percentage of memory buffer currently in use. As more characters are added to the buffer, the bar immediately displays the change.

One of the features that makes **Mi-Term** an outstanding program is its so-called macro capability. Up to fifty-six individual command strings can be assigned to unique keys. The twenty-six alphabetical keys are used with the ALTERNATE key, and function keys F1 to F10, are used individually, as CTRL-F1 to F10 and as ALT-F1 to F10, to provide what MichTron calls "presets." The definitions of these presets are saved in the **Mi-Term** configuration files for future use.

With the presets menu from the top menu bar, you can view, change or add new character strings to your function keys. An extensive set of options is available with this feature. The various special functions available within the preset strings are implemented by imbedding certain control sequences (displayed on the screen for ease of use) in the preset string. Waiting for certain characters from the host, setting character and line delays, toggling the screen—or just some of the functions—on and off.

There are too many options to describe here, but basically you have the ability to build macros that will work with any on-line computer system imaginable. As a thoughtful and useful touch, MichTron has provided several files on the distribution disk, showing previously created presets for such popular on-line services as CompuServe, Delphi, MCI Mail, etc.

Uploading can be performed in either DFT or XMODEM protocols. To upload a file, you simply click on the protocol desired, at which time a dialog box appears, listing the files on the disk. Once you've clicked on a file, another dialog will appear on-screen, showing the name of the file selected and the total number of blocks required to send it, along with the message *awaiting handshake*.

Once communication with the remote system begins, **Mi-Term** constantly informs you of the block number being sent, the percentage of the file already sent, and the percentage of blocks sent that did not require retransmission (error-free rate). Also displayed are the number of re-tries for the current block in progress and any messages relevant to the upload. Again, the quality of the program is apparent— when the upload is complete, the computer will beep at you as a signal.

Downloading is as straightforward as uploading. You would select either X-MODEM or DFT protocol, and a dialog appears listing the files on the disk. Point and click at a filename or type in a new one, to begin the download. The same dialog box is used as in the upload mode, to monitor the progress of the file transmission.

To download an ASCII file, no special protocol is necessary. Either open and close the capture buffer manually to receive ASCII text, or, if the remote system supports the capture buffer transfer mode (also known as DC2/DC4), it will be done automatically.

Finally, you can use the options menu to edit, load and save **Mi-Term**'s options and parameters. Some of the options you can select are: dump incoming text to a printer; toggle the screen on and off; toggle a character filter on and off, to strip out unwanted control codes; select one of three line feed modes, to add or not add a line feed to each incoming carriage return character; toggle the clock display on and off and reset it; and send a true break.

You can change your RS-232 parameters (parity, baud rate, number of stop bits, etc.), in order to suit the remote system you're communicating with. This is done by calling up the dialog box and clicking the mouse button on your choices. Information about the current status of **Mi-Term** and the RS-232 port is always displayed in the status line, just beneath the menu bar at the top of the screen. Functions that are active are displayed in black on the white background, whereas inactive functions are shaded in gray.

Overall, **Mi-Term** is an excellent telecommunications program. Kudos should be given to the author, John Weaver, for not only creating a useful program, but designing it in such a way as to be easy to use. If you want or need more features than are provided in, say, **ST-Talk**, yet don't want to spend a $100 for **PC/Intercom** and get only a text-based program, then you should seriously consider MichTron's latest product.

Further, all of the commands and options work with the intuitiveness of the GEM interface—point to the desired menu name, it drops down, and you point and click on the command. I can't think of an easier, more feature packed program than **Mi-Term**.

CIRCLE #130 ON READER SERVICE CARD

ST INDEX TO ADVERTISERS

READER SERVICE #	ADVERTISER	PAGE #
118	Abacus Software	50ST
131	Applied Computers, Inc.	82ST
132	Atari Corp.	83ST, 84ST
128	Beckemeyer Development Tools	75ST
130	CAL COM	82ST
127	Coast to Coast Computers	74ST
121	Commnet Systems	55ST, 82ST
129	Eugi/Haba	78, 79ST
125	Martin Consulting	66ST
124	Megamax, Inc.	59ST
120	MegaSoft	54ST
119	Progressive Computer Products	52ST
126	Regent Software	66ST
139	Rocky Mountain Atari Service	82ST
122	Serious Software	55ST
123	Soft Logik	58ST

CIRCLE #139 ON READER SERVICE CARD

CIRCLE #121 ON READER SERVICE CARD

CIRCLE #131 ON READER SERVICE CARD

Desk File View Options
✓ Show as Icons
Show as Text
✓ Sort by Name
Sort by Date
Sort by Size
Sort by Type
A:\
bytes used in 31 items.
B:\
238934 bytes used in 11 items.
SC1224
ATARI

THE ST™ COMPUTER LINE FROM ATARI.®

IT'S LIKE GETTING THE POWER AND SPEED OF A FERRARI® FOR THE PRICE OF A FORD.®

When Atari introduced the 520ST™, we set the personal computer industry on its ear.

Nobody had ever produced a machine so powerful and technically advanced for such an incredibly low price. Nobody but Atari has done it yet.

The competition was stunned.

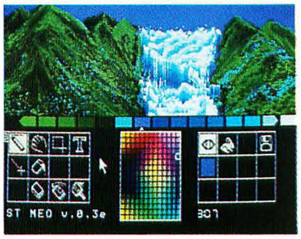
The critics wrote rave reviews.

And consumers were ecstatic.

We could have rested on our laurels, but we didn't.

Instead, Atari extended the ST concept to a new computer called the 1040ST™.

The amazing new 1040ST is even more powerful than the 520ST and years ahead of all the competition at almost any price. The only question in 1986 isn't which company to buy a computer from, but which computer to buy from Atari.



	ATARI® 1040ST*	COMMODORE® AMIGA™	IBM® PCAT™	APPLE® Macintosh™	APPLE IIc®
Price	$999	$1795	$4675	$1995	$1295
CPU Speed MHz	68000 8.0	68000 7.16	80286 6.0	68000 7.83	65002 1.0
Standard RAM	1 MB	256K	256K	512K	128K
Standard ROM	192K	192K	64K	64K	16K
Number of Keys	95	89	95	59	63
Mouse	Yes	Yes	No	Yes	Optional
Screen Resolution (Non-Interlaced Mode) Color Monochrome	640x200 640x400	640x200*** 640x200***	640x200 720x350**	None 512x342	560x192 560x192
Color Output	Yes	Yes	Optional	None	Yes
Number of Colors	512	4096	16	None	16
Disk Drive	3.5"	3.5"	5.25"	3.5"	5.25"
Built-in Hard Disk (DMA) Port	Yes	No	Yes	No	No
Midi Interface	Yes	No	No	No	No
# of Sound Voices	3	4	1	4	1

Atari 520ST with 512K RAM, $799.

*Connects to standard color T.V. For RGB color monitor add $200.
**With optional monochrome board (non bit-mapped).
***Interlace Mode – 640x400.

At $799, the 520ST gives you 512 Kbytes of RAM, a high-resolution monochrome monitor, 2-button mouse, and 3.5" disk drive.

At $999, the 1040ST gives you 1024 Kbytes of RAM, an ultra high-resolution monochrome monitor, 2-button mouse, and a built-in double-sided 3.5" disk drive, plus built-in power supply. Both the 520ST and the 1040ST can be connected directly to your own color T.V. Or you can add an Atari RGB color monitor to get the sharpest, most colorful images possible. Add $200 for color monitor.

It's simply a matter of choosing which model best fits your needs.

And whether you choose the 520ST or the 1040ST, you'll be getting the power and speed of a Ferrari for the price of a Ford.

In fact, you'll save hundreds and in some cases thousands of dollars over comparable computers. Which is why consumers are still ecstatic. Why the critics are still writing rave reviews. And why the competition is still stunned.

Ferrari is a registered trademark of Ferrari Italia SpA and Ferrari of America, Inc. Ford is a registered trademark of Ford Motor Company. IBM and PCAT are registered trademarks of International Business Machines Corp. Commodore and Amiga are trademarks of Commodore Electronics LTD. Apple, Apple IIc, and Macintosh are trademarks of Apple Computer, Inc. Atari, 520ST, 1040ST, and ST are trademarks of Atari Corp.

ATARI®

POWER WITHOUT THE PRICE.™

CIRCLE #132 ON READER SERVICE CARD