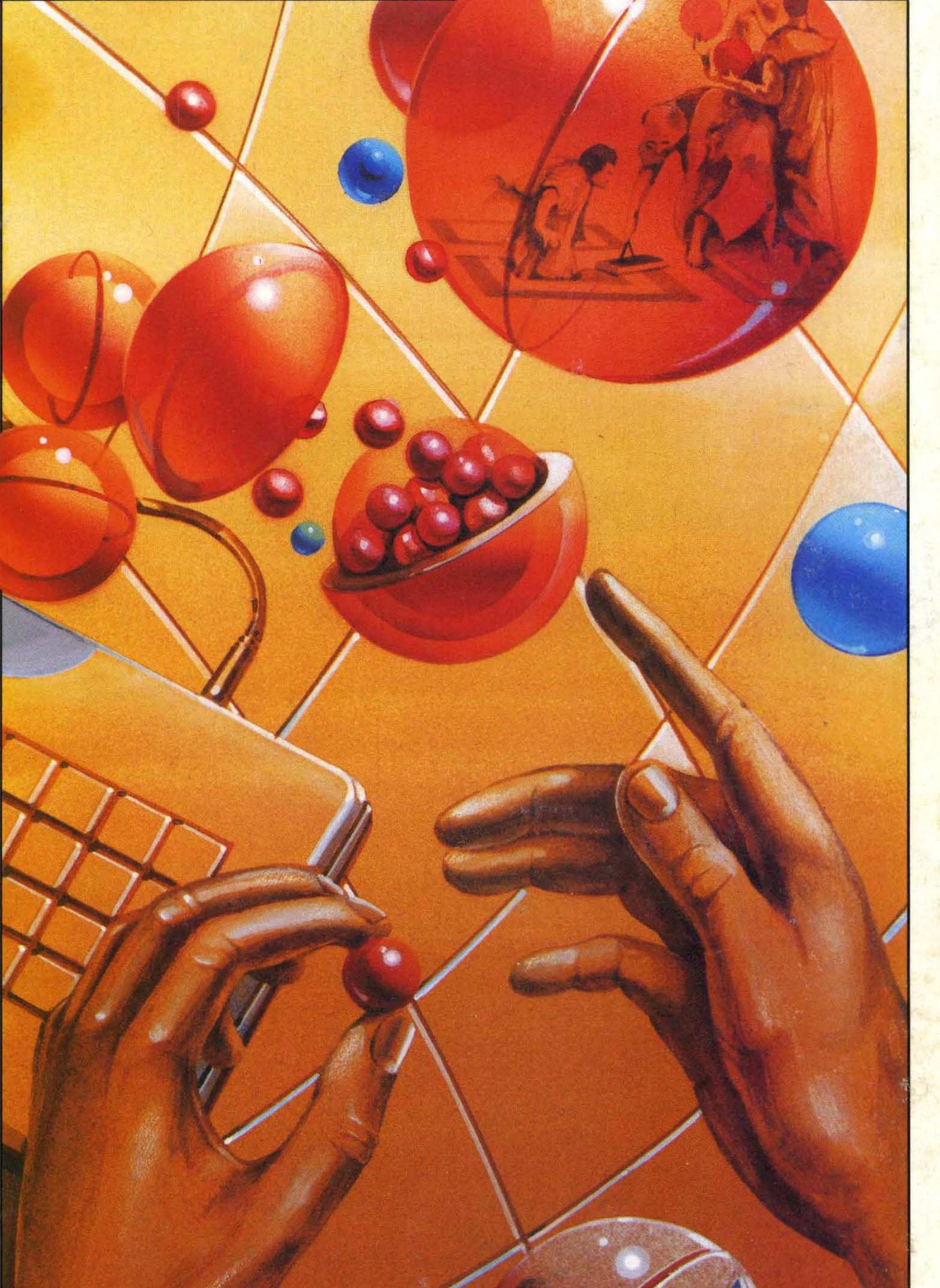
No. 52 March-April 1990 $3.95

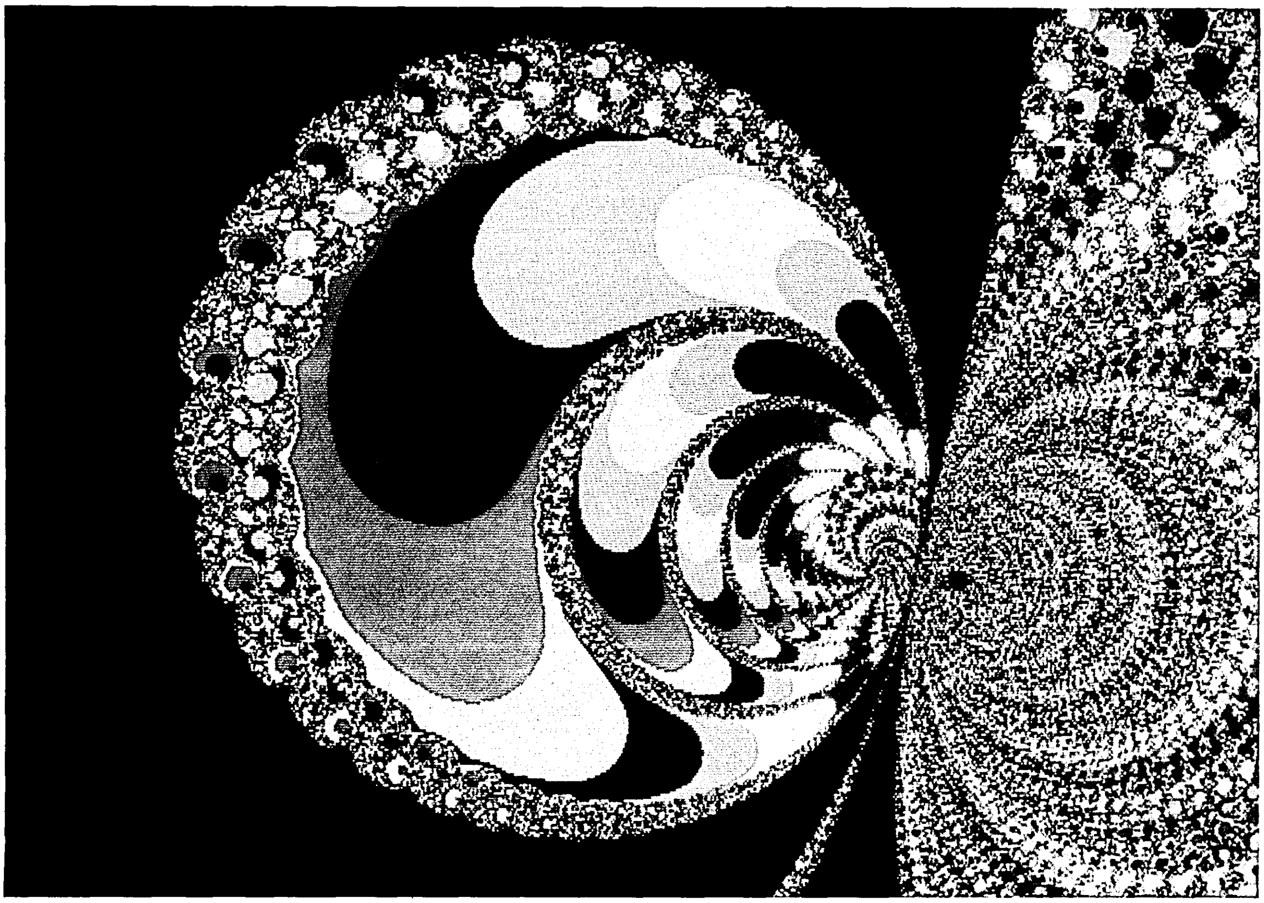
THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA



Object Oriented Programming

First it was BASIC, then it was structures, now it's objects. C++ afiçionados feel, of course, that objects are so powerful, so encompassing that anything could be so defined. I hope they're not placing bets, because if they are, money's no object.



"CHAOS MADE TO ORDER"™

Explore the Magnificent and Infinite World of Fractals

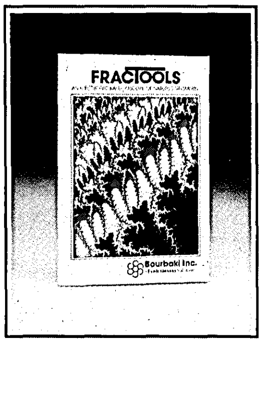
with **FRACTOOLS**™

AN ELECTRONIC KALEIDOSCOPE OF NATURES GEOMETRY™

With *FracTools*, you can modify and play with any of the included images, or easily create new ones by marking a region in an existing image or entering the coordinates directly. Filter out areas of the display, change colors in any area, and animate the fractal to create gorgeous and mesmerizing images. Special effects include *Strobe, Kaleidoscope, Stained Glass, Horizontal, Vertical and Diagonal Panning,* and *Mouse Movies.*

The most spectacular application is the creation of self-running *Slide Shows.* Include any PCX file from any of the popular "paint" programs. *FracTools* also includes a *Slide Show Programming Language,* to bring a higher degree of control to your shows. (Slide Shows are saved as ASCII text files.)

On one level, *FracTools* is an engaging graphics "toy". On another level, it is great for stress management/relaxation therapy. Last, but not least, *FracTools* provides a visual example, or window, on the evolving science of *chaos theory.* Each new image reveals patterns not perceivable in its "parent" image. If you are familiar with James Gleick's popular book *Chaos—Making a New Science,* you are probably aware of the significance of fractals in the visualization of chaos.



System Requirements: IBM PC/XT/AT/PS2 and compatibles, EGA or VGA monitor, DOS 2.0 or higher, 640K of memory and a hard disk. A math coprocessor is desirable, but not required.

Suggested Retail Price: $59.00. To Order Call 1-800-289-1347.

Bourbaki Inc.

rEvolutionary Software P.O. Box 2867 - Boise, ID 83701

Reader Service Number 190

ERAC CO.

8280 Clairemont Mesa Blvd., Suite 117
San Diego, California 92111 • (619) 569-1864

Baby 386-20/24 Motherboard

One 32 bit-slot, Five 16-bit slot, Two 8-bit slot. Maximum 8 meg on board (1 meg SIMMS). 0K on board. 80287 or 80387 or Weitec Co-Processor slot. Shadow RAM.

-20 $630 -24 $650

1 meg SIMMS-80ns $115

16 BIT VGA Card

RESOLUTION	COLOR
640x480	256 of 256*
800x600	16 of 256*
1024x768	16 of 256*

*with 512K RAM

Runs all standard VGA modes, 256K RAM on board. Drivers for AutoCad, Lotus, Framework, GEM, VP, WP, WS & MS Windows. 1 year warranty.

For 512K $30 extra $185

MOTHERBOARDS 386DX/SX/286

DX5000 386/25MHz-32K Cache-16 Meg-Simms-AMI Bios-Full	$996
DX4000 386/20/24MHz Shadow-8 Meg-Simms-AMI-Baby	630/650
P9400 386SX/16MHz 8 Meg-Simms-Phoenix-Bios-Baby	375
P9200 386 SX/16MHz 2 Meg-Dip-AMI-Bios-Baby	365
SUPABOARD 286/12MHz 4 Meg-Dip-Dallas Clk-AMI-Baby	197

I/O, KB, CONTROLLERS, Etc.

XT Enhanced Hard Disk Drive Controller with Cables	48
DC-11M AT Hard/Floppy Contr. 1:1 438Kb/Sec, To 2048 Cyl.	93
AT I/O PLUS 1 Par(Lpt 1-3), 2 Ser(Com 1-4), Game, Cables, Ser 2(opt)	38
VGA-16 16-bit VGA Board 800x600 with Driver Software	118
2400 Baud External Modem, with Software and Manual	99
KB5161 AT/XT 101 Keyboard, Cherry Keyswitches (Click)	47
COLORMOUSE Black-Red-Blue-Beige-Green-Yellow, Software & Manual	39
VOICE MASTER KEY, Add voice commands to software XT/AT	147

MONITORS

VGA 1489 1024x768, .28 Dot, 14", Swivel, Hor. 31/35KHz, Vert. 50, 60, 70, 87 Hz	430

CASES

MINI-TOWER, 230 W, Reset, Turbo, Keylock, Speaker	143
TOWER, 230W, Reset, Turbo (2 Dgt), Keylock, Speaker	229

KAYPRO Equipment Bargains

9" Green Monitor-83, 84, K16	$60
Host Interface Board	15
Keyboard	50
Replacement Power Supply	70
Drivetek 2.6M FDD (Robie or K4X)	75

We Repair CPM Kaypros

CPM COMPUTERS

K4-84	425
K10	495
K4X	425

We carry all IC's for Kaypro repair.

IC's

81-189 Video Pal	$15
81-194 RAM Pal	15
81-Series Char. Gen. ROMs	10
81-Series Monitor ROMs	10

Test Equipment

OSCILLOSCOPES

TEK 7403N/7A18N/7B50A 60 MHz	$650
Leader LB0520 30 meg Dual Trace	300
TEK 475 Dual Trace 200 MHz	1250
Scope Probe x1, x10 100 MHz	25

ANALYZERS

TEK 491 10MHz-40GHz	$3500
HP851B/8551B 10HMz-40GHz	1500
Biomation 805 Waveform Rcrdr	195
Biomation 8100 2-Channel Waveform Recorder	295
HP1600A Logic Analyzer 16ch	295
HP1600A/1607A Logic Analyzer 32ch	495
Gould K40 32ch Logic Analyzer	750

MISCELLANEOUS

Optronics 550 MHz Freq Cntr	$95
Heatgun 120Vac 7 A	35

TERMINALS

Televideo 925	$99.00

NiCds

AA Cells .6ah	$1.00
12V Pack AA Cells .7ah	6.50
Sub-C Cells 1.5ah	1.50
12V Pack Sub-C	10.00
Double D Cell 2.5V 4ah unused	8.00
C Cells	1.75
7.2V RC-Pack 1.2ah	18.00

GEL CELLS

6V 5ah	5.00
6V 8ah	6.00
12V 15ah	15.00
12V 2.5ah	8.50
D Cell 2.5ah	2.00

ROBOTICS

5V DC Gear Motor w/Tach 1"x2"	$7.50
Z80 Controller with 8-bit A/D	15.00
12V Gear Motor 30 RPM	7.50
Cable: DB9M-DB9F 1' length	2.00
High Voltage Power Supply Input 15-30V DC Output 100V 400V 16KV	6.50

SWITCHERS

AT 200W Pulls, tested	$35.00
5V/75, 12V/6, -12V/3, -5V/5	85.00
5V/9.5A, 12V/3.8A, -12V/.8A	39.00
5V/3A, 12V/2A, -12V/.4A	19.50
5V/6A, 12V/2A, -12V/1A	29.00
5V/6A, 24V/1 1/4A, 12V/.6A, -12V/.6A	29.00
5V/30A	39.00
5V/100A	100.00
5V/120A	110.00
HP DC/DC 12Vin,5V/8A,12V/5A,-5V/.3A	45.00

VERSATEC 8222F 22"
Electrostatic Printer Plotter
200 dots per inch. Up to D size.
1" per second $2,999

AT 80286-6 CPU BOARD
with reset and mono/color switch. Connector for KB, Battery & SPKR. Phoenix Bios (tested with Award 3.03), 6MHz, can be upgraded to 8 or 10MHz. Use with backplane, add memory board, I/O board, etc.
ONLY $99

HOURS: Mon-Fri 9-6, Sat 10-4
For more information, please call (619)569-1864.

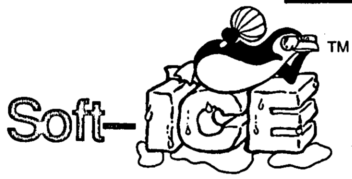
MINIMUM ORDER $25.00
TERMS: VISA OR MasterCard (Add 3%). Certified Checks. Money Order. NO COD. Personal Checks must clear BEFORE we ship. Include shipping charges. California residents add 7 1 4% Sales Tax.

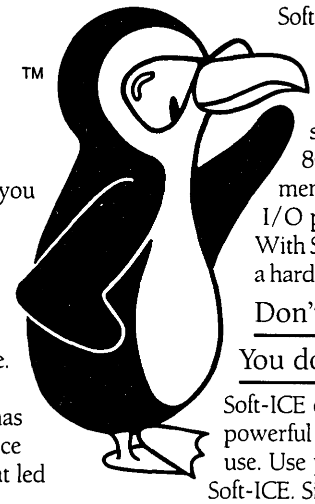
Reader Service Number 93

FINALLY. A debugging tool tough enough to handle the DOS Nasties.

New Version 2.0

Soft-ICE™





Nasty over-write? No sweat!

Soft-ICE memory range break points help you track down memory over-write problems whether you are doing the over-writing or another program is over-writing you.

Hung program? No problem!

When the system hangs, you now have hope. With Soft-ICE you can break out of hung programs no matter how bad the system has been trashed. And with Soft-ICE's back trace ranges you can re-play the instructions that led up to the crash.

Program too large? Not with Soft-ICE!

Soft-ICE runs entirely in extended memory. This means you can debug even the largest DOS programs. And since your program runs at the same address whether Soft-ICE is loaded or not you can find those subtle bugs that change when the starting address of your code changes.

System debugging? Soft-ICE is a natural!

Soft-ICE is ideal for full source level debugging of TSRs, interrupt service routines, self booting programs, DOS loadable device drivers, real-time kernels, non-DOS O/Ss and ROMs. Soft-ICE can even debug within DOS & BIOS.

How Soft-ICE Works

Soft-ICE uses the power of the 80386 to surround your program in a virtual machine. This gives you complete control of the DOS environment, while Soft-ICE runs safely in protected mode. Soft-ICE uses the 80386 to provide real-time break points on memory locations, memory ranges, execution, I/O ports, hardware & software interrupts. With Soft-ICE you get all the speed and power of a hardware-assisted debugger at a software price.

Don't want to switch debuggers? You don't have to!

Soft-ICE can run stand-alone or it can add its powerful break points to the debugger you already use. Use your favorite debugger until you require Soft-ICE. Simply pop up the Soft-ICE window to set powerful real-time break points. When a break point is reached, your debugger will be activated automatically.

MagicCV with Soft-ICE

Using Soft-ICE with CodeView gives you the features necessary for professional level systems debugging. MagicCV and Soft-ICE can work in concert with CodeView to provide the most powerful debugging platform you will find anywhere.

"These may be the only two products I've seen in the last two or three years that exceeded my wildest expectations for power, compatibility and ease-of-use."

—Paul Mace
Paul Mace Software

Soft-ICE	$386
MagicCV	$199
MagicCV for Windows	$199
Buy Soft-ICE & MagicCV(W)	—Save $86.
Buy MagicCV and MagicCVW	—Save $100.
Buy All 3	—Save $186.

30 day money-back guarantee
Visa, MasterCard and
AmEx accepted

1988 BYTE Award of Distinction

New Soft-ICE 2.0 features

- ☐ Back Trace Ranges
- ☐ Symbolic & Source level debugging
- ☐ EMS 4.0 support with special EMS debugging commands
- ☐ Windowed user interface

Nu-Mega TECHNOLOGIES

CALL TODAY (603) 888-2386
or FAX (603) 888-2465

RUN CODEVIEW IN 8K

MagicCV™

CodeView is a great integrated debugger, but it uses over 200K of conventional memory. MagicCV uses advanced features of the 80386 to load CodeView and symbols in extended memory. This allows MagicCV to run CodeView in less than 8K of conventional memory on your 80386 PC.

NEW VERSION FOR C 6.0

Attention Windows Developers!
Version available for CVW.

P.O. BOX 7607 ■ NASHUA, NH ■ 03060-7607

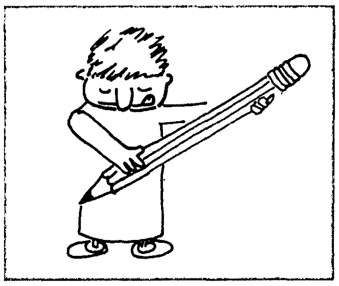
Reader Service Number 110

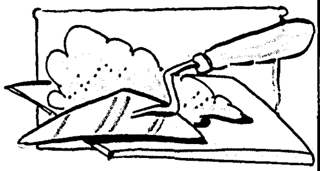
THE MICRO TECHNICAL JOURNAL

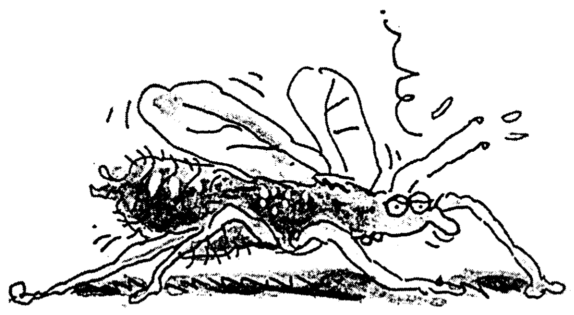
MICRO CORNUCOPIA

MARCH/APRIL 1990 - ISSUE NO. 52

FEATURES



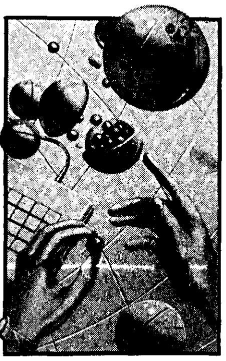




COLUMNS

FUTURE TENSE

COVER



Cover Illustration by Rob Sanford.

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

Editor and Publisher
David J. Thompson

Associate Editors
Gary Entsminger
Larry Fogg
Cary Gatton

Contributing Writers
Bruce Eckel
Michael S. Hunt
Scott Ladd
Karl Lunt

Accounting
Sandy Thompson

Advertising & Distribution
Reader Services
Nancy Ellen Locke & Laura Shaw

Graphic Design & Production
Carol Steffy

MICRO CORNUCOPIA (ISSN 0747-587X) is published bi-monthly for $18 per year by Micro Cornucopia, Inc. 155 NW Hawthorne, Bend, OR 97701. Second-class postage paid at Bend, OR and additional mailing offices. POSTMASTER: Send address changes to MICRO CORNUCOPIA, PO Box 223, Bend, OR 97709. Recycled Paper

SUBSCRIPTION RATES:

1 yr. (6 issues)	$18.00
2 yr. (12 issues)	$34.00
3 yr. (18 issues)	$48.00
1 yr. Canada & Mexico	$26.00
1 yr. Other foreign (surface)	$36.00
1 yr. Foreign (airmail)	$50.00

Make orders payable in U.S. funds on a U.S. bank.

CHANGE OF ADDRESS:
Please send your old label and new address to:

MICRO CORNUCOPIA
P.O. Box 223
Bend, Oregon 97709

READER SERVICES:
For orders and subscription problems call 1-800-888-8087, 9 am to 5 pm, Pacific time, M-F. FAX your VISA or MC order to us, our FAX number is (503) 389-6833.

BBS - 24 hrs. 300-1200-2400 baud
8 Bits, No Parity, 1 Stop Bit 503-382-7643

TECHNICAL ASSISTANCE
For help call 503-382-8048,
9 am to noon Pacific time, M-F

Copyright 1990 by Micro Cornucopia, Inc.
All rights reserved
ISSN 0747-587X

The Audit Bureau

AROUND THE BEND

By David J. Thompson

Preemptive Editorial

Debugging Without Tears

Software manufacturers constantly face the question: When is a package clean enough to release? I mean, put yourself in their shoes; how much time and money would you want to spend cleaning up when your biggest competitor just unleashed a salvo across your bow (complete with dealer incentives)?

And think of all the copies he's sending to reviewers. By the time you've finished pest control, he'll be gracing the front pages of *PC, PC World, PC Week,* and *PC Resource.* You may be clean, but you're old news and you're broke.

That doesn't have to be the scenario, however. In one quick stroke, you can get your package debugged quickly and cheaply; at the same time, you can prevent your competition from reaching the marketplace.

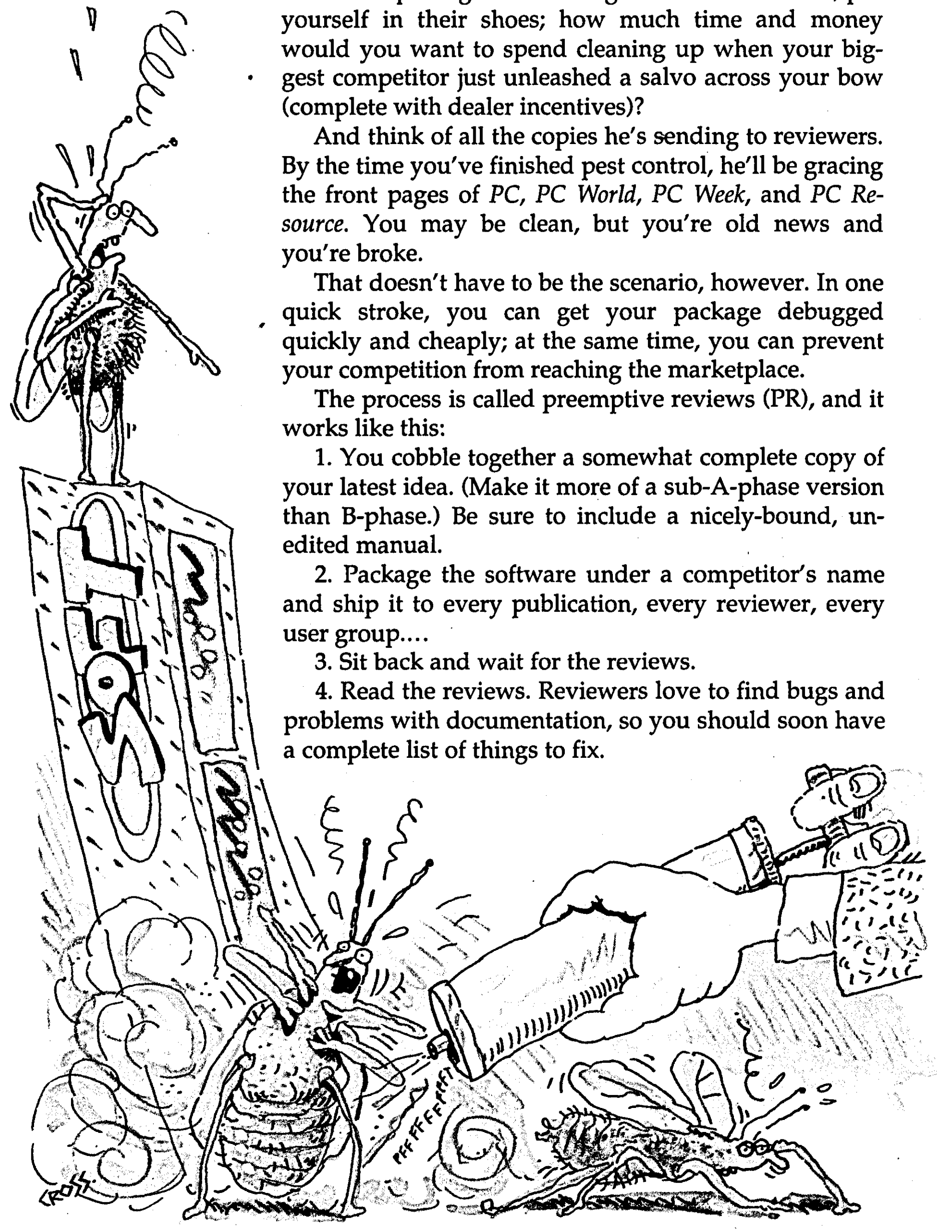
The process is called preemptive reviews (PR), and it works like this:

1. You cobble together a somewhat complete copy of your latest idea. (Make it more of a sub-A-phase version than B-phase.) Be sure to include a nicely-bound, unedited manual.

2. Package the software under a competitor's name and ship it to every publication, every reviewer, every user group....

3. Sit back and wait for the reviews.

4. Read the reviews. Reviewers love to find bugs and problems with documentation, so you should soon have a complete list of things to fix.

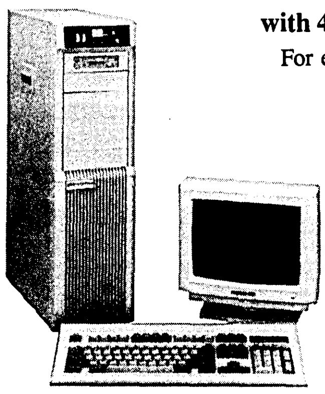


Continued on page 74

GREAT VALUES FROM MICROSPHERE!

SUPER 25Mhz 386 SYSTEM!

with 4MB RAM and Rotary Voice Coil Hard Drive



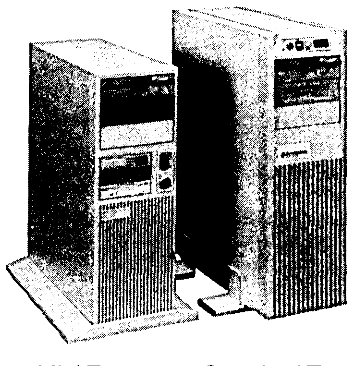
For extra speed and reliability we've included a 66MB Mitsubishi MR535 voice coil hard drive with an access time of 28ms, and a super fast 1:1 interleave HD controller card.

- 25Mhz 80386 CPU, AMI Bios, Full Size MB
- 4MB of RAM, expands to 8MB on MB
- 1.2MB and 1.44MB Floppy Drives
- 12" Amber Monitor w/Mono Graphics Card
- 101-Key Enhanced Keyboard, Focus or Maxiswitch
- Serial/Parallel/Game Ports, Clock/Calendar
- 200-Watt Power Supply
- Socket for 80287, 80387 or Weitek
- Full 1 Year Warranty

$2350 *(Mega Tower)*

w/Std AT Case	**$2250**
w/1MB	$250 Less
20Mhz 386	$100 Less

Tower Cases w/Power Supply



Mini Tower	*Standard Tower*
$225	*$269*

AT /386-SX SYSTEMS

Includes: 1MB RAM, 1.2MB & 1.44MB FD, 40MB Hard Drive (28ms), Fast Hard Drive Controller, Mono Graphics Video Card, 12" Amber Monitor, 101 Key Keyboard, Serial/Parallel/Game Ports, Clock/Calendar, Full 1 Year Warranty. **FREE** assembly and testing.

6/10Mhz 286/AT	$1249
8/12Mhz 286/AT	1295
8/16Mhz 286/AT	1395
16Mhz 386-SX	1449

Video upgrades for our systems (includes video card and monitor)

ATI Graphics Solution	39
CGA Color	175
EGA Color	349
VGA Color (Analog, 16 bit w/Amazing monitor)	379
CGA/EGA/VGA (Multisync)	450

The Most Cost Effective Way to Speed Up your AT or 386 System!

Upgrade your hard drive controller. Discover the NEXT generation of hard drive controller cards. As the chart below demonstrates, the speed improvements are incredible!

Controller Card Type	Data Xfer Rate
Standard IBM AT, 3:1 Interleave, MFM	167 KB/Sec.
DTK WA2, 2:1 Interleave, MFM	261 KB/Sec.
NCL, 1:1 Interleave, MFM	**522 KB/Sec.**
Western Digital 1:1, RLL, w/Cache	**799 Kb/Sec.**

***Note:** Test results using 10Mhz AT, Mitsubishi MR535 Hard Drive and SpinRite Disk Optimizer.*

VGA Color ONLY $379... LOOK

When you buy a complete MicroSphere Computer System. Upgrade includes: 16 bit VGA video card and 14" Analog VGA Monitor with a .28 mm dot pitch.

MOTHERBOARDS

XT/Turbo 4.77/10Mhz	$75
AT 6/10Mhz	189
AT 6/12Mhz	225
AT 8/16Mhz	289
386SX 16Mhz	339
386 8/20Mhz w/Phoenix Bios, holds up to 8MB on board	699
386 16/25Mhz w/AMI Bios holds up to 8MB on board	825
386 33Mhz	Call

POWER SUPPLIES

150-Watt XT Power Supply	50
200-Watt AT Power Supply	78

EXPANSION CARDS

Clock	18
Game (Joystick)	14
Parallel Port (LPT1, 2 or 3)	18
Serial Port, 2 ports, 1 installed, (COM1 or 2)	18
2nd Serial Port Kit	18
Serial Port, 4 ports installed	99
Multi Drive Controller, up to 2 drives, Supports 360K, 720K, 1.2MB & 1.44MB	39
3 & 4th Floppy Controller	79

AT/386

AT Multi-IO, Ser/Par/Game	33
2nd Serial Port Kit	20
AT 3MB EMS Memory Card (0K)	99

CHIPS/ETC.

Memory 64K, 256K, 1MB	Call
Math Coprocessors	Call
XT Components and Complete Systems	Call

FLOPPY DRIVES

360K	73
1.2MB	77
3 1/2" 720K	79
3 1/2" 1.44MB	85

HARD DRIVES

XT 20MB Miniscribe/Kalok 8425 (65ms)	239
8425 w/controller	279
XT 30MB Miniscribe/Kalok 8438 (65ms)	249
8438 w/controller	299
AT 44MB Mitsubishi MR535, Rotary Voice Coil (28ms)	429
AT 71MB Micropolis 1335 Voice Coil (28ms)	595

VIDEO CARDS

Mono Graphics w/Parallel port	39
ATI Graphics Solution Herc w/CGA Emulation	79
CGA/EGA w/256K	95
CGA/EGA/VGA 8-Bit	119
CGA/EGA/VGA 16-Bit	129

CONTROLLER CARDS

NCL 1:1 MFM Controller	$110
WD 1006-SR2 1:1 RLL w/Cache	139

CABINETS

XT Slide Case, Lock, LED	38
AT Slide Case, Lock, LED 3 half ht., 1 full ht. drives	72
Baby Tower, w/200 watt PS, 2 half ht., 2 - 3 1/2" Drives, Baby Motherboard ONLY	145
Mini Tower, w/200 Watt PS, 3 half ht, 2 - 3 1/2" Drives, holds Full or Baby size MB	225
5 Bay Std. Tower, 250 Watt PS, 3 half ht., 1 full ht drives	269
6 Bay Mega Tower, 200 Watt PS	239

MONITORS

12" Amber Monochrome TTL	89
CGA Color RGB	249
EGA/CGA Autoswitch	362
CGA/EGA/VGA Multisync (.31 dot)	489
VGA Analog (Amazing .28 dot)	379

KEYBOARDS

Maxiswitch (AT Only) 101-Key	$65
Keytronic* Enhanced 101-Key	59
Focus* 101-Key Tactile, Switchable Control/Caps Lock, Dust Cover	65

(#1 Find by Micro C Staff)

* Keyboards are XT/AT switchable

SOFTWARE

MS-DOS 3.3 w/GW Basic	59
DR-DOS 3.3 w/GEM	49
SpinRite II Disk Optimizer by Gibson Research	79
386Max Memory Manager for 386 Systems	69

MicroSphere INC. COMPUTERS "HARDWARE MANUFACTURER SINCE 1983"

Orders Only **Please!** **1-800-234-8086**

Tech Calls: **(503) 388-1194** Hours: Mon-Fri 9:00-5:30

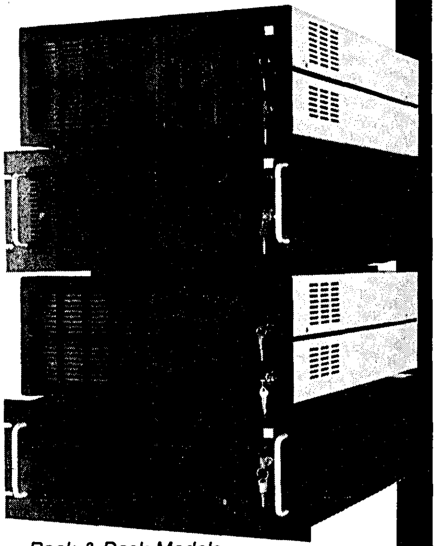
855 N.W. WALL • BEND, OREGON 97701

**Prices are subject to change without notice. Shipping CHARGES will be added. *1-year warranty/30-day money back (subject to restrictions)*

Reader Service Number 2

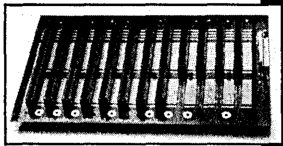
Rack & Desk PC/AT Chassis

Integrand's new Chassis/System is not another IBM mechanical and electrical clone. An entirely fresh packaging design approach has been taken using modular construction. At present, over 40 optional stock modules allow you to customize our standard chassis to nearly any requirement. Integrand offers high quality, advanced design hardware along with applications and technical support *all at prices competitive with imports.* Why settle for less?



- *Rack & Desk Models*
- *Accepts PC, XT, AT Motherboards and Passive Backplanes*
- *Doesn't Look Like IBM*
- *Rugged, Modular Construction*
- *Excellent Air Flow & Cooling*
- *Optional Card Cage Fan*
- *Designed to meet FCC*
- *204 Watt Supply, UL Recognized*
- *145W & 85W also available*
- *Reasonably Priced*

Now Available Passive Backplanes

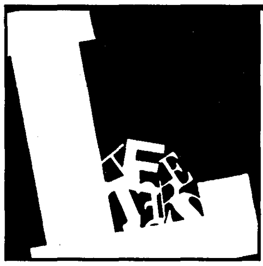


INTEGRAND RESEARCH CORP.

Call or write for descriptive brochure and prices:
8620 Roosevelt Ave. • Visalia, CA 93291
209/651-1203
TELEX 5106012830 (INTEGRAND UD)
FAX 209/651-1353
We accept Bank Americard/VISA and MasterCard

IBM, PC, XT, AT trademarks of International Business Machines. Drives and computer boards not included.

Reader Service Number 22



Letters

Hot Tips

Two points not mentioned in Karl Lunt's (very good) article, "Getting Started In Hardware," in Issue #51, were:

(1) If your soldering iron falls off the bench, do not try to catch it. I mention this because I and most of my friends have made this mistake at one time or another—admittedly, not more than once each!

(2) A beginner often puts too much solder on a joint, and $4 for a cheap de-soldering pump (Radio Shack again) makes one worth having from the start. The job ends up much neater when you have the option of cleaning off and starting again.

I hope you continue with this series, which might culminate in descriptions of systems such as wirewrap, Verowire, and Speedwire. There are a lot of good prototyping products in the commercial world that are too expensive for experimenters to buy "on spec." But once you know their capabilities, they more than repay their cost.

C. W. Rose
850 State St. #215
San Diego, CA 92101

Low Cost "Big Screen"

I was reading through "Around the Bend" in your July/August issue and was struck by the plight of the woman who couldn't read normal sized characters on a 12" screen. I think I may have a low cost solution for her.

Every so often you see an ad for a "magnifying sheet." These are sheets of clear plastic that have a flat fresnel lens cut into one side. Turn the smooth side toward you, and you have a reasonably powerful magnifier.

The University of Chicago bookstore sells these for the use of people who have to do a lot of reading (everyone at the University of Chicago has to do a lot of reading!). They come in several sizes—from 2" to 8" or so diagonally.

Distortion is relatively small, and the light transmission is good. One possibility is to mount one of these lenses in front of the monitor. You can adjust how much magnification you get by moving the lens with respect to the screen.

I wish I could remember the name of the manufacturer, but at the moment I'm squeezed into a coach seat en route to Utah. I do know the price for the largest lens I saw was well under $10. Also, the idea isn't original; I believe an ad in your magazine or the late *Profiles* (I can't remember which) used to sell something called "Maggie." It was advertised as a remedy for Kaypro owners who were tired of small screens.

This might turn out to be a cost-effective solution for computer users with relatively mildly impaired vision. It cer-

Letters continued on page 78

You Asked for ***more...***

Tele™ gives the ***boot*** *to limitations*

Berry Computers presents The Tele Toolkit — a complete Operating Systems Kernel

TELE IS BOOTABLE

The Tele operating system is now available in a bootable version that emulates the MS-DOS kernel. Tele executes binary programs intended for MS-DOS in a multitasking, windowing environment.

You can use Tele to run several programs at once thereby eliminating boring and unproductive delays. For instance, in a software development environment, you can set a compiler running and simultaneously edit another source file.

Tele uses a preemptive task scheduler. Its file system is fully compatible with MS-DOS. It runs on any processor compatible with the Intel 8086 family.

Tele improves display performance by a factor of 2, whether you use multiple windows or not! On processors slower than 20 MHz the improvement is even greater. Tele windows replace the BIOS display driver with a much superior algorithm. Tele windows, unlike Microsoft Windows, are simply a way of packing multiple display screens onto a single physical device. You control the priority tasks with the task scheduler, not the display algorithm. And your programs will never be asked to repaint their displays.

Most programs intended for MS-DOS will multitask under Tele. Because MS-DOS was hastily designed, and does not incorporate modern features (even for 1980 when it was developed), Microsoft could find no way to make MS-DOS multitasking and still support all its existing applications. Their solution was to give MS-DOS version 5 a new name, OS/2, introduce new and incompatible programming paradigms, require an 80286 or 80386 processor, and consume vast quantities of memory for the system to sit in even while idle.

Tele simply does the best possible. Tele services the documented interface to MS-DOS, fits in less than 100K of memory, and provides the important features of OS/2, including installable file systems.

Source code in C and assembly is included for the interface to MS-DOS. If you have to have MS-DOS idiosyncrasies, you can easily put them in yourself.

Treat yourself to an operating system where the marketing and financial folks had no say at all in the design. The only considerations in Tele's design were what it would be like to use and program under. The result is an efficient, structured program that either provides or specifically anticipates all features of modern mainframe and desktop operating systems.

The Tele Operating System includes all object modules and the source code for MS-DOS emulation which is an external function. The Tele Tool Kit components are only required if you need the source code for the internal functions. Object modules are provided for Microsoft C version 5.1 unless otherwise requested.

Tele Tool Kit	
Multitasking (MT)*	**$ 40**
Windows (WI)*	**$ 40**
File System (FS)	**$ 40**
Tele Operating System and Tool Kit	**$200**
Demonstration Disk	**$ 5**

*MT formally called SK. WI formally called CD.

Tele is available from:

Crosby Associates
P.O. Box 248
Sutter Creek, California
95685

CALL NOW TO ORDER:
(209) 267-0362 FAX (209) 267-0115

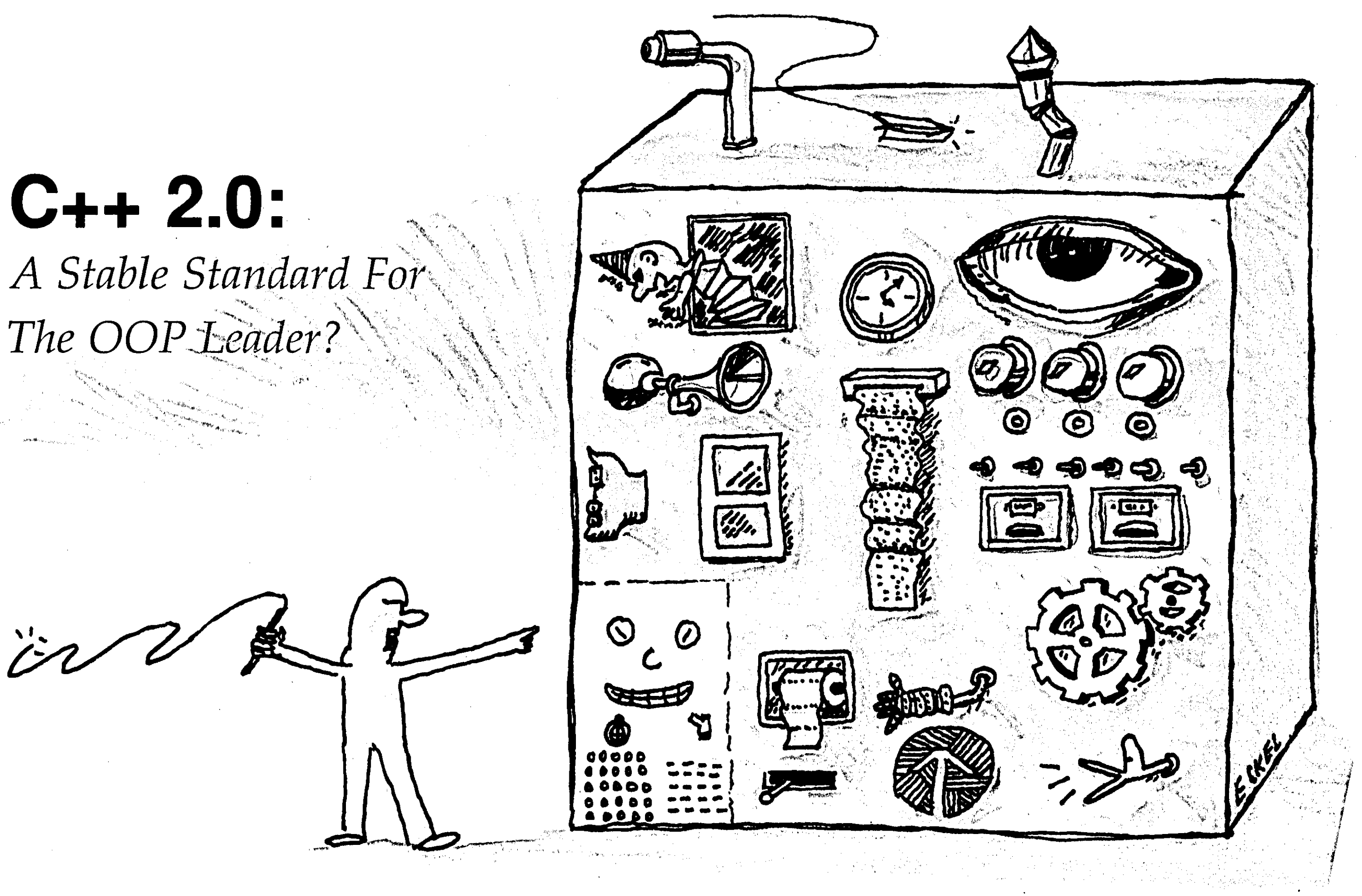
Visa, Mastercard, American Express & Discover Card accepted. Orders outside U.S.A. please use credit card.

MS-DOS and OS 2 are trademarks of Microsoft Corporation.

Reader Service Number 147

C++ 2.0:

A Stable Standard For The OOP Leader?



YOU CAN BUILD & MANAGE BIGGER, MORE COMPLICATED PROGRAMS WITH OOP

If you're reading this introduction just to groan at a few cheap C puns, you're going to be disappointed. C++ is a serious tool for serious programmers and, as you'll see, release 2.0 is even seriouser.

In the sixties, Norwegian computer scientists found that existing programming languages weren't well-suited for simulating and solving real-world problems. Simulation is both data and event driven, so it doesn't fit easily into real-time or batch-processing categories.

Simulations generally consist of many entities which have known internal states (or properties) and external operations (or behaviors). These entities fall into categories (or types). Types can share properties with other types. Types which share some properties and behaviors inherit those properties and behaviors from some base type.

In C++, a "class" corresponds to a type and an "object" to a variable (or instance) of a type. We invoke an object (or use a variable) by sending it a message.

OOPs (object-oriented programmers) have been sending messages to objects officially since 1967, when Jens-Ole Dahl and Kristen Nygarrd developed Simula-67, the first OOP language.

Since then, and particularly in recent years, new languages have sprung up, borrowing from Simula's design. And several existing languages have added OOP features (see Gary's Tidbits columns on Turbo Pascal 5.5 in Issues 49 and 50 of *Micro C* for a good look at OOP, Pascal-style). There's even a movement afoot to add OOP extensions to COBOL!

C++ is an object-oriented extension of C developed in 1985 by another Scandinavian—a Dane, Bjarne Stroustrup, who works at AT&T Bell labs. Although AT&T paid to create the language, they didn't mistakenly prevent others from implementing it—in fact, they made it easy. You can port their version (cfront) to different computer architectures, a big reason C++ has been so successful.

The C++ which has become popular conforms to AT&T release 1.2. Last summer, AT&T released a new spec for C++ which significantly improves the language while minimizing incompatibilities with old code. This article will focus on the features of this new release (2.0) and review two of the implementations currently available for the PC (Zortech C++ 2.06 and Comeau Computing's cfront).

Other implementations of C++ for the PC may be available by the time you read this. Borland and Microsoft are both developing C++ compilers. MetaWare is creating one for the 386. JPI has been promising one for a *long* time.

There are also ports of cfront to other platforms (Oasys makes many) and several native-code compilers (Oregon Software C++ for the Sun, and Gnu C++—from the Free Software Founda-

By Bruce Eckel
P.O. Box 760
Kennett Square, PA 19348

tion—for the Sun and other UNIX machines). There's even a port of cfront (from Lattice) for the Amiga.

What C++ Isn't

At OOPSLA '89 (Object-Oriented Programming Systems, Languages and Applications conference), Peter Deutsch, one of the key developers of Smalltalk, called C++ a "500 pound gorilla," as in "what does a 500 pound gorilla get for its birthday? Anything it wants."

There's been some resentment in the object-oriented community because of the success of C++. Other languages—like Smalltalk—have been around longer and have more mature environments, yet C++ has stolen the show.

At the "C++ at Work" conference in November, 1989, Stroustrup indirectly explained this phenomenon by showing the family tree of object-oriented languages (see Figure 1). Notice that Smalltalk and similar languages (like Actor) are in one branch, C++ is in another, and the Common-Lisp Object System (CLOS) is in yet another. The point here is that C++ is *not* Smalltalk.

Although AT&T paid to create the language, they didn't mistakenly prevent others from implementing it—in fact, they made it easy.

The Smalltalk and C++ designers had different goals in mind. Smalltalk is a "pure" OOP language: everything must be an object, and all objects are derived from some other object. This allows two things to be built into the language: garbage collection and a debugger.

Smalltalk effectively replaces the operating system, thereby creating its own environment on non-MS-DOS systems (this isn't quite the case on MS-DOS systems). All objects in Smalltalk are created on the heap (free store). Smalltalk (which has always been interpreted) only handles references (addresses) to the objects. When you're done with an object, you simply release it. For example, in the Smalltalk expression:

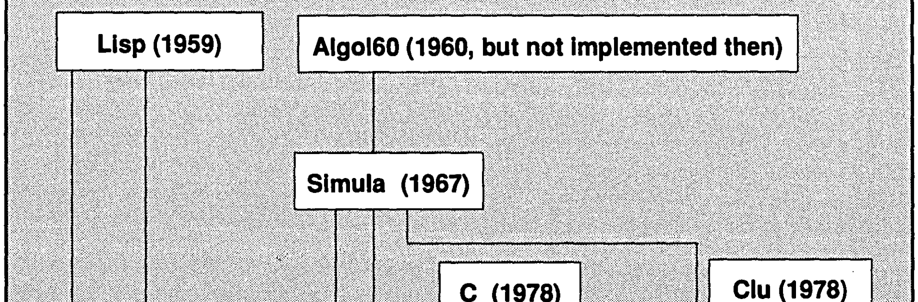
```
D := A + (B + C).
```

a temporary object is created to hold the result of B+C. This object is then added to A, and the result bound to a second temporary, which is finally bound to D. (This binding is fast since it only involves copying an address.) When the first temporary is no longer needed, it's released.

This means that the heap space contains an object you no longer have a reference to. Eventually, the heap will fill up with these "garbage" objects. When you run out of space, the garbage collector looks through the heap for objects no longer in use and releases them.

C++ creates objects (including temporaries) on the stack and destroys them when they go out of scope. You can explicitly create objects on the heap, but then you're responsible for cleaning them

Figure 1—Language Evolution with Approximate Dates



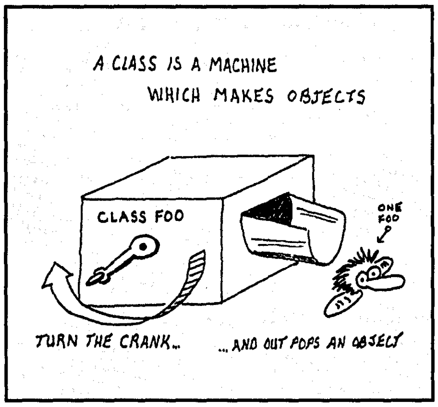
up (or you can write your own garbage collector for specific applications).

Run-time debugging, garbage collection, an impressive assortment of standard classes, and a built-in user interface are built into the Smalltalk environment. Having so many built-in tools makes Smalltalk good for prototyping and development. You can try out new ideas quickly, and your finished applications run from within the Smalltalk environment, so you don't have to rebuild an environment for individual applications.

In C++, you generate binary executable code (which you can distribute!), which is neither larger nor slower than its C equivalent. You can also handle real-time situations more reliably because there's no automatic garbage collector to start up at an inopportune moment.

Strengths are weaknesses in the wrong situation—and vice versa.

To understand Smalltalk, think in terms of messages. If you send a message foo to an object, and the object has no method (definition of what to do with a message) for foo, it passes the message up to its base class.


A CLASS IS A MACHINE WHICH MAKES OBJECTS
CLASS FOO
ONE FOO
TURN THE CRANK..
...AND OUT POPS AN OBJECT

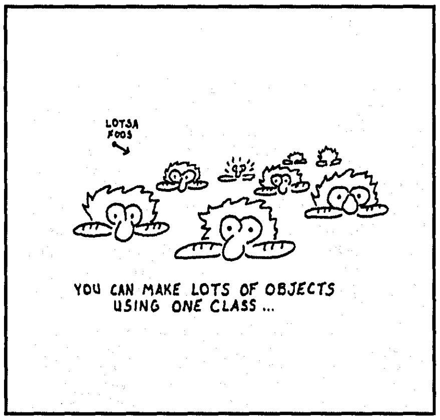
If the base class has no method for foo, it passes the message up to *its* base class, and so on. All this message passing goes on at run-time, so the length of time it takes to execute a message depends partly on how high up the inheritance tree the method is.

If there's no method for foo, Smalltalk flashes a run-time error: "method not found." You can only discover this by running your program and sending all possible messages. During rapid prototyping, this is actually a feature because it means you can execute parts of your Smalltalk program without fleshing out the whole thing.

In C++, the equivalent of message passing occurs at compile-time. The compiler or linker will complain if you try to send a message to an object that has no corresponding method. The time required to execute a method doesn't depend on the inheritance hierarchy.

The difference between the way C++ and Smalltalk (and CLOS) handle messages is the difference between strong typing and weak typing. Smalltalk is weakly typed, which means you can send any message to any object at run-time and the message will be accepted.

C++ follows the Simula model: it's strongly typed (Simula also had added run-time checks, which C++ does not). In C++ a message can only be sent to an object if the corresponding method exists for that object. Otherwise, the C++ compiler can't guarantee that the message can be executed.

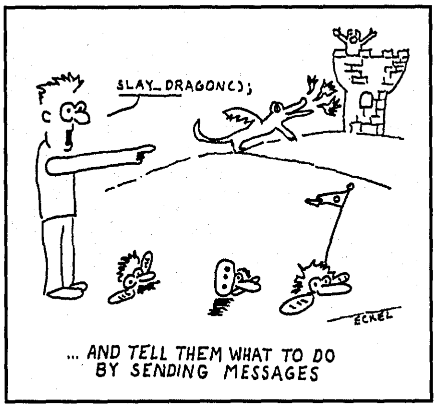


Note that Smalltalk, CLOS, and similar languages emphasize experimentation and flexibility (with the attendant run-time overhead). C++ is conservative, which insures the robustness of the final program, and allows the compiler to do most of the work (so the program runs faster).

Some insiders call C++ a "hybrid" language, since it (1) allows functions which aren't associated with any particular object, and (2) has C's data types which don't have all the properties of objects (you can't inherit from them, for instance).

In C++, everything doesn't have to be an object. *(Gary's note: The OOP extensions in Turbo Pascal are also like C++ in this respect. You add objects to existing code (or even new code) when you need to, not because you have to. C'ers and Pascal programmers who want to OOP (object-oriented program) can, without learning an entirely new system or language.)*

Smalltalk optimizes rapid prototyping and experimentation; C++ optimizes production of practical applications. Each excels in its own area. The designers of C++, Smalltalk, and all the other OOP languages had different goals in mind. Remember this, and you can stay out of religious wars.



New Features In 2.0

Stroustrup and the gang at AT&T have added several fascinating features to the C++ specification in release 2.0. These include: multiple inheritance, type-safe linkage, explicit support for abstract classes, automatic definition of the copy-constructor and operator=(), class-by-class overloading of the operators new and delete, additional operator overloading, pointers to members, new qualifiers for member functions, a variety of neat tricks I call "new forms of initialization," and some small changes for clarity.

I'll discuss the major changes; you can find the rest in Chapter 11 of *Using* C++ (or other C++ books which describe release 2.0).

Multiple Inheritance

Single inheritance (SI) allows you to fan multiple types out from a single base type. You can say "a Porsche is a car, a Volvo is a car," etc.... If you recall Venn diagrams from "new math" (what a joke), you know that most things belong to more than one set. That is, "a Porsche is a car and a vehicle." A bicycle is also a vehicle.

C++ release 1.2 allows only single inheritance, which leaves a conceptual hole when you want to express membership in more than one set. Multiple Inheritance (MI) in C++ 2.0 fixes this by allowing you to derive a class from more than one base class. MI isn't a prerequisite for an object-oriented language; Smalltalk, Actor, and Objective-C don't support it.

The syntax of MI is straightforward. With SI, inheritance is expressed like this:

```
class base { /* ... */ };
class derived:
      public base { /* ... */ };
derived::derived() : () { /* ... */ }
```

where derived::derived() is the constructor for the derived class and the empty parentheses after the colon (the initialization list) is the call to the base class constructor. Since you're only inheriting from one base class with SI, it's obvious which constructor should be called, so you use empty parentheses with no function name in release 1.2.

Multiple inheritance is expressed this way:

```
class base1 { /* ... */ };
class base2 { /* ... */ };
class mderived: public base1,
                public base2 {
  /* ... */ };
mderived::mderived() : base1(),
            base2() { /* ... */ }
```

Notice MI is a simple extension to SI, but that the calls to the base-class constructors in the initialization list must use the names of the base classes (otherwise the compiler won't know which one you're talking about). This is exactly how you initialize member objects.

In the above examples, the explicit call to the constructors with no arguments in the initialization lists isn't necessary. The compiler will automatically call the default constructors if you don't.

Virtual Base Classes

Suppose you're welding multiple base classes together using MI. You discover that two of the base classes are conceptually referring to the same sub-object which lives higher up in the inheritance tree.

For example (to revert to what Rob Murray at AT&T calls "the bloody shape example," since we've beaten on it so many times), consider a class displayableShape and a class printableShape which you can display on the screen and send to the printer, respectively. If you want to create a class storableShape which can be displayed on screen, printed, *and* stored to disk, you could use MI:

```
class storableShape :
  public displayableShape,
  public printableShape {
    /* ... */
};
```

Presumably, displayableShape and printableShape each contain internal representations of a shape. Let's say they're both derived from a base class baseShape. Now, if you want to change the color of the shape to blue, should you be responsible for making sure that the baseShape sub-objects in both displayableShape *and* printableShape agree that the color (and everything else) is the same? No, this doesn't make sense because you're talking about the same subobject.

To solve the problem, specify that the same sub-object be used with the keyword virtual. In this case, make baseShape a virtual base class in both displayableShape and printableShape:

```
class displayableShape :
  virtual public baseShape {
    /* ... */ }
class printableShape :
  virtual public baseShape {
    /* ... */ }
```

The definition for class storableShape remains the same, but only one sub-object of class baseShape will be created for a storableShape. Test this by printing the sizeof() the most-derived class with and without virtual base classes.

The responsibility of initializing virtual base classes always lies with the most-derived class—in this case, printableShape. If you don't explicitly call the constructor for the virtual base class in the most derived class, the constructor with no arguments is automatically called for you.

Type-safe Linkage

One problem with plain C is that it lets you use functions improperly. For example, you can call a function like this:

```
x = foo(bar);
```

If the compiler hasn't seen a prototype for foo(), it'll just assume you know what you're doing: that foo() takes an argument of whatever type bar is and returns a value of whatever type x is. The linker will look for a function with an identifier that probably looks like _foo.

If it finds it, it links it in, regardless of the arguments and return values for foo(). Thus, in plain C, there's no type safety (i.e., no type checking) of arguments and return values during linking.

Function prototypes in C are, of course, a big help, *if you use them.* If you precede the above call to foo() with a declaration of foo() including the full prototype, the compiler will insure that the calls are made correctly. Note, however, that function prototypes are only useful at compile time, and they're easily disabled (by not using them!).

Release 2.0 forces you to use prototypes. A function called before it's declared will generate an error. Although this insures that the function you call matches a declaration somewhere, it doesn't guarantee that the function you declare matches the one the linker finds. For this, 2.0 introduces type-safe linkage.

Type-safe linkage is usually implemented by what C++ programmers call name mangling. When the C++ compiler comes across a function declaration or definition, it mangles the function name, argument types, return values, and class name (if the function is a member of a class) into a single identifier. For instance, if you put the following piece of code through AT&T cfront:

```
short foo(int,float);
void foo2(float,float);
double foo3(int,int,int);
main() {
  int x = 0;  float f = 0.0;
  double d;  short y;
  y = foo(x,f);
  foo2(f,f);  d = foo3(x,x,x);
}
```

and then examine the output, you'll find these identifiers:

```
__1y = foo__Fif (__1x, __1f);
       foo__FfT1 (__1f);
__1d = foo3__FiN21 (__1x, __1x,
                    __1x);
```

The information at the end of the function name is scrambled to reduce the size of the identifier (remember the ANSI C maximum limit for identifier length is 31 characters). You can see, however, that there's a lot more information in the identifier than simply the function name.

When the linker looks for your function name, it has to find the mangled one, which insures that the correct function is linked (an error is generated if you've made a mistake). This eliminates a lot of frustrating mistakes. Notice that the variable identifiers are also modified.

What happens if you want to call plain C functions from within a C++ program? C++ 2.0 has an escape mechanism: you declare a function using a linkage specification. This is a string which describes the type of linkage you want.

Each implementation of C++ must support the specifications "C" and "C++," but this also provides a way to support other types of linkage, for example "Pascal." There are two ways you can specify alternate linkage (linkage is C++ by default):

```
extern "C" int pop(float);
```

or, for a group of functions:

```
extern "C" {
  int snap(double);
  int fizz(short);
}
```

Specifying C linkage means the identifiers won't get mangled, and the usual underscore will be prepended.

Explicit Support For Abstract Classes

We use abstract classes as "common interfaces" for other derived classes. For example, you'd never make a shape object; you'd instead make a specific circle, square, or line object, which all derive from class shape.

shape has a virtual method, draw(), which is redefined for each subclass. You can create a list of circle, square, and line objects, which are all shapes, and then go through the list of shapes and draw() each one. Each shape will figure out how to draw itself.

You don't want the user to ever be able to create a plain shape object. In C++ 2.0 you can explicitly prevent this by making the virtual functions "pure" using a pure-specifier. This means you simply set the body of the function to zero in the class definition, like so:

```
class shape {
  // ...
public:
  // ...
  virtual void draw() = 0;
};
```

Now, if the user ever tries to define a plain shape object, the compiler will generate an error message.

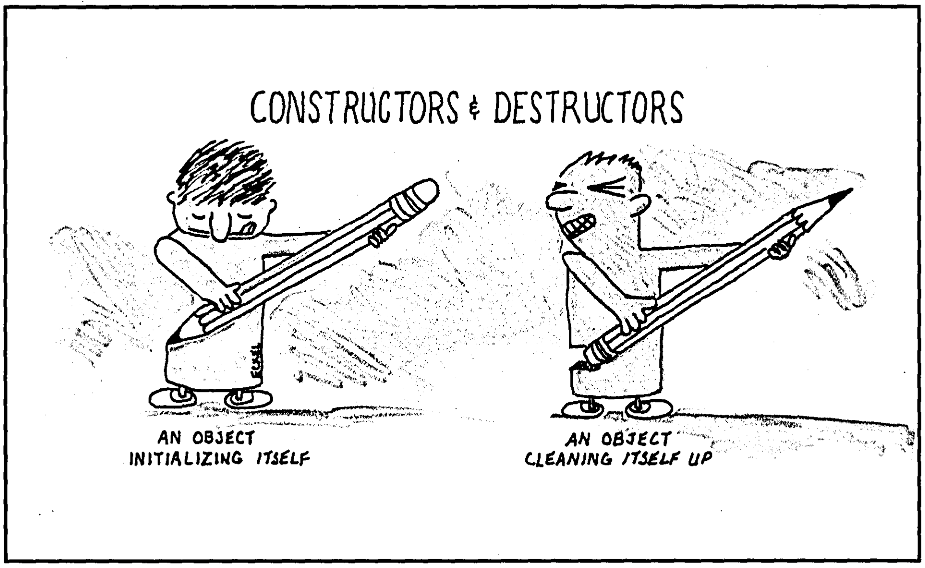
Automatic Copying & Initialization

Novice C++'ers often stumble over assignment, copying, and initialization. You don't realize at first that you're building new data types into the system when you define a class.

If you're going to pass objects of your new data type, by value, into and out of functions, the compiler must know how to perform this pass by value (passing addresses is easy). In C++ 1.2, the compiler would complain if you hadn't defined a copy-constructor X(X&) in the following situation:

```
class X { /* ... */ };
X nib(X arg) {
  return arg;
}
```

Here, arg must be copied into the function frame of nib as the argument, and copied out again as the return value. This means the compiler must know how to make an X from another X, which is what the X(X&) copy-constructor does.


CONSTRUCTORS & DESTRUCTORS
AN OBJECT INITIALIZING ITSELF
AN OBJECT CLEANING ITSELF UP

Similarly, whenever you assign:

```
X a,b;
// ...
a = b;
```

the compiler must have an X& operator=(X&) defined. C++ 1.2 printed confusing messages when you tried to pass or return arguments by value, or to assign, without defining the copy-constructor or operator=. This has been such a problem for novices that C++ 2.0 will automatically generate a copy-constructor and operator= for you if you don't.

If your class contains member objects which define copy-constructors and operator= functions, those are used by the automatically created functions. Most of the time, the automatically created functions do what you want; when they don't, you can define the functions yourself.

Overloading Operators new & delete

C programmers have always said that "C gives you enough rope to hang yourself." C++ allows you to change the rules, so you can come up with all kinds of new ways to do the deed. In particular, the meaning of operators can be "overloaded," or redefined, so they mean different things when used with different types of objects. This extends to the memory allocation operators new and delete.

new and delete are like ANSI C's malloc() and free() library functions—they allow you to create objects at run-time. They differ in that new calls the constructor, and delete calls the destructor.

This insures that you can have no uninitialized objects to cause errors. To prevent these errors, dynamic memory allocation has become part of the core of the language instead of library functions (as in C).

C++ 1.2 allowed you to overload the global meaning of new and delete, but this would affect the whole system. If you only wanted to change the way memory was allocated for a subset of classes, you had to revert to the abomination known as "assignment to **this**," creating all kinds of confusion. See Chapter 6 of *Using* C++ for more details.

C++ 2.0 eliminates the need for assignment to **this** by allowing you to overload new and delete on a class-by-class basis. Here's the (corrected) example from *Using* C++:

```
#include <stddef.h> // for size_t
class keep_track {
```

ZORTECH

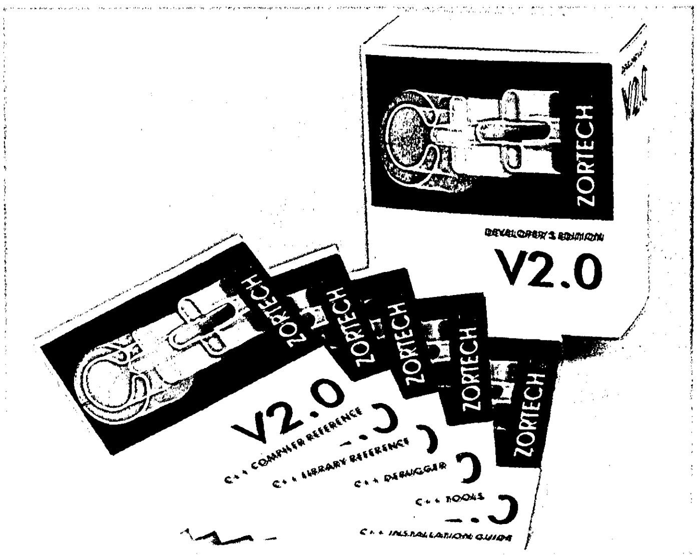
NEW! AT&T C++ RELEASE 2.0 SPECIFICATION

NEW! MS WINDOWS COMPATIBILITY

NEW! EASY PORTABILITY FROM MICROSOFT C

NEW! C++DEBUGGER & EXPANDED C++ TOOLS

NEW! OS/2 UPGRADE AVAILABLE NOW!



We listened carefully to what you wanted in a next generation MS DOS C++ compiler. The answer is Zortech C++ V2.0 Developer's Edition.

You wanted the latest AT&T V2.0 features with the power offered by multiple inheritance and type safe linkage, so here it is.

You wanted compatibility with MS WINDOWS, we added it.

You repeatedly asked for easier portability from Microsoft C, we got the message, and have written the library functions you need.

You wanted the world's first MS DOS C++ source level DEBUGGER, and now the wait is over.

You wanted expanded and improved documentation, we both listened and delivered.

You wanted to be able to upgrade to an OS/2 version compiler supporting Presentation Manager, you did not want it to cost a fortune, so it's available for $150.

You want to look at the standard library SOURCE CODE, so we are including it.

For many, EMS programming support, built into the compiler is important, so it's in there too.

You were happy using the 18 classes provided in C++ TOOLS, but we revised and expanded it anyway.

You never asked for a free TSR library to be included, but we knew you'd love to use our neat little package, so we included it free.

You liked our FLASH GRAPHICS package for its speed, but wanted a C++ Class interface, so we've written it.

SAVE $200

Get the Developer's Edition for only $450 comprising

C++ Compiler	($199.95)
C++ Debugger	($149.95)
C++ Tools	($149.95)
Library Source	($149.95)
Total Value	$649.80

Here is our list of highly recommended C++ books:

C++ Language/Stroustrup	$32.25
C++ Answer Book/Hansen	$26.95
C++ for C Programmers	$29.95
C++ Primer/Lippman	$30.25

Ask about our new C++ Video Tutorial

How To Order:

Already own Zortech C++? Call the order hotline for details of our low cost upgrades.

To order Zortech C++ for the first time, just call the order hotline. We accept payment by Mastercard/Visa/COD.

Alternatively, mail the coupon below with your check or credit card details.

*ZORTECH INC.,
1165 Massachusetts Avenue, Arlington, MA 02174, USA
Voice 617-646-6703
Fax 617-643-7969*

*ZORTECH LTD.,
106-108 Powis Street, London, SE18 6LU, ENGLAND.
Voice (44)-1-316-7777
Fax (44)-1-316-4138*

CALL 1-800-848-8408

Yes! Please rush me the following C++ V2.0 items:

*Name__________
Address__________
____
City______ State___ Zip____
Visa/MC# __________
Exp.Date_____ Tel_______*

- ❑ *DEVELOPER'S EDITION $450 (Save $200)*
- ❑ *C++ COMPILER $199.95*
- ❑ *C++ DEBUGGER $149.95*
- ❑ *C++ TOOLS $149.95*
- ❑ *LIBRARY SOURCE CODE $149.95*
- ❑ *COMPILER & LIBRARY SOURCE $299.95*

For US orders please add $5.05 shipping

- ❑ *OS/2 COMPILER UPGRADE $149.95*
- ❑ *C++ VIDEO COURSE $499.95*
- ❑ *C++ Language /Stroustrup $32.25*
- ❑ *C++ Answer Book/Hansen $26.95*
- ❑ *C++ for C Programmers/Phol $29.95*
- ❑ *C++ Primer/Lippman $30.25*

Overseas orders at international mail rates.

All MicroSoft trademarks are acknowledged.

```
  static int newcalls;
public:
  void * operator new(size_t sz) {
    newcalls++;
    return ::new unsigned char[sz];
  }
  void operator delete(void * dp) {
    newcalls--;
    ::delete(dp);
  }
};
```

The variable newcalls keeps track of how many objects have been created on the free store. Notice the use of the scope resolution operator :: to call the global versions of new and delete.

The only other advantage of assignment to **this** was it allowed you to place an object at a specific memory location. In C++ 2.0 you can use a placement specifier with operator new to perform the same function. To use the placement specifier, simply put the starting address in parentheses following new:

```
class toad { /* ... */ };
toad * ip = new(0xff) toad;
```

Here, however, you can't call delete to destroy the object, since you didn't go through the "normal" allocation scheme to create it. So, in this situation, you get to call the destructor explicitly:

```
ip->toad::~toad();
```

Zortech 2.06 doesn't support the placement specifier for operator new.

Additional Operator Overloading

C++ 2.0 allows you to overload two new operators: the comma operator, and the operator ->, sometimes called the "smart pointer."

The comma is used in two places: as a separator in function argument lists, and to indicate sequential evaluation of expressions. It's the latter use which you can overload. If you use the comma operator like this:

```
foo(1), A + B, C * D;
```

the function foo() will be called, A will be added to B, and C will be multiplied by D, in that order. The values produced by all but the expression C*D will be discarded, and C*D will be the value produced by the whole expression. All expressions in a comma-separated list except the last one are evaluated only for their side effects.

When you overload the comma operator, the expression preceding the comma must evaluate to an object (or a reference to an object) which has the overloaded operator comma defined. Overloading operator comma allows you to add an extra side effect when evaluating expression lists involving objects.

The term "smart pointer" is somewhat misleading, because it implies you have control over pointer dereferencing (which you don't). The overloaded operator->() can only be invoked on class objects or references to class objects, not pointers to class objects.

In addition, note that operator->() is unary, not binary. It must produce either a pointer or an object for which operator->() has been defined. For a complete example, see Dewhurst & Stark.

Pointers To Members

In plain C, a pointer to a variable allows you to select variables at run-time, and a pointer to a function allows you to select functions at run-time. In C++, a pointer to a class member allows you to select a member (data or function) of an object at run-time.

This is tricky because in plain C, variables and functions occupy specific addresses in memory, while a class (which contains the member identifiers from which you get the addresses for your pointer to member) is just a concept. Only objects have physical addresses, so a pointer to a member is actually an "offset" into the class. The physical address can only be resolved at run-time when the pointer is dereferenced for the physical object.

Pointers to members are particularly useful for creating functions which apply an operation (i.e., call a member function) to each object in a list. If, for instance, shapelist is a list of shape objects, and class shape has member functions called draw() and flop(), you could write a function called apply():

```
apply(shapelist, &shape::flop);
apply(shapelist, &shape::draw);
```

Pointers to members have been in the language specification prior to AT&T release 2.0, but they haven't been universally supported. Zortech C++ 2.06 doesn't support pointers to members (but will in a future release).

New Qualifiers For Member Functions

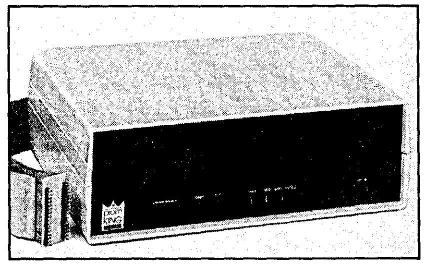
For greater programmer control over classes, C++ 2.0 introduces three new qualifiers for class member functions: const, volatile, and static. You use the const and volatile qualifiers similarly:

```
class zip {
  int i;
public:
  int f1() const { return i; }
  void f2() volatile { i++; }
};
```

Before C++ 2.0, you could declare an object const, but the compiler wouldn't complain if you then called a function which modified that object. In C++ 2.0, you can only call const member functions for const objects, and const member functions may not modify the data members of an object, or call non-const member functions. volatile objects and member functions have identical constraints.

The C++ compiler prevents const objects from being modified in the program environment (but they can be modified outside the program environment). The

STOMP OUT EPROM MADNESS



The PROM KING emulates EPROMS, saving both time and money during your development cycle. Programmable in seconds via your PC printer port or any computer RS232 port, it can emulate most 27xxx devices.

• 8K-8M bit devices	• 8-256 bit downloads
• High speed download:	• Easily expandable:
-Universal RS232	-4 EPROMS per unit
-PC printer port	-Up to 8 units
• Menu driven software	• Also programs like
• Battery backup	a real EPROM

$599 for 150nS units with 256K bits
Ask for pricing of other options

Made in USA by:

TRAXEL LABS INC.
BOX 239 • RONKONKOMA, NY • 11779
516•737•5147

Reader Service Number 178

compiler treats a volatile object as one which can be unpredictably changed by forces outside the program environment, so it makes no assumptions about the stability of the object during optimization. Thus, an object (and a member function) can be both const and volatile.

Zortech C++ 2.06 allows the use of const and volatile as member function modifiers, but doesn't support it.

Static Member Functions

You can declare a class data member to be static. Only one instance of that data member is created for all the objects in the class—all objects share the instance, and they can communicate through it.

C++ 2.0 adds static member functions, which operate on the whole class. static member functions have no **this**, so they can only perform general-purpose operations, which don't depend on specific class members or operations on static data members.

Before static member functions, you had to call the equivalent of a static member function using some arbitrary object (which is nonsensical to the reader, and requires the superfluous passing of the object's **this**) or create a friend function (which "pollutes" the global name space). Here's how you define and use static member functions:

```
class H2O {
  static float temperature;
public:
  // ...
  static void heat(float ht) {
    temperature += ht;
  }
};

main() {
  H2O::heat(10.1);
}
```

Notice you specify the class, rather than a particular object, when you call a static member function (although you can also call it in the normal way, associated with an object).

static data members are automatically initialized by the compiler to 0. Release 2.0 has added a way to initialize static data members explicitly (especially useful for member objects). The statements are placed outside of all functions (i.e., at file scope) like this:

```
float H2O::temperature = 80.0;
```

Notice you must completely specify the class membership *and* type of the object in the initializer. You can also use variables and function calls in the initializer.

New Forms Of Initialization

A lot of your coding time, especially with object-oriented programming, goes into setting up the objects—initializing them to the proper state. Since this can be tedious and confusing, it's an invitation to errors. C++ 2.0 allows many new ways to initialize variables. For instance, you can initialize a static object (any object with global lifetime) using any general expression, including a function call using some other static object.

One of my favorite features is aggregate initialization of objects. Consider the following class:

```
class Q {
  // ...
public:
  Q(int);
  Q(int,int);
};
```

Now you can make an array of these objects with almost no effort:

```
Q qa[] = {1, 2, 3, Q(1,1), Q(2,2)};
```

You can make qa global or auto. The best feature (which only Zortech 2.06 supports) allows you to execute expressions for aggregate initialization, so you can make an array of pointers to objects (all derived from the same base class) in one statement:

```
base * ba[] = {
  new derived1(1,1),
  new derived2(2,2,3,4),
  new derived3(0)
};
```

Future Features

C++ release 2.0 promises to be a stable standard for the foreseeable future. However, some important features are planned which won't require any changes to code written for C++ 2.0. Two in particular look promising: parameterized types and exception handling.

Suppose you're building a matrix

GRAPHICS!

Add lightning fast graphics to your programs quickly and easily through the popular PCX file format. Why reinvent the wheel? Make your programs immediately compatible with hundreds of packages from Aldus PageMaker to ZSoft's PC Paintbrush with these linkable graphic libraries.

"An exceptional product" - Programmer's Journal, Aug
NEW! Version 3.5 of the **PCX Programmer's Toolkit** gives you over 60 powerful functions to manipulate bitmapped graphics. Use Virtual screens, Super VGA modes, LIM 4.0 support, a 300 page manual, 9 utilities including screen capture and display, and the fastest routines on the market. **$195**

Need Special Effects, but caught in a GRASP?
Why create a demo when you can create the real thing? Don't be trapped in a slideshow editor or demo program when you can use **PCX Effects** for the PCX Toolkit and your favorite programming language. A Music Language and spectacular effects for exploding your graphics! **$99**

Blazing Graphics Text
With **PCX Text** you can display text with graphics as fast as it always should have been. Display characters, strings, fixed and proportional text, background transparency, and more. Includes a font editor, 85 fonts, and text utilities for blazing graphics bitmapped text. **$149**

All packages support 12 compilers for C, Pascal, Basic, Fortran, Assembly, and Clipper. All modes of the Hercules, CGA, EGA, VGA, and Super VGA adapters are supported, up through 800x600x256 (22 modes in all). Assembly Language source code is optionally available. Trademarks are property of their respective holders.

GENUS MICROPROGRAMMING
11315 Meadow Lake • Houston, Texas 77077 • (713) 870-0737

No Royalties! 30-day Money Back Guarantee. VISA/MC/AMEX/COD/PO accepted.

1-800-227-0918

STAGE#1 -Action Diagram Editor

STAGE#1 is a full screen editor that automates the generation and editing of action diagrams, the most versatile diagramming method available today. Action diagrams can be used to define software structure from the top-most overview down to the lowest level pseudo code. No other diagramming method has that capability.

More than 70 commands provide the familiar word processor functions, plus a few unique features. For example:

CONVERTS ACTION DIAGRAMS;

```
┌─ *Print_Character_Set
│
│  ╒═ For line Ø to 7
│  │     Put leading spaces in the output string
│  │     Put the line number and ': ' in the output string
│  │     Calculate offset (line * 32)
│  │
│  │  ┌─ If line is not Ø or 4
│  │  │
│  │  │  ╒═ For count equals Ø to 31
│  │  │  │     Character value is count plus offset
│  │  │  │     Add character and space to the output string
│  │  │  /* Endfor */
│  │
│  │  /* Else */
│  │     /* Add 'Control characters, can't print them!' to the */
│  │       /* output string */
│  │  /* Endifelse */
│  │
│  │  /* Display the output string */
│  │  /* Add CR and LF to the output string */
│  │  /* Print the output string */
│  /* Endfor */

   /* Print a formfeed */
```

INTO COMMENT LINES!

STAGE#1 also doubles as a normal text editor for entering program code. Direct access to DOS allows trial compilation or assembly without exiting the editor.

System requirements: IBM PC/XT/AT, PS/2 or compatible; PC or MS-DOS, 2.1 or higher; minimum of 256KB of memory; CGA, EGA, VGA or monochrome monitor; two disk drives; printer with either the full ASCII or the IBM graphics (line-draw) character set. A plotter is not required.

INTRODUCTORY PRICE $159.00

Price: $189.00 (US), includes shipping (UPS ground) within the continental US. Washington residents add 7.8% sales tax. Educational discounts and site licensing are available. NOT copy protected. Specify 3 1/2 or 5 1/4 inch disk.

Why settle for a program that will produce only flowcharts and organization charts when you can have much more?

SIMPLY POWERFUL

Mt. St. Helens Software
P.O. Box 3319
Pasco, Washington 99302-3319

For orders or information:
(509) 547-2582
8 AM - 5 PM (PST)

Reader Service Number 165

class, and you find that some of your problems need float matrix elements while others need double elements. In the future, users may require int matrices or complex matrices. You can't do a search-and-replace of all float types to some other type—what about variables that shouldn't be changed?

Parameterized types will let you create a generic matrix class (or any other class composed of other objects) and then create objects of that class utilizing specific types. The keyword template has been reserved in the language for this purpose. The syntax will look something like this for our matrix problem:

```
template<class T> class matrix {
  // class definition using T
};
matrix<float> fm;
matrix<double> dm;
matrix<int> im;
```

In his book, Stroustrup shows a technique for mimicking most aspects of parameterized types using #define.

Exception Handling

An exception is a catastrophic condition which the normal flow of a program can't handle. In a lot of plain C situations, this isn't a problem—you just bail out back to a known good state. The stack is shoved up, and you just forget about any variables defined in the frame where the exception occurred.

In C++, however, objects may have been defined in the frame. If those objects have destructors, the destructors must be called. Thus the implementation of exception handling involves keeping track of constructed objects and properly destroying them if there's an exception.

Fortunately, the syntax for exception handling is straightforward, and the programmer doesn't need to worry about how it's implemented. The keywords "try" and "catch" have been reserved for future use with exception handling. The syntax will look something like this:

```
extern exception out_of_range;
extern exception overflow;
// ...
try {
  foo X(1,20);
  buzz(X);
  // ...
}
catch(out_of_range) {
  // code executed for exception
}
catch(overflow) {
  // code executed for exception
}
```

The try-statement is always executed; if an exception occurs in the middle of the try-statement, the stack is unrolled (appropriate destructors are called) and the corresponding catch-clause is executed. You can have any number of catch-clauses. Note that exceptions are objects of a class defined by the C++ system.

It may be several years before we see these additions, but they'll be useful.

C++ 2.0 Compilers

I have two implementations of C++ release 2.0 for the PC: Zortech's C++ 2.06, and a beta release of Comeau Computing's port of AT&T's cfront (the same scheme used by Glockenspiel and Guidelines, neither of which yet have a 2.0 release or beta.)

It's important to realize that C++ is a difficult language to implement, and pointing out unsupported features is by no means a negative review. In fact, in the 2½ years I've worked with C++, I haven't come across a compiler which is even close to being "clean."

Zortech has done a commendable job with C++ 2.06, and I recommend it highly. Their documentation is vastly improved, and the developer's edition includes all the library source (including a method for ROMing the library code), as well as a debugger and a set of C++ classes.

I haven't encountered any of the "choke and die" problems which were present in earlier versions. The only problems I've discovered are unsupported features. The only difficulty I have with these is the way the compiler handles them: it should say "sorry, not implemented" (as cfront does) rather than claiming a syntax error.

The debugger is impressive! When my code gets more complex, the debugger doesn't always deliver the promised information (object member values, dereferencing NULL pointers), but it doesn't die. It does furnish data I'd have a hard time getting any other way.

Neither the compiler nor the debugger is perfect, but they both work well enough, and Zortech is perfecting them (there'll be an update by the time you read this). I'm pleased with the package. If you can afford it, get the developer's version—it's worth the extra money.

Because no implementation of C++ is perfect, I find it extremely useful to have cfront handy. I can test code for compatibility and my compilers for bugs. I've been using a beta of Comeau Computing's cfront 2.0 for the PC, and it seems to work fine—apparently AT&T worked hard to make 2.0 easier to port than 1.2 to strange processors like the 8086. cfront is $250, but worth it.

Keeping Everyone Honest

We'll be seeing lots of new C++ compilers in the coming years. It's easy for the compiler writers to miss some of the subtleties of the language, and hard for magazine reviewers and other evaluators (including end users) to find these problems in a short time. To help solve this problem, I'm putting together a test suite for C++ compilers (*not* a validation suite, although it may eventually become that).

Most test suites cost a fortune—this one will be freeware, that is, copyrighted but freely sharable. I copyright to maintain control over the sources and prevent them from being sold (other than by shareware dealers).

You can at any time get the current suite (which I will update as I encounter problems) by sending a check for $25 to Revolution2. If you have a piece of code which breaks a C++ implementation and you want to add it to the suite, please send it to me on disk or to one of my electronic addresses.

Also, Abraxas Software specializes in portability, especially C portability. One of their products, CodeCheck, will analyze your C code and point out places where you will have portability problems. Abraxas has recently added the ability to analyze C++.

Wrap Up

There are oodles of conferences and workshops coming up: OOPSLA, which will be combined with ECOOP (the European version) and held in Canada; SCOOP (Seminars & Conference in OOP, June 4-6, 1990, in Boston), put on by the same folks at SIGS publications (they publish *JOOP*, the *Journal of OOP*, *HOOT*, *Hot-Line of OO Technology*, *The C++ Report*, and the *Journal of Pascal, ADA & Modula-2*; the Miller-Freeman Software Development Conference (SD90) February 6-9; and the venerable Usenix C++ Conference, April 9-11 in San Francisco.

Several new books also cover the 2.0 specification:

- *C++ Primer*, by Stanley Lippman (Addison-Wesley, 1989).
- *Programming in C++*, by Stephen Dewhurst and Kathy Stark (Prentice-Hall, 1989).
- *Using* C++, by Bruce Eckel (Osborne/McGraw-Hill, 1989).
- *The C++ Answer Book* by Tony Hansen (Addison-Wesley, 1990).

For those of you into numerics, Tom Keffer and Rick Romea, collectively known as Rogue Wave Associates, have a C++ matrix package that will compile with Zortech C++ and on UNIX machines (it *should* also work with Glockenspiel C++). The package includes vectors and matrices of all kinds, and a very easy to use Fast-Fourier Transform.

I've seen it, and it's shipping now for only $99. Tom's a physicist/oceanographer and Rick's a mathematician, so these guys know their numbers. I worked on a project with Tom while I was learning C++ so I know firsthand that he's an excellent programmer. Looks like the bottom line in matrix packages.

Bruce Eckel is the author of Using C++ *(Osborne/McGraw-Hill 1989) and the owner of Revolution2, a C++ consulting and embedded-systems development firm. You can reach him on BIX as beckel, compuserve as 72070,3256, or the internet as 72070.3256@compuserve.com.*

Products Mentioned

CodeCheck
Abraxas Software, Inc.
7033 S.W. MacAdam Ave.
P.O. Box 19586
Portland, OR 97219
(503) 244-5253

Comeau Computing C++
91-34 120th Street
Richmond Hill, NY 11418
(718) 849-2355

Vector, Matrix & FFT Classes
Rogue Wave Associates
P.O. Box 85341
Seattle, WA 98195
(206) 523-5831

Zortech C++ Developer's Edition
Zortech Inc.
1165 Massachusetts Ave.
Arlington, MA 02174
1-800-848-8408 (617) 646-6703

♦ ♦ ♦

Training A Neural Network:

Build An NNT In C, Part 2

In Part 1 (Micro C, *#51), Russ and Roy led us inside a neural network detailing the theory, algorithms, and C code for programming a complete neural network (or NNT as they call it).*

This issue they'll show us how to train (i.e., teach) a network using a simple yet sophisticated technique called, "backward error propagation." Oxymoron?

For those just joining, check out last issue's article or download the complete C source for their NNT from the Micro C BBS (503) 382-7643. Note: we've excerpted Parts 1 and 2 of this series from Russ and Roy's book on neural nets which Academic Press will publish this year.

We train a network by combining: (1) feedforward output state, and (2) weight adjustment calculations.

This training technique, called "backward error propagation," is the key to the success of a back-propagation NNT. Central to this technique is how we define network error. In other words, what should we use to measure a network's performance relative to the training sets?

Rumelhart and McClelland (see reference) define an error term that depends on the difference between the output value an output PE (or neuron) is supposed to have (called the target value t_l) and the value the PE actually has as a result of the feedforward calculations o_l. We define an error term for a specific pattern and sum it over all the output nodes which that pattern affects.

Equation 5 defines error. The subscript "p" denotes that the value is for a specific pattern. Note that when we calculate error in the BP training algorithm, we do it on a node-by-node basis over the entire set (or epoch) of patterns, rather than on a pattern-by-pattern basis. Then we sum the error over all nodes, giving us a grand total for all nodes and all patterns.

In addition, after we get the grand total error, we divide it by the number of patterns to obtain an average sum-squared error value. Since the number of patterns will vary among training sets, we need an average error to compare sets.

Look over Equation 5, the term for calculating error (note: since the factor 0.5 is constant, we usually delete it from our calculations):

$$E_p = 0.5 \sum_{l=1}^{n_l} (t_{pl} - o_{pl})^2 \qquad \text{Equation 5}$$

Training the network in effect minimizes this average sum-squared error over a training set of patterns.

We won't rigorously derive any training algorithms in this article. If you're rigorous-minded (i.e., interested in these kinds of details), check out Chapter 8 of Rumelhart and McClelland. Even their derivation lacks rigor, but reviewing it should give you an insight into where these equations come from, and should help make you more comfortable using them.

Remember from Equation 4 (last issue) that the output of a PE in the output layer is a function of its input, or, $o_l=f(i_l)$. The first derivative of the function, $f'(i_l)$, is important in error back-propagation. We represent the error signal by δ_l for output layer PEs, and define it in Equation 6:

$$\delta_l = f'(i_l)\,(t_l - o_l) \qquad \text{Equation 6}$$

For the sigmoid activation function of Equation 4, the first derivative is $o_l(1-o_l)$. So we end up with the expression for the output layer error signal, calculated for each output PE (see Equation 7).

$$\delta_l = (t_l - o_l)\,o_l\,(1 - o_l) \qquad \text{Equation 7}$$

We now propagate this error value back and perform appropriate weight adjustments, using either of two methods:

(1) Propagate the error back and adjust weights after the network "sees" each training pattern. This is online, or single pattern training.

(2) Accumulate the δs for each PE for the entire training set, add them, and propagate back the error based upon the grand total δ. This is batch, or epoch training.

We use batch training in our NNT. Rumelhart and McClelland assumed that weight changes occur only after a complete cycle of pattern presentations. They point out that it's all right to calculate weight changes after each pattern (online training) as long as the learning rate η is sufficiently small. It does, however, add significant computational overhead. Since we want to speed up training, we batch.

Before we can update weights, however, we must have something to update. We initialize each weight to some value. We can't just start with all the weights equal to zero (or all equal to any single number, for that matter) or the network won't be trainable. You can see why if you look at the equations for weight updating (see Equations 8, 9, etc., below).

It's typical to initialize the weights to random numbers between -0.3 and 0.3. Why pick -0.3 and 0.3 as the bounds? Simply put, "it works." And since many BP NNTs train fast with these bounds, we often start training at -0.3 and 0.3.

Living In A Bumpy Bowl

Let's look at how we use δ_l to update weights that feed the output layer, w_{lj}. To a first approximation, Equation 8 de-

By Russ Eberhart and Roy Dobbins
6770 Halfcrown Court
Columbia, MD 21044

scribes the updating of these weights. Here, η is the learning coefficient. We assign it values between 0 and 1.

$$w_{lj}\,(new) = w_{lj}\,(old) + \eta\ \delta_l\, o_j \qquad \text{Equation 8}$$

This kind of weight updating sometimes has a problem—the system gets caught in what we (and others) call local energy minima. Imagine a bowl-shaped surface containing a lot of little bumps and ridges.

The error minimization process is analogous to minimizing the energy of our position in the bumpy, ridgelined bowl. Ideally, we'd like to move our position (perhaps marked by a very small ball bearing) to the bottom of the bowl, where the energy is minimum. This position is called the globally optimal solution.

Depending on how much or how little we can move the ball bearing at one time, however, we might get caught in some little depression or ridge. This situation is most likely with small limits on each individual movement, which corresponds to small values of η.

We can help the situation by taking into account the "momentum" of our ball bearing. That is, we consider its previous movement as reflected in its previous weight change. We multiply this previous weight change by a momentum factor that we label α.

The momentum factor α can take on values between 0 and 1. Equation 9, which is just Equation 8 with the momentum term added, becomes the equation we'll use in our BP NNT to update the weights feeding the output layer.

$$w_{lj}\,(new) = w_{lj}\,(old) + \eta\ \delta_l\, o_j + \alpha\ [\Delta\, w_{lj}\,(old)] \qquad \text{Equation 9}$$

Watch out! We've just thrown another delta at you. This one, $\Delta w_{lj}(\text{old})$, stands for the previous change in the weight.

We train a network by combining: feed-forward output state and weight adjustment calculations.

Put another way (in words) the new weight is equal to the old weight plus the change in weight. The change in weight consists of the δ error signal term and the α momentum factor term.

The momentum term is the product of the momentum factor α and the previous (old) change in the weight. By taking into account the previous movement of the weight, we're imparting momentum to our ball bearing, and are much more likely to reach the globally optimum solution for the network.

Keep in mind that we have PEs called bias nodes, or bias PEs, as indicated by the PEs with the letter "b" in them in Figure 1. These PEs always have an output, or activation value, of 1.

They serve as threshold units for the layers to which they're connected. The weights from the bias PEs to each of the PEs in Equation 9 are adjusted exactly like the other weights. In Equation 9, for each of the output PEs, the subscript j takes on values from 0 (the bias PE) to n_j (the number of hidden PEs).

Now that we have the new values for the weights feeding the output PEs, we turn to the hidden PEs. What's the error term for these units? It isn't as simple to figure this out as it was for the output units. There we could intuitively reason that the error should be some function of the difference between the desired output t_l and the actual output o_l.

We don't really have any idea what the value for a hidden PE should be. Again we refer to the derivation by Rumelhart and McClelland. They showed that the error term for a hidden PE is given by Equation 10:

$$\delta_h = f'(i_h) \sum_{l=0}^{n_l} w_{lh}\, \delta_l \qquad \text{Equation 10}$$

As was the case in the output layer, the output of a PE in the hidden layer is a function of its input, or $o_h=f(i_h)$. The left part of the right-hand term in Equation 10 is the first derivative of this function. Again, we're using the sigmoid transfer function, so this derivative is $o_h(1-o_h)$, resulting in the hidden PE error term defined by Equation 11.

$$\delta_h = o_h\,(1 - o_h) \sum_{l=0}^{n_l} w_{lh}\, \delta_l \qquad \text{Equation 11}$$

The weight changes for the connection weights feeding the hidden layer from the input layer are now calculated in a manner analogous to those feeding the output layer. Equation 12 defines the weight changes:

$$w_{ji}\,(new) = w_{ji}\,(old) + \eta\ \delta_j\, o_i + \alpha\ [\Delta\, w_{ji}\,(old)] \qquad \text{Equation 12}$$

As before, we include bias nodes in the calculations. For each hidden node, the subscript i takes on values of 0, representing the bias node, to n_j, the number of input nodes.

We now have all the equations we need to implement the backpropagation of errors and adjustment of weights for both of the groups of weights. Equations 7, 9, 11, and 12 are the ones we need.

We first calculate the error terms expressed by Equation 7 for each output PE, then by Equation 11 for each hidden PE, for each pattern in the training set. We accumulate the error terms by summation. Then after the network has seen all the training patterns (the epoch) once, we calculate the weight adjustments expressed by Equations 9 and 12.

Keep a few things in mind. One—since we're updating by batch (or epoch), the δs we express in Equations 9 and 12 are grand totals (for each PE) for the entire training set. Whereas the δs in Equations 7 and 11 are calculated pattern by pattern and summed after one epoch.

Another thing—while values for η and α can be assigned on a layer-by-layer basis, or even a PE-by-PE basis, there's typically only one value selected for each in a given BP NNT implementation. These values are often adjusted, changed, and otherwise fiddled with, in the process of getting a network to train successfully; but once chosen, they are usually left alone.

A third—when calculating the δs for the hidden layer in Equation 11, the old (existing) weights (rather than new ones you might have calculated from Equation 9) from the hidden to the output layer are used. This is a potential problem only if you decide to update the weights after the net sees each training pattern. If you use epoch training, the weights aren't updated until after the net sees all patterns, so you don't have to worry.

Experiment

What values do you pick for η and α? The best choices depend on your application. Rumelhart and McClelland recommend values of 0.5 and 0.9, as a place to start. We've found, however, that for our applications, these values often seem to throw the network into oscillation, or put the PEs into saturation. (Saturation often causes all the outputs in the training and test sets to be about the same value, typically near zero or near one.)

For an electroencephalogram (EEG) spike detection NNT we developed, we often have good results with values of η=0.15 and α=0.075. Other times we've used values of 0.04 and 0.02.

Another parameter you'll have to experiment with is the number of iterations of the training patterns needed to give you an acceptable average mean-squared error value. First, pick a reasonable average mean squared error value, perhaps 0.04 or 0.05. If you can't train in 1,000 or 2,000 iterations, you probably should adjust η or α. You might also need to normalize your inputs differently.

You'll no doubt have to experiment. Just don't be disappointed when your NNT doesn't train successfully the first time. It almost certainly won't!

Editor's note: you'll find the C and object code for the neural net on the Micro C BBS (503 382-7643) and on the Issue #52 disk. The disk is $6 ppd. ($8 foreign) from Micro Cornucopia, P.O. Box 223, Bend, Oregon 97709. (800) 888-8087, VISA/MC.

Key Reference

D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Vol. 1: Foundations.* MIT Press, Cambridge, Mass., 1986.

♦ ♦ ♦

TechnoJock's Turbo Toolkit

Attention Turbo Pascal programmers........

Cut your program development time in half and create sophisticated user interfaces in minutes with *TechnoJock's Turbo Toolkit!*

The *Toolkit* contains twelve units with all source code included, and a 300 page manual full of examples. These range from a comprehensive unit that provides a Sidekick Plus style directory window with all the expected sorting features, to a basic procedure that writes text centered on the screen.

And it's easy to use! If you've bought other toolkits but didn't use them because of all their complicated rules ... TTT is for you.

TechnoJock
SOFTWARE

PO Box 820927
Houston, TX 77282-0927

- ultra fast screen writing
- EGA/VGA extended lines
- exploding boxes
- virtual screens
- sorted directory listings
- form input
- pulldown menus
- nested menus
- string manipulation
- windows
- full mouse support
- keyboard control
- date manipulation

Only $49.95!

Plus $5.00 shipping and handling (Canada $10.00, overseas $15.00). Texas residents add 8% Sales Tax.

Send checks to TechnoJock Software.
Send Visa/Mastercard details to:
Public (Software) Library
PO Box 35705
Houston TX 77232-5705

"TechnoJock's Turbo Toolkit may just cut your work in half! It provides easy-to-implement procedures .... designed for all Turbo Pascal programmers. With this Toolkit everyone becomes a TechnoJock."
Turbo Users Group

"Here, despite the whimsical name, is shareware at its best: an innovative product, a superb manual, quick mail-order service, and a reasonable price ... If you're programming in Turbo Pascal, you absolutely *must* have TTT!"
Public-Domain Software & Shareware
2nd edition

"Looks great! We have decided to implement your Toolkit in in-house programs for Tech Support."
Rich Kubota
Technical Support
Borland International

Orders: 1-800-242-4775
Enquiries: (713) 493-6354

By Walter Bright

Zortech Inc.
1165 Massachusetts Ave.
Arlington, MA 02174

Debugging C Pointers Using MEM

For Anyone Struggling With Dynamic Memory Allocation

You know how easy it is to write good clean C, right? You know the joys of dynamic memory allocation; it's right up with self-modifying code, right?

The C programming language is powerful, but with that power comes the scourge of the *pointer bug*. Normally confident and capable programmers are known to cower at the mere mention of the beast. Symptoms of pointer bugs include: hung machines, scrambled disks, failures that occur once-in-10,000 iterations, irreproducible results, and male pattern baldness.

I'll present a technique I've used and refined in my own code for several years. This process has reduced my code's pointer bugs by as much as 75%. It's simple to work into existing programs and adds little or no overhead. Sound too good to be true? Of course it does.

Philosophy

Programs are getting larger and more sophisticated (and occasionally better), hence more difficult to debug. You can significantly increase your ability to test and debug programs by inserting code which checks for out-of-range and so-called "impossible" values. The MEM package I'll describe here adds as much checking as possible for testing and debugging a program's use of free store.

Two schools of thought define what should happen when a program finds bugs in itself: (1) the program should terminate immediately, displaying a message about the problem; (2) the program should ignore or try to repair the damage and continue.

I've never quite understood the adherents to the second philosophy. What's the point in continuing if the program has already failed? The program must stop immediately for the following reasons:

- Stopping it immediately might avoid rebooting the machine.
- Stopping it as close as possible to the bug makes the bug easier to find.
- A runaway program may scramble the hard disk.
- If the software is running mission or life critical software, it may go on and do something terrible. (Such critical applications should have a human backup, another computer running similar software in parallel, or have the ability to reset and restart themselves. The self-detected fault should alert a human, tell the other computer to take charge, and/or reset the system.)

Errors like disk full, out of memory, etc., are not program bugs, unless the program fails to deal with these exceptions.

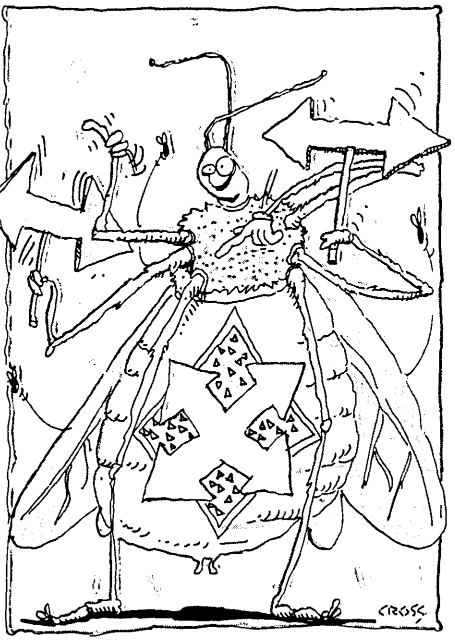
MEM, of course, subscribes to the first philosophy. It aborts the application with a message if it detects any faults directly. And it will (we hope) cause the application to fail quickly on faults that it doesn't detect directly.

Using MEM

Modify the header file toolkit.h for the specifics of your C compiler and operating system (see Figure 1). The one in the listing is set for Zortech C and MS-DOS.

The MEM functions exactly parallel the storage allocator functions. #include "mem.h" in all source files (see Figure 2). Then do a global search and replace, replacing all functions:

```
malloc() -> mem_malloc()
calloc() -> mem_calloc()
realloc() -> mem_realloc()
free() -> mem_free()
strdup() -> mem_strdup()
```



Symptoms of pointer bugs include: hung machines, scrambled disks, failures that occur once-in-10,000 iterations, irreproducible results, and male pattern baldness.

At the beginning of main(), add a call to mem_init(). At the points where the program returns to the operating system, add a call to mem_term(). Add mem.c to the list of files to be compiled and linked in (see Figure 3).

MEM has two modes of operation: debugging-on and debugging-off. Use debugging-on during program development. Use debugging-off for the production compile (this will eliminate most of the MEM overhead). You control the debugging status by predefining the macro MEM_DEBUG to turn debugging on; the default is off. If debugging is off, the MEM functions become trivial shells around the malloc, free, etc., functions.

What MEM Does

- ANSI-fication of free store:

Many older C library implementations don't follow the ANSI C standard on how malloc/realloc/free should work. For instance, in many libraries realloc doesn't behave correctly when passed a 0 for the new size or a NULL for the pointer. You can adjust the corresponding MEM function to account for this, and still present ANSI behavior to the caller. That way you don't need to rewrite the library routines, or program to account for the lowest common denominator.

- Logging of all allocations and frees:

All MEM's allocation and free functions pass the arguments _ _FILE_ _ and _ _LINE_ _. During allocation, MEM makes an entry in a linked list for that allocation and stores the file and line information in that list. Thus, every outstanding allocation is represented in that list.

This list forms the basis of much of what MEM does. If MEM detects a bug, you can usually print the file and line number where the pointer was allocated, which greatly facilitates tracking down the critter.

- Verification of frees:

Since MEM knows about all allocations, when a pointer is freed, MEM can check to make sure it's an outstanding allocation. MEM will only allow a pointer to be freed once.

```
p = mem_malloc(5);
mem_free(p);
mem_free(p);  /* this will cause an
      assertion failure from mem */
```

The freed data is overwritten with a

Figure 1—toolkit.h

```
#ifndef TOOLKIT_H
#define TOOLKIT_H

/* Define stuff that's different between machines.
   PROTOTYPING          1 if compiler supports prototyping
   HOSTBYTESWAPPED      1 if on the host machine the bytes are
           swapped (1 for 6809, 68000, 0 for 8088 and VAX). */
#if __ZTC__
#define PROTOTYPING     1
#define HOSTBYTESWAPPED 0
#define EXIT_SUCCESS    0
#define EXIT_FAILURE    1
#define BITSPERBYTE 8
#define SIZEOFINT   sizeof(int)
#define SIZEOFLONG  sizeof(long)

#else
#include        "host.h"
#endif

/* Static definitions do not appear in the linker .MAP file. Override
   the definition here to make them global if necessary. */
#ifndef STATIC
#define STATIC  static
#endif

#define arraysize(array)        (sizeof(array) / sizeof(array[0]))

/* Macros so that we can do prototyping, but still work with non-
   prototyping compilers: */
#if PROTOTYPING
#if MSC
#define _0
#else
#define _0                      void
#endif
#define _1(v)                   v
#define _2(v1,v2)               v1,v2
#define _3(v1,v2,v3)            v1,v2,v3
#define _4(v1,v2,v3,v4)         v1,v2,v3,v4
#define _5(v1,v2,v3,v4,v5)      v1,v2,v3,v4,v5
#define _6(v1,v2,v3,v4,v5,v6)   v1,v2,v3,v4,v5,v6
#define _7(v1,v2,v3,v4,v5,v6,v7) v1,v2,v3,v4,v5,v6,v7
#define _8(v1,v2,v3,v4,v5,v6,v7,v8) v1,v2,v3,v4,v5,v6,v7,v8
#define _9(v1,v2,v3,v4,v5,v6,v7,v8,v9) v1,v2,v3,v4,v5,v6,v7,v8,v9
#define _10(v1,v2,v3,v4,v5,v6,v7,v8,v9,v10) v1,v2,v3,v4,v5,v6,v7,v8,\
                                          v9,v10
#else
#define _0
#define _1(v)
#define _2(v1,v2)
#define _3(v1,v2,v3)
#define _4(v1,v2,v3,v4)
#define _5(v1,v2,v3,v4,v5)
#define _6(v1,v2,v3,v4,v5,v6)
#define _7(v1,v2,v3,v4,v5,v6,v7)
#define _8(v1,v2,v3,v4,v5,v6,v7,v8)
#define _9(v1,v2,v3,v4,v5,v6,v7,v8,v9)
#define _10(v1,v2,v3,v4,v5,v6,v7,v8,v9,v10)
#endif

#if PROTOTYPING
#define P(s)    s
#else
#define P(s)    ()
#endif

#ifdef DEBUG
#define debug(a)        (a)
#else
#define debug(a)
#endif

#endif /* TOOLKIT_H */
```

♦ ♦ ♦

C CODE FOR THE PC

source code, of course

	MS-DOS File Compatibility Package (read & write MS-DOS file systems on non-MS-DOS computers)	$750
	CSource Application Program Generator by MBAC (includes all source code; generator & libraries)	$500
	dB2c (dBase-to-C translator; includes db_Files for C and db_Tools for C)	$325
	CQL Query System (SQL retrievals on B-trees plus windows)	$325
	GraphiC 5.0 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	Drasch Clisp with Crules (Lisp library and programming environment with rule processing capability; natural language example)	$300
	PC Curses (Aspen, Software, System V compatible, extensive documentation)	$290
	C-Data Manager (object-oriented data management, persistent objects from runtime definitions, network and entity models)	$250
	MEWEL (extensible window and event library by Magma Software; message-passing & object-oriented; SAA-compatible; dialog editor)	$250
	TurboTEX (Release 2.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont)	$250
	PC PostScript (complete PostScript interpreter (ROM Version 47.0A), 80286/386 only, many device drivers, optimized graphics, fast)	$250
	db_File & db_Retrieve by Raima (B-tree and network database with SQL query and report writer; multi-user $475)	$245
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF; specify compiler)	$225
	CDirect (multi-user hashed file manager; variable length fields, binary or ASCII data, alternate keys)	$210
NEW!	SilverComm (complete asynchronous communications library)	$210
	Installation Toolkit 3.0 (interactive script development program; many special cases covered)	$195
	QuickGeometry Library (large collection of mathematics, graphics, display & DXF subroutines for CAD/CAM/CAE/CNC)	$170
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$165
	TurboGeometry (library of routines for computational geometry, Version 3.0)	$160
	AT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for ATs)	$160
	WKS Library Version 2.01 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	OS/88 (industrial-strength U**x-like operating system, many tools, cross-development from MS-DOS)	$150
	Cephes Mathematical Library (over 100 high-quality, double-precision scientific functions)	$150
	ME Version 2.1 (programmer's editor with C-like macro language by Magma Software; Version 1.31 still $75)	$140
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
NEW!	Rogue Wave Vector & Matrix Classes (inc. C++ overloadings for standard operators, matrix inversion & FFT; Zortech or GNU C++)	$125
	Power Search by Blaise Computing (regular-expression compiler; generates machine code on the fly)	$120
	Install 2.3 (automatic installation program; user-selected partial installation; CRC checking)	$120
NEW!	B-Strings (dynamic string handling; cut, copy, paste, search, user input, etc.; non-fragmenting memory management)	$105
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$105
	Minix Operating System (Version 1.3; U**x-like operating system, includes manual)	$105
Updated!	PC/IP (CMU/MIT TCP/IP for PCs; Ethernet, Appletalk & NETBIOS drivers, RVD, gateways)	$100
	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
	The Profiler (program execution profile tool)	$100
	QC88 C compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
Cheaper!	Booter Toolkit (floppy disk bootstrap routines, DOS file system, light-weight multitasking, windows, fast memory management)	$85
	Otter 1.0 (beautiful theorem-prover by Bill McCune; includes manual & two books by Wos; complete starter kit)	$80
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
NEW!	HY-PHEN-EX (a hyphenator for American English with over 4,800 rules)	$75
	Make (macros, all languages, built-in rules)	$75
	eval() (C function to evaluate ASCII infix expression string; 17 built-in functions)	$75
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
	Professional C Windows (lean & mean window and keyboard handler)	$70
	Heap Expander (virtual memory manager using expanded memory, extended memory, and disk space)	$65
	Quincy (interactive C interpreter)	$60
	Symtab/Ptree (general-purpose symbol table/parse tree construction and management package; specify Symtab or Ptree)	$60
	Coder's Prolog (Version 3.0; inference engine for use with C programs)	$60
	Async-Termio (Unix V compatible serial interface for MS-DOS; stty, ioctl, SIGINT, etc.)	$55
	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
NEW!	Floppy TAR (TAR backup and restore on MS-DOS devices; direct access to non-standard devices)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	REGX Plus (search and replace string manipulation routines based on regular expressions)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	Multi-User BBS (chat, mail, menus, sysop displays; does not include Hayes modem driver)	$50
NEW!	LaplaceB (LaPlace polynomials, real and complex)	$50
	CLIPS (rule-based expert system generator, Version 4.3; advanced manuals available)	$50
	Kier DateLib (all kinds of date manipulation; translation, validation, formatting, & arithmetic)	$45
	Fortran-to-C Translator by Polyglot (Fortran-IV-like Fortran to ugly C; plan to adapt to your own flavor of Fortran)	$40
	DES Encryption & Decryption (2500 bits/second on 4.77 MHz PC for on-the-fly encryption at 2400 baud)	$40
	FlexList (doubly-linked lists of arbitrary data with multiple access methods)	$40
	Virtual Memory Manager by Blake McBride (LRU pager, dynamic swap file, image save/restore)	$40
	Heap I/O (treat all or part of a disk file as heap storage)	$40
Updated!	Bison & BYACC (YACC workalike parser generators; documentation; no restrictions on use of BYACC output)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
	RXC & EGREP (Regular Expression Compiler and Pattern Matching; RXC makes finite state machine from regular expression)	$35
	CCALC (handy extended-precision calculator; real and complex models; many built-in functions)	$30
	GNU Awk & Diff for PC (both programs in one package)	$30
Updated!	6-Pack of Editors (seven public domain editors for use, study & hacking; includes microEmacs 3.10 & Stevie, a vi clone)	$30
	Crunch Pack (14 file compression & expansion programs)	$30
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Wirth standard Pascal)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX; official BSD Version 2.1 with docs)	$25
	List-Pac (C functions for lists, stacks, and queues)	$25
NEW!	Using C++ Library (the code from the book by Bruce Eckel and then some; Zortech 2.0 compatible)	$25
	A68 (68000 cross-assembler)	$20
	XLT Macro Processor (general purpose text translator)	$20
	Data	
NEW!	Moby Pronunciator (150,000 words & phrases encoded with full IPA pronunciation & emphasis points; 900 distinguished by part-of-speech)	$160
NEW!	Moby Part-of-Speech (200,000 words and phrases described by prioritized part(s)-of-speech)	$120
NEW!	Moby Hyphenator (150,000 words fully hyphenated/syllabified)	$105
	Moby Words (500,000 words & phrases, 9,000 stars, 15,000 names)	$65
	Smithsonian Astronomical Observatory Subset (right ascension, declination, & magnitude of 258,997 stars)	$60
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
Updated!	Interactive Computer Ephemeris (high-precision moon, sun, planet & star positions; USNO (no source) & Downey 4.8 (C source))	$30
	U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

info@acw.com

Voice: (512) 258-0785
BBS: (512) 258-8831
FAX: (512) 258-1342

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

Reader Service Number 4

nonzero known value. This flushes out problems, such as continuing to refer to data after it's been freed. The value written over the data is selected to maximize the probability of a segment fault or assertion failure if the application continues to refer to it. MEM obviously can't directly detect "if" instances like:

```
mem_free(p);
if (*p) ...
```

But by guaranteeing that p points to garbage after mem_free() returns, we hope code like the above will *never* work, and thus be easy to find.

- Detection of pointer over and underrun:

Pointer overrun occurs when a program stores data past the end of a buffer. A common programming error does this:

```
p = malloc(strlen(s)); /* didn't
 allocate space for terminating 0*/
strcpy(p,s); /* overrun: storing 0
                 past end of p */
```

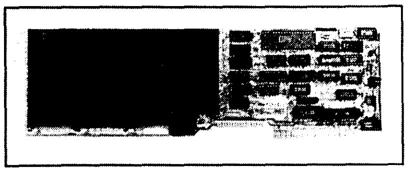
Pointer underrun occurs when a program stores data before the beginning. This error occurs less often. MEM detects

Figure 2—mem.h

```
/* Copyright 1986-1988 by Northwest Software      */
/* All Rights Reserved - Written by Walter Bright */

#ifndef MEM_H
#define MEM_H   1

#ifndef TOOLKIT_H
#include        "toolkit.h"
#endif

/* Memory management routines.
Compiling:
  #define MEM_DEBUG 1 when compiling to enable extended debugging
  features.

  Features always enabled:
    o mem_init() is called at startup, and mem_term() at close, which
      checks to see that number of allocs is same as number of frees.
    o Behavior on out-of-memory conditions can be controlled via
      mem_setexception().

  Extended debugging features:
    o Enabled by #define MEM_DEBUG 1 when compiling.
    o Check values inserted before and after alloc'ed data to detect
      pointer underruns and overruns.
    o Free'd pointers are checked against alloc'ed pointers.
    o Free'd storage cleared to smoke out references to free'd data.
    o Realloc'd pointers are always changed, and previous storage is
      cleared, to detect erroneous dependencies on previous pointer.
    o mem_checkptr() is provided to check an alloc'ed pointer.
 */

/********************* GLOBAL VARIABLES *************************/
extern int mem_inited;/* != 0 if mem package is initialized. Test this
     if you have other packages that depend on mem being intialized */

/********************* PUBLIC FUNCTIONS *************************/
/* Set behavior when mem runs out of memory.
Input:
  flag =
    MEM_ABORTMSG: Abort program with the message 'Fatal error: out of
      memory' sent to stdout. This is the default behavior.
    MEM_ABORT: Abort the program with no message.
    MEM_RETNULL: Return NULL back to caller.
    MEM_CALLFP: Call application-specified func. fp must be supplied.
      fp Optional function pointer. Supplied if (flag == MEM_CALLFP).
         This function returns MEM_XXXXX, indicating what mem should
         do next. The function could do things like swap data out to
         disk to free up more memory.
      fp could also return:
         MEM_RETRY: Try again to allocate the space. Be careful not to
                    go into an infinite loop. */

#if __ZTC__
enum MEM_E {MEM_ABORTMSG,MEM_ABORT,MEM_RETNULL,MEM_CALLFP,MEM_RETRY};
void mem_setexception(_2(enum MEM_E,...));
#else
#define MEM_ABORTMSG    0
#define MEM_ABORT       1
#define MEM_RETNULL     2
#define MEM_CALLFP      3
#define MEM_RETRY       4
void mem_setexception(_2(int,...));
#endif

/* Allocate space for string, copy string into it, and return pointer
   to the new string. This routine doesn't really belong here, but it
   is used so often that I gave up and put it here.
   Use: char *mem_strdup(const char *s);
   Returns: pointer to copied string if successful.
            else returns NULL (if MEM_RETNULL)*/
char *mem_strdup(_1(const char *));

/* Function so we can have a pointer to function mem_free(). This is
   needed since mem_free is sometimes defined as a macro, and then the
   preprocessor screws up. Pointer to mem_free() used frequently with
   the list package.
   Use: void mem_freefp(void *p);
*/
```

Continued on page 26

You've Seen
Your Computer Run,
Now Watch It Fly!

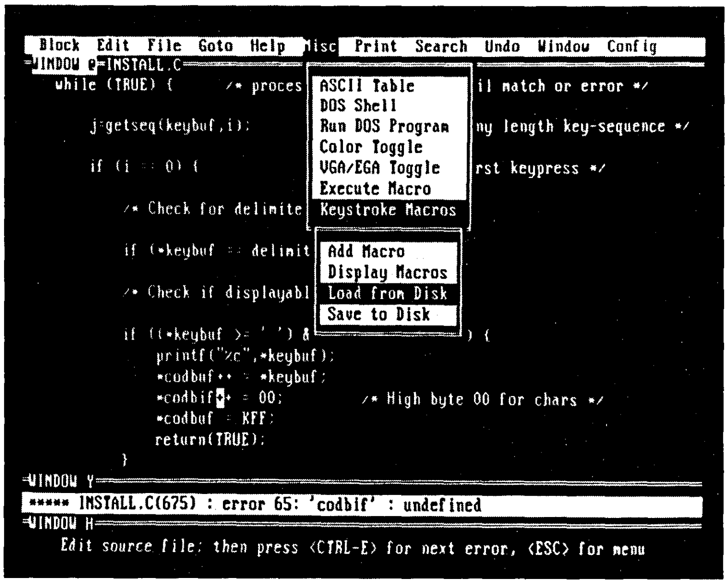


IBM-PC, XT, AT, '386 Blue Flame II
SemiDisk Solid State Disk Emulator

Featuring:
- PC-DOS, MSDOS, and Concurrent DOS Compatible
- Very Fast Access: 6.4 Mbits/sec
- High Capacity: Up to 8 MB Per Board
- Expandable to 32 MB
- Battery Backup Option
- Hardware Parity Checking
- No Mechanical Wear
- No Special Interfacing
- Prices Start Under $600.

SemiDisk
Systems, Inc.
11080 S.W. Allen #400
Beaverton, OR 97005
(503) 626-3104
FAX (503) 643-0625

Reader Service Number 162



```
 Block  Edit  File  Goto  Help  Misc  Print  Search  Undo  Window  Config
=WINDOW @=INSTALL.C=================================================
   while (TRUE) {      /* proces  ASCII Table         il match or error */
                                  DOS Shell
       j=getseq(keybuf,i);        Run DOS Program     ny length key-sequence */
                                  Color Toggle
       if (i == 0) {              VGA/EGA Toggle      rst keypress */
                                  Execute Macro
           /* Check for delimite  Keystroke Macros

           if (*keybuf == delimit  Add Macro
                                   Display Macros
           /* Check if displayabl  Load from Disk
                                   Save to Disk
           if ((*keybuf >= ' ') &                    ) {
               printf("%c",*keybuf);
               *codbuf++ = *keybuf;
               *codbif++ = 00;          /* High byte 00 for chars */
               *codbuf = KFF;
               return(TRUE);
           }
=WINDOW Y===========================================================
 ***** INSTALL.C(675) : error 65: 'codbif' : undefined
=WINDOW H===========================================================
     Edit source file; then press <CTRL-E> for next error, <ESC> for menu
```

Introducing . . .

The 1st Family of Low Cost, Powerful Text Editors

VEDIT Jr.	$ 29
VEDIT	$ 69
VEDIT PLUS	$185

FREE Evaluation Copy*
Call 1-800-45-VEDIT

Finally, you can choose the best editor for your needs without compromising performance or paying too much. And organizations that want the "same" editor for everyone can pick VEDIT® for most users and VEDIT PLUS for their power users.

The new family of VEDIT text editors are upwards compatible, easy to use and offer exceptional performance, flexibility and stunning speed. (3 to 30 times faster than the competition on large files where speed really counts.)

Call for your free evaluation copy today. See why VEDIT has been the #1 choice of programmers, writers and engineers since 1980.

VEDIT Jr.—Unmatched performance for only $29.
All VEDIT editors include a pull-down menu system with "hot keys," context sensitive on-line help, pop-up status and ASCII table, a configurable keyboard layout and flexible, unlimited keystroke macros. Edit files of any size and any line length. Perform block operations by character, line, file or column. Undo up to 1000 keystrokes— keystroke by keystroke, line by line, or deletion by deletion. Automatic indent, block indent and parentheses matching speed program development. Word wrap, paragraph formatting, justification, centering, adjustable margins and printing for word processing. Run DOS programs.

VEDIT—A best value at only $69.
Simultaneously edit up to 36 files and split the screen into windows. Search/replace with regular expressions. Includes the best compiler support available—menu driven, easy selection of compiler options, supports "Include" files and MAKE utilities.

VEDIT PLUS—Ultimate programmer's tool for only $185.
VEDIT PLUS adds the most powerful macro programming language of any editor. It eliminates repetitive editing tasks and permits creating your own editing functions. The macro language includes testing, branching, looping, user prompts, keyboard input, string and numeric variables and control over the size, position and color of windows. Source level macro debugging with breakpoints and tracing. Macros developed with VEDIT PLUS also run under VEDIT.

30 day money-back guarantee. Call for pricing of XENIX, OS/2 and FlexOS versions. Very attractive quantity pricing is available for schools, hardware and software vendors.

Compare Features and Speed

	VEDIT	BRIEF 2.10	Norton 1.3	QEdit 2.07
Pull-Down menus	Yes	No	No	Yes
Pop-Up ASCII table	Yes	No	No	No
Keystroke macros	100+	1	No	100+
Regular Expressions	Yes	Yes	No	No
"Cut and Paste" buffers	36	1	1	100
Text (book) markers	10	10	No	No
Undo keystroke by keystroke	Yes	Yes	No	No
Undo line by line	Yes	No	No	No
Normal/max Undo levels	500/1000	30/300	–	–
Variable tab positions	Yes	Yes	No	No
Configurable keyboard	Yes	Yes	No	Difficult
Integrated mouse support	Yes	No	Yes	No
FILE LIMITS				
Edit files larger memory	Yes	Yes	Difficult	No
Maximum line length	>8096	512	65,535	512
Maximum lines/file	8,388,607	65,535	>65,535	20,000
COMPILER SUPPORT			None	None
Menu driven	Yes	No	–	–
Select Compiler options	Menu	Difficult	–	–
Support "Include" files	Yes	No	–	–
BENCHMARKS 50K FILE				
Simple search	0.2 sec	1 sec	1 sec	0.3 sec
Save and continue	1 sec	2 sec	2 sec	1 sec
1000 replacements	**3 sec**	**19 sec**	17 sec	2.5 sec
BENCHMARKS 3 MEG FILE				
Simple search	1:40 min	1:36 min	Cannot	Cannot
Save and continue	1:05 min	3:23 min	Cannot	Cannot
60,000 replacements	**3:18 min**	**1:44 hour**	Cannot	Cannot
Block-column copy (40 x 200)	**2 sec**	**30 sec**	Cannot	2 sec
Insert 1 Meg file in middle of 1 Meg file	**1:11 min**	**15:13 min**	Cannot	Cannot
PRICE	**$69**	**$195**	**$75**	**$54.95**

VEDIT and CompuView are registered trademarks of CompuView Products, Inc. BRIEF is a trademark of UnderWare, Inc. Norton Editor is a trademark of Peter Norton Computing Inc. QEdit is a trademark of SemWare.

*Supports IBM PC, XT, AT, PS/2 and clones with CGA, MGA, EGA, VGA, Wyse 700, Amdek 1280 and other displays. Also supports Concurrent DOS, DESQview, Microsoft Windows, PC-MOS/386 and most networks.
*Also available for MS-DOS (CRT terminals), TI Professional and others.
*Free evaluation disk is fully functional and can edit small files.

CompuView
1955 Pauline Blvd., Ann Arbor, MI 48103
(313) 996-1299, Fax (313) 996-1308

this error by allocating a bit extra at each end of the buffer. A known value, called a sentinel, is placed at the ends. When the buffer is freed, if the sentinels have changed then there's been an underrun or overrun.

- Dependence on values in buffer obtained from malloc:

The most common value in a buffer obtained from malloc is 0. Programs can develop creeping dependencies on things being 0, or other specific values, in data returned by malloc. mem_malloc() prevents this by always setting the data in a buffer to a known nonzero value before returning a pointer to it.

This also prevents another common error: MS-DOS doesn't clear unused memory when loading a program. Thus, memory returned by malloc() may contain values left over from a previous program. mem_malloc() prevents this.

- realloc problems:

Common problems with using realloc are: (1) depending on realloc *not* shifting the location of the buffer in memory, and (2) depending on finding certain values in the uninitialized region of the realloc'd buffer. MEM flushes these out by *always* moving the buffer and stomping on values past the initialized area.

```
p = mem_malloc(5);
memset(p,0,5);
mem_realloc(p,10); /* error: p may
                      be shifted */
memset(p+5,0,5);
mem_free(p); /*will flag an error*/
```

- Memory leak detection:

Memory "leaks" are areas that are allocated but never freed. This can become a big problem in programs that have to run for many days at a time (like bulletin board software). If there are leaks, eventually the program will run out of memory and fail.

Editor's note: Explains why people put memory catch pans beneath the drain holes on their PS/2s.

Another problem is that a memory leak may indicate that a piece of memory allocated should have been added into some central data structure, but wasn't: this is a bug. MEM finds memory leaks by keeping track of all allocations and frees. When mem_term() is called, a list of all unfreed allocations is printed, along with the files and line numbers where the allocations occurred.

- Pointer checking:

Sometimes it's useful to verify that a

Continued from page 25

```
/* Check for errors. This routine does a consistency check on storage
   allocator, looking for corrupted data. It should be called when the
   application has CPU cycles to burn.
   Use: void mem_check(void);*/
void mem_check(_0);

/* Check ptr. In range of allocated data? Cause assertion failure if
   it isn't.
   Use: void mem_checkptr(void *ptr);
*/
void mem_checkptr(_1(void *));

/* Allocate and return a pointer to numbytes of storage.
   Use: void *mem_malloc(unsigned numbytes);
        void *mem_calloc(unsigned numbytes); allocated memory cleared
   Input: numbytes  Number of bytes to allocate
   Returns: if (numbytes > 0)
                 pointer to allocated data, NULL if out of memory
               else return NULL
*/
void *mem_malloc(_1(unsigned)),*mem_calloc(_1(unsigned));

/* Reallocate memory.
   Use: void *mem_realloc(void *ptr,unsigned numbytes);
*/
void *mem_realloc(_2(void *,unsigned));

/* Free memory allocated by mem_malloc(), mem_calloc(), mem_realloc().
   Use: void mem_free(void *ptr);
*/
void mem_free(_1(void *));

/* Initialize memory handler.
   Use: void mem_init(void);
   Output: mem_inited = 1
*/
void mem_init(_0);

/* Terminate memory handler. Useful for checking for errors.
   Use: void mem_term(void);
   Output: mem_inited = 0
*/
void mem_term(_0);

/* The following stuff forms the implementation rather than the
   definition, so ignore it.
*/

#if MEM_DEBUG               /* if creating debug version    */
#define mem_strdup(p)       mem_strdup_debug((p),__FILE__,__LINE__)
#define mem_malloc(u)       mem_malloc_debug((u),__FILE__,__LINE__)
#define mem_calloc(u)       mem_calloc_debug((u),__FILE__,__LINE__)
#define mem_realloc(p,u)    mem_realloc_debug((p),(u),__FILE__,__LINE__)
#define mem_free(p)         mem_free_debug((p),__FILE__,__LINE__)

#if PROTOTYPING
extern char *mem_strdup_debug(const char *,char *,int);
extern void *mem_calloc_debug(unsigned,char *,int),
        *mem_malloc_debug(unsigned,char *,int),
        *mem_realloc_debug(void *,unsigned,char *,int);
extern void mem_free_debug(void *,char *,int),mem_freefp(void *);
#else
extern char *mem_strdup_debug();
extern void *mem_calloc_debug(),*mem_malloc_debug(),
        *mem_realloc_debug();
extern void mem_free_debug(),mem_freefp();
#endif

void mem_setnewfileline(_3(void *,char *,int));

#else
#define mem_freefp          mem_free
#define mem_check()
#define mem_checkptr(p)

#endif /* MEM_DEBUG */

#endif /* MEM_H */
```

♦ ♦ ♦

End diskette compatibility problems. Call Emerald Microware.

CompatiCard I by Micro Solutions

This four drive universal floppy controller will let you run up to 16 disk drives (4 per CompatiCard), including standard 360K, 96 TPI, high density 1.2M, 8″ (SSSD or DSDD), and 720k/1.44M 3½″ drives. The CompatiCard I comes with its own MS-DOS driver, utility programs, and will let you boot on an XT (must be used as a secondary controller on an AT or 386). Use it with UniForm-PC for maximum versatility.

CompatiCard I Board	$ 119.95
CompatiCard I with UniFORM-PC	$ 179.95
8″ drive adaptor board	$ 15.00
External drive cable set	$ 15.00

CompatiCard II by Micro Solutions

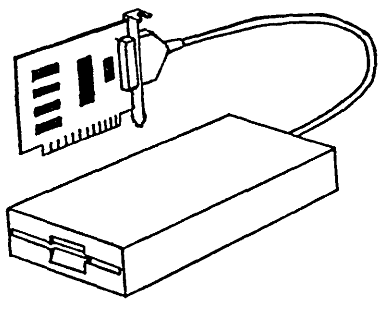
Two drive version of the CompatiCard, for the XT or AT. Same drive support as the CompatiCard I except no 8″ or single density.

CompatiCard II	$ 89.95
★★★ Special ★★★ CompatiCard II with internal 1.2M or 1.44M drive	$ 199.95

CompatiCard IV by Micro Solutions

Meet the newest four drive controller in the CompatiCard family. This CompatiCard may be used as a primary or secondary controller in almost any PC, AT, or 386 System. Boot or use 360k, 720k, 1.2M, 1.44M, or 2.88M, at any location in your system. The CompatiCard IV has a BIOS ROM on board so no external driver software is required.

CompartiCard IV	$ 139.95



Megamate by Micro Solutions

You don't have to be a computer expert to install this attractive 3½″ external drive on your PC or AT. Just plug the MegaMate controller board into any empty slot, attach the drive cable, run the installation software, and you're ready to run 720k or 1.44M diskettes.

Megamate	$ 329.95

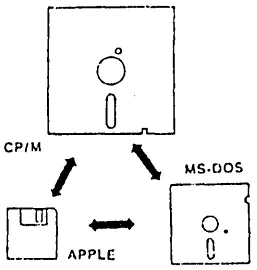
Apple ⇔ MS-DOS

MatchPoint-PC by Micro Solutions

Apple II or **NorthStar** diskettes in your IBM? The MatchPoint-PC board for the PC/XT/AT works with your standard controller card to let you read and write to NorthStar hard sector and Apple II diskettes on your PC. **INCLUDES UniForm-PC** program, as well as utilities to format disks, copy, delete, and view files from Apple DOS, PRODOS, and Apple CP/M disks.

MatchPoint-PC Board	$ 179.95

VISA MasterCard



UniForm-PC by Micro Solutions

Have you ever needed to use your CP/M diskettes on your PC? Now you can access your CP/M files and programs on your MS-DOS computer just as you would a standard MS-DOS diskette. UniForm allows you to use standard DOS commands and programs right on your original diskette without modifying or copying your files. UniForm-PC allows you to read, write, format, and copy diskettes from over 275 CP/M and MS-DOS computers on your PC, XT, AT, OR 386. With UniForm-PC and the CompatiCard, you can use 5¼″ high density, 96TPI, 3½″ (720k/1.44M), and even 8″ drives.

UniForm-PC by Micro Solutions	$ 64.95

Also available for Kaypro, & other CP/M computers

CP/M ⇔ MS-DOS

UniDOS Z80 Coprocessor Board by Micro Solutions

Don't throw out all of those old, reliable CP/M programs, run them at **LIGHTNING** speed on your PC or AT with the UniDOS 8MHz. Z80 coprocessor board. And the UniDOS Z80 runs so smoothly and transparently that you won't even be able to tell whether you're running DOS or CP/M. UniDOS emulates most common computers and terminals such as Kaypro, Xerox 820, Morrow, Osborne, VT100, and many others. Supports all standard CP/M system calls, and now works with MS-DOS version 4. **Includes UniForm-PC.**

UniDOS Z80 Coprocessor Card	$ 169.95

UniDOS by Micro Solutions

If you have a fast machine or have a V20 chip installed, you may not need to use a card slot to run your CP/M programs. Run 8080 code directly on the V20, or use emulation mode for Z80 programs.

UniDOS by Micro Solutions	$ 64.95
UniDOS w/UniForm & V20-8 chip	$ 135.00

MatchMaker by Micro Solutions

Now you can copy your **Macintosh** diskettes right on your PC/XT/AT with the MatchMaker. Just plug your external 3½″ Macintosh drive into the MatchMaker board and experience EASY access to your Mac diskettes. Includes programs to read, write, initialize, and delete files on your single or double sided Mac disks.

MatchMaker Board	$ 139.95
MatchMaker w/External Mac Drive	$ 325.00

(503) 641-8088

Call or write for our complete catalog of software, parts, accessories and complete repair services for the Kaypro, Xerox 820, and IBM PC/AT.

Copy II PC by Central Point Software

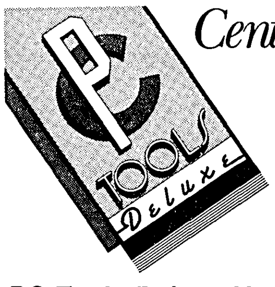
Don't let a damaged copy protected diskette stop you cold. Copy II PC lets you back up your master disks so you can keep going even when your key disk can't.

Copy II PC	$ 24.95

Copy II PC Deluxe Option Board by Central Point Software

Have a copy protected diskette with a particularly stubborn protection scheme? Would you like to be able to read your **Macintosh** disks in your 3½″ internal drive in your PC or AT? Repair a disk that's damaged, even between sectors? How about speeding up your hard disk backups (if you are using PC Tools Deluxe)? The Copy II Deluxe Option Board can help you do all of this, and more. A must for the sophisticated user.

Copy II Deluxe Option Board	$ 139.95



Central Point Software INCORPORATED

PC Tools Deluxe V5.5 by Central Point Software

This is one of the great bargains in MS-DOS utility software. But with so many features built in, we think that many people are overlooking the REAL value in PC Tools: **DISASTER RECOVERY!** Sure, the shell is great for copying, viewing, editing, and deleting files, and the desk top environment is nice for its appointment calendar, note pad, phone dialer, calculators, and ASCII table. The fast hard drive backups can't be beaten by any other program, and the file unfragmenter speeds up hard drive accesses. But where else can you get all of that along with UNDELETE, REBUILD, and UNFORMAT? Have you ever entered "ERASE *.*" and realised when all was said and done that you were in the wrong directory? How about those dreaded messages from CHKDSK, like "File allocation error...."? The very first time you recover your missing files will make you a believer. Don't wait until it happens.

PC Tools Deluxe V5.5	$ 99.95

EMERALD MICROWARE

P.O. Box 1726
Beaverton, OR 97075

VISA and Mastercard accepted. Please include $6.00 shipping and handling, $8.50 for COD, UPS-Blue or RED Label additional according to weight. Prices subject to change without notice. Please include your phone number with all correspondence.

Reader Service Number 10

pointer is actually pointing into free store. mem_checkptr(void *p) does this.

• Consistency checking:

Occasionally even MEM's internal data structures get stomped on by a wild pointer. When this happens, you can track it down by temporarily sprinkling the code with calls to mem_check(), which does a consistency check on the free storage.

• Out-of-memory handling:

An irritation in using dynamic storage is handling the cases where the storage allocator runs out of memory. MEM can be set to exhibit various behaviors when out of memory. They are:

(1) Present an "out of memory" message and terminate the program. This is the most common usage and relieves the burden on the programmer to check the return value of every malloc.

(2) Return NULL to the caller. This mimics the behavior of the ANSI malloc/calloc/realloc/strdup.

(3) Call a specified function. This is useful if the program has an "emergency reserve" pool of memory. If a program is using software virtual memory, this function can flush buffers to disk, thereby freeing up memory. If a program needs to do special cleanup before terminating, this function can do that before invoking case 1.

• Companion techniques:

To take advantage of mem_free's strategy of stomping on freed data, here's a method of detecting if pointers point to valid data or not. For each structure in general use, add the definitions:

```
struct ABC {
#if MEM_DEBUG
#define ABC_SIGNATURE 0x1234
              /* arbitrary value */
#define abc_validate(p)\
   assert((p)->id == ABC_SIGNATURE)
#define abc_init(p)\
   ((p)->id = ABC_SIGNATURE)
   int id;
#else
#define abc_validate(p)
#define abc_init(p)
#endif
   .. other members ..
};
```

Whenever a struct ABC is allocated

```
p = (struct ABC *)
    mem_malloc(sizeof(struct ABC));
abc_init(p);
```

Figure 3—mem.c

```
#if VAX11C
#define __FILE__  "mem.c"
#endif

#include        <stdio.h>

#if __ZTC__
#include        <stdlib.h>
#include        <io.h>
#else
extern void *malloc();
extern void *calloc();
extern void *realloc();
#endif

#ifndef MEM_H
#include        "mem.h"
#endif

#ifndef assert
#include        <assert.h>
#endif

#if MSC
#include        <dos.h>
#endif

#ifndef VAX11C
#ifdef BSDUNIX
#include <strings.h>
#else
#include <string.h>
#endif
#else
extern char *strcpy(),*memcpy();
extern int strlen();
#endif  /* VAX11C */

int mem_inited = 0;                     /* != 0 if initialized */

static int mem_behavior = MEM_ABORTMSG;
static int (*fp)(_0) = NULL;           /* out-of-memory handler */
static int mem_count;       /* # of allocs that haven't been free'd */

/* Determine where to send error messages */
#if MSDOS
#define ferr    stdout    /* stderr can't be redirected with MS-DOS */
#else
#define ferr    stderr
#endif

void mem_setexception(flag,handler_fp)
int flag;
int (*handler_fp)(_0);
{
    mem_behavior = flag;
    fp = (mem_behavior == MEM_CALLFP) ? handler_fp : 0;
#if MEM_DEBUG
    assert(0 <= flag && flag <= MEM_RETRY);
#endif
}

/* This is called when we're out of memory.
   Returns:
     1: try again to allocate the memory
     0: give up and return NULL
*/
static int near mem_exception()
{   int behavior;

    behavior = mem_behavior;
    while (1)
    {
        switch (behavior)
        {
            case MEM_ABORTMSG:
#if MSDOS || __OS2__
```

```
                /* Avoid linking in buffered I/O */
            {   static char msg[] = "Fatal error: out of memory\r\n";

                write(1,msg,sizeof(msg) - 1);
            }
#else
                fputs("Fatal error: out of memory\n",ferr);
#endif
                /* FALL-THROUGH */
            case MEM_ABORT:
                exit(EXIT_FAILURE);
                /* NOTREACHED */
            case MEM_CALLFP:
                assert(fp);
                behavior = (*fp)();
                break;
            case MEM_RETNULL:
                return 0;
            case MEM_RETRY:
                return 1;
            default:
                assert(0);
        }
    }
}

#if MEM_DEBUG
#undef mem_strdup

char *mem_strdup(s)
const char *s;
{
        return mem_strdup_debug(s,__FILE__,__LINE__);
}

char *mem_strdup_debug(s,file,line)
const char *s;
char *file;
int line;
{
        char *p;

        p = s
          ? (char *)mem_malloc_debug((unsigned) strlen(s)+1,file,line)
          : NULL;
        return p ? strcpy(p,s) : p;
}
#else
char *mem_strdup(s)
const char *s;
{
        char *p;

        p = s ? (char *) mem_malloc((unsigned) strlen(s) + 1) : NULL;
        return p ? strcpy(p,s) : p;
}

#endif /* MEM_DEBUG */

#ifdef MEM_DEBUG

static long mem_maxalloc;               /* max # of bytes allocated */
static long mem_numalloc;           /* current # of bytes allocated */

#define BEFOREVAL       0x12345678      /* value to detect underrun */
#define AFTERVAL        0x87654321       /* value to detect overrun */

#if SUN || SUN386
static long afterval = AFTERVAL;          /* so we can do &afterval */
#endif

/* The following should be selected to give maximum probability that
   pointers loaded with these values will cause an obvious crash. On
   Unix machines, a large value will cause a segment fault. MALLOCVAL
   is the value to set malloc'd data to. */

#if MSDOS || __OS2__
#define BADVAL          0xFF
#define MALLOCVAL       0xEE
#else
```

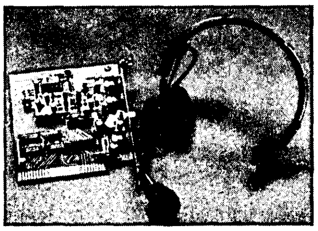
Continued on page 30

VOICE MASTER KEY®
VOICE RECOGNITION SYSTEM
FOR PC/COMPATIBLES & TANDY 1000 SERIES
A FULL FEATURED VOICE I/O SYSTEM

GIVE A NEW DIMENSION TO PERSONAL COMPUTING. . .The amazing **Voice Master Key System** adds voice recognition to just about any program or application. Voice command up to 256 keyboard macros from within CAD, desktop publishing, word processing, spread sheet, or game programs. Fully TSR and occupies less than 64K. Instant response time and high recognition accuracy. Voice recognition tool-box utilities are included. **A genuine productivity enhancer!**

SPEECH RECORDING SOFTWARE. . .Digitally record your own speech, sound, or music to put into your own software programs. Software provides sampling rate variations, graphics-based editing, and data compression utilities. Create software sound files you can add to macros for voice recognition verification response. **A complete, superior speech and sound development tool.**

SOFTWARE CONVERSION CODES. . .The **Voice Master Key System** operates a growing list of third party talking software titles using synthesized phonetics (text-to-speech) or digitized PCM, ADPCM, and CVSDM encoded sound files. **Voice Master Key System does it all!**



EVERYTHING INCLUDED. . .**Voice Master Key System** consists of a plug-in card, durable lightweight microphone headset, software, and manual. Card fits any available slot. External ports consist of mic inputs and volume controlled output sockets. **High quality throughout, easy and fun to use.**

ONLY $149.95 COMPLETE

ONLY $89.95 FOR TANDY 1000 SL/TL MODELS—SOFTWARE PACKAGE ONLY.
Requires Tandy Brand Electret microphone.

ORDER HOTLINE: (503) 342-1271
Monday-Friday, 8AM to 5PM Pacific Time

Visa/MasterCard, company checks, money orders, CODs (with prior approval) accepted. Personal checks subject to 3 week shipping delay. Specify computer type and disk format (3½" or 5¼") when ordering. Add $5 shipping charge for delivery in USA and Canada. Foreign inquiries contact Covox for C & F quotes. 30 DAY MONEY BACK GUARANTEE IF NOT COMPLETELY SATISFIED. ONE YEAR WARRANTY ON HARDWARE.

CALL OR WRITE FOR **FREE** PRODUCT CATALOG

COVOX INC. 675-D Conger St.
Eugene, Oregon 97402 U.S.A.
TEL: 503-342-1271 • FAX: 503-342-1283

Reader Service Number 143

I've used MEM for years with several large and successful products. It incorporates the suggestions of many users, and has proved to be very useful.

sprinkle the code in strategic locations with:

```
abc_validate(p); /* verify that p
        points to a valid ABC */
```

If p doesn't point to a valid ABC, abc_validate() will cause an assertion failure.

When All Bets Are Off

MEM is no panacea (the bad news) for pointer bugs. For example, if a pointer bug trashes MEM or MEM's internal data structures, all bets are off. But I've used MEM for years with several large and successful products. It incorporates the suggestions of many users, and has proved to be very useful. It's been successfully used on MS-DOS, OS/2, Suns, VMS, UNIX, and even (cheers rising from the audience) the Macintosh!

Many people resisted at first. Then after they were victimized by obscure bugs in their program, they became frustrated enough to try anything. So they installed MEM, and frequently it found the problem. Another MEM-convert.

In short (bets back on), a little bit of overhead for MEM is a small price to pay for the improved debuggability. Grab it from the Micro C BBS (503-382-7643) or order the Micro C Issue #52 disk, $6 ppd. ($8 foreign). Micro Cornucopia, P.O. Box 223, Bend, Oregon 97709, 800-888-8087.

♦ ♦ ♦

Continued from page 29

```
#define BADVAL          0x7A
#define MALLOCVAL       0xEE
#endif

/* Disable mapping macros       */
#undef  mem_malloc
#undef  mem_calloc
#undef  mem_realloc
#undef  mem_free

/* Create list of all alloc'ed pointers, retaining info about where
   each alloc came from. This is a real memory and speed hog, but who
   cares when you've got obscure pointer bugs. */

static struct mem_debug
{       struct mh
        { struct mem_debug *Mnext;                  /* next in list */
          struct mem_debug *Mprev;         /* previous value in list */
          char *Mfile;                /* filename of where allocated */
          int Mline;               /* line number of where allocated */
          unsigned Mnbytes;                /* size of the allocation */
          long Mbeforeval;                /* detect underrun of data */
        } m;
        char data[1];                /* the data actually allocated */
} mem_alloclist =
{
   {    (struct mem_debug *) NULL,
        (struct mem_debug *) NULL,
        "noname",
        11111,
        0,
        BEFOREVAL
   },
   AFTERVAL
};

/* Convert from a void *to a mem_debug struct.  */
#define mem_ptrtodl(p)  ((struct mem_debug *)\
                        ((char *)p - sizeof(struct mh)))

/* Convert from a mem_debug struct to a mem_ptr.        */
#define mem_dltoptr(dl) ((void *) &((dl)->data[0]))
#define next            m.Mnext
#define prev            m.Mprev
#define file            m.Mfile
#define line            m.Mline
#define nbytes          m.Mnbytes
#define beforeval       m.Mbeforeval

/* Set new value of file,line */
void mem_setnewfileline(ptr,fil,lin)
void *ptr;
char *fil;
int lin;
{
    struct mem_debug *dl;

    dl = mem_ptrtodl(ptr);
    dl->file = fil;
    dl->line = lin;
}

/* Print out struct mem_debug. */
static void near mem_printdl(dl)
struct mem_debug *dl;
{
#if LPTR
   fprintf(ferr,"alloc'd from file '%s' line %d nbytes %d ptr x%lx\n",
           dl->file,dl->line,dl->nbytes,mem_dltoptr(dl));
#else
   fprintf(ferr,"alloc'd from file '%s' line %d nbytes %d ptr x%x\n",
           dl->file,dl->line,dl->nbytes,mem_dltoptr(dl));
#endif
}

/* Print out file and line number. */
static void near mem_fillin(fil,lin)
char *fil;
int lin;
{
        fprintf(ferr,"File '%s' line %d\n",fil,lin);
        fflush(ferr);
}

/* If MEM_DEBUG is not on for some modules, these routines will get
   called. */
void *mem_calloc(u)
unsigned u;
{
        return mem_calloc_debug(u,__FILE__,__LINE__);
}

void *mem_malloc(u)
unsigned u;
{
        return mem_malloc_debug(u,__FILE__,__LINE__);
}

void *mem_realloc(p,u)
void *p;
unsigned u;
{
        return mem_realloc_debug(p,u,__FILE__,__LINE__);
}

void mem_free(p)
void *p;
{
        mem_free_debug(p,__FILE__,__LINE__);
}

void mem_freefp(p)
void *p;
{
        mem_free(p);
}

/* Debug versions of mem_calloc(), mem_free() and mem_realloc(). */
void *mem_malloc_debug(n,fil,lin)
unsigned n;
char *fil;
int lin;
{   void *p;

    p = mem_calloc_debug(n,fil,lin);
    if (p)
        memset(p,MALLOCVAL,n);
    return p;
}

void *mem_calloc_debug(n,fil,lin)
unsigned n;
char *fil;
int lin;
{
    struct mem_debug *dl;

    do
        dl = (struct mem_debug *)
            calloc(sizeof(*dl) + n + sizeof(AFTERVAL) - 1,1);
    while (dl == NULL && mem_exception());
    if (dl == NULL)
    {
#if 0
        printf("Insufficient memory for alloc of %d at ",n);
        mem_fillin(fil,lin);
        printf("Max allocated was: %ld\n",mem_maxalloc);
#endif
        return NULL;
    }
    dl->file = fil;
    dl->line = lin;
    dl->nbytes = n;
    dl->beforeval = BEFOREVAL;
#if SUN || SUN386 /* bus error if we store a long at an odd address */
    memcpy(&(dl->data[n]),&afterval,sizeof(AFTERVAL));
#else
    *(long *) &(dl->data[n]) = AFTERVAL;
#endif
```

INTRODUCING

on-axis™

4 CHANNEL 24-bit

QUADRATUDE-MODE COUNTER

Acquires and displays position information from optical encoders. Resolution is four times the encoder. Complete with demo software and driver source code.

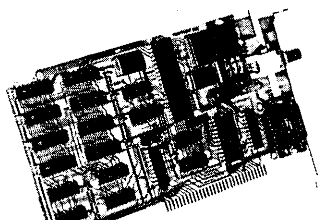
$299.00 INTRODUCTORY PRICE

(add $150 for optional connector to Bauch & Lomb "glass scales".)

étude™

25 MHz 8-bit

ANALOG-TO-DIGITAL CONVERTER



Based on the TRW THC10681 hybrid flash converter, its high signal-to-noise ratio yields excellent accuracy at the Nyquist limit.

- 4 KB of cache SRAM or to host as converted at DMA speed
- I/O or DMA data transfer
- 10 MHz full-power bandwidth
- 3.92 mV resolution
- Factory callibrated
- 16 jumper selectable base addresses
- External clock and trigger inputs TTL compatible
- Software source code included

PRICE: $495.00

Each product requires: PC compatible ½ length 8-bit expansion slot. DOS 2.11 or greater. EGA, VGA or Hercules display needed for graphic representation of data.

Silicon Alley Inc.

P.O. BOX 59593
RENTON, WA 98058
(206)255-7410

Reader Service Number 177

Continued from page 31

```
    /* Add dl to start of allocation list       */
    dl->next = mem_alloclist.next;
    dl->prev = &mem_alloclist;
    mem_alloclist.next = dl;
    if (dl->next != NULL)
        dl->next->prev = dl;

    mem_count++;
    mem_numalloc += n;
    if (mem_numalloc > mem_maxalloc)
        mem_maxalloc = mem_numalloc;
    return mem_dltoptr(dl);
}

void mem_free_debug(ptr,fil,lin)
void *ptr;
char *fil;
int lin;
{
        struct mem_debug *dl;

        if (ptr == NULL)
                return;
#if 0
        {       fprintf(ferr,"Freeing NULL pointer at ");
                goto err;
        }
#endif
        if (mem_count <= 0)
        {       fprintf(ferr,"More frees than allocs at ");
                goto err;
        }
        dl = mem_ptrtodl(ptr);
        if (dl->beforeval != BEFOREVAL)
        {
#if LPTR
                fprintf(ferr,"Pointer x%lx underrun\n",ptr);
#else
                fprintf(ferr,"Pointer x%x underrun\n",ptr);
#endif
                goto err2;
        }
#if SUN || SUN386 /*Bus error if we read a long from an odd address*/
  if (memcmp(&dl->data[dl->nbytes],&afterval,sizeof(AFTERVAL)) != 0)
#else
        if (*(long *) &dl->data[dl->nbytes] != AFTERVAL)
#endif
        {
#if LPTR
                fprintf(ferr,"Pointer x%lx overrun\n",ptr);
#else
                fprintf(ferr,"Pointer x%x overrun\n",ptr);
#endif
                goto err2;
        }
        mem_numalloc -= dl->nbytes;
        if (mem_numalloc < 0)
        {       fprintf(ferr,"error: mem_numalloc = %ld,
                  dl->nbytes = %d\n", mem_numalloc,dl->nbytes);
                goto err2;
        }

        /* Remove dl from linked list   */
        if (dl->prev)
                dl->prev->next = dl->next;
        if (dl->next)
                dl->next->prev = dl->prev;

        /* Stomp on the freed storage to help detect references */
        /* after the storage was freed.                         */
        memset((void *) dl,BADVAL,sizeof(*dl) + dl->nbytes);
        mem_count--;

        /* Some compilers can detect errors in the heap.        */
#if DLC
        {       int i;
                i = free(dl);
                assert(i == 0);
        }
#else
```

```
        free((void *) dl);
#endif
        return;

err2:
        mem_printdl(dl);
err:
        fprintf(ferr,"free'd from ");
        mem_fillin(fil,lin);
        assert(0);
        /* NOTREACHED */
}

/* Debug version of mem_realloc(). */
void *mem_realloc_debug(oldp,n,fil,lin)
void *oldp;
unsigned n;
char *fil;
int lin;
{   void *p;
    struct mem_debug *dl;

    if (n == 0)
    {   mem_free_debug(oldp,fil,lin);
        p = NULL;
    }
    else if (oldp == NULL)
        p = mem_malloc_debug(n,fil,lin);
    else
    {
        p = mem_malloc_debug(n,fil,lin);
        if (p != NULL)
        {
            dl = mem_ptrtodl(oldp);
            if (dl->nbytes < n)
                n = dl->nbytes;
            memcpy(p,oldp,n);
            mem_free_debug(oldp,fil,lin);
        }
    }
    return p;
}

void mem_check(_0)
{   register struct mem_debug *dl;

    for (dl = mem_alloclist.next; dl != NULL; dl = dl->next)
        mem_checkptr(mem_dltoptr(dl));
}

void mem_checkptr(p)
register void *p;
{   register struct mem_debug *dl;

    for (dl = mem_alloclist.next; dl != NULL; dl = dl->next)
    {
        if (p >= (void *) &(dl->data[0]) && p < (void *)((char *)dl +
            sizeof(struct mem_debug)-1 + dl->nbytes))
                goto L1;
    }
    assert(0);

L1:
    dl = mem_ptrtodl(p);
    if (dl->beforeval != BEFOREVAL)
    {
#if LPTR
            fprintf(ferr,"Pointer x%lx underrun\n",p);
#else
            fprintf(ferr,"Pointer x%x underrun\n",p);
#endif
            goto err2;
    }
#if SUN || SUN386  /*Bus error if we read a long from an odd address*/
    if (memcmp(&dl->data[dl->nbytes],&afterval,sizeof(AFTERVAL)) != 0)
#else
    if (*(long *) &dl->data[dl->nbytes] != AFTERVAL)
#endif
    {
#if LPTR
```

Continued on page 34

Confusing Code?

```
#include <stdio.h>
text_count(){int c,nlines,nwords,nchars,inword;
inword=NO;nlines=nwords=nchars=0;while((c
=getchar())!=EOF){++nchars;if (c=='\n')
++nlines;if ((c==' ')||(c=='\n'))inword=NO;else
if(inword==NO){inword=YES;++nwords;}}printf(
"%d %d %d\n",nlines,nwords,nchars);}
```

C It Your Way!

```
#include <stdio.h>

text_count ()
{
     int c, nlines, nwords, nchars, inword;

     inword = NO;
     nlines = nwords = nchars = 0;
     while ((c = getchar()) != EOF) {
          ++nchars;
          if (c == '\n')
               ++nlines;
          if ((c == ' ') || (c == '\n'))
               inword = NO;
          else if (inword == NO) {
               inword = YES;
               ++nwords;
          }
     }
     printf ("%d %d %d\n", nlines, nwords, nchars);
}
```

NOW with **C-Clearly™**, format C source code exactly the way you want it. C-Clearly's context sensitive analysis will format any C program in your own personal or corporate style.

EASY to use, C-Clearly's style templates are a snap to modify, since they resemble C source code you edit into your preferred format. Templates are included for several common styles as well as standard K & R.

LISTINGS can also be created with function names and comments highlighted for improved readability. Listing options include line numbers, headers and/or footers and flow lines.

IDEAL for making obtuse code clear. Allows all of your source code to be presented in a consistent format of your choosing. Also great for code walkthroughs and final documentation listings.

WORKS with all IBM PC, XT, AT, PS/2 and compatibles, with 512K RAM. Automatically processes all include files and pre-processor statements. ANSI-C compatible. Not copy protected.

C-Clearly $129.95

Shipping & Handling USA $3; Canada/Mexico $10; Other Countries $15; CA Residents add sales tax. Visa/MasterCard/COD accepted.

For orders or information call:
1-800-662-8266

V COMMUNICATIONS, INC.
3031 Tisch Way, Suite 802, Dept. MC5
San Jose, CA 95128 (408) 296-4224

Reader Service Number 62

Get organized instantly.

Create,
Find,
View, & Edit
notes,
addresses,
documents,
data.

* No setup.
* No keywords to define.
* Configurable, Full-screen editor.
* Turbo fast & little.

WIZARD

the intelligent information processor.

Only $49.

Acquired Intelligence

PO Box 2091 Davis, CA 95617
916-753-0360

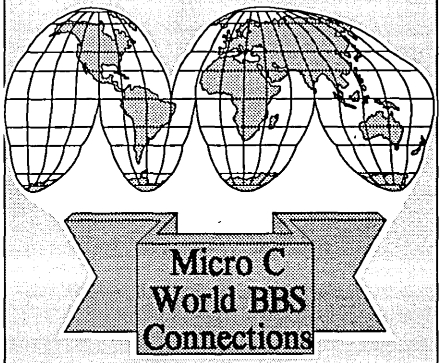
Reader Service 72

Continued from page 33

```
            fprintf(ferr,"Pointer x%lx overrun\n",p);
#else
            fprintf(ferr,"Pointer x%x overrun\n",p);
#endif
            goto err2;
    }
    return;

err2:
    mem_printdl(dl);
    assert(0);
}

#else

void *mem_malloc(numbytes)
unsigned numbytes;
{       void *p;

        if (numbytes == 0)
                return NULL;
        while (1)
        {
                p = malloc(numbytes);
                if (p == NULL)
                {       if (mem_exception())
                                continue;
                }
                else
                        mem_count++;
                break;
        }
        /*printf("malloc(%d) = x%lx\n",numbytes,p);*/
        return p;
}

void *mem_calloc(numbytes)
unsigned numbytes;
{       void *p;

        if (numbytes == 0)
                return NULL;
        while (1)
        {
                p = calloc(numbytes,1);
                if (p == NULL)
                {       if (mem_exception())
                                continue;
                }
                else
                        mem_count++;
                break;
        }
        /*printf("calloc(%d) = x%lx\n",numbytes,p);*/
        return p;
}

void *mem_realloc(oldmem_ptr,newnumbytes)
void *oldmem_ptr;
unsigned newnumbytes;
{   void *p;

    if (oldmem_ptr == NULL)
        p = mem_malloc(newnumbytes);
    else if (newnumbytes == 0)
    {   mem_free(oldmem_ptr);
        p = NULL;
    }
    else
    {
        do
            p = realloc(oldmem_ptr,newnumbytes);
        while (p == NULL && mem_exception());
    }
    /*printf("realloc(x%lx,%d) = x%lx\n",oldmem_ptr,newnumbytes,p);*/
    return p;
}

void mem_free(ptr)
void *ptr;
```

```
{
    /*printf("free(x%lx)\n",ptr);*/
    if (ptr != NULL)
    {   assert(mem_count > 0);
        mem_count--;
#if DLC
        {       int i;

                i = free(ptr);
                assert(i == 0);
        }
#else
        free(ptr);
#endif
    }
}
#endif /* MEM_DEBUG */

void mem_init()
{
        if (mem_inited == 0)
        {       mem_count = 0;
#if MEM_DEBUG
                mem_numalloc = 0;
                mem_maxalloc = 0;
                mem_alloclist.next = NULL;
#endif
                mem_inited++;
        }
}

void mem_term()
{
        if (mem_inited)
        {
#if MEM_DEBUG
                register struct mem_debug *dl;
                for (dl = mem_alloclist.next; dl; dl = dl->next)
                {       fprintf(ferr,"Unfreed pointer: ");
                        mem_printdl(dl);
                }
#if 0
                fprintf(ferr,"Max amount ever allocated==%ld bytes\n",
                        mem_maxalloc);
#endif
#else
                if (mem_count)
                        fprintf(ferr,"%d unfreed items\n",mem_count);
#endif /* MEM_DEBUG */
                assert(mem_count == 0);
                mem_inited = 0;
        }
}

#undef next
#undef prev
#undef file
#undef line
#undef nbytes
#undef beforeval
```

♦ ♦ ♦



Bill Bolton
Software Tools
P.O. Box 225
St. Ives, NSW
Australia 2075

Scott Ladd
705 W. Virginia
Gunnison, CO 81230
(303) 641-6438
BBS: (303) 641-5125
Duck Tower
FidoNet 1:104/708

Barney Lejeune
412 Estelle Dr.
Carencro, LA 70520
BBS: (318) 788-7522
4 p.m. to 7 a.m. weekdays
24 hours on weekends
300, 1200, 2400 baud
SWLVTS-BBS

Tadas Osmolski
Generation Five Inc.
12061 Tech Road
Silver Spring, MD 20904
BBS: (301) 680-9300

Steve O'Toole
16592 Malaga Ct.
Fontana, CA 92336
Tec-PC
BBS: (714) 355-0373

Jim Wheater
185 Parmalee Ave.
Hawthorne, NJ 07506
Computer Nookery
BBS: (201) 423-4258

The AT Keyboard Interface

Roll Your Own

Ever hanker to cobble up one of those fancy AT keyboards to your latest project? If so you'll understand why this is a key article. (Touchy, touchy.)

A customer of my consulting firm needed a new keyboard for his Z80-based composite-video character generator. He liked the IBM AT type keyboards because of their ready availability in a variety of styles. Also, the size made them easy to incorporate into his product.

After a quick reading of the IBM *AT Technical Reference Manual,* I figured the project was easy enough. Later experiments with clone keyboards indicated, however, that if you like the RS-232 "standard," you'll love the AT keyboard "standard."

The AT type keyboard is more complicated than the XT type but offers about twice the speed and error checking. Besides, most keyboards sold now are either AT only or are switchable between AT and XT.

The interface presented here offers a simple, general purpose solution for many one-of-a-kind or small volume applications. You can easily interface the venerable old 8251 USART to most CPUs, and it works with both the 84 key and 101/102 key models.

The "Standard"

The AT keyboard DIN connector has:
(1) Clock
(2) Data
(3) No connection (sometimes reset)
(4) Ground
(5) +5 volts

XT keyboards use pin 3 for a reset (active low), and some AT versions retained this. IBM's *AT Technical Reference* says pin 3 is reserved, so the keyboard has to do its own power-on reset. Open collectors drive pins 1 and 2 so when neither side has anything to say, these lines stay high. (They're pulled up by resistors on the keyboard.)

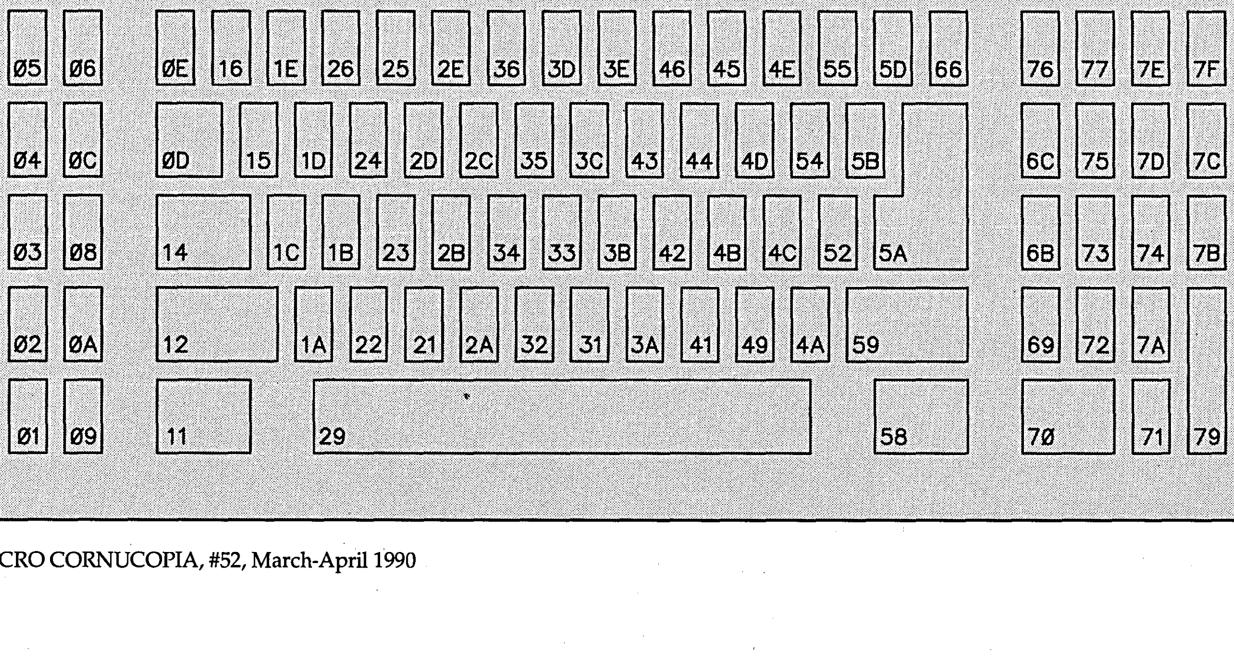
Data transmits one bit at a time. The keyboard handles clocking in both directions at about 10K bits per second. The current data bit is placed on the data line, the clock line is forced low, and then released. The data bit is then changed and the process continues until all bits have been sent.

A complete character transmission consists of a start bit (always 0), 8 data bits (least significant bit first), an odd parity bit, and a stop bit (always 1). Keyboard data should be valid on both the falling and rising edges of the clock. When the keyboard receives data, it latches it on the clock's rising edge.

Either the keyboard or the computer can initiate a transmission, but the computer is always in charge. When the computer is too busy to be bothered by the keyboard, it pulls the clock line low.

Before the keyboard sends a character, it checks the clock line. If the clock line is high, the keyboard sends the data; otherwise, it stores the character. It can store up to 16 keypresses.

Figure 1—84 Key Make Codes


Ø5 Ø6 ØE 16 1E 26 25 2E 36 3D 3E 46 45 4E 55 5D 66 76 77 7E 7F
Ø4 ØC ØD 15 1D 24 2D 2C 35 3C 43 44 4D 54 5B 6C 75 7D 7C
Ø3 Ø8 14 1C 1B 23 2B 34 33 3B 42 4B 4C 52 5A 6B 73 74 7B
Ø2 ØA 12 1A 22 21 2A 32 31 3A 41 49 4A 59 69 72 7A
Ø1 Ø9 11 29 58 7Ø 71 79

By Don Rowe

Digital Arts
P.O. Box 323
Julian, CA 92036
(619) 765-2463

The computer can interrupt a character (by forcing the clock line low) as long as the keyboard has not sent the 10th bit. After the 10th bit, the computer has to let the keyboard finish the current character. If there's an error, the computer can request retransmission.

When the computer has a command to send to the keyboard, it forces the data line low. When the keyboard sees the data line low and the clock line high, it clocks 10 bits (start bit, 8 data bits, parity bit), forces the data line low in acknowledgement, and clocks once more.

The keyboard takes the initial low level on the data line as the start bit; the first data bit must be valid on the rising edge of the first clock. The keyboard responds to most commands by returning a hex FA. If it detects an error, it requests retransmission with a hex FE.

The Data

Most characters are keycodes sent from the keyboard to the computer. Keycodes are either "make" codes (sent when you press a key), or "break" codes (sent when you release a key). A break code is usually hex F0 followed by the make code and treated as a two byte sequence.

The AT type keyboard is more complicated than the XT type but offers about twice the speed and error checking.

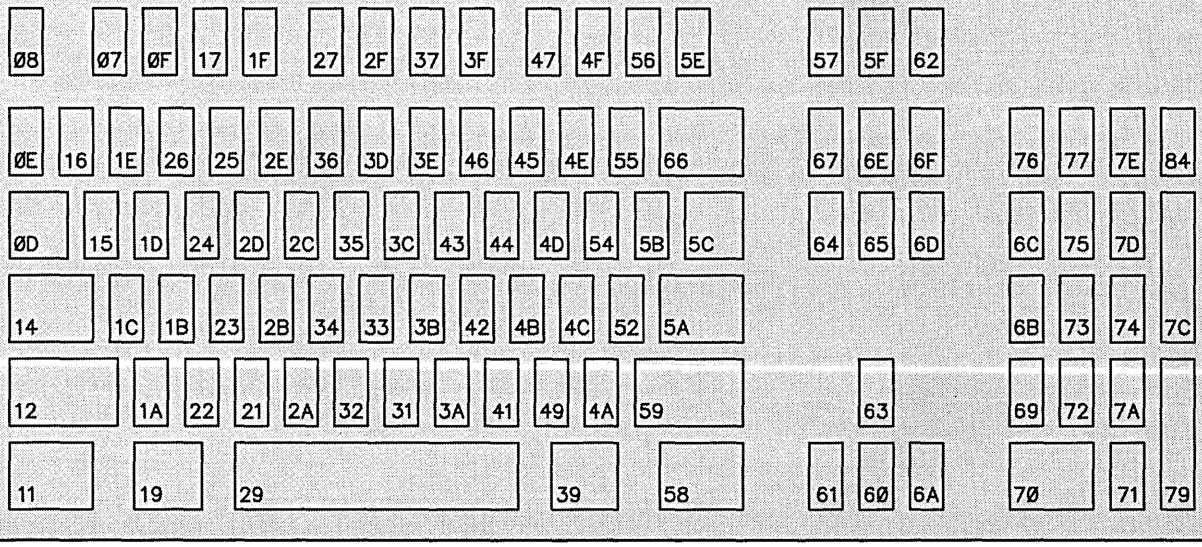
If the computer interrupts transmission of the second byte by forcing the clock line low, the keyboard resends both bytes when the clock line is released. Figure 1 shows the make codes in hex for an 84 key keyboard.

IBM 101/102 key keyboards support three different sets of make/break codes. Set 1 is the XT code set. Each key is assigned a 7 bit code. The eighth bit is 0 for a make code and 1 for a break code.

Set 2 is an extended 84 key code set. The keys common to both the 84 and 101 key keyboards use the same make/break codes. The new keys on the cursor pad are sent with a hex E0 prefix. (In other words, it sends the same code sent by the equivalent key on the 84-key unit, but the code is prefixed by E0.)

The up arrow key normally has a make code of hex 75 on an 84 key keyboard. Pressing the up arrow key on the cursor pad of a 101 key keyboard sends hex E0 75 if numlock is off and the shift key is up. The break code would be hex E0 F0 75. If the Numlock key is down, the make code is hex E0 12 E0 75, which is equivalent to shift up arrow on an 84

Figure 2—101 Key Make Codes, Mode 3


Ø8 Ø7 ØF 17 1F 27 2F 37 3F 47 4F 56 5E 57 5F 62
ØE 16 1E 26 25 2E 36 3D 3E 46 45 4E 55 66 67 6E 6F 76 77 7E 84
ØD 15 1D 24 2D 2C 35 3C 43 44 4D 54 5B 5C 64 65 6D 6C 75 7D
14 1C 1B 23 2B 34 33 3B 42 4B 4C 52 5A 6B 73 74 7C
12 1A 22 21 2A 32 31 3A 41 49 4A 59 63 69 72 7A
11 19 29 39 58 61 6Ø 6A 7Ø 71 79

key keyboard. The break code would be E0 F0 75 E0 F0 12.

If the shift key is down, the make code would be E0 F0 12 E0 75, which is the break code for shift and the make code for the up arrow key. The break code would then be E0 F0 75 E0 12, which is the break code for the up arrow key and the make code for the shift key. Gesundheit!

Mode 3 eliminates all the confusion of mode 2 by assigning each key a unique code. Figure 2 shows the make codes in hex for mode 3 on a 101 key keyboard. Not all keyboards support mode 3. The BTC 101 key keyboard my customer ultimately selected did not accept the "change mode" command, leaving me stuck in mode 2.

This wouldn't be bad if we used each key for its originally intended function—the computer could just ignore the hex E0 prefixes. Unfortunately, some cursor pad keys were assigned new functions unrelated to their counterparts on the numeric keypad.

I had to treat the E0 code as an alternate code selector, much as the regular shift, ctrl, and alt keys work on a standard keyboard. This gets complicated with my application because the shift and control key make/break codes within an E0 sequence require special handling. If you must use a 101 key keyboard, I recommend getting one that supports mode 3.

Besides the keycodes, there are several command codes that the computer can send to the keyboard, and status codes which the keyboard sends to the computer. Figure 3 shows the status codes sent from the keyboard to the computer, while Figure 4 shows the commands from the computer to the keyboard.

On power up, the keyboard runs a self-test and sends a hex AA if all is well, or hex FD if it detected a problem. Once the diagnostics finish, the keyboard starts scanning keys and sending keycodes. It also monitors the data line for commands from the computer.

I discovered an "interesting" variation in the BTC keyboard. It ran the self-test and sent a hex AA, but continued sending hex AAs as fast as it could. If the data line was forced low, the keyboard clocked in the data, but it ignored all commands and immediately resumed sending hex AAs.

After much hair pulling, teeth gnashing, and fruitless phone calls to BTC, I finally noticed that the AAs were sent with even parity rather than the usual

Figure 3—Status From an 84 Key Keyboard to the Computer

```
00 The 16 key buffer overflowed
AA Self-test passed
EE Sent in response to EE from the computer
F0 Break code prefix
FA Acknowlege receipt of command without error
FD Self-test failed
FE Resend last character
101 key keyboards send FC instead of FD for test failure.
```

♦ ♦ ♦

Figure 4—Commands From the Computer to an 84 Key Keyboard

```
Unless stated otherwise, keyboard responds with FA acknowledge
code. If a second byte is required, both are acknowledged with FA.

ED Set or reset LEDs according to next byte
   Bit 0 = Scroll lock (1=on, 0=off)
   Bit 1 = Num lock (1=on, 0=off)
   Bit 2 = Caps lock (1=on, 0=off)
   Bits 3-7 must be 0
EE Echos EE as a diagnostic aid.  No FA is sent.
F3 Sets key repeat and delay timing according to next byte
   Bit 7 = 0
   Delay = ((binary value of bits 5-6)+1) * 250 ms.
   Repeat rate: A = binary value of bits 0-2
                B = binary value of bits 3-4
                Rate = (8+A) * 2^B * 4.17ms.
F4 Clears the key buffer and starts scanning the keys
F5 Restore default settings and wait for an enable command
F6 Restores default settings
FE Re-transmits the last character sent
FF Reset

Additional commands for the 101 key keyboard:
F0 Requires a second byte of 1,2 or 3 to select the scan codes
   A second byte of 0 causes the keyboard to return the current
   selection. The BTC keyboard does not accept this command.
F2 Sends 2 ID bytes after FA acknowledge
```

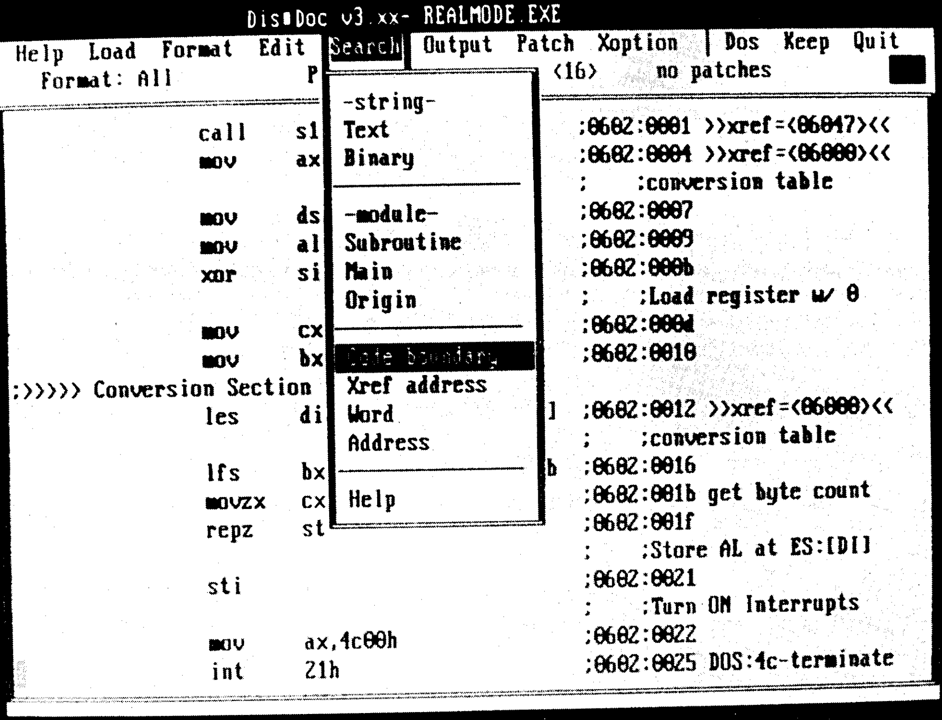
♦ ♦ ♦

Figure 5—8251 Pins

```
Pin: Name:       Function:
1    D2          Data line 2
2    D3          Data line 3
3    RxDAT       Serial data input
4    GND         Ground
5    D4          Data line 4
6    D5          Data line 5
7    D6          Data line 6
8    D7          Data line 7
9    TxCLK       Transmitter baud rate clock
10   /WR         Write enable, active low
11   /CS         Chip select, active low
12   C/D         1 selects control regiser, 0 selects data register
13   /RD         Read enable, active low
14   RxRDY       High when a character has been received
15   TxRDY       High when ready to transmit a character
16   SYNDET/BD   Synch/break detect, active high
17   /CTS        Clear to send input
18   TxEMPTY     High when transmitter shift register is empty
19   TxDAT       Serial data output
20   CLOCK       Master chip clock
21   RESET       Active high
22   /DSR        Data set ready input
23   /RTS        Request to send output
24   /DTR        Data terminal ready output
25   RxCLK       Receiver baud rate clock
26   Vcc         +5 power
27   D0          Data line 0
28   D1          Data line 1
```

♦ ♦ ♦

KNOWLEDGE = POWER



```
                    Dis•Doc v3.xx- REALMODE.EXE
 Help  Load  Format  Edit  Search  Output  Patch  Xoption | Dos  Keep  Quit
   Format: All          P                  <16>     no patches
               call    sl  -string-                ;0602:0001 >>xref=<06047><<
               mov     ax  Text                    ;0602:0004 >>xref=<06000><<
                           Binary                  ;      ;conversion table
               mov     ds  ----------              ;0602:0007
               mov     al  -module-                ;0602:0009
               xor     si  Subroutine              ;0602:000b
                           Main                    ;      ;Load register w/ 0
               mov     cx  Origin                  ;0602:000d
               mov     bx  ----------              ;0602:0010
;>>>>> Conversion Section   Code boundary
               les     di  Xref address         ]  ;0602:0012 >>xref=<06000><<
                           Word                    ;      ;conversion table
               lfs     bx  Address              b  ;0602:0016
               movzx   cx  ----------              ;0602:001b get byte count
               repz    st  Help                    ;0602:001f
                                                   ;      ;Store AL at ES:[DI]
               sti                                 ;0602:0021
                                                   ;      ;Turn ON Interrupts
               mov     ax,4c00h                    ;0602:0022
               int     21h                         ;0602:0025 DOS:4c-terminate
```

Unlimited file size. . . Fully automatic

Batch mode and interactive

Locates Data/Code boundaries

Built-in BIOS Preprocessor

All Data formats including DB, DW, DD & DUP

EXE Unpacker included!

No Source Code? No Problem.
New **Dis•Doc** *Professional* is your **dual-mode** key to any DOS source code. It works in batch **and** interactive modes simultaneously, allowing you to generate the core information of even the most complex programs **fast**. . . and modify them even faster. Most programs will come apart in just two minutes. Imagine what you can do with a tool this powerful! **Dis•Doc** sifts through programs **eight times** for guaranteed accuracy. When code gets mixed up with data, our toolbox comes to the rescue with **smart** search and **easy** edit utilities. **Dis•Doc** can handle any instruction set up to and including 486 and offers a variety of other great features that you can sample on our **Free Demo Disk.**

Warning: Dis•Doc ***Professional*** **may change the way you work** ***forever.***
Programmers who used to shy away from fixing outmoded programs with no source code are going to discover a **valuable** new talent: the ability to modify and revise codes that would cost way too much to start over (it's a programming manager's dream). Save your employer huge new-programming fees and enhance your marketability. **Dis•Doc** is so easy to learn, you'll be a high-dollar hero in no time!

Knowledge really ***is*** **power.**
Dis•Doc *Professional* is an amazing new teaching tool. Learn how programs work. . . take them apart and see the writing techniques that top pros use. Use it to assist in debugging. Hunt down viruses and write killers. **Dis•Doc** can save you **years** of frustration, and it only costs **$149.95** including the EXE Unpacker (until January 1). To order your **Dis•Doc** *Professional* kit or our free demo disk, simply call:

1-800-446-4656

WITHIN CT & OUTSIDE THE U.S., CALL (203) 953-0236

MasterCard & VISA ● Shipped Immediately Via UPS
RJSwantek, Inc., 178 Brookside Road ● Newington CT 06111

odd parity. This may have something to do with the auto-sensing procedure to determine if it is connected to an XT or AT type computer, although the AT data format is used consistently. Once the computer sends a hex FE (the resend command) acknowledging the error, the keyboard begins normal operation.

The Hardware

Required parts:
(1) 8251A USART
(1) 74LS03 quad NAND
(1) 74LS14 hex schmitt trigger inverter
(1) 74LS74 dual D type flip flop

All parts are readily available, usually for under $5. One supplier is:

Jameco Electronics
1355 Shoreway Road
Belmont, CA 94002
(415) 592-8097

This is not intended to be a complete tutorial on the 8251 chip. Before attempting any kind of design, you should obtain and read the complete chip specifications and make sure your system and software meet all timing requirements!

I use the 8251 USART because it's available, cheap, and simpler than newer chips for this application. Figure 5 shows the pinout of the 8251. Those of you familiar with the RS-232 serial data format will have noticed its similarity to the keyboard data format.

The 8251 calculates parity, handles conversions between serial and parallel, and does error checking, oblivious to the fact that it's talking to a keyboard rather than another USART. Figure 6 shows the final circuit.

The keyboard drives the 8251's clock line, and an open-collector chip connects the serial data output line to the keyboard's data line. Since rise times of the open-collector clock and data lines are unspecified, I used schmitt triggers to guarantee sharp rising and falling edges.

With nothing more than the above connections, the 8251 could receive data as long as that data was valid on the rising edge of the clock. One keyboard I tried changed data half way through the clock pulse. The BTC keyboard only works reliably when data is read on the rising edge of the clock.

For safety, I added a 74LS74 to latch data on the falling edge of the clock. A jumper selects data either from the doubly inverted data line or from the /Q output of the 74LS74. If you have an oscilloscope, you can hold a key down and watch the clock and data lines to see which "standard" your keyboard supports.

If the computer is busy, it might not see an incoming keypress right away. If more keypresses come in, you could get overrun errors and lost data. So I added a 74LS03 to force the clock line low when the RxRDY line goes high in response to an incoming character.

You cannot interrupt two-byte combinations, such as break codes, without causing retransmission of both bytes. When the first byte comes in, RxRDY goes high and forces the clock line low. The CPU reads the character, resetting RxRDY and releasing the clock line.

Since the keyboard was interrupted when the clock line was forced low, it once again sends the first byte of the sequence—chasing its tail forever. The other input of the 74LS03 connects to the normally high /RTS output. When the computer receives the first character of a two-byte transmission, it sets /RTS low. Now the output of the 74LS03 floats regardless of what RxRDY does and the entire sequence comes in.

Transmitting is a little more difficult. The keyboard begins clocking when it sees the data line low. But the 8251 will not shift the 0 start bit to the data line until it gets a clock pulse—Oops!

Figure 6—AT Keyboard Interface Schematic

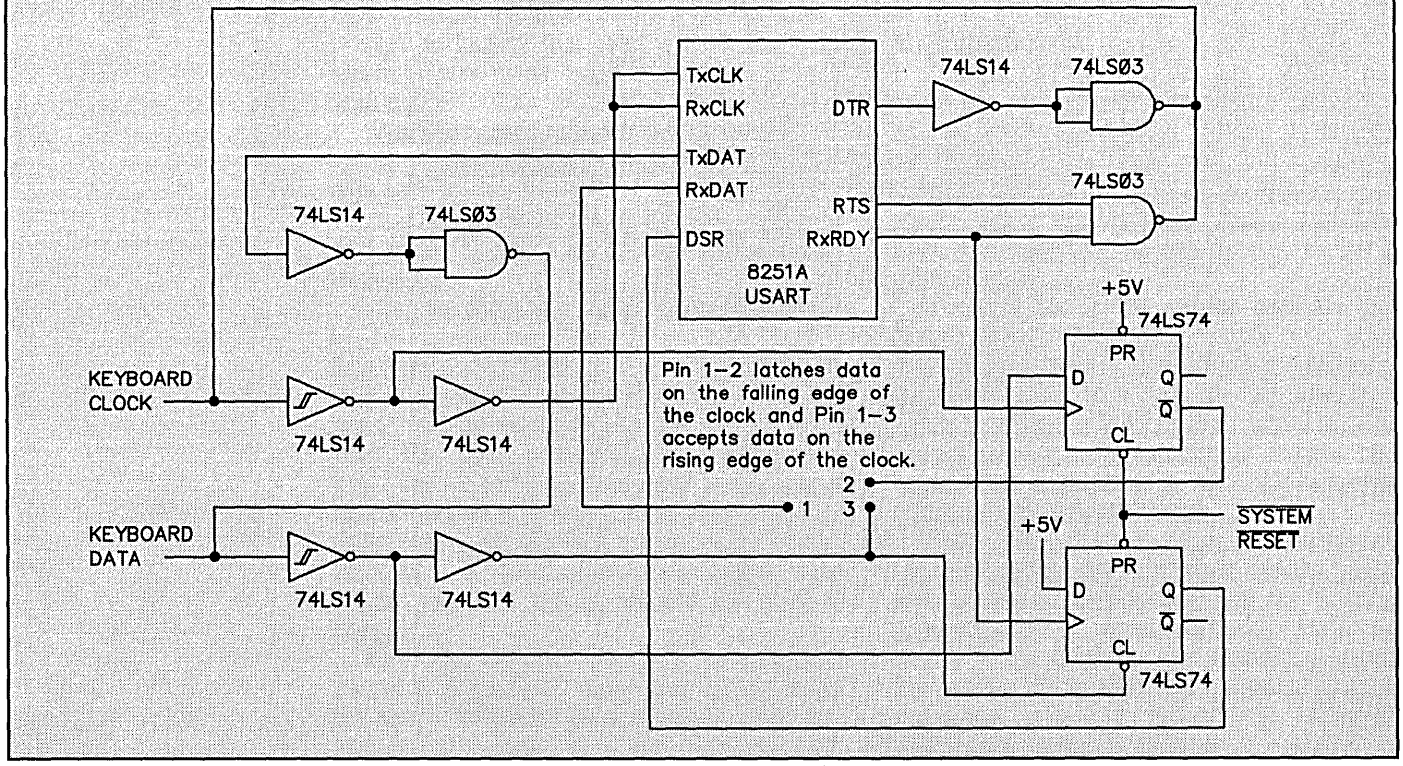


Figure 7—Program Structures

```
IF condition
  Routine to do only if condition is true
  Branch to END_IF
ELSE
  Optional routine to do only if condition is false
END_IF

WHILE condition
  Test condition and branch to END_WHILE if condition is false
  Routine to do if condition is true
  Branch back to WHILE and check condition again
END_WHILE

UNTIL condition
  Routine here is always done
  Test condition at the end of the routine
  If condition is false, branch back to UNTIL
END_UNTIL

CALL subroutine

RETURN from subroutine

;;Comments are preceded by semicolons
```

♦ ♦ ♦

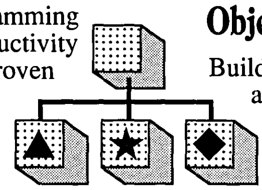
The /DTR output connects to the clock line via an open-collector driver, forcing the line low as needed to inhibit further transmissions. /DTR can clock the 8251 "manually," giving it a push to get started. Once the start bit appears on the data line and the clock line is released, the keyboard takes over.

What happens if the keyboard and 8251 both try to transmit at the same time? A fight! Suppose the keyboard was on the 10th bit. The keyboard continues clocking with both the 8251 and keyboard driving the data line simultaneously. The keyboard might notice that the clock line was forced low. But if the 8251 shifted a 1 onto the data line, the keyboard would not notice that a character was waiting, possibly hanging the system.

I used the other half of the 74LS74 as a "keyboard now transmitting" detector. Whenever the data line goes low, as it must on a start bit, it clears the 74LS74, forcing the Q output to 0. The rising edge of RxRDY sets Q high again, which is monitored via /DSR.

Since the computer still can't tell how

The Professional Library for Object-Oriented Pascal



Object-oriented programming multiplies your productivity by promoting reuse of proven software modules. Don't start from scratch! Use **Object Professional 1.0**, a powerful library of over 30 object types containing over 1000 methods.

Powerful User Interfaces

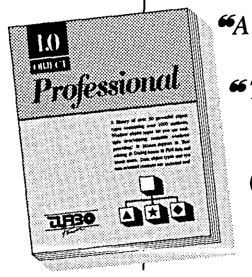
The window object types let you use multiple overlapping and resizeable windows. The windows provide capability for ■ mouse support ■ scroll bars ■ menus ■ text editing ■ dialog boxes ■ pick lists ■ scrolling data entry screens ■ printed forms ■ help capability, and more. The window classes are incredibly flexible — you can create text-mode PM look-alikes or your own unique look and feel.

Object Oriented Data Too

Build your programs from proven and documented object types like stacks, linked lists, virtual arrays, and more. Make your own custom data types by simply inheriting from one of the provided types and adding your own methods and instance variables.

The Impossible, Made Easy

System-oriented routines provide ■ swappable TSRs in only 6K of RAM ■ DOS and BIOS capabilities ■ EMS management ■ keyboard macros ■ interrupt management ■ swapping Exec manager, and much more.



Complete Documentation, Full Source

Object Professional has three volumes of complete documentation, online reference guide, free technical support, and full source code. You pay no royalties. Hot demo programs show you how to use the power of Object Professional. You'll get up to speed fast with OOP!

Object Professional is the successor to the acclaimed Turbo Professional:

"A superbly crafted toolbox."
Kent Porter, DDJ, 4/88

"The range of this toolkit is simply astonishing."
Jeff Duntemann, DDJ, 5/89

Object Professional 1.0, only $150.

Call toll-free to order.

1-800-333-4160

8AM - 5PM PST Monday through Friday, USA & Canada.
For more information call (408) 438-8608.
TurboPower Software PO Box 66747 Scotts Valley, CA 95066-0747

Satisfaction guaranteed or your money back within 30 days. Turbo Pascal 5.5 is required. Add $5 for shipping in U.S. and Canada. Elsewhere add $35 per unit. T.PRO customers may purchase O.PRO for $100 plus shipping. Include your serial number.

TURBO Power

Reader Service Number 194

many bits have been sent, it just waits to receive the character. It won't matter if both begin sending at the same time as long as the clock line is forced low before the keyboard sends its 10th bit. Either way, the computer wins.

The Software

Rather than present a complete program for a particular CPU, I'll stick to pseudo-code. See Figure 7 for the program structures I use.

For an overview of the 8251's internal registers, refer to Figure 8. See Figure 9 for the 8251 initialization.

Alternate Solutions

Since all the action takes place on just two lines, you could do everything "manually" if your system had a couple of spare I/O bits and lots of spare time. The software would be much more complicated since you would have to continuously monitor the clock and data lines.

One approach would be to leave the clock line low, raising it only when convenient and watching to see if the keyboard has anything to say. If no transmission begins within a reasonable time, lower the clock line again and go about your business. If an interrupt is available for the clock line, you could avoid timing loops, but you must service the interrupt quickly.

XTs can get by with just a shift register, but the XT data format only requires a start bit and 8 data bits.

You could use other serial chips, but the "old fashioned" 8251 with its dedicated RxRDY line is simple. Newer chips usually have a single interrupt line that triggers on several different events. This may not be a problem if you can always service interrupts quickly.

The 2651 is similar to the 8251. It costs more and may be harder to find, but timing is less critical and no chip clock is needed. If CPU overhead must be kept to a minimum, you could add more logic to control the write sequence completely.

You can only determine the ultimate solution for your system by carefully studying your needs. If you only need to receive, you can get by with a simpler circuit. A few spare gates or chips may also influence your design. And, of course, if you hand-select keyboards....

♦ ♦ ♦

Figure 8—8251 Registers

```
The control register is selected by high level on the C/D pin.
After a reset, the 8251 awaits a mode command. This is a single
byte. Once the mode command has been written, all subsequent
writes access the status register.

Mode register:
Bits 0-1:
     00 = Sync mode
     01 = 1X baud rate
     10 = 16X baud rate
     11 = 64X baud rate
Bits 2-3:
     00 = 5 bits
     01 = 6 bits
     10 = 7 bits
     11 = 8 bits
Bits 4:
     0 = Parity disabled
     1 = Parity enabled
Bit 5:
     0 = Odd parity
     1 = Even parity
Bits 6-7:
     00 = Invalid
     01 = 1 stop bit
     10 = 1.5 stop bits
     11 = 2 stop bits

Status register write:
Bit 0 = Transmitter enable when set
Bit 1 = /DTR low when set
Bit 2 = Receiver enable when set
Bit 3 = Send break when set
Bit 4 = Reset error flags when set
Bit 5 = /RTS low when set
Bit 6 = Internal reset, new mode command must follow when set
Bit 7 = Search for sync character when set

Status register read:
Bit 0 = Ready to send a character when set
Bit 1 = Character received when set
Bit 2 = Transmitter shift register empty when set
Bit 3 = Parity error when set
Bit 4 = Overrun error when set
Bit 5 = Framing error when set
Bit 6 = Sync/break detect when set
Bit 7 = /DSR is low when set
```

♦ ♦ ♦

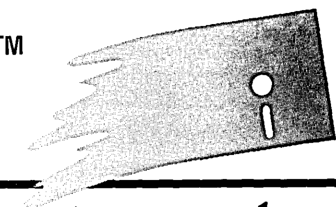
Figure 9—8251 Initialization

```
8251 initialization:
Set for 1X baud rate, 8 data bits, 1 stop bit, odd parity.
Set /DTR and /RTS high and enable the receiver.
You should now be ready to receive characters from the keyboard.

;;RX_CHAR subroutine
;;Wait for a valid character
;;Request re-tranmission if an error is detected

RX_CHAR
  UNTIL No receive errors
    Wait for a character
    IF an error was detected
      Send hex FE resend command
    END_IF
  END_UNTIL
RETURN

;;RECEIVE subroutine
;;Return a keypress
;;Ignore break (hex F0) codes
;;Shift key pressed or released updates SHIFT flag
```

```
RECEIVE
  FLAG=false
  UNTIL FLAG=true
    CALL RX_CHAR
    IF this character is a hex F0 break code
      Set /RTS low
      UNTIL not hex F0 break code
        CALL RX_CHAR       ;;F0 may be re-sent
      END_UNTIL
      Set /RTS high
      IF this is a shift key
        SHIFT = false
      END_IF
    ELSE                   ;;Make codes processed here
      IF this is a shift key
        SHIFT = true
      ELSE
        FLAG = true        ;;Got a keeper
      END_IF
    END_IF
  END_UNTIL
RETURN

;;TRANSMIT subroutine
;;Send a command to the keyboard

TRANSMIT
  IF /DSR input is 0  ;;Keyboard is sending if /DSR=0
    CALL RX_CHAR         ;;Wait for transmit from kbd
    Do something with keypress just received
  END_IF
  UNTIL character received is hex FA acknowledge
    Set /DTR low
    Disable 8251 receiver ;;will hear its own echo
    Enable 8251 transmitter
    Write character to 8251
    Set /DTR high
    Set /DTR low   ;;Clock start bit to serial output
    Delay 1ms with clock low in case kbd wants to send
    Set /DTR high  ;;Clock high, let kbd see low data
    UNTIL 8251 transmitter empty flag is set
      Check 8251 transmitter status
    END_UNTIL
    Set /DTR low to inhibit keyboard
    Delay 1ms while kbd forces data low to acknowledge
    Enable 8251 receiver
    Set /DTR high
    CALL RX_CHAR         ;;Get FA or FE
  END_UNTIL
RETURN
```

♦ ♦ ♦

QEdit™ ADVANCED



The fast, easy to use, fully featured text editor at an affordable price.

THE *Quick* EDITOR

If you are looking for the right combination of price and value in a text editor, then give QEdit a try. At ***ONLY $54.95*** and a money-back guarantee—you just can't go wrong.

QEdit is fast, easy to use, and simple to install. At the same time you get all of these features and more.

- Completely configurable, including keyboard and colors
- Edit as many files simultaneously as will fit in memory
- Open up to eight windows
- 99 scratch buffers for cut-and-paste or template operations
- Exit to DOS (or a DOS shell) from within QEdit
- "Pop-Down" menu system and customizable Help Screen
- Column Blocks
- Easy to use macro capability including keyboard recording
- Wordwrap and paragraph reformat capabilities
- Recover deleted text
- Automatic indentation for C programming
- Import files and export blocks
- Locate matching braces and parentheses
- Execute command line compilers from within QEdit
- QEdit supports 101-key keyboards, EGA 43-line mode, and VGA 50-line mode
- Great for use with laptops—QEdit edits files entirely in memory, saving drain on laptop batteries
- Compact—Even with all these features, QEdit requires less than 50k of disk space

Full 30 day money-back guarantee

System Requirements
QEdit requires an IBM PS/2, PC/AT, PC/XT, PC, PC/Jr, or compatible. Minimum system requirements are 64 KB of memory, PC-DOS 2.0 or MS-DOS 2.0 or greater, 50 KB of disk space. QEdit runs GREAT on floppy based systems and laptops.

PC MAGAZINE

“QEdit is a **great** piece of software. **Highly recommended.**”
John C. Dvorak, PC Magazine September 12, 1989

“QEdit is, without question, the smallest, fastest, most versatile text editor in the DOS world.”
P. L. Olympia, Ph.D., DBMS July, 1989

QEdit™ ADVANCED $54.95 ALSO AVAILABLE QEdit™ for OS/2 **$79.00**

To order direct call
404-641-9002

MasterCard VISA AMERICAN EXPRESS Cards

Add $3.00 for shipping—$10.00 for overseas shipping. **UPS 2nd DAY AIR available within the U.S. for ONLY $5.00**
COD's accepted—please add $3.00
Georgia residents add 4% sales tax

SEMWARE™
4343 Shallowford Rd. • Suite C-3
Marietta, GA 30062-5003

QEdit and SemWare are trademarks of Applied Systems Technologies, Inc.
© 1989 Applied Systems Technologies, Inc.

Reader Service Number 127

Filling In The Holes On Your XT

Adding Memory The Cheap Way

Want an extra 128K to house TSRs or for a RAM disk? Want a battery backed memory space? Stay tuned. This project is classic Micro C. Cheap.

No, I'm not talking about your XT's ventilation holes, I'm talking about those interesting gaps in its memory map. You can use them quite profitably.

Most of you know that, even though the 8088 can address up to 1 megabyte, DOS limits you to the first 640K. The 8088 has what is known as a segmented architecture, and we refer to the segments as 0000:0000, 1000:0000, and so forth, up to F000:0000. For convenience, let's call these the 0 block, 1 block ... F block. The first ten blocks (blocks 0 through 9) access the normal 640K.

What's The Rest Of It For?

The remainder of the addressable memory has been set aside for special purposes. In the B block we find the space for video, both monochrome and CGA. Right in the middle of the C block is where the hard disk controller BIOS lives. Finally, up in the attic at the F block we find the ROM BIOS and, if you have a true-blue XT, ROM BASIC.

On a monochrome XT, blocks A, D, and E are empty (an EGA or VGA system will also use the A block and the beginning of C block). Anyway, at least D and E blocks (128K) aren't doing anything.

How Can I Get At It?

I had known all this, in a vague way, for some time, but I didn't think much about it until my last job (working with Kaypro PCs). These PCs used three rows of 256K RAM chips and mapped the upper 128K into blocks D and E. It wasn't long before I found out how handy it was to have a private chunk of memory.

Since I don't have a Kaypro at home, I began casting about for a way to implement this feature. It had to follow Shannon's Rules of Projects:

(1) It must be cheap.
(2) It must be easy to do.
(3) It must have a good chance of working.
(4) It must be cheap.

Did I mention it shouldn't cost very much? Well, it boils down to a classic build-or-buy decision. Naturally, I decided to do both.

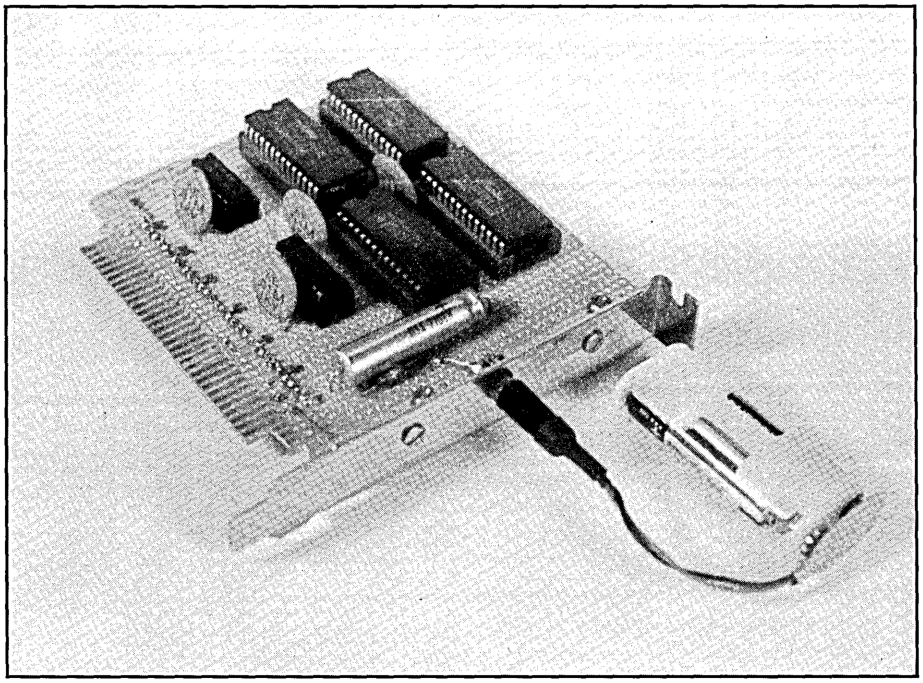
What We Have To Work With

The makers of the IBM PC (and the clone folks) have thoughtfully provided us with access to all the signals and control lines we'll need; we call them slots. Figure 1 lists all the significant lines.

The Home Brew Approach

Since I had a couple of 32K static RAMS on hand, I decided to take the build-a-board approach. I priced out the standard prototype boards, but they're a little pricey and have much more room than I needed. Then, whilst wandering around my local Radio Shack store, I noticed they have a new board (Part Number 276-192) which has the proper finger spacing (0.1") and more contacts (72) than needed (62). It's only big enough to make a half card, but that's big enough for me. Also, it's only $5.

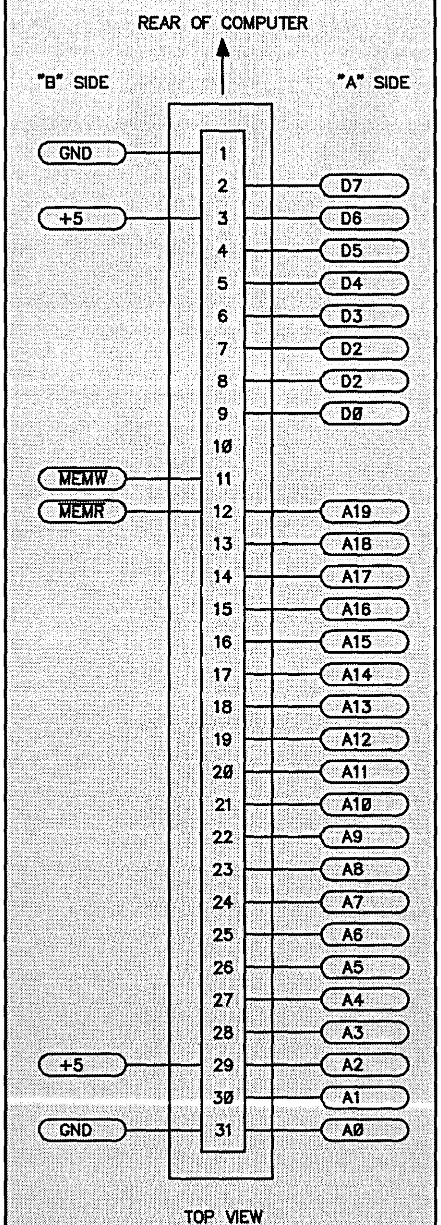
You have to nibble off the extra contact fingers (don't use your teeth) and cut the board down to size (use another half-size card as a template), but this takes only a minute or two. Fashion some sort of attachment to a spare blank



Board Photo by John Zack.

By Larry Shannon
5615 Truscott Terrace
Lakeview, NY 14085

Figure 1—IBM PC/XT "Slot"



The makers of the IBM PC (and the clone folks) have thoughtfully provided us with access to all the signals and control lines we'll need...

bracket; I used a short piece of ½" aluminum angle and #4-40 hardware.

After setting up the board, the rest of the project went swiftly. I obtained two more static RAMs and the junk box yielded some wire wrap sockets.

The circuit is very simple (see Figure 2). I've done no buffering since my motherboard (a Wavemate Bullet) buffers all the slots. The IBM is not buffered, except on slot J7, so be a little careful. I've tried my board on an IBM XT with no problems. The additional complexity of buffering would have edged it out of the simple project class, so I didn't do it.

Some Details

IBM thoughtfully brought out memory read (MEMR) and memory write (MEMW) to separate pins. I figured that if a memory access was going on, one of them would be low, so the output of the 74LS00 nand gate would be high. If no memory requests were going on, both would be high, resulting in MEMRQ going low. This would gate off the 74LS138 decoder and also Read/Write. By inverting MEMR I get MEMW. A little bit backwards, but it works fine.

Address lines A18 and A19 are always high when accessing this part of memory so I used them to generate the negative gating signal for the decoder. The '138 generates chip select for blocks D000, D800, E000, and E800—one CS for each chip.

The spare gate on the '00 can be used with another '138 for selecting the A block region.

There are a lot of spare select lines doing nothing, but the chips cost about 50¢ apiece so I figured it wasn't much of a waste.

Tips From The Top

Some tips I either figured out or stumbled onto:

(1) Most of the connections are to the "A" side of the slot. Orient the board so the solder side faces the "B" side, and use a strip of single post headers. Solder through and you can now wire to a firm post.

(2) Spot solder the sockets a few places to anchor them, but don't solder all the pins. Not necessary.

(3) Use wire-wrap wire (#30) and tweezers to wrap the wires around the posts. After you've soldered all wires to a pin, clip it short. If you wire wrap them, the board will wind up too thick and will slop over to the next slot.

(4) Use different colored wire for data lines, address lines, etc. Much easier to debug. Wire all the same-color connections at once.

(5) Take frequent breaks! You can go buggy peering at those teeny wires for too long.

(6) I built the board to cover the D and E blocks first; one more 74LS138

decoder gives me chip selects for the A block. You can add this on later. The additional memory chips can be piggybacked on existing chips. Solder all legs except chip select (pins 20 and 22).

Connect the supply voltage to the TTL chips if you're going to battery back up the board. You need not back up the A block.

The Commercial Approach

At one time, IBM apparently sold an add-on memory card that could be mapped almost anywhere, and clone manufacturers followed suit. This was the so-called 64/256 memory card and is now considered obsolete. To you and me, obsolete means cheap!

After carefully studying *Computer Shopper*, I located two sources. Electrified Discounters sells the board fully populated for $69, and American Design Components sells the board and memory separately. Don't overlook flea markets and swap meets. You can recognize the board by its three rows of twelve memory sockets and the DIP switch in the upper right-hand corner.

Figure 3 shows the dip switch configurations and memory requirements for either a 64K chunk for the A block or a 128K piece for the D and E blocks. Unfortunately, it isn't possible to map both onto a single board. If you're using EGA, you probably can't use the A block anyhow.

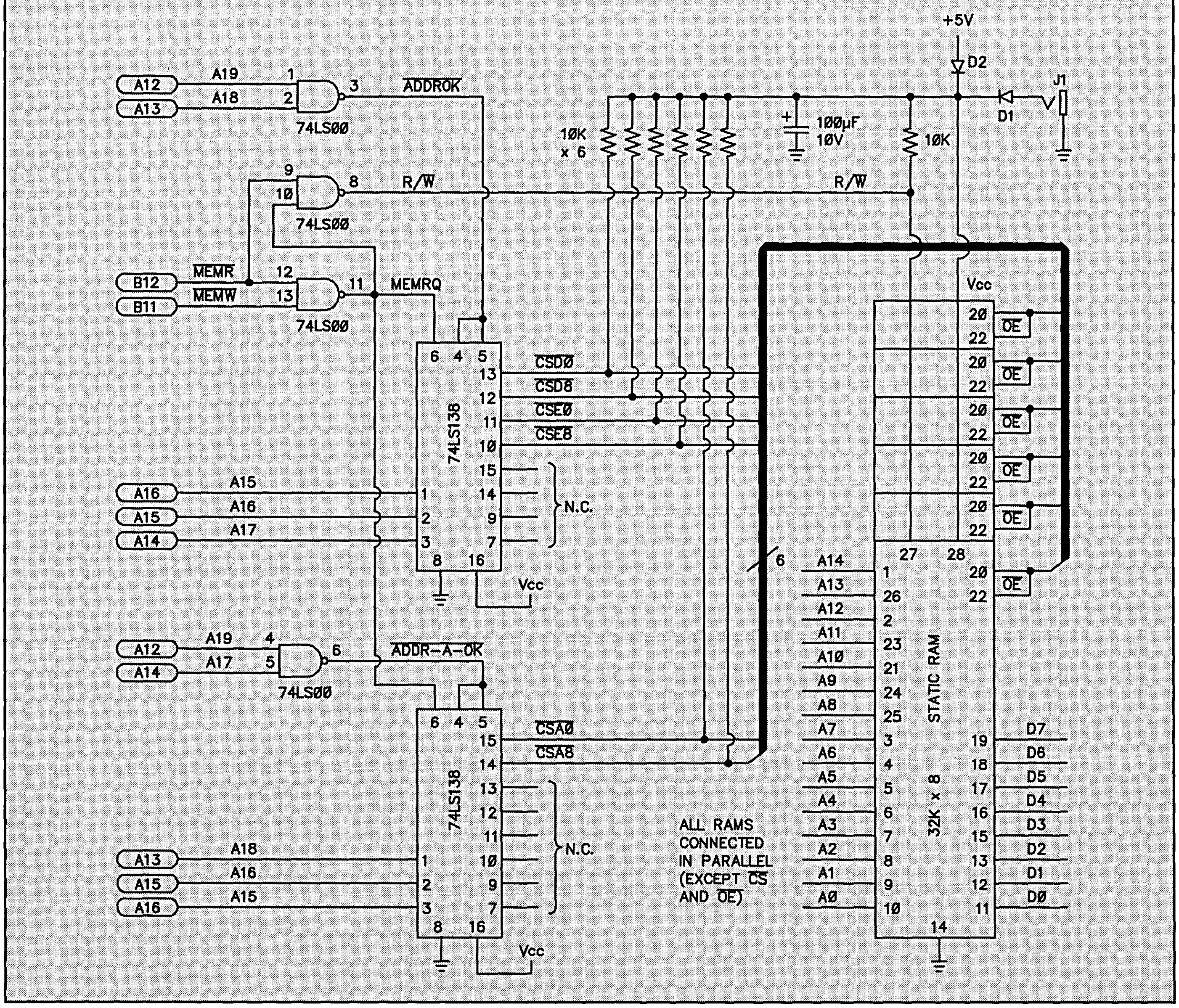
Prime It!

These boards use nine chips per bank (with parity). DOS does not initialize anything above 640K, so if you try to use it you'll get a "parity check" message. The solution is to write up there before reading anything. I have a little utility that fills a given block with zeros and I run it in the AUTOEXEC file.

How Fast Is It?

This commercial board is fully buffered so there should be no problem with any machine. But, because of their delay structure, they only run at a pokey 4.77 MHz. This is okay, too, 'cause most clones run their I/O chan-

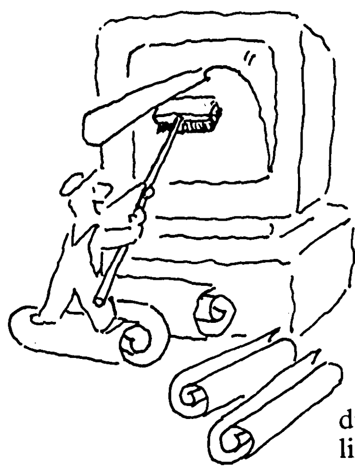
Figure 2—RAM Block Schematic



Two great tools.

NEW VERSION 2.8
CLOCK & EDIT UNITS, DIALOG BOXES, AND PROGRESS BARS!

SAYWHAT?!
The lightning-fast screen generator.



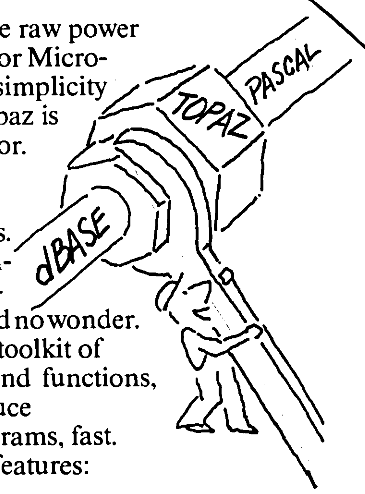
Whether you're a novice programmer longing for simplicity, or a seasoned pro searching for higher productivity, you owe it to yourself to check out Saywhat. You see, with Saywhat, you can build beautiful, elaborate, color-coded screens in minutes! That's right. Truly *fantastic* screens for menus, data entry, data display, and help-panels that can be displayed with as little as one line of code in *any* language.

Here's what you get:

- Design screens, windows, and *moving bar menus!*
- Easy-to-use, powerful editor lets you create screens in a jiffy.
- Pop up your screens and menus with one line of code in dBASE, all the dBASE compilers, your favorite BASIC, Pascal, *or any other language!*
- Screen Library Manager.
- Generates runtime code.
- No runtime license or royalty fees.
- Comes with a 100 page manual, plus dozens of sample programs and free utilities.

$49^{95}

TOPAZ
The breakthrough DBMS toolkit for Pascal



If you'd like to combine the raw power and speed of Turbo Pascal or Microsoft's QuickPascal with the simplicity and elegance of dBASE, Topaz is just what you're looking for. That's because Topaz was specially created to let you enjoy the best of *both* worlds. The result? You create complete, truly dazzling applications in a very short time. And no wonder. Topaz is a comprehensive toolkit of dBASE-like commands and functions, designed to help you produce outstanding, polished programs, fast. Check out these powerful features:

- Over 220 routines all with easy-to-use, dBASE-like syntax.
- Data entry routines like **SAY, GET, PICTURE, RANGE,** color selection, unlimited data validation.
- Open up to 10 DBF files, with up to 15 indexes with **USE, SELECT, SKIP, APPEND, PACK, INDEX ON, SET INDEX TO,** and **FIND.**
- No need to buy dBASE. **CREATE, BROWSE** and **REPORT** utilities included.
- Easily implement Saywhat and Lotus-style moving bar menus.
- **BROWSE** or **EDIT** any DBF file with just one line of code! Programmable and windowed too.
- Pick from windowed data or file-names with one line of code.
- Comprehensive Time & Date math in 7 international formats.
- Powerful code and report generators included!
- Comes with a complete 320 page manual, plus sample programs to get you started.

$74^{95}

ORDER NOW. YOU RISK NOTHING.
Thousands of satisfied customers have already ordered from us. Why not visit your dealer or call toll-free, right now and put Saywhat and Topaz to the test yourself? They're fully guaranteed. You don't risk a penny.
Special limited-time offer! Save $26. Buy Saywhat?! and Topaz together for just $99 (plus $5 shipping and handling, $10 Canada, $25 International, Calif. residents add 7%).

Visit your nearest dealer
or call toll-free:
800-468-9273
In California: 800-231-7849
International: 415-571-5019
Software Science, Inc.
100 Valley Drive, Brisbane, CA 94005

MONEY BACK GUARANTEE.
If you aren't completely delighted with Saywhat or Topaz, for any reason, return them within 30 days for a prompt, friendly refund.

Dealers: SAYWHAT?! and TOPAZ are available from Kenfil Distribution, and in Europe from ComFood Software, W. Germany 49-2534-7093

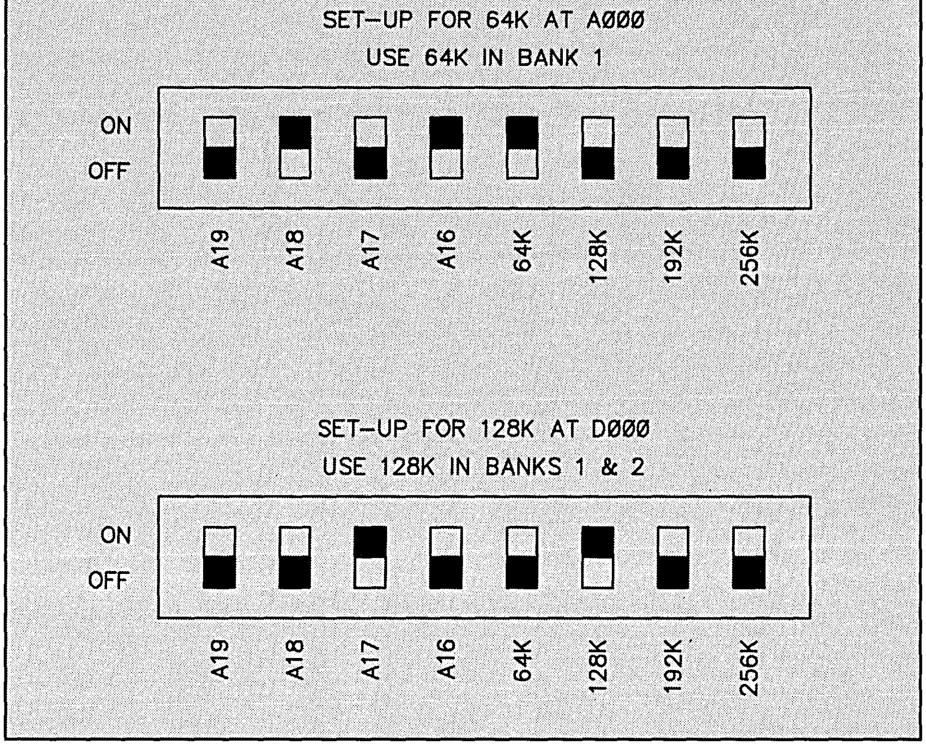
Guaranteed!

S O F T W A R E S C I E N C E I N C .

nels at this speed to ensure compatibility.

The home brew board can run up to 10 MHz or so, depending on the speed of the SRAMs you get, so you can also use it on an AT clone. While nobody seems to spell it out, you can run 8-bit cards in the 16-bit slots of an AT. It checks its auxiliary slot to determine if you have a 16-bit board.

Figure 3—DIP Switch Settings



Now That I've Got It, What Do I Do With It?

Actually, I'm presenting this backwards; I had definite plans for the extra memory and then went about finding a way to get it. Now, I'm showing you how to build it and then telling you at the end what to do with it.

Offhand, I've found one poor and three very good uses for this virgin territory.

(1) Use it as an extra data storage area. This is the poor reason. It's perfectly doable, of course, but it requires some extra programming and hardly seems worth the effort. We'll say no more about this option.

(2) Make a RAMDISK. This makes sense. If you're using MS-DOS version 3.2 or so, you got a program called VDISK. This can be used to establish a RAMDISK in the D and E blocks. By the way, the IBM version of VDISK (in PC-DOS) won't let you do it.

While this is an excellent use for this newfound memory, there is an even better method. The public domain program EDISK is specifically written to set up a RAMDISK in the D and E areas. It has the additional virtue that it will survive a reboot!

As I do a lot of assembly programming, I'm always crashing the system. With this program, I just reboot and my files are still there. In fact, the way my BIOS works, this RAMDISK even survives a reset, so I'm really protected.

Since I use static RAM to cover the D and E blocks, I modified EDISK slightly (to EDISK-S) for use with a battery backup system I'm building. With this in place, it will survive everything, even a power off. You can't imagine how handy that is when you're debugging code that plays around with the system. Dynamite stuff.

(3) Load your TSRs up there. This notion occurred to me after I played with the board for a few days. I usually run with SideKick and CED (command editor) in place, a good size print spooler, WAITASEC (scrollable screen buffer), and a handful of miscellaneous other TSRs. All told, they took up over 200K of my precious DOS memory. So... I had to find a way to put these guys out of the way and ease the dreaded ramcram.

I dood it. It wasn't really that tough, but you have to be careful in controlling the revectoring a typical TSR does. If it wants to capture INT 21H, which it also uses to reset the vectors, the self reference can get out of hand!

Anyway, it's done. Now my TSRs politely get out of the way and I make do with 576K of available RAM for DOS.

(4) Add more DOS memory—It seems that an old trick is to take memory in the A block and fake DOS into thinking it has more than 640K (704K, actually). I have some PD software that purports to do this and I would dearly love to make it happen. But my BIOS is such that I haven't figured out how to make it work. You're welcome to try on your machine.

Battery Backup

I've always found battery backup schemes more complicated than they need be—at least in my opinion. So, in line with my "keep it simple" philosophy, I wired up the circuit of Figure 2. The jack J1 was scrounged from an old transistor radio. I used that rather than a coin battery because I didn't have a holder (or a battery). Besides, I didn't know how much current these little fellows would draw under standby conditions.

The XT bus has two 5 volt pins. I used one for the TTL chips and the other for the RAM. This separation made it easy to wire in the backup circuit. You don't want to provide back-up power for the TTL chips!

Diode D1 blocks the system 5 volts from back-charging the battery, and diode D2 keeps the battery from trying to run the whole computer. You can use any diode for D1. D2 has special requirements, and thereby hangs a tale.

Silicon Follies

I initially used a little silicon diode for D2, but operation of the board was erratic: "bad sector" messages at random, etc. When I measured the voltage it turned out my system voltage was 4.86 volts. This is within spec and everything works reliably, but the voltage drop across the silicon diode was about .5 volts. So the operating voltage on the RAMs was about 4.4 volts. This is out of spec and caused weird things.

I assumed a germanium diode would solve the problem, but the current load worried me. The spec sheets talked about 60 mA operating current. With 4 RAM chips, that's nearly a quarter of an amp.

I had visions of popping noises and little wisps of smoke. I don't mind light emitting diodes, but I didn't want a flame emitting diode!

But I Measured It

So... get out the old Simpson and let's see. What's this—100 microamps? Hmmm. Let's try reading and writing to it. Aha! The needle bounces around when there's memory activity, but otherwise almost nothing. Apparently, that 60 mA figure only applies to read/write action. It appeared a germanium diode would handle the steady state load no problem, and a small electrolytic helps with the pulse load.

By the way, I still had to select diodes to get one with low forward voltage drop. Old style 1N34s won't hack it. I had a batch of unknown computer diodes and tested them until I found some with less than 0.25 volt drop. The system now perks along happily with about 4.7 operating volts. If your system board is closer to a nominal 5 volts, you may not have this problem.

And To Cap It Off...

Originally I had used a 10 µF capacitor as an aid to smoothing out the current peaks when writing, and to save the diode as explained above. It turns out that with this relatively skimpy amount of capacitance, some memory positions would change during the power off/on cycle.

A RAMdisk that is a "little bit unreliable" is just like a "little bit pregnant." Drove me crazy until I thought of beefing it up. Using 2000 µF worked but was a bit of overkill; 100 - 200 µF is plenty.

(Editor's note: Tantalum electrolytics are definitely better than standard because of their low impedance.)

...The Pièce de Résistance

The 10K resistors ensure that the chip select and write lines will be nailed high except when driven low. When the power goes off, some of these lines have a tendency to float. Before installing the resistors, I noticed that the chips would idle at several milliamps and wouldn't hold data worth beans.

Quarter-watt resistors are fine; use a resistor pack if you have one, but it's easier to thread separate leads among the wires. The value isn't critical, but I wouldn't go much below 5K.

Final current draw in backup mode is about 2 µAmp. I still don't have a coin battery or a holder, so I put two AA cells in a little holder and connect it to the jack J1. Makes it easy to replace batteries without removing the case. They should last a long time.

A RAMdisk that is a "little bit unreliable" is just like a "little bit pregnant."

Warning! Warning! Warning!

This battery backup system is rather simple minded though I've been using it with no problems for some time. But, you never know.... Manufacturers specify power up and power down sequences for static RAM and I'm probably violating at least some of the rules. (An engineer from Hitachi told me that if I used HCT instead of LS TTL chips, I'd have less trouble with RAM losing its memory. But I haven't had any problems.)

An Embarrassment Of Riches

It's just not possible to put everything in just 192K; I can't put all the TSRs I want up there and still run the nonvolatile RAMDISK, too. Small batch files rename preset AUTOEXEC and CONFIG files so I can set the machine up as I need it. You'll probably think of other ways to do what you want. The possibilities are endless.

Where To Get The Stuff

As mentioned, the boards for the home brew memory are available from Radio Shack, as are the sockets and wire. The static RAMS are available from any number of sources—look for the best prices.

The speed doesn't matter on an XT; they don't make static RAM slow enough to make a difference. If you build the board for an AT, 120 ns parts should work on an 8 MHz bus with no wait states. Try to get low-power (LP) versions if you want to make the board battery backed up. Don't forget flea markets.

The commercial boards (XTs only) are available from the sources mentioned and surely other places. And 64K memory chips (200 ns is okay) should be a drug on the market. I've seen them for less than a buck apiece.

Software

All (non-commercial) software mentioned above is available from me for $5 and a blank floppy. It includes source (where I have it) and ready-to-run .COM files. Lengthy .DOC files describe what you need to know to set them up in excruciatingly minute detail. *(Editor's note: You can also find the software on the Micro C BBS (503-382-7643) or on the Issue #52 disk, available from Micro C for $6 U.S., $8 foreign.)*

Incidentally, the software I wrote was written for Eric Isaacson's A86 assembler. It's no big deal to make it ready for MASM. But A86 is so much slicker and faster than MASM, there's no real reason to do that. If you haven't tried A86 you're in for a very pleasant surprise. While this article is not about A86, I'm so hot on it that I'll send you the latest version (v3.19) for another $5 and a blank floppy.

Sources Mentioned

American Design Components
815 Fairview Ave.
P.O. Box 220
Fairview, NJ 07022
(800) 524-0809
(201) 941-5000 (in NJ)

Electrified Discounters
1066 Sherman Ave.
Hamden, CT 06514
(203) 287-1976

Static RAMs - 43256 etc.

I've seen these as low as $10 each (120 ns). Check *Computer Shopper* for suppliers.

♦♦♦

Logic Families

Selecting The Right Chip For The Job

I remember standing in a computer hardware lab one sunny afternoon watching the instructor field questions. He handled everything with the aplomb of years of practice, until a fellow asked why a 74ALS10 wouldn't work as a replacement for a 7410. "They're different," the instructor mumbled, "I think."

Back in the old days, you used 7400 series TTL (Transistor Transistor Logic) chips when you wanted to build a logic circuit. They were all you had. If you wanted a 7404 hex inverter, you ordered a 7404 hex inverter.

Now, however, it's not that simple. You can select from the 74L series (Low power), 74H (High power), 74C (CMOS), 74F (Fast), 74S (Schottky), 74LS (Low power Schottky), 74AS (Advanced Schottky), 74ALS (Advanced Low power Schottky), 74HC (High speed CMOS), 74HCT (High speed CMOS with TTL input levels), 74AC (Advanced CMOS), and 74ACT (Advanced CMOS with TTL input levels). Of course, I haven't even mentioned the CD4000 or the HE4000 CMOS families, or even ECL (Emitter Coupled Logic)....

All the chips in the 7400 families have the same pin assignments and do the same logical functions. Sometimes, though, the chips aren't direct replacements for each other because of different electrical properties. For instance, TTL logic voltage levels are lower than CMOS levels, and internally the chips work a lot differently.

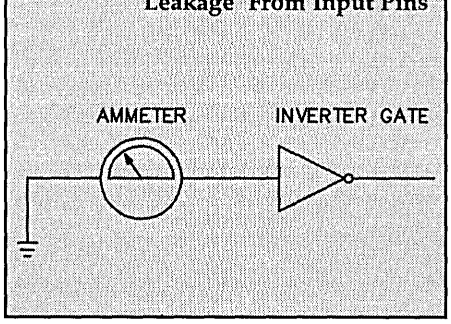
However, you could safely replace that 7404 inverter with a 74S04 (for speed) or a 74LS04 (lower power) or a 74HCT04 (for speed and lower power).

Three Kinds Of Power

When you talk about power usage, you have to consider three kinds of power. The first two are obvious. They are:

1. The current that is sent out the chip's outputs.
2. The current the chip uses for its own operation. (Usually called static or quiescent current.)

Figure 1—Measuring Current Leakage From Input Pins



Part of the reason chips draw static current is that transistors aren't perfect. (Many of us have this problem). Theoretically, transistors are switches that turn on and off. In reality (oh no, not that), transistors are slightly on even when they're off.

On CMOS chips, the leakage of current through the transistors accounts for almost all the quiescent power usage. (CMOS transistors are particularly good switches.) The older TTL chips not only have older style transistors, but their designers threw in a few resistors and whatever other leaky odds and ends they happened to have at their benches.

So, a TTL 7404 at rest can draw as many as 30 milliamps; a 74HC04 draws only 2 microamps (1⁄15,000 as much).

When I mentioned that 7400s leak, I wasn't just talking about current trickling out of the outputs or through the innards; 7400s even leak out their inputs. (Hardly box trained, if you ask me.)

I ran an experiment (see Figure 1) and measured 1 milliamp coming out of an input pin from a 7404. I then checked the LS version. The current coming out its inputs was about ⅕ as much. I also ran the experiment for CMOS. I saw some leakage, but it was less than a microamp (1⁄1000 as much).

Sinking Fast

This current leakage means that if you want to pull the input of a 7404 to ground (logic 0), whatever you connect to pull that input down has to draw 1 milliamp away from that input. We call this "sinking" current because it involves taking someone else's current and sinking it to ground. Not only do you have to drain the current away, if you want to run fast, you have to be quick about it.

According to the Texas Instruments *TTL Databook*, the 7404 and 74L04 use the same arrangement of transistors, diodes, and resistors (see Figure 2). The only difference between a 7404 and a 74L04 is that the resistor values in the L04 are ten times as high as they are in the 7404. The circuit is the same, but the higher resistor values allow only 1⁄10 the current to pass through.

Unfortunately, TTL needs a lot of power to run fast; the more power it has, the faster it can "clamp" or settle all the transistors that have to switch. When you have less current, things take longer to fill up and settle down. So the 74L family, with only 1⁄10 the current, also runs at only 1⁄10 the speed of regular TTL (which is already bloody slow).

In this same vein, the 74H family uses twice as much power as 7400 and goes twice the speed. TTL is already a power hog, so the only people who use the H series are toaster manufacturers.

Where The Input Leakage Comes From

If you look at the schematic for a 7404,

By Nathan Engle
6465 Piping Rock Lane #30
Indianapolis, IN 46254

the source of the current out the inputs is obvious. The 7404 has a pullup resistor built into every input. When somebody tries to pull the input to ground (for instance, through an ammeter), current flows through the pullup resistor and the transistor base. (Look at Figure 2 again.)

CMOS chips differ in a couple of ways. One is that the transistors used in CMOS have a very high input impedance. This limits the current flow between the input (base) of the transistor and the other parts. The transistors in TTL chips aren't like that, so they allow a lot of juice to slosh around.

Another difference is that CMOS chips don't pull up their inputs. The only current you have to drain away to pull a CMOS input to ground is the charge stored on the base of the input transistors—plus whatever tiny amount of current manages to trickle through from the rest of the circuit. This means that CMOS chips let their inputs dangle; they don't guarantee them to be either 1 or 0 when they're disconnected. If you want them to be something, it's *your* responsibility.

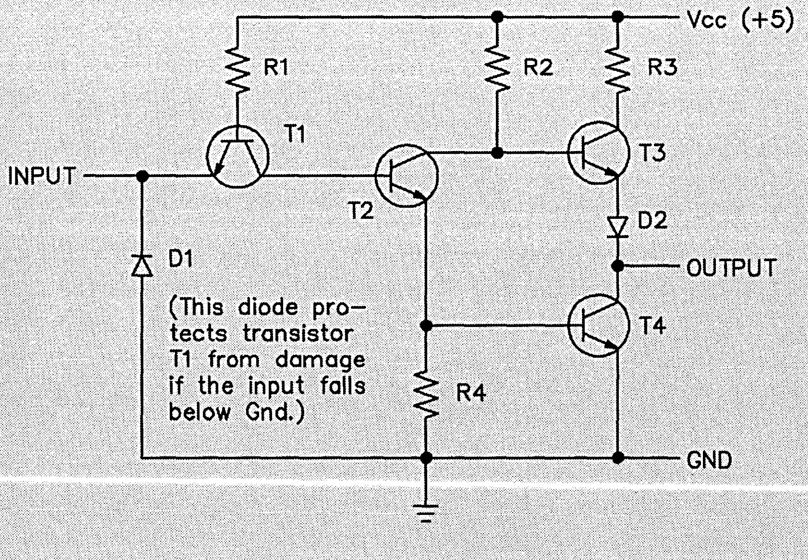
Theoretically, transistors are switches that turn on and off. In reality (oh no, not that), transistors are slightly on even when they're off.

And you *do* want to connect them to something; if you don't, unpleasant consequences can occur. When that happens, you start to have problems with the other kind of power.

The Other Kind Of Power (Add "Twilight Zone" Music As Appropriate)

The other power consumption is called "dynamic" usage. Because of the way output drivers work, all these logic families (both CMOS and TTL) consume

Figure 2—Schematic For the 7404 and 74L04.


Vcc (+5)
R1
R2
R3
T1
T3
INPUT
T2
D2
D1
OUTPUT
(This diode protects transistor T1 from damage if the input falls below Gnd.)
T4
R4
GND

	R1	R2	R3	R4
7404	4K	1.6K	130 Ω	1K
74L04	40K	20K	500 Ω	12K

Note: If the input is pulled down, current will flow through R1 and T1 and eventually out the input pin.

a little extra gulp of power when their outputs switch from 1 to 0, or vice versa (see Figure 3).

The extra gulp is due to each family of logic chips having characteristic trigger voltages that define what it's going to call a 0 and what it's going to call a 1. For CMOS families, the 0 trigger voltage is 1.35 volts (or 30% of V_{cc}); anything less than 1.35 volts is a logic 0. The 1 trigger voltage is 3.15 volts (or 70% of V_{cc}); anything above 3.15 volts is a logic 1.

A grey area exists between 1 and 0 where there's no guarantee the input will be either 1 or 0. We sometimes refer to this area as the "linear region" because, when an input is in this region, the output driver transistors begin to exhibit linear characteristics (like amplifiers instead of switches). For several reasons, linear mode is something to avoid.

For example, when an input voltage falls in the linear region, both the high and low output transistors are partially turned on (Figure 3c). When this happens, it creates a low resistance path connecting +5 volts and ground. When the resistance drops, current flows through the driver transistors. The output drivers will gulp current at this increased rate as long as the input remains in the uncertain region or until the chip burns out.

We Got Rhythm

Another consequence of linear mode operation is oscillation. We start with almost the same setup as before. The example is a CMOS inverter gate. But instead of a dangling input, this time we use a 2.1 volt source so we know that the input will fall in the linear region all the time. This makes both output drivers turn on (partially).

As before, when both driver transistors switch on, the chip's current usage jumps quickly. In fact, the current jumps so quickly that the local reference voltage levels for V_{cc} and ground get distorted.

Chip leads (the wires that come out of an IC chip package) cause this distortion by having something called "parasitic" inductance. Any wire that has current moving through it generates a magnetic field. The magnetic field causes the wire to resist increases and decreases in current flow. For chips like logic circuits, the current flow can change a lot. Whenever it does change, parasitic inductance can cause a sort of bouncing effect.

This effect is usually called ground bounce, even though it occurs in the V_{cc} level as well. This naming convention probably arises because most of these logic families use ground as their reference voltage.

The bouncing caused by the switching gulp can be bad enough to make a chip malfunction. When the internal reference voltages are distorted, a chip can be fooled into thinking that its input level is 1 or 0, even when the outside world thinks it's somewhere between.

If a bouncing reference voltage fools a chip into believing that its input is either a 0 or 1, it shuts off one of the output drivers. Almost immediately, the internal voltage levels begin to return to normal. When the reference levels are close enough to normal, the chip realizes that the input is *not* 1 or 0, so both output drivers turn on again.

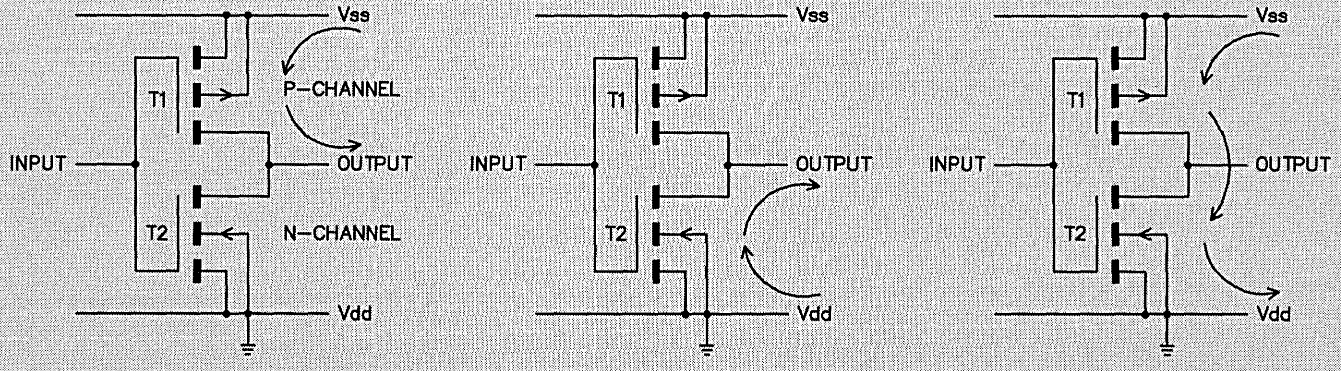
This brings back the current surge, and the day wears on. Depending on things like the amount of charge required to turn on the output drivers, the rate of this oscillation can go as high as 100 MHz.

Bypass Capacitors

Current spikes caused by switching output drivers are the reason you have to add bypass capacitors to digital circuits. Placing a capacitor between the power and ground of each chip is like putting a tank on a water heater. When you take a shower, all the water comes from the utility company. If you want to shower with hot water, you have to store enough hot water beforehand.

Capacitors store a charge the way a water heater tank holds water. When a chip's output drivers switch and the current flow surges, the bypass capacitor supplies the extra current to the chip. Without the capacitor, the ripples caused

Figure 3—CMOS Inverters (In Three Different States)



A
Low inputs cause T1 to conduct so the output is +5v.

B
High inputs cause T2 to conduct so the output is 0v.

C
Dangling inputs can allow both T1 and T2 to conduct, which makes the chip "gulp" current.

by output switching could affect other chips the same way they affect the one that's switching.

One sideline point: you connect the bypass caps between +5 and ground at every IC, trying to keep the cap's leads as short as possible. Often you can't get the caps the same distance from +5 and ground. That's okay. A good rule of thumb for bypass caps is to place one as close as possible to every ground pin. The distance to +5 is less important (but still keep it short).

Schmitt Triggers

This linear region problem has been around for a long time. It's been present in every family all the way back to TTL, so chip designers have had some time to think about it.

They've come up with a solution. If you know you're going to have an input to your circuit that will be in the linear region very long, you can use chips with a special kind of input called a Schmitt trigger.

A Schmitt trigger is an extra step that gets added to a normal input to make the output waveform cleaner. If you try to use normal inputs with slow rising or dropping signals, the output of the gate can oscillate or jitter during the time the input lies in the linear region.

But with Schmitt triggers, the extra step lets the input trigger once. After that it clamps out any jittering until the input has settled to a real 1 or a real 0. Effectively, it's a guard that takes control when the input is in the linear region; it also substitutes a clean, correct output signal for the garbage that I normally see coming out of things I design.

When I first heard about Schmitt triggers, I thought, "They must not have a linear region, eh? Just switch at 50% V_{cc} and that's that." Well, it's not quite that simple. An application note in the *Signetics HCMOS Designer's Guide* burst my bubble. The Schmitt stage still has a linear region, and current use still goes up when the input is in it. But the current use is less than it would be for normal chips, mostly because of what's happening (or to be more exact, what's *not* happening) to the rest of the chip.

(Engle's Axiom: Always read the application notes. You get a whole bunch of examples and design tips, intermixed with discussions of things you haven't considered. This isn't school anymore; all exams are now open book. You got crib notes? Use 'em!)

In several places here I've griped

CITIZEN PRINTERS

Citizen 120D 120/25 CPS 80 COL	$165.00
Citizen 180D 180/35 CPS 80 COL	$180.00
Citizen HSP500 300/66 CPS 80	$350.00
Citizen HSP550 300/66 CPS 120	$475.00
Citizen GSX140 200 CPS 24 PIN	$345.00

GOLDSTAR MONITORS

MBM-1210A 12" TTL AMBER	$ 90.00
MBM-1401A 14" Flar Screen TTL	$135.00
MCH-1420 14" EGA 640x350	$345.00
MCH-1430 14" VGA 640x480	$375.00

VIDEO CARDS

ATI VIP Card VGA,CGA,EGA,MGA, ANALOG AND TTL 800x560	$175.00
ATI VGA Wonder 256 1024x768	$225.00
ATI VGA Wonder 256 16 bit	$375.00
Paradise Basic EGA 640x350	$ 99.00
Paradise Basic VGA 640x480	$175.00
Paradise VGA Plus 800x600	$199.00
Paridise VGA Plus 16 Bit	$275.00

HARD DRIVE CONTROLLERS DOR XT & AT

WD WX1 Auto Config PC/XT	$ 65.00
WD WX2 Auto Config PC/XT	$ 55.00
LONGSHINE LCS-6210D PC/XT	$ 45.00
WD WAH AT 16 Bit 2 Hard Drive	$ 75.00
WD 1003SM1 AT 2:1 Int.	$ 99.00
WD 1003SM2 Hard/Floppy 2:1	$125.00
WD 1006SM1 AT 1:1 8K Cashe	$125.00
WD 1006MS2 Hard/Floppy 1:1	$135.00
WD 1003SR1 RLL 2:1 Int	$125.00
WD 1003SR2 RRL Hard/Floppy	$135.00
WD 1006SR1 RLL 1:1 8K Cashe	$135.00
WD 1003SR2 RLL Hard/Floppy	$145.00

SEAGATE DRIVES AND CONTROLLER IN STOCK
Please call for current price.

********** 2400 BAUD MODEM **************
by Computer Peripherals
2400 Baud Internal and Software $85.00
2400 Baud External and Software $140.00

CASCADE ELECTRONICS, INC.
ROUTE 1 BOX 8
RANDOLPH, MN 55065
507-645-7997

Please ADD shipping on all Orders
COD Add $3.00 Credit Card ADD 5%
MN Add 6% Sales Tax Subject to change

Reader Service Number 15

about how you should connect unused inputs on CMOS chips. I'm lazy; I don't like to do it. But to be completely fair, I think I should also mention that every TTL application note I've read advises you to "terminate" unused inputs. That means just connecting them to either +5 volts or ground so that they'll have a defined level.

If you follow the rules, you should do the same thing for TTL designs that you do for CMOS. The difference arises when, for whatever reason, you connect something wrong or forget to connect it at all. TTL families may still work. CMOS probably won't. And that's not all. Sometimes the CMOS chips will go into a sort of auto-self-destruct mode....

Latch-up

"Latch-up" is another problem and it happens with all the CMOS families, though it's not as bad with new ones as it was in the old days. (I've read that this happens to some TTL families, too, but I've never had it happen to me.)

CMOS logic chips are built out of circuits that have been squished down onto a silicon substrate, so they're very small and close together. The components are made by alternating layers of metal oxide and silicon on top of a substrate. When you put them all together, they make a sort of 3D circuit.

Unfortunately, the silicon in the space between components can act like a transistor—an unintentional or "parasitic" transistor. (Gives you an idea what chip designers think about them.) Under normal conditions, they don't do anything.

Things like very high supply voltage levels and application of negative voltages to output pins cause them to appear. In general, anything that makes current flow in the wrong direction turns them on.

Once the parasitic transistors switch on, they allow so much current to pass through the chip that the condition gets locked-in, even when you've removed the original cause. The current flow can go up into the hundreds of milliamps (but not for long). It can be more than enough to destroy the chip.

One way to tell that a chip is latched up is that it will be *very* hot to the touch. CMOS logic chips usually run at room temperature, unless they happen to pass large amounts of current (i.e., they're latched up). Of course, by the time you locate the problem this way, the chip will probably be long gone.

Free Advice On Logic Chips

I'm calling this section free advice because, like most free advice, it's worth just about what you paid for it. However, after using some of these families, I have some impressions and opinions about them.

My favorite logic family for prototyping is 74LS. LS runs on about the same amount of power as 74L, but runs as fast as normal 7400 TTL. LS uses more power than CMOS, but the pullup resistors on 74LS inputs can save a lot of headaches if things aren't connected exactly as they should be. Maybe most important of all: LS is very common and, therefore, fairly inexpensive. That means you don't have to pay so much for replacements when your design starts talking with smoke signals.

ALS is a nice variation on LS; it's faster and uses half the power. But the chips usually cost more than the equivalent LS devices. Recently, Jameco ran a sale, so I stocked up. Some of the prices are lower than for LS because they're trying to close out their stock.

74S, 74F, and 74AS are all very fast, but they're also fairly power-hungry. 74S draws the most current, but it's an older technology so you have to expect that. (After all, who's going to come out with a product line with the primary feature of using *more* power than 74S?) 74AS is the fastest, and the newest, of the lot. I haven't seen AS advertised in any of the mail order places yet; maybe in a couple years.

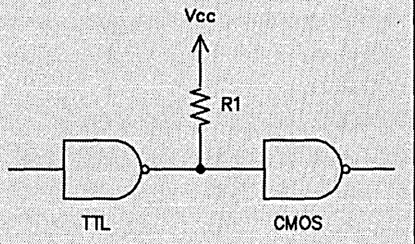
74HC is nice when you have low power requirements and you don't need to run much faster than 74LS. A good example application is a battery powered data collection device. However, harping back to my favorite theme, I still prefer to use chips that won't self-destruct if I connect something wrong (or not at all).

74AC is a fairly new CMOS family. It's very similar to HC except for a much smaller component size. That gives it a substantial performance increase over HC (it can run up to 100 MHz) while retaining all the standard advantages and shortcomings of CMOS: low power, etc.

The TTL input level versions of AC and HC (ACT and HCT) can be useful, but they're not the only way to get TTL to drive CMOS inputs. Check out Figure 4. It shows how to go about it. By adding a pullup resistor, you boost the output of a TTL chip to an acceptable CMOS level. When the TTL chip wants to drive a logic 0, it sinks all the current from the pullup resistor so the CMOS input reads 0.

I've also seen (but never used) another

Figure 4—Driving CMOS Inputs With TTL Outputs



Normally, TTL logic 1 voltage is not enough to drive CMOS inputs, but a pull-up register (R1) can be used to bump the level up to what CMOS expects.

PC TIMER TOOLS

PCHRT is the definitive answer to execution profiling and embedded timer requirements in a PC/MSDOS environment. 46 functions manage three distinct classes of high resolution timer functionality:

21 functions implement multiple microsecond timers. Timers may be stopped, started, suspended, resumed, named, queried, and complete reports of all timer activity may be generated with a single function call.

16 functions manage microsecond timers that may be inserted in virtually any PC interrupt vector. Interrupt timers may have results listed by interrupt function requested, so BIOS, DOS, and EMS interrupts may be profiled effortlessly!

9 functions implement precision microsecond delays, perfect for interfacing data acquisition and process control hardware where precise timing is required.

PCHRT includes libraries for TC, TP, and MSC, example programs, manual, full source, and is $49.95 ppd USA, elsewhere add $4.00. VISA/MC accepted. A 30 day "No Questions Asked" money back guarantee insures your satisfaction. Call or write for our seven page brochure.

RYLE DESIGN

POB 22, Mt. Pleasant, MI 48804
517-773-0587

Reader Service Number 171

family, HCTLS. I read an HCTLS databook and it looks like just an improved version of the HCT family. The specs for HCTLS are a little more stringent than for HCT. HCTLS outputs have just as much drive as LS (HC and HCT provide only half as much drive). The main result of the tighter specs is that HCTLS can be connected directly to just about any other logic family without any pullups or voltage level converters.

What's Coming Up

Off in the distance, there's a new family coming on the scene. Texas Instruments has announced a 74BCT family based on a "bipolar-CMOS" process. It supposedly offers speeds better than 74F, with CMOS power usage. I sent off a card for a BCT designer's kit, but I'm still waiting.

Postscript: Months have passed, but all I've gotten from TI has been a letter saying that part is "temporarily out of stock." The same ad has appeared three times now in two months, and I sent in the card every time. Every time I got a letter saying the kits were "temporarily out of stock." At this point I'm *really* not holding my breath. Maybe 1991....

Another recent breakthrough, also from TI, is the BiQRTT. The letters stand for "Bipolar Quantum Resonant Tunnelling Transistor." TI claims that BiQRTT-based devices could be 100 times denser and 1,000 times faster than today's best. I think 100K ECL is supposed to switch in 100 picoseconds or less. TI's best measuring equipment isn't quite equal to the task of measuring the switching speed of a BiQRTT, but they estimate numbers in terms of femtoseconds (quadrillionths of a second—three orders of magnitude less than picoseconds).

It'll be a while before we see any products based on this technology, but it has enormous potential. Just as an aside, a group in Japan is also working on BiQRTTs, with their traditional emphasis on manufacturing processes.

Outa Here

If you want to find out more about logic chips, there's lots of material to study. Most manufacturers will send you databooks, and even sample parts, for free if you exercise tact and judgement in requesting them. For instance, don't just say you're interested in learning about chips. These companies prefer to hear that you want to buy a whole lot of their chips as soon as you can figure out what the part numbers are....

Bibliography

TTL Cookbook, by Don Lancaster, 1974, Sams, ISBN 0-672-21035-5.

CMOS Cookbook, by Don Lancaster, 1988, Sams, ISBN 0-672-22459-3.

Standard TTL Databook, Vol. 1-4, Texas Instruments Engineering Staff, Texas Instruments, Inc. Vol. 1-1984, ISBN 0-89512-090-9; Vol. II-1984, ISBN 0-89512-096-8; Vol. III-1986, ISBN 0-89512-153-0; Vol. IV-1986, ISBN 0-89512-154-9. *(Editor's note: This is the standard reference set for IC data. Excellent books.)*

High Speed CMOS Designers Guide, Signetics.

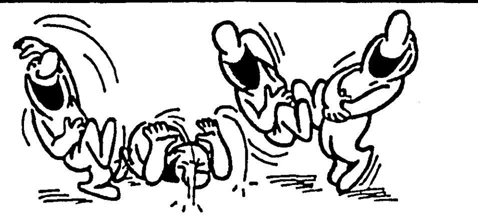
FAST Logic, Signetics.

FACT Databook, Motorola.

Computer Interfacing with C and Pascal, by Bruce Eckel, 1988, Eisys, ISBN 0-89716-211-0. (Available from Micro Cornucopia, $30 in the U.S., $40 foreign.)

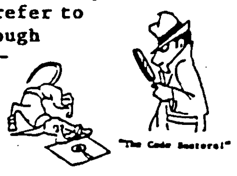
"TI's Quantum Leap," by Richard Doherty, *Electronic Engineering Times,* Dec. 19, 1988.

♦ ♦ ♦



YOU WANT THE SOURCE?!

WELL NOW YOU CAN HAVE IT! The MASTERFUL DISASSEMBLER (MD86) will create MASM compatible source code from program files (EXE or COM). And the files are labeled and commented so they become USEABLE. MD86 is an interactive disassembler with an easy to use, word processor like interface (this is crucial for the REAL programs you want to disassemble). With its built-in help screens you won't have to constantly refer to the manual either (although there are valuable discussions on the ins and outs of disassembling which you won't want to miss).



MD86 is a professionally supported product and yet costs no more than "shareware". And of course, it's not copy protected. **VERSION 2 NOW AVAILABLE!**

MD86 V2 is ONLY $67.50 ($1.50 s&h) + tax

C.C. Software, 1907 Alvarado Ave., Walnut Creek, CA 94596, (415) 939-8153

Reader Service Number 31

DOS IN EPROM

Or any other code, for that matter! **PromKit** allows you to create Eproms that look like read-only disk drives in your PC-compatible systems. Use PromKit **even if you're not a programmer.** Just use PromKit to convert any disk into EPROM images for your Prom blaster! Copy system files, batch files, data files, or anything else you want. Use Proms for read-only, SRAMS for read-write! **Includes source code in C.** Over 180 pages, including disk, only $179.

FREE We'll include a free copy of the pocket-sized **XT-AT Handbook by Choisser and Foster** with each PromKit if you mention this ad when you order. Of course, this $9.95 value is also available by itself. Or buy five or more for only $5.00 each.

MasterCard **800-462-1042** VISA
In California 619-271-9526

Annabooks
12145 Alta Carmel Ct Suite 250-262
San Diego, California 92128 **Money-back guarantee**

Reader Service Number 160

EPROM Programming,

The Software Way

This is probably the simplest EPROM burner you'll find. Read on for the complete hardware and software for a weekend project.

I needed to burn a 27C64 EPROM for my project, but I didn't have an EPROM programmer. So, I wrote one.

Okay, there was a little bit of hardware involved. But only one IC, and the project just took a couple evenings to build. Honest.

EPROMs make developing home-brew microcontroller projects possible. They retain their data even after power loss. You can store your test program in an EPROM on one machine, then move the chip to the target machine and try it. Because they are erasable, you can reprogram them until you get the code right.

EPROMs make developing home-brew microcontroller projects possible. They retain their data even after power loss.

The Right Light

Erasing an EPROM takes nothing more than a shortwave ultraviolet (UV) lamp. Each manufacturer recommends specific erasure procedures for their chips. Intel, for example, suggests you use a 2537 Angstrom light with a power-rating of 12000 µWatt/cm^2. You should place the EPROM within one inch of the lamp's surface and expose it for 15 to 20 minutes.

Or, you can improvise. I've used a shortwave UV mineralogist's lamp for many years; it contains a screw-base bulb made by Sylvania that emits plenty of UV light. If you can locate such a bulb, you could build a nice EPROM eraser for very few bucks. I wish I could give you the Sylvania number for the bulb, but it only carries the text "4W," which I assume is the bulb's power draw.

Of course, you can always opt for a commercial EPROM eraser. Many of the larger mail-order houses offer erasers; you can find anything from a simple single-chip unit to the top-of-the-line model, generally with pullout drawer and adjustable exposure timer.

Whichever way you choose to erase your EPROMs, make sure that your setup prevents anyone from looking at the UV light, either directly or by reflection. The light can sunburn your eyes, perhaps causing permanent damage. This is dangerous stuff; treat it with respect.

To erase an EPROM, first make sure the quartz window in the top of the chip is uncovered and clean. Place the EPROM on a metal tray or press the pins into antistatic foam. Then put the chip under the UV eraser so the top of the chip lies less than one inch from the UV source.

Close the eraser so no stray UV light leaks out, then turn on the UV lamp. Expose the chip for a minimum of 15 minutes. Longer times are okay, although Intel warns you can expect permanent damage to the chip with exposure times greater than one week.

The Nuts And Bolts

An erased EPROM cries out for data, but changing its bits calls for nontrivial software and some fairly simple hardware. It might help to go over the concepts.

Each bit in an EPROM consists of a single metal-oxide substrate (MOS) transistor containing a special "floating" gate. This floating gate lies between the transistor's normal (or select) gate and its substrate (see Figure 1).

The data bit stored in an EPROM cell depends on the concentration of spare electrons in the floating gate. If, for example, the floating gate contains a large number of free electrons, it stops the normal switching function of the MOS transistor. Because the transistor does not switch when the usual gate select voltage is applied, the cell contains a 0.

Similarly, a floating gate without a large number of free electrons allows the normal switching of the MOS transistor to occur. Such an EPROM cell contains a 1.

You program an EPROM cell to a 0 by moving free electrons to the floating gate. You erase an EPROM cell (return it to a 1) by forcing the free electrons off the floating gate and back onto the transistor's substrate.

Erasing an EPROM requires a large dose of shortwave UV light. The high-energy photons in the UV light raise the energy level in the free electrons of the floating gate. Eventually, the electrons reach such a high energy state that they can cross the oxide layer between the floating gate and the substrate.

Upon crossing the oxide barrier, the free electrons lack enough energy to cross back to the floating gate; they remain trapped on the transistor's substrate.

So much for erasing. Programming an EPROM cell requires raising the energy level of the free electrons to a point that

By Karl Lunt
2133 186th Pl., S.E.
Bothell, WA 98012
(206) 483-0447

Figure 1—EPROM Cell

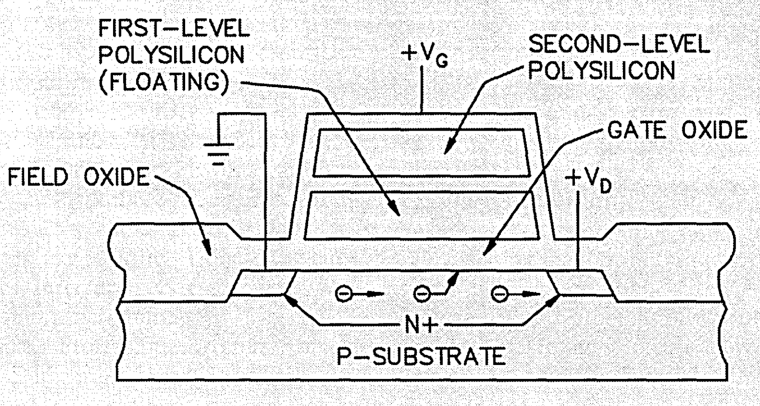
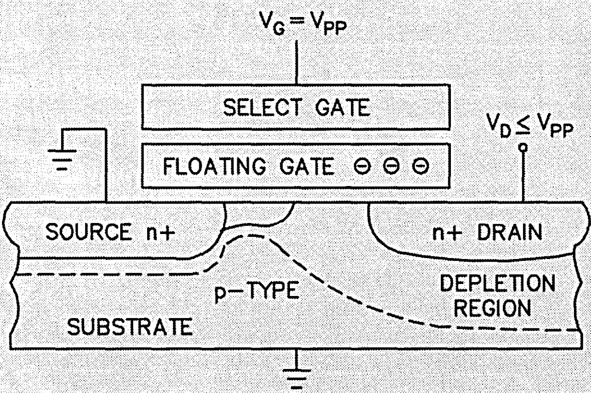


Figure 2—Programming a Cell



they can recross the oxide barrier and return to the floating gate (see Figure 2).

During the programming operation, an EPROM cell may be written to 0 or left unchanged as a 1. In either case, the transistor's drain receives a higher-than-normal voltage, V_{PP}, instead of the usual V_{dd}. For the newer Intel chips, this corresponds to 12.5 volts versus 5.0 volts.

The higher drain potential accelerates the electrons as they move from the source to the drain. When you want to program an EPROM cell to contain a 0, you simply apply the V_{PP} voltage to the select gate as well as the drain.

The highly-energized electrons now have two targets. They can move directly to the drain, as normal, or they can cross the oxide layer and move toward the select gate.

Some of the electrons moving toward the select gate collect and remain on the floating gate; such an EPROM cell will later read as a 0 bit.

Yeah, But How Do You Do That?

When you want to read an address from an EPROM, you first set the address lines, then lower the chip enable line. Finally, you drop the output enable line. The data contained in the selected address then appears on the EPROM's data pins.

Programming a byte into an EPROM is only slightly more difficult. You start by setting the address and data lines. Next, you lower the chip enable line. You then raise V_{PP} from 5.0 volts to 12.5 volts. Finally, you bring the chip's program line low momentarily. This last operation, called a programming pulse, activates the programming circuitry inside the EPROM, writing the data into the addressed byte.

Note that all the operations described above occur very rapidly. Using Intel's Inteligent *(Author's note: Yoo-hoo, that last word is spelled properly and is an Intel trademark!)* programming algorithm, it takes about 1.0 msec to setup and apply one programming pulse to one EPROM byte.

Even though that seems like a very short time, remember that a 27256 contains 32,768 such bytes. This works out to 32.8 seconds to apply a single pulse to each byte, not counting the time to move from one address to another. Then, of course, you have to verify the thing and reprogram any bytes that didn't catch. Intel claims that a typical Inteligent programming cycle for a 27256 EPROM takes about 158 seconds.

But you can change the programming algorithm and significantly improve these times. As an example, Intel's application note AP-277 describes the Quick-Pulse programming algorithm; it can program that same 27256 in about 4.5 seconds.

About The Hardware

The EPROM programming circuit is little more than an 8-bit latch and a power supply (see Figure 3). The design requires connection to two 8-bit I/O ports controlled by a host computer such as a PC or a microcontroller. In my case, I used a 68HC11 single-board computer similar to the Motorola 68HC11 evaluation board.

The power supply contains two three-pin regulators (see Figure 4). The 12 volt regulator only needs to handle a worst-case load of 50 mA; the 78L12 does the job well. The silicon diode in the ground leg of the 78L12 floats the regulator above ground by 0.7 volts, giving an output of 12.7 volts. The old silicon-diode-in-the-ground-leg trick is a cheap way to gain a little extra voltage. Note that the input of this regulator should be at least 15.7 volts, to allow ample room for the regulator's voltage drop.

The 7805 regulator provides 5 volts for both the 74LS373 and the EPROM you're programming. You need a heatsink on this device for a couple of reasons. The regulator will have at least 15.7 volts on its input pin. At the same time, a 2764 EPROM can draw up to 100 mA. This combination means the regulator will have to dissipate as much as a watt.

Just about any source of 16 to 30 volts will run the power supply. I used a small 18-volt transformer and diode bridge. You could also use a bench supply, or even two high-current 9-volt batteries.

The '373 connects to parallel port A from the host computer. This chip can, under program control, latch the state of its input pins and keep that value available on its output pins. Essentially, you get twice the mileage from the same output port.

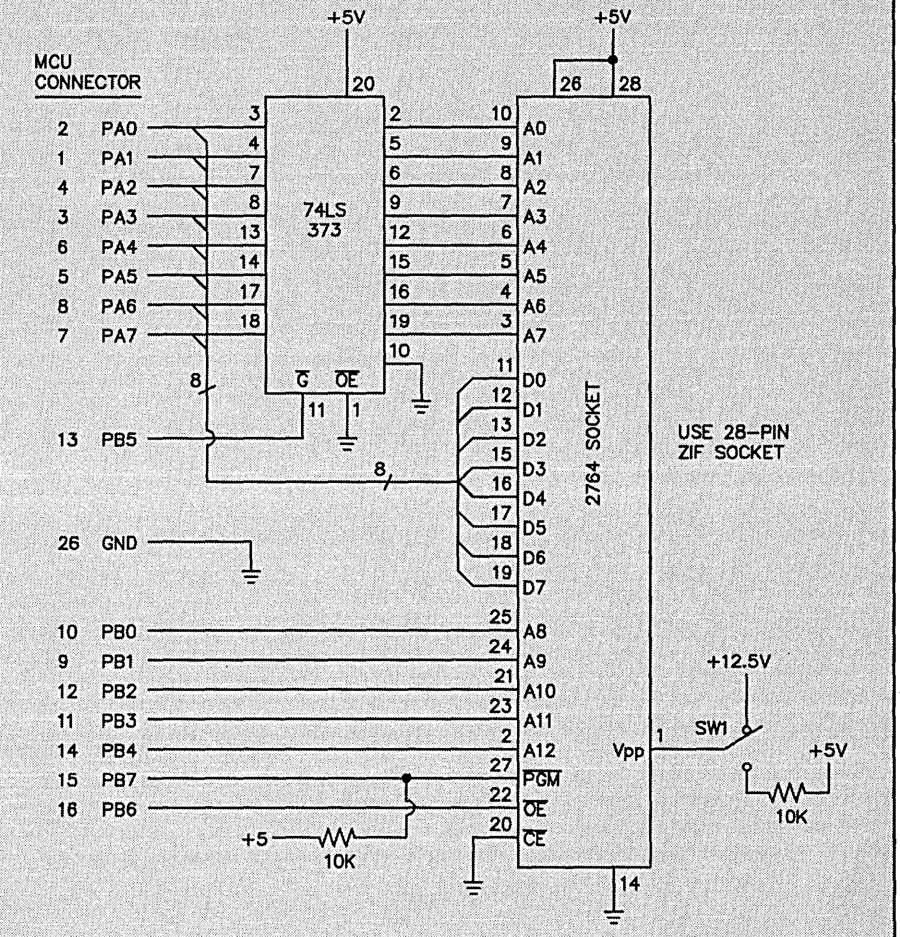
Here, I use the '373 to first latch the low eight bits of the EPROM's address bus. Once latched, I can then use the same output port to supply the EPROM's data byte. Parallel port B provides the higher address bits and some control signals.

Incidentally, you could use the same latching technique in adding a second '373. You could then latch 16 address lines, leaving port B available for the control signals. This arrangement would let you program up to a 65Kx8-bit EPROM; not bad for two chips!

When you build this project, be sure to use a zero-insertion force (ZIF) socket for the 2764. Repeated use of a standard IC socket will eventually damage the tiny springs inside, and it's no fun replacing a 28-pin socket.

Editor's note: In a pinch you can use two normal 28-pin sockets. Solder one into the board, then plug the other into the top of the soldered one. When the top socket wears out, just replace it. The sockets with the round, machined pins work best for this.

Figure 3—EPROM Programmer Schematic



You also must supply two parallel I/O ports from the controlling computer; one of these ports must be bidirectional. Many vendors offer simple parallel cards for plugging into a PC; you could also build your own PC card containing a Motorola 6821 Peripheral Interface Adapter (PIA) or Intel equivalent.

Regardless of the programming system you use, you'll need a spare RAM buffer as large as the largest EPROM you want to program. In my case, I simply reserved an 8K block of RAM on the 68HC11 board to hold the EPROM data.

On To The Software

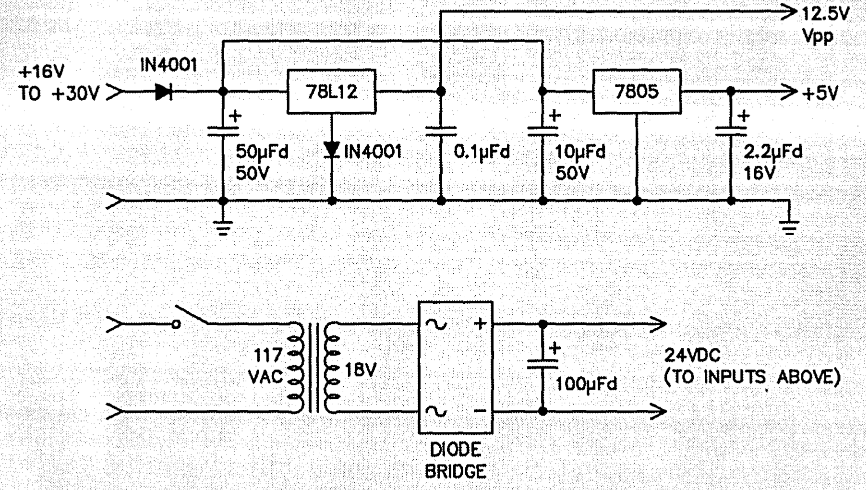
If the hardware for such a project is so simple, you gotta know the software is going to be extensive.

I wrote my programming code for a Motorola 68HC11 microcontroller. This 8-bit MCU has an instruction set similar to the 6809, but hardware-wise it's much nicer. It has such extra features as on-board RAM, serial port, and timer system.

The evaluation board built around this MCU adds two 8-bit parallel ports using a Motorola 6821 PIA. I run Motorola's BUFFALO monitor for debugging software.

You can find a wealth of information on all these chips and systems from your local Motorola distributor. In many of the

Figure 4—EPROM Programmer Power Supply



larger, hi-tech cities, you can sometimes reach a local Motorola field applications engineer (FAE). If you have such a resource person available, make the effort to call up and inquire about literature. The Seattle FAE, Dave Hyder, has been very supportive; I hope your local FAE is as good.

Another excellent source of information and software for the Motorola chips is the FREEWARE bulletin board system (BBS). Motorola sponsors this board and uses it to spread technical information, working programs, marketing literature, and solid examples of MCU design. Give them a call at (512) 891-3733 (8 data, 1 stop, no parity) and look around.

I used my XT clone as a host system for developing the 68HC11 assembler code. Borland's Sidekick notepad editor and the Motorola asm11 cross assembler (available from the FREEWARE BBS) completed my development tools.

I wrote my programming code in assembly language, but higher-level languages such as C or Pascal will also work. The only time-critical section of code involves the generation of the 100 µsec programming pulse. If necessary, you could do this with a short, in-line, assembly language module.

A Closer Look

Refer to Figure 3 for the following discussion. Port A bits 0 through 7 (PA0-PA7) of the PIA deliver the multiplexed data and low address lines to the 74LS373 and to the EPROM. The state of port B, bit 5 (PB5) controls the latching of these eight lines. As PB5 drops to zero, the values of PA0-PA7 get latched and appear on the eight output lines of the '373. These output lines in turn serve as the low eight address lines on the EPROM.

This makes it easy to deliver the low address bits and the eight data bits to the EPROM. First, write the low address byte to PA0-PA7, then latch it into the '373 by bringing PB5 low. Then, write the data byte to PA0-PA7. This value will remain on the EPROM's data lines for later programming.

Note that the above sequence also works for reading an EPROM location; you just read the data byte at PA0-PA7. Whether you read or write a byte depends on how you set the control lines (described below).

The high address lines of the 2764 appear on PB0-PB4. The value of these lines must be stable just prior to reading or writing a data byte.

Performance Programming TOOLS

For IBM PC, XT, AT, PS/2 & compatibles

Commenting Disassembler

Sourcer™ creates detailed commented assembly source code and listings from memory and executable files. Built in data analyzer and code simulator produce great results. Supports 8088/87 through 80386/80387 and 80486 instruction sets. *"Sourcer is the best disassembler we've ever seen!" PC Magazine, January 17, 1989, page 101.* $99.95

BIOS Source

The **BIOS Pre-Processor™** with Sourcer provides the first means to obtain accurate legal source listings for any BIOS. Identifies entry points with detailed in-line comments, explains functions, and more! Requires Sourcer. $49.95; Sourcer & BIOS $139.95 (Save $10)

Unpack EXE Files

The **Unpacker™** group of programs assists with the debugging and analysis of software. Unpack packed EXE files and convert multi-segment COM files back to EXE. Includes handy interrupt viewer and video test program. $39.95.

Assembly Code Analyzer

ASMtool™ is an assembly language source code analysis tool that simplifies the understanding of complex software. Performs automatic flowcharting, analysis of stack and register usage, procedure tree diagrams, and static timing analysis. $89.95

Zoomable File Viewer

With **View-It™** zoom out to see up to 8 times more information on a single screen. View-It handles files of any size, provides dual windows, advanced search and mouse support, and is incredibly fast! EGA or VGA required for zoom feature. $69.95

Assembly Tools Package

ASM ProPak™ includes Sourcer with BIOS Pre-Processor, Unpacker, ASMtool and View-It. $299.75 (Save $50)

C Code Your Way

C-Clearly™ formats C source code according to your style preferences. Make your own style template or use and modify supplied templates such as standard K & R and others. Also produces formatted listings with optional flow lines, line numbers, headers/footers and more. $129.95

Shipping & Handling: USA: $3; Canada/Mexico $10; Other Countries $15; CA Residents add sales tax. Visa/MasterCard/COD accepted. If within 30 days of purchase you find our product does not perform in accordance with our claims, call our customer service department and we will gladly arrange a refund.

NOT COPY PROTECTED
30-DAY MONEY BACK GUARANTEE

For orders or information call:

1-800-662-8266

V COMMUNICATIONS, INC.
3031 Tisch Way, Suite 802, Dept. MC5
San Jose, CA 95128 (408) 296-4224

Reader Service Number 62

Line PB6 controls the output enable line (OE*) of the EPROM. Raise PB6 during a write operation, and lower PB6 to read an EPROM byte.

Line PB7 serves as the program pulse line (PGM*) for the EPROM. After all data, address, and control lines are set properly, pulse PB7 low to program a byte into the EPROM. Make sure this line is high at all other times; you can easily corrupt the EPROM data through careless use of PB7.

Reading And Burning Bytes

The two routines shown (EREAD1 and EBURN1) form the core of the programming software. Refer to Figure 5 for details.

EREAD1 reads a single byte from a specific EPROM address. On entry, the desired address must be in the 16-bit X register.

First, EREAD1 resets all the output pins of the PIA that directly control the programmer. It brings pins PB5-PB7 high; this disables the '373 latch, the EPROM output drivers, and the programming circuitry. Next, the routine writes the low address bits to PA0-PA7

68000

SK*DOS - A 68000/68020 DOS containing everything you expect in a DOS - on-line help, multiple directories, floppy and hard disk support, RAM disk and/or disk cache, I/O redirection, and more. Supplied with editor, assembler, Basic, powerful utilities. Supported by Users' Group and BBS. Software available from other vendors includes C compiler, Basic, editors, disassemblers, cross-assemblers, text formatter, communications programs, etc. Priced at $165 with configuration kit, less if already configured for your system.

HARDWARE - 68xxx systems start at $200. Call or write.

Star-K Software Systems Corp.
P.O.Box 209
Mt. Kisco NY 10549
(914) 241-0287 / Fax (914) 241-8607

Reader Service Number 40

Figure 5—EPROM Read/Write Code

```
*    EREAD1
*    Read one byte from the EPROM. The EPROM address appears in
*    IX. Byte read is returned in AR.

EREAD1:
        PSHB                       SAVE BR
        PSHX                       SAVE IX
        LDAA    #%11100000         OE* = 1, LATCH* = 1, PRGM* = 1
        STAA    PORTB              CONFIGURE THE PROGRAMMER BRD
        XGDX                       GET ADDR IN AR AND BR
        STAB    PORTA              WRITE LOW BITS OF ADDR TO EPROM
        LDAB    #%11000000         OE* = 1, LATCH* = 0, PRGM* = 1
        STAB    PORTB              LATCH THE LOW ADDR INTO 74373
        CLRB                       NOW SET PORT A TO INPUT
        STAB    PORTA+1            ACCESS THE PORT A COMMAND REG
        STAB    PORTA              WRITE A $00 (ALL BITS SET TO INPUT)
        LDAB    #$04               NOW RESTORE PORT A TO DATA PORT
        STAB    PORTA+1            PORT A NOW SET TO INPUT 8 BITS
        ANDA    #%00011111         LEAVE ONLY TOP ADDR BITS IN AR
        ORAA    #%10000000         OE* = 0, LATCH* = 0, PRGM* = 1
        STAA    PORTB              WRITE TO EPROM BOARD
        LDAA    PORTA              READ DATA FROM EPROM
        LDAB    #%11000000         OE* = 1, LATCH* = 0, PRGM* = 1
        STAB    PORTB              DISABLE THE EPROMS OUTPUT LINES
        CLRB                       NOW RESTORE PORT A TO OUTPUT ONLY
        STAB    PORTA+1
        DECB                       $FF = ALL LINES OUTPUT
        STAB    PORTA              SET ALL LINES
        LDAB    #$04               NOW SWITCH ON THE DATA REGISTER
        STAB    PORTA+1            PORT A NOW SET TO OUTPUT 8 BITS
        LDAB    #$55               DISARM THE COP
        STAB    COPRST
        LDAB    #$AA
        STAB    COPRST
        PULX                       RESTORE IX
        PULB                       RESTORE BR
        RTS


*    EBURN1
*    Program one byte in the EPROM.  Upon entry, IY points to the
*    68HC11 address containing the data, and IX contains the EPROM
*    address to hold that data.
*
*    This module assumes the EPROM programmer is already set up
*    and that Vpp has been switched to the proper programming
*    level.
*
*    All registers are preserved.

EBURN1:
        PSHA
        PSHB
        JSR     EREAD1             GO READ CURRENT EPROM CELL
        CMPA    0,Y                IS IT ALREADY THERE?
        BEQ     EBURNX             BRANCH IF YES

        LDAA    #%11100000         OE* = 1, LATCH* = 1, PGM* = 1
        STAA    PORTB              SET THE COMMAND LATCH
        PSHX                       MOVE THE EPROM ADDRESS
        PULA                       HIGH BYTE INTO AR
        PULB                       LOW BYTE INTO BR
        STAB    PORTA              WRITE A0-A7 TO PRGRM LATCH
        LDAB    #%11000000         OE* = 1, LATCH* = 0, PGM* = 1
        STAB    PORTB              LATCH THE LOW ADDR BITS
        ANDA    #%00011111         LEAVE ONLY A8-A12 IN AR
        ORAA    #%11000000         OE* = 1, LATCH* = 0, PGM* = 1
        STAA    PORTB              WRITE HIGH ADDR
        LDAB    0,Y                GET THE DATA BYTE
        STAB    PORTA              PUT ON EPROM DATA LINES
        LDAB    #10                HIT EACH BYTE 10 TIMES
        PSHX                       SAVE IX
EBL01:
        ANDA    #%01111111         SET PGM* = 0
        LDX     #33                SET COUNTER TO DELAY 100 US
        STAA    PORTB              START THE BURN CYCLE
```

```
EBLOOP:
          DEX                         3 CYCLES
          BNE         EBLOOP          3 CYCLES

          ORAA        #%10000000      SET PGM* = 1
          STAA        PORTB           END THE BURN CYCLE
          DECB                        COUNT THIS PRGM PULSE
          BNE         EBL01           BRANCH IF NOT FINISHED
          INCB                        GET A 1
          STB         BURNED          REMEMBER THAT WE CHANGED A LOC
          PULX                        RESTORE IX

EBURNX:
          PULB
          PULA
          RTS

                                ♦ ♦ ♦
```

and latches them into the '373 by dropping PB5 low.

Then, EREAD1 changes the PIA so that port A becomes eight input lines. As these lines already connect to the eight data lines of the EPROM, port A will be used to read the requested EPROM data.

Finally, the routine sets PB0-PB4 to the correct high address lines, and brings PB6 (OE*) low. The data byte for the desired EPROM address now appears on lines PA0-PA7. The remaining code in EREAD1 simply resets the PIA port A to output mode and returns the EPROM control lines to a known state.

EBURN1, the routine that writes a data byte into EPROM, is only slightly more complicated. After latching the low address bits into the EPROM, EBURN1 writes the proper data byte to PA0-PA7, then sets the upper address bits into PB0-PB4. Finally, the routine holds PB7 low for 100 µsec (to apply a programming pulse), then updates some global variables used by other portions of the code.

Note that EBURN1 assumes the V_{PP} programming voltage has already been connected to the EPROM; this function must be performed elsewhere in the programming code.

The remainder of the software serves as user interface and to control the programming algorithm. You can really customize your EPROM programmer here.

Burning Bytes Quickly

A good programming algorithm strives for reliably writing the data in a minimum amount of time. You can achieve both these goals with judicious use of a read-before-write technique.

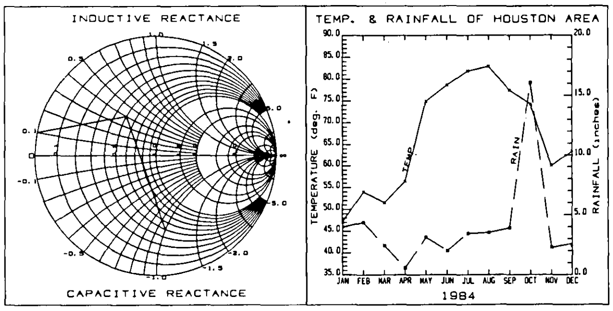
Intel's literature describes an in-house study of its Quick-Pulse programming algorithm, as applied to the 27C256 EPROM. This study shows that over 60% of an EPROM's cells program correctly after the application of a single 100 µsec programming pulse; nearly 100% of the cells are properly programmed after just 10 such pulses.

INGRAF 2.10

A multi-device graphics library for Scientific, Engineering, and Business users! Supports video, printer, and plotter graphics for personal computers.

Over 100 routines to create bar, pie, and smith charts; linear, log (semi-log and log-log), and polar plots; axes and grids with tick marks and labels, markers, line types, curves, arcs, circles, ellipses, and more. *(Full source code; no royalties or run-time fees!)*

The window function allows you to use up to 32 windows. You can set individual characteristics (such as sizing, scaling, labeling, etc.) for each window or identical characteristics for all windows.

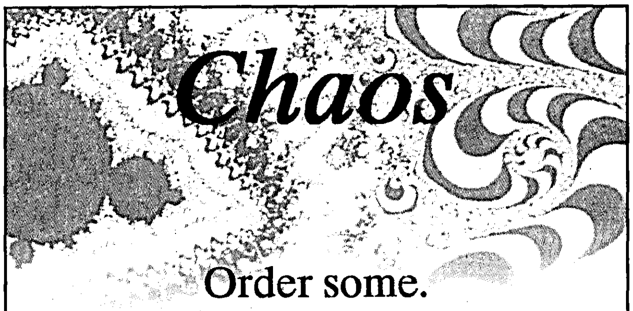


INGRAF is available for C, FORTRAN-77, and QuickBASIC (Pascal available soon); support for numerous compilers.

We also have GRAFLIB 4.10 for video and printer graphics; PLOTLIB 4.10 for pen plotter graphics, and FORTLIB 4.05, a FORTRAN enhancement library.

Sutrasoft
P.O. Box 1733 • Sugar Land, TX • 77487
Order Line : 1-800-888-8460
All other calls : (713) 491-2088

Reader Service Number 173



Mandelbrot Explorer 3.2

Fantastic fractal graphics on 16-color EGA/VGA to 800x600. Magnifies up to 16.5 trillion times. Stop and start at will, save and retrieve, collage, full control over color boundaries, "zoom box," display of periodic orbits, auto-backup, all optimizations for speed including pixel interpolation and 386 integer support. Comes with seven ready-made pictures for immediate gratification.

$34.95

Peter Garrison
1613 Altivo Way
Los Angeles, CA 90026
213 665 1397

When ordering please specify EGA/VGA and disk format
Overseas orders please add $4

Reader Service Number 112

This leads to the use of an algorithm that burns an EPROM cell only if necessary, using a small number of pulses. The program then reads all the locations to verify accurate programming, and only reburns those cells that need it.

By employing these techniques in my code, I can reliably program a 2764 in ten seconds.

(The Intel Quick-Pulse programming algorithm, in its pure form, requires more extensive hardware than that shown here. For example, programming an EPROM with Quick-Pulse uses a V_{cc} of 6.0 volts, rather than the 5.0 volts in my system. However, my software checks the accuracy of the programming to ensure the bits get burned the way they should. Consult Intel applications note AP-277 for details on the Quick-Pulse algorithm.)

Limitations And Improvements

This system will only program the 27C64 and 2764A EPROMs. Modifications to handle other devices require only minimal changes to the hardware and software.

To program the older 2764 EPROMs (these need a V_{PP} of 21 volts), simply provide 21 volts to pin 1 (V_{PP}) of the EPROM during programming. Use a 78L18 regulator with four diodes in its ground leg, instead of the 78L15 with one diode as shown. Note that this change means you must supply at least 24 volts to the unit's power input terminals.

The programming code supplied here works equally well on the older EPROMs, once the higher V_{PP} voltage is used.

Note: If you put in the 21-volt modification, *do not* use the higher voltage to program a 27C64 or 2764A; you will fry these newer chips.

Programming larger EPROMs (such as the 27128A) means you must supply additional address lines to the EPROM socket. As you can see, all the parallel port bits have been used already. Therefore, you must add another '373 latch if you want to burn larger devices. Consult Figure 6 for details on how to add this second latch.

Since the tweaking in this programmer takes place in software, you can add all kinds of features without modifying the hardware. My programming code, for example, allows me to write a background pattern of $FF to all bytes in my RAM buffer before I load in the data for programming. Since an erased EPROM byte reads $FF, this feature really cuts down burn time; unmodified bytes already appear to be properly burned.

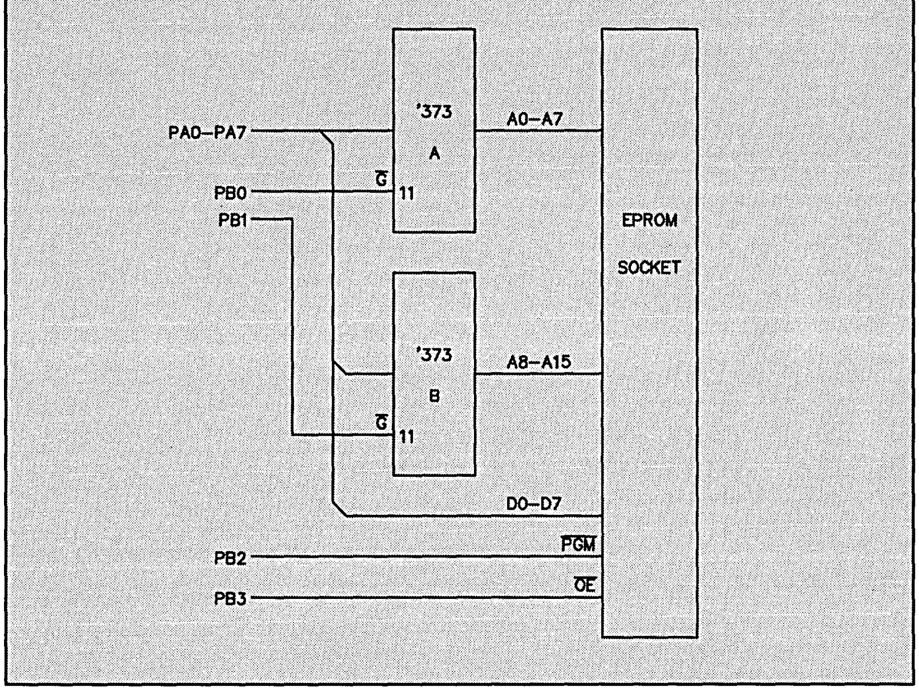
You can also experiment with the number of programming pulses per burn cycle. My code uses 10 pulses for each pass through the EPROM; through experimentation, you might discover that 6 pulses (for example) will do the trick. This would cut the programming time by 40%.

You could also add the ability to program individual bytes of an EPROM selectively. Although you cannot use the programmer to change a 0 bit back to a 1, you could still provide enough editing capability to save yourself a complete erase-and-burn cycle.

How about adding the code to calculate a checksum or CRC for the EPROM data? If necessary, you could have your software burn the CRC value into the first (or last, or whatever) bytes of the EPROM. This would provide a reliable way of verifying EPROM integrity and firmware revision level; the startup sequence for the target computer would simply recalculate the CRC for the EPROM and match its value against that stored in the EPROM.

You could even modify this circuit to permit simultaneous programming of multiple EPROMs. You would want to switch V_{cc} and V_{PP} from one EPROM to another as needed. You would also have to carefully control OE* so that only one EPROM was driving its output lines at any one time. Still, this would offer some interesting possibilities.

Figure 6—Expanding the EPROM Programmer



In Conclusion

I started this project out of necessity and ended up learning a lot about EPROM programming. This project is so simple that I am tempted to add the hardware to a dedicated 68HC11 MCU system and build my own, stand-alone EPROM programmer.

For information on EPROMs and programming algorithms, consult the *Intel Memory Components Handbook* (literature number 210830-008). For details on the Motorola 68HC11 MCU, 6821 PIA, and related support chips, consult Motorola's *Microprocessor, Microcontroller and Peripheral Data Book,* Volume 2 (literature number DL139).

Special thanks go to Phil Clyde of United Products for materials in building this project. If you are ever in Seattle, stop by U.P. (1123 Valley St., Seattle, WA 98109-4425) and say "Hi." Phil will be the big guy behind the counter.

♦ ♦ ♦

By Deborah Norling
Grassroots Computing
P. O. Box 460
Berkeley, CA 94701

Controlling Runaway C With CC-Rider:
Hypertext Editing For C'ers

If you do much C programming, you know how hard it is to keep track of all your routines. Well, here's help, pardner. Saddle up CC-Rider with your tex' editor and you'll have a lot less trouble corralling wayward code.

You're trying to calculate the difference between two dates: how many days between May 3rd and August 22nd? You don't want to reinvent the wheel; besides, there isn't time. You call a friend—sure enough, he has two public domain programs which perform date conversions. *Editor's note: You go out with my date, I'll go out with yours.*

You think your troubles are over. Unfortunately, they're just beginning.

Your friend's code works. You throw a few test dates at it and everything seems okay. Then you link it into your application only to see it grow by 36K and slow to a jog.

You search through the date math handling routines, trying to find and remove the unnecessary code. But the code is vintage '78 K&R C, from the days when prototype meant "mock up." It'll take hours, maybe days, to filter out the unnecessary routines. And when you finish, there's always the possibility that the date routines won't emerge unscathed. What can you do? Get an XREF.

Why Use XREF?

A cross reference utility, XREF produces a road map of your code. It lists all your symbols (variables, functions, and constants) along with where they occur. For instance, you can look up the symbol "BLUE" and discover that it's a constant defined in CONIO.H as "#define BLUE 1."

In the back of your mind, you remember that somewhere in your code you have another "blue." But what did you use it for? Run XREF and there it is—"blue()" is an external function defined in your user environment module. It returns TRUE if the user has faked the attribute underline on a CGA display (the color blue on CGA is represented by the same bit value as underline on MDA monochrome.) Using an XREF utility, you can locate identifiers and figure out what they do.

If you want to experiment with XREFs, plenty are available for downloading. Some are shareware—a few are C-specific. The program XC has been around since the early 80s, and many of the C formatted print listers have an XREF option.

You'll soon discover the problem with these XREFs: they're paper-oriented. If you make judicious use of an old-fashioned XREF, you can easily spend half your day at the water cooler, waiting for the printer to finish. Score one for procrastination.

Hypertext Editing

Enter CC-Rider, a package that bills itself as a "hypertext editor" for C programmers. Technically this is true. Point to the symbol you wish to cross-reference and that symbol acts as a hyperlink.

Think of CC-Rider as an online help for your own code. It's an even better help for other people's code. (Sometimes other people's code needs all the online help *you* can get.) In fact, skip the hypertext hype, just think of CC-Rider as a popup XREF.

Here's how CC-Rider works. You're cursoring through last year's code, looking for that blasted string search function. You know it bombs horribly when you feed it a null string. Since the code is a year old, you can't even remember where it resides.

Suddenly, you run across a call to the function srch_str. Yep, that's the culprit. Since CC-Rider is resident, you press Alt-Shift to bring up its "symbol entry" window. CC-Rider recognizes the function call under the cursor, flashes the drive light briefly, and retrieves your definition for srch_str.

Ah-hah! You prototyped it in FILESTUF.H and coded it in FINDER.C. You even commented its prototype well. Too bad the function definition and the prototype don't match.

Automated Prototyping

Let's return to your friend's date conversion functions. To merge them into your own code, you must flip through pages of listings to locate the start of each function. Then you must manually add the appropriate prototype. Why not let CC-Rider's paste command do the work?

Just point to the function definition. CC-Rider inserts a prototype of the function for you. Here's how it works:

(1) Using your editor, move the cursor to the desired location.

(2) Pop up the symbol entry window.

(3) Type in the name of the desired function prototype.

(4) If the function has previously been defined, CC-Rider will fetch a copy of the definition and insert it at the cursor, in prototype format.

(5) CC-Rider returns to your editor.

You can also track variables. Assume that in some places, your friend's code uses "i" as a general-purpose pointer, while in others, "i" is an integer loop counter. Put your cursor on "i" and pop up a window with the local declaration for the current invocation of "i."

Within the window, use CC-Rider's F for Forward or B for backward com-

Tired of the limitations of MS-DOS ?
Thinking about switching to another OS ?
Well DON'T, now there is :

PRO-CLI v4.1

CLI is an intelligent front & back end processor that enhances the MS-DOS system kernel. CLI provides many features of advanced OS's, like VMS and UNIX, to MS-DOS without any re-training.

The core of the CLI kernel is based on recursive definitions and references to Alias, Logicals, Symbols, and Lexical-Functions from the command line, batch programs, and existing application programs (without any modifications).

Powerful command programs can be quickly written for a wide range of purposes such as:

- flexible control of code & data management
- code development & code support tools
- automated file/device maintanence utilities
- creation of generic command procedures
- installations & system re-configurations

For the first time, designers, developers, and managers can coordinate software development and production efforts without the major limitations and restrictions in the MS-DOS environment.

- VMS is a trademark of D.E.C.
- UNIX is a trademark of AT&T
- MS-DOS is a trademark of MicroSoft

Alphax Systems Inc
1005 Sussex Blvd
Broomall, PA 19008

Price: $ 149
S/H: $ 6
215-543-2658

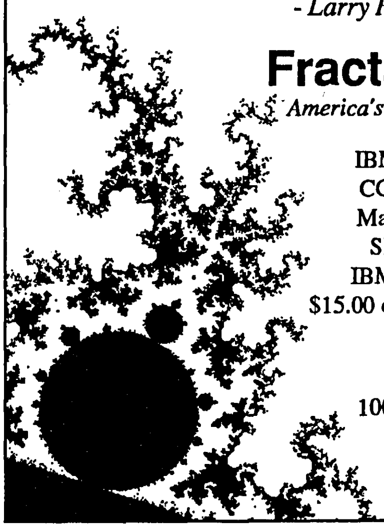
Reader Service Number 187

"It's Glorious!"

"No color PCompatible is complete without a copy of this; get one and see what I mean." *- Jerry Pournelle, BYTE*

"Infinite... Neither of us (the chief engineer of the Galileo Jupiter probe and I) may ever be heard from again." *- Arthur C. Clarke*

"I could watch an animated image for hours." *- Larry Fogg, Micro C (M/A 89)*



FractalMagic 5.0
America's Premier Fractal Program

IBM PC, PS/2 (3.5" or 5.25")
CGA, EGA, & VGA: $35.00
Macintcsh: Plus, 512KE, SE, SE/30, and Mac II's: $35.00
IBM Expansion Modules 1-4: $15.00 each, or all four for $45.00

Sintar Software
"Software for the Mind"
1001 4th Avenue, Suite 3200
Seattle, WA 98154
(206) 625 - 1213
Visa - MC - AmEx

mands to jump through the database to the next or previous declaration of "i." When a comment appears together with a declaration, you'll see the comment also.

Unfortunately, the sample date code is so messed up that you decide to change a function's parameters. With CC-Rider, point to a function call, pop up the window, and select EDIT. Immediately you're in the source file, right where the function's defined. This way, you can alter the definition before you forget what you need to change.

Two Programs

CC-Rider is composed of two programs. CCSYM prepares the way for the TSR CCRIDER.

You run CCSYM immediately after compilation, which creates an up-to-date database of your identifiers. Pour another cup of coffee unless you have a speedy drive on a '386, because CCSYM will churn a while as it analyzes your source files. Only after this process do you get CC-Rider's power.

CCSYM has lots of command line options. If you use batch or MAKE files, you can automate this process. The CC-Rider manual devotes a section to smoothly integrating the CCSYM program into a MAKE. If you use an integrated environment, you'll have to produce another solution, such as SuperKey macros, to automate the updating of the symbol database fully.

Keystroke Macro File

CC-Rider must be able to tell your editor to load a file and to go to a specific line. So you create a 128-byte file containing hex codes for the keystrokes your editor uses. You get sample macro files for Emacs and Microsoft's M editor.

I use QEdit, customized for my quirky keystrokes. Pulling out three scancode/extended key code charts, an ASCII table, and my hex conversion calculator, I dutifully fired up DEBUG and pegged the proper bytes at the proper offsets. Eventually I got it right. If CC-Rider ever expands to require knowledge of more than two editor keystroke sequences, let's hope we'll be able to enter them in ASCII, such as "goto_line = ctrl-home" and "open_new_file = ctrl-K R."

What do all these contortions accomplish? Let's assume that you run across the correct call to a function named OpenWindow. You want to edit an out-of-date definition of the OpenWindow function in a second source file. If you have your macro configuration properly set up, CC-Rider can auto-pilot your editor into the second source file and land the cursor on the target line.

If you skip the DEBUG routine, CC-Rider will still let you view other source files from within its window. You just won't be able to edit them on autopilot. The Paste command doesn't require knowledge of your editor's keystrokes.

They've included two demos with CC-Rider. The smaller demo is a four-module implementation of a reverse polish notation calculator. The symbol databases have already been created; you merely load the code into your editor and start snooping around.

The second demo is a complete public domain implementation of a UNIX-style MAKE utility. You can browse through the MAKE program's multiple modules, popping up XREF info about any of the symbols. If your C compiler has a brain-damaged MAKE, here's a replacement.

CC-Rider has a strong OS/2 flavor. MS-DOSers will have to ignore the many paragraphs in the manual which begin "In the OS/2 implementation...." The disk includes the OS/2 executable version of the program and numerous .CMD files. (The OS/2 files are useless to MS-DOSers.) If you develop under OS/2, you can analyze source code in the background.

There would have been a little less confusion if the author had put the OS/2 and MS-DOS information in two different manuals and the program versions on different disks.

Treats & Virtues

CC-Rider has a special treat for programmers using Microsoft's Quick C. CCSYM can generate a Quick-Help database in addition to the normal CC-Rider symbols database. Since the Quick-Help TSR can display your symbols database and facilitate the pasting-in of prototypes, you won't need to load the CC-Rider TSR if you use this special database.

The CC-Rider TSR is well-behaved and I couldn't crash it (Lord knows, I tried). This might not seem very significant, but it takes a lot of behind-the-scenes work to craft a stable TSR.

I've seen too many packages which are top-heavy on features, precariously balancing on a crumbling foundation. For example, the Norton Guides don't coexist well with Turbo C's Thelp TSR. My favorite cache, Super PC Kwik, doesn't like Turbo C's integrated environment.

But CC-Rider coexists peacefully with every program I own. It can remove itself from memory within a batch file, so you can load it before you run your editor, and unload it before you compile. (Give those memory-hogging compilers all the room they need.)

Wish List

Without user wish lists, there would be no reason for developers to write new versions. I'd like a future version of CC-Rider to:

- Understand the format of online help databases included with both Turbo C and Zortech C/C++.
- Work with more editors (including simplified macro file creation).
- Be able to jump from one function to the next or previous (like moving around by pages and paragraphs.)
- Display calls to any particular function.
- Create a tree chart showing relationships among functions.
- Generate statistics, such as frequency of invocation.

Most importantly, CC-Rider needs a way to locate a symbol when you're not sure how it's spelled. Currently, when you've forgotten the name of a symbol or are looking at someone else's code and don't know the symbol names, CC-Rider isn't much help. I'd love to be able to search in the database via wildcards, regular expressions, soundex, or by a word in the comment.

For example, if you knew you'd written a function to draw borders, but couldn't remember its name, you could search for "box," "frame," and "border," assuming you used one of those words in your comments. Also, it would be nice if "*str*" would produce all your string functions.

In the future, CC-Rider might even maintain a database of user-created libraries. This could be given a read-only file attribute to prevent it from being modified, while the databases for the current work project would be constantly updated. You would only rebuild the database when you changed the library.

In Sum

CC-Rider, though somewhat limited, is a stable and handy tool. It frees you from the tedium of poring through a mound of printouts. It quickly locates misplaced function definitions. It enables you to track down forgotten variables. It liberates you from the tyranny of prototyping by hand.

It doesn't have lots of bells and whistles and it won't help you locate a symbol if you don't remember its name. But CC-Rider will automate some of your work, will coexist with cranky TSRs, and is nearly bug free. If you program large projects, then CC-Rider is well worth $89. For more info—

CC-Rider, ver. 1.3
Western Wares
Box C
Norwood, CO 81423
(303) 327-4898

♦ ♦ ♦

The $25 Network

Try the 1st truly low cost LAN

- Connect 2 or 3 PCs, XTs, ATs
- Uses serial ports and 5 wire cable
- Runs at 115 K baud
- Runs in background, totally transparent
- Share any device, any file
- Needs only 14K of ram

Skeptical? We make believers!

Over 15,000 sold worldwide

imodes

Information Modes
P.O. Drawer F
Denton, TX 76202
817-387-3339 or 1-800-628-7992

Why waste money on simple file transfer systems?

Reader Service Number 149

CULTURE CORNER

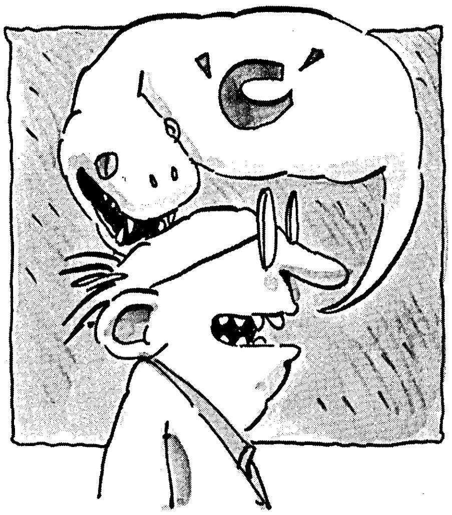
Glossary Of Computing Terms

Thomas J. Cook, Ph.D.
Astartė
P.O. Box 821413
Dallas, TX 75382

It has come to our attention that many *Micro Cornucopia* readers are just breaking into the technical side of computing. As a service to these readers, I have compiled this list of commonly used—but often incorrectly-defined—terms.

Assembly language—Language of choice for Scrabble players. Allows you to write the smallest, fastest routines in five months. Extra points for variable names rich in Qs and Zs. Assembly language programmers often become self-proclaimed legends.

Bulletin boards—Mechanisms allowing the socially autistic to masquerade as people, communicating with one another by posting clever near-random commentary on remote computers.



C—Short for "chutzpah," a quality needed before tackling even the simplest program with this language. C is also the symbol for the speed of light, but that has absolutely nothing to do with how quickly one can learn the language. C encourages self-documenting structured programming through constructs such as—

```
(*wnd->func)(*++addr)
```

which means, call the routine whose address is stored in the "func" variable for the structure pointed to by "wnd," and pass to it the contents of the cell pointed to by the pointer in "addr" after it (the pointer, not the contents) has been incremented. Or something like that.

Clone—An acronym standing for "Copied Low-cost Optimal Non-IBM Equipment." Often used as a cure for the dreaded Big Blue. Texas, land of independent self-styled individualists, is currently "Siliclone Valley," where imagination is limited only by IBM. See **IBM**.

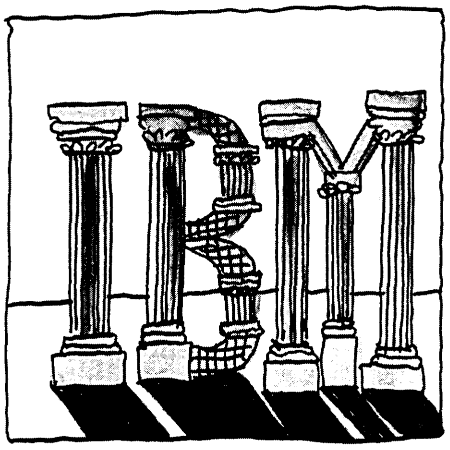
Cyberpunk—High-tech version of the 50s hood who is into computers rather than cars. This cause-without-a-rebel cruises AT&T networks instead of Main Street, and breaks into Pentagon computers rather than condom machines.

EISA—Chinese for "we copied it without duplicating it." Inscrutable alternative to the Micro Channel Architecture (MCA) backed by everybody but IBM.

Gang of Nine—Originally the Gang of None, this is a group of 100+ coming-of-age companies marked by their newfound willingness to tell IBM jokes in public, and their unwillingness to pay IBM bus royalties. Sample joke—
Answer: EISA, MCA, and Greyhound.
Question: Name two dogs and a bus.

Hacker—A programmer who grew up tapping out Morse Code on a ham radio and has never forgiven IBM for not putting a switch panel on the original PC. Recognizable by the oversize ring of keys on the belt that pulls his/her pants down just past the shirt tail.

Also recognizable by the list of phone numbers and passwords in the wallet. To a hacker, "hot date" is the day the newest eunuchs—er...UNIX patches ship.



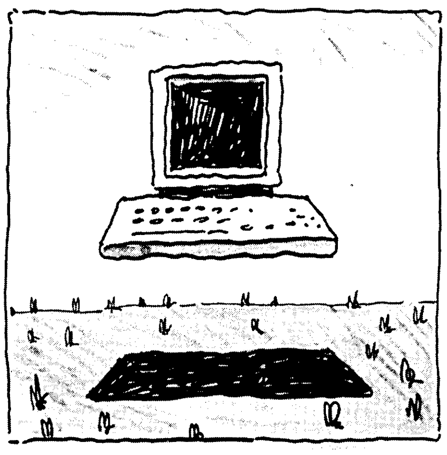
IBM—Standards proposing organization. IBM develops hardware architectures and builds slow underpowered prototypes for other companies to improve upon. See **Clone**.

Marketing—Formerly known as Sales until yuppies declared smoking, greasy hair, and overt aggression as Out; and wine, colorful suspenders, and schmoozing (covert aggression) as In.

Considered a no-brainer by most developers, who are in turn considered the equivalent of fertilizer (necessary for growth, but unpleasant to deal with, and best stored in a dark shed) by sales people.

Micro Channel Architecture (MCA)—IBM's new bus that carries information in 32-bit packets. This is the first bus developed solely by lawyers, and is considered copy-proof (the theory being that no one would want anything created by lawyers). The bus is actually 48 bits wide, but the lawyers take 1/3 of anything they work on. This is a not-so-subtle attempt to limit the PC market to IBM.

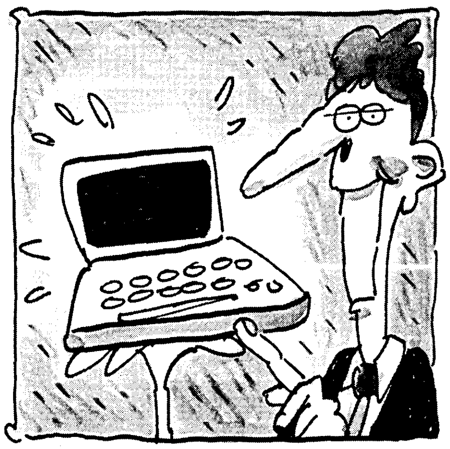
NeXT—Experimental computer backed by Ross Perot and powered by charisma. The main problem is that few homes or offices have charisma outlets. Name-wise it's reminiscent of "The Last One," a CP/M program supposedly powerful enough to create all your future application programs (making it the *last* program you would have to buy). The Last One was also powered by charisma.



Pascal—Language named after the dead mathematician who dreamed up the first computer. That computer, like the language, was non-programmable. But it was a good idea.

Ph.D.—A user with more sense than money. Ph.D.s generally have elegant solutions to problems which don't yet exist. The (top-down, of course) solutions are always correct because they're never implemented.

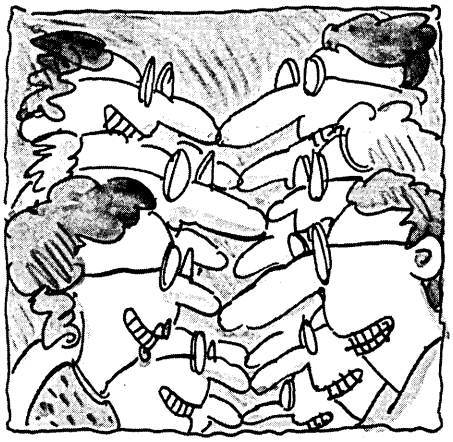
Power user—A user with more money than sense. A power user subscribes to all the magazines, buys all the latest hardware, spends long hours running short timing tests, and grants 30+ interviews per week to columnists. Proud to know every variation of the DOS dir command. As a kid, the power user was first in line to see "Star Wars" (all 22 times s/he saw it) and acted as assistant equipment manager for the football team.



Terminate and Stay Resident (TSR)—Modeled after the tenure system in schools (where teachers retire on the job, and can't be excised); this type of program runs and terminates, but does not give up its memory space. Many TSRs will reawaken any time you inadvertently hit a special key combination (the so-called "hot key"). Inevitably, the resulting confusion deletes all files.

UNIX—Conceptual granddaddy of DOS and OS/2, this operating system is condemned as too cryptic for PC users (with commands like *cd* and *mkdir*). UNIX is also hampered by ready-availability on diverse hardware (wouldn't it be confusing if your VAX and PC operated with the same commands?), a notable lack of bugs, and new versions with DOS compatibility.

IBM and Microsoft fully understand the power of UNIX. Though neither profits by its use (AT&T gets the bucks), they try hard to get people to use it. Alas, they report users just keep clamoring for OS/2 (which, coincidentally, IBM and MS make a lot of money on).



User group—A collection of longtime bulletin board users who couldn't stand not knowing what each other looked like. To cure this curiosity, they periodically get together. It works—most members can now stand not knowing what other BBSers look like. Kind of official SOG. See **Bulletin board**.

♦ ♦ ♦

C'ING CLEARLY

A Programmer's Toolbox

By Scott Robert Ladd
705 West Virginia
Gunnison, CO 81230
Voice: (303) 641-6438
BBS: (303) 641-5125

Now that we know the answer to "life, the universe, and everything" ("Culture Corner," Micro C Issue #45), it's time to move on. So, Scott has taken on the task of creating "life, the universe, and everything." Here he tackles non-carbon-based forms.

If you don't believe the flow of time is variable, you don't work as a writer. Though I'm writing this column at the beginning of December, it will appear in the March-April issue of *Micro C*—which actually arrives on newsstands and in mailboxes at the beginning of February. I always knew *Micro C* existed in a time warp (or at least a warp of some kind).

In the last two years, I've done a big wrap-up of MS-DOS C compilers for the March-April issue. This year, I'm going to delay that review for one or two issues. For starters, a big wrap-up review is a *lot* of work, and right now time is somewhat short for me.

Second, several major vendors are releasing upgrades of their products within the next few months. So, I'm going to wait until I have the time to do a quality job reviewing the latest products.

TOOLBOX

People often ask me about the tools I use. My work as a writer for computer magazines gives me the opportunity to examine a wide variety of tools; many people correctly theorize that the best of these tools end up as parts of my personal programming environment.

Of course, everyone's definition of "best" varies; my opinions are based on how I do my work. You may or may not agree with my choices of the best software. What I use works for me, and I hope it works for you, too.

I keep several C compilers resident on my development PC, Bull. Currently, I have Borland Turbo C 2.0, Microsoft C 5.10, Microsoft QuickC 2.01, and Zortech C++ 2.0. Watcom C 7.0 and Lattice C 6.0 are what I call "temporary installations"; they're installed, but I'll remove them the moment I need more disk space.

I keep Turbo C and the Microsoft compilers installed to check the compatibility of my C programs with them. Borland and Microsoft, after all, put out good products which most of the MS-DOS C programming community use. Neither is my primary compiler, however. That honor goes to Zortech C++.

As I mentioned in the last column, I do most of my work in C++ these days. I just can't live without those objects, methods, and the polymorphism. Zortech C++ is a superior product, combining very fast compiles with an excellent global code optimizer. To be honest, I just plain like the "feel" of Zortech C++; the whole product is comfortable.

Why don't I use the Watcom compiler? Everybody, myself included, has been touting Watcom's code optimization during the last year or so. Unfortunately, the more I work with Watcom C, the less I like it. It's simply too slow for everyday use; Watcom's compile times are often so long I wonder if the computer has locked up.

In addition, Watcom C suffers from problems with memory usage. On a PC with 560K of free memory after boot, Watcom will often complain that it can't fully optimize functions. Zortech is quite capable of doing equivalent optimizations in far less memory. While Watcom C is a good product, it doesn't live up to my expectations or needs.

Lattice C 6.0 is a great product (see Issue #51), and I really haven't had time to put it through its paces. The same goes for the recently arrived JPI TopSpeed C compiler. JPI claims to produce the fastest code in the business (who doesn't?), but I have some reservations about their claims. By the next issue, I should be able to put enough code through TopSpeed C to judge.

Compilers are less important to productivity than editors, in my humble opinion. Isn't the editor where a programmer spends most of his or her time? I use Microsoft's M editor the most; it's fast, I love the way it handles blocks, and it uses compiled as well as recorded macros. Alas, you can only get M if you buy a Microsoft compiler. It easily competes with Brief, Epsilon, and other professional editor programs. I think Microsoft is missing the boat by failing to put M on the open market.

If you don't have access to M, I highly recommend QEdit 2.08 from SemWare. It's a shareware product available from many BBS systems, including mine. Sammy Mitchell has created an excellent tool.

Something many programmers overlook is "make." A standard UNIX utility, make uses a file which contains a description of a program's construction, and uses that description to "build" the program. Since make only recompiles or relinks the parts of a program which have changed (by comparing the dates of various files), it can make program development much easier.

Almost all C compilers for MS-DOS come with a make. Many of these makes are weak; in particular, Microsoft's original make was laughable. While Microsoft has recently joined other vendors by providing a good make utility, none of the freebie makes can hold a candle to Opus Make.

Opus Software produces Opus Make, by far the most powerful make I've seen. It supports version control systems, libraries, initialization files, complex macros, multiple targets, and *all* the features of the original UNIX make.

Perhaps Opus Make's greatest strength is its use of an initialization file to automatically set conditions for a compile. My system currently contains 17 compilers for 9 different programming languages. The Opus Make initialization file on my system understands all these languages and flawlessly builds programs without my having to even think about the compilation process. Frankly, I couldn't program without Opus Make.

C EXPLORATIONS

ZipGraph (introduced in issue #51) has undergone an upgrade. The original version presented here last issue had some minor typos and bugs in the handling of 320x200 16 color and 640x480 2 color modes. The latest version is available on my BBS.

I spent much of November finishing my first book on C++ (see **Resources**) and didn't get a chance to finish the higher level code for ZipGraph. By the time you read this column, of course, you should find the new ZipGraph modules on my BBS. I will talk about them in Issue #53, where we'll begin looking at 3-dimension modelling.

You may remember my Critters program from a few issues back. Critters is a very simple simulation of life on a microscopic level, with critters crawling around a torroidal universe in search of food. As time goes on, and new generations of critters are born, they change their behavior by modifying their genetic material through random mutations. The critters actually begin exhibiting behavior as they evolve.

Critters was a slight diversion, and I never expected it to be popular. I looked at it as an interesting application to show how C could be used to solve a complex problem. Surprisingly, the response to Critters was overwhelming; I've never written such a popular program. And amidst the turmoil, I discovered a whole new science was evolving under the name Artificial Life (AL).

Ever since humans first began building things, we've been trying to duplicate life. Even simple structures like statues are an attempt to imitate real life. Our ability to generate lifelike forms has improved in parallel with improvements in our technology.

I'm not just talking about lifelike forms. To many people, artificial life means robots. While robots are imitations of living forms, the science of AL tries to get closer to the core of the issue: it tries to answer the *what*, *why*, and *how* of life. Larry may find the meaning of life deep inside the Mandelbrot set; AL researchers find it in the study of simulated life forms.

AL is quickly becoming an important adjunct to traditional biological research. Biologists have always been hampered by having only one form of biology to study—the carbon-chain-based variety.

There's no reason other types of life couldn't exist, and AL brings alternate life forms into the laboratory for study. In effect, we can create an artificial environment via a computer simulation and use it to better understand ourselves and the world of life around us.

Artificial Intelligence (AI) is related to Artificial Life, although the two sciences take opposite approaches. AI researchers work top-down. They begin by defining what intelligence is. Once they have a definition in hand, they write programs based on their definition. Neural networks are an example of this type of design.

AL, on the other hand, uses a bottom-up approach. A central theory behind AL is that behavior is the result of an amazing number of underlying—and simple—actions. In reality, life is the behavior exhibited by a massively parallel system.

What is a human being? In a strictly biological sense, we are a colony consisting of trillions of cells. These cells interact via simple rules. Light reaches part of our eye, which chemically reacts and sends electrical impulses to the brain, which then interprets the information. None of the electrochemical reactions is in itself complex; what they do together is amazing.

In AL, it is *how* the system behaves that is important. If you look at a beehive, you can see it as a single entity or as a colony of individual bees. Each bee has a specific, simple task. Each bee in a hive does its job—such as gathering nectar for producing honey—and interacts with its fellow bees.

An AL approach to simulating a beehive would be to create parallel processes which imitate the behavior of individual bees. Each "bee-omata" would know exactly what its job was and how it should interact with other bees. One bee-omata by itself is uninteresting; but put them together, and they act like a real beehive.

AL is involved with other life forms as well. The late Nobel Prize-winning physicist Richard Feynman was actively interested in nanotechnology. Nanotechnology is concerned with molecule-sized machines. After all, the components of our own bodies are simple, molecular machines.

Nanotechnology

With nanotechnology, we will build our own tiny robots, which we can use for everything from genetic surgery to wrecking the computer of the star ship Enterprise. (If you're not a Star Trek fan, some explanation is in order. A recent episode of *Star Trek: The Next Generation* revolved around "nanites" that got loose, causing havoc when they began eating the Enterprise's computer.)

Then, of course, there are cellular automata. A cellular automaton is a grid of individual cells, all of which have a state. The state of a cell changes in accordance with specific rules.

Conway's Game of Life, discussed here in the past, changes the state of a cell from live to dead by examining the state of the surrounding cells. All changes in a cellular automaton change simultaneously, at the tick of an imaginary clock.

While Life can be a fascinating diversion, cellular automata have many practical uses. They can simulate everything from gasses, to conductivity, to the growth of bacterial cultures. Some cellular automata can rival fractals in complexity and beauty.

A fractal image can be drastically altered by minor changes in the formula used to create the image; cellular automata are also very sensitive to the rules upon which they are built. However, cellular automata have one step up on fractals: fractals are static, while cellular automata are dynamic.

In the last few months, I've become heavily involved in AL. In fact, I'll be attending a conference in New Mexico on AL about the time you see this magazine.

Called Artificial Life II, the conference is sponsored by the Santa Fe Institute. The Santa Fe Institute is the pure-research arm of Los Alamos National Laboratories. The conference is held February 5-9 and will be attended by physicists, biologists, and computer scientists from all over the world. The proceedings of the first AL conference are available in book form; see the **Resources** section below.

For those interested in this field, I maintain copies of several AL programs on my BBS. Feel free to drop by and browse around. By the time you see this article, the advanced version of Critters should be available. I've rewritten the program in C++ and have added predators and scavengers. I'll demo the program in Santa Fe.

NEWS AND REVIEWS

AutoDesk produces the most famous CAD program for PCs, AutoCAD. This is an expensive product (costing many thousands of dollars), and it requires a powerful PC. I never expected AutoDesk to also be the price/performance leader in other areas of programming endeavors.

For under $60, Rudy Rucker's Cellular Automata Laboratory may be the best bargain on the market. Rucker is a science fiction writer turned computer scientist. He became fascinated with cellular automata and has since written this package for AutoDesk in conjunction with Autodesk co-founder John Walker. It contains two programs: Rucker's "RC," and Walker's "CA."

RC is a simplified program, designed for people just beginning to experiment with cellular automata. It uses a text display, instead of graphics, to show the automata. This means you can use RC on any IBM-compatible, even if it doesn't support graphics. I like this program because it allows you to make instant changes in the dynamics of the cellular automata. It's also very fast.

CA is much more complex and can simulate an almost infinite number of cellular automata. It works in the 320x200 mode of IBM-standard video cards and will operate in 256 colors on a VGA system. CA supports more variations than does RC, and it can be "programmed" using traditional languages such as C or Turbo Pascal.

Both these programs come with numerous demonstrations, predesigned rules, and hints. The documentation, written by Rucker, is the best introduction to cellular automata I've yet run across. If you want to see a leading edge of computer science research, there's no better place to start than with Rucker's CA Lab.

Alas, I wish I had more time to sing the praises of Autodesk's other recent release, Animator. Using a standard VGA card in 320x200 256 color mode, Animator allows you to build realistic and professional animations on a PC. This finally gives owners of IBM-compatibles the facilities owners of Amigas, Suns, and Apollos have had. The capabilities and design of Animator have quite impressed my wife Maria, who's a semiprofessional animator. For under $300, it's a bargain.

RESOURCES

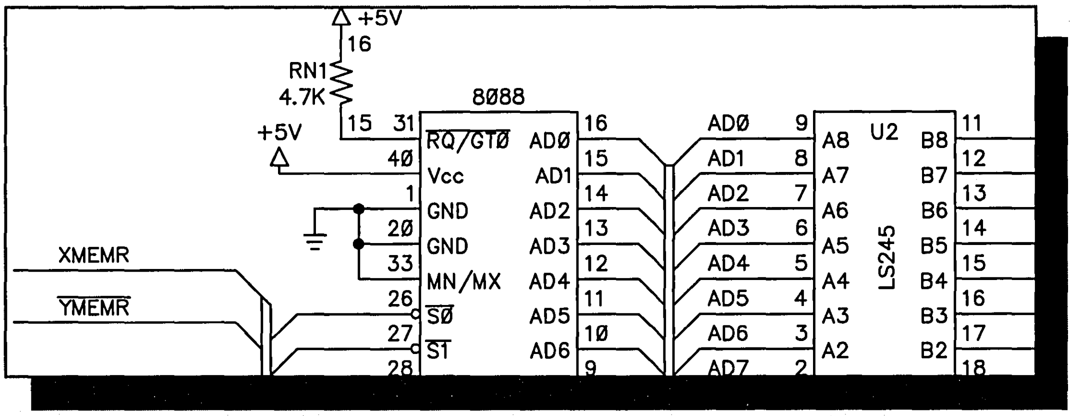
As mentioned above, I've recently finished a book on C++ entitled *C++ Applications and Techniques* for M&T Press (the publishers of *Dr. Dobb's Journal*). This isn't designed as a C++ tutorial; that job was ably done by Stanley Lippman in his recent book, *C++ Primer* (Addison Wesley, ISBN 0-201-16487-6).

My book tries to show you how to *use* C++ in practical applications. In particular, I gave considerable attention to the how and why of object-oriented programming. I hope you have as much fun reading it as I had writing it.

If you really want to understand Artificial Life, you need to get the proceedings from the first AL conference, held in 1987. The book, called (boringly enough) *Artificial Life*, is available from Addison-Wesley (ISBN 0-201-09356-1). There are 650 pages of papers on every aspect of AL, written by the people who are making this new science. I found this book the most stimulating I've read in the last year.

♦♦♦

XT SCHEMATIC



At last you can plumb the mysteries of your computer with this single sheet (24" x 36") schematic of the IBM XT's main board. A wealth of information for both True Blue *and* clone owners.

Although clones use slightly altered board layouts and different chip location names, they're close enough to the original for this schematic to be very useful. As an example — you have a dead clone. Lil sucker won't even beep. A look at the schematic shows the location of parallel port A. You know that the power-on self test loads a checkpoint number into port A before each test. So now all you have to do is read port A with a logic probe to see how far the system went before it died.

We include these checkpoints and other trouble shooting information with the schematic.

IBM PC-XT Schematic.. **$15.00**

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

Dept. 52, P.O. Box 223, Bend, Oregon 97709

Monday-Friday, 9AM-5PM PST

1-800-888-8087

FAX: 1-503-389-6833 (24 hours)

VISA

MC

ON YOUR OWN

Computer Forensics

The Quincy Of Computers

By Allyn R. Franklin
Drive Masters
6275 Canterbury Dr. #204
Culver City, CA 90230
(213) 645-2563

If you're looking for a (very) legal way to make a living, then this is the business you'll want to court. (But judge for yourself.)

I'm sitting at my bench fixing my 1,356,490,231st drive (we keep good records) when the phone rings. It's a man in distress. (All these mysteries start the same way. Well, not all. Occasionally it's a voluptuous blonde who bursts through the door unannounced. She usually has a portable in hand or problems with her laptop.)

"Can you come out right away and rescue this crashed hard disk for one of our customers?"

"Crashed?" I ask.

"Yeah, that's what the customer said."

I tell him I rarely do field service anymore. Could he bring it into the shop next week? I keep all my diagnostic software and hardware here. I further explain that the word "crashed" is used so loosely that even when I did field service I was never really sure what to take with me.

"Am I supposed to repair this drive?"

"If necessary, but first see if you can get the data off."

Then, of course he insists he can't wait until next week, he needs an opinion or evaluation immediately.

"Can we come over today or tomorrow?"

The Typical Assignment

I ask further what this is all about. He explains that he works for an engineering company. His company has been hired by a computer service company's attorney to find out if loss of data on a hard drive was due to: 1) machine failure, 2) software problems, 3) human error, or 4) a combination of the above. They would also like me to recover the data, if possible.

He's just defined the field of computer forensics. Engineering companies have applied the term "forensics" to accident investigation and reconstruction. They also use the term while drafting reports for lawsuits and insurance cases.

Dictionary definitions include such phrases as "argumentative exercise," "the art or study of argumentative discourse," or "suitable for a court of law or public debate."

Recognizing Needs And Finding Niches

I've been very active in the Los Angeles Kaypro User's Group and the U.C.L.A. PC User's Group. I suppose that makes me bisystemal. I started Drive Masters about seven years ago, and at that time people told me it was crazy to fix drives. They figured it was better or cheaper to buy new ones.

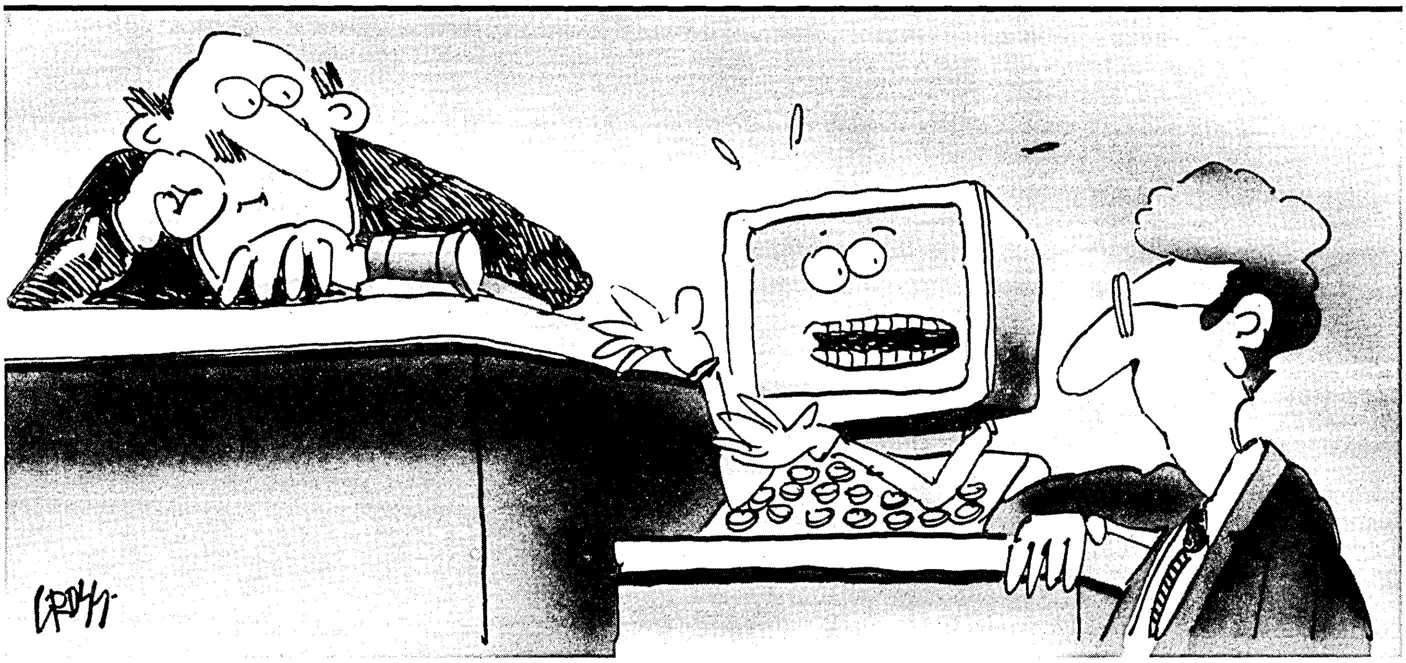
Since I couldn't see why people would throw something away when it could be easily fixed, I started working in my first niche. My instructor at Tandon said he didn't know of anyone else doing it. So ... here I am seven years later ... still finding new niches for my trade (to keep from getting bored).

My Current Field

It seems obvious that the more parts a machine has, the greater the chance something will go wrong. Generally, moving parts wear out faster than the other kind, but computers have an awful lot of both. Furthermore, computers cannot run without software and firmware, and these can fail, too. Finally, we stir in a few humans....

If you're not in the service business, you cannot imagine how some people react when their computer goes down. Sometimes entire businesses grind to a halt. Owners lose large sums of money as well as their tempers. Most people are like early automobile owners, they haven't the slightest idea why the computer works or why it doesn't work.

This means there is great fear, suspicion, and doubt about the equipment and the people who service it. And considering the financial consequences of down systems, there are bound to be lawsuits and insurance claims. This creates the need for the computer forensics consultant. Me. I'm retained by plaintiffs, defendants, claimants, anyone.



Meanwhile, Back At The Drive

I told the caller that I needed not only the drive, but also the controller card and a copy of the operating system—apparently I couldn't get the whole computer. (As I later learned, no one really understood the significance of my request.)

First, the drive arrived. Finally, a controller card showed up. Legal documents specified that the drive had been running under MS-DOS 3.1. And, the documents specified the original problem: the drive had not allowed an employee access one morning.

When the original problem occurred, the company had called a technician from the service company who "ran tests." The technician concluded that the drive had crashed so he replaced it. Then the technician began running DOS RECOVER on the problem drive. He intended to let RECOVER restore all the files rather than just the main datafile the client most needed.

(That's not the way I'd use RECOVER. In fact, since running RECOVER globally takes all the FAT chains and makes files out of them, you wind up with a mess. Mace's remedy and unformat work much better. I try my best to do stuff that doesn't destroy anything before progressing to the really heavy activities.)

Part way through the RECOVER, the business owner insisted the technician stop. (The owner apparently had heard some nasty things about RECOVER and wanted to get a second opinion.)

A few months later, the owner filed suit against the service company for loss of revenue, cost of reentering a large database, etc. He then turned the drive over to his attorney. Mind you, Watson, I knew none of this until I read the owner's depositions.

So I have a hard drive, a controller, and two clients: the service company and its attorney. Analyzing the drive, I conclude that the last thing someone did was a low level format! (Makes it tough to recover data.)

I'm working with the service company's defense attorney as an expert witness. As a witness, I have to go to the city to give my deposition—3½ hours counting breaks. I rather enjoyed tearing the plaintiff's case apart. Following the testimony, I received a call from our attorney telling me that the plaintiff has asked for a continuance to call in another expert (to rebut my expert opinions).

If you're not in the service business, you cannot imagine how some people react when their computer goes down.

We learn that the controller card they sent me was not the brand or model used in the owner's machine and the original DOS was actually 2.11! This information had been specified by the plaintiff earlier, but not under oath.

A technician who works at the hard drive manufacturer comes in for a deposition. Meanwhile, I educate our attorney (he's not even a computer owner) on how to get the information we need. I'm not allowed in the room during depositions since I'm a witness. (They do, however, allow consultants to coach attorneys during depositions.) This is the information discovery phase prior to going to court.

So, How Did It End?

The plaintiff dropped the suit, agreeing to settle out of court for under 20% of the original claim, just enough to cover his attorney fees.

I think computers have joined the ranks of car crashes, slippery floors, weak bridges and buildings, and many other appliances and products for which both manufacturers and service providers will be held liable for performance. Along with this data processing age comes a new profession: the computer forensics consultant. Me.

♦ ♦ ♦

5. Fix the problems and send out copies under your own name.

There's one risk. Reviewers might ignore the bugs and the targeted company might snare a copy and start shipping. (Makes me wonder who really wrote dBASE IV and 1-2-3 version 3.)

Not So Simple Editors

It's fun to watch the new word processors. As they get fancier and fancier, the files they generate get more and more interesting.

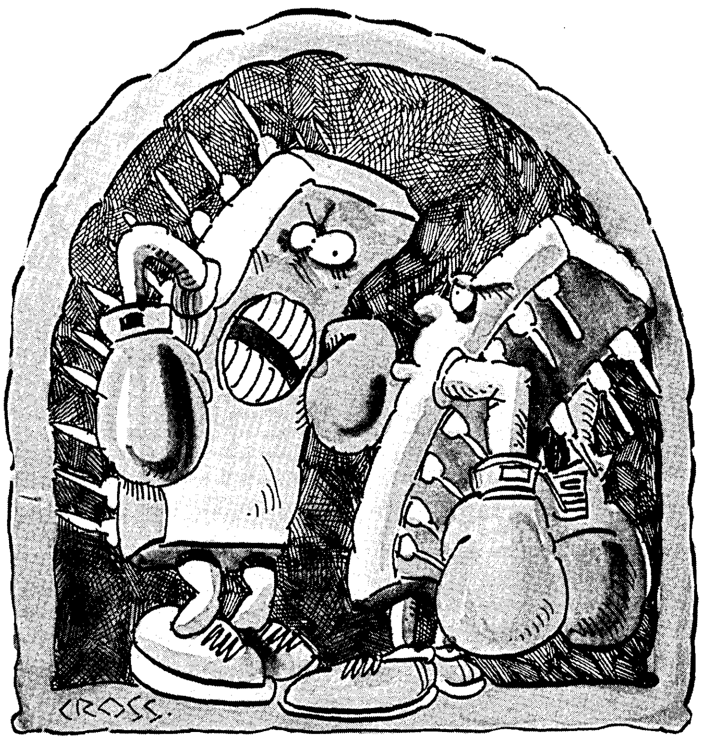
A few weeks ago Sandy had trouble making Word Perfect 5.0 accept a merge file. She had two files: one contained a form, the other contained a list of *Micro C* advertisers. She wanted WP to merge the two and output the results on our PostScript printer.

Well, WP suddenly got very creative. It produced about a third of the forms correctly, the rest had any number of weirdnesses. Sometimes there were letters missing in the names, sometimes the type face changed right in the middle of a line, sometimes lines disappeared entirely. Each time she reran the merge, the problems changed. By running the merge five times, she got a clean copy of all but one form.

Great, if you like running five merges.

Then she tried outputting the merge onto a daisy wheel printer. It worked. First time. However, the daisy couldn't reproduce the graphics. If she wanted graphics she had to output PostScript, and when she output PostScript....

Ah, well. If PostScript gets popular, WP will probably clean up its driver.



Clone CoProcessors

Intel may have shared its 8086, 186, and 286 technology with second sourcers, but it has carefully kept its coprocessors to itself. Sure, 80286s might be had for a pittance, but be prepared to take out a second mortgage to get an 80287.

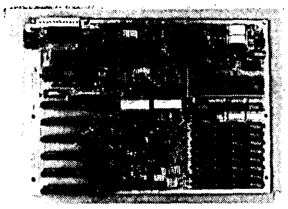
Now it appears that Intel's $300/chip has attracted some competition, and I noticed in the November 27th Issue of *Info World* that Intel has taken off the gloves.

Intel doesn't feel, for instance, that Information Technology's IIT-2C87 is really compatible. Something about not getting "identical numerical results" during tests with 1-2-3 and AutoCAD. IT responded by saying that its chip meets or exceeds the IEEE standards and that the differences resulted from the chip's greater accuracy. (Intel says it will be running tests on other coprocessor clones soon. I can't wait.)

I'd be interested in an article on math coprocessor clones, especially with some information on how good, how fast, and how cheap. Heck, gooder, faster, and cheaper can't be all bad.

Meanwhile, with the 80486's continuing bugs and its initial price of $1,000 a pop, there might be an independent who could try its hand at processor design. And it might even work.

68000 SINGLE BOARD COMPUTER



- 512/1024K DRAM
- FLOPPY DISK CONTROLLER
- 4 RS-232 SERIAL PORTS
- REAL TIME CLOCK

NEW LOWER PRICES!

BASIC KIT (8MHZ) - BOARD, MICROPROCESSOR, HUMBUG MONITOR/ BASIC IN ROM, 4K SRAM, 2 SERIAL PORTS $200

PACKAGE DEAL - COMPLETE KIT WITH 10MHZ MICROPROCESSOR, SK*DOS OPERATING SYSTEM, 512K DRAM $499

SYSTEM BOARD (12MHZ) - ASSEMBLED/TESTED, 1MEG RAM, 6 PC/XT PERIPHERAL PORTS, SK*DOS $699

COMPLETE INFORMATION AVAILABLE UPON REQUEST

PERIPHERAL TECHNOLOGY
1710 CUMBERLAND POINT DR., #8
MARIETTA, GA 30067 404/984-0742

COD/MASTER CARD/VISA/CHECK

SK*DOS IS A TRADEMARK OF STAR-K SOFTWARE SYSTEMS CORP.

Reader Service Number 119

Thankyou Thankyou

Last issue I mentioned I was looking through the Micro C software cupboard for a copy of Microsoft C 5.1 to check out C-Thru-ROM. Unfortunately, we didn't have a copy. Within hours of our release of 51.0 (okay, the January/February issue), I received a *huge* brown box from Microsoft.

I know what they put in boxes that size. Everything you'd ever want to C.

I opened it up.

"Microsoft Announces Tool for Powerful Application Development:"

Wow, must be a whole new release!

"The Microsoft BASIC Professional Development System."

Gee. They shouldn't have.

(I can't complain, though. I called them up, asked for a C, and bingo, 5.1 showed up in under 24 hours.)

Beware Of Databases

I had a follow-up appointment with my doctor (he wanted to see how his latest project—me—was working). After sitting in the waiting room the prerequisite half hour, I was ushered into the fearsome little cubicle decked out with all the nasty apparatus. Usually I get to commune with the nasties for another 45 minutes, but this time he appeared almost instantly.

"How are we feeling today?"

"We're fine."

"Any more symptoms?"

"Not that we're aware of."

"Great. Should I purchase a 286 for the office or a 386?"

It turned out he was planning to purchase a medical package and he could get the software and a 386 clone together for $16,000 or the software alone for a mere $7,000. I figured he could pick up a 386 for less than $9,000, but he brought up a good point. If he purchases the hardware and software separately and there's a problem, he could be subjected to a lot of finger pointing rather than action. (It's his fault, no it's his....)

I mentioned that if hardware were going to be a problem, it would probably fail quickly. The software, especially when it's a database package, will usually wait a year or two.

Why do database packages fail? They get confused. After a year or two inserting new patients, deleting others, multiplying bills, figuring diagnoses, changing diagnoses, changing diagnoses again, little nasties start appearing. (I don't mean viruses or warts or anything like that, just nasties.)

A local dentist friend found out all about that. He'd decided to purchase a patient management package. (Designed for those specializing in orthodontics and other medieval tortures.) When he called around, most of his friends said there was one package they really loved.

A year later, however, those same orthodontists were bracing themselves for a barrage of unretainered clients. They'd

---

ICs PROMPT DELIVERY!!!
SAME DAY SHIPPING (USUALLY)
QUANTITY ONE PRICES SHOWN for DEC. 31, 1989

OUTSIDE OKLAHOMA: NO SALES TAX

DYNAMIC RAM			
SIMM	AST Prem386/33Mhz		$275.00
SIMM (1)	256Kx36	80 ns	275.00
SIMM (2)	1Mx9	80 ns	110.00
SIMM (2)	256Kx9	100 ns	35.00
1Mbit	1Mx1	80 ns	10.25
41256	256Kx1	60 ns	4.65
41256	256Kx1	80 ns	3.40
41256	256Kx1	100 ns	2.40
41256	256Kx1	120 ns	2.15
4464	64Kx4	120 ns	3.75
41264 (3)	64Kx4	100 ns	7.50
EPROM			
27C1000	128Kx8	200 ns	$18.00
27512	64Kx8	200 ns	7.80
27256	32Kx8	150 ns	6.50
27128	16Kx8	250 ns	3.75
STATIC RAM			
62256P-10	32Kx8	100 ns	$10.95
6264P-12	8Kx8	120 ns	4.50
6116AP-12	2Kx8	120 ns	4.25

(1) PS2/70 PS2/80 30/286 1Mbyte
(2) FOR: AST AT&T COMPAQ DELL MAC'S MYLEX PS/2 Tandy WYSE WD
(3) 2-PORT VIDEO RAM

IIT MATH COPROCESSORS
80387-25 80387-33
80387-16 80387-20
80287-8 $210.00
8087-2 $110.00

OPEN 6 DAYS, 7:30 AM-10 PM: SHIP VIA FED-EX ON SAT.

SAT DELIVERY INCLUDED ON FED-EX ORDERS RECEIVED BY:
Th: S-2 $6.25/4 lb
Fr: P-1 $14.25/1 lb

MasterCard/VISA or UPS CASH COD
MICROPROCESSORS UNLIMITED, INC.
24,000 S. Peoria Ave.,
BEGGS, OK. 74421 **(918) 267-4961**
No minimum order. Please note: prices subject to change!
Shipping, insurance extra, up to $1 for packing materials.

Reader Service Number 37

If you thought Borland's popup product was great just wait until you see OpalFire's MyFLIN for Pascal.

MyFLIN is a TSR program that captures procedure and function details directly from the screen while you are programming. It saves this information in an indexed database for instant recall at any time you need it.

MyFLIN will save you hours of searching thru pieces of paper looking for parameter details when calling a procedure. Simply type part or all of the name and press the hotkey. **MyFLIN** will popup over your source code and display the nearest named description you have stored in your database, complete with parameters and comments.

To save a new description, positon the cursor on the procedure name, popup **MyFLIN** and press "A" to add. The details will be captured from the screen and saved with a single keystroke and you can add a comment line as well.

Price $69.00 + $5.00 P&Post.
Visa / Mastercard / American Express are accepted.

OpalFire Software Inc.
329 North State Street,OREM, UTAH 84057
Phone 1-800-336-6644
MyFLIN requires PC/XT/AT Computer and MS-/PC - DOS 2/3.xx.

Reader Service Number 161

been over-billing some, under-billing others, and losing patients (patience) altogether. (But getting my dentist friend to change to another system has been like pulling teeth.)

Meanwhile back at the office consultation: as I was describing the dentist's experience, my doctor mentioned that he had just talked to a physician whose Mac had just eaten half his customers. What a headache.

I'll bet somewhere out there, there's a dental programmer who wishes he'd kept his stacks a little cleaner and his variables a little more local. (More than a few dentists would give their eyeteeth for a two-hour crack at the fellow's molars.)

Marketing

If you follow "On Your Own," you know that last issue's tome by Kim and Don Jindra was really a handbook on marketing software.

What you might not have realized is that the article itself turned out to be the most effective marketing they've done for Information Modes. (They're selling a $25 network as well as a group of other technical products.)

The January/February issue of *Micro C* was mailed the first of December. "By December 14 we'd surpassed our best month ever. In fact, the first week was incredible, totally amazing. We knew your readers were extremely loyal, but we had no idea.

"Kim's been working day and night just filling orders."

No Rocky Mountain SOG

I've just been notified by the Aspinal Wilson Center and by Scenic River Tours that as of press time (January 8), they have not been paid for last year's Rocky Mountain SOG. Because of this, I can't, in good conscience, lend Micro C's name to a 1990 SOG in Gunnison.

This has been a most difficult decision. Scott Ladd has been a strong SOG supporter and he worked hard to make the first Rocky Mountain SOG a success. It was a success, but attendees paid to go rafting, to attend a barbeque and banquet, and to use the facilities at the Aspinal Center. That money has not been passed along.

However, you'll have your choice of three other SOGs this year (they're all looking for speakers).

Longhorn SOG

Friday & Saturday, June 29th & 30th in Denton, Texas (just North of Dallas) Contact:
Kim Jindra
Information Modes 817-387-3339

SOG East

Friday & Saturday, Aug. 17th & 18th in York Pa. Contact:
John Ribar
717-854-3861 8-9am Eastern
717-729-5108 8-12pm Eastern

ATTENTION ASSEMBLY LANGUAGE PROGRAMMERS

BUILDING BLOCKS FOR THE ASSEMBLY LANGUAGE PROGRAMMER

QUANTASM POWER LIB

Price $99.95
With Source Code $299.95

S/H $5.00
Outside U.S. $15.00
CA Residence Add 7% Sales Tax
MC and VISA

Windows
Menus Screen
Cursor Date Time
Filename Strings Sound Math

- *Over 256 Assembly Language Routines*
- *No Need to Re-Invent the Wheel Every Time You Write a New Application*
- *Permits Faster Application Development*
- *Overlapping Windowing System in Less Than 3K*
- *Fast and Compact*
- *Ideal For Programming TSR's*
- *C and Pascal Interfaces Available*
- *MASM 5.X Compatible*

QUANTASM CORPORATION
19855 Srevens Creek Blvd, Suite 154
Cupertino, CA 95014
(408) 244-6826

ALSO AVAILABLE:
ASMFLOW $99.95
An Assembly Language Flow Charting and Source Code Analysis Tool

Reader Service Number 139

Are You Moving?

Take

Micro Cornucopia

With You.

Please send us your new address at:
Micro Cornucopia
PO Box 223
Bend, OR 97709

Old Address
Name: ____________________
Street: ____________________
City: ____________________
State: ____________________
Zip Code: ____________________

New Address
Street: ____________________
City: ____________________
State: ____________________
Zip Code: ____________________

Robo-SOG

Set aside August 9-12 to visit the Seattle area and be a part of our first SOG. Robo-SOG.

Sponsored by the Seattle Robotics Society, Robo-SOG '90 will offer much of what you expect from a SOG: quality talks (let us know if you'd like to speak), barbeque, banquet, fresh air, and, of course the Jolt-SIGs.

We'll also have maze-running robots, camping (for those so reclined), and a Junque de jour tour. The robotics maze-run is open to all (robots). If you would like more information on preparing your metal monster to do battle, drop us a postcard.

We'll be sending out complete details on prices, location, accommodations, and schedules after March 15. We'll also have maze specifications and contest guidelines at that time.

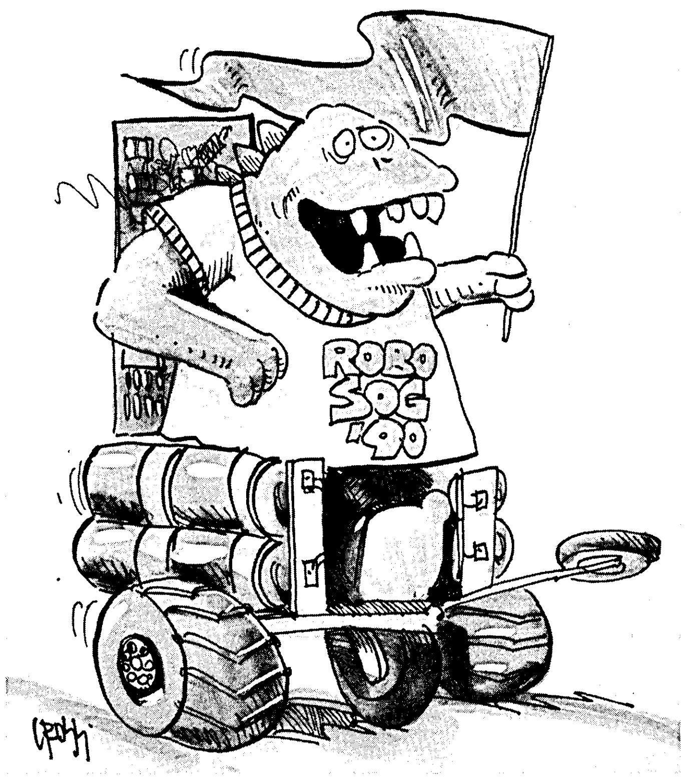
Watch the pages of *Micro C* for more details.

Mike Thyng
11036 40th NE
Seattle, WA 98125
Voice: (206) 362-5373
BBS: (206) 362-5267

[illegible]

David J. Thompson, Editor

♦ ♦ ♦



Show Us Your Stuff!

Larry, Carol, and I (Dave) were chatting it up in the graphics department when Carol suggested we have readers send in samples of the images they're creating on computer. (That's what happens when you plan issues a year ahead, people take the subjects seriously.) We really liked the idea, now it's your turn.

Send us your graphics. What kind of graphics? All kinds: mechanical renderings, scanned and edited pixel images, vector drawings, pixel drawings, anything.

Hard copy is easiest for us to view but we'll take files also. (Or even better, send both files and hard copy.) If you send files, send all the information about how you created them. If you used a commercial drawing package, include its name and version.

Mail should be sent to:

Graphics
Micro Cornucopia
P.O. Box 223
Bend, Oregon 97709

Or leave files on the Micro C BBS (503) 382-7643.

We'd like to see your stuff by the first of June. That way we'll have two months to view and choose the art. Plus, it'll give us a chance to get back to you for further information or even an article.

Any questions? Call Dave, Carol, or Larry at Micro C, (503) 382-8048, or get on the Micro C BBS, (503) 382-7643, and leave a message for the SYSOP (Larry).

EGA VGA SVGA **Unix Xenix**
Graphics Library

C Source and Binary Versions: *both include C source to a full-featured GIF viewer and several other demonstration programs.*

No CGI or BIOS - Direct Video Card Access: *Fast!*

12 Video Modes: *including 800x600x16 and 360x480x256 on VGA systems, and 640x400x256 on Paradise SVGA cards.*

Multiple Active Graphics Virtual Terminals: *Compatible with X Windows and VP/ix.*

Primitives: *point, line, solid, bitblt, clipping, ellipse, more. Twelve alu operations (really) in 16 color video modes. And programs can be written independent of video mode and adapter.*

Software Cursor for Bus Mouse: *port X Window clients to a high speed dedicated screen (just one idea)!*

Fifty BDF Fonts: *plus font compiler and examination utilities.*

Fully Documented.

Operating System Version	Source	Binary
Interactive 386/ix (all versions)	$199	$99
Xenix/386 2.3	$199	$99
AT&T System V/386	$199	$99
others	CALL	CALL
GIF viewer binary (specify Unix version)		$39

Send for More Information and a Stunning Demo!

JDyx *Enterprises*
907 Tuxworth Circle, Decatur, GA 30033
(404) 320-7624 BBS: (404) 321-5020

Reader Service Number 191

Letters *continued from page 6*

tainly beats shelling out for a monster monitor!

Andrew Marchant Shapiro
815 Hattie St.
Schenectady, NY 12308

But Will It Do Windows?

I read your mag. I understand some of it, but why have a robot project for such an inane thing as maze running? Don't misunderstand. I want a robot, and *now*. But a mere maze solver isn't worth the effort. I will build or buy a robot only if it will clean my bathroom.

Let it be schedulable weekly for the porcelain, counters, and floors; monthly for the walls and ceiling. Let it do everything within the doorway except towels. I hope it isn't too expensive because I want it *now*.

A. K. Holmes
4210 7th Ave. N. E.
Seattle, WA 98105

Random Ramblings

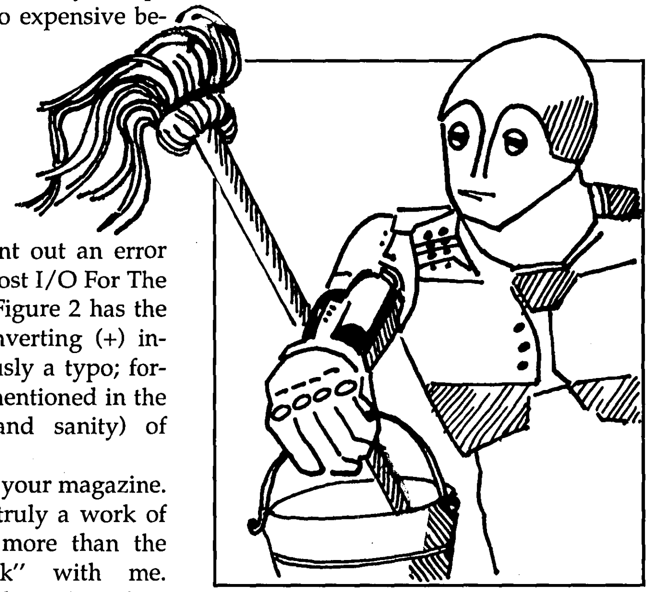
Thought I might point out an error in Bruce Eckel's "Low Cost I/O For The PC" article (Issue #49). Figure 2 has the inverting (-) and non-inverting (+) inputs transposed. Obviously a typo; forgivable, but should be mentioned in the errata for the sake (and sanity) of newcomers.

Now for a beef about your magazine. The new cover style is truly a work of art but, unfortunately, more than the concepts within "stick" with me. During the course of a heated reading frenzy, anxious perspiration develops on my quivering fingers (remember your first love?). Either I have a truly unique metabolism, or common sweat is an excellent solvent for your cover ink (front and back, all colors). Sorry, gloves slow me down; the solution (no pun intended) is up to you.

Perhaps that Fogg fella might detour off the fractal fanaticism track for a while. Do we really need encouragement to waste time? Clever ploy using excellent PostScript info as an excuse to diddle with more fractals; two points for Larry.

Might I also suggest publishing more useful programs that demonstrate the nature and advantages of Object Oriented Programming (preferably in C++ and Oh Oh Pascal). As the wave of the future, it behooves us to keep abreast of this technology lest we be left behind. The ether is too thin in this area, and the more others understand, the more the rest of us will understand (ref: "The 100th Monkey").

Based on the sterling reviews *Micro C* and other journals bestowed on the OrCAD CAD software packages, I purchased both their schematic design and printed circuit board layout packages. While I am duly impressed with the vast (and I do mean *vast*) range of displays, printers, and plotters they support, and the professional results that can be obtained, I must disagree with the comment about "doing useful work in an hour."



One can generate a lot of work in an hour or two, but be forewarned: it may not be useful. For drawing schematics and such, the SDT III is a breeze and a joy to use. For using that work with the layout software, there can be pitfalls. Thankfully, their technical support *is* as good as reviewed, and they really do send updates to people besides reviewers without being asked!

My major complaints are the documentation and limitations of the layout package. A chapter on the overall philosophy of the package would double its worth.

As a user, the question comes to mind: why bother to support a board up to 32" by 32" when it has a devil of a time handling a board 1/8 that size? Another bitch is having zero info on file formats. There are many occasions where a small self-written utility could save a heap of work. Hey, I'm over 30 and time's growing too short to have to hack every damn thing just to have a fighting chance.

So, just thought I'd mention the typo in Bruce's article. Look at the time! Did my mind wander again?

Dave Stojan
10310 Lybert Rd.
Houston, TX 77041

Editor's note: Actually, I think the schematic is correct. Normally the - input is placed on top, which might have thrown you a bit. (That's why we've ignored your letter.) As for dissolving our petroleum-based, superstick, cover ink with your bare hands, there might be a job for you with Exxon's Alaskan division.

Return To The Radar Equation

Sigh. You've finally done it: printed another letter about the great optical sensor debate, and forced me to write (see Letters, Issue #50).

Let's ignore the fact that for at least one detector the transmitter range is not the same as the receiver range, so you can't just multiply the two inverse square ranges and call this an inverse fourth. The by-now semi-mythical radar equation deals with power, and goes something like this: first you transmit power, then you reflect power, and finally you detect power. Now to get specific.

A transmitter radiates power over some beam angle, and at a given range produces a power density PD1, proportional to the inverse square of the range. This power is reflected by a target with area AT. Thus the reflected power PR is equal to PD1 x AT and is proportional to $1/R$ squared.

The reflected power produces a power density PD2 at the detector, which has an area of AD. Total power at the detector PD is equal to PD2 x AD, and the fraction of PR which reaches the detector is also proportional to $1/R$ squared. So the total power at the detector is proportional to PR/R squared or $1/R$ cubed. Simple, yes?

No. Sorry, folks. The inverse square relationship for PR assumes that the target is smaller than the beam at that range: in other words, that the target is unresolved. This is correct for most radars, which look at long ranges using long wavelengths, but it doesn't necessarily apply to optical systems. It certainly doesn't apply to a maze running robot which just wants to keep from running into walls.

If the target (such as a wall) is bigger than the transmit beam, *all* of the transmitted power will be reflected. The final power at the detector is proportional only to the inverse square of the range.

I won't go into details, but you can show that if the target is resolved by either the transmitting or the receiving system (or both), the overall response of the system is an inverse square relationship.

It gets worse. If you use a bare detector (no lens in front), the 1/R squared relationship for PD2 is not valid unless the illuminated target is pointlike. The usual rule of thumb calls for a distance of ten times the target diameter, which for this application requires a fairly small spot.

You can put a lens in front of the detector to get around this, but then you have to focus the lens. If you could do that, you'd know the range to the target already.

Also, the inverse square law does not necessarily apply to a transmitting lens system. Indeed, any simple lens which focuses a point source to a spot smaller than the lens will produce an area in which the power density increases with increasing range.

My suggestions? Use a simple lens in front of the transmitter LED to get as small a spot as possible. Use a bare detector with a snout to restrict its field of view.

If you don't chop the LED output, AC couple the detector, and put an IR filter over it, things like patches of sunlight will eat you for lunch. Even better, go to phase sensitive detection.

Oh, and one more thing (he says, turning around in his rumpled trench coat): pay close attention to the article in Issue #50. You're going to need that bumper plate.

James Martin
Merriam Hill Road
Greenville, NH 03048

CONSIDERING THE COMPUTER: TIP # 101
BE CONSIDERATE TO YOUR MACHINE; GET DRESSED, BRUSH YOUR TEETH, AVOID GRIMACING...

Yo Ho

Yo! (Generic young-adult salutation.) Well gosh, I never thought I'd find myself writing my favorite magazine ... etc., etc. Of all the magazines I read each month, only youse guys and Steve Ciarcia's *Circuit Cellar Ink* make me feel the same way that I felt after my first date.

As for your content, keep it up! The casual, occasionally flippant attitude, the philosophic bent, etc., sum up to one word: Fun. The magazine is fun to read and it appears that it's fun to write.

BYTE and *DDJ* used to be fun, but.... It's like watching my older friends graduate: "Well, I guess I'll cut my hair above my collar, start shaving, mothball my jeans and T-shirts. And Oh hey! Men's Warehouse has a special deal on suits this week!"

Some advice *not* to take from the new money-men:

- "If you cut back your editorial to one page, you can devote more pages to articles." (Hah!)
- "This is not relevant. This is a *computer* mag, after all."
- "You look tired. I know a good editor from PC??? mag who's dying to help out."

As for Issue #51's new paper type: I like it. What's good for *Micro C* is good for me.

Larry Liska
182 Loch Lomond
League City, Texas 77573

Applause For The Poet

I would like to applaud your magazine for carrying the article "The Poet and the Computer" by Norman Cousins in Issue #51. *Micro C* has a marvelous blend of articles which cover every aspect of computing. Very few magazines would publish an article on so philosophical a topic, especially one which questioned the desirability of even having a computer.

I think it is important to address the role that computers have in society and to see that they are used to further man's development, not retard it.

I also enjoy the nuts-and-bolts approach in your articles. They are not written to dazzle the reader, but to show how something works. The enthusiasm which the authors have for their subjects comes through. Congratulations on your Renaissance computer magazine.

Dave Christy
2039 Hudnut Rd.
Schwenksville, PA 19473

♦ ♦ ♦

UNITS & MODULES

Writing A C Simulator & Compiler

Writing Microcontroller Tools In Modula-II

By Michael S. Hunt
2313 N. 20th
Boise, ID 83702
(208) 336-7413

Boy, what's this crazy column come to? Let's see, the authors (yep, authors) are writing a C cross compiler in Modula-II so folks can put together microcontrollers. Only in Micro C.

I have always wanted to learn more about computer hardware. When I got to Rocky Mountain SOG, I was pleased to see Karl Lunt giving two hardware talks. After his second talk, I asked Karl how a software person like myself could make the plunge. Karl mentioned that a high level language compiler for the microcontrollers would make the learning experience much easier for beginners.

We discussed the idea again that night with Dave at the Jolt SIG. Soon we decided that I would write a compiler for the MC68HC705C8 microcontroller and Karl would develop some projects using the compiler. Dave got off easy, all he has to do is publish the results.

Just as I was gearing up to write my first compiler, Micro C called. John Ribar of SOG East fame was interested in joining the project. John has more compiler experience than I do, so he took over the compiler part of the project. That leaves nothing to do but write a software simulator for the 'HC705.

Inside The Simulator

The simulator lets you develop and debug 'HC705 programs. To be useful the simulator must do many things. It must display any memory location, stack and registers included, validate all instructions and addresses, and count clock cycles. It is essentially a debugger.

The memory structure of the simulator (see Figure 1) can handle up to 16K even though the 'HC705 has only 8K of memory. This makes it easier for the simulator to support the rest of the HC05 micro controllers.

A memory element consists of three fields: first, the memory value; second, is a read/write read/only flag; and the third flags the element if it's used for the current chip configuration.

Stony Brook's Modula-2 has structured constants. The definition of the instruction set (see Figure 2) uses a structured constant to initialize the opcode, number of bytes to increment the program counter, and the number of clock cycles the instruction takes to execute.

The eight instructions I've shown in Figure 2 are only a sample of the 209 total. The record for the instruction set also includes a procedure variable that points to the procedure that will actually execute the instruction. The procedures are defined separately so that each will be easy to maintain (see Figure 3).

The instruction set array is initialized one element at a time (see Figure 4). It is assigned the constant whose opcode matches its element index. The instruction procedures are assigned according to the opcode they represent (see Figure 5).

With only 209 instructions, that leaves 47 array elements (out of 256) unused. Their procedure variables are initialized to an error handling routine (see Figure 6).

The instructions could have been executed using a large case statement, but that would have been unwieldy and inefficient. Or I could have stored the procedures in an array, sorted by opcode. This method is better; but because not all opcodes are used, a binary search must be used to access the procedures.

So I decided to store the procedures in array elements that corresponded to their opcodes. This let me directly execute the procedure and it let me trap any invalid opcodes.

Notes From John Ribar

I'm not doing anything tricky in my compiler development. At first, I thought that using a "compiler compiler" would make the project simpler, but then the project would involve yet another thing to learn.

So I'm building a handmade compiler. This takes a bit more work but is a better learning experience and usually produces superior results.

I'm using TopSpeed Modula-2, and my implementation covers a basic subset of the C

Figure 1—Simulator Memory Structure

```
DEFINITION MODULE MemHC05;

TYPE
    Mem_Type = RECORD
        mem  : CHAR;        (* memory value    *)
        r_w  : BOOLEAN;  (* read/write flag *)
        used : BOOLEAN   (* usable flag     *)
    END;

VAR
    Memory : ARRAY [0..16383] OF Mem_Type;

END MemHC05.
```

♦ ♦ ♦

Figure 2—Instruction Set Definition

```
DEFINITION MODULE ISetHC05;

TYPE
    Opcode_Type = CARDINAL [0..255]; (* valid byte values *)

    Instr_Type = RECORD
        opcode : Opcode_Type; (* numeric value for opcode     *)
        bytes  : CARDINAL;    (* nbr of bytes to increment PC *)
        cycles : CARDINAL     (* nbr of clock cycles to       *)
    END;                      (* execute instruction          *)

    Instr_Rec = RECORD
        instr : Instr_Type;
        proc  : PROCEDURE   (* proc that executes instruction *)
    END;

CONST
    add_imm     : Instr_Type = [ 0ABH, 2, 2 ];
    bit_dir     : Instr_Type = [ 0B5H, 2, 3 ];
    jmp_ext     : Instr_Type = [ 0CCH, 3, 3 ];
    mul_inh     : Instr_Type = [ 042H, 1, 11];
    neg_inh_x   : Instr_Type = [ 050H, 1, 3 ];
    sub_imm     : Instr_Type = [ 0A0H, 2, 2 ];
    tst_dir     : Instr_Type = [ 03DH, 2, 4 ];
    wait_inh    : Instr_Type = [ 08FH, 1, 2 ];

VAR Instr_Arr : ARRAY [0..255] OF Instr_Rec;

END ISetHC05.
```

♦ ♦ ♦

language. I chose Modula-2 to show its usefulness in larger projects. The choice of C as the target language was to pacify those of us who had to learn the language, and now want a reason to use it. *(John wrote a C book for Osborne/McGraw Hill called* TopSpeed C Made Easy, *but he is actually a bigger fan of Modula-2!)*

Meanwhile, I'm building the compiler in two pieces—the front end and the back end.

The front end reads in the source code, checking syntax and generating a pseudo-code version of the program. It also builds the necessary tables for jumps, procedure calls, data areas, etc.

The back end takes the pseudo-code and generates either M68HC05 assembly code or machine language. The machine language is built for one of the '05 memory models, and is then ready to download to the project board. The assembly code allows the inquisitive reader to see what kind of code the C source is generating.

To the user, this is a simple command-line driven compiler. Using your favorite editor, write a simple C program, and then compile it in a way similar to most C compilers. I haven't implemented C's more esoteric features, but if you have a weekend or two....

Modula-2 Wars Revisited (Michael Hunt Again)

Just after returning from SOG, I received Stony Brook's new Modula-2 compiler. JPI's TopSpeed Modula-2 had the overall best execution times in the Issue #47 compiler review. However, Stony Brook's new Professional Modula-2 v2.02 simply blows TopSpeed away (see Figure 7). All this speed and still one unused optimization. I did not use the inline procedure expansion because it would have inflated the code size. Use of the procedure expansion would further reduce execution times but would increase the code size.

The Professional Modula-2 now has the QuickMod development environment. From the same environment, you can use the QuickMod compiler to develop programs and then use the Professional optimizing compiler to produce extremely fast compact production versions. The optimizing compiler has ten different optimizations plus the ability to expand procedures inline.

Until recently, if you wanted the best in optimized code you had to choose assembly or C. If you are a Modula-2 fan, now your code can compete in the performance-oriented PC software market.

Hard Disk TLC

While working on this article, I experienced my first hard disk error. CHKDSK found one lost cluster in one chain. While this is not as serious as a read failure, it did spook me.

I looked at the volume date and realized I had not formatted the hard disk in two years and two days. It had 33 MB of data that had not been backed up in a long time. I subjected myself to the lengthy backup, low-level format, and restore process. It took six hours of stuffing disks, but now the hard disk is backed up, unfragmented, and I have a warm secure feeling.

Do yourself and your hard disk a favor, backup regularly and perform a low-level format every six to twelve months.

Next Time

I've chosen the Professional Modula-2 compiler for the simulator project so I can get a feel for how it performs on larger projects. I'll continue coverage of the simulator and compiler. For the next several issues, I'll present smaller projects while working on the simulator. Happy bit-twiddling to ya.

Jensen & Partners International
1101 San Antonio Road, Suite 301
Mountain View, CA 94043
(800) 543-5202 US
(800) 543-8452 Canada
(415) 967-3200
(415) 967-3288 FAX

Stony Brook Software, Inc.
187 E. Wilbur, Suite 9
Thousand Oaks, CA 91360
(800) 624-7487
(800) 496-5837 CA & International
(805) 496-7429 FAX

♦ ♦ ♦

Figure 3—Instruction Procedure Definitions

```
IMPLEMENTATION MODULE IPrcHC05;

FROM ISetHC05 IMPORT Opcode_Type, Instr_Arr;

  PROCEDURE Exec_Instr (opcode : Opcode_Type);
  BEGIN
      :
    Instr_Arr[opcode].proc (* executes procedure for desired opcode *)
      :
  END Exec_Instr;

  PROCEDURE Add_Imm;         (* Add without Carry *)
  BEGIN
    :
  END Add_Imm;

  PROCEDURE Bit_Dir;         (* Bit Test *)
  BEGIN
    :
  END Bit_Dir;

  PROCEDURE Jmp_Ext;         (* Jump *)
  BEGIN
    :
  END Jmp_Ext;

  PROCEDURE Mul_Inh;         (* Multiply Unsigned *)
  BEGIN
    :
  END Mul_Inh;

  PROCEDURE Neg_Inh_X;       (* Negate *)
  BEGIN
    :
  END Neg_Inh_X;

  PROCEDURE Sub_Imm;         (* Subtract *)
  BEGIN
    :
  END Sub_Imm;

  PROCEDURE Tst_Dir;         (* Test for Negative or Zero *)
  BEGIN
    :
  END Tst_Dir;

  PROCEDURE Wait_Inh;        (* Enable Interrupt, *)
  BEGIN                      (* Stop Processor    *)
    :
  END Wait_Inh;
END IPrcHC05.
```

♦ ♦ ♦

Figure 4—Instruction Set Array Initialization

```
IMPLEMENTATION MODULE InitOCod;

IMPORT ISetHC05;

  PROCEDURE Init_Opcode;
  BEGIN     (* assign instruction information *)
    ISetHC05.Instr_Arr[ 0ABH ].instr := ISetHC05.add_imm;
    ISetHC05.Instr_Arr[ 0B5H ].instr := ISetHC05.bit_dir;
    ISetHC05.Instr_Arr[ 0CCH ].instr := ISetHC05.jmp_ext;
    ISetHC05.Instr_Arr[ 042H ].instr := ISetHC05.mul_inh;
    ISetHC05.Instr_Arr[ 050H ].instr := ISetHC05.neg_inh_x;
    ISetHC05.Instr_Arr[ 0A0H ].instr := ISetHC05.sub_imm;
    ISetHC05.Instr_Arr[ 03DH ].instr := ISetHC05.tst_dir;
    ISetHC05.Instr_Arr[ 08FH ].instr := ISetHC05.wait_inh
  END Init_Opcode;
END InitOCod.
```

♦ ♦ ♦

Figure 5—Instruction Procedure Initialization

```
IMPLEMENTATION MODULE InitIPrc;

FROM ISetHC05 IMPORT Instr_Arr;
IMPORT IPrcHC05;
FROM ErrHC05 IMPORT Invalid_Opcode;

  PROCEDURE Init_Instr_Proc;
  BEGIN      (* assign opcode procedures *)

    Instr_Arr[ 0ABH ].proc := IPrcHC05.Add_Imm;
    Instr_Arr[ 0B5H ].proc := IPrcHC05.Bit_Dir;
    Instr_Arr[ 0CCH ].proc := IPrcHC05.Jmp_Ext;
    Instr_Arr[ 042H ].proc := IPrcHC05.Mul_Inh;
    Instr_Arr[ 050H ].proc := IPrcHC05.Neg_Inh_X;
    Instr_Arr[ 0A0H ].proc := IPrcHC05.Sub_Imm;
    Instr_Arr[ 03DH ].proc := IPrcHC05.Tst_Dir;
    Instr_Arr[ 08FH ].proc := IPrcHC05.Wait_Inh;

    (* Set invalid opcodes to point to error handler procedure *)

    Instr_Arr[ 031H ].proc := Invalid_Opcode;
        :
    Instr_Arr[ 0AFH ].proc := Invalid_Opcode

  END Init_Instr_Proc;

END InitIPrc.
```

♦ ♦ ♦

Figure 6—Error Handler

```
IMPLEMENTATION MODULE ErrHC05;

  PROCEDURE Invalid_Opcode;
  BEGIN
      :
     Display_Err('Invalid Opcode')
      :
  END Invalid_Opcode;

END ErrHC05.
```

♦ ♦ ♦

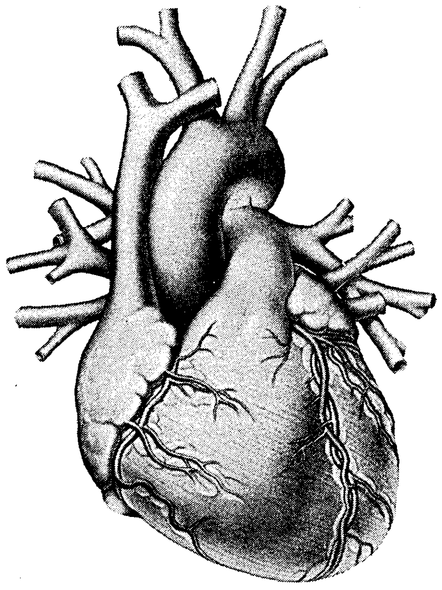
Figure 7—Modula-2 Compiler Comparison

	Execution Time			EXE Size		
	JPI	ProMod	QuickMod	JPI	ProMod	QuickMod
Sieve	39.43	35.65	54.40	1717	929	1042
MemTest	63.49	59.04	65.58	2879	3486	4693
Fileio	77.00*	80.02	80.80	4901	5294	9446
Whetston wo/8087	80.29	27.24	28.01	13759	11052	13641
Dhryston	49.87	39.93	56.25	3821	4552	7674
QuikSort	24.11	15.44	37.84	1895	1078	1250
CQuikSrt	41.74	36.63	48.05	2901	1667	1859
Empty				1611	836	930

* JPI's Mathlib0 uses LONGREAL instead of REAL

♦ ♦ ♦

Good News For Anyone Who Has One Of These.



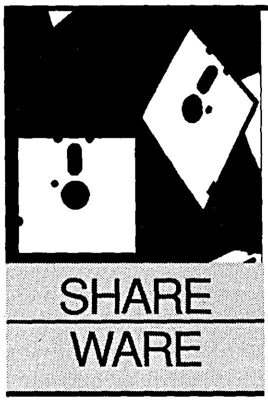
The good news is you can lower your major risk factors for heart disease. And the best news is that Americans have done just that. You've quit smoking, balanced your diet and watched your blood pressure.

In fact, since 1977, death rates from heart attack have gone down by 30.9%. Death rates from stroke have gone down 37.3%. So keep up the good work. And we'll keep on working to support the scientific breakthroughs and medical innovations that help us all make progress against America's number one killer.

To learn about reducing your risk, write the American Heart Association, 7320 Greenville Ave., Box 9, Dallas, TX 75231.

Your Life Is In Your Hands.

American Heart Association



Free Software

By Larry Fogg
Micro C Staff

Boy, you can't make book on keeping a good columnist. We just got Tony broken in and some silly book publisher asks him to write a book. Fortunately, Larry leapt into the breach.

With Tony Barcellos gone to the Land Of Many Pages (book publishing, that is), we find our stable of writers a bit short. (Listen. You can hear the writers now: snorting and pawing at the snow to uncover the dictionaries and thesauri. We really should bring them indoors during the winter, but they're incredibly messy and tend to frighten visitors.)

We're sorry to see Tony go. His work on the Shareware column has been an editor's dream: always in close to deadline, always lucid and interesting, and always written in English. But an able replacement waits in the wings. Next issue Nelson Ford, famed in Shareware circles for his Public (Software) Library, tries on this column.

For now, I'll try to fill Tony's shoes by covering three programs that caught my eye recently. This will be a bit of a departure from the normal Shareware column. I'll talk about free software: a trio of programs written purely for the fun of it and given away. Ya gotta admit, free is a very good price. And, in these cases, you get more than your money's worth.

Ghostscript

Last issue, Tony mentioned that we might see an increasing number of PostScript utilities from folks other than Adobe (PostScript's creator). One of these utilities is Ghostscript, from Aladdin Enterprises.

Ghostscript interprets the Ghostscript language, a very close clone of PostScript. Then it uses its library of EGA graphics functions to display the results. Yep, EGA required.

For learning the fundamentals of PostScript programming, I highly recommend GhostScript. In its interactive mode, you enter commands and immediately see the results on the screen. Or you can feed Ghostscript a file on the command line. When your code heads off to "where no program has gone before," Ghostscript prints a reasonable error message and displays the contents of the stack to aid in the post mortem.

Ghostscript doesn't implement the showpage operator (i.e., it can't print a hard copy), but who cares. You'll use Ghostscript as a learning tool and for debugging. It's simple enough to send a file off to the printer once it's ready for prime time. A few other operators didn't make it into Ghostscript, and it has some limitations when dealing with large numbers. But I tested the interpreter with several PostScript files I had kicking around and didn't run into any major problems with its limitations.

One complaint. On my system, Ghostscript doesn't exit very gracefully. I'm left in a curious text mode with no cursor. A quick "MODE CO80" cleans up the screen. I could change it if I weren't so lazy. That's the bad part of getting source code with a program; you can't bitch about anything. Especially if you pay $0 for it.

Documentation: Terse. Adequate.

Ghostscript's license covers only two concerns. First, access to the program and its source code must not be restricted. In a nutshell, no one should try to stop anyone from giving away copies.

Second, be sure to take credit for any changes you make to Ghostscript. The authors don't want to become infamous for any oogies you introduce. Reasonable requests, both.

LHarc

The recent ARC wars left a handful of very strong survivors. Among those, LHarc stands out for its highly efficient compression method, a very tight self-extraction capability, and low price (excuse me, no price).

Developed in Japan with a strong emphasis on compression, LHarc succeeds well. This is the package to use if your major concern is file size.

The ability to create self-extracting archives makes LHarc very attractive for bulletin boards, Shareware authors, and anyone else distributing files to an unknown audience. No worries; users don't need to have a compatible extraction utility to get at the archive contents. They just execute the archive and it takes care of the rest.

Here, LHarc shines. Its self-extraction function adds only 1263 to 1295 bytes—much less than PKZIP's 15,512 bytes. This means that bulletin boards can afford to make more files self-extracting without paying a huge price in extra disk space and download time.

Largely because of this slick ability, we've switched from ARC to LHarc on the Micro C BBS. Look for it under the name LH113C.EXE (a self-extracting archive, of course).

Haruyasu Yoshizaki (the author) doesn't make any claims for fast execution, and LHarc *is* considerably slower than the rest of the pack. But the lack of blinding speed didn't trouble me; for my purposes, it's fast enough.

See Figure 1 for a comparison of LHarc, PKZIP from PKware, and ARC from System Enhancement Associates. For this test, I archived a 216,722 byte mix of 18 text and executable files.

Figure 1—Archive Utility Comparison

Program	Archive Size	Creation time	Extraction Time
ARC v. 6.02	150143	0:46	0:33
PKZIP v. 1.01	123317	0:59	0:21
LHarc v. 1.13c	122230	1:43	1:02

♦ ♦ ♦

A good deal of effort has gone into naming the different file compression algorithms. We used to get by with squeeze and unsqueeze. Now we crunch and uncrunch, implode and explode, and throw in the occasional squeeze for old time's sake. LHarc freezes and melts its archive elements; keep a towel handy.

Adherence to a "standard" always comes as a shock in the computer world. You'll feel right at home with LHarc if you've ever used ARC or PKZIP; all three programs use similar command line syntax.

Complete documentation describes the program and includes a distribution policy very similar to Ghostscript's. These folks just want to keep their efforts free-ranging through the computer cosmos. Nice.

FRACTINT

In an absolute return to the public domain spirit of computing's early years, a group of programmers has created the best fractal program I've seen, bar none.

FRACTINT does everything imaginable in the way of producing mesmerizing images on your screen. It computes a wide range of fractal types: Mandelbrot set, Julia sets, Newton domains of attraction, Lambda sets, and my favorite—plasma clouds.

Plasma clouds have nothing to do with fractals, but they're at least as much fun to watch. They use a recursive technique to divide the screen into rectangles. Each corner of a rectangle gets a random color assignment, with a sort of averaging technique filling in the rectangles to achieve a cloud-like effect. For pure viewing pleasure, an animated plasma cloud surpasses even the venerable lava lamp.

FRACTINT even does things only *you* can imagine by providing a stub called testpt(). Write your own fractal-generating innards for this function, recompile, and test the function (with full access to FRACTINT's support for mouse, 3-D, animation, etc.).

Did I say recompile? You bet. FRACTINT comes with full source code written in Microsoft C 5.1. Some modules were coded with MASM 5.1 for speed, but don't despair if you don't own this particular assembler; OBJ files come with the source code so all you have to do is link 'em in.

Just for fun I recompiled in Turbo C. I had to make only minor repairs, most of them changing the C++ commenting convention ("//") to standard C ("/*"). In just a few minutes the program was up and running again.

FRACTINT doesn't autodetect the graphics adapter on your system; you select from a slew of possibilities. Very nice if you have a VGA but would like to explore in a lower resolution mode. The only common adapter left off the list is Hercules, and it wouldn't be much of an effort to add that capability.

Fractal aficionados will find all the features you've come to expect from fractal software: mouse support, screen save/restore, animation (not true animation, but varying the color map gives a nice animation-like effect), 3-D, printer dump (LaserJet and Epson compatibles), and much more.

If you can take advantage of the 386-specific fixed point routines, fractals will fly across your screen. If not, emulation of these routines lets the program run (more slowly) on anything down to the 8088. In default mode, FRACTINT can make some good guesses in areas of solid color. This speeds up the program even more, with very little loss of detail.

Another interesting speed-up technique makes use of the periodic behavior of points in the middle of the Mandelbrot set. While iterating its little heart out, if FRACTINT recognizes a repeating series of values it can bail out of the loop early and save a lot of wasted effort. Smart program.

The friendly, amusing, and complete documentation begins with the banner, "An experiment in 'Stone Soup' development. Got anything for the pot?" It ends the distribution policy. The reader is encouraged to use, modify, and distribute the program. And finally, the bottom line. The plug for contributions: "Don't want money. Got money. Want admiration."

Bert Tyler, Timothy Wegner, and a host of others have created a work of art. They certainly have *my* admiration.

Programs Mentioned

All three programs are alive and well on the Micro C BBS (503-382-7643) and on our Issue #52 Listings Disk.

Ghostscript vers. 1.2
Distributed by:
The Free Software Foundation, Inc.
675 Mass Ave.
Cambridge, MA 02139
(617) 876-3296

LHarc vers. 1.13c
Contact Yoshi through:
Kenjirou Okubo
CompuServe 74100,2565 or
GEnie via K.OKUBO

FRACTINT vers. 9.0
Authors on CompuServe,
PICS S 16 forum

♦ ♦ ♦

011
100
100
TIDBITS

Finding Text Fast

By Gary Entsminger
P.O. Box 2091
Davis, CA 95617

When I dug into this column, I quickly found lots of text on quickly finding lots of text. My I/O may not be as fast as a 386SX, but then I'm not sure how to expand my buffers.

Finding a fast way to search a string for a pattern (or substring) is a fundamental computing task, well-researched and whittled. After you've bought a faster computer, a faster hard disk, and a faster language, it hinges on two tracks: (1) an algorithm; (2) I/O implementation. Cleverness and common sense, one might say.

Recently, while developing a text retrieval program I call *Wizard,* I needed to optimize a pattern search. I learned a couple of simple ploys which you might consider.

As seems often the case, I was knee-deep in an I/O bound situation. I wanted to search a database of text files (stored on a hard disk, CD ROM, Bernoulli Box, etc.), find files that contained specific patterns, and act on the results. My opening algorithm was simple: *Read; Compare; Act.*

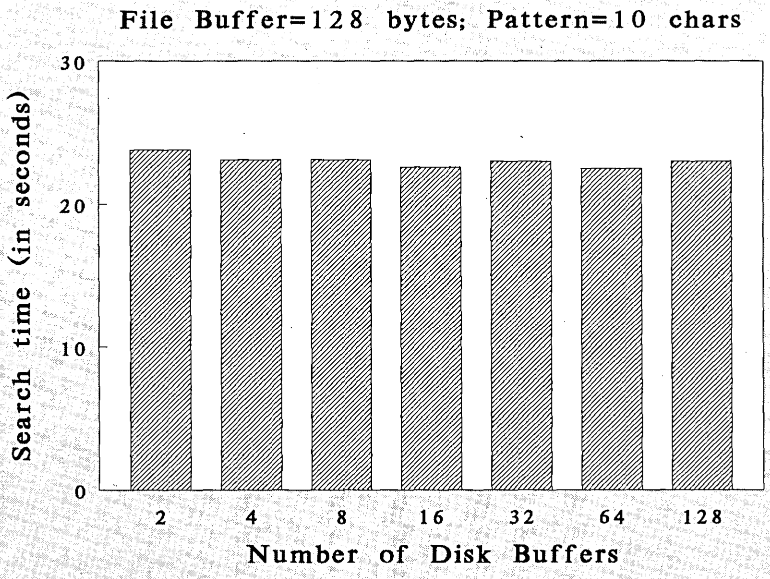
Via Buffers

When we read a file, employing say Readln in Turbo Pascal, we use DOS to read data a chunk at a time into—(1) first a disk buffer; (2) then a program buffer where Readln then reads it a line at a time.

The DOS disk buffer and the Turbo Pascal file buffer are, by default, transparent. DOS 3.3 and 4.0 (the versions I'm using) set the default number of disk buffers to correspond to system configurations. For example, on a 640K XT or AT using a hard disk, DOS 3.3 and DOS 4.0 set 15 buffers. DOS 3.1 (running on 640K XTs and ATs) sets 2 and 3 buffers, respectively.

In the default setup, each time a program calls Readln, it waits for DOS to look first in its disk buffers for the data. If the data isn't there, it waits while DOS reads 512 bytes times the number of buffers. Finally it waits for 128 bytes to transfer from a disk buffer into the program's file buffer.

Figure 1—Effects of Disk Buffers on Search Speed



If DOS finds the data in a buffer, it doesn't have to read the disk, which saves time. Increasing the number of buffers increases the amount of workspace DOS has for data. In theory, the more data already in memory, the less often DOS has to access the disk. Hence—the faster the I/O.

The DOS disk buffer and Turbo Pascal file buffer defaults are intentionally set low. This is to save memory and because (as I'll show) different situations benefit from different configurations.

Bigger isn't necessarily faster or more efficient. DOS can, for example, waste time looking through buffers when it should go straight for the disk.

Each DOS disk buffer holds 512 bytes, and you can allocate between 1 and 99 disk buffers in DOS versions before 3.3. You can allocate between 2 and 255 in versions 3.3 and 4.0. (However, I've only been able to allocate more than 99 buffers when I've had expanded memory.)

Set the number of 512 byte disk buffers by using—

```
Buffers = <Number>
```

in CONFIG.SYS. Use the /x command after "Number" to put the buffers in expanded memory.

Increasing the number of DOS disk buffers improves I/O speed a little. But frankly, I'm not impressed. Figure 1 shows a graph of one profile of the effects of buffer number on search speed.

(Note: Each benchmark test assumes the worst case—the pattern we're looking for doesn't exist in any of the files to be searched. This forces the program to search every line of every file. *Wizard* abandons a search and moves on to the next file as soon as it finds a match, so successful searches are often considerably faster.)

Buffers Within Programs

You'll improve a program's performance more by fiddling with its file buffers than you will by allocating more disk buffers. And you'll have more control over your program once it's out of your hands, since you won't have to count on modifying your users' CONFIG.SYS files.

We control file buffer size in Turbo Pascal with the built-in function—SetTextBuf. C and C++ers use setvbuf. The rest of you, check your manuals.

We simply set up a buffer (I'll use an array of chars here) and tell our program we want to use that buffer on a particular file—

```
var
  F   : Text;
  Buf : array[0..8191] of char;
begin
  assign(F,FileName);
  Reset(F);
  SetTextBuf(F,Buf);
  ...
end;
```

To test various buffers, I ran identical searches on a 1 megabyte database of files. I doubled the buffer size each time trying buffers of—128 bytes, 256 bytes, 512 bytes, 1K, 2K, 4K, 8K, 16K, 32K, and 64K. See Figure 2 for the results.

Each search uses a so-called brute force algorithm that yields roughly linear results. The files vary in size from 10K-120K. I used a 32 ms drive (no disk caching or other tricks) and a 16 MHz 386SX. For an approximate 8 MHz AT comparison, double the times.

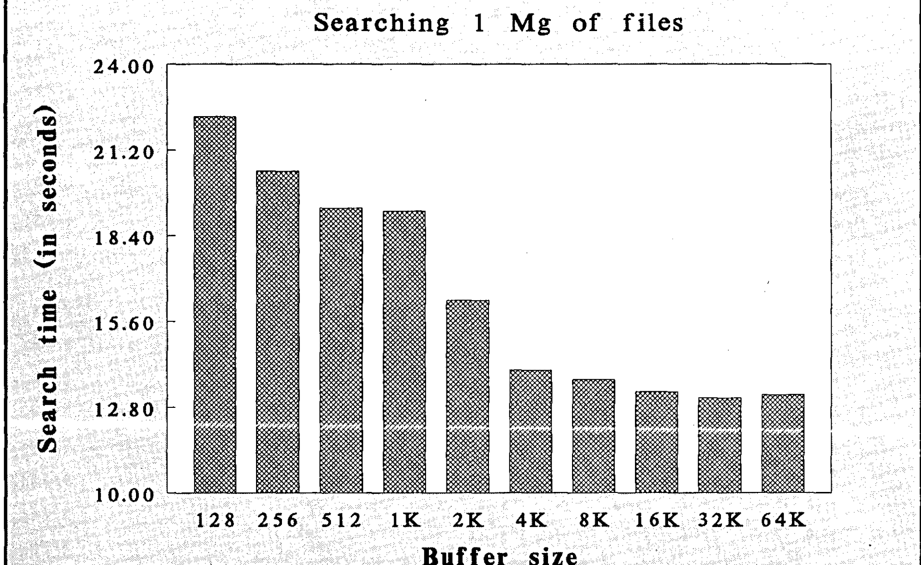
The graph indicates a very useful trend—we can increase the program file buffer size a little and increase speed considerably. And there's beauty in the compromise—we don't pay much (a 4K buffer almost doubles the speed). Note also that a huge (64K) buffer is only a hair better than 4K or 8K.

Next, I wondered—what happens when a file buffer is between 2K and 4K? Between 4K and 8K? What if a buffer is a hair smaller or larger? I expected this one to be a wild goose chase, but my recent years' work on chaos theory and thresholds in neural networks makes me hesitant to jump to conclusions.

Figure 3 shows the results of this experiment—an 8K buffer minus 1 byte and a 4K buffer minus 1 byte.

Subtracting a byte really helps. Adding a byte, by the way, doesn't. Explore a few more of these in betweens and let me know what works in your applications, on your systems.

Figure 2—Effects of Buffer Size on Speed



Algorithms

Let's move on to the algorithm. The previous one (used in Figures 3 and 4) does nothing fancier than begin at the beginning of the string and move through character by character looking for a match. Brute-force, general purpose, and faster on average than you might expect.

I snooped around a little and found that lots of folks have played with and written about search algorithms: Wirth, Knuth, Robinson, Dromey, etc.

I tried several of the algorithms they talk about; the best alternatives to the brute-force use some variation of a lookup table. The algorithms try to uncover ways of eliminating some characters in the target string from the search. They then build a lookup table for each search pattern which tells the program how to handle the search depending on the current character.

For example, if the current character is an "e," we might be able to skip four characters. Generally, the program uses the table to find out how many characters to skip over. This saves, in best cases, lots of time. It costs, in worst cases, a little time.

The results in Figure 4 show how search speed varies between a very good table lookup system (one using the Boyer-Moore algorithm) and the brute-force system I used earlier.

When a search string or the volume of data is small, brute-force usually searches faster. Generally, the longer the string, the faster the search using either of the algorithms. Very long strings lead to particularly quick searches using a table-lookup.

Tables inherently have two drawbacks—(1) the time it takes to build them, and (2) the time it takes to use them.

If the search pattern is short enough, it's often quicker to look for it directly. Too much time is lost building the table and then looking up characters.

Usually though, indirect systems such as a table lookup can speed up I/O. An intelligent program might look at a search pattern and available memory then *quickly* determine the optimal algorithm to use.

Objects

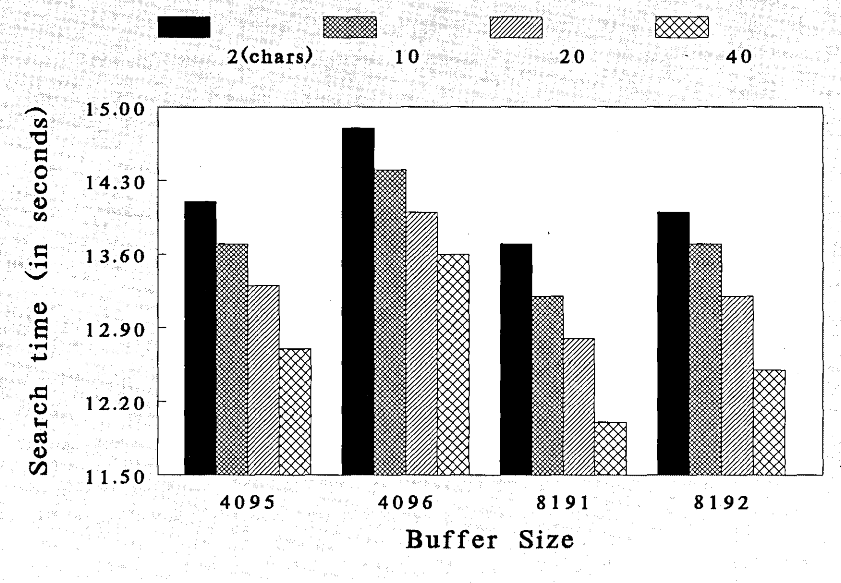
Just when I thought I'd figured enough out to finish my program (and Tidbits), I noticed that although I was using only a little memory for a buffer, I was using too much.

I was declaring a global array which Turbo Pascal was allocating in the data segment (where all global variables are allocated). Problem #1—after *Wizard* completes its search, it builds a list of matching files and then pops file selections into an editor. My editor uses the data segment for its buffer, and it didn't want to share it.

Global variables stored in the data segment weren't going to work well enough in my situation. Local variables (those of procedures and functions) are stored on the stack. So I moved my buffer into the procedure that actually handles the search.

This is a good solution if there's room on the stack, and if other procedures don't need to know about the buffers. For reasons I won't go into, I needed to preserve the buffer between procedures and sparingly use the stack. I needed a different solution.

Figure 3—Effects of Slight Changes in the File Buffer

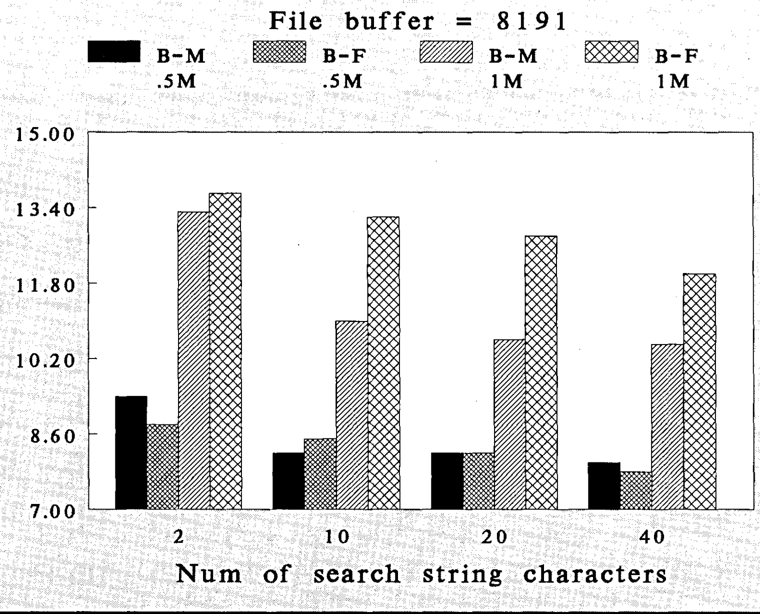


I considered passing a pointer to the buffer, but a more general solution came to mind. I incorporated all the procedures and functions that needed to share the buffer in an *object*. I then set up a pointer to the object (making it dynamic) and allocated space for it (the object) on the heap. This preserves the data segment, frees the stack for other matters (recursion for example), lets the pertinent procedures know about the buffer, and in general simplifies the program.

The object looked something like:

```
type

LookPtr = ^Lookup;
Lookup = object(List)
  MyList:array[1..255] of StrList;
  Flag        : boolean;
  Listsize    : byte;
  Array_index : byte;
  FileName    : string;
  Patterns    : string;
  constructor init;
```

Figure 4—Effects of Algorithms on Search Speed



```
function GetFileName : string;
procedure match_lines;
procedure use_file_list;
destructor done; virtual;
end;      {Lookup}
```

This again reinforces my feelings about the Pascal (and C) extensions to OOP. You use them when you need to without rethinking the entire program. Making the few adjustments in *Wizard* that objects required took a few hours at most.

Useful Tools

I'd like to point you toward a few tools that help my projects develop faster and more smoothly. Three are Turbo Pascal programming toolkits and one is a graphics program.

The Turbo Pascal tools—

Object Professional and Turbo Professional (from Turbo Power near Santa Cruz) are thorough, professional toolkits. They include procedures and functions for building TSRs, data entry routines, interrupt services, menus, windows, and much more.

Object Professional offers object-oriented libraries; Turbo Professional is a non-OOP version of most of Object Professional. I highly recommend both. They're complete, well-documented, professional! Good deals at $150 (for Object) and $99 (for Turbo). Source is included.

Turbo Toolkit from Techno Jock down in Houston includes source code and libraries for fast screen writing, keyboard input, menu and list building, string and I/O handling, and oodles more. At $49.95, it's a steal; the easiest to use Turbo Pascal toolkit I've used, good for novices and professionals alike. I've learned a lot about Turbo Pascal programming just by using the kits.

The third tool is a program (not a programming toolkit), SlideWrite Plus from Advanced Graphics in Berkeley. I've used SlideWrite Plus intermittently for a year and a half now to create graphs, tables, text, and combinations of the three.

It's a powerful general purpose graphics program and especially good for creating custom high-quality graphs. You can import data from statistical packages and other graphics programs. Edit these images, create new graphs, print to a wide variety of printers, and create postscript, .TIF, or .PCX files for export to Ventura, Word Perfect, PageMaker, etc. Recommended.

And that, as the sun fades into this hazy, gasoline-polluted day (air-quality: marginal), is Tidbits.

References

Dromey, R.G.; *How To Solve It By Computer*; 1982; Prentice-Hall.

Jamsa, Kris; *Using DOS 4*; 1989; Osborne McGraw-Hill.

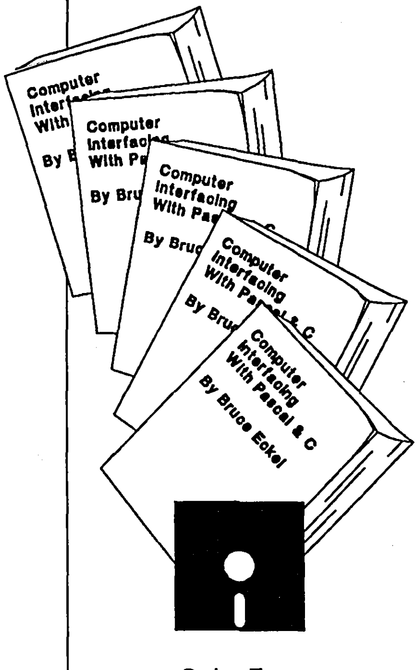
For more information:

Advanced Graphics Software
333 W. Maude Ave., Suite 105
Sunnyvale, CA 94086
(408) 749-8620

TechnoJock Software
P.O. Box 820927
Houston, TX 77282-0927
(800) 242-4775

Turbo Power Software
P.O. Box 66747
Scotts Valley, CA 95066-0747
(800) 333-4160

♦ ♦ ♦



Order From:
Micro Cornucopia
PO Box 223
Bend, OR 97709
1-800-888-8087

Now Available From Micro C...

Computer Interfacing with Pascal & C by Bruce Eckel

- Use your PC parallel port for digital input and output
- Build an Adapter Card for your PC
- Design and build electronic circuits

"With wit and superb technical figures, Bruce captures the essence of making electrons out of bits and vice versa."
Jeff Dunteman, *Dr. Dobbs*

Only $30 ppd.
Add $10 for foreign orders
Includes Book & Disk

NOW ON NEW PLATFORMS

VAX/VMS
HP 9000/300
XENIX/386
Motorola UNIX V.3
Sun-3
NCR Tower

OREGON C++
- A true compiler – not a translator
- Three compilers in one: C++, ANSI C or K&R C
- Source-level debugger
- Conforms to AT&T V2.0 C++

C++ SOFTWARE ENGINEERING SEMINARS

Motorola UNIX V.3
VAX/VMS & ULTRIX
Sun-3, 386i
XENIX/386

PASCAL-2
- Full ISO Level 1 standard
- Comprehensive error checking
- Access to command-line arguments
- Extensions include full string type, separate compilations, external variables & more

XENIX/386
Sun-3
VAX/VMS

OREGON MODULA-2
- Conforms to emerging ISO standard
- Excellent support of coroutine and interrupts
- Ideal for embedded systems

680X0 CROSS-COMPILERS ALSO AVAILABLE

CALL 1-800-874-8501
(503) 245-2202 FAX: (503) 245-8449

OREGON SOFTWARE

6915 SW Macadam, Suite 200, Portland, OR 97219 USA

The following are trademarks: Oregon Software, Oregon C++, Oregon Modula-2, Pascal-2, Oregon Software, Inc., Xenix, Sun-3, 386i, VAX, VMS, Motorola, UNIX, AT&T, ULTRIX, NCR Tower, HP 9000

Reader Service Number 193

TECHTIPS

AT Keyboard Speedup

AT Miscellany

For a keyboard speedup program that doesn't take up *any* RAM, try KBSPD (see Figure 1). It works only on an IBM AT (or clone) keyboard, since it resets some parameters in the keyboard's micro, but it sure runs slick. It's written in Lattice C vers. 3.10.

Oh, another interesting thing I found out about AT-class toys. If you have an old one whose hard drive tables do not support your drive fully (say, the original IBM AT), you can take your hard drive and controller card to a newer machine that does. Format the drive there (low level, FDISK, and FORMAT) and return the pair to your original machine.

Next, run SETUP and tell it your hard drive is whatever type has heads <= # of actual heads and cyls <= actual # of cyls. When you boot, it runs. It even uses all of the disk the other machine could get to. Only catch is, it goofs up what CORETS27 thinks about your drive's performance specs.

I suspect you could set these to whatever you wanted by changing the hard disk parameter tables at boot-up, between when it reads the partition table and when it starts loading up the bootable partition.

Oh, by the way, PC-DOS leaves about an 8K hole at the start of your disk. (MS-DOS may not do this—Compaq doesn't.) You can use this to hide things, or to add pre-boot tweaks like this and the DRAM refresh-speed changer. (See last issue's Techtips.) I have code to do this, and if there's an interest I'll kludge together an article on it.

John Welch
1310 Kenneth Circle
Elgin, IL 60120

Figure 1—AT Keyboard Speedup

```
/***********************************************************************
                              KBSPD
            a utility that speeds up AT clone keyboards
                           by John Welch
          Note: this will NOT work for XT & PC keyboards.
***********************************************************************/

#include <stdio.h>
#include <dos.h>

struct REPRATE {
    char *rate;
    char data;
};

struct DELAY {
    char *input;
    char data;
};

struct REPRATE reprate[] = {
    { "30",0x00 },
    { "25",0x02 },
    { "20",0x04 },
    { "15",0x08 },
    { "10",0x0c },
    { "5",0x14 },
    { "3.3",0x19 },
    { "2",0x1f },
    { "0",0x00 }
};

struct DELAY delay[] = {
    { "250",0x00 },
    { "500",0x40 },
    { "750", 0x20 },
    { "1000",0x60 },
    { "0",0x00 }
};


main(argc,argv)
int argc;
char **argv;
{
    int i, j;
    unsigned char c;

    if (argc != 3)
    {
HELP:
        puts("KBSPD  sets AT keyboard speeds\n");
        puts("Usage: kbspd reprate delay");
```

Continued on page 92

Micro Ads

A Micro Ad is the inexpensive way to reach over 22,000 technical folks like yourself. To place a Micro Ad, just print out your message (make it short and sweet) and mail it to Micro C. We'll typeset your ad (no charge) and run it in the next available issue. You can also send camera ready copy. Rates: $99 for 1 time, $267 for three times, $474 for 6 times (a best buy at only $79 per insertion). Full payment must accompany ad. Each ad space is 2 1/4 inches by 1 3/4 inches.

VIRTUAL MEMORY for C DATA

BREAK out of 640K!

Virtual memory manager for data-intensive MS-DOS C programs. VMEM/C contains C library functions which automatically bring needed data into memory and offload it back to disk. Often-used data stays in memory. 8 or 9K of runtime. No royalties. No risk.

DOS 2.0+, Microsoft C4.0+, Turbo C 1.5+ 30-day return, 6-mo. free updates. (Now includes separate expanded memory library.)

$180 (with portable source)

Arizona Softool Association
5245 E. Larkspur , Scottsdale, AZ 85254
(602) 996-3110

Reader Service Number 199

WISDOM OF THE AGES

First electronic book of quotes, sayings & ideas brings over 1,000 of the world's greatest minds to PC screens & printers.

Select a subject. Use the vivid parade of timeless knowledge to act, write & speak better; earn more.

Subjects are organized into sections to give you the most benefit in the least time. Uses Filters to customize output; Dynamic mode to stimulate creativity.

It's never too early to get a head start.
Introductory Offer: $79.00 for all 5 disks.
30 day money-back guarantee.

Requires 256K, DOS 2.0 or later & 2 floppies or hard disk. Add $2.00 S/H. CA residents add 7%. Outside U.S. add $10.

MCR Agency, Inc.
6116 Merced Ave. #81MC
Oakland, CA 94611
1-800-767-6797
FAX 415-444-6561

Reader Service Number 181

Why you want BATCOM!

BATCOM is a batch file compiler that compiles your *.bat* files to *.exe* files to make them faster, more professional, and more capable. **BATCOM** extends DOS with new commands so you can read keyboard input, perform arithmetic, use subroutines, and much more. In addition, **BATCOM** protects your source code, and you can distribute your compiled programs without royalties. For IBM PC. Only **$59.95.** Order today!

Wenham Software Company
5 Burley St.
Wenham, Ma. 01984
(508) 774-7036

Reader Service Number 124

Technology for the Arts

Willow VGA-TV NTSC Output $599
Publishers' VGA/Frame Grabber $599
Panasonic CCTV B&W Camera $225
Specials: Sanyo Color Camera $395
Color Computer Eyes Digitizer $350

MIDI (music) Interface $119
MIDI Programmers Toolkit $39
Free MIDI or Graphics Flyers

joel Sampson Engineering
P.O. Box 550363 (214) 328-2730
Dallas, TX 75355 BBS 328-6909

Reader Service Number 176

Experience Embedded Power

with the new Control-R II 8031 µController Module

- √ 5 volt single supply operation (even with RS232)
- √ Data/Address/Control Bus Headers for expansion
- √ Direct Access to 8031 Ports 1 and 3
- √ 8K on board EPROM space
- √ Socket for 8K of on-board Static RAM (6264)
- √ MAX232 socket to provide RS232 compatible port
- √ Memory decoded to allow use of 8052-BASIC µP
- √ Assembled and Tested, *not a kit*
- √ Compact size: 3.5" x 4.5"

Control-R II, Enhanced 8031 Module (SRAM & MAX232 optional)	64.95
Control-R II, Bare PC Board w/docs	25.00
Control-R I, 8031 Module (Port 1 & 3 access and MAX232 option only)	39.95
Control-R I, Bare PC Board w/docs	14.00
8K Static RAM I.C.	10.00
MAX232 for serial I/O	6.95
PseudoSam 51, 8031 family cross-assembler for MSDOS computers (5.25 inch disk w/docs)	50.00

Shipping $3.00 US / $5.00 Canada
Illinois Residents Add 6.25% Sales Tax

Cottage Resources Corporation
Suite 3-672C, 1405 Stevenson Drive
Springfield, Illinois 62703
(217) 529-7679

Reader Service Number 158

COMPUTER BOOKS

up to 40 % OFF LIST

Send for FREE information
COMPUTER BOOKS

P.O. 70746
Sunnyvale, CA. 94086-0746

Reader Service Number 167

STOCKS OPTIONS FUTURES

Turn Your PC Into A
MARKET QUOTATION MONITOR

100 page book covers satellite and radio data reception of financial news and quotes for your PC. $19 (includes demo diskette). Free informative catalog of:

- Data receivers and kits
- Quote processing and display software
- Descrambling software utilities

303-223-2120 $5 Demo Diskette
DATArx
111 E. Drake Rd. Suite 7041
Fort Collins, CO 80525

Reader Service Number 133

8748 EMULATOR

Simulate your 8748 programs in software before burning the EPROM. See all internal registers and i/o pins. Script files can simulate complex external events. Includes source.

Other Products

DOS Source
Fully commented 1.1, 2.1 . . . . $15/$45
Hercules Tools
Graphics stuff w/source $15
Math Library
ascii Floating pt. w/source $15

Information Modes
P.O. Drawer F
Denton, TX 76202
817-387-3339 **1-800-628-7992**

Reader Service Number 149

WHITNEY EDITOR $49

Small and fast
Uses all available memory
Split-screen editing
Configurable keyboard
Regular expression search
One key compile
Features for writing documentation
Condensed/Outline display
Runs on IBM PC's, AT's, and PS/2's

USA shipping & handling $3; Outside USA $15
CA residents add sales tax

Whitney Software, Inc.
P.O.Box 4999, Walnut Creek, CA 94596 (415) 933-9019

Reader Service Number 164

DSP32C COPROCESSOR BOARD

- ➤ 25 MFLOP Floating Point DSP
- ➤ High speed NUMERICS and GRAPHICS
- ➤ 32 BIT on board data bus
- ➤ 16 bit PC/AT interface
- ➤ ALL ON BOARD MEMORY DUAL PORTED
- ➤ Parallel and codec serial IO
- ➤ 15 ms 1024 point FFT
- ➤ Assembler, monitor, and libraries

Base board and ALL software $950
640K 85ns memory $300

SYMMETRIC RESEARCH
15 Central Way, Suite #9, Kirkland, WA 98033
(206) 828-6560

Reader Service Number 182

EGA FRACTAL MASTER

Whether fractal novice or fractal hacker, this easy to use yet powerful package lets you explore:

The Mandlebrot set **Julia sets**
Dragonland **Self-squared dragons**
Biomorphs **Newton's method**
Peano curves **Fractal landscapes**

Interrupt, save and resume plots at any time. Easily change colors, set boundaries, animate colors, create mirror images. Supports fast integer computations (even WITHOUT an 80386) and 80x87 math co-processors. Includes a slideshow program. Specify 5 1/4 or 3 1/2 disk.

Only $25 (postage included)
Paul W. Carlson
602 North Avenue, #23
Burlington, Vermont 05401

Reader Service Number 185

Continued from page 90

```
        puts(" where reprate is the repeat rate (2, 3.3, 5, 10, 15,\
               20, 25 or 30) in cps");
        puts(" and delay is the initial delay in ms (250, 500, 750 or\
               1000)");
        exit(-1);
    }

    i = j = 0; /* start at the beginning of the arrays */

    while (reprate[i].rate[0] != '0')/*look for match on repeat rate*/
        if (strcmp(reprate[i].rate,argv[1]) == 0)
            break;
        else
            i++;
    if (reprate[i].rate[0] == '0') /* if no match, exit giving help */
        goto HELP;

    while (delay[j].input[0] != '0')   /* look for a match on delay */
        if (strcmp(delay[j].input,argv[2]) == 0)
            break;
        else
            j++;
    if (delay[j].input[0] == '0')  /* if no match, exit giving help */
        goto HELP;

    printf("Setting keyboard to %s cps with initial delay of %s...",
            reprate[i].rate,delay[j].input);

    c = reprate[i].data | delay[j].data; /* mask the delay and the
                                                    rep rate together */

    outp(0x60,0xf3); /* wake up keyboard */
    /* note: this loop could be infinite if your keyboard is fubar. */
    while (inp(0x60) != 0xfa);  /* wait for keyboard to acknowledge */

    outp(0x60,c); /* send delay & repeat rate to keyboard */
    /* note: this loop could be infinite if your keyboard is fubar. */
    while (inp(0x60) != 0xfa);  /* wait for keyboard to acknowledge */
    puts(" Done.");
    exit(0);
}
```

♦ ♦ ♦

Hercules Video Mode Determination

I was recently converting some of our CGA graphics code to work with the Hercules Video board. The code needed to recognize the Herc board, then we needed to be able to tell what mode (text or graphics) the board was in. The code resided in a TSR program and had no control of what happened to the video just prior to its call.

Though I searched long and hard, I could find no way to determine the board's mode. The only references I found stated that "the current mode state cannot be determined."

To start off, we wrote a small routine that would let us put the board into either mode. Then we added code to monitor everything we could monitor. (We were working without a real technical manual, using just the short manuals supplied with the cards. Plus we have only two Hercules systems—both are clones, both are different.)

It was quickly obvious the solution wasn't going to be simple. For instance, the IBM monochrome board looks the same as a Herc board for all practical purposes. However, unlike the IBM board, most Herc type boards supply a vertical sync pulse and a horizontal sync pulse in status port bits. This solves that problem.

Reasoning that the vertical sync and horizontal sync pulses *may* be different from mode to mode, we attacked the status ports. The H-sync would be too fast to measure easily so we used the vertical sync. Guess what. There's a slight difference in the vertical sync rate in graphics mode versus text mode. Due to the small difference and the PC's 18.2 tick/second clock, we had to monitor for at least 5 seconds to see a difference.

Fortunately, it was easy to increase the PC clock to 1820 ticks per second. We reset timer 0 and installed new INT 08 and INT 1C code. The 1C timer just does IRET to eliminate any current timer take over. The 08 timer services the original 08 timer at its normal times even while we're doing our tracking.

After the timer's been changed, we call a loop that starts the clock and watches the 03BAh port for vertical sync pulses. Once the time's up, we save the count and reinstall the original timer parameters and original interrupts.

This solved the problem! We can easily tell the mode with a test run of 112 ticks (0.06 seconds). The sync count is now very steady and correct at 50 pulses per second.

Of course we're not certain that the vertical sync rate changes on all hercules boards, so we'd like to hear from anyone who has made this test.

The MASM vers. 5.0 source code on the Micro C BBS (503-382-7643) and the Issue #52 listings disk includes: TSTVSYNC.ASM—a general testing program that allows repeated testing with parameter line input for the test time (in ticks); and HERCTEST.ASM—the final stripped code built into a DEMO program to indicate how it is used. We've also included executable versions.

CAUTION! We did *not* take the time to build in precautions. Do not Ctrl-C or Ctrl-Break out of these programs; your machine may be left in a weird state. If the programs do not return to DOS normally, be sure to reset your machine before using it further.

The systems we tested included:

- ACS-XT 8086/8088 machine, MS-DOS 3.2, Dual speed 4.7 & 10 MHz. Video is Taiwanese Herc clone.
- PCL-AT-8 80286 machine, PC-DOS 3.2, Dual speed 6 & 8 MHz with 80287. Video is CGA and Herc compatible clone.

Thank you ahead of time for any testing or inputs you give us. If there are questions or problems, please call or write or leave a message on the Micro C BBS (we do not get on very often, though, so please be patient with us).

Joe Blanford
ReBec Custom Software
2003 Saddlewood Trail
Dothan, AL 36301
(205) 792-1039

♦ ♦ ♦

more Micro Ads...

Are You Developing Software for Motion Control or CAD/CAM?
Write robust programs in hours with the...
QuickGeometry Library
- vectors, points, lines, arcs, ellipses and NURB splines
- rotate, scale, translate, mirror, intersect, offset, trim, interpolate

Includes C source code and a RISK-FREE 30-day money-back guarantee!
CALL 617-628-5217 TODAY FOR INFORMATION OR TO ORDER!
Building Block Software P.O. Box 1373 Somerville, MA 02144

Reader Service Number 196

Since 1986, companies worldwide have been choosing Az-Tech security products. If you demand the strongest protection available, why not chose one of these "proven leaders":
- **EVERLOCK** Copy Protection
- **EVERTRAK** Software Protection
- **EVERKEY** Hardware "Key" Software Security

For IBM and Compatibles. 30 day money back guarantee.
Free info and demo disk available.

Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085

(800) 227-0644
(816) 776-2700 Fax: (816) 776-8398

Reader Service Number 197

XenoCopy-PC $79.95+S/H
PC-DOS program lets your PC
Read/Write/Format
over 350 formats

XENOFONT $49.95 +S/H
high quality text screen printouts
ideal for use in software documentation
Bold face and reverse video supported.

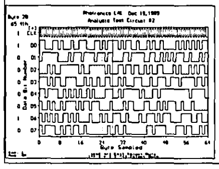
XenoSoft 2210 SIXTH STREET
BERKELEY, CA 94710
415-644-9366

Reader Service Number 39

BS DEGREES
...from fully-accredited Colleges. We help Computer Professionals avoid years of unnecessary class work. Get the job offers and recognition you deserve. Phone UDA for more information and our free booklet "Career Tactics and Strategies."
University Degree Advisory
☎ **800-765-7272**

Reader Service Number 168

LOW-BAND PC LOGIC ANALYZER
- Designed for general applications (below 1 MHz) where high-cost analysis is not needed
- Works with 5V digital logic
- 8 color-coded data lines & external clock input
- 6 range internal clock (1Hz-100KHz) for periodic sampling
- 8-bit logic probe mode
- Card plugs into IBM PC/XT/AT/386



36" cable with micro-clips
Full-featured software
Microsoft-mouse compatible
Fast assembler data acquisition
Help system & manual

PHOTRONICS $99.95
109 Camille Street
Amite, LA 70422
504-748-7090
MC,VISA, Company PO, COD

Reader Service Number 189

IN-CIRCUIT EMULATORS
8051 **FAMILY**
68HC11
CALL OR WRITE FOR FREE DEMO DISK AND BROCHURE!
nOHau CORPORATION
51 East Campbell Avenue
Campbell, CA 95008
(408) 866-1820
FAX (408) 378-7869

Reader Service Number 186

SHAREWARE AUTHORS: Give your programs the chance they deserve! With **Megapost**, your program is uploaded to Compuserve, GEnie, Delphi, BIX, and EXEC-PC. The low fee is probably less than you would pay just to open accounts with all these services. Included are one half-price update and notification of the downloading procedure and 3-month download count for each service. 15% discount for ASP members. Write for a free brochure.

Andrew M. Saucci, Jr.
641 Koelbel Ct. Baldwin, NY 11510-3915

Reader Service Number 200

PSFX TSR prints *any* Epson FX-85 or IBM Graphics Printer compatible output as PostScript, with graphics and IBM screen characters. $85.00
Also available for Novell's NetWare.

PSPlot Convert HP-GL to PostScript or EPS, any line width or color. PostScript oriented editor with test downloader and error display. ASCII file print, any font or size. AutoCAD to PostScript. Batch download of PC *and* MAC fonts. $185.00

EPScreen Capture PC screens as tiny EPS files, frames optional. Includes IBM screen font. $99.00

Legend Communications, Inc.
54 Rosedale Avenue West
Brampton, ON, Canada L6X 1K1
(416)450-1010 30 day guarantee (800)668-7077

PostScript

Reader Service Number 198

16 Megabytes EMS and/or Extended Memory
- Works on 8 or 16 bit bus
- 16 bit transfer on AT bus
- Single board design
- Includes RAM disk and extensive diagnostics
- Quantity/OEM discounts

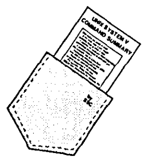
XT and AT Compatible

Designed, Manufactured, Sold and Serviced by P-tech
907 North 6th St. Lake City, MN 55041 (612)345-4555

Reader Service Number 3

Pocket References for UNIX - $6
BSD 4.2/4.3, System 5.2/5.3, XENIX
MS-DOS Pocket Reference - $3
C Library Reference - $6
We have other UNIX & VI products
Call or write for a complete list

SSC, Inc.
P.O. Box 55549-C
Seattle, WA 98155
(206)FOR-UNIX
(206)527-3385



Reader Service Number 188

sPORTt
serial PORT tester
sPORTt is a serial port tester for people who don't want to be data communications experts, but use or write software for serial devices, such as plotters, mice, digitizers, modems, serial printers, or computer to computer links.
- displays all data going in and out
- displays ASCII, decimal, hex, binary
- lets you choose handshaking lines
- sends/receives disk files
- automatically runs port diagnostics

Order today only $80
WHS
3037 Grass Valley Hwy #8201 Auburn, CA 95603
916-885-2480

Reader Service Number 192

EMCL
THE EXPERTS MATHEMATICAL C LIBRARY
EMCL sets the standards in numerical problem solving software in C.
EMCL's research and development staff represent years of experience in computer science, mathematics, and numerical analysis. EMCL is the mathematical library you can rely on. Available on all machines. Call or write for more information.
(404) 231-9582
IGS
15 Normandy Court Atlanta, GA 30324

Reader Service Number 195

Looking forward to your Micro Ad in the next issue!...

Complete Your Education . . .

Fill Out Your Collection of Micro C today!

ISSUE #1 (8/81)
Power Supply
1/2 PFM.PRN
16 pages

ISSUE #2 (10/81)
Parallel Print Driver
Drive Motor Control
16 pages

ISSUE #3 (12/81)
4 MHz Mods
Configuring Modem 7
Reverse Video Cursor
FORTHwords Begins
16 pages

ISSUE #4 (2/82)
Keyboard Translation
More 4 MHz Mods
Modems, Lync, and S10s
Undoing CP/M ERASE
20 pages

ISSUE #5 (4/82)
Two Text Editors
Double Density Review
20 pages

ISSUE #6 (6/82)
BBI EPROM Programmer
Customize Your Chars
Double Density Update
24 pages

ISSUE #7 (8/82)
6 Reviews Of C
Adding 6K Of RAM
On Your Own Begins
24 pages

ISSUE #8 (10/82)
SOLD OUT

ISSUE #9 (12/82)
BBII EPROM Program
Relocating Your CP/M
Serial Print Driver
Big Board I Fixes
32 pages

ISSUE #10 (2/83)
ISSUE #11 (4/83)
SOLD OUT

ISSUE #12 (6/83)
Bringing Up BBII
Double Sided Drives for BBI
Packet Radio
5 MHz for Kaypro
40 pages

ISSUE #13 (8/83)
CP/M Disk Directory
More 256K for BBI
Mini Front Panel
Cheap Fast Modem
BBI Printer Interface
Kaypro Reverse Video Mod
44 pages

ISSUE #14 (10/83)
BBII Installation
The Perfect Terminal
BBI Video Size
Video Jitter Fix
Kaypro Color Graphics Review
48 pages

ISSUE #15 (12/83)
Screen Dump Listing
Fixing Serial Ports
Playing Adventure
Upgrading Kaypro II To 4
Upgrading Kaypro 4 To 8
48 pages

ISSUE #16 (2/84)
Xerox 820 Column Restarts
BBI Double Density
BBII 5"/8" Interface Fix
Kaypro ZCPR Patch
Recovering Text From Memory
52 pages

ISSUE #17 (4/84)
Voice Synthesizer
Kaypro Morse Code Interface
68000-Based System Review
Inside CP/M 86
56 pages

ISSUE #18 (6/84)
Kaypro EPROM Programmer
I/O Byte: A Primer
Serial To Parallel Interface
Business COBOL
60 pages

ISSUE #19 (8/84)
Adding Winchester To BBII
6 MHz On The BBI
Bulletin Boards
Track Buffering On Slicer
4 MHz For The 820-I
64 pages

ISSUE #20 (10/84)
HSC 68000 Co-Processor
DynaDisk For The BBII
Serial Printer On BBI Sans S10
Cheap & Dirty Talker For Kaypro
Extended 8" Single Density
72 pages

ISSUE #21 (12/84)
Analog To Digital Interface
Installing Turbo Pascal
Low Intensity BBI Video
Turbo Pascal, The Early Days
80 pages

ISSUE #22 (2/85)
Xerox 820-II To A Kaypro-8
Sound Generator For the STD Bus
Reviews Of 256K RAM Expansion
88 pages

ISSUE #23 (4/85)
Automatic Disk Relogging
Interrupt Drive Serial Printer
Low Cost EPROM Eraser
Smart Video Controller
Review: MicroSphere RAM Disk
86 pages

ISSUE #24 (6/85)
C'ing Into Turbo Pascal
8" Drives On The Kaypro
68000 Versus 80x86
Soldering: The First Steps
88 pages

ISSUE #25 (8/85)
Why I Wrote A Debugger
The 32-Bit Super Chips
Programming The 32032
Modula II
RS-232C: The Interface
104 pages

ISSUE #26 (10/85)
Inside ZCPR3
Two Megabytes On DSI-32
SOG IV
The Future Of Computing
Graphics In Turbo Pascal
104 pages

ISSUE #27 (12/85)
SOLD OUT

ISSUE #28 (2/86)
Rescuing Lost Text From Memory
Introduction To Modula-2
Inside The PC
104 pages

ISSUE #29 (4/86)
Speeding Up Your XT
Prototyping In C
C Interpreters Reviewed
Benchmarking The PCs
104 pages

ISSUE #30 (6/86)
PROLOG On The PC
Expert Systems
Logic Programming
Building Your Own Logic Analyzer
PC-DOS For Non-Clones
104 pages

ISSUE #31 (8/86)
RAM Resident PC Speedup
Practical Programming In Modula-2
Game Theory In PROLOG and C
104 pages

ISSUE #32 (10/86)
Public Domain 32000: Hardware And Software
Writing A Printer Driver for MS-DOS
Recover A Directory By Reading & Writing Disk Sectors
96 pages

ISSUE #33 (12/86)
ISSUE #34 (2/87)
ISSUE #35 (4/87)
ISSUE #36 (6/87)
ISSUE #37 (9/87)
SOLD OUT

ISSUE #38 (11/87)
Parallel Processing
Laser Printers, Typesetters
Build A Graphics Scanner For $6, Part 2
Writing A Resident Program Extractor In C
96 pages

ISSUE #39 (1/88)
PC Graphics
Drawing The Mandelbrot And Julia Sets
Desktop Graphics
Designing A PC Work-station Board
Around the TMS-34010
96 pages

ISSUE #40 (3/88)
The Great C Issue
11 C Compilers
Writing A Simple Parser In C
C++, An Object Oriented C
Source Level Debugger For Turbo C
96 pages

ISSUE #41 (5/88)
Artificial Intelligence
3-D Graphics
Neural Networks
Logic Of Programming Languages
Applying Information Theory
96 pages

ISSUE # 42 (7/88)
Maintaining PCs
Keeping Your Hard Drives Running
Troubleshooting PCs
XT Theory of Operation
Simulating A Bus
Ray Tracing
96 pages

ISSUE #43 (9/88)
Building Databases
Build a C Database
Selecting a dBase III Compatible Compiler
Working with Paradox
Designing Custom PC Cards
Accessing dBase III Plus Records from Turbo Pascal
96 pages

ISSUE#44 (11/88)
Object-Oriented Programming
A Taste of Smalltalk
Actor
Thinking Objectively
Building MicroCad
Peripheral Technology-PT68K-2
96 pages

ISSUE #45 (1/89)
Computer Aided Design
CAD In A Consulting Business
Choosing PCB Layout Systems
Building Circuits With Your Computer
Secrets of Optimization
Finding Bargains in the Surplus Market
MASM 5.1
96 pages

ISSUE #46 (3/89)
Software Tools
The Art of Disassembly
Handling Interrupts With Any C
Hacking Sprint: Creating Display Drivers
Greatest C Compilers
Turning A PC into An Embedded Control System
96 pages

ISSUE #47 (5/89)
Robotics
The LIMBO Project
Starting A Robotics Company
How To Write and Use A SystemProfiler
Problem Solving and Creativity
Turn Your XT Into A Controller
Writing Code For Two Operating Systems
96 pages

ISSUE #48 (7/89)
Tools For The Physically Impaired
The Adventure Begins
Selecting A Talking Computer For A Blind Friend
Writing Software For The Blind
File Transfer Via Parallel Port
The LIMBO Project—Part Two
PCX Compatibility
A 68000-Based Multitasking Kernel
The Very Early Days of Computing
96 pages

ISSUE #49 (9/89)
I/O, I/O...
Build A Computer The Easy Way
PostScriptals
Prog. Logic Controllers
Driving Stepper Motors
Writing TSR Programs
Low Cost I/O For The PC
Interfacing 16-Bit Devices
96 pages

ISSUE # 50 (11/89)
3-D Graphics
3D Surface Generation
PC Video Frame Grabber
LIMBO, Part Three
PostScript, Part Two
UNIX For The PC
Capturing & Graphing A Voice In Real Time: Part 1

ISSUE #51 (1/90)
Embedded Systems
Embedding An XT Motherboard
Writing A Neural Network in C
LIMBO, Part 4
Getting Started in Hardware
Capture & Graph a Voice,
3D Surface Generation, Pt 2
Marketing Your Own Software

♦♦♦

To Order:

Phone: **1-800-888-8087**
Mail: **PO Box 223**
Bend, Oregon 97709

United States,	
Issues #1-34	$3.00 each ppd.
Issues #35-current	$3.95 each ppd.
Canada & Mexico	
All issues	$5.00 each ppd.
Foreign (air mail)	
All Issues	$7.00 each ppd.

MICRO C ADVERTISER'S INDEX

ISSUE 52

RS#	Advertiser	Page #
72	Acquired Intelligence	34
187	Alphax Systems, Inc.	64
160	Annabooks	55
199	Arizona Softool Associates	91
4	Austin Codeworks	23
197	Az-Tech Software, Inc.	93
147	Berry Computer	7
190	Bourbaki, Inc.	Inside Front
196	Building Block Software	93
31	CC Software	55
185	Carlson	91
15	Cascade Electronics	53
167	Computer Book Outlet	91
7	Compuview Products, Inc.	25
158	Cottage Resources	91
143	Covox	29
133	Data Rx	91
10	Emerald Microware	27
93	Erac Co	1
112	Garrison	61
**	Genus Microprogramming	15
195	IGS	93
149	Information Modes	65, 91
22	Integrand	6
191	JDYX Enterprises	77
198	Legend Communications	93
181	MCR	91
**	Micro Cornucopia	35, 71, 79, 89
37	Microprocessors Unlimited	75
2	Microsphere, Inc.	5, Inside Back
165	Mt. St. Helens Software	16
186	NOHAU	93
110	Nu-Mega Tech	2
161	Opal Fire Software	75
193	Oregon Software	89
3	PC Tech	93, Back Cover
119	Peripheral Technology	74
189	Photronics	93
139	Quantasm Corp	76
**	RJSwantek Inc	39
**	Revolution 2	32
171	Ryle Design	54
188	SSC	93
176	Sampson Engineering	91
200	Saucci	93
127	SemWare	43
162	Semi-Disk Systems	24
177	Silicon Alley	31
**	Sintar Software	64
129	Software Science, Inc.	47
40	Star-K Software Systems	60
173	Sutrasoft	61
182	Symmetric Research	91
**	TechnoJock's Software	20
178	Traxel Labs	14
194	Turbo Power	41
168	University Degree Advisory	93
62	V Communications	33, 59
124	Wenham Software Co	91
192	Western Hydrologic Systems	93
164	Whitney Software	91
39	Xenosoft	93
70	Zortech	13

** Contact Advertiser Directly.
When you write for information, please tell these folks you read about their products in **Micro Cornucopia.**

Issue #53 Micro Controllers

- **Building a Data Logger With the MC68HC11**
- **Building an LCD RS-232 Terminal Using the 68705**
- **Rational Numbers Revisited**
- **Voice Control**
- **Neural Networks: Case Studies**

By Gary Entsminger
P.O. Box 2091
Davis, CA 95617

Computing On Campus:

Laser Printers & Espionage

I knew Gary's appetite in reading material was a bit off, but this book sounds like just the ticket.

A striated sky lit the edge of campus—rapidly darkening like a bad mood—as I pulled into the parking lot nearest the U.C. Davis PC lab and read the sign at the open gate, **$1 night parking.**

I parked, picked up my floppy, and got out. I looked around, nowhere to buy a ticket. (The lot is semi-automated: you don't get a ticket at the gate; you buy one from a sixties-vintage machine, usually somewhere in the vicinity.)

I'd come to print graphs on the lab's much-in-demand laser printers (especially this late). I'd been warned that the lab closes promptly at midnight, about the time a printer usually frees up.

I crossed the street to a second lot. Saw nothing but cars. Where did they get their tickets? Inside information, no doubt. It was clear to me: I could park for a dollar only if I could solve the mystery of how to purchase a ticket before a campus cop wrote me one.

I sprinted the last hundred yards back to the truck. No cop (mixed luck since I was sure I could convince her I was trying to buy a ticket). Having no more time to waste, I decided I'd see if I could get right on a PC and a printer so I could be back in ten minutes.

Inside, keyboards were clacking. Rush hour in the PC lab. Students hovered over computers, sat on tables, and waited for printers to light up and put out. Eventually I found an unoccupied PC, loaded my floppy, and queued the graphs.

The Cuckoo's Egg

An hour later I was still sitting, reading now, since I'd dashed out to the truck to time-stamp a note to the cop and pick up a book I'd left lying on the truck seat. (Since my sophomore year in college, I've made a point never to go anywhere without a book, notepad, pen, and running shorts.)

This book was new, *The Cuckoo's Egg* by Clifford Stoll, an astronomer by training and a computer security expert by accident. I'd found it lying in one of Larry's stacks on my last trip to Bend. Since he didn't recall where he'd gotten it, and wasn't planning on reading it soon, he sent it back with me to California.

I read, still optimistically expecting to get my prints any minute. "Any hour" was more like it, since the PC network on campus doesn't think in minutes during rush hour. Fortunately, Cliff's story was diverting.

In a nutshell, his grant had run out, and he'd been transferred from the Keck Observatory to the Lawrence Berkeley Computing Lab. "Lucky for me that my laboratory recycled used astronomers." Not long at his new job as a system manager, he discovered that the UNIX accounting system, which handled all the billing for users of their dozen or so VAX computers, didn't balance. The previous month's bills showed a 75¢ shortfall.

No one much cared about the 75¢ (not a surprising revelation), but Cliff's cohorts decided that tracking down the error would be a good way for him to learn the system. "An error of a few thousand dollars is obvious and isn't hard to find. But errors in the pennies column arise from deeply buried problems, so finding these bugs is a natural test for a budding software wizard."

Cliff dug in. Within the first week (and 22 pages), he discovered a hacker breaking in via a bug in Richard Stallman's Gnu-Emacs editor (available on any UNIX system). The hacker swapped his special *atrun* file in for the legitimate one, allowing him to set up his own free account and cover his tracks.

He swapped in his phony *atrun*, set himself up as a remote system manager, then erased his and put the original *atrun* back. A neat trick, one that had apparently worked for months before Cliff discovered him.

The book would have ended on page 22 if Cliff had just disconnected the hacker, fixed the bug, and tightened security. But he didn't—he wanted to know why the hacker was hacking (call it sport, perhaps). And frankly, he wasn't sure he could tighten security enough to prevent this hacker or another like him from coming back. So he spent a year following crumbs through the international computing network.

Before the cuckoo's captured, Cliff finds out a lot about *how things work*—the international network, the FBI (they won't investigate a case unless it's worth a million dollars), the CIA, telephone tracing and tapping, links between databases, holes in military security systems, and the best places to shop for organic produce in Berkeley.

His trace eventually led to the Hanover Hacker and the University of Bremen, Germany. (What's their parking situation like?)

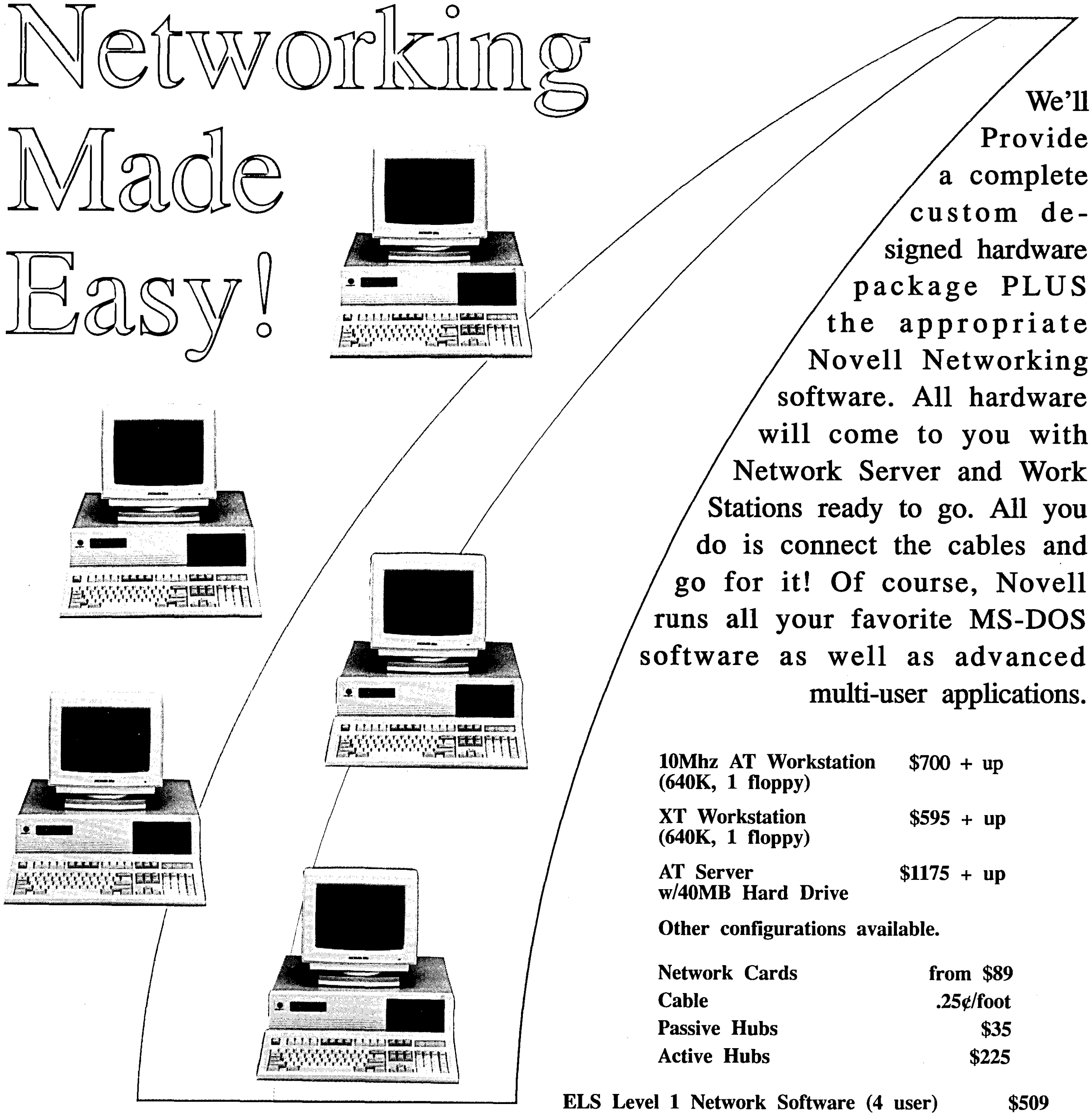
Intrigued by networking, UNIX, strange-but true espionage? Then check out *Cuckoo.*

As for my ticket mystery, I didn't buy one; but I didn't get one either. And despite the rush hour traffic, I got six printouts.

Stoll, Clifford; *The Cuckoo's Egg*; 1989; Doubleday.

♦ ♦ ♦

Networking Made Easy!



We'll Provide a complete custom designed hardware package PLUS the appropriate Novell Networking software. All hardware will come to you with Network Server and Work Stations ready to go. All you do is connect the cables and go for it! Of course, Novell runs all your favorite MS-DOS software as well as advanced multi-user applications.

10Mhz AT Workstation (640K, 1 floppy)	$700 + up
XT Workstation (640K, 1 floppy)	$595 + up
AT Server w/40MB Hard Drive	$1175 + up

Other configurations available.

Network Cards	from $89
Cable	.25¢/foot
Passive Hubs	$35
Active Hubs	$225

ELS Level 1 Network Software (4 user)	$509
ELS Level 2 Network Software (8 user)	$1100
Novell Advanced Netware (100 users)	$1995

MicroSphere INC. COMPUTERS "HARDWARE MANUFACTURER SINCE 1983"

Orders Only Please! 1-800-234-8086

Tech Calls: **(503) 388-1194** Hours: Mon-Fri 9:00-5:30

855 N.W. WALL • BEND, OREGON 97701

NOVELL Authorized Dealer

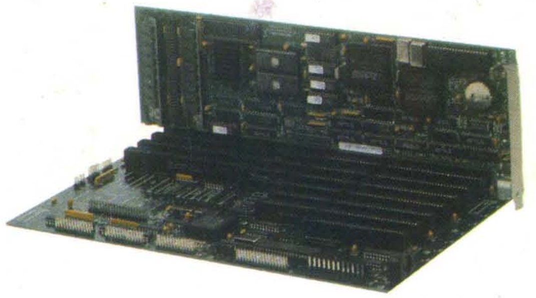
**Prices are subject to change without notice. Shipping CHARGES will be added. *1-year warranty/30-day money back (subject to restrictions)*

VERY HIGH PERFORMANCE
Processors, Memory, and Display Adapters

The X24 High performance processor

- 12 or 16 MHz 80286 with NO WAIT STATES!
- Small size ("XT" height and length) passive bus design
- 1 to 4 Mbyte 0 wait state dynamic memory
- Fully "AT" compatible Award BIOS
- Runs DOS versions 2.2 and later, Xenix and OS/2

The X24 combines the best of motherboard and backplane designs in a 100% AT compatible system. Incorporating a 16 MHz 80286, the X24 processor is designed to operate with the PC Tech Advanced System Motherboard, which contains the peripheral interfaces (hard disk, floppy disk, two serial ports and a parallel port). The X24 processor can also be used with other totally passive bus backplanes. Most critical components including the microprocessor and up to 4 megabytes of fast memory are contained on a single PC size plug-in card. This allows the processor and main system memory to be serviced or upgraded without disturbing other peripherals such as serial ports and disk drives.

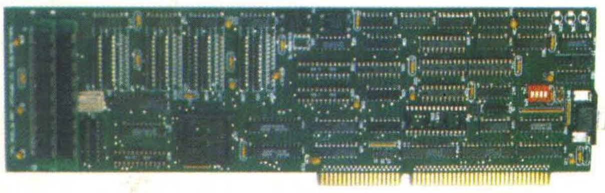

PC Tech X24 and ASMB

The PC Tech Advanced System Motherboard

- Built in "IDE" interface for AT interface type hard drives
- Fully AT compatible floppy disk support for 3.5", 5.25" drives, capacities of 360k, 1.2m and 1.44m
- Two serial ports and one parallel port
- 8 total expansion slots PC/XT/AT compatible (4 slots have 32 bit bus)

The PC Tech Advanced System Motherboard is designed to complement PC Tech's X24 and X32 high performance processor cards. It contains the mass storage interfaces necessary for a complete system, plus the basic I/O required in most systems. Extra care has been given to FCC compliance by design.

34010 Monochrome Graphics Adapter II


PC Tech Mono-II

- Up to 384k bytes display memory
- Up to 2 Megabytes program memory
- Software is RAM based, allowing complete operating software replacement and timing re-programming from the host bus
- 34010 program loader included. Assembler, debugger, and C compiler available.
- Full hardware and software CGA, MDA and Hercules emulation
- Single bit shared memory bit-map with optional resolution up to 2048 x 1536 (736 x 1008 standard)
- Very high resolution COLOR version available
- Custom 34010 software development available

The TMS34010 is a true general purpose graphics processor. PC Tech makes the total processing power of the 34010 available to both programmers and end users. Our 34010 Monochrome Graphics Adapter is designed to allow programming from the PC/XT/AT host bus. You can completely replace our 34010 software with yours to directly harness the incredible image processing power of the TMS 34010 for your application. We make a complete set of development tools available, including an assembler, C compiler, program loader, 34010 debugger, and PC interface tracer/debugger. Our standard product includes support for extended CGA, MDA and Hercules emulation as well as a host addressable graphics bit- map. We also support and recommend the DGIS graphics interface standard (from Graphic Software Systems) for applications development as an alternative to native 34010 software development. Ready to run drivers are available for most major applications software packages as well.

Custom Designs Available

PC Tech will license most products for non-exclusive manufacture. We will also customize any of our designs to better meet your needs on our in-house CAD systems. All of our standard products are available in private label versions.

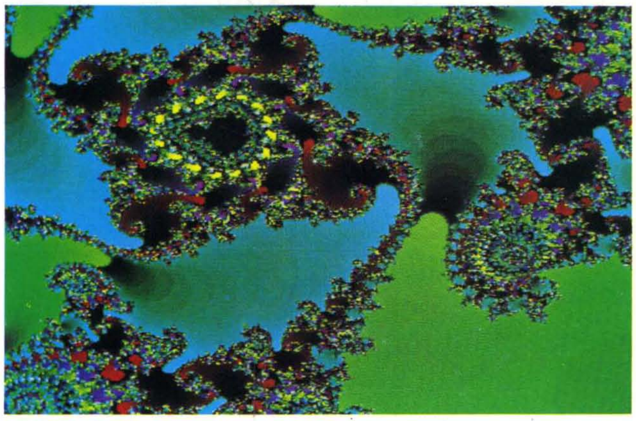
About PC Tech

PC Tech has been designing, manufacturing and marketing high performance PC related products for over three years. Our standard product line includes processor, memory, and video products. All products are designed, manufactured and supported in our Lake City, Minnesota facilities.

Designed, Sold and Serviced By:

PC Tech

907 N. 6th St., Lake City, MN 55041
(612) 345-4555 • (612) 345-5514 (FAX)


High resolution fractal produced on the PC Tech COLOR 34010

Reader Service Number 3

PC, XT, AT, DGIS, Hercules, and Award BIOS are trademarks or registered trademarks of their respective companies.