No. 43 September/October 1988 $3.95

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

Building Databases

Whether you're working with databases professionally or just collecting data as a hobby, this issue will fill you in on all the sorted details.

Accessing dBASE III Plus Records From Turbo Pascal

An inside look at the structure of dBASE files and a close look at ways to access them with Pascal. page 8

Build A C Database page 14
Selecting A dBASE III Compatible Compiler page 36
Working With Paradox page 30
Build A Pascal Database page 70

And A Special Project:

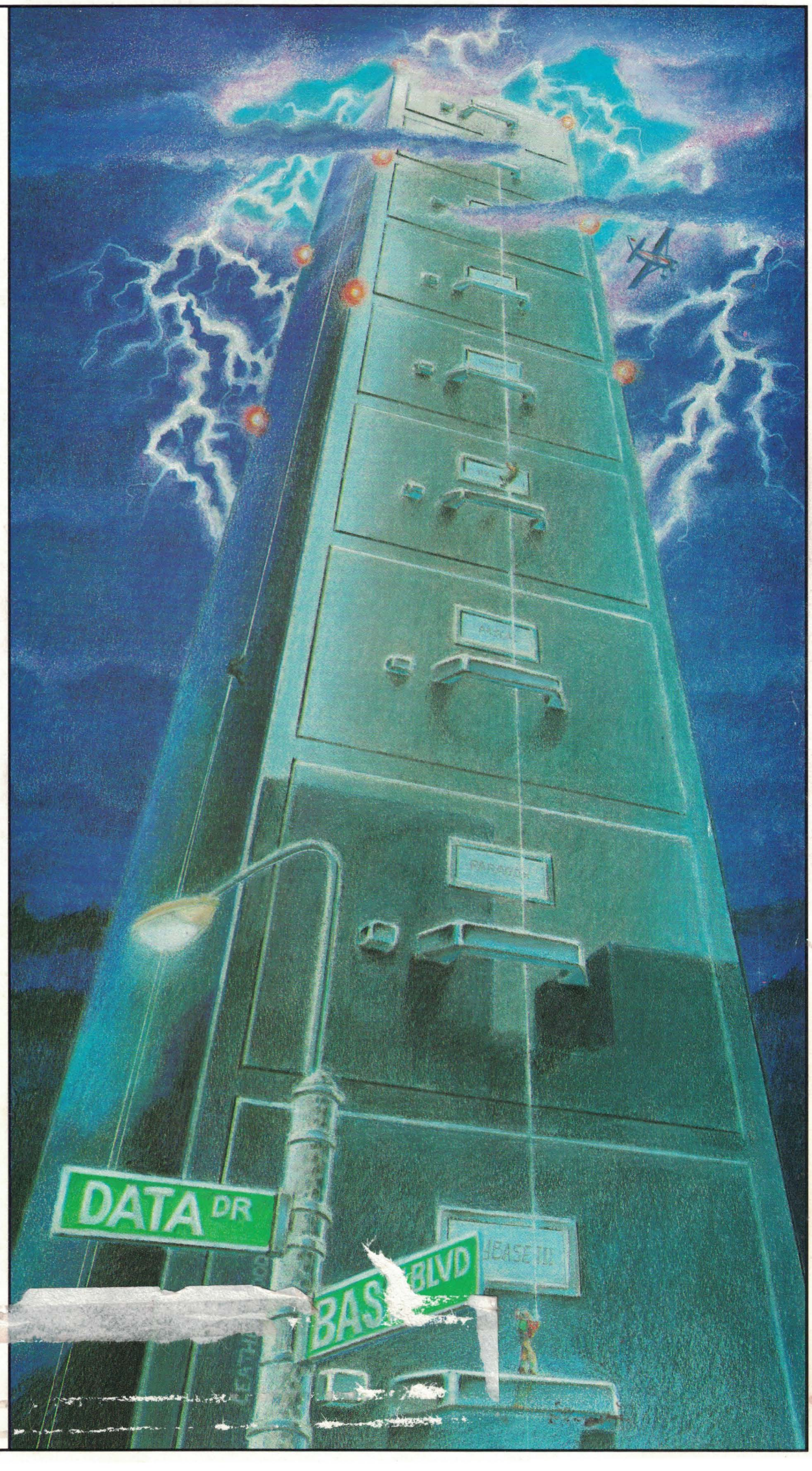
Designing Custom PC Cards

Designing custom PC cards has remained the private domain of the hardware guru. Now Bruce Eckel shows you how to produce your own in an hour. page 42

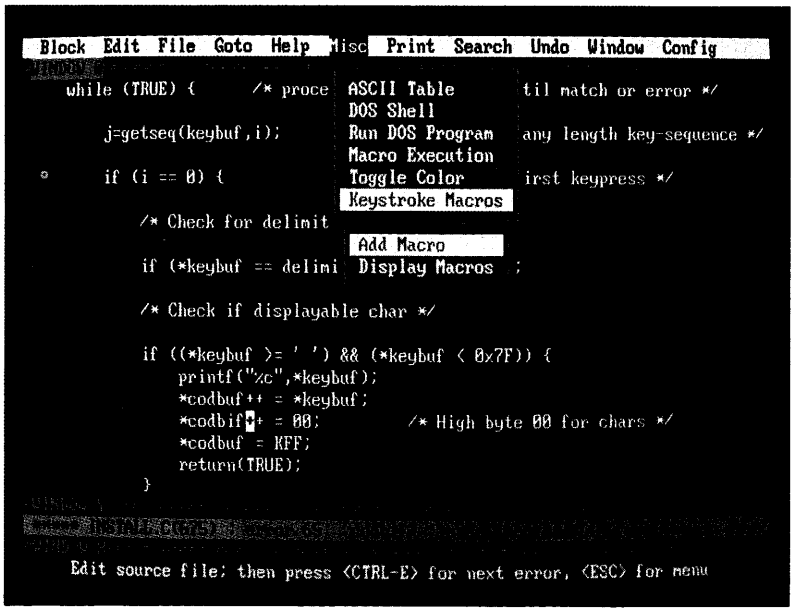
Plus:

The Early Days Of Micro C page 64
Very Fast Fractals page 22
New D Compiler page 69

And Much, Much, More



0 74470 19388 3 08



VEDIT PLUS

#1 PROGRAMMABLE EDITOR

NEW VERSION 3.0

- **Best Multi-Level Undo**
- **Regular Expressions**
- **Pop-Up ASCII Table**
- **Pull-Down Menus**
- **Compiler Support**
- **Column Blocks**

Until now, if you wanted the best Undo, the best compiler support, regular expressions and column blocks you chose BRIEF™. If you wanted unlimited keystroke macros, the best configurability, "off the cuff" command language macros and blazing speed, you chose VEDIT PLUS.®

Now the Choice is Easy

The all new VEDIT PLUS 3.0 gives you the best Undo of any editor, the best compiler support, unequaled windows, true regular expressions and extensive new features. We're leading the way with easy to use pull down menus, context sensitive help, a pop-up ASCII table, new printing options and much more. Incredibly, VEDIT PLUS 3.0 is now twice as fast as before and, at only 60K in size, it loads fast!

Completely Configurable

Change a few keys or redefine the entire keyboard, VEDIT PLUS adjusts to your editing style in minutes. You can even create new editing functions using simple keystroke macros or fine tune existing ones. VEDIT PLUS is so configurable that it easily emulates other editors and word processors (WordStar and Word Perfect emulation included). Quickly access editing functions with a single key or through the pull-down menus.

Try before You Buy

We challenge you to experience the dazzling performance and exceptional features that make VEDIT PLUS the best choice. Our evaluation disk includes the complete editor.* Learn VEDIT PLUS using our extensive "training" macro that gives instructions in one window while you experiment in another. See for yourself why no other macro language comes close.

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

Supports the IBM PC, XT, AT and PS/2 including DESQview, Microsoft Windows, PC-MOS/386, Concurrent DOS and most networks. Also available for MS-DOS, CP/M-86 and FlexOS. (Yes! We support windows on CRT terminals.) $185.

*Also available for TI Professional, Tandy 2000, DEC Rainbow, WYSE 700, Amdek 1280 and Others.
*Free evaluation disk is fully functional and can even edit small files.

FREE EVALUATION COPY* Call 1-800-45-VEDIT

- **Fully Network Compatible**
- **Call for XENIX and OS/2 versions**
- **30 Day Money-back guarantee**

Features of VEDIT PLUS 3.0

- Simultaneously edit up to 37 files of unlimited size.
- Variable sized windows; multiple windows per file.
- Execute DOS commands and other programs.
- Flexible "cut and paste" with 36 "scratch-pad" buffers.
- Block operations by line, character or column.
- Search with pattern matching or regular expressions.
- Configuration—determine your own keyboard layout, create your own editing functions, support any screen size.
- Select window colors, support 43 line EGA, 50 line VGA.

EASY TO USE

- Modern pull-down menu system. Pop-up ASCII table.
- Context sensitive on-line help is user changeable.
- Multi-level Undo (100 to 1000 levels). Undo keystroke by keystroke or line by line.
- On-line integer calculator (also algebraic expressions).
- Keystroke macros speed editing, menu function "hot keys."

FOR PROGRAMMERS

- Automatic Indent/Undent for "C," PL/I, PASCAL.
- Match/check nested parentheses, e.g. "{" and "}" for "C."
- Flexible macro runs popular compilers and automatically moves cursor to each error in your program. Easily changed to support new compilers and assemblers.

FOR WRITERS

- Word wrap, paragraph formatting and justification.
- Convert to/from Wordstar and mainframe files.
- Flexible printing; fully adjustable margins and Tab stops.

MACRO PROGRAMMING LANGUAGE

- If-then-else, looping, testing, string compare, branching, user prompts, keyboard input, 24 bit algebraic expressions.
- Flexible windowing—forms entry, select size, color, etc.
- Simplifies complex text processing, formatting, conversions.
- Extensive 400 page manual with hundreds of examples.

1955 Pauline Blvd., Ann Arbor, MI 48103
(313) 996-1299 • Telex 701821 • Fax (313) 996-1308

CompuView

QUALITY PRODUCTS AT COMPETITIVE PRICES!

CASES & POWER SUPPLY

150 Watt Power Supply (XT)	50.00
200 Watt Power Supply (AT)	80.00
XT Slide Case	34.00
XT Flip Top or XT Slide with Lock & LED	38.00
AT with Lock & LED	65.00

MONITORS

EGA/CGA (Auto Switch)	395.00
VGA/EGA/CGA Multi Sync	495.00
CGA Color	295.00
Amber 12" TTL	89.00
Green 12" TTL	89.00
VGA Analog	575.00

VIDEO CARDS

Color/Graphics/Parallel	52.00
256K EGA Graphics	125.00
Mono/Graphics/Parallel	49.00
ATI Graphics Solution— Mono, Herc. Color Emulation on Mono CGA (List 299)	125.00
ATI Wonder Auto Switch Mono, Herc CGA, EGA, VGA Any monitor, Any software, Auto conversion (List 499)	240.00
EGA, CGA, VGA (640x480)	185.00
VGA Analog	320.00

EXPANSION CARDS

Clock Card	22.00
Dual Floppy Disk Controller	25.00
Joystick	25.00
Gravis Analog Joystick	49.95
Game Port	19.00
Multi-Function, 1 ser/par/clk/game/ 2 floppy	61.00
Parallel (printer)	18.00
Dual Serial Port Card — 1 installed switchable Com 1, 2, 3, or 4	22.00
Kit for 2nd Port	20.00
640K RAM (ØK installed)	35.00

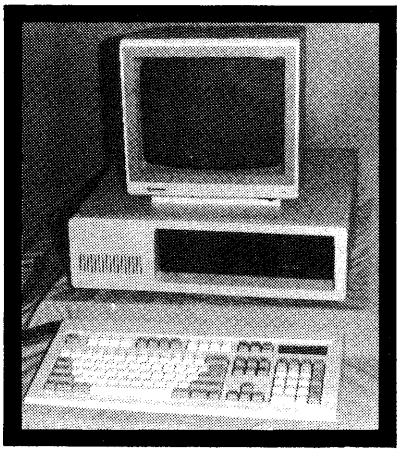
Prices are subject to change without notice. Shipping CHARGES will be added.

MOTHERBOARDS

XT/Turbo 4.77/10mhz	99.00
AT 6/10 mhz Choice of Award, Phoenix or DTK Bios	279.00
XT/Turbo 4.77/8 mhz	89.00
80386 8/16 mhz/DTK Bios	1195.00
80386 8/20 mhz/DTK Bios	1395.00
For XT/AT memory	$Call

FLOPPY DISK DRIVES

Fujitsu 360K	89.00
Toshiba 360K	99.00
Teac 1.2 MB	110.00
Toshiba 3½" Drive Kit 720K	110.00
Toshiba 3½" Drive 1.44mb	145.00



— Pictured keyboard is 5339 —

KIT OPTIONS

MS DOS 3.21 w/GW Basic	85.00
MS DOS 3.3 w/GW Basic	95.00
*5339 Keyboard Sub	24.00
*Color Options: (Includes video card & monitor)	
CGA Color	200.00
CGA/EGA Color	380.00
CGA/EGA/VGA Color	450.00
ASSEMBLY AND TESTING	
XT Systems	60.00
AT/80386 Systems	80.00

XT KIT W/ 2 Floppy Drives.

Includes: 640K RAM, Serial, parallel and game ports, clock/calendar, AT-Style keyboard, cabinet, power supply, mono graphics card and amber or green monitor. Keyboard switchable turbo.

8 mhz with lock, LED, Reset & Turboswitch	825.00
10mhz with lock, LED, Reset & Turboswitch	895.00

XT KIT W/20MB Hard Drive.

Includes: 640K RAM, Serial, parallel and game ports, clock/calendar, AT-Style keyboard, cabinet, power supply, mono graphics card and amber or green monitor. Keyboard switchable turbo.

8 mhz with lock, LED, Reset & Turboswitch	1095.00*
10mhz with lock, LED, Reset & Turboswitch	1165.00*

*(For 30MB Miniscribe add $15.00)

80386 KIT —

Includes: 1MB RAM, 1 360K floppy drive, 1-1.2 MB FD, 1 40MB HD, DTK bios, switchable keyboard, monochrome monitor, monographics. Serial/parallel ports, case, power supply, game port, clock/calendar. Main board made in U.S.A.

8/16 mhz	2795.00
8/20 mhz	2995.00

80286 - AT KIT

Includes: 640K RAM, 1.2 MB FD, 1 360K floppy drive and 40 MB Miniscribe 3650 HD, 6/10mhz, serial, parallel and game ports, clock/ calendar, AT-style keyboard, cabinet, power supply, monographics card, amber or green monitor, keyboard switchable turbo.

40 MB HD (MFM)	1525.00
60 MB HD (RLL)	1595.00

KEYBOARDS

5339 Professional XT-AT w/12 function key	69.00
5060 Keyboard AT Style	49.00
KB101 Keytronic	67.00

Free Instructions with Each System

HARD DRIVES & CONTROLLERS

AT 40 MB Miniscribe 3650	375.00
AT ST 4053 HD	525.00
AT (MFM) Hard Drive & floppy controller (WD)	130.00
AT RLL HD & FD controller	189.00
20 MB Miniscribe HD with controller (XT)	340.00
30 MB Miniscribe HD with controller	355.00

SOFTWARE

The Twin Spreadsheet	49.00
Leading Edge Word Processor	49.00
Ventura Desktop Publisher by Xerox	525.00
Learning Dos-MicroSoft	45.00

ACCESSORIES

1200 Baud Modem — Internal (Leading Edge Model L) Hayes compatible	99.00
2400 Baud Modem — Internal (Leading Edge Model L) Hayes compatible	179.00
1200 Baud Modem — External Hayes compatible	119.00
V20-8mhz	14.00
Memory Chips	**(call for prices)**

DEPEND ON MICROSPHERE

The components and products we sell are chosen specifically because they have been proven in our own use and testing. We guarantee our cards will be compatible when purchased all together.

NEW REVISED!

BUILDING YOUR OWN CLONE

****FREE BOOKLET****

*90-day warranty/30-day money back (subject to restrictions)

MicroSphere INC. COMPUTERS

MicroSphere, Inc.
P.O. Box 1221
Bend, Oregon 97709
(503) 388-1194
Hours: Monday-Friday 9:00-5:30

Reader Service Number 2

THE MICRO TECHNICAL JOURNAL

MICRO CORNUCOPIA

SEPT/OCT 1988 - ISSUE NO. 43

FEATURES

8 Michael Greene
Accessing dBASE III Plus Records From Turbo Pascal
There's a lot of dBASE formatted data out there just waiting to be used by real programmers. So warm up your copy of Turbo and have at it.

14 Scott Robert Ladd
Building A Database With C
C isn't know as a database language, but where there's C there's a library. Scott compares four C database libraries.

18 Patrick McGuire
Selecting A Database Compiler
OK, so you've got dBASE III code. You're stuck, right? Not at all. There are some very fast compilers for dBASE III that add some some very interesting extensions.

22 H. W. Stockman
Fast Fractals: Programming The 386 Under MS-DOS
If fractals are anything, they're slow. Well, they don't have to be. Just use fixed point math & some 386 instructions.

30 Ben Sevier
The PARADOXical Developer
If you're going to look at the database environments you'll have to look closely at Paradox.

36 Brett Fishburne
Menu Generator For dBASE III

42 Bruce Eckel
Delving Into The Black Arts
This is the hardware article you'll be digging for within months, I guarantee it. There's so little information on do-it-yourself custom XT/AT cards that by itself this'll clear out the back issues.

48 Larry Fogg
Fractal Miscellany
Larry's machine is back to generating fractals. Only now they're EGA fractals. Kinda'colors his outlook.

COLUMNS

52 **C'ing Clearly**
ANSI. The new standard.

56 **86 World**
Repertoire and AT block moves. Laine's in fine style.

61 **ShareWare**
Protecting yourself from viruses.

64 **On Your Own**
Dave recounts the early days of Micro C.

69 **Culture Corner**
Way beyond C++: Announcing the new D compiler.

70 **Pascal Column**
John fires up Borland's database toolbox.

90 **Technical Tips**

CP/M CORNER

82 **CP/M Notes**

84 **Kaypro Column**

FUTURE TENSE

72 **Tidbits**

96 **Last Page**

Cover illustration by Paul Leatherwood

THE MICRO TECHNICAL JOURNAL
MICRO CORNUCOPIA

Editor and Publisher
David J. Thompson

Associate Editors
Gary Entsminger
Cary Gatton

Technical Department
Larry Fogg

Director of Advertising & Distribution
Laura Logan/Jackie Ringsage

Accounting
Sandy Thompson

Order Department
Tammy Westfall

Graphic Design
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.

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 (surfaca)	$36.00
1 yr. Foreign (airmail)	$50.00

Make all orders payable in U.S. funds on a U.S. bank, please.

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

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

CUSTOMER SERVICE: For orders and subscription problems call 503-382-8048, 9 am to 5 pm, Pacific time, M-F.

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

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

By David Thompson

Hey, Where'd That Book Go?

The entire west wall of my office is covered with shelves, shelves filled with magazines and books. The magazines do their own thing, slowly filling their allotted space and then spilling inexorably onto the floor. (No lack of breeding, I suppose.)

Books are a whole different problem. Books are quite invaluable and they know it. Once you've become attached to a book, it leaves (to who knows where). Take *The C Programming Language* by Kernighan and Ritchie; I've bought four or five copies of that book and every single one has relocated. I borrowed Larry's copy the other day but within an hour it too was gone (he'd reclaimed it).

Review books have their own shelf. They're a lot like magazines, they just multiply on their shelf until there's no more room. Then they too start hitting the floor.

Occasionally we get a review book that looks so interesting it isn't left in the purgatory of the review shelf. (Of course, then it becomes indispensable and disappears.)

One Good Book

If you're currently into electricity, or on a low voltage power trip, then *Living on 12 Volts with Ample Power* is your book (ISBN 0-945415-02-8). There's a whole battery of information herein. This is the best information I've seen on batteries (especially large lead cells) since Gutenberg invented acid remarks.

I knew an old geezer at a battery plant who designed battery packages and chargers for forklift manufacturers. He knew more about restoring, maintaining, and driving lead acid cells than any three university types.

Some of what he knew rubbed off on me and I spent hours poring over the leather bound, acid stained, 1200-page manual he lent me.

Well, *Living On 12 Volts* isn't 1200 pages and it isn't leather bound, but it deals with batteries clearly and concisely. I'm impressed. (You'd be surprised how much longer a properly used battery will work.)

But this book isn't just about batteries. Authors David Smead and Ruth Ishihara have put together a sailboat complete with computer, refrigerator, freezer, lights, electronics, solar panels, windmill (?), batteries, you name it. They walk you through the calculations of average electrical load and the resultant generation and storage requirements.

Good book for those of you who want (or need) to detach yourself from the mains.

(Continued on page 76)

No kidding. The new version of Genifer® supports dBASE IV™, dBASE III PLUS™, CLIPPER™, dBXL™, FoxBASE+™ QUICKSILVER™, and other dialects, too!

No. We didn't use a crystal ball to figure out exactly what the new dBASE IV would look like. Instead, we designed Version 2 of Genifer to include loads of built-in flexibility, so you get the advantage of each dialect's unique syntax. The result? Genifer generates the very best performing application possible in each dialect. *And* delivers unmatched flexibility.

What exactly is Genifer?

Genifer is a one-of-a-kind, stand-alone program that helps you develop superb applications, quickly and easily. All *you* do is describe your data structures, define the characteristics of each field, paint out your menu, screen, and report design, and wait a few minutes while Genifer does all the work. In no time you'll have the application you want, written in beautiful, clean, well-structured dBASE code. To complete your application, Genifer now builds comprehensive documentation, *automatically!*

What makes our new version of Genifer special?

It's the powerful, new features and enhancements we've added. The ones many of our ten thousand loyal Genifer users have told us they've been looking for. Features like enhanced multi-file capability for both screens and reports. Automatic index expression generation for fast linkage between files. An easier developer interface, including context sensitive help. An expanded Data Dictionary, plus multi-page screen programs, more powerful report programs, automatic compiler support, and an all new and fully documented template language.

It's Genifer's unique Template Language that lets us support *all* dBASE dialects. And helps you produce applications that exactly meet your users' needs. The templates, which define the form and structure of generated programs, are normal ASCII files that you can modify any way you like.

Genifer, Version 2, comes with pre-written templates for menu programs, maintenance, inquiry, and report programs. *Use them "as is" to produce complete, ready-to-run applications* of a quality that Genifer has always been famous for. Special needs?

> The dBASE IV Promise:
>
> Within 30 days of dBASE IV's availability in retail stores, Bytel will provide a free Genifer upgrade (to all registered Genifer Version 2 users) that will support the new dBASE IV syntax!

No sweat! If you can read dBASE code, you can modify our templates or create brand new ones of your own!

Best of all, you won't have to worry about new dialects any more. Just develop your application today with Genifer. Then use our dBASE IV templates to regenerate. Your user will get the full advantages of all that's new in dBASE IV!

Get a fact-filled brochure, absolutely FREE!

To find out more about Genifer, Version 2, visit your dealer or call toll-free for our free brochure, Genifer – A Breakthrough in dBASE Application Development. We think you'll be impressed with what Genifer can do for *you!*

NOTE: Genifer Version 2 includes templates supporting single-user dBASE and any other fully compatible dialect. Add-on templates, with usage notes, are available for networked dBASE, Clipper, FoxBASE+, and Wordtech's dBXL/Quicksilver. All, except the dBASE set, include both single user and multi-user (LAN) templates.

Call toll-free: 800-631-2229
In California: 800-541-3366
(415) 527-1157 Telex: 176609

Genifer.

☐ **Yes!** I want to find out more about what Genifer Version 2 can do for me so send me your free brochure, *Genifer – A Breakthrough in dBASE Application Development*

NAME _____

TITLE _____

COMPANY _____

ADDRESS _____

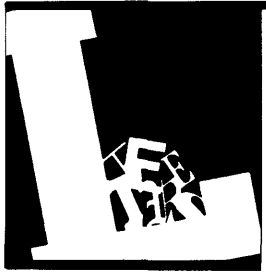
CITY _____

STATE _____ ZIP _____

( ) PHONE _____

bytelcorporation 1029 Solano Avenue Berkeley, CA 94706

Reader Service Number 104



Letters

Long Live Ludefisk

In reference to Mr. Stump's article in *Micro C* Issue #41, I would like to say that, in part, I agree with him: he has become (or more likely always has been) a boring guy.

Both the introduction and the article say what a culture shock it was to move to Minnesota. True, we don't have acres and acres of sunbaked land with cobras, tarantulas, and scorpions. But we do enjoy our parks, wildlife, and over 10,000 lakes. And, I've heard it's hard to ski, snowmobile, or ice fish in Turkey in the winter.

Just to clarify another point: very few Minnesotans actually eat ludefisk! Actually, the majority of Minnesotans have a Germanic heritage. Although I personally have never tried ludefisk (fish), given the choice, I would much rather try it than some Turkish delicacies.

He insinuates a lack of adventure here. Obviously he hasn't braved the Apple River in Somerset, Wisconsin, in the middle of July. It's where a lot of Minnesotans go for a three-hour float down the river on an inner tube, drinking beer or what have you. Although not as exotic as fighting off wild hippos from a reed canoe, it can be just as dangerous with rock infested rapids and hundreds of half-crazed people behind you.

I am continually amazed at life's little ironies. Mr. Stump finds it amusing to publish his bigoted views, and life has given him such an appropriate family name.

Glenn T. Puchtel
Native Minnesotan
11121 Cottonwood St. NW
Coon Rapids, MN 55433

Tale Of The Trojan Burner

The Sunshine PROM burner you reviewed in Issue #36 has a bug. I'm using the four hole model EW-904B with EPROM.EXE version 5.7 in an AT clone running the DTK 286 BIOS, version 2.34.

The burner works flawlessly with Intel EPROMs, but when burning TI TMS27C256-25 parts it reports an error at location $4000 and won't complete the burn. I first confirmed that the same EPROMs could be programmed on other types of burners, then called the vendor, McTek.

I spoke to Charles, who said he knew of the problem. In fact, he said that the unit seems to unreliably burn American parts in general. But hey, this is really no problem because, after all, it burns Japanese parts just fine! Score one for point of view. My fault for not asking.

As a last resort, I asked him when they expected to have the problem fixed. His response was not encouraging — it's not a problem, so why fix it? And yes, I have the latest software.

This reminded me of the early sci-fi stories about household appliances with a secret personality. One day all of the washing machines awake, arise, and destroy America's laundry. Wonder what's in those import PROMs. I'm glad I don't have a food processor (I've got lots of the other kind). Avoid smart bathroom appliances.

Well, all is not entirely black because I've had good luck burning Intel D27256 parts. Maybe only the CMOS variants are affected. I'd surmise the problem lies in the quick burn code they're using. Any ideas?

Pat Barrett
7035 SW 83rd Ave.
Portland, OR 97223

FCB Size

Congratulations on creating a journal which is such a pleasure to read that I actually look forward to receiving my next issue. I've been enjoying *Micro C* for six months now. I find it informative and interesting while possessing a personable quality.

In Issue #41, Scott Ladd's C'ing Clearly column was helpful in describing some internals of MS-DOS that were new to me. Unfortunately, anyone who used the Program Segment Prefix layout in Figure 3 on page 50 may have experienced difficulties accessing the command tail fields.

For reasons I'm not sure of, the second File Control Block should be 27 bytes long instead of 16 as shown in the table. It's a small oversight, but I thought it might be helpful to others if it were pointed out.

The SOG sounds like a great time. Recalling a certain Turkish rafting expedition, getting Laine a couple of extra paddles might not be a bad idea.

R. Bruce Miller
Innovative Systems
7617 Weatherworn Way
Columbia, MD 21046

Editor's note: Near as we can tell, the second FCB actually takes up 20 bytes with a seven byte extension tacked onto the beginning. But the offsets given by Scott for the command tail length byte (80h), and the command tail itself (81h), should work fine.

By the time this issue hits the streets, SOG will be a fond memory. As for Laine and some extra paddles, we're more likely to use the paddles on him.

Seems he was having so much fun rafting those Turkish rivers that he couldn't be bothered with anything so mundane as a column for Micro C. *I'm sure we'll eventually get some kind of lame excuse like, "My pet gibbon ate it," or "I lost it in some class IV rapids."*

Addendum to editor's note: Good ol' Laine came through after all. We got yet another "Extremely Urgent" express package from Turkey. People in Bend are starting to wonder about us.

(Continued on page 67)

Call for Extra Savings on Weekly Specials

DISCOVER PARADISE

Programmer's Paradise Gives You Superb Selection, Personal Service and Unbeatable Prices!

Welcome to Paradise. The microcomputer software source that caters to your programming needs. Discover the Many Advantages of Paradise...

- **Lowest price guaranteed**
- **Huge inventory, latest versions**
- **Technical support**
- **Immediate shipment**
- **30-day money-back guarantee***
- **Knowledgeable sales staff**

Over 500 brand-name products in stock — if you don't see it, call!

We'll Match Any Nationally Advertised Price.

	LIST	OURS
386 SOFTWARE		
386-TO-THE-MAX	75	66
ALSYS 386 ADA	3655	3399
DESQVIEW	130	115
FOXBASE + /386	595	399
HIGH C-386	895	839
MICROPORT DOS MERGE (2-USERS)	399	359
MICROPORT SYS V/386 (COMPLETE)	899	799
MS WINDOWS/386	195	130
NDP C OR FORTRAN-386	595	535
PC-MOS/386 (1-USER)	195	181
PHARLAP 386|ASM/LINK	495	422
SCO VP/IX (2-USERS)	495	399
SCO XENIX SYS V 386 (COMPLETE)	1595	1199
VM/386	245	182
ARTIFICIAL INTELLIGENCE		
MULISP-87 INTERPRETER	300	199
PC SCHEME	95	86
T.I. PROCEDURE CONSULTANT	495	435
TURBO PROLOG V. 2.0	150	109
TURBO PROLOG TOOLBOX	100	69
ASSEMBLERS/LINKERS		
ADVANTAGE DISASSEMBLER	295	279
MS MACRO ASSEMBLER	150	105
OPTASM	195	179
PLINK86PLUS	495	279
BASIC		
DB/LIB	139	121
FINALLY!	99	90
FLASH-UP	89	80
MACH 2	79	60
MS BASIC COMPILER 6.0	295	229
MS QUICKBASIC	99	69
QUICKPAK	69	60
QUICKWINDOWS W/SOURCE	99	90
TRUE BASIC	100	90
TURBO BASIC	100	69
TURBO BASIC TOOLBOXES	100	69

DBASE TOOLS

		LIST	OURS
CLIPPER		695	399
D'ACTION		80	70
D'ANALYST		229	209
DBASE III PLUS		695	399
DBUG		195	149
EAGLE	*NEW*	495	395
FOXBASE +		395	249
FOX TOOL BOX		295	CALL
GENIFER		395	269
HI-SCREEN XL		149	129
R&R		150	129
SAY WHAT?!		50	40
SCANALYZER		50	45
TOM RETTIG'S LIBRARY		100	79
UI PROGRAMMER		295	229

OVER 100 DATABASE PRODUCTS IN STOCK

	LIST	OURS
C COMPILERS/INTERPRETERS		
C-TERP	298	232
INSTANT C	495	384
LATTICE C	450	259
MICROSOFT C	450	299
QUICK C	99	69
RUN/C PROFESSIONAL	250	159
TURBO C	100	69

	LIST	OURS
C LIBRARIES		
C ASYNCH MANAGER	175	137
C TOOLS PLUS/5.0	129	101
C UTILITY LIBRARY	185	125
CXPERT	395	335
DEVELOPER'S TOOLKIT FOR C	495	395
DESQVIEW API C LIBRARY	200	CALL
ESSENTIAL COMMUNICATIONS	185	125
COMMUNICATIONS PLUS	250	199
GREENLEAF TURBO FUNCTIONS	109	79
GREENLEAF COMM LIBRARY	229	155
GREENLEAF FUNCTIONS	209	145
PFORCE	395	215
RESIDENT C W/SOURCE	198	169
TIMESLICER	295	279
TURBO C TOOLS	129	101
COBOL		
MICROFOCUS COBOL/2	900	733
W/COBOL/2 TOOLSET	1800	1465
PERSONAL COBOL	149	121
MICROSOFT COBOL	700	465
OPT-TECH SORT	149	105
REALIA COBOL	995	799
W/REAL MENU	1145	929
REALCICS	995	799
RM/COBOL	950	763
RM/SCREENS	395	339
SCREENIO	400	382
DEBUGGERS		
ADVANCED TRACE-86	175	121
BREAKOUT	125	89
PERISCOPE I	455	373
PERISCOPE II	175	141
PERISCOPE III 10 MHZ	1395	1143
PFIX 86 PLUS	395	215
DISK/DOS/KEYBOARD UTILITIES		
COMMAND PLUS V. 2.0	80	70
DISK OPTIMIZER	60	55
FASTBACK +	189	142
MACE UTILITIES	99	90
NORTON COMMANDER	75	56
NORTON UTILITIES ADVANCED	150	99
PC TOOLS DELUXE	80	70
VFEATURE	80	75
XTREE PRO	129	111
EDITORS		
BRIEF	195	CALL
W/DBRIEF	275	CALL
EDIX	195	169
EMACS	295	268
EPSILON	195	151
KEDIT	150	129
MKS Vi	75	66
MULTI-EDIT	99	90
NORTON EDITOR	75	69
PC/EDT +	295	269
Pi EDITOR	195	165
SPF/PC	245	185
VEDIT PLUS	185	131
FILE MANAGEMENT		
BTRIEVE	245	185
XTRIEVE	245	189
REPORT OPTION	145	109

	LIST	OURS
BTRIEVE/N	595	455
XTRIEVE/N	595	459
REPORT OPTION/N	345	279
CBTREE	159	141
C-TREE	395	318
D-TREE	495	418
R-TREE	295	241
COMBINATIONS AVAILABLE	CALL	CALL
DBC III	250	172
DBC III PLUS	750	599
DB_VISTA OR DB_QUERY	195	CALL
INFORMIX PRODUCTS	CALL	CALL
XQL	795	599
FORTRAN		
LAHEY FORTRAN F77L-EM/32	895	799
LAHEY PERSONAL FORTRAN 77	95	86
MS FORTRAN	450	299
RM/FORTRAN	595	479
DIAGRAM'ER OR DOCUMENT'ER	129	115
GRAFMATIC OR PLOTMATIC	135	119
MAGUS NUMERICAL ANALYST	295	252
SPINDRIFT LIBRARY	149	135
GRAPHICS		
ADVANTAGE GRAPHICS (C)	250	229
ESSENTIAL GRAPHICS	299	229
GSS GRAPHIC DEV. TOOLKIT	495	399
HALO '88	325	229
HALO '88 (5 MICROSOFT LANG.)	595	399
METAWINDOW PLUS	275	232
METAWINDOW/PREMIUM	495	419
TURBOWINDOW/C	95	80
TURBO HALO (FOR TURBO C)	99	80
MODULA-2		
LOGITECH MODULA-2		
COMPILER PACK	99	81
DEVELOPMENT SYSTEM	249	199
TOOLKIT	169	141
SOLID B + TOOLBOX	100	89
STONYBROOK MODULA-2	195	179
MOUSE PRODUCTS		
LOGITECH HIREZ OR SERIES 2	CALL	CALL
MICROSOFT MOUSE BUS	150	99
OTHER VARIETIES	CALL	CALL
SUMMAMOUSE	119	99
OBJECT-ORIENTED PROGRAMMING		
ACTOR	495	439
ADVANTAGE C + +	495	479
PFORCE + +	395	215
SMALLTALK/V	100	85
APPLICATION PACKS	50	45
SMALLTALK/V286	200	175
OPERATING SYSTEMS		
MICROPORT 286 DOS MERGE	249	219
MICROPORT SYS V/AT	649	579
SCO XENIX SYSTEM V (COMP.)	1295	979
WENDIN-DOS	99	80
OTHER MICROPORT, SCO, WENDIN PRODUCTS	CALL	CALL
PASCAL COMPILERS		
MICROSOFT PASCAL	300	199
PASCAL-2	229	199
TURBO PASCAL	100	69
TURBO PASCAL DEV. LIB.	395	289

	LIST	OURS
TURBO PASCAL ADD-ONS		
ASCII TURBO PROG. (COMPLETE)	289	259
DOS/BIOS & MOUSE TOOLS	75	70
METRABYTE DATA ACQ. TOOLS	100	90
OVERLAY MANAGER 4.0	45	40
SCREEN SCULPTOR	125	96

OBJECT-ORIENTED LANGUAGES

ACTOR — Powerful new language built around object-oriented programming, with windows being defined as objects. Actor makes it easier to include and control windows in application programs.
List: $495 **Ours: $439**

SMALLTALK/V NEW V.2.0 — New version is a high-performance, production quality, object-oriented programming environment. Includes:
- Advanced user interface featuring windows, pop-up — menus and optional mouse.
- A set of tools for organizing and browsing the Smalltalk — source code.
- An incremental program development capability.
- Bitmap graphics with optional color support.

List: $100 **Ours: $85**

PFORCE + + — This C + + library provides everything necessary to build complete applications. Includes high-level classes for windows, databases, B-trees, fields, menus, rings, lists, communication tasks, time/date stamps, BIOS and DOS access, etc. Complete source code included.
List: $395 **Ours: $215**

ADVANTAGE C + + — ADVANTAGE C + + now has **MS Window's Support** and gives you the speed support and reliability you need to develop large complex programs with fewer bugs. Latest version supports MS C 4.0/5.0 and QuickC faster.
List: $495 **Ours: $479**

	LIST	OURS
T-DEBUG PLUS	45	41
W/SOURCE	90	79
TURBO ANALYST	75	69
TURBO ASYNCH PLUS	129	101
TURBO GEOMETRY LIBRARY	150	135
TURBO HALO	95	80
TURBO MAGIC	99	90
TURBO PASCAL TOOLBOXES	100	69
TURBO POWER TOOLS PLUS	129	101
TURBO PROFESSIONAL 4.0	99	89
TURBO WINDOW/PASCAL	95	80
SCREENS/WINDOWS		
C-SCAPE V 3.0	CALL	CALL
C-CELL	CALL	CALL
CURSES W/SOURCE	250	172
GREENLEAF DATA WINDOWS	295	229
HI-SCREEN XL	149	129
JYACC JAM	750	684
MICROSOFT WINDOWS	99	69
MS WINDOWS DEVELOPMENT KIT	500	329
PANEL PLUS	495	395
PANEL/QC OR /TC	129	99
SCREENSTAR W/SOURCE	198	169
TURBO POWER SCREEN	129	101
VIEW MANAGER	275	219
VITAMIN C	225	162
VC SCREEN	149	119
WINDOWS FOR DATA	295	CALL
W/SOURCE	590	CALL
ADDITIONAL PRODUCTS		
BABY/36 (RPG II)	3000	2699
CARBON COPY PLUS	195	142
CO-SESSION	195	179
DAN BRICKLIN'S DEMO PROGRAM	75	60
DEMO PROGRAM II	195	179
DB2C	299	272
EUREKA	167	119
FLOW CHARTING II	229	207
INTERACTIVE EASY FLOW	150	125

OS/2 DEVELOPMENT TOOLS

	LIST	OURS
BTRIEVE FOR OS/2	595	455
CALIFORNIA TEN PACK	99	79
EPSILON FOR OS/2	195	151
GREENLEAF DATA WINDOWS OS/2	395	279
GSS DEV TOOLKIT FOR OS/2	695	559
HELPME	99	75
KEDIT V. 4.0	175	139
MICROFOCUS COBOL/2	900	733
MICROSOFT LANGUAGES	CALL	CALL
PANEL PLUS FOR OS/2	495	395
RBASE FOR OS/2	895	CALL
VITAMIN C FOR OS/2	345	285
WINDOWS FOR DATA FOR OS/2	495	CALL

	LIST	OURS
JANUS ADA JET SET	170	160
MAGIC PC	195	179
MATHCAD	395	279
MKS RCS	189	169
MKS TOOLKIT	169	139
MS OS/2 PROG. TOOLKIT	350	239
MUMATH	300	189
NORTON GUIDES	100	69
PC-LINT	139	101
POLYMAKE	149	135
POLYTRON PVCS	CALL	CALL
PRE-C	295	159
PROTEUS	99	89
SEIDL VERSION MANAGER	300	269
SOURCE PRINT	97	80
TREE DIAGRAMMER	77	70

Full Line of MS-DOS, OS/2, Xenix, Macintosh Products Available.

Call or Write for Latest Free Catalog!

Terms and Policies
- **We honor MC, VISA, AMERICAN EXPRESS**

No surcharge on credit card or C.O.D. Prepayment by check. New York State residents add applicable sales tax. Shipping and handling $3.95 per item within the U.S., sent UPS ground. Rush and international service available. Call for prevailing rates.
- Programmer's Paradise will match any current nationally advertised price with equivalent terms for the products listed in this ad.
- Prices and Policies subject to change without notice.
- Hours 9AM EST — 7PM EST
- ***Mail Orders include your phone number.***

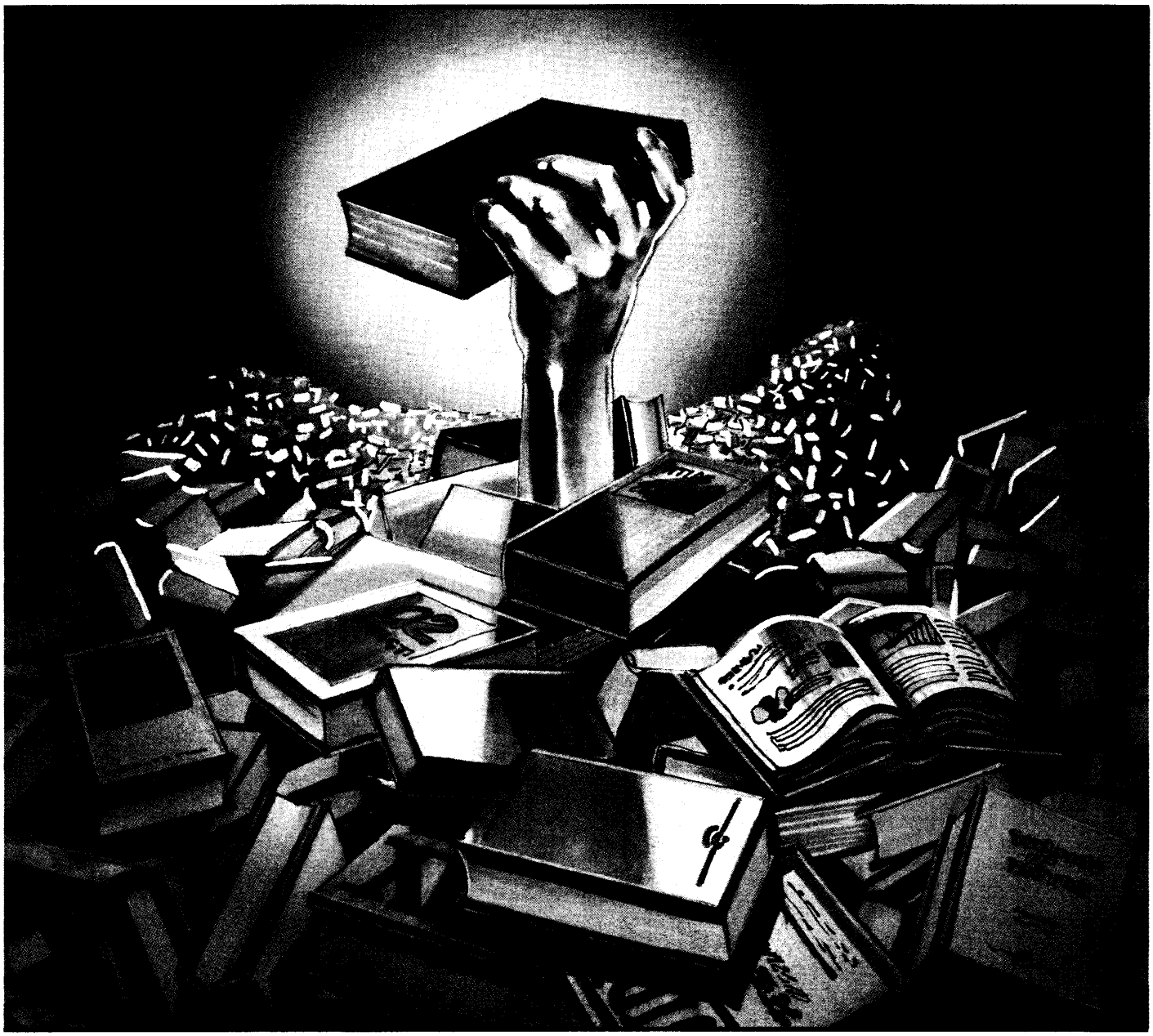
*Ask for details. Some manufacturers will not allow returns once disk seals are broken.

Dealers and Corporate Buyers — Call for special discounts and benefits!

1-800-445-7899
In NY: 914-332-4548
Customer Service: 914-332-0869
International Orders: 914-332-4548
Telex: 510-601-7602

Programmer's Paradise™
A Division of Hudson Technologies, Inc.
42 River Street, Tarrytown, NY 10591

Reader Service Number 68



Accessing dBASE III Plus Records From Turbo Pascal

Opening A World Of Data

Those of us who use dBASE have a love-hate relationship with it. What it gives us in brevity, it takes away in versatility. What it does for us with dot commands, it takes away in applications speed. So why not use dBASE III Plus for setting up databases, for indexing, and for short queries? Then use Pascal for the real applications.

But even if you don't plan on writing Pascal routines, you should read Michael's description of dBASE's files. It's not a bad piece of data.

dBASE III Plus provides a wide assortment of high level commands for accessing its relational databases. But these commands, despite their power, are often limiting. For instance, dBASE III doesn't support —

By Michael Greene
Max Software Consultants, Inc.
4101 Greenmount Avenue
Baltimore, MD 21218

- complex window-based user interfaces
- sophisticated scientific and business applications
- serial communications
- interactive graphics
- process control
- or software interfacing

These often require the power and flexibility of a more traditional language.

But dBASE III Plus has become a de facto standard in the PC-DOS marketplace, so it makes sense (if you're in the software business) to learn to live with it. By some estimates, there's more dBASE code lying around than any other language. Certainly, there's a vast store of data and software.

So How Do We Deal With dBASE?

One way is to use Turbo Pascal to access dBASE files.

Using Turbo, we can manipulate dBASE formatted data but still build user interfaces with pop-up and scrollable windows, dynamic fields, special cursor control, etc. We can graph, calculate, crunch numbers, and transfer dBASE data to other file formats.

Several commercial Turbo Pascal function libraries support this kind of development.

The purpose of this article is to explain how to access dBASE III files via Turbo Pascal. The flexibility and speed you'll gain with Pascal will more than justify the effort.

Let's begin by looking at how dBASE stores its data.

Data Storage

The three types of dBASE files are DBF, NDX, and DBT. Most of the information is stored in records in a DBF (data base file). The NDX (index) files contain sorted pointers to the records in a DBF file. DBT files hold Memo fields. These fields can exceed the 254 character maximum length for a DBF record.

DBF File Header

DBF files have a large heading (control block) that specifies many of the details about the file. To read a DBF file in Pascal, assign it as an untyped file and reset it with a block size of one.

```
Var
  DBF   :  File;
  NAME  :  String[20];
Begin
  Name := 'MY.DBF';   {File name}
  Assign (DBF, NAME);  {Get hndl}
  Reset (DBF,1);  {1 byte blocks}
  Seek (DBF,0); {Pnt to 1st byte}
End;
```

The second parameter in the Reset command indicates that we'll use a block size of one byte in BlockReads.

Reading The Header

Now we can read the information in the file header (byte by byte) and interpret it. The beginning of the file (first 12 bytes) always contains the same information in the same order. This makes it easy to define a record to read the heading into a structured variable. Set up a Type and then define a variable as follows —

```
Type
  DBFHeaderRec = Record
    HeadType: byte;  {Control byte}
    Year    : byte;
    Month   : byte;
    Day     : byte;
    RecordCount :LongInt;  {# recs}
    HeaderLength:Integer;{in bytes}
    RecordSize  :Integer;{in bytes}
  end;

Var
  DBFHeading    : DBFHeaderRec;
```

The first byte is a bit map and identifies the file as a DBF file. This byte also indicates whether or not a Memo file (DBT) file is associated with the DBF file.

When the value of HeadType is 03 hex, there's no memo file; when the value of HeadType is 83 hex, there is one. If HeadType AND 7F hex doesn't yield 3, it's not a dBASE III Plus file.

The next three bytes contain the date of last access. It's stored year, month, day. You can convert them to a string such as MM/DD/YY, or you can ignore the date. dBASE changes this date to the current system date whenever the DBF file is used. The year, of course, is in this century (add 1900).

The RecordCount is four bytes, equivalent to a long integer. RecordCount contains the number of data records currently stored in the file. HeaderLength is the number of bytes needed to store field attributes. You can also use HeaderLength as a pointer to the beginning of the data records.

Read the DBFHeading —

```
BlockRead (DBF, DBFHeading,
           Sizeof (DBFHeading));
```

Use the SizeOf function to insure that you only read as many bytes from the file as are allocated for your variable.

The next twenty bytes are reserved. There's no need to examine their contents. dBASE commonly fills these spaces with garbage.

The rest of the heading contains information specific to each field.

So far, getting information from the Control block has been fairly simple. But we don't know any good stuff yet, such as the field names or the number and types of fields.

Collecting Field Data

The fields are described in 32-byte records. They're terminated by a single byte having a value of 0D hex (carriage

return). When a file contains no data records, the OD hex is the last byte before the end-of-file marker. Be careful not to try to read past the end of the file when there are no data records!

The field definitions begin with the 33rd byte in the file. After reading DBFHeading, the file pointer will be positioned at byte 13. Move the file pointer to position 33 with a Seek. Notice that Seek numbers the bytes in a file starting at zero —

```
Seek (DBF,32);
  {Position pointer at 33rd byte}
```

Set up another type and define a variable for the field data. Read the fields one at a time in a loop until the carriage return is reached —

```
Type
  DBFFieldRec = Record
    FieldName:array[1..11] of char;
    FieldType:char;  {C,N,D,L or M}
    Spare1,
    Spare2:integer;{Reserved bytes}
    Width: byte;      {Field Length}
    Dec: byte;      {Decimal places}
    WorkSpace:array[1..14] of byte;
  end;

Var
  DBFField      : DBFFieldRec;
  i, FieldCount: integer;

Begin
  i := 1;
  DBFField.FieldName[1] := ' ';
  While (DBFField.FieldName[1]<>$0D)
  do Begin
    {Read 1 byte to avoid end of file
     error when no records exist}
    Blockread (DBF,
            DBFField.FieldName[1], 1);
    if (DBFField.FieldName[1] <> $0D)
    then Begin {Check for end of defs}
      {Read rest of the field def}
      Blockread
        (DBF, DBFField.FieldName[2],
         SizeOf(DBFField) - 1);
      ProcessField(DBFField,i);
      i := i + 1;
    end;
  end;
  FieldCount := i - 1; {1 extra read}
end;
```

The Procedure ProcessField turns the fields, one at a time, into arrays. They're defined from 1 to 128 because the maximum number of fields dBASE allows in one DBF file is 128. The Field names are the trickiest part.

They're read into an array of characters, but to make use of them we'd like them to be strings. FieldName is stored as an ASCII-Z string; that is, FieldName is terminated by a byte with value zero (CHR(0) or #0). That's why we need 11 bytes to store the 10 character FieldName (as read from the file).

```
Var
  Names: array[1..128] of string[10];
  Lengths:    array [1..128] of byte;
  Types:      array [1..128] of char;
  Decimals:   array [1..128] of byte;

Procedure ProcessField(F:DBFFieldRec;
                       i:Integer);
Var j : integer;
Begin
  With F do
  Begin
    Names[i] := '';{Init to null str}
    j := 1;       {Init char counter}
    While (J < 11) and
          (FieldName[j] <> #0) do
    Begin {Names are ASCII-Z strings}
      j := j + 1; {Increment counter}
      {Copy into the string}
      Names := Names + FieldName[j];
    end;
    Lengths [i] := FieldLength;
    Types [i] := FieldType;
    Decimals [i] := Dec;
  end; {With}
end;
```

Block Reading Field Data

Another way to get at the fields involves setting up a pointer and allocating just enough memory on the heap for the number of fields that must be read. This doesn't save memory, but we can read all the field information in one BlockRead. No loops!

Since each field definition occupies 32 bytes, the initial heading segment takes up 32 bytes plus a carriage return.

Calculate the number of fields from DBFHeading.HeaderLength—

```
#ofFlds :=
  ((DBFHeading.HeaderLength-1)/32 - 1)
```

Now get just enough 32-byte blocks on the heap for #ofFlds fields —

```
Type
  FieldsPtr: ^byte;

Var
  Fields: FieldsPtr;

Begin
  GetMem (Fields,
          ((DBFHeading.HeaderLength-1)
          / 32 - 1) * 32);
  BlockRead (DBF, Fields,
          ((DBFHeading.HeaderLength-1)
          / 32 - 1) * 32);
End;
```

This way, we read the entire block into memory. However, making use of it requires some sticky manipulation of pointers. One way to unravel the pointer mess is to write two functions.

The first moves the address of the pointer ahead 32 bytes at a time; the second puts the field information into a record structure —

```
Function MovePointer
          (Pointer:FieldsPtr)
          :FieldsPtr;
begin
  MovePointer := Ptr (Seg (Pointer^) +
                      Ofs (Pointer^) + 32);
end;

Function GetField
            (PointerToField:FieldsPtr):
             DBFFieldRec;
Var
  RetField: DBFFieldRec
            absolute PointerToField^;
Begin
  GetField := RetField;
end;
```

To make use of the fields on the heap, move them into the Type of array shown above. Use the following —

```
Var
  TempPointer : FieldsPtr;
Begin
  TempPointer := Fields;
  for i := 1 to
        DBFHeading.HeaderLength do
  Begin
    DBFField := GetField(TempPointer);
    ProcessField (DBFField, i);
    MovePointer (TempPointer);
  end;
end;
```

The method you choose to access the data is purely a matter of taste. Both will produce the same results.

Reading Records

Reading the records from the file is straightforward. The length of the heading (DBFHeading.HeaderLength) tells where the first record begins. Each record is stored as a series of bytes.

The first byte in any record is the deletion flag. If this byte is equal to an asterisk, the record is marked for deletion. The record occupies one more byte than the sum of the field widths. dBASE left-justifies character fields and fills them out with spaces.

Date fields are stored as 19YYMMDD so they take up eight bytes. Logical fields are one byte and contain either "T" or "Y" for true or "F" or "N" for false. Numeric fields are stored as character strings of numbers. To obtain their value, use the Pascal Val procedure.

Memo fields contain a ten-character number filled with leading zeros. The number is a pointer to the block in the DBT or MEMO file that contains the memo. More on memos later.

To read a record, set up a data record that conforms to the database in use. Say that a name and address database has the following definition —

Field	Field Name	Type	Width
1	FIRSTNAME	Char	15
2	LASTNAME	Char	20
3	ADDRESS	Char	30
4	CITY	Char	15
5	STATE	Char	2
6	ZIP	Char	10
7	PHONE	Char	15
** Total **			108

Set up the following record structure to read this database:

```
Type
  DataRec = Record
    DeleteFlag:char;
    FIRSTNAME: array [1..15] of char;
    LASTNAME:  array [1..20] of char;
    ADDRESS:   array [1..30] of char;
    CITY:      array [1..15] of char;
    STATE:     array [1.. 2] of char;
    ZIP:       array [1..10] of char;
    PHONE:     array [1..15] of char;
  end;

Var
  DataRecord:  DataRec;
  RecordNum:   LongInt;
```

Notice the extra byte taken up by DeleteFlag. If you add up the sizes of the fields in the definition, you'll see that the total is always one greater than the sum of the field sizes. Now the records can be read from the file into this structure.

To position the file pointer at the beginning of a record, use DBFHeading.HeaderLength as a base offset. Add the product of the desired record number minus one and the length of each record (DBFHeading.RecordSize). Then call the Seek procedure.

```
Seek (DBF, DBFHeading.HeaderLength +
      DBFHeading.RecordSize *
      (RecordNum - 1));
```

The file is now set up to read record number RecordNum.

```
BlockRead (DBF, DataRecord,
           DBFHeading.RecordSize);
```

The data for record number RecordNum can be loaded into DataRecord. At this point you'll probably want to check the first byte (the DeleteFlag) to see if the record is marked for deletion.

DBT Files

DBT files have a 512-byte control block. The only data stored in the control block of a DBT file is the number of the next available data block. The blocks are always 512 bytes long, and the end of the memo text is indicated by a control-Z (1A hex).

When a memo runs over the end of a block, the following block will not be referenced.

For example, say that a memo file has no memos in it. The control-Z will appear at byte 513. The next available block will be one. When a memo that is, say, 400 bytes long is written into the file starting at byte 513, the next available block (beginning at byte 1025) in the control block becomes a 2.

On the other hand, if the memo written to byte 513 is, say, 700 bytes long, the next available block in the control block becomes a 3; in which case, block 2 wouldn't be referenced.

Memo fields in DBF files are 10 characters long and contain the number of the DBT file block where the memo begins. Block numbers stored in memo fields of DBF files are right justified with leading zeros.

If you edit a memo, dBASE throws away the reference to the old block and puts the new memo in the next available block. Then it changes the block reference in the memo field of the DBF file.

This is a real waste of disk space if you edit memos very often, but you don't have to worry about squeezing a larger memo into a smaller space.

NDX Files

NDX files are the indexes into a dBASE III database. They're B+ trees. The leaves of the trees are index records consisting of a key and the DBF record number of the record containing that key. A key can be almost any function of one or more fields in the table.

It could be a customer's last name or a part number for example, or it could be a zip code together with the first three letters of a subscriber's last name. We define the key by a key expression which we store in the header of the index file.

We store each index in a separate NDX file. These provide Indexed Sequential Access Method (ISAM) to records stored in the associated DBF data file.

An index file, like a memo file, is built in blocks of 512 bytes with block numbers from zero to the total number of blocks.

The first block, block zero, is the header block. It contains the block number of the root of the tree, the number of blocks (also the next available block), the length of the key and also the total length of an index record (10 + key length), the number of records per block, the key type (0 for character, 1 for numeric or date) and the actual key expression which may be up to 100 characters long.

When searching for a key in a B+ tree, you go first to the root block, whose block number has been read from block zero, and look through the index records until you find a key that's greater than or equal to the key you seek.

If this block isn't a leaf, it has the block number of the next block to search. Repeat until you reach a leaf. The DBF record number where the key can be found resides in the leaf.

In a very small index (where the total number of records is less than the total number of records per block), the root may be the leaf.

In Sum

The DBF and DBT files are mostly straightforward in their presentation of data. And we can easily define data types that correspond to the positions and types of their data in Pascal.

We can assemble procedures to read, update, and write records using simple data types. We can then manipulate these dBASE records with procedures we design or buy: I/O routines, special sorting algorithms, graphics packages, etc. In short, we can bring a little more sanity into the dBASE world.

NDX files are also useful. They provide a fast access ISAM that we could apply to any type of information, not just data stored in dBASE databases.

Unfortunately, an in-depth discussion of NDX files, their layout, access methods, adding and deleting keys in B+ trees is beyond the scope of this article.

Try experimenting with the DBF and DBT files in Pascal, and then take a look at what dBASE puts into index files. Although the algorithm for searching a B+ tree is trivial, adding and removing keys is another story.

♦ ♦ ♦

Building A Database With C

Four File Handling Utilities Compared

The disadvantage of a dedicated database language such as dBASE is that it's dedicated. Try to include communications or special screen handling and you've got problems. Try to port to another system and you have real problems. But then writing in a more general language is a lot more work. Maybe...

There are two ways of approaching the design and development of a database management system (DBMS): with a dedicated database product (such as dBASE III Plus or Paradox) or with a traditional language (such as C or COBOL). This article concerns the latter, where a programmer uses C with a library to produce a DBMS.

Why use a language like C when dedicated database products (for which I'll use the abbreviation DDBP) are available? I've developed quite a few DBMSs in the last decade, and have built them using both conventional languages and DDBPs.

The established wisdom is that software development is faster when using a DDBP, since it can do much of your work. Languages like C require greater technical knowledge and greater effort. Like most conventional wisdom, there's some truth in it.

Advantages

With a DDBP, even a relative novice can create a database. Plus, experts work faster in a DDBP since they need fewer commands to get the database up and running.

Most DDBPs include a programming language (such as PAL for Paradox) which can be used to customize the entry and retrieval of information in the database. These languages usually use a more English-like syntax than a real programming language like C.

Disadvantages

Most DDBP programming languages are interpreted, making them relatively slow. While there are some DDBP compilers (such as Clipper for dBASE III Plus), the resulting applications are often slower and larger than the equivalent in C or Pascal.

And, there's portability. Don't assume that an application will remain on any one hardware or operating system environment. It's possible that a DBMS may need to be resident in multiple environments simultaneously. While some DDBPs are available for more than one system (Paradox being the best example), they're not available for all systems.

The number of development tools available for conventional languages far outstrips the number available for most DDBPs. Debuggers and make utilities are terrific time savers. I've spent many an hour debugging dBASE III Plus programs, wishing I'd had something like CodeView so I could step through the code.

The last advantage to using a conventional language is its flexibility. A DDBP is excellent for creating a DBMS, but is (in general) lousy at communications, word processing, and other tasks. Many DBMSs need to be integrated with other types of applications, and a language like C is just the ticket.

As you can see, I like C.

I'll talk about four add-on systems which can be used by C programmers when developing a DBMS. These are: Btrieve from Novell, db_Vista from Raima, ISAM and Btree from Softfocus, and CBTree from Peacock. The Softfocus and CBTree products are not MS-DOS specific.

This is not exactly a review. I'm not going to compare the four products head to head. There are many ways to handle database functions in C, which makes benchmarking nearly impossible. So, I'll talk about my experiences with each package, explaining what it can (and can't) do.

For testing, I used Microsoft's C version 5.10.

Btrieve

I wasn't able to get a review copy of this product before deadline, in spite of several calls to Novell. Heck, they didn't even call back! Fortunately, while doing some recent contract work, I was heavily exposed to Btrieve, and feel I can give it a fair shake.

Btrieve is a memory-resident product which uses anywhere from 25 to 90K of memory. Up to 64K of this memory (the data segment) can be shunted into expanded (LIM EMS) memory. The resident size depends upon command line options which specify such things as buffer size.

Btrieve supports a number of languages, including C, Fortran, COBOL, BASIC, and others. That's great. Unfortunately, I wasn't impressed by their code.

The manual is well-organized but poorly written. I've seen clearer mud. All of the examples in the main text are done in BASIC; other languages are relegated to appendices. Unfortunately, none of their C examples would even compile under Microsoft C 5.1. If these people wanted to give examples of poor C programming techniques, they've done a *wonderful* job.

Novell's technical support didn't thrill me, either. Actually, the support is through Softcraft — the company which created Btrieve before being bought out by Novell. In any case, the answers I received were vague, and their "Windows and C" expert didn't know how Windows worked.

Btrieve is fast, and the interface is very simple — one function call does it all. Just pass Btrieve's interface function an operation code (tells Btrieve what to do) and a set of pointer parameters.

By Scott Robert Ladd
P.O. Box 61425
Denver, CO 80206
(303) 322-7294

Since Btrieve is resident, it works well with Microsoft Windows (a somewhat picky environment when it comes to file I/O). Unfortunately, this complicates distribution. You have to include instructions to the user on how to load Btrieve.

Multi-user and network versions of Btrieve are available. Since I don't have a review copy, I was unable to check out its limits (file sizes, index sizes, etc.).

db_Vista

This is the most extensive package I've ever seen for C database development. It not only includes a sophisticated database function library and utilities, but also has an SQL-compatible query library. There's a new add-on which lets you revise a database without damaging its original data. The product comes precompiled for your C package (including Microsoft, Lattice, and Borland).

Databases can be modeled in several ways. Two of the better known and popular versions are the relational and network.

A relational database (the more common variety) is made up of multiple files linked by common fields. A network has owners and members, where one record owns one or more other records.

db_Vista supports the network model, with a one-to-many relationship between an owner and its members. The primary advantage of a network database is that it defines all of the data relationships up front.

A relational database makes its linkages via tables or indexes. Thus, a relational database is simpler to implement, while a network model is often faster.

Unlike the other three products, db_Vista is a complete database development system rivaling products like dBASE III Plus and Paradox.

The other systems (Btrieve, ISAM & BTree, and CBTree) are merely managers for indexed files. You can use them to create network or relational models but you do most of the programming. Db_Vista is locked into the network model, but it takes a lot of the burden.

A relational database makes its linkages via tables or indexes. Thus a relational database is simpler to implement, while a network model is often faster.

Creating A Database With db_Vista

To create a database with db_Vista, you first define a "schema," or organization. The schema defines which records exist, what these records contain, and how they interrelate.

You can then compile the schema to produce a database definition file and a C-language header file with data definitions (primarily record structures). A second utility creates and initializes the database. Finally, you write your C program.

There are more than 100 functions in the db_Vista library so it's not a simple system. While the documentation is amazingly good, it can take awhile to figure out exactly what does what with what, and why.

As I mentioned above, the documentation is well-written and filled with examples. Full source code is available, along with classes and a selection of phone support options.

The db_Query package is a useful addition. It provides SQL-like database queries which make a db_Vista database act similar to a relational database.

There is a royalty-free license for your application programs. There's another license if you plan to port the code to another operating system.

I couldn't find a limit for the number of records-per-file. Index keys can be no larger than 246 bytes.

ISAM And BTree

This package is a database file manager which uses b-tree file indexes. It comes on a single 360K diskette — a remarkable feat considering all the features.

BTree is a set of functions which create and maintain b-tree indexes. These indexes need not be related to files, BTree can be used to index just about anything.

ISAM is a high-level interface to BTree, designed to hide index details. ("ISAM" is an IBM creation standing for "Indexed Sequential Access Method.")

This package comes only with source-code. Softfocus claims that ISAM and BTree have been used with MS-DOS, VAX, and other systems, and that it takes less than an hour to convert the code for a new environment. This package has been used with nearly every MS-DOS compiler.

When I compiled this package, Microsoft C 5.1 came up with several errors. Most of these related to the memset() and memcpy() functions. (Softfocus includes source code for these.) I fixed the errors by using the versions of these functions from Microsoft's library. A significant number of warnings were issued by the compiler, but all of these related to the lack of function prototypes.

Softfocus has (obviously) avoided prototypes (an ANSI-standard feature) because many compilers still do not support them.

I really like this simple, straightforward, and friendly package. Several of the extras they included were written by third-parties who just wanted to help out. The documentation is well written, and the code is commented in a clear, conversational style.

You can use up to 16.7 million records in a data file or index; keys and records have no defined maximum (or, in other words, can be as large as your compiler's memory model allows).

CBTree

This package has a lot in common with Softfocus' products. There are three disks in the package, two contain source code, the other contains compiled versions of some utilities.

The manual is 84 pages, housed in a thick three-ring binder (there's room for future expansion). The print quality is poor in spots; it looks like the pages were printed on a laser printer and then photocopied.

CBTree is a b-tree indexing system with 18 functions and 6 system utilities. Compile procedures and batch files for different compilers are located in special subdirectories. The Amiga and XENIX are supported — I'm sure that with a little work, CBTree would work on any operating system with a C compiler.

Once again, things didn't compile smoothly. In the case of CBTree, the problem was with a lack of function prototypes. No standard headers are included in any of the modules, leaving standard library functions (like strncpy() and printf()) without prototypes.

One function, cbtree(), does most of the work. The programmer passes cbtree() an operation code from which the function determines the meaning of the other parameters.

CBTree is the simplest of the systems, but it leaves the most work for the programmer. It's easy to customize, relatively fast, and has good documentation. Its library and header files take about 45K of disk space.

The programmer is responsible for the organization of data and how it is interrelated (just as with ISAM and Btree). I could not find a table of system limitations (e.g., maximum records-per-database).

Conclusions

I wish I had more space in which to cover these products. Their developers obviously spent a lot of time fine-tuning them. All have strengths and weaknesses. Which is the best?

I wouldn't consider using Btrieve unless you run under Microsoft Windows, or need to work with more than one language.

db_Vista (and its add-ons) are very complete but the cost (nearly $1,800 for the complete set) is a big drawback. I highly recommend this product to those of you working with large, complex databases, especially where the utilities and maintenance facilities would be important.

ISAM and Btree are in the same class as CBTree: b-tree file indexers distributed in C so they can be ported to different environments. They are inexpensive and well documented. I like ISAM and BTree better because this package is smaller, cheaper, and more flexible than CBTree.

In the long run, choosing one of these packages depends on your pocketbook, how much work you want to do, and the type of database you're creating.

Products Reviewed

Btrieve 4.11b ($245)
Novell, Inc.
122 East 1700 South
Provo, Utah 84601
(801) 379-5900

db_Vista III v3.00 ($595)
db_Query v2.00 ($595)
db_Revise v1.00 ($595)
Raima Corporation
3055 112th Avenue N.E.
Bellevue, WA 98004
(206) 828-4636

BTree v2.6 ($75)
ISAM v2.6 ($40)
Softfocus
1343 Stanbury Drive
Oakville, Ontario
Canada L6L 2J5
(416) 825-0903

CBTree v2.31 ($159)
Peacock Systems, Inc.
2108-C Gallows Road
Vienna, VA 22180
(703) 847-1743

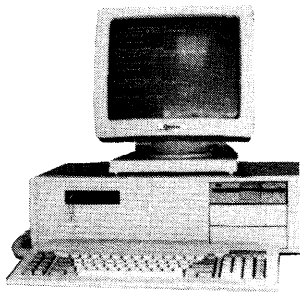
♦ ♦ ♦

Stuff Dreams are made of...

The Dream-286

AT 6-10 MHz 0 wait state motherboard
1 Megabyte of on board RAM
1 5.25 " TEAC 1.2M floppy drive
40 Mb Seagate ST-251 hard drive
Hard/floppy disk controller card
12" amber Samsung monitor (tilt/swivel)
Hercules compatible mono graphics card
AT style keyboard
AT case (UL and FCC approved)

Complete! $1495



The Dream-386

Quality 80386 based motherboard
1 Megabyte 80 ns on board RAM
2 Parallel ports and 2 serial ports
640 x 480 on board EGA/VGA card
1.2 Megabyte 5.25" floppy drive
Western Digital WA-2 hard/floppy controller card
Professional enhanced (101 key) keyboard
Case (UL and FCC approved; reset switch, power and turbo LEDs, keyboard lock

Complete! $2295

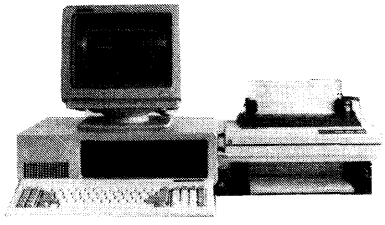
Novell Networks

Taylored to your exact needs. Call for more information.

The Dream-88

XT 10 MHz Turbo motherboard
640 K of on board RAM
1 5.25 " 360K Fujitsu floppy drive
30 Mb Seagate ST-238 Hard drive
Floppy controller card (controls 2)
Hard disk controller card
12" amber Samsung monitor (tilt/swivel)
Hercules compatible mono graphics card
AT style keyboard
XT slide case (UL and FCC approved)

Complete! $895



Hard Drives

Seagate 20 MB ST-225	$269
Seagate 30 MB ST-238	$299
Seagate 40 MB ST-251	$395
Micropolis 71MB Hard Disk	$695
Seagate 80 MB ST-4096	$750

Floppy Drives

Fujitsu 360K 5.25"	$ 79
Fujitsu 1.2 MB 5.25"	$ 99
Fujitsu 720K 3.50"	$115
Fujitsu 1.44 MB 3.50"	$145

Floppy drives come with mounts

Citizen Printers

180-D (180 cps 9 pin)	$189
MSP-40 (260 cps 9 pin)	$339
MSP-45 (wide carriage)	$489
Tribute 124 (200 cps 24 pin)	$545
Tribute 224 (wide carriage)	$689
Premiere 35 (daisy wheel)	$499

Communications

1200 Baud internal	$ 95
2400 Baud internal	$185
1200 Baud external (w/ cable)	$ 99
2400 Baud external (w/ cable)	$235
PC FAX (card & software)	$415

Accessories

PC Mouse w/ paint	$115
M-8 Logitech mouse	$ 99
Kraft IBM/Apple Joystick	$ 25

Laptops

Sharp 4051	$ 795
NEC EL	$1685
NEC EL HD	$2495

Software

Borland Turbo C	$ 79
Borland Turbo Pascal	$ 69
Borland Turbo Basic	$ 69
Borland Quattro	$139
Borland Sidekick	$ 49
Microsoft Works	$149
Microsoft Windows v2.03	$ 69
Microsoft DOS 3.21	$ 69
Microsoft DOS 3.30	$ 99
Norton Utilities (Advanced)	$ 89
Norton Commander	$ 45
Stella Business Graphics II	$ 99
Peachtree Accounting System	$179

DreamTech

5175 Moorpark Avenue
San Jose, CA 95129
Open Monday - Saturday
10 AM to 7 PM
Phone: (408) 996 - 2373

MasterCard VISA

All systems carry a full 1 year warranty (original system configuration only). Prices are subject to change without notice. All orders are shipped UPS FOB San Jose unless otherwise specified at time of order. California residents add 7% sales tax. All registered trademarks are recognized. Dream-88, Dream-286, and Dream-386 are trademarks of DreamTech Computers.

Reader Service Number 16

Selecting A Database Compiler

A Developer's Look At The dBASE Environment

Pat called right after he read we were looking for hot tips on database packages. He wanted to look at the dBASE compiler scene from a developer's point of view. Great. That's where a lot of us are messing around right now. It turned out he had some definite ideas about what worked and what didn't. I think you'll enjoy this piece, whether you agree or not.

Right after I sent the first version of this article to Micro C, Dave called me. "You say you're writing a review of dBASE III, Foxbase, Clipper, and Quicksilver, but the article is very heavily weighted toward Clipper and Quicksilver," he noted. I explained that my prejudices had been hard won: only after using all four packages had I picked my favorite tools.

However, Dave asked me to go back and take another look at all the packages. That second look has changed my mind about a couple of things.

First: Within a few hours I'd found that the support scene had changed.

Ashton-Tate (dBASE) was impossible to reach by phone (at least the people at Ashton-Tate weren't talking). My recent contacts with Nantucket (Clipper) were nearly as fruitless. Gone are the days of being able to talk to a Brian Russell or a Ray Love (whom all of the "old time" Clipper developers know).

Wordtech, on the other hand, was fast, courteous, and yes, even helpful. My questions were answered and my calls returned. And the new kid on the block, Fox Software, was wonderful. I needed their 2.0 Foxbase+ Multiuser to test. They "red labeled" it to me. Then they called me back and answered all my questions in one call. As Andy Rooney says, "I like that."

If support is an issue with you, and it's a big one for me, take a look at Wordtech and Fox Software.

Second, we've been using Clipper (version Summer '87) for our quick and dirty projects around the office. At Dave's behest, I got the latest Foxbase. After trying it, I switched. Foxbase+ compiles faster and runs faster in most cases than Clipper. And, Foxbase includes most of Clipper's commands.

So with all that in mind, let's take a detailed look at the four packages.

If support is an issue with you, take a look at Wordtech and Fox Software.

Foxbase+

In general: Foxbase is an interpreter and an incremental compiler. Generally, it runs at least six times faster than dBASE III, and in some cases it beats the true compilers (Clipper and Quicksilver).

As it comes, it runs in demo mode only. As long as you don't activate the normal mode, you can return the package for a refund. If you decide to keep it, then you open the envelope to get the activation number. Foxbase is not copy protected.

The code developed with Foxbase must run under the development package or under a run-time version of the package. To distribute your code you must purchase the run-time module ($500 single user, $700 multiuser). This is a one-time fee, and you can make as many copies of the run-time module as you need.

Strengths: Foxbase is the only package which works on PCs, Macs, UNIX, and XENIX. Quite a feature. Get their 386 package and you can generate code for 386s, 286s, and 8088s.

If you ask a developer about application speed, you'll get: "If it's fast enough for the client, it's fast enough for me." But behind closed doors all developers take pride in raw run-time speed and Foxbase is usually the winner.

Weaknesses: Foxbase+ is not a true compiler, so distribution can be a problem. Anybody who has used a run-time package knows that this is a sloppy way to distribute a commercial product. One of our applications requires five disks for Foxbase versus two disks when we use a compiler.

Foxbase+ will not incorporate assembly language or C routines, and it doesn't give you access to the computer's I/O ports. It uses a minimum of 360K RAM.

Foxbase shares some of dBASE's problems: poor network implementation, little support for import/export, and it does poorly on non-clone machines like Wang, DEC, etc. Also, it lacks the little extras like windows, graphs, and third-party software.

The manual is weak, containing few examples.

dBASE IV (and III Plus)

In General: dBASE IV has yet to be released, but the gossip is already running rampant. The new dBASE will include some of Nantucket's additions such as UDF, arrays, and more memory variables. It'll have a more extensive application development package, and SQL.

Also, Ashton-Tate is claiming speed — not to be confused with real speed like Foxbase+, but at least 2.5 times faster than dBASE III Plus.

Strengths: Ashton-Tate will probably be around for a while, guaranteeing continued product enhancements. (Anyone remember C English?) That means an OS/2 version will be along someday.

By Patrick McGuire
New Horizons Computers
800 Grand Ave. #AG-7
Carlsbad, CA 92008
(619) 438-0242

Also, SQL is definitely in vogue and dBASE supports SQL. That means it's not necessary to use dBASE commands for retrievals and updates, since one SQL command can often replace a screenful of dBASE.

Weaknesses: Ashton-Tate is not famous for its user support. In conjunction with poor support, its new releases are usually riddled with bugs. I still remember a copy of dBASE III which enjoyed eating its datafiles. Six months later they had a fix but didn't bother to warn customers or send the fix.

The dBASE IV manual comes in three large volumes. (Three volumes of poor coding.) Figure on spending ten minutes to find anything.

A handful of lesser problems include:

As with Foxbase+, dBASE IV is a memory hog — a problem when multitasking.

dBASE IV requires a run-time package.

dBASE IV doesn't include solid networking, windowing, statistical functions, or import/export facilities. There's no real way to edit memo fields, no real linking to C, and, of course, it's still slow.

Clipper

In general: Clipper is the pioneer of the true dBASE compilers. Clipper produces Intel OBJ code that can be linked by most linkers (DOS Link 3.0, Plink+, etc). This produces a stand alone EXE file, no run-time needed. Clipper also runs under UNIX.

Clipper's summer '87 release has set the pace for others to follow, including Ashton-Tate. Most packages include what Clipper pioneered: UDFs, arrays, valid, and save screen. Nantucket includes a word processor for editing memo fields. It also includes a DBU for accessing files without dBASE. Source for the DBU is included.

Strengths: The summer '87 release included three important changes: A sizable, easy to read manual including excellent examples, a tutorial, and a good cross reference.

The compiler is very fast (compiles lines in increments of 100) and there's excellent support for linking in C routines. Clipper allows seven variables to be passed to the C language procedure, instead of the usual one. Also, the parameters passed are passed by reference (pointers in C). So, for example, if one wanted to use a C math function, the scenario would run like so:

In Clipper:

```
Function dSIN
parameter nbmr
x=nmbr
call cSIN with x
return x
```

In C:

```
SIN(xx)
double *xx;
{
   double sin();
   *xx=sin(*xx)
}
```

This produces the sin of x.

A non-fixed length variable is a great idea (the memo field), but only Clipper knows how to handle it.

dBASE IV and Foxbase are memory hogs. Clipper would be (one of our applications runs 680K) too, except that it uses overlays. For example, you could have a a main menu that has an AR (accounts receivable) overlay, AP (accounts payable) overlay, and GL (general ledger) overlay. The GL might have sub overlays like POST (posting), INC (income statement), and BAL (balance sheet). We've shoehorned this into 320K.

Weaknesses: Clipper has no matched interpreter. Prototyping is important and an interpreter saves lots of time during the testing and debugging phases. We've tried using dBASE III Plus as the interpreter, but then we can't use UDFs, arrays, etc.

Nantucket's networking is weak: no way to know who has a record, no way of knowing if a record has been updated while in memory, and no way to save a physically locked and crashed file.

Finally, support is a problem. Traditionally, Nantucket has provided good support, even promoting user groups, but if my last phone calls are any indication, I would say that Nantucket is imitating Ashton-Tate.

Quicksilver

In general: Quicksilver, with its diamond release, has really polished up the rough spots. If you're writing network code or windows, look no further: Quicksilver does them best. Plus, Quicksilver offers my favorite features — support, and a matched interpreter. This is a complete development system.

Strengths: Quicksilver and its sister interpreter, DBXL, really work well together, speeding the cycle of test, correct, test, correct. The user interface is also very important and, in my mind, the only way to handle a good user interface is with windows. Quicksilver supports layered windows for data entry, function key selections, and more. The rumor is that Ashton-Tate may try to copy this.

Quicksilver does graphs with just a few lines of code. You can place the output in a file or create an image for GEM or Ventura Publisher. You get your choice of pie chart, bar chart, step graph, line graph, scatter plot, or regression line.

And, in networking, Quicksilver has all the usual functions like Rlock(), Flock(), Set Exclusive on/off, and on neterror(). But it also contains some unique commands. For example, autolock. Use autolock and Quicksilver will, in its own way, take care of all the locking tasks. Not pretty, but an especially quick

way to convert single user code to multi-user code.

Other commands include senserange() and whohasit(). Quicksilver can also send a task to another work station, thus letting an unused machine do the chore.

Quicksilver has three features I especially like: automem, export, and Quicksilver's access to the computer's ports.

Automem replaces the following:

```
** without automem
use test
store field1 to mfield1
store field2 to mfield2
store field3 to mfield3
store field4 to mfield4
store field5 to mfield5
@ 1,10 get mfield1
@ 2,10 get mfield2
@ 3,10 get mfield3
@ 4,10 get mfield4
@ 5,10 get mfield5
read
replace field1 with mfield1
replace field1 with mfield1
replace field1 with mfield1
replace field1 with mfield1
replace field1 with mfield1
**17 lines
```

with:

```
use test
store automem
@ 1,10 get m->field1
@ 2,10 get m->field2
@ 3,10 get m->field3
@ 4,10 get m->field4
@ 5,10 get m->field5
read
replace automem
** 9 lines
```

And, there's little need to write assembler. Quicksilver primitives include: dosint(), in(), out(), and bitset().

Weaknesses: Quicksilver has some problems. You can't easily edit memo fields. And its method for creating overlays is so confusing that most developers don't bother.

Quicksilver creates a LNK file, but only after the developer has created a template. Then Quicksilver spits out files like menu001, menu002, menu003 that have very little correlation to the actual compiled files. This makes modifying the LNK file almost impossible.

And finally, Quicksilver allows for only a limited number of functions and procedures, and these must be placed in a single file.

Which Do We Use?

Most of our code is written in Quicksilver because of its features and its support. However, if we need a front end for C language modules, we use Clipper. If we need a "down and dirty" program that is continually being modified and enhanced, we use Foxbase+. And, if we ever write an SQL program, we'll probably use dBASE IV.

(Note: the code for the tables in Figure 1 is on the Micro C RBBS and on the issue #43 disk for $6.)

Clipper, Summer '87
Nantucket Corp.
12555 W. Jefferson Blvd., #300
Los Angeles, CA 90066
Price: $697

Quicksilver, Diamond release 1.2
Wordtech Systems, Inc.
P.O. Box 1747
Orinda, CA 94563
Price: $599

Foxbase+
Fox Software Inc.
118 W. South Boundary
Perrysburg, OH 43551
Price: $395 single/$595 multi/$595 80386
Run-time: $500 single/$700 multi

DBASE III Plus (IV)
Ashton-Tate Corp.
20101 Hamilton Ave.
Torrance, CA 90502
Price: $695 III+ /$795 IV
$1,295 developer

♦ ♦ ♦

Figure 1 — Compilers Compared

1000 Iteration Loop and Screen Write

PRODUCT	COMPILE	LINK	RUN	RUN SIZE
dBASE III+	N/A	N/A	17	384K
FOXBASE+	2	N/A	1	384K
CLIPPER	1	45	2	148K
QUICKSILVER(D)	2	20	7	256K
QUICKSILVER	5	11	4	134K

TEST 2 - APPEND 1000 RECORD

PRODUCT	COMPILE	LINK	RUN	RUN SIZE
dBASE III+	N/A	N/A	62	384K
FOXBASE+	1	N/A	8	384K
CLIPPER	2	45	5	148K
QUICKSILVER(D)	3	20	10	256K
QUICKSILVER	7	12	8	135K

TEST 3 - REPLACE NUMERIC FIELD IN 1000 RECORDS

PRODUCT	COMPILE	LINK	RUN	RUN SIZE
dBASE III+	N/A	N/A	41	384K
FOXBASE+	1	N/A	16	384K
CLIPPER	2	48	5	148K
QUICKSILVER(D)	3	19	15	256K
QUICKSILVER	7	13	12	134K

TEST 4 - INDEX 1000 RECORD FILE ON NUMBERIC FIELD

PRODUCT	COMPILE	LINK	RUN	RUN SIZE
dBASE III+	N/A	N/A	45	384K
FOXBASE+	1	N/A	3	384K
CLIPPER	1	45	4	166K
QUICKSILVER(D)	3	21	8	256K
QUICKSILVER	5	12	6	155K

Figure 1 *Continued*

Charts:
chart of the features of the compilers:

FEATURE	CLIPPER	FOXBASE	QUICKSILVER	dBASE III/IV
CODE SPEED		x		
COMPILE SPEED	x			
DEBUGGING			x	
C/ASM	x		x	
OVERLAYS	x		x	
THIRD PARTY	x		x	
UDF'S	x	x	x	
PROCS	x	x	x	
ARRAYS	x	x	x	
WINDOWS			x	
MENUS	x	x		
MEMOS	x			
MULTI TASKING			x	
DOS ENVIRONMENT			x	
IMPORT/EXPORT			x	
SQL				x
MAIN/MINI FRAME				x
STATISTICS			x	
SECURITY				x
MANUAL			x	
USER SUPPORT		x	x	
TECH SUPPORT		x	x	

chart of which compiler for which job

FEATURE	CLIPPER	FOXBASE	QUICKSILVER	dBASE III/IV
NETWORK			x	
MAC		x		
MAINFRAME				x
OVERLAYS	x			
MEMORY			x	
GRAPHICS			x	
WINDOWS			x	
MATCHED INTERP		x	x	
SPEED	x	x		
LINK TO C	x			

Fast Fractals:

Programming The 386 Under MS-DOS

This is one of our trick articles. The fractals are really just a sneaky way to get your attention. After all, how many of you would read an article about speeding up 386 software by a factor of 100? (On second thought...)

As excitement over the 386 dies down, we're left with an uncomfortable feeling. A 32-bit operating system — one that will provide memory management over huge address spaces — is a long way off, and when it comes, it will cost a lot. But don't despair; the 386 has many powerful features that can be used under a 16-bit operating system, features that can give your programs a real kick.

This article shows how to use 32-bit instructions on the 386 under MS-DOS. As an example, we'll turbocharge the familiar Mandelbrot "zoom." Our 386 version will use fixed point math to obtain the equivalent of one MFLOPS (million floating point operations per second), without a math coprocessor. Consequently, the program will take minutes, not the usual hours or days, to produce fractal diagrams like Figure 1.

To show how simple the process is, the program will be written for a 16-bit C compiler/assembler — that is, a compiler/assembler that doesn't understand 386 mnemonics. I begin with an overview of 386 "modes," then move into a brief discussion of the Mandelbrot calculation and fixed point math.

To minimize repetition, I'll assume you've read Larry Fogg's article on the Mandelbrot set (*Micro C* issue #39 Jan./Feb. 1988) and Earl Hinrich's article on fixed point (*Micro C* issue #41 May/June 1988). Another useful reference is *The 80386/387 Architecture* by S. Morse, E. Isaacson and D. Albert.

Modes

The 386 has three modes — real, protected, and virtual 8086 (V8086).

Real mode is the default, or how the processor wakes up after you turn on the computer. Segments in real mode are 16 bits long, and there are no memory protection schemes.

When the 386 gets nudged into protected mode, the segments can be 32 bits long, allowing huge address spaces, and all sorts of multitasking and memory protection features become available.

V8086 mode is very handy for running old 8086 applications within a multitasking operating system; programs under V8086 behave very much like real mode programs.

32-Bit Instructions

In all modes, it's possible to access 32-bit registers and use 32-bit instructions; the 32-bit extensions of the familiar 8086 registers are called eax, ebx, ecx, edx, edi, esi, ebp and esp. The lower 16 bits of these registers can still be accessed as ax, bx, etc. MS-DOS is designed for real mode, so that's the mode we'll use.

By default, instructions in real mode operate on 8 or 16-bit quantities. To use 32-bit operands, an instruction must be prefixed with an override byte — 66H for register operations, and 67H for addressing.

If you have a 386 assembler, you won't have to worry about the override bytes. You can freely mix 16-bit and 32-bit instructions in the same assembly language program, and the assembler will automatically insert the overrides in the object code.

If you don't have a 386 assembler, you can still use the 32-bit instructions by inserting "DB 66H" or "DB 66H, 67H" before a 16-bit instruction (normally, DB — define byte — is used in the data segment, but most assemblers also allow DBs in code). If there is no corresponding 16-bit instruction, you can put the entire byte encoding for the operation after a DB. For example, to code the 386 instructions —

To use 32-bit operands, an instruction must be prefixed with an override byte — 66H for register operations, and 67H for addressing.

```
add  eax,ebx
shld edx,eax,16
```

With a non-386 assembler, you could write:

```
DB 66H
add ax,bx       ;add eax,ebx

DB 66H, 0FH, 0A4H, 0C2H, 10H
                ;shld edx,eax,16
```

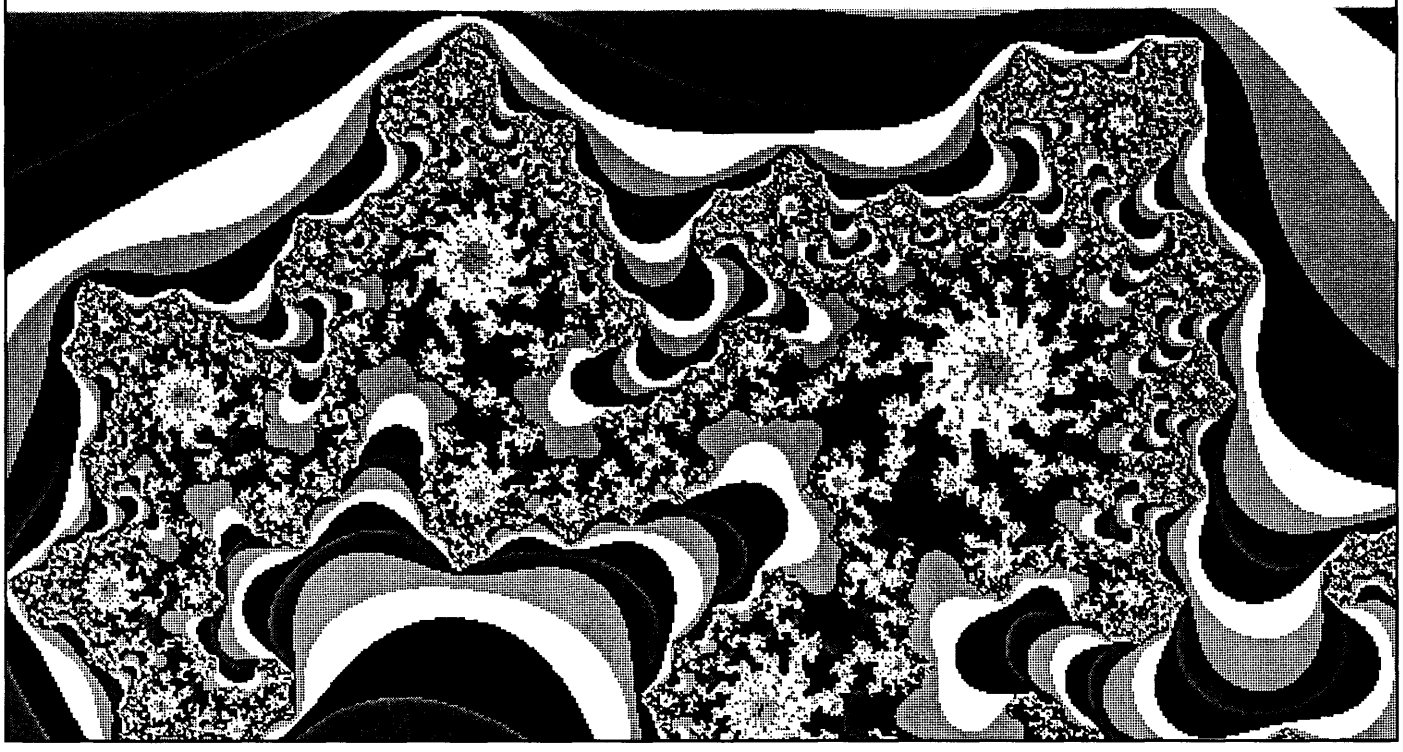
The first instruction is easy, since the older 80x86 processors have a 16-bit "add" that corresponds exactly to the 32-bit add; however, we have to replace the "shld" with a byte encoding, since the earlier processors had no double shift instructions. Byte encodings can be determined from any good 386 programming guide.

Defined Bytes

There are good reasons for using the

H. W. Stockman
5308 Noreen Drive N.E.
Albuquerque, NM 87111

Figure 1— p = -0.717 to -0.705 q = -0.3135 to -0.306



"DB" approach, even if you have a 386 assembler. For example, it is often desirable to call an assembly language function from a high level language.

Many compilers require that the function be assembled with a particular version of Microsoft's MASM. Unfortunately, MASM puts a peculiar header on functions assembled specifically for 386 real mode, even if the function contains nothing but 8086 instructions. The header is enough to confuse the heck out of non-Microsoft linkers, like the linker for Turbo C.

The DB approach circumvents the header problem, since you never have to tell the assembler that the module contains 386 code. Furthermore, many C compilers come with their own assemblers; typically, these assemblers can't understand 386 mnemonics, so there is no alternative to the DB method. I've taken the DB approach in this article because it is the most general, and because my favorite compiler is DeSmet C/asm88, which doesn't understand 386 mnemonics.

Mandelbrot Calculations

The core of the Mandelbrot calculation is the series —

$$z_{n+1} = z_n^2 + c,$$

where Z and C are complex numbers, and $Z_0 = 0$. In terms of real and imaginary components, we define $Z = X + iY$ and $C = P + iQ$. It can be shown that if $Z_n^2 \equiv X_n^2 + Y_n^2$ exceeds 4, the series will diverge.

Mandelbrot maps, such as Figure 1, show the divergence behavior of the series, as a function of P (horizontal axis) and Q (vertical axis). At each (P,Q) point in the map, we calculate the series until n hits an upper limit nmax (typically 100 to 1000), or until $Z_n^2 > 4$. The color of each point is keyed to the value of n when the calculation stopped.

Figure 1 contains 640 by 400 pixels, or 256,000 different P,Q combinations. Each pixel required an average of about 100 iterations before divergence, and each iteration required the equivalent of 13 or 14 floating point operations. Thus, Figure 1 took the equivalent of 300 to 400 million floating point operations; that's why Mandelbrot diagrams are said to be "calculation-intensive"!

Fixed Point Math

One way to speed up Mandelbrot calculations — especially on computers that don't have floating point hardware (e.g., an 80387 or 80287 math coprocessor) — is to substitute fixed point for floating

point math. This idea isn't new; see, for example, the article by H. Katz in *Dr. Dobb's Journal*, Nov. 1986. However, the 386 has instructions that make fixed point math very easy and very fast.

Fixed point numbers have an integer and a fractional portion, separated by a conceptual binary point. In our program, we'll store fixed point numbers as 32-bit integers, with the integer portion in the upper 8 bits, and the fractional portion in the lower 24 bits; thus the binary point is between bits 23 and 24.

With this system, we can represent positive or negative numbers with absolute values between about $6*10^{-8}$ and 127.9999999. That range is fine for Mandelbrot diagrams; X and Y will never get too big, because the calculation will stop if either X_n^2 or Y_n^2 exceeds 4. And it turns out that regions of the diagram with both P and Q < 0.25 are pretty uninteresting.

Fixed point numbers are added, subtracted, and compared just like ordinary 32-bit integers, but multiplication is slightly more complicated. To calculate the fixed point product of X and Y, we might try the C statement —

```
PROD = (X*Y)/16777216;
```

that is, the integer product X*Y must be divided by 2^{24} = 16777216 to get the correct fixed point product PROD. Of course, there is a problem with this C code; the product X*Y could be 64 bits long, which would cause overflow in most *compiled* code.

The problem disappears in assembly language, since most 32-bit processors have no trouble multiplying 32-bit numbers to form a 64-bit product. For example, suppose we have two fixed point numbers stored in the 386 registers eax and ebx; to multiply these numbers, and place the fixed point product in edx, requires only two instructions:

```
imul ebx ;eax is implicit destination
shld edx,eax,8 ;result now in edx
```

The imul places the 64-bit product in the register pair edx:eax (the high bits in edx), and the double shift instruction adjusts the binary point (we shift left by 8 bits, instead of right by 24 bits, because we want the product to end up in edx).

Some programmers prefer fixed point with 16-bit integer and 16-bit fractional portions. However, I've seen Mandelbrot calculations done with the 16/16 format, and the inaccuracies were pretty obvious, even at modest magnifications.

Figure 2 — Typical Floating Point Mandelbrot

```
/* Floating point version of mandel() for DeSmet C.                        */
/* ncol and nrow are number of columns and rows that can be displayed by */
/* chosen graphics adapter/mode; e.g., (ncol,nrow) = (720,348) for herc, */
/* (640,350) for EGA; endcolor is one less than max # colors the graphic */
/* adapter/mode can display; e.g., endcolor is 1 for hercules, 15 for    */
/* most EGAs; ptptr is pointer to pixel plotting function. nmax is the   */
/* maximum number of iterations we will allow for Mandelbrot series.     */
/* DeSmet function csts() reads keyboard buffer, doesn't wait if no key  */
/* was pressed. similar to Turbo C combination of kbhit() and getch()    */

mandel(Pmin, Pmax, Qmin, Qmax, nmax, ncol, nrow, ptptr, endcolor, switchpt)
double    Pmin, Pmax, Qmin, Qmax;
int       switchpt,nmax,ncol,nrow,endcolor;
void      (*ptptr)();
{
     int       color,row,col,n,transform();
     double    P,Q,dP,dQ,x,y,ytemp,modulus_sqrd;

     dP = Pmax-Pmin;
     dQ = Qmax-Qmin;

     for(col=0; col<ncol; ++col){
          for(row=0; row<nrow; ++row){
               P = Pmin + dP*col/(ncol-1);
               Q = Qmin + dQ*row/(nrow-1);
               n = 0;
               x = y = modulus_sqrd = 0.0;
               while(modulus_sqrd <= 4.0 && n < nmax){
                    ytemp = x*y;
                    x = x*x - y*y + P;
                    y = ytemp + ytemp + Q;
                    modulus_sqrd = x*x + y*y;
                    n++;
                    }
               /* transform the stopping iteration n to a color */
               color = transform(n,switchpt) & endcolor;
               (*ptptr)(col,row,color);
               }
          /* if a 'q' is pressed at keyboard, abort the funtion */
          if(csts()=='q') return('q');
          }
}

END OF LISTING
```

Figure 3 — Fixed Point Version of Mandelbrot

```
/* Fixed point version of mandel() */

mandel(Pmin, Pmax, Qmin, Qmax, nmax, ncol, nrow,
       ptptr, endcolor, switchpt)
double    Pmin, Pmax, Qmin, Qmax;
int       switchpt,nmax,ncol,nrow,endcolor;
void      (*ptptr)();/* ptptr is pointer to pixel plotting function*/
{
     int       color,row,col,n,mandwhile(),transform();
     long      P,Q,P0,Q0,dP,dQ;
     long      muldiv();

     dP = (Pmax-Pmin)*16777216 + 0.5;  /* 16777216 = 2 to the 24th */
     dQ = (Qmax-Qmin)*16777216 + 0.5;
     P0 = Pmin*16777216 + 0.5;
     Q0 = Qmin*16777216 + 0.5;

     for(col=0; col<ncol; ++col){
          for(row=0; row<nrow; ++row){
               P = P0 + muldiv(dP,col,ncol-1);
               Q = Q0 + muldiv(dQ,row,nrow-1);
               n = mandwhile(P,Q,nmax) + 1;
               /* +1 above depends on how 1st iteration is defined */
               color = transform(n,switchpt) & endcolor;
               (*ptptr)(col,row,color);
               }
          if(csts()=='q') return('q');
          }
}

END OF LISTING
```

KOALA COMPUTERS, INC.

213-316-5866
9 to 6 PST M - Sat

30-Day Money Back Guarantee
Software sales are final except replacement of defective media. Merchandise must be undamaged for full refund.

11½"x 14⅞" Greenbar Paper with Carbon Paper - 45 lb. $15

2 part 1500 sets, 4 part 700 sets. Two holes punched for binders. Case approximately 45 pounds. Send self-addressed stamped envelope for free sample.

SUPERIOR XT* STYLE KEYBOARD for the TOUCH TYPIST - $25

Replacement XT* style keyboard with LED in Num Lock and Cap Lock keys. Manufactured by **CHERRY**. Complete with coil cord and plug ready to slip into your case. Our service department uses these when possible instead of repairing the old one as most people appreciate the nice feel of these **made in USA** CHERRY keyboards. 2 lb.

MONOCHROME DISPLAY ADAPTER with PARALLEL PORT - $35

Japanese mfg. board. This is the MDA that can be plugged in with a CGA (Color Graphics Adapter) at the same time. 2 lb.

384K EXPANSION WITH IO and RAM - $139

Parallel, serial, game, clock calendar, and 384K of memory installed and tested. **$139 3 lb.**

NO SCREWDRIVER REQUIRED PARALLEL PRINTER CABLE - $10 - $16 - $20

Not the cheap ones but the DB25 with the knurled fastener. **6 foot** $10 one lb., **10 foot** $16 two lb., **15 foot** $20 3 lb.

The COMPLETE DBase* SYSTEM for the SMALL BUSINESS - $4

DBase* command code and operational instructions are contained in this excellent 335 page 8½"x 11" book DBase APPLICATIONS in BUSINESS 2 lb.

HELP - The COMMAND DOS* FORGOT - $20

DOS HELP by Flambeaux. Complete with examples of DOS* command usage. Far easier to use and faster than a manual. Not copy protected. 1 lb.

1200/300 Baud Internal modem with 5 YEAR guarantee - AND - PROCOM, ACCESS and POP-UP DESKSET+ - $90

Hayes compatible - PC, XT, AT compatible - Complies with BELL 212A and 103 STANDARDS. 3 lb.

SURGE SUPPRESSOR 6 OUTLET POWER STRIP with EMI/RFI FILTER - $15

Full 3 line protection in normal and common modes. Noise filter helps protect against noise interference. "Push to reset" circuit breaker protects against overloads. 6 foot 14/3 SJT power cord. On/Off switch. Continuous surge protection for your valuable electronic equipment. UL listed. 3 lb.

***TERMS** - Cashiers check, immediate shipment - All others must clear. No COD. Prices FOB Torrance, CA. Call us and give your ZIP code and we will quote freight charges. California residents add 6½% sales tax. Add $2 handling to orders under $25.*

KOALA COMPUTERS INC.

4306 Torrance Boulevard, Torrance, California 90503 (Offices only)
CALL 213-316-5866 9 to 6 PST M-SAT

*XT trademark IBM, DOS - Microsoft, DBase - Ashton Tate

Reader Service Number 88

Figure 4 — Assembly Language Fixed Point Calculations

```
; muldiv(long A, int b, int c)
; Multiplies 32-bit "A" by 16-bit "b", then divides by
; 16-bit "c" such that abs(c) >= abs(b)...NO CHECK FOR 0 DIVISOR...
;
; NOTE how DeSmet figures stack upon entry:
; 16-bit return address at sp, "A" at sp+2, b at sp+6, c at sp+8 ...
; NOTE DeSmet does not use "word ptr" formalism ...
;
; REGISTERS:            eax     ...initially holds "A"
;                       ebx     ...holds b
;                       ecx     ...holds c
;                    edx:eax    ...product A*b (before idiv)
;
; RETURN result in dx:ax.

cseg
public    muldiv_
muldiv_:
     db 67h,66h,8bh,44h,24h,02h         ;mov eax,dword [esp+2]
     db 67h,66h,0fh,0bfh,5ch,24h,06h    ;movsx ebx,word [esp+6]
     db 67h,66h,0fh,0bfh,4ch,24h,08h    ;movsx ecx,word [esp+8]
     db 66h,0f7h,0ebh                   ;imul ebx
     db 66h,0f7h,0f9h                   ;idiv ecx
     db 66h,0fh,0a4h,0c2h,10h           ;shld edx,eax,16
     ret                                ;short return...

END OF LISTING
```

With our 8/24 format, and our limited P,Q range, we'll have accuracy similar to IEEE single precision. At some point, even this accuracy isn't enough; if we try to examine regions with very small P and Q ranges (e.g., P_{max} - P_{min} < 0.00001), part of our Mandelbrot map could be noise.

Also, note that we can get a fixed point number in C by multiplying the corresponding "float" by 2^{24}, then truncating the result to a long integer. In general, though, we will try to do all our fixed point math with integer operations.

The Program

In this section we'll discuss the workhorse functions that do the Mandelbrot calculations over a fixed P,Q range. I'll leave it up to you to supply the calling program. Alternatively, you can download the complete program and executable code from the Micro C RBBS (503) 382-7643, or send $6 to Micro Cornucopia, P.O. Box 223, Bend, OR 97701, for the source listing (ask for Issue Disk #43). Once again, I strongly recommend reading Larry Fogg's *Micro C* article on the Mandelbrot set.

Figure 2 shows how the Mandelbrot calculation is typically coded in floating point math; this is a modification of Larry Fogg's mandel(). Figure 3 shows how the function is altered to use fast assembly language routines. The assembly functions are described below.

To save space in the listings, I've converted the 386 instructions completely to byte encodings, but all the mnemonics are commented in, so it should be easy to adapt the programs to any compiler/assembler. If you wish to use another compiler/assembler, be sure you understand the order in which values are pushed onto the stack. Not all compilers are like DeSmet C, so you may have to change the values of "m" in instructions containing "[esp+m]."

Also note that some compilers will require you to save the si and di registers at the beginning of each function. Finally, we'll assume that the upper 16 bits of esp are zeroed and considered meaningless by our operating system; since we're launching the program from MS-DOS, that's a valid assumption.

The Assembly Language Functions

The first assembly function, muldiv(), performs the operations A*b/c, where A is a 32-bit integer, and b and c are 16-bit integers such that $|b| \leq |c|$. The calculation is done in a manner that avoids overflow and truncation. This type of function is extremely useful in DSP (digital signal processing) applications; since muldiv() requires very few 386 instructions (see Figure 4), it serves as a good beginning example.

Note that with the 386 we can use [esp] directly to address the stack; that is, we don't have to go through the "push bp, mov bp, sp ..." formalism so familiar to 8086 programmers. However, we must make sure esp actually has a 16-bit value when we address the stack, else we'll generate a real mode stack exception error.

Figure 5 — Looping Routine

```
; mandwhile(long P, long Q, int nmax)
; Performs all calculations for "while" loop in L.Fogg's mandel()
; function, using fixed pt math with binary pt between bits 23 and 24...
;
; FOR: DeSmet C/asm88 small case...
;
; In REAL MODE; uses 32-bit overrides on register length and addressing to
; gain performance of 32-bit regs and instructions...
; Returns # iterations before divergence (16-bit int returned in ax).
; Note 1/3 rd of instructions are for rounding, to get just 1/2 extra bit of
; accuracy; if you aren't that picky, dump these instructions for 10% or more
; extra speed...
;
; REGISTERS:            ebp     ...ytemp at top of loop, modulus_sqrd
;                                  at bottom of loop
;                   edx:eax     ...multiplication and temporary storage
;                       edi     ...P
;                       esi     ...Q
;                       ebx     ...x
;                       ecx     ...y
;
; RETURNS stopping number of iterations in ax.

 dseg
 public    counter
 counter   dw   0
 cseg
 public    mandwhile_
 mandwhile_:
         db 66h, 55h                     ;push ebp
        ;---get P ...
         db 67h, 66h, 8Bh, 7Ch, 24h, 06h  ;mov edi,dword [esp+6]
        ;---get Q ...
         db 67h, 66h, 8Bh, 74h, 24h, 0Ah  ;mov esi,dword [esp+10]
        ;---get nmax...
         db 67h, 8Bh, 44h, 24h, 0Eh       ;mov ax,word [esp+14]
        ;---initialize counter ...
         mov word  counter,ax
        ;---zero out x and y storage ...
         db 66h, 33h, 0DBh                ;xor ebx,ebx
         db 66h, 33h, 0C9h                ;xor ecx,ecx

 looper: ;---ytemp=x*y---
         db 66h, 8Bh, 0C3h                ;mov eax,ebx
         db 66h, 0F7h, 0E9h               ;imul ecx
         db 66h, 05h, 00h, 00h, 80h, 00h  ;add eax,800000H   ;for rounding...
         db 66h, 83h, 0D2h, 00h           ;adc edx,0         ;for rounding...
         db 66h, 0Fh, 0A4h, 0C2h, 08h     ;shld edx,eax,8    ;div by 2**24...
         db 66h, 8Bh, 0EAh                ;mov ebp,edx
        ;---x=x*x - y*y + P------
         db 66h, 8Bh, 0C3h                ;mov eax,ebx
         db 66h, 0F7h, 0EBh               ;imul ebx
         db 66h, 05h, 00h, 00h, 80h, 00h  ;add eax,800000H
         db 66h, 83h, 0D2h, 00h           ;adc edx,0
         db 66h, 0Fh, 0A4h, 0C2h, 08h     ;shld edx,eax,8
         db 66h, 8Bh, 0DAh                ;mov ebx,edx
        ;---
         db 66h, 8Bh, 0C1h                ;mov eax,ecx
         db 66h, 0F7h, 0E9h               ;imul ecx
         db 66h, 05h, 00h, 00h, 80h, 00h  ;add eax,800000H
         db 66h, 83h, 0D2h, 00h           ;adc edx,0
         db 66h, 0Fh, 0A4h, 0C2h, 08h     ;shld edx,eax,8
         db 66h, 2Bh, 0DAh                ;sub ebx,edx
         db 66h, 03h, 0DFh                ;add ebx,edi      ;add P...
        ;---y=(ytemp<<1) + Q----
         db 66h, 0D1h, 0E5h               ;shl ebp,1
         db 66h, 03h, 0EEh                ;add ebp,esi      ;add Q...
         db 66h, 8Bh, 0CDh                ;mov ecx,ebp
        ;---modsqrd = x*x + y*y---
         db 66h, 8Bh, 0C3h                ;mov eax,ebx
         db 66h, 0F7h, 0EBh               ;imul ebx
         db 66h, 05h, 00h, 00h, 80h, 00h  ;add eax,800000H
         db 66h, 83h, 0D2h, 00h           ;adc edx,0
         db 66h, 0Fh, 0A4h, 0C2h, 08h     ;shld edx,eax,8
         db 66h, 8Bh, 0EAh                ;mov ebp,edx
        ;---
         db 66h, 8Bh, 0C1h                ;mov eax,ecx
         db 66h, 0F7h, 0E9h               ;imul ecx
         db 66h, 05h, 00h, 00h, 80h, 00h  ;add eax,800000H
         db 66h, 83h, 0D2h, 00h           ;adc edx,0
         db 66h, 0Fh, 0A4h, 0C2h, 08h     ;shld edx,eax,8
         db 66h, 03h, 0EAh                ;add ebp,edx
        ;---test if modsqrd > "4"---(67108864 = 4*(1 << 24))---
         db 66h, 81h, 0FDh, 00h,00h,00h,04h  ;cmp ebp,67108864
```

```
          jae home
          dec word  counter
          jnz looper
 home:    mov ax,word  counter
          neg ax
          db 67h, 03h, 44h, 24h, 0Eh       ;add ax, word [esp+14]
          db 66h, 5Dh                      ;pop ebp
          ret

END   OF FIGURE 5
```

Figure 6 — Smoothing Color Transitions

```
; 2-20-88 transform(int n, int switchpt)
;
; FOR: DeSmet small case C/asm88
;
; ...returns in ax the "color" which is then anded with (maxcolors-1)...
; converts the iterations returned by mandwhile() to a log scaling, starting at
; the switchpt. Equiv. to C coding:
; if(n < switchpt) color = n;
; else for(color=i=switchpt,inc=1; i<n; i=switchpt+(inc<<=1),color++);
; ...However, 386 is so fast we need to translate to asm to get full benefit!
;
; NOTE DeSmet short return via jmp !!

cseg
public    transform_
transform_:         pop di          ;return address...
                    pop cx          ;n...
                    pop bx          ;switchpt...
                    sub sp,4        ;required by DeSmet for short return...
                    cmp cx,bx
                    ja do_loop
                    mov ax,cx
                    jmp di          ;short return...
                    ;------
          do_loop:  mov ax,bx       ;init color (ax) with switchpt...
                    mov si,1        ;si = inc...
                    mov dx,bx       ;init i (dx) with switchpt...
                    ;------
          loopit:   cmp dx,cx       ;compare i (dx) to n (cx)...
                    jge done
                    shl si,1        ;inc <<= 1...
                    mov dx,bx
                    add dx,si       ;i = switchpt + inc...
                    inc ax
                    jmp loopit
                    ;------
          done:     jmp di          ;short return, color in ax...

END OF FIGURE 6
```

Figure 7 — Detects Presence of 386

```
; is_386() ... no arguments...
; Checks for presence of 386...
;
; FOR: DeSmet C small case, asm88
; RETURNS: 0 in ax if processor not a 386; else returns non-zero ...
; NOTE short return via jmp.
;
; REF: Juan E. Jimenez, Turbo Technix Jan/Feb 88, p. 55.

cseg
public    is_386_
is_386_:            pop di          ;return address ...
                    pushf
                    xor ax,ax
                    push ax
                    popf
                    pushf
                    pop ax
                    and ax,8000h
                    sub ax,8000h
                    jz home2
                    ;------
                    mov ax,7000h    ;if here, either 286 or 386 ...
                    push ax
                    popf
                    pushf
                    pop ax
                    and ax,7000h
          home2:    jmp di
END OF FIGURE
```

The "movsx" instruction (move with sign extend) is new to the 80x86 family, and is handy in converting 16-bit quantities for use in 32-bit instructions.

Perhaps the most notable aspect of muldiv() is the use of shld edx, eax,16 at the end of the function. This instruction is needed because DeSmet C, like most 16-bit 80x86 compilers, returns 32-bit integers in dx:ax. After the idiv instruction, the desired result is in eax; the shld instruction leaves eax unchanged, but copies the upper 16 bits of eax into dx, so ultimately the correct value is returned in dx:ax.

Looping Calculations

The second and more important function, mandwhile() (see Figure 5), replaces the inner "while" loop in Figure 2; it returns only the stopping value of n.

After the 32*32 bit multiplications, the "add eax,800000H" and "adc edx,0H" instructions assure proper rounding (even for negative numbers) when we use the shld instruction. This step adds only a little accuracy, so you can eliminate the rounding for about 10% more speed and substantially shorter coding. Again, note that we actually accomplish the 24-bit "division" — required to adjust the binary point — with an 8-bit left shift.

mandwhile() is well suited for the 386 architecture, and should run fast even on computers with slow memory, because the algorithm uses the 386 pre-fetch queue very efficiently.

The imul instruction is relatively "slow," averaging about 30 clocks over the P,Q region of interest. While each imul executes, there is plenty of time to fetch and decode the faster add and mov instructions. The queue gets emptied only at the end of each loop iteration, and is quickly refilled.

The shld instruction is ideal for adjustment of the binary point after multiplication. Typically, the equivalent operation takes 3 instructions on other 32-bit microprocessors — e.g., if a 68020 had the 64-bit product in D2:D1, you would probably code:

```
lsl.l #8,D2
lsr.l #24,D1
or.l   D1,D2
```

to replace the shld. Furthermore, the 68020 32*32 multiplication is somewhat slower than the 386 multiplication, so the 386 really has an advantage.

Final Functions

The last two functions called by mandel() are transform() (see Figure 6) and (*ptptr)(). The purpose of transform() is to "slow down" the alternation of colors in rapidly changing portions of the map, making it easier to see broad patterns. Values of n up to the "switch point" are returned unchanged by the function, but values above switchpt are converted to $\log_2$(n-switchpt) + switchpt. This function is easily written in C, but for speed I've also included an assembly language version.

The value returned by transform() is "anded" with endcolor to obtain the pixel color (the "anding" process assumes your graphics adapter can display 2^m colors, with endcolor = 2^m - 1). The transform() function is optional; if you don't want to use it, simply code color = n & endcolor instead. The color and coordinates of the point are then passed to (*ptptr)(), the pixel plotting function.

Since ptptr is an argument of mandel(), we can allow main() to detect the graphics adapter in the computer and choose the appropriate plotting function. If you know for sure that you will be using only one graphics adapter, you can remove ptptr from the argument list of mandel() and replace (*ptptr)() with an explicit plotting function.

The plotting functions can be coded in C (e.g., *Micro C* issue #39 Jan./Dec. 1988, pp. 24 and 84), but the rest of mandel() is so fast, it is preferable to use assembly language.

I've included another useful function in Figure 7; is_386() returns a non-zero value if the computer has a 386 processor, else it returns a zero. Thus you can keep non-386 computers from trying to run 386 code (and consequently going off into never-never land).

How Fast?

How much speed did we gain by using 32-bit instructions? In the past, I've written fixed point Mandelbrot programs in 16-bit 8086 code; on a 16 MHz 80386, 16-bit code takes eight times longer to run than the 32-bit version.

In fact, our 32-bit version is at least three times faster than a 16 MHz 80387 with hand-coded assembly language; 12-15 times faster than an 8 MHz 80287; 22 times faster than fixed point routines on a Macintosh Plus; 100-200 times faster than software floating point on an 8 MHz 80286; or as fast as an 8600 VAX superminicomputer.

There are probably more efficient algorithms than Figure 2. If you find them, perhaps you'll get the calculation time down to seconds.

Final Words

We barely skirted the 386 features you can use under MS-DOS. We didn't cover advanced addressing modes, nor all the powerful new instructions. The 386 is full of goodies, such as BSF and BSR (bit scan forward and bit scan reverse), which make software floating point fast and very easy.

It's important to explore. The chip has so much potential, it would be criminal to use the 386 as nothing more than a fast 286. Instead of chewing your fingernails while Microsoft inflates OS/2 a few more megabytes, start lacing your MS-DOS programs with high-speed 32-bit instructions.

Yes, you're still stuck with 16-bit segments; but how many of your programs really need data structures larger than 64K? Probably very few, and OS/2 can't help you with that problem anyway. When the protected mode operating system finally rolls around, you will be that much ahead of the pack.

♦ ♦ ♦

The PARADOXical Developer

A Database Development Tool With The Power Of A Real Language

I've been hearing a lot of interesting things about Paradox. It's great, it's powerful, it's fast, it's unique, it's smart, etc. Well, someone's got to really improve on the current flock of database packages before dBASE III (and company) will be dethroned. I don't know if Paradox will do that, but it sounds like it could.

A program that could be both powerful and easy-to-use was such a contradictory notion, Ansa decided to name its database manager "Paradox."

In short, Paradox is a paradox. It's —

- easy to use,
- well-documented,
- and powerful.

A "query by example" interface and friendly tabular structure make Paradox easy for even the newest of computer users. And its programming language (PAL) gives you the tools to write stand-alone business applications for clients ('nuff said).

In this introduction to the language, I'll give you a flavor of Paradox developer-power, and whet your appetite for ways to improve your future applications.

A Developer's Biography

I've been designing small business applications (i.e., accounting, inventory, etc.) for a few years and have worked with several database managers, programming languages, and "modifiable" systems.

For a while I used Condor III to produce programs in CP/M. Although it's a friendly database manager, it's limited and not easy to customize. And to make a database system do what you (and your client) want, you almost always have to customize at least a little. To do this efficiently, you really need to create a function library.

So when I needed to customize a system for a client, I switched from Condor III to a programming language, CB86. Then after my client and I decided on the system, I programmed it. When I moved to PC-DOS, I used similar tools.

But now my clients want to modify the system themselves. They want to adjust it for changes in their businesses, especially in such areas as report formatting and data capturing. They don't want to write a report generator from scratch (neither do I), but they want to be able to modify the system without me hanging around. So I moved to more sophisticated database managers, like dBASE III.

Then a friend introduced me to Paradox and I gave it a workout (converting an internal inventory system from dBASE III to Paradox). The conversion went well, and I feel I've found the database developer's dream — a simple record update system and a "language" that's powerful enough for me, yet allows my clients the freedom to customize reports and work magic on *their* data.

A Developer's Needs

A developer's needs are few. He (or she) wants a database manager that makes —

- design easy,
- coding easy,
- & debugging easy.

Easy design means quick prototyping and easy rework.

Easy coding means easy to learn command set, logical structure, good debugging facilities, maintainable/modifiable code, and a good correlation between the command language and the underlying system. Of course, the user also wants an easy-to-use program.

The database manager should allow a developer to replicate the actual operation of the client company as much as possible. Screen representations of company forms, data flow that follows the company's daily routines, and reports that match expectations make clients comfortable.

Too many database managers in the past haven't given a developer the assistance he needs in order to meet client demands. Either the command language is obtuse (requiring a long learning curve) or too limiting.

Prototyping tools are often nonexistent, debuggers primitive, or (the worst, for me) the command language works entirely unlike the interactive version of the program. And, without powerful, easy-to-use report and screen generators, it's a long and arduous job to replicate a business's "paper" on the computer.

Finally, if you have to construct everything yourself anyway, it's easier to use a "regular" language. Fortunately, Paradox lends a hand.

Paradox Specs

Paradox is a full-function database manager developed by Ansa, which is now a part of Borland International. It runs on IBM or compatible clones with at least 512K memory, dual floppies, monochrome video, and DOS 2.x or higher.

But it runs better if you have 640K of memory, a hard disk, and a fast processor. If you have EMS or EEMS memory, Paradox can use it.

Paradox is highly interactive, giving you a visual look at the data and letting you work directly with it to produce queries, reports, and the like.

Paradox allows up to 2 billion records in a database table, with up to 255 fields of 255 characters in each field. And with 15 report formats and 15 screen formats per table (plus 15 screens per format).

For relational work, you can use an unlimited number of tables. You can pull information from many different databases at one whack to form a temporary (answer) table for reports or further processing.

Paradox comes in a network version

By Ben Sevier
90 Neck Rd.
Madison, CT 06443

with full record locking and programmer control of multi-user applications. (This is in version 2.01, but most of my comments apply to older versions as well.)

The user interface resembles the Lotus 1-2-3 horizontal menuing system, so typical business users already familiar with Lotus can quickly get up to speed.

To facilitate data exchange with other programs, Paradox can read and write Lotus 1-2-3, Symphony, dBASE II & III, Visicalc (DIF), and ASCII files.

Query By Example

The heart of Paradox lies in its "query by example" method of working with data. This means that a user, working with a representation of the data table (a file), puts an "example" of the data in the field to be searched. This example can be explicit (i.e., "Jones" in a name field), or more generic (i.e., "Jon..") to find any record starting with Jon, using a wild card match.

The program then selects records based on the example, and returns them in the "answer" table. You can then examine, print, edit, restructure, export, save, or query the answer table.

Everything in the system works around the query principle. And (in theory) there's no limit to the number of tables you can query at one time in order to combine information.

Personal Programming

A program generator called the Personal Programmer lets you create menus, tables, and reports. These are compiled into a standalone system (or application) for users not comfortable with Paradox's interactive mode.

The procedural language, called "PAL," is for the programmer. It supports procedural scripts, keyboard macros, program encryption, and will tokenize the application in libraries.

Ansa has also tossed in an interactive debugger, error trapping procedures, and complete keyboard control capabilities. What more could you ask for? (Well, a few things, actually, but...)

For those who like to market their work, a runtime module is available. For $9 you can distribute the runtime routine to up to 250 clients. That's it, no fuss, no royalties.

Figure 1 — Structure of a Sample Table

STRUCT	Field Name	Field Type
1	Client ID	N
2	Last Name	A13
3	First Name	A10
4	Last Visit Date	D
5	Current Balance	$

Developing A System

With all these goodies where do we start?

Well, the beginning is usually the best, which in a database application means planning the system. Sandy Brabandt covered this very well in her database series in *Micro C*, issues #35, #36, & #37. I suggest you look over her articles (they're available as a *Micro C* reprint, $5 per copy, postpaid anywhere) for design ideas.

Then when you've brought the design far enough along, you can begin the fun.

Your first order of business is to define tables. This is fully interactive in Paradox. You:

- name the fields in a table on the screen,
- assign data types such as number, date, alphanumeric, fields, to the
- fill in the table by entering data in a spreadsheet-like format,

and if you want to modify the structure or data type at any time you can, without losing anything.

Figure 1 shows the structure of a sample table. Notice that field names can be long and have spaces.

Screens

Once the table is somewhat stable (!), you can lay out screens using the powerful full-screen painting editor.

You place each field and associated text where you want it, along with lines, boxes and display attributes such as intensity or reverse. If you work with calculated dollar amounts, you'll enjoy using calculated fields on the screen just as if they were part of the table.

Reports

The report generator is also easy to use and powerful, with either a default "tabular" structure or a free form design. You have fairly complete control over field and text placement, with break control based on groupings or number of records.

Like most report generators which have to deal with the many unknowns of printer capability, this one requires a little experimentation.

Built-in printer control strings help, as does the ability to embed control strings within the report. A built-in label format for printing mailing labels is also useful.

Using the report writer, I've been able to generate invoices, form letters, style reports, and, in short, almost any type of report I need.

Like the screen version, the report generator treats calculated fields as if they're part of the data, and can do interesting calculations with dates as well as numbers. (I especially like being able to calculate "TODAY - 45" for aging reports, and get the correct answer). As a final fillip, the user has the choice of printing to the printer, to the screen, or to a disk file.

Queries

The final interactive step is setting up the queries. You do this on screen, using query by example. These queries can be saved, then played back on demand.

Figure 2 shows a sample query table. Checks denote fields you want to include in the answer. "Jones" in the Last Name field sets up the selection criteria.

Once you've set up the tables, saved the queries, and readied the screens and reports, you can run through an interactive session with your client, deciding on any changes to structure flow or results.

When your client's satisfied, tie it all together with the scripting language, PAL (Paradox Application Language).

Gary's note: I've been using Paradox for several months now, and I'd like to emphasize the power of querying, a feature which makes Paradox stand out among relational databases.

Once you've stored information in Paradox tables, you can then select, combine, manipulate, and retrieve information very quickly by querying. You can ask "what if" questions about your data. You can calculate, rearrange, enumerate, and summarize data.

You can set ranges, use logical operators (like >, <, etc.), and retrieve information by matching patterns. And each time you query, your results are automatically put into a new table (or Paradox database). You can then use this new (answer) table for new queries, reports, etc.

Querying is very powerful and I haven't seen anything quite like it.

Figure 2 — Sample Query Table

```
CLIENT     Client ID  Last Name    First Name  Last Visit Date
                      Jones
```

Figure 3 — Sample PAL Fragment

```
PROC Please.Wait(X)
     Clear                              ; clears the screen
     If upper(X) = "NO MESSAGE" then    ; converts & checks
        x = ""                          ; if message is to be
     Else                               ; shown
        x = " - " + x                   ; puts in "-"
     Endif

     @ 10,0 ?? format("w80,ac","  Please Wait" + x)
     @ 1,1                              ; prints message on
ENDPROC                                 ; screen

(Sample PAL script showing definition of a procedure)

While CHOICE <> "Quit"                  ; sets exit
      ShowMenu                          ; puts menu on screen
          "NewClient"     :  "Enter a NEW CLIENT"
          "Update"        :  "Change or Delete records"
          "View"          :  "Look at client files"
          "Quit"          :  "Return to Main Menu"
      Default "View"                    ; sets default
      To CHOICE                         ; sets variable
          Switch                        ; selects next action
               CASE Choice = "NewClient" :
                    Run "New"           ; calls another script
          EndSwitch
....
EndWhile                                ; end of loop

END OF LISTING
```

Using The Language

PAL is Pascal-like and it lets you set up menus, control program flow, call up tables for edit and data entry, run reports, and so on. It has a very rich command set, running the gamut from system level primitives to procedure library calls. You have full control of the screen and the keyboard.

You can use the Paradox Script Editor for editing scripts, or (since the scripts are ASCII text) you can use your favorite editor. In fact, you can set up Paradox to call your editor instead of the Paradox Script Editor (a step I recommend since the script editor is limited).

The sample PAL fragment (Figure 3) shows how to set up a line menu at the top of the screen.

This menu acts just like one in Lotus 1-2-3, with highlighting of each item by cursor key, or selection by first letter. (<CR> selects CHOICE: variable name; the script moves to the proper CASE statement and continues on.)

After you've developed a script, you run it to check out the operation.

Debugging The Script

The Paradox Debugger is online anytime a script is running. It watches for syntax errors, runtime errors, or anything else that causes the script to fail.

You can then work interactively with the debugger to step through a script and make changes on the spot.

And you can trap all errors before they invoke the debugger by using the "errorproc" facility, and write your own error-handling procedures. This is handy for catching printer offline, missing disk errors, etc. Any errors you don't wish to trap can be passed on to the debugger.

Personal Programmer

Or you can use the Personal Programmer (PPG), a code generator, to trap errors and the like. Many developers use the PPG to set up the basic menuing and security system, then finish up with PAL.

The PPG, like most program generators, writes code that works, though it's a bit bloated and slow. But the PPG does offer an easy (and instructive) way to write tricky procedures. In fact, some operations can only be accomplished with the PPG.

ERAC CO.

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

AT

Motherboard 6 & 10 Meg
Zero Wait State
8 Expansion Slots
1 Meg RAM On-Board
Math Co-processor Option
Phoenix Bios
200 Watt Power Supply
Hercules Compat. Video Bd.
Parallel Port
2 Serial Ports Active
Game Port
Clock/Calendar
Hard Disk & Floppy Controller
20M Hard Drive
1.2M 5¼" Floppy Drive
360K 5¼" Floppy Drive
5061 Keyboard
Case with Turbo & Reset,
Hard Drive Light and
Keyboard Disable Switch
Amber Graphics Monitor

$1581

EGA ADD $449
40M HD ADD $150
6 & 12 MHz ADD $73

BABY AT

Motherboard 6 & 10 Meg
Zero Wait State
8 Expansion Slots
80286 Processor
Math Co-Processor Option
1 Meg RAM On-Board
Phoenix Bios
200 Watt Power Supply
Hercules Compat. Video Bd.
Parallel Board
2 Serial Ports Active
Game Port
Clock/Calendar
Hard Disk & Floppy Controller
20M Hard Drive
1.2M 5¼" Floppy Drive
360K 5¼" Floppy Drive
5061 Keyboard
Mini AT Case with Turbo &
Reset, Hard Drive Light and
Keyboard Disable Switch
Amber Graphics Monitor

$1531

EGA ADD $449
40M HD ADD $150

XT/TURBO

Motherboard
5 & 8 MHz Switchable
8088 — V20 Optional
Optional Co-processor
8 Expansion Slots
ERSO or Bison Bios
640K RAM
150 Watt Power Supply
Hercules Compat. Video Bd.
Parallel Board
2 Serial Ports Active
Game Port
Clock/Calendar
Hard Disk and
Floppy Controller
20M 5¼"Hard Drive
2 ea. 360K 5¼" Floppy Drive
AT Style Keyboard
Standard Slide Case
Amber Graphics Monitor

★ ★

$999

EGA ADD $429
40M HD ADD $150
5 & 10 MHz ADD $21

NiCds

AA Cells .6aH	$1.00
12V Pack AA Cells .6aH	6.50
Sub-C Cells 1.5aH	1.50
12V Pack Sub-C	10.00
Double D Cell 2.5V 4aH unused	8.00

GEL CELLS

12V 1.9Ah	$7.50
D Cell 4aH As Is	1.00
12V 24aH	35.00

ROBOTICS

12V DC Gear Motor 1"x3", 30RPM	$7.50
5V DC Gear Motor with Tach 1"x2"	7.50
Joystick, 4 switches, 1" knob	5.00
Z80 Controller with 8-Bit A/D	15.00
Brushless 12VDC 3" Fan	7.50
Capacitor, .47farad 10V 1"x1¾"	4.00
Solar Cells .5V .5A, .8x.6	2.50
High Voltage Power Supply	
Input: 15-30V DC	
Output: 100V 400V 16KV	6.50

ELGAR UNINTERRUPTIBLE POWER SUPPLIES

200 Watt MODEL SPR201 $129 — 400 Watt MODEL SPR401 $149 THESE SUPPLIES MAY HAVE SOME MINOR COSMETIC DAMAGE, BUT ARE ELECTRICALLY SOUND. SQUAREWAVE OUTPUT. RUN ON INTERNAL OR EXTERNAL 24VDC BATTERY WHEN LINE GOES DOWN. TYPICAL TRANSFER TIME = 12MS.

Battery Supplied — Not Guaranteed **NEW 24V INTERNAL BATTERY $75**

KAYPRO EQUIPMENT

9" Green Monitor	$50.00
Keyboard	75.00
PRO-8 Mod. to your board	149.00
Host Interface Board	15.00
Kaypro II	250.00

KAYPRO IC'S

81-189 Video Pal	$15.00
81-194 RAM Pal	15.00
81-Series Character Gen. ROMs	10.00
81-Series Monitor ROMs	10.00

POWER SUPPLIES

0-8VDC 100A Metered	$249.00
Volt & Current Regulated	
5V/1A. -5V/.2A, 12V/1A, -12V/.2A, -24V/.05A	9.90

HOURS: Mon. - Fri. 9 - 6 — Sat. 10 - 4
MINIMUM ORDER — $15.00
TERMS: VISA, MasterCard, Certified Checks, Money Order, NO COD. Visa and MasterCard add3%. *Personal checks must clear BEFORE we ship.* Include shipping charges. California residents add 6½% Sales Tax. For more information please call.

CPU & SUPPORT CHIPS

MC68000-8 CPU	$8.00
Z80 CPU	.75
Z80A CPU	1.50
Z80 CTC	1.50
Z80A PIO	2.00
Z80A SIO	5.00
8088	6.50
8089-3	6.50
D8284A	2.50
SIP DRAM 256-12	7.00
4164-10	2.50
4164-12	2.10
1793	6.00
1797	7.00
6845	5.00
VC3524 Switching Regulators	5.00
1458 Dual Op-AMP	.70
LM2877P 4W Stereo Amp Dual	2.50
MB81464-15	2.75
2716	3.00
2732	3.25
2764	3.50
27C128-1	9.00
74HC00	.38
74LS125	.30
74LS373	.50
74LS174	.30

SWITCHERS

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/1¼A, 12V/.6A, -12V/.6A	29.00
5V/10A	19.00
5V/20A	24.00
5V/30A	39.00
5V/75A, 12V/8A, 24V/5A	55.00
5V 100A	100.00
5V 120A	110.00

TEST EQUIPMENT

OSCILLOSCOPES

TEK 7403N/7A18N/7B50A 60 MHz	695

ANALYZERS

TEK 491 10MHz - 40 GHz	$4500.00
Nicolet 500A 1 Hz 0-100 KHz	1800
Biomation 805 Waveform Rcrdr	259.00
Biomation 8100 2-Channel Waveform Recorder	795.00
Wavetek 142 10MHz Func. Gen.	250.00
HP1600A/1607A Logic Anlyzr	1000.00

DBASE BOOK OF BUSINESS APPLICATIONS *by Michael J. Clifford*
Reg. $19.95 *NOW ONLY* **$3.95**

Reader Service Number 93

Libraries

When your application runs the way you want, you can put your script into a "library" where it's tokenized (or compiled, in Ansa terminology).

The Paradox memory manager allocates memory for running scripts, and, since it maintains a "recently used" buffer of available library procedures, applications run faster if they've been placed in a library.

Besides putting scripts into libraries (which makes them pretty unreadable), you also have the option of encrypting them for security. Encryption also works for tables, forms, or report formats.

Which brings up the question of security, both for your work and protecting the client's data. Paradox provides a fairly robust security system, with multiple levels of access based on passwords. They claim that their encryption is unbreakable, a claim I won't dispute.

Wrap Up

The documentation is generally good, covering the main parts in detail. You'll find, however, that many sections of the manuals don't make sense until you're comfortable with the interactive aspects of the system. But, there's help!

If you develop with Paradox, Ansa has a Paradox Developer's program with disks full of sample scripts and tools, as well as a direct tech support hot line. I've used both, and gotten answers (pretty much) as needed.

There are also books written on the subject, and The Cobb Group puts out the *Paradox User's Journal.*

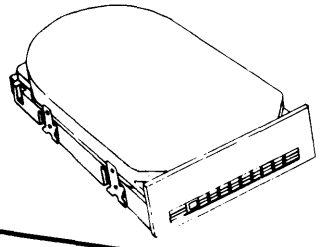
The best source of Paradox help, however, comes from the Los Angeles Paradox User's Groups (LAPALS). They have an excellent newsletter full of problems, solutions, and script ideas and they have chapters throughout California. (Of course, anyone can subscribe to the newsletter.) *(Editor's note: For third party support, including user groups, call (800) 447-4700.)*

Finally, I highly recommend Paradox for user and developer alike.

The Cobb Group
P.O. Box 24480
Louisville, KY 40224-9985

Borland International, Inc.
4585 Scotts Valley Dr.
Scotts Valley, CA 95066
(408) 438-8400; TLX: 172-373

♦ ♦ ♦



20 MEG HARD DRIVES

SEAGATE Model ST-225

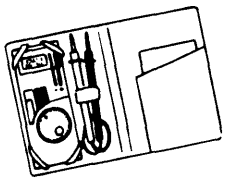
$219.95 (full 6 Mo. Warranty)
(less face plate)

Western Digital Controller For Model ST-225 Drive
$69.95 x/cable

Yes!!! These are for IBM compatible machines

5.25" HARD DISK CONTROLLER CARD

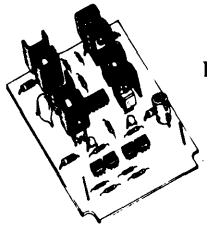
For any Modern 5.25" Hard Drive
$89.95 w/cable



BECKMAN DM78 MULTIMETER

- Pocket Size
- A-C, D-C Volts & Ohms
- Continuity Beep Tester
- Autoranging
- L.C.D. Display
- Carrying Case

Our Biggest Seller!
$29.95



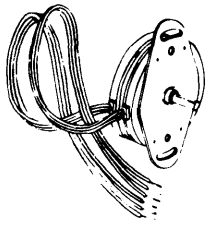
D-C Motor Controller

- Control 2 D-C motors with a computer or other logic source
- For motors rated 6-24 VDC
- Control forward/reverse/run/cw/ccw/stop
- Up to 6 Amp starting surge, 4 Amp cont.
- Dynamic breaking (capable)
- Will also run most 4-lead stepper motors

29.95

STEPPER MOTORS

Copal #SP-57
1/4" Shaft,
7.5 deg./step,
36 Ohm, 12
VDC
$6.95



Superior #
M061-FD02
1/4" Shaft,
35 oz."
torque,
2.1A., 5VDC
$9.95

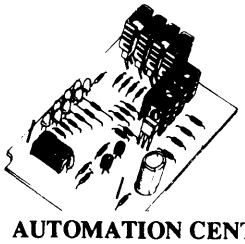
XT CLONE

Building?
Upgrading?

We Stock The Parts And Hardware

You Need

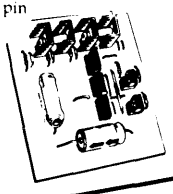
Techna-Kit



USMD-C

- Control standard 6-lead stepper motors with a computer or other logic source
- For motors rated 1.7-12.0 VDC
- Optical Isolation
- Control: forward/reverse/step rate/stop
- Industry standard 22 pin edge card connector

$29.95



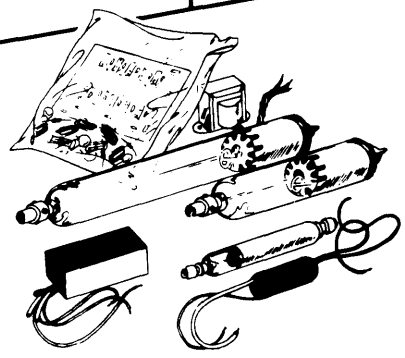
AUTOMATION CENTER

- Use your computer to provide automation
- 8 seperate driver ports per card

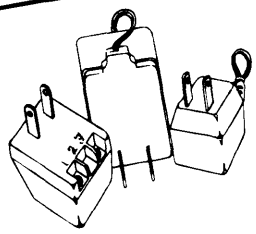
8 TTL/Cmos inputs

- 1 user defined sense switch
- 6-24 VDC
- 4 Amps/driver (max current)

$29.95



REPLACEMENT WALL TRANSFORMERS



#45-753 Universal Out Put $8.40
3 volt - 4.5 volt - 6 volt - 7.5 volt - 9 volt - 12 volt @ 500ma DC

#45-773 16 volt @ 1.1A AC $8.40

#E-54749 4.5 volt @ 265ma DC $2.49

#WP-410608 8 volt @ 750ma AC $2.49

#YS-4510 Nicad Charger 1.5 Volt @ 400ma DC $2.49

Lasers

5 MW Laser Tube	$89.95
Power Supply Kit (115VAC)	$69.95
Power Supply (wired) (12VDC)	119.95
1 MW Laser Tube	119.95
Power Supply (Bat. Pwrd.)	99.95
7 MW Laser Tube	149.95
Power Supply (12 VDC)	119.95

(These lasers are brand new and guaranteed to have a cosmetic defect or not meet manufactures full specifications. All are tested in our lab to insure your satisfaction.)

WARNING:
Voltages present and use by lasers can be lethal...Permanent eye damage could result from direct exposure to an oncoming laser beam. Only those persons qualified to handle such potentials should do so.

OUR 20TH Year

united products corporation
DISTRIBUTORS OF ELECTRONICS SINCE 1968

1123 VALLEY STREET • SEATTLE, WA 98109-4425

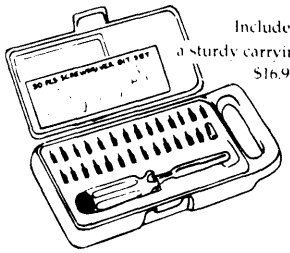
PHONE: (206) 682-5025

FAX: (206) 682-5593

VISA AMERICAN EXPRESS

M-F 9-6
SAT 9-5

30 PIECE SCREWDRIVER BIT SET



Includes a sturdy carrying case
$16.95

Reader Service Number 101

A Menu Generator For dBASE

Improving The User Interface

There are a lot of commercial menu generators for dBASE. Here's one with source that you can use immediately or you can expand it to meet your own needs. Bret begins with a look at the art of designing menus.

Menus can make your applications much friendlier. But building them can be a pain. In this article I'll discuss my idea of good menu design, and show you how to create an automatic menu generator in dBASE III Plus.

From the beginning, I'd like to emphasize that consistency is a key element of menu design. Menus that are consistent throughout the application make users feel comfortable. Deep into an application, they don't want to be surprised.

If you're more interested in code than theory, you can skip the rest of the article and check out Figure 4, the listing for the menu generator. (Example menus and other dBASE III support files are available on the Micro C RBBS (503) 382-7643 or the Micro C Issue #43 disk for $6.)

With that out of the way, let's take a look a menu design.

Menu Design Basics

A few simple rules will help you design your menu interface —

(1) Make the menu information stand out.

(2) Take advantage of the line drawing features of the PC to make your menus distinctive.

(3) Use double-line windows ("the box") to set your menus off from their surroundings. This works especially well for pop-up menus.

(4) If your application has a number of different categories of options, use a single-line box to enclose them. This helps a user decide. Let her start generally (e.g., to print a report), and move down to particulars.

(5) If your program will run on a color monitor, take advantage of it. Use a soft blue or gray for the background and a sharp white or black for the foreground. Or, paint distinctive color tiles to show option types.

But don't go overboard — too many colors can be an eyesore. Use fewer colors in subtler tones — save the bright reds and yellows for emergencies.

Figure 1 - Menu Screen

```
<system>          <title>          <time>    <date>

                    <category>
                     --------

                A -- Option 1
                B -- Option 2
                C -- Option 3
                      .
                      .
                      .

                L -- Option 12
```

By Brett Fishburne
5009 Delaware St.
College Park, MD 20740

(6) Remember your audience. If you're aiming for a middle-aged or older user, space out the text, and don't put too many options on a screen at once.

While you might mute and simplify the color for the adults, you'll find the kids fascinated by swirls of bright graphics. Plus, musical accompaniment will really brighten a children's menu.

(7) And finally, don't upstage the program. A color menu is great — but a simple monochrome program will look crude after all the flash.

Your Menu Is A Road Map

A good menu (system) guides an unfamiliar user through the maze of options. So, let the user know where he is all the time.

Begin with something like the header field I've shown in Figure 1. The leftmost entry is reserved for the application name, or system, which the user is in. This is important if you plan more than one package with the same menu design.

The second field from the left contains the menu name. Highlighting this field draws the user's attention, just in case the user isn't where he expected.

The final two fields contain the date and time. In general, these aren't that useful. But they do give the user a visual check of the day (and time) the system thinks it is.

At this point, the user knows where he is and how he got there. Since menus are supposed to guide the user through the program, it should be easy to traverse the menu structure. The following key assignments are helpful in such a context:

ACTION	KEYSTROKE
Previous Menu	Pg Up
Top Menu	Home or Esc
More Choices	Pg Dn
Exit the program	End or F10
Help	F1

Thus, the user can move freely through the menus, identifying present position from the header.

Menu Layout

Good menus are clear and helpful. Put options under category names whenever possible. Consider expanding each section into a series of submenus.

The category title should be in capitals and centered above the options list. The entire configuration should be enclosed in a single line box to set it off from the other options.

Center the most important, or most used, categories. A user usually focuses on the center of the screen. So put the options in the center (or top center) of the screen.

If there are two categories, put the more important one on the left. Americans read from left to right, so things on the left are considered more important.

For three or more categories, use a pyramidal or diagonal structure. If one category really stands out above the other two, put it centered at the top and place the other two on the bottom, as in a pyramid. If the three are nearly equal, put the more important on the top left, then trail the other two towards the bottom right. This takes advantage of both the left-right tendency and centering.

Finally, use simple, consistent notation. If you're putting up report options, call the category "REPORTS" instead of "OUTPUTS."

The Auto Menu Generator

The auto menu generator uses two databases to prepare each menu. One database, menufile.dbf, holds the data to go into the menu, and the second, tilesets.dbf, holds the layout for the menu.

Menufile (in Figure 2) has the first two fields of the header in the SYSTEM and TITLE fields. The TILE field corresponds to a box for a category.

Nine such categories are possible, and the value in TILE indicates where the data in the remainder of the fields belongs. ELEMENT NO contains the

Figure 2 - Menu Data

MENUFILE.DBF

SYSTEM	character	33
TITLE	character	10
TILE	numeric	1
ELEMENT NO	numeric	2
ELEMENT TY	numeric	1
ELEMENT SI	numeric	2
ELEMENT	character	78

position of the category title relative to the left edge of the enclosing box or the number of the option title. ELEMENT TY is the type of element defined in this record.

A value of one indicates a category title, a value of two refers to an option title, a value of three means a continuation, and a value of four denotes a message to be displayed in the category box. ELEMENT SI is the size of the element and ELEMENT is the element itself.

Tilesets (see Figure 3) is much simpler. Each of the ten fields (TILE0-TILE9) holds a descriptor which defines the box associated with the category which is supposed to be in that field. The data in the fields is in the form "##,## TO ##,##." The "##,##" signs indicate the left top and right bottom corners of the enclosing category box.

The data from menufile.dbf can go into any of the configurations defined in tilesets.dbf. If there aren't enough tiles defined in tilesets.dbf, you get an error. This process is handled by DrawMenu.prg (a support file).

I designed DrawMenu to give you an idea of what the menu will look like

before you use MakeMenu to generate dBASE code.

The program makemenu.prg (Figure 4) generates the code. MakeMenu and DrawMenu are similar, except MakeMenu stores the commands in a program file.

The support file, PictMenu, lets you define new tile sets or modify previous

Figure 3 - Menu Layout

TILESETS.DBF

TILE0	character	14
TILE1	character	14
TILE2	character	14
TILE3	character	14
TILE4	character	14
TILE5	character	14
TILE6	character	14
TILE7	character	14
TILE8	character	14
TILE9	character	14

sets. Once the tile sets have been filled in (modified), it's possible to insert the data into different tile sets if you're not satisfied with the screen. PictMenu shows you exactly how things will look while you design the screen.

Wrap Up & Enhancements

This menu generator is just the beginning and I've left hooks for enhancement.

For example, I haven't included Help Screens. Each entry in menufile.dbf could have a corresponding entry in helpfile.dbf, for context-sensitive help.

Finally, menu concepts aren't language dependent. Thus, you could generate code for any language from the data contained in menufile.dbf and tilesets.dbf. You can do this by translating the code in MainMenu (which generates the dBASE III Plus source code).

You could also add a mouse interfaced to simplify the menu building process. (Using the mouse to define the boxes is quicker than using keys.)

Editor's note: Plus, the menus themselves could support mouse selection.

Or you could define a pop-up menu protocol which would allow the generation of pop-up menus. A field could be added to menufile.dbf to indicate which menu came next. The options are virtually endless. Good luck, and *enjoy*!

♦ ♦ ♦

Figure 4 — Automenu Generator

```
*********************************************************
*** NAME: MAKEMENU.PRG
***
*** AUTHOR: BRETT FISHBURNE
***
*** PURPOSE: CREATE A PROGRAM WHICH MAKES A MENU
***
*********************************************************
PARAMETER Layout, Instruction, Menu

*** PREPARE FOR NEW PROGRAM
DELETE FILE TEMP.MNU
SET ALTERNATE TO TEMP.MNU

*** ESTABLISH ENVIRONMENT
SET TALK OFF
SET ECHO OFF
SET STATUS OFF
SET SCOREBOARD OFF
SET EXACT ON

*** PREPARE THE DATABASES
SELECT 1
USE MENUFILE
SET FILTER TO TRIM(TITLE) = TRIM(MENU)
GOTO TOP
COUNT FOR ELEMENT_TY = 2 TO MAXLETTER
MAXLETTER = CHR(MAXLETTER + 64)
GOTO TOP

SELECT 2
USE TILESETS

SELECT 1

*** GENERATE PROGRAM
SET ALTERNATE ON

*** GENERATE TITLE HEADER
? REPLICATE('*',60)
? '*** NAME: ' + Menu + '.PRG'
? '***'
? '*** AUTHOR: AUTOMATIC MENU GENERATOR (AMG)'
? '***'
? '*** PURPOSE: DRAW THE SCREEN FOR THE ' + Menu
? '***'
? REPLICATE('*',60)
?

*** GENERATE CONSTANTS
? '*** INITIALIZE CONSTANTS'
? "MAXLETTER = '" + MAXLETTER + "'"
?

*** ALLOW THE MENU TO RUN THROUGH ITERATIONS
? '*** SET UP AN INFINITE LOOP'
? 'DO WHILE .T.'
?
*** GENERATE STANDARD MENU BOX
? '*** DRAW BOX'
? 'CLEAR'
```

(continued from page 38)

```
'@ 2,0 TO 2,79 DOUBLE'
? '@ 0,0 TO 21,79 DOUBLE'
?
*** GENERATE HEADER AND FOOTER
? '*** FILL IN HEADER AND FOOTER'
? '@ 1,2 SAY "' + SYSTEM + '"'
? '@ 1,' + STR(40 - INT(LEN(TRIM(TITLE))/2),2) + ' SAY "' +;
  TITLE + '"'
? '@ 1,55 SAY TIME()'
? '@ 1,71 SAY DATE()'
? '@ 22,' + STR(40 - INT(LEN(TRIM(Instruction))/2),2) + ' SAY "';
  + Instruction + '"'
?
*** DETERMINE FORMAT
SELECT 2
GOTO Layout

*** GENERATE TILE DESCRIPTORS
? '*** DRAW TILES'
i = '0'
temp2 = 1
DO WHILE temp2 # 0

  temp = TILE&i
  mline&i = VAL(LEFT(temp,2)) + 1
  mhorz&i = VAL(SUBSTR(temp,4,2)) + 2

  ? '@ ' + temp

  i = STR(VAL(i) + 1)
  IF i = '10'
    EXIT
  ELSE
    i = LTRIM(i)
    temp2 = LEN(TRIM(TILE&I))
  ENDIF

ENDDO
?
*** GENERATE OPTIONS AND TITLES FOR TILES
? '*** PUT OPTIONS AND TITLES IN TILES'
SELECT 1
DO WHILE .NOT. EOF()

  temp = STR(TILE,1)
  DO CASE

    CASE ELEMENT_TY = 1
      *** TITLE
      ? '@ ' + STR(mline&temp,2) + ',' + STR(mhorz&temp + ELEMENT_NO,2);
        + ' SAY "' + TRIM(ELEMENT) + '"'
      ? '@ ' + STR(mline&temp + 1,2) + ',' + STR(mhorz&temp;
        + ELEMENT_NO,2) + ' SAY REPLICATE("-",' + STR(ELEMENT_SI,2);
        + ')'
      mline&temp = mline&temp + 2

    CASE ELEMENT_TY = 2
      *** OPTION
      ? '@ ' + STR(mline&temp,2) + ',' + STR(mhorz&temp,2);
        + ' SAY "' + CHR(64 + ELEMENT_NO) + ' -- ' + TRIM(ELEMENT);
        + '"'
```

(continued from page 39)

```
mline&temp = mline&temp + 1

    CASE ELEMENT_TY = 3
      *** CONTINUATION
      ? '@ ' + STR(mline&temp,2) + ',' + STR(mhorz&temp + 5,2);
        + ' SAY "' + TRIM(ELEMENT) + '"'
      mline&temp = mline&temp + 1

    CASE ELEMENT_TY = 4
      *** MESSAGE
      ? '@ ' + STR(mline&temp,2) + ',' + STR(mhorz&temp + ;
        ELEMENT_NO) + ' SAY "' + TRIM(ELEMENT) + '"'

  ENDCASE

  SKIP
ENDDO
?

*** GENERATE OPTION HANDLER
? '*** WAIT FOR USER RESPONSE'
? 'I = 0'
? 'DO WHILE I = 0'
? '  @ 1,55 SAY TIME()'
? '  I = INKEY()'
?
? '  *** CALL USER OPTION'
? "  IF (UPPER(CHR(I)) >= 'A') .AND. (UPPER(CHR(I)) <= MAXLETTER)"
IF ' ' $ MENU
  NAME = RTRIM(LEFT(LEFT(MENU,AT(' ',MENU)),7))
ELSE
  NAME = RTRIM(LEFT(MENU,7))
ENDIF
? "     SUBPROG = '" + NAME + "' + CHR(I)"
? '     DO &SUBPROG'
? '  ELSE'
? '     I = 0'
? '  ENDIF'
? 'ENDDO'
?
? 'ENDDO'
?
? 'RETURN'

*** CLEAN UP AFTERWARDS
SET ALTERNATE OFF
SET ALTERNATE TO
CLOSE DATA
RETURN

END OF LISTING
```

Performance and versatility For your CP/M or MS-DOS computer

QP/M

QP/M by MICROCode Consulting
Fed up with the message **"BDOS error: R/O"**? With QP/M, you'll never lose another file because you changed a diskette. QP/M offers full CP/M 2.2 compatibility with outstanding performance and more commands WITHOUT eating up precious program space. Get such features as automatic disk relogging, simple drive/user selection using either a colon or semi-colon, 31 user areas, drive search path, multiple program command line, archive bit maintanence, and transparent time/date stamping; all in the same space as CP/M 2.2. Installs from a convenient customization menu, no software assembly required. Bootable disks available with CBIOS for Kaypro, Xerox (8" or 5¼", -1 or -2), & BBI.

QP/M Operating System, bootable - specify system .... $ 64.95
QP/M without CBIOS (installs on any Z80 system) ..... $ 49.95

Networks

QP/M Network File System by MICROCode Consulting
QP/M Network File System is an efficient local area network allowing up to seven CP/M computers to share peripherals and data resources.

- Transparent operation at speeds up to 11,000 bytes/second in syncronous mode
- Speeds of up to 1,920 bytes/second in asyncronous mode
- Local/remote disk drive and printer support
- Remote peripheral support for modems and real-time clocks
- All stations need not be on the network even though connected
- Local drive access protection and control
- Simple menu oriented configuration utility
- Extended DOS calls are provided for addition of custom network utilities.

Works with interrupt driven Z80 systems such as Xerox 820, Kaypro (KayPLUS & Advent ROMs), Eagle, and other computers running QP/M, or CP/M 2.2

QP/M Network File System ..... $ 39.95

Hard Disks

Need more speed and storage on your system?
Improve the productivity of your Z80 computer with a hard disk.

HDS Host Board
This daughter board provides a convenient interface for connecting a Western Digital WD1002-05 hard disk controller to your computer.

- Plugs into the Z80 socket, no other wiring required
- 40 pin interface for a WD1002-05 (or HDO) controller board
- Switch selectable I/O port addressing
- Comes as bare board or assembled & tested
- Kaypro '84 host board also available

Winchester Connection by MICROCode Consulting
The most simple and comprehensive hard disk software package available for CP/M.

- Designed for use with the WD1002-05 controller board
- Works with one or two hard disks - 5 to 64 meg
- Menu installed, no software to assemble
- Complete hardware tests and error handling
- Automatic swap, for warm boots from hard drive
- Software drivers install above or below CP/M
- Allows custom partition sizes and mixed drive types
- Independent block and directory sizes on each partition
- Includes manual, format, test, park, and swap utilities

Winchester Connection Software only ..... $ 39.95
HDS Board with Winchester Connection Software ..... $ 79.95
HDS Bare Board with software ..... $ 59.95
HDS Board, WD1002-05, and software $245.00
Call or write for other pricing options

WD1002-05 HARD DISK CONTROLLER BOARD by Western Digital

- Standard ST506 drive interface
- Same size as standard 5¼" drive
- 40 pin interface to host computer
- WD2797 floppy disk controller interface on board
- Can control up to three hard drives
- Direct replacement for **Kaypro 10** controller

WD1002-05 Controller Board $185.00
Other Western Digital boards available

Prices subject to change without notice. VISA and Mastercard accepted. Include $5.00 shipping and handling, $7.50 for COD, UPS-Blue or RED Label additional according to weight. Please include your phone number with all correspondence.

Kaypro

KayPLUS ROM Set by MICROCode Consulting
Want more performance and flexibility from your Kaypro? With the KayPLUS ROM set you can have the advantages of a Kaypro 4 or 10, even on your Kaypro 2.

- Install up to four floppies and two hard drives
- Boots from floppy or hard disk
- Supports 96 TPI and 3½" disk drives
- Can use any ST506 type hard drive - 5 to 64 Meg
- 32 character type-ahead keyboard buffer
- Automatic screen blanking (not avail. on 83 series)
- 12 disk formats built-in, unlimited configurable
- Full automatic disk relogging with QP/M
- Internal real-time clock support
- No software assembly required

Includes manual, format, configuration, diagnostics, sysgen, diskette customization utility, AND hard disk utilities. Available for '83 and '84 series Kaypros.

KayPLUS ROM Set, specify model ..... $ 69.95
KayPLUS ROM Set with QP/M $125.00

Parts and accessories for the Kaypro
Kaypro 2X Real-time Clock parts kit ..... $ 29.00
Kaypro 2X Hard disk interface parts kit ..... $ 16.00
Kaypro 10 or '84 series Hard Disk host board ..... $ 49.00
Kaypro four drive floppy decoder board ..... $ 35.00
Complete parts and repair services available

Xerox 820

PLUS2 ROM and X120 Double Density Board by MICROCode Consulting and Emerald Microware
About had it with single density diskettes on your Xerox 820-1? Get unsurpassed versatility with our X120 Board and PLUS2 ROM package.

- Run up to four floppy disk drives at once
- Mix 8" and 5¼" at the same time
- Software compatible with Kaypro and Xerox 820
- Built in drivers for most serial and parallel printers
- Get mini-monitor functions and auto-boot capability
- 19 built in disk formats, including Xerox and Kaypro
- Includes custom disk format definition program
- Banked ROM BIOS for more space in your TPA
- Composite video adaptor on X120 board
- Runs 48 TPI diskettes on 96 TPI drives
- Supports real time clock from Z80-CTC
- Works on the Xerox 820-1 and Big Board I
- Both ROM and X120 board are required for operation

PLUS2 ROM Set and X120 Board A&T $114.95
PLUS2 ROM Set and X120 Bare Board ..... $ 49.95
PLUS2 ROM Set only ..... $ 39.95
120 Bare Board only ..... $ 15.00
★★★ **Special** ★★★ 2 boards for $25, 5 for $50
Other kits, parts, and packages available

Parts and accessories for the Xerox 820
Xerox 820-2 CPU Board - new ..... $ 75.00
Xerox 820-2 Floppy Controller board - new ..... $ 65.00
Xerox 820-2 CPU board w/ Floppy Controller $125.00
Xerox 820-1 CPU board - new ..... $ 75.00
Xerox 820 complete high profile keyboard ..... $ 65.00
Xerox 820 bare high profile keyboard - new ..... $ 25.00
Xerox 820 5¼" drive cable ..... $ 9.00
Xerox internal video cable w/brightness control .... $ 9.00
Xerox 820 power supply ..... $ 35.00
Power connector, specify board or cable ..... $ 2.50
Xerox parallel printer interface cable ..... $ 35.00
Dual Half Height 5¼" Disk Drives - DSDD,
in cabinet with standard Xerox cable $265.00
Complete parts and repair services available

EMERALD MICROWARE
P.O. Box 1726, Beaverton, OR 97075
VISA **(503) 641-0347** MasterCard
30 day money back guarantee on all products.

IBM PC

CP/M, NorthStar, Macintosh, Apple II, MS-DOS, and PS/2 - Don't let incompatible diskette formats get you down, read them all with your PC.

UniForm-PC by MicroSolutions
How often have you wished you could use your CP/M diskettes on your PC? Now you can access your CP/M disks and programs on your MS-DOS computer just as you would a standard MS-DOS diskette. Once the UniForm driver is installed, you can 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, or AT. With UniForm-PC and the Compaticard, you can use 5¼" high density, 96TPI, dual format 3½" (720k/1.44 meg.- PS/2), and even 8" drives.

UniForm-PC by MicroSolutions ..... $ 64.95
Uniform for Kaypro and other machines ..... $ 64.95

CompatiCard by MicroSolutions
Meet the CompatiCard, THE universal disk drive controller card. This half card will let you run up to 16 disk drives (4 per CompatiCard) on your PC or XT, including standard 360K, 96 TPI, high density (1.2 meg, dual speed), 8" single or double sided (SD or DD), and dual format 3½" drives (720k/1.44 - PS/2). The combinations are almost unlimited. Comes with its own MS-DOS driver and format program for high density and 3½" diskettes. Use it with UniForm-PC for maximum versitility. 8" adaptor and additional cabling available.

CompatiCard Board $169.95
CompatiCard with UniFORM-PC.. ★★ **Special** ★★.. $225.00
CompatiCard with UniFORM-PC & high density or
3½" drive ..... ★★★ **Special** ★★★ $350.00

MatchPoint-PC by MicroSolutions
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 a copy of the UniForm-PC program, as well as utilities to format disks, copy, delete, and view files on Apple DOS, PRODOS, and Apple CP/M diskettes.

MatchPoint-PC Board $169.95

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

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

Frustrated because your PC can't speak CP/M?

UniDOS by Micro Solutions
Run CP/M programs on your PC? Of course. UniDOS is a memory resident program that can use the NEC V20 CPU chip to actually RUN your favorite 8080 programs. Use UniDOS with UniForm-PC, and automatically switch to CP/M mode as you log on your CP/M diskette. Switch to emulation mode to run Z80 code programs or for systems without a V20. UniDOS directly converts video and keyboard emulation for Kaypro, Xerox 820, Morrow, Osborne, VT100, and eight other displays. All standard CP/M system calls are supported. Note: The NEC V20 CPU is a fast, low power, CMOS replacement for the 8088 CPU chip that includes a full 8080 instruction set as well as the standard 8088 set. Systems using an 8086 may substitute a V30 chip.

UniDOS by MicroSolutions ..... $ 64.95
UniDOS w/UniForm and V20-8 chip $135.00

UniDOS Z80 Coprocessor Board by MicroSolutions
This **8 Mhz. Z80H** half-card will run your Z80 and 8080 code programs at LIGHTNING speed on your PC or AT. Functions just like the UniDOS program, except NO V20 or emulation mode is required to run your programs. Now includes UniForm-PC!

UniDOS Z80 Coprocessor Card $169.95

Delving Into The Black Arts:

Build A PC Adapter Card In An Hour, Or So

Your boss wants a specialized monitor/controller, so you look through the catalogs for an off-the-shelf solution. None. You check the yellow pages for an off-the-shelf controller designer. None. Why not do it yourself? Design your own adapter card, plug it into a PC... (You've never designed a PC adapter card? Read on then, read on.)

Building a PC adapter card for a computer backplane (or slot) is a mysterious and marvelous art. And, to say the least, one misunderstood. But its beauty lies in its simplicity.

With the tools I'll discuss in this article, you can have your first PC adapter card working in an hour or so. I'll show you a generic bus interface and a sample circuit for digital I/O. Then we'll write a communication program in Turbo C.

The sample circuit is generic so you can easily replace it with a design of your own.

Tools

Two tools make building a card almost effortless. The first is a PC bus prototyping card with super-strips fastened to it. You just push the chips and wires into the holes. No soldering or wire-wrapping necessary.

All the bus signals are brought out to special labeled connector blocks, and there's a DB-25 connector for the outside world.

A number of places market cards like this. I chose mine from a picture in a catalog, but I think it was about the nicest, albeit most expensive, at $79.

The second tool is very useful, especially if you have a fairly full PC. It's called a card extender, and it lifts one slot of the bus out above the others; so you can plug your prototype in and test it without having to reach down into the narrow (and dangerous) confines of your PC. It also has test points for all the PC bus signals, so you can easily clip a probe onto any bus line.

Vector made mine, but you can get similar ones from JDR Microdevices and Active.

A Plethora Of Busses

The computer bus actually consists of several busses, each a group of wires —

- The data bus carries the data back and forth between the processor and memory or peripherals.
- The address bus selects the memory cell or peripheral device the processor is talking to.
- The control bus tells the processor or peripherals what's going on (this is a read from memory, this is a write to an I/O device, an interrupt has happened, stop what you're doing, I need the bus...etc.).

Some computers, in particular PCs, have a group of wires for the DMA (Direct Memory Access) controller, a special chip dedicated to transferring data very quickly from one location (memory or I/O) to another.

The bus is a fundamental, but often confusing, part of every computer. Some computers (PCs, Apple IIs, S-100 machines) have parts of the bus brought out to a "backplane" or "slots," which allows you to plug in hardware cards to customize your system.

The backplane is sometimes also called a bus, but it often doesn't contain all the lines the main board bus does (e.g., the PC). It can be confusing when someone says, "this processor is 8 bits" or "this is a true 16-bit system." Not only is there a backplane bus and a main board bus, but a bus inside the microprocessor.

The Microprocessor's Internal Bus

The number of bits in a processor refers to the width of the data bus, width of the registers, and the width of the arithmetic/logic unit (i.e., the chunk of information the processor handles at once).

The sample circuit is generic so you can easily replace it with a design of your own.

Width is one indication of speed: if two processors are running at the same clock rate, the 16-bitter should process information more quickly than the 8-bitter.

When Intel brought out the 8088 (the CPU running things in the slowest clones), 8-bit peripheral chips were common but 16-bit peripheral chips were rare. To keep computer designers comfortable, Intel created the 8088 with a 16-bit internal architecture, but the chip worked with an 8-bit external data bus. That way designers could use the common 8-bit peripherals.

Since the 8088 manipulates 16 bits at a time, its bus interface unit grabs two bytes, one at a time, for every (16-bit) operation. (The bus interface unit is the only real difference between the 8088 and the 8086, which has a 16-bit window to the outside.) The 8088's 8-bit window slows things down a bit. An 8086 is a true 16-bit processor, an 8088 is a sorta 16-bit processor.

The Bus Delivers

If there are 16 address lines (as in a Z80), the processor may select up to 2^{16} distinct memory locations, or 64K. To find the number of different values a group of wires can

By Bruce Eckel
Eisys Consulting
1009 N. 36th Street
Seattle, WA 98103

number of states each wire can be in and raise it to the power of the number of wires.

The 8088, for instance, has 20 address wires and each wire can be in one of two states: 2^{20} = 1 Megabyte. An 8-bit data bus can represent 2^8 = 256 different quantities (counting from 0-255; zero is a legitimate state).

The PC Backplane

Figure 1 shows a PC card connector as it appears after you've uncovered your computer and peered down into its bowels. The descriptions on each pin are fairly self-explanatory, but some could stand further elucidation.

The only lines we'll need for our design are address, data, I/O read and write, and address enable. So you don't need to learn everything now. But when you take on something more complicated, you'll need to consider the rest of the bus lines.

Interrupts

There are eight interrupts available on the PC/XT.

- Interrupts 0 and 1 are used by the motherboard (for the timer and

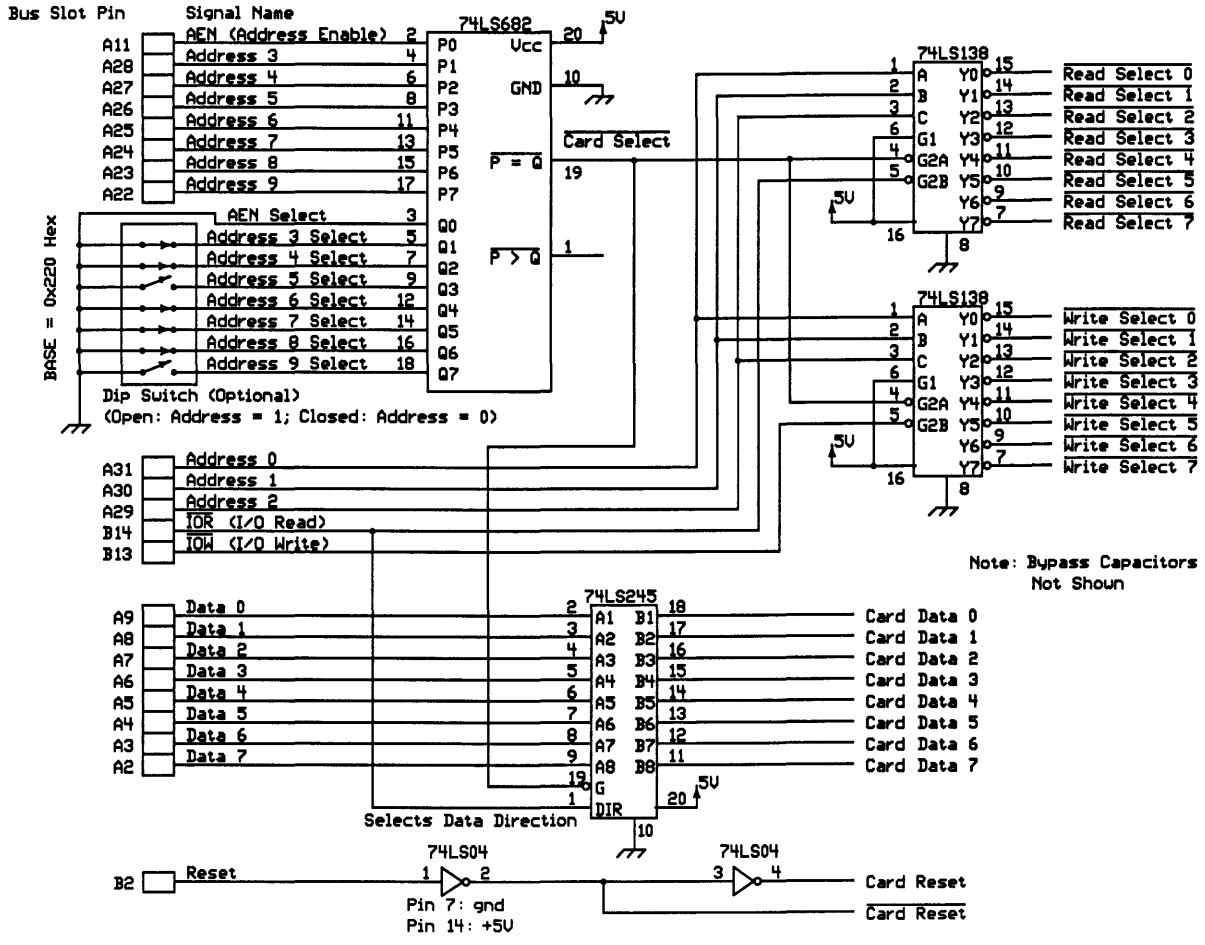
Figure 1— PC/XT Bus Slot

Rear Panel

Ground	B1	A1	$\overline{\text{I/O CH CK}}$	I/O Channel Check (From Card)
Reset (To Card) RESET DRV	B2	A2	SD7	Data 7
+5 Volts DC	B3	A3	SD6	Data 6
Interrupt Request IRQ 2	B4	A4	SD5	Data 5
-5 Volts DC	B5	A5	SD4	Data 4
DMA Request From Card DRQ2	B6	A6	SD3	Data 3
-12 Volts DC	B7	A7	SD2	Data 2
Zero Wait State (From Card) 0WS	B8	A8	SD1	Data 1
+12 Volts DC	B9	A9	SD0	Data 0
Ground	B10	A10	I/O CH RDY	I/O Channel Ready (From Card)
Card Memory Write $\overline{\text{SMEMW}}$	B11	A11	AEN	Address Enable (DMA has control)
Card Memory Read $\overline{\text{SMEMR}}$	B12	A12	SA19	Address 19
Card I/O Write $\overline{\text{IOR}}$	B13	A13	SA18	Address 18
Card I/O Read $\overline{\text{IOW}}$	B14	A14	SA17	Address 17
DMA Request Acknouledge $\overline{\text{DACK3}}$	B15	A15	SA16	Address 16
DMA Request From Card DRQ3	B16	A16	SA15	Address 15
DMA Request Acknouledge $\overline{\text{DACK1}}$	B17	A17	SA14	Address 14
DMA Request From Card DRQ1	B18	A18	SA13	Address 13
Memory Refresh Occurring $\overline{\text{REFRESH}}$	B19	A19	SA12	Address 12
System Clock From Motherboard CLK	B20	A20	SA11	Address 11
Interrupt Request IRQ 7	B21	A21	SA10	Address 10
Interrupt Request IRQ 6	B22	A22	SA9	Address 9
Interrupt Request IRQ 5	B23	A23	SA8	Address 8
Interrupt Request IRQ 4	B24	A24	SA7	Address 7
Interrupt Request $\overline{\text{IRQ 3}}$	B25	A25	SA6	Address 6
DMA Request Acknouledge $\overline{\text{DACK2}}$	B26	A26	SA5	Address 5
DMA Terminal Count Pulse T/C	B27	A27	SA4	Address 4
Buffered Address Latch Enable BALE	B28	A28	SA3	Address 3
+5 Volts DC	B29	A29	SA2	Address 2
14.31818 Mhz Clock From Motherboard OSC	B30	A30	SA1	Address 1
Ground	B31	A31	SA0	Address 0

Top View

Figure 2— Generic PC Adapter-Card Interface



keyboard) and are not brought out to the backplane.

Of the interrupts 2 through 7 on the backplane —

- IRQ6 is used for disk controllers,
- IRQ3 & IRQ4 are used for serial ports,
- IRQ7 is used for the printer port.

IRQ2 and IRQ5 are the safe ones for experimentation (unless you have a bus mouse, which uses one of these two).

DMA

Of the four Direct Memory Access (DMA) channels in the PC/XT, two are used by the system: Channel 0 (which isn't brought out to the backplane) for memory refresh and channel 2 for disk transfer. The other two channels are available. (On a vanilla PC/XT, for example, DMA can transfer close to 1/2 Megabyte per second.)

A considerable number of lines on the backplane support DMA.

The DRQ lines allow a board to initiate a DMA transfer. The DMA logic (on the motherboard) receives this signal and puts the processor "on hold" at an appropriate spot.

The DMA controller drops the DACK line, causing the card to put its data on the bus. The DMA controller then puts the memory address of the destination on the address bus, and yanks on the control lines to shoot the data directly into the destination. The 8088 never touches it. It's spilled-oil slick and *very* fast.

While the DMA controller messes with the address and control lines, other cards on the backplane could unwittingly think they're being talked to. To prevent this, the AEN line goes high when the DMA controller is using the bus. So we need to include the AEN line when decoding the card's address (see Figure 2).

Clocks

Two clocks (CLK & OSC) connect to the backplane. The speed of the system clock (CLK) depends on your computer. On a two-speed XT (4.77 and 10 MHz), CLK will be 4.77 MHz or 10 MHz.

The OSC signal always runs at 14.31818 MHz, regardless of the machine (ATs included). This can be useful if you always want the same response no matter which computer the card is plugged into. But OSC isn't synchronized with the system clock CLK, so you can't use it for any data-transfer circuits.

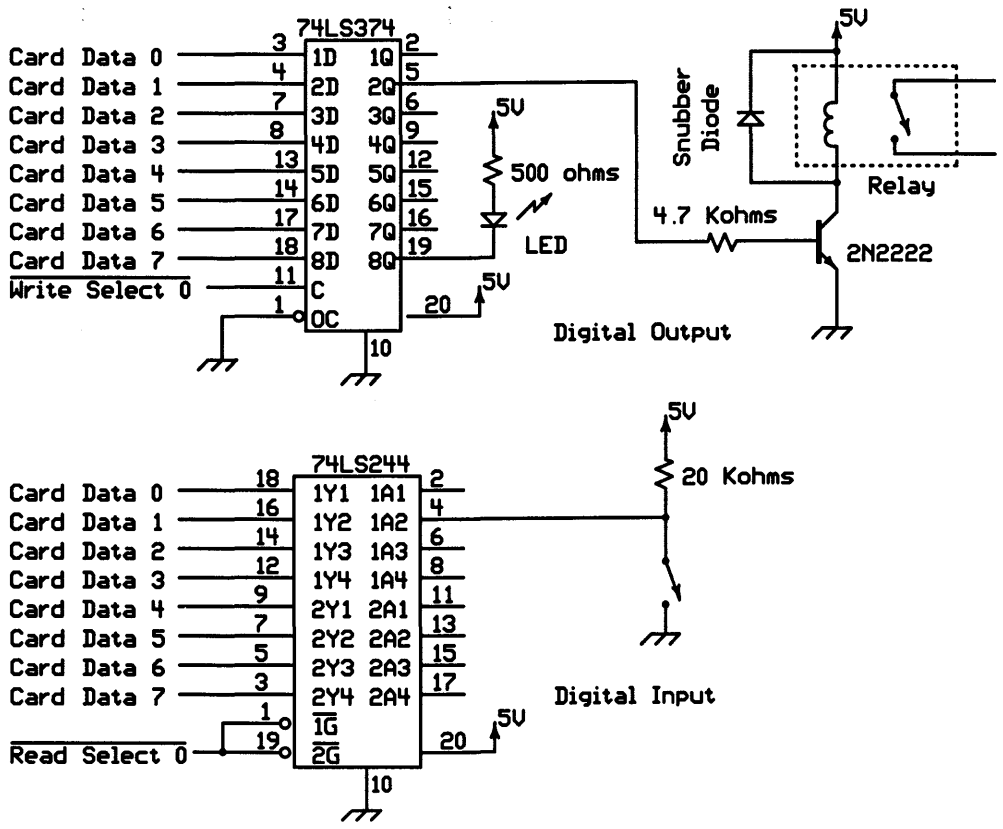
I/O

Although the 8088 is capable of addressing 64K of I/O (using the bottom 16 address lines: 2^{16} = 64K), the PC only brings 10 of these lines out to the backplane (thus, limiting the backplane to a 1K I/O space). Some of this space is already used by the motherboard, and some is reserved for serial ports, display adapters, etc.

When experimenting, it's important to choose an I/O address which doesn't conflict with any other card. Some safe chunks of space are 0x220 - 0x277, 0x280 - 0x2CF, and 0x320 - 0x35F. In Figure 2 I chose 0x220.

When we read an I/O port, the IOR

Figure 3—Digital Output and Input



(I/O read) signal is low. When we write it, the IOW (I/O write) signal is low (memory reads and writes have their own signals). That's all there is to it — watch the I/O control lines to see if we're reading or writing (and the AEN line to make sure it isn't the DMA), and you're in business.

Assorted Chaff

A few other lines on the backplane serve odd purposes.

I/O Channel Check and I/O Channel Ready allow the card to tell the motherboard if something is wrong. If a card raises the 0WS line, it means there's zero-wait-state memory on the card.

REFRESH indicates a memory refresh is in progress. T/C indicates the end of a block DMA transfer.

BALE is the 8088's Address Latch Enable (ALE) signal, brought out to the backplane to give an indication of the state of the CPU (the address and data are already separated by the time they get to the backplane). But, generally you don't need to worry about these signals.

Decoding A Block Of Addresses

Figure 2 shows a general-purpose circuit for decoding a block of eight read and write addresses and buffering the data bus between the card and the backplane. This requires only four chips (plus a 74LS04 if you need the reset signal). The circuit will work for virtually any card design, and is overkill for many.

The 74LS682 determines if an address is meant for the card by looking at address lines 3-9 and AEN. This chip is an address comparator: if its "P" inputs are equal to its "Q" inputs, it asserts the "Card Select" signal, telling the rest of the chips, "the show is on."

The 74LS682 also has pull-up resistors on its "Q" inputs to pull any open lines up to logical 1. Ground the address lines you want to be zero, and leave the "ones" open. The dip switch on the diagram shows the configuration for base address 0x220.

Notice that the Q0 line is wired to ground. To prevent card selection when DMA is active, the AEN signal must always be low.

You can see in Figure 2 that the card select goes to the "G" (enable) inputs of the other three chips. It's the master decision maker.

Addressing Within The Block

The 74LS138s are 3-to-8 decoders. If you put a 3-bit binary number in at A, B, & C, the corresponding "Y" line will drop. If, for example, address lines 0, 1, and 2 are all at logical zero, Y0 (Read Select 0) will drop. The "G" lines are called qualifiers; the selected line will drop only if the qualifiers are correct.

In Figure 2, the CARD SELECT line connects to G2A on both 74LS138s. The upper 74LS138 is also qualified with the I/O READ line, and the lower one with the I/O WRITE line. Thus, the upper chip only selects when the card is selected and an I/O READ is occurring. The lower chip acts the same for an I/O WRITE.

Driving The On-Card Bus

The 74LS245 is an "Octal Bus Transceiver." It boosts the signal from the backplane onto the card, and from the card onto the backplane. This chip is also "qualified" with the CARD SELECT signal into the "G" pin.

The DIR (direction) pin, pin 1, tells the chip whether the signals should be flowing from left to right or right to left. The I/O READ signal connects to the DIR pin.

When I/O READ (which is active low) is low, we're reading, and the 74LS245 gates the data bus from the card to the backplane. When I/O READ is high, we're writing, so the data bus is

gated from the backplane to the card.

Reset

Many chips require a reset when the power is turned on so they'll start out in a known state. You should buffer (boost) the reset from the board whenever it goes onto a card. I do this in Figure 2 with 74LS04 inverters.

Bypass Capacitors

I haven't shown bypass capacitors on the diagram. These should have values between 0.01 and 0.1 microFarads and connect from power to ground near several of the chips.

When chips are working they draw short surges of current from the 5V supply. These surges cause dips in the supply voltage, and if the dips get large enough they'll cause problems for other ICs on the same 5V line. Small capacitors connected between power and ground provide a quick supply of electrons (current), thus minimizing the dips. (If you want to know more about this, check my description back in *Micro C* Issue #36, June-July, 1987.)

A Simple Application

Figure 3 shows a simple application for the generic adapter card interface in

Figure 4 — Demonstration Program for Expansion Card

```
/* Generates the Battlestar Galactica "Cylon Eye" effect on the
LEDs connected to the digital output port, while reading and
displaying the input port. */
#define BASE 0x220
/* selected via 74LS682 (see figure 2) */

void print_binary(unsigned char byte) {
/* show the ones and zeroes in a byte */
  int i;
  for ( i = 7; i >= 0; i--)
        /* The following uses C's "ternary expression," which
        returns the value after the '?' if the expression is true
        or after the ':' if it is false. */
        putch(byte & (1 << i) ? '1':'0');
}

main() {
int i;
do{
  for(i = 0; i < 8; i++) { /* scan up */
        outportb(BASE, ~(1 << i));
        /* 'delay()' is a function from Turbo C 1.5 to wait for
        a number of milliseconds. If you have version 1.0, write
        a simple "for" loop. */
        print_binary(inportb(BASE)); putch('\r');
        /* I use a carriage return with no linefeed to stay on
        the same line. */
        delay(40);
  }
  for(i = 7; i >= 0; i--) { /* scan back down */
        outportb(BASE, ~(1 << i));
        print_binary(inportb(BASE)); putch('\r');
        delay(40);
  }
} while(!kbhit()); /* loop until a key is pressed */
}

END OF LISTING
```

Figure 2. The upper chip (74LS374) is an 8-bit digital output port, and the lower chip (74LS244) is an 8-bit digital input port. As you can see, there isn't much to either one — you simply hook up the data bus and the read or write select line.

I'll be honest: a better choice for the output chip is a 74LS273, but I didn't have one handy and I did have the 74LS374. The pinouts are the same, but the '273 has a reset on pin 1 where the '374 has an output control. Thus, on my card the digital outputs come up in an unknown state, something which isn't always desirable.

Figure 3 shows some simple digital I/O circuits. On the '244, a switch with a pull-up resistor will read simple on-off conditions. We need the pull-up resistor to make sure the chip reads a "1" when the switch is open.

The demonstration program, PC_CARD.C (see Figure 4), makes a simple back-and-forth pattern on eight light-emitting diodes, one of which I've shown on Figure 3.

The other circuit shows an inexpensive NPN transistor (you can buy 2N2222s in quantities of 50 for under $20 from Active) used as a switch to drive a relay coil. This particular relay was rated at 5 Volts. The resistor on the base limits the base current and keeps the transistor from burning out.

The diode (any one will do) gives the inductor current somewhere to flow when the transistor turns off (thus preventing the inductive kick from punching a hole in the transistor).

Parts

#PB-88/2 from Global Specialties; 1-800-345-6251. JDR Microdevices (EXT-8088, $30, 1-800-538-5000) and Active (#56093, $34, 1-800-343-0874) carry all the parts used in this article.

Editor's note: Bruce has mentioned other simple data-acquisition and control circuits (which you can adapt to this card) in other articles. Some of these are available via back issues of Micro C. All are available directly from Bruce. A book of his articles and a disk of all the programs from the articles is $30 from: Eisys, 1009 N. 36th Street, Seattle, WA 98103.

♦ ♦ ♦

Fractal Miscellany

On To EGA

The last time Larry putzed around with fractals, he shut down the editorial department for three weeks. (How can we write when all our computers are fractaling?) This time was better. He was producing four-minute fractals (rather than four-day fractals), and he did them on a borrowed system. Who knows, the rest of us may learn to appreciate weird graphics, too.

My clone has spent a good portion of its life cranking out close ups of various portions of the Mandelbrot set — that wonderfully complex entity pictured on the cover of *Micro C* Issue #39 (Jan.-Feb. 1988). In that issue I talked about the Mandelbrot set and how it could be generated on a computer.

I finished with a brief discussion of techniques for speeding up the sometimes days-long process. Since then several readers have sent in their own efforts at faster fractals. Of course, you could always cough up the bucks for a 386 system and use Harlan Stockman's code (see his "Fast Fractals" article in this issue). But some of you, like me, prefer to spend your hard earned money on junk food and sleazy arcade games.

So, here are a few methods guaranteed to free up some computer time for more important pursuits. We'll also talk about a problem with the screen save/restore function from Issue #39 and go over CGA and EGA fractals.

We're Working Too Hard

The first, and most obvious, time saver has to do with the value of max_iterations — the maximum number of feedback loops for each point on the screen. It turns out that, unless you have a very high resolution graphics board, 500 iterations or so does the job quite well. I originally used 1000 iterations — overkill for Herc and EGA systems.

So points which used the full max_iterations (i.e., those within the Mandelbrot set) took twice the time they should have. Cutting back to 500 iterations doesn't necessarily halve the execution time, but it does speed things up appreciably. The time saved will depend on how much of the screen lies inside or close to the set.

Swear off Twinkies and Jolt Cola for a week and spend the savings on a coprocessor

A picture of the full Mandelbrot set (Pmax = 0.5; Pmin = -2.0; Qmax = 1.25; Qmin = -1.25) takes 4 hours 2 minutes to generate at max_iterations = 1024. That's using Turbo C on a 10 MHz XT clone with EGA, V20 and an 8087 math coprocessor. (I used this system for all comparisons in this article.) Dropping to max_iterations = 512 gave a time of 2 hours 4 minutes with no real loss of resolution.

8087

I know it ain't cheap, but an 8087 will turn your fractal programs into real screamers. Swear off Twinkies and Jolt Cola for a week and spend the savings on a coprocessor — your stomach will thank you, your processor will thank you, your boss will thank you.

I zoomed back to speed things up (Pmax = 10.0; Pmin = -10.0; Qmax = 10.0; Qmin = -10.0; max_iterations = 512). Without the 8087 (yawn), this Mandelbrot slug took 92 minutes 30 seconds to slime its way across the screen. Whittle that down to 4 minutes 25 seconds, or 20 times faster, with an 8087 in the picture. Semi-impressive, I'd say.

Of course this comparison also tests the quality of Turbo C's floating point library. But Turbo C finished first out of ten compilers in a test of software floating point execution speed (see Scott Ladd's C review in *Micro C* Issue #40 Mar.-Apr. 1988). So we're testing the 8087 against the best 8088 floating point code in the MS-DOS C compiler market.

The 8087 code above was generated by Turbo C with its 8087 option switched on. Dan Haney sent in some 8087 assembly code that gives even more impressive results — 2 minutes 49 seconds.

Admittedly, we're doing a little apple/orange comparison here. Dan linked his assembly code to an Eco C vers. 3.2+ Mandelbrot program. But Turbo C beat Eco C in all of Scott's benchmarks except display speed. So Dan's 8087 assembly code with Turbo C would have sped things up even more.

Dan also noted (along with several other readers) a bottleneck in the fractal functions from Issue #39. A fair amount of unnecessary floating point math took place. Figure 1 shows an improved mandel() function. The time saved depends only on the number of pixels on the screen. Both 4 minute and 4 hour fractals ran about 10 seconds faster on the EGA system I used. Not a tremendous improvement, but it's always nice to code more efficiently.

I made one other change in mandel(). It no longer plots the set upside down (this was *so* embarrassing).

Speaking of bottlenecks, floating point operations present a big one. Fractal programs are obvious choices for fixed point math. Again, take a look at

Larry Fogg
Micro C Staff

Figure 1 — New and Improved Mandelbrot Function

```
#define sqr(X) (X*X)

void mandel ()
{
  float Pmax, Pmin, Qmax, Qmin;
  int color, row, col, max_iterations, max_size;
  float P, Q, modulus, deltaP, deltaQ,
        Xcur, Xlast, Ycur, Ylast;

  get_params (&max_iterations, &max_size);
  get_mandel_params (&Pmax, &Pmin, &Qmax, &Qmin);
  deltaP = (Pmax - Pmin)/max_col;    /* real axis increment */
  deltaQ = (Qmax - Qmin)/max_row;    /* imaginary increment */
  P = Pmin;                    /* start with left-most point */
  for (col = 0; col <= max_col; col++)
  {
    Q = Qmax;                  /* start with top point in column */
    for (row = 0; row <= max_row; row++)
    {
      Xlast = Ylast = modulus = 0.0;
      color = 0;
      while ((modulus < max_size) && (color < max_iterations))
      {     /* go until function blows up or max_iterations */
        Xcur = sqr (Xlast) - sqr (Ylast) + P;     /* next Z */
        Ycur = 2 * Xlast * Ylast + Q;
        color++;
        Xlast = Xcur;                        /* update last Z */
        Ylast = Ycur;
        modulus = sqr (Xcur) + sqr (Ycur);/* find size of Z */
      }
      draw_point (col, row, (color % max_colors));
      Q -= deltaQ;             /* determine Q for next point */
    }
    P += deltaP;               /* determine P for next point */
  }
}  /* mandel */

END OF LISTING
```

Harlan's article and also Earl Hinrich's graphics piece in *Micro C* Issue #41 (May-June 1988).

Color Fractals

A number of folks have asked about fractal code for their CGA and EGA systems. Figure 2 shows part of the code affected by the change of graphics modes. Since we're now using graphics adapters *officially sanctioned* by the Blue Meanies, BIOS calls exist for most functions. (We had to roll our own functions for the Hercules card.)

Might as well use BIOS calls for reading and writing individual pixels. They're painfully slow but, in this case, we don't care. Filling an entire screen using BIOS calls takes 2 minutes 30 seconds longer than talking directly to the EGA controller — a long time if you're doing interactive stuff, but a blink of the eye in fractal land. And the BIOS function is much simpler.

Gary showed us a function in Issue #39's *Tidbits* column that bypasses the BIOS for EGA pixel writes. (See Figure 3.) The code executes perfectly and quickly, but absolutely defies understanding.

Take a look at Sam Azer's PCX article in *Micro C* Issue #42 (July-Aug. 1988) for a bunch of good information on the EGA. Sam used the same pixel write routine that Gary did with the comment that, "I don't have the foggiest notion how it works." (Hey, that's *my* line.) But, just for fun, let's try to make some sense of this bizarre function.

EGA Mysteries

It would be nice to just write directly to video memory when we want to change the screen. This technique works well for Hercules and CGA systems, but EGA's a different kettle of fish. Its four bit planes of video memory won't fit into the PC's 64K worth of video address space, at least not all at the same time. So the processor can't directly address video memory and we're forced to rely on the EGA video controller — definitely a step better than the BIOS, but still frustrating.

In Figure 3 (the code in question from the excellent *Advanced Graphics in C* by Nelson Johnson), base gives the byte offset of the memory location containing the point we want to write. Then mask shows which bit within the byte corresponds to that point.

Since all four bit planes map into the same 64K address space, the point address actually locates four bits, one in each of the bit planes. And four bits means 16 possible colors for the point.

No problem so far. But what's this exist_color? We assign a value to it and then ignore it. Well, it turns out that the EGA controller sees memory accesses within its address space. A read of the byte pointed to by base actually loads four bytes (one for each plane) into registers in the controller. exist_color is just a dummy variable to allow that read.

With the current color of the pixels loaded in the controller, four port writes set the color, enable the controller, specify a logical combination of the old and new color, and set the mask for a particular pixel out of the eight possibilities.

Finally, a dummy assignment to the location pointed to by base gets picked up again by the controller. It then writes

new values to EGA memory at base, depending on the old values and on how we've loaded the registers.

The last four port writes put the controller back to sleep. They aren't necessary if you're just going to plot another point. And some BIOSs apparently do them as part of the mode change back to text. But, to be on the safe side, knock out the controller when you're done plotting.

Figure 3 also shows a get_point function — again from Johnson.

CGA

Not much to say about CGA. It's the simplest of the group, and the least satisfying. But hey, I like simple. We can do everything except screen save and restore with BIOS calls. And the screen stuff is a trivial change from the Herc versions.

About That Screen Save...

In Issue #39 I used screen save/restore functions of dubious integrity. They worked when they felt like working (much like myself). The problems came from the pointer to video memory and the type of file I/O (should have been binary).

Figure 4 shows new and improved Herc/CGA screen functions which work well (really). So you can stop cussing me out every time you get a screen full of dog poop after recalling 2 1/2 days worth of fractal efforts.

I'd dearly love to find a way to get at EGA memory directly for screen save, but it looks tough. I know it can be done, though. A demo disk of the Tekmar Graphics Library from Advanced Systems Consultants (818-407-1059) made a believer of me. The demo lets you watch as each of the bit planes reads into memory. They're obviously selecting and writing a complete plane at once.

A slow and dirty screen save can be done by reading individual pixels with either BIOS calls or video controller programming. (See Figure 5.) Here's a case where use of the video controller makes good sense — an impossibly slow process becomes merely ponderous.

A BIOS screen save takes 2 minutes 50 seconds (beer break!). The video controller version executes in 50 seconds. The two versions of screen restore run in 2 minutes 50 seconds (BIOS) and 45 seconds (controller). Obviously there's room for improvement here. Tekmar restores a screen in just a few seconds. If I ever figure out the method, I'll let y'all know.

I put a quick effort into decreasing a screen's file size. Since only four bits

Figure 2 — BIOS Graphics Functions

```
#include <dos.h>

#define text 3                          /* text mode for CGA/EGA */
#define graphics 16  /* 16=highest EGA res, use 6 for CGA */
/* sets EGA to 640X350 16 color or CGA to 640X200 2 color */
/* change max_col, max_row, and max_colors accordingly    */

void set_mode (int scr_mode)            /* sets video mode */
{
  union REGS r;

  r.h.al = scr_mode;                     /* load registers */
  r.h.ah = 0;
  int86 (0x10, &r, &r);               /* call video interrupt */
}  /* set_mode */

void draw_point (int col, int row, char color)
{
  union REGS r;

  r.h.ah = 12;                         /* draw point function */
  r.h.al = color;
  r.h.bh = 0; /* set page 0 for EGA (unnecessary for CGA) */
  r.x.dx = row;
  r.x.cx = col;
  int86(0x10, &r, &r);                /* call video interrupt */
}  /* put_point */

char get_point (int col, int row)
{
  union REGS r;

  r.h.ah = 13;                         /* read point function */
  r.h.bh = 0; /* set page 0 for EGA (unnecessary for CGA) */
  r.x.dx = row;
  r.x.cx = col;
  int86 (0x10, &r, &r);               /* call video interrupt */
  return (r.h.al);                   /* returns color of point */
}  /* get_point */

END OF LISTING
```

Figure 3 — EGA Pixel Write/Read Without BIOS

```
#define TRUE 0xff
#define ENABLE 0x0f
#define INDEXREG 0x3ce                      /* controller registers */
#define VALREG 0x3cf
#define OUTINDEX(index, val)  {outp(INDEXREG, index);\
                               outp(VALREG, val);}
#define EGA_BASE 0xa0000000L        /* base address for EGA memory */
#define WIDTH 80L

void draw_point (x, y, color)
int x, y, color;
{
  unsigned char mask = 0x80, exist_color;
  char far *base;

  base = (char far *)(EGA_BASE + ((long)y * WIDTH + ((long)x / 8L)));
  mask >>= x % 8;                        /* locate point within byte */
  exist_color = *base;        /* dummy assignment to read video mem */
  OUTINDEX (0, color);              /* set up controller registers */
  OUTINDEX (1, ENABLE);
  OUTINDEX (3, 0x18);                    /* XOR with existing color */
  OUTINDEX (8, mask);
  *base &= TRUE;              /* dummy assignment to write video mem */
  OUTINDEX (0, 0);                                /* disable controller */
  OUTINDEX (1, 0);
  OUTINDEX (3, 0);
  OUTINDEX (8, TRUE);
}  /* draw_point */

char get_point (x, y)
int x, y;
{
  int color;
  unsigned char mask = 0x80;
  char far *base;

  base = (char far *) (EGA_BASE + ((long)y * 80L + ((long)x/8L)));
  mask >>= x % 8;                      /* locate point within byte */
  OUTINDEX (5, 0x8);                        /* select read mode 1 */
```

```
  outp (INDEXREG, 2);
  for (color=0; color<=0xf; ++color)
  {          /* test pixel against each color until match found */
    outp (VALREG, color);
    if ((*base & mask) == mask) break;
  }
  return (color);                    /* return the matching color */
}  /* get_point */

END OF LISTING
```

Figure 4 — Herc/CGA Screen Save and Restore

```
const int screen_size = 0x8000;          /* for Herc, 0x4000 for CGA */
const char *screen = 0xb0000000L; /* for Herc, 0xb8000000L for CGA */
#include <fcntl.h>                       /* has file type definitions */
        /* hard-coded for page 0 on the Herc. CGA only has page 0 */

void save_screen (char fname [13])                /* write to disk */
{
  FILE *f;

  _fmode = O_BINARY;                     /* set up binary file write */
  f = fopen (fname, "w");
  fwrite (screen, screen_size, 1, f);
  fclose (f);
}  /* save_screen */

void get_screen (char fname [13])                 /* read from disk */
{
  FILE *f;

  _fmode = O_BINARY;                      /* set binary file read */
  f = fopen (fname, "r");
  fread (screen, screen_size, 1, f);
  fclose (f);
}  /* get_screen */

END OF LISTING
```

Figure 5 — EGA Screen Save and Restore

```
#include <fcntl.h>                       /* has file type defintions */
#define line_size 320     /* # bytes in one line of mode 16 screen */

void save_screen (char fname [13])                /* write to disk */
{
  FILE *f;
  int row, c;
  char line_buf [line_size];     /* 1 graphics line, 2 pixels/byte */

  _fmode = O_BINARY;                     /* set up binary file write */
  f = fopen (fname, "w");
  for (row=0; row<=max_row; row++)
  {
    for (c=0; c<line_size; c++)      /* get the two pixels per byte */
      line_buf [c] = (get_point (2*c, row) << 4)
                   + get_point (2*c+1, row);
    fwrite (line_buf, sizeof (line_buf), 1, f);   /* save the line */
  }
  fclose (f);
}  /* save_screen */

void get_screen (char fname [13])                 /* read from disk */
{
  FILE *f;
  int row, col;
  char line_buf [line_size];     /* 1 graphics line, 2 pixels/byte */

  _fmode = O_BINARY;                      /* set up binary file read */
  f = fopen (fname, "r");
  for (row=0; row<=max_row; row++)
  {
    fread (line_buf, sizeof (line_buf), 1, f);      /* get one line */
    for (col=0; col<line_size; col++)
    {                                        /* draw the first pixel */
      draw_point (2*col, row, line_buf [col] >> 4);
                  /* mask the first pixel and draw the other one */
      draw_point (2*col+1, row, line_buf [col] & 0xff);
    }
  }
  fclose (f);
}  /* get_screen */

END OF LISTING
```

define each pixel, we can shift one pixel into the high nybble of a byte and stick another pixel in the low nybble. That cuts the file size in half to a slim, trim 112,000 bytes.

More could be done by taking advantage of the fact that adjacent pixels often have the same color (see Johnson). But I'm a week past deadline and Cary will have my head (or something even more important) if I drag this out any longer.

Next Time

Mikey (a fellow fractal fanatic and all around good guy who gave a rather chaotic talk at SOG VII) and I recently braved the mountain roads and traveled to Oregon State University to hear a talk by Professor G. D. Chakerian of U. C. Davis. He spoke on infinite sets, fractals, and other mind boggling topics.

Editor's note: Ignore the Chakerian stuff, Larry came back muttering something about larger infinite sets vs. smaller infinite sets. The rest of you don't need to have your minds boggled, too.

We collared the good Professor after his presentation (and after loading up our pockets with cookies for the return trip) and asked the big question: "Who's actually *using* fractals for anything other than pretty pictures?"

His response confirmed our suspicions. He had read numerous nebulous references to fractal applications but couldn't lead us to first sources describing those applications. It seems that fractals have a lot in common with the weather: everyone talks about them, but no one *does* anything about them.

Next time around I'll try to dispel this conviction that there's no such thing as a practical fractal.

And many thanks to Micro C's buddy Alan Chambers who, in the face of mutiny by son David, donated his EGA system to the cause. You really weren't expecting it back, were you Alan?

Editor's note: You'll find 8087, Hercules, CGA, and EGA fractal programs (source included) on our Issue #43 disk for $6. Or check in the CURRENT ISSUE area of the Micro C RBBS.

♦ ♦ ♦

ANSI, The Grand New C

By Scott Robert Ladd
P.O. Box 61425
Denver, CO 80206
(303) 322-7294

Scott concentrates, this issue, on the numerous additions the ANSI committee has made to the C language. Plus, he updates us on the new versions he's received and the new versions he's heard rumors about.

This column is going to be another grab-bag, just like the last one. While there have been no new compiler releases, other products of interest have been piling up on the sofa in my office. So let's dig in...

New C BBS Phone Number

Wouldn't you know it — just as issue #42 of *Micro C* was going to press, which contained my column mentioning Barry Lynch's C BBS, Barry moved! The board's new phone number is (703) 440-0240. Such is life...

K&R 2nd Edition

Wandering through a bookstore a few weeks back, I ran across the new edition of Kernighan and Ritchie's classic book, *The C Programming Language.* This is an updated version of the work known among C veterans as "K&R." I commend Kernighan and Ritchie for avoiding the tendency these days toward 1000-page tomes; the second edition is only slightly larger (at 270+ pages) than its predecessor.

So what's new? Primarily, the second edition follows the draft standard being developed by ANSI (the American National Standards Institute).

The original examples have been refined, and new examples have been added. The discussion of pointers has been rewritten and expanded. Some ambiguities in the text of the original have been clarified. There are also some new utility programs which (for example) convert complex declarations to human-readable text; these are an excellent aid in learning a somewhat confusing subject.

For those of you still using UNIX or K&R (first edition) standard compilers, Prentice-Hall has promised to continue publishing the first edition of K&R until "compilers based on the ANSI standard are universally available."

I highly recommend this book to anyone who seriously programs in C. The cover price is high ($29 to $32, depending on where you buy it) for such a "skinny" book, but it still is one of the clearest expositions of features and uses of the C language.

What Is ANSI C?

A good question! While the standard isn't complete, most of the primary details have been hammered out. While some compilers adhere to the standard, many do not. Some vendors are waiting for the standard to be finalized, an event which probably won't occur until late this year.

Let's take a look at some of the things which have been changed or added with the ANSI standard. I'll try to explain why these changes were made, and how to use them.

The ANSI committee's goal is to officially standardize the C language, based on current programming practice. The standard will promote the portability of source code from one compiler (and environment) to another. Also, the committee is attempting to solve some of the problems with K&R C, such as a lack of type checking in function calls.

C is well-known for having very few reserved words (keywords) — only 27 in the original K&R version, as opposed to over 200 in a modern dialect of BASIC. ANSI has added 5 more, for a total of 32. The new keywords support new or enhanced data types.

const & volatile

Two new keywords are "const" and "volatile," both of which are qualifiers. When a data item is declared as const, the compiler will not allow its value to be changed. This can make the final object code more efficient — for instance:

```
#include "stdio.h"

float const pi = 3.1515927·
main()
```

```
{
float diameter;
printf("PI = %f",pi); /*pi*/
diameter = 10.0 * pi; /*pi*/
}
```

In a traditional program, pi would be put into a #define directive, like "#define pi 3.1415927," and a separate constant would be created in the program's data area for each reference to pi (some compilers are smart enough to make only one constant for identical references, but this is not true in most cases).

Using const, only one copy of pi is generated, and that single copy will be accessed by both references to pi. And, the compiler won't allow any statement which changes pi's value.

Declaring a variable to be volatile indicates that it can change at any moment, asynchronously. For example, a global variable which is accessed by an interrupt service routine would be declared volatile to prevent the compiler from making any assumption about its value.

void

A new type has been implemented, called "void." Something declared to be of type void indicates that it has no values. (For instance, some criminals are of type void.) An example of this would be a function which does not return any data.

You can declare a pointer to void; it isn't a pointer to nothing, but rather a pointer to anything. Any type of pointer can be assigned to a void pointer, and vice versa, without a cast. Functions such as malloc() return a void pointer under ANSI C.

enum

ANSI C has added the "enum," or enumerated, type. An enumerated data type is used to define a special set of related integer (int) values. For example:

```
enum rankings {first, second,
               third, fourth};
```

The names listed in "rankings" are actually integer constants, with first set equal to 0, second to 1, third to 2, and fourth to 3. A variable of a specific enumerated type may only be assigned an integer constant already defined for it. For instance:

```
enum rankings cur_rank;
cur_rank = first;
```

Actually, cur_rank is an integer data item set to 0 (the value associated with first).

You can also specify special values for enumeration constants, as in

```
enum coins {penny=1, nickel=5,
            dime=10, quarter=25};
```

If you declare a variable to be of type coins then that variable can only take the values listed. Not only can enumerated types make your program more readable, but they can also prevent you from assigning invalid values to variables.

Prototype

In my opinion, the most valuable ANSI enhancement to C is the function prototype. A prototype is a function declaration which includes information on its return and parameter types. Prototypes for library functions are in standard header files, while prototypes for functions in the current module are placed near the beginning of the program. This is an example prototype:

```
char *fputs(int, FILE *);
```

When the compiler sees a call to a function declared with function prototype, parameters passed to the function will be checked to be certain they are of the correct type. K&R C did no parameter type-checking, allowing (for example) an int value to be passed as a pointer (or vice versa). This can lead to very subtle errors which are hard to track down.

The above program line makes sure that any calls to fputs have two parameters, an integer and a stream (file) pointer. Type-checking is also done on the returned value to make sure it is assigned to a pointer to char.

Note that type-checking can be circumvented by using explicit casts.

The void data type can be very useful in function prototyping, as in these examples:

```
void *malloc(unsigned);
void init(void);
```

In the first example, malloc is declared as a function with a single (unsigned-type) parameter, which returns a generic (void) pointer. Init is declared to have no parameters and no return value.

Function definitions also have undergone some changes. A classically-defined main() function might look like this:

```
int main(argc argv)
int argc;
char *argv[];
   {
   /* program statements */
   }
```

whereas under ANSI, it could be defined as:

```
int main(int argc, char *argv[])
   {
   /* program statements */
   }
```

I'm not sure which is clearer. The second version of main() looks a lot like a Pascal function definition. The choice of how to define a function is left up to the programmer.

These are just a few of the new features the ANSI committee has added. Other new items include structure passing and the standardization of header file names and contents. With all these new features, though, nearly every program written under the K&R standard will compile without problem under the ANSI standard.

Thinking Of MS-DOS...

I almost went into shock when an entire month went by without *any* new C compilers showing up at my door. After the deluge at the beginning of the year, things seem to have slowed down a bit. Slowed, but not stopped.

Power C Bug

MIX Power C *still* has a few bugs. Version 1.1.1 (which is the latest as of this writing) has probably the dumbest bug I've yet encountered in a compiler. Any statement which combines integer constants and floating point numbers will crash the optimizer pass!

Adding decimal points to the constants solves the problem; however, a C compiler is supposed to do an automatic cast of an integer constant to a float in these cases. While I still think Power C is a good buy (for the manual alone), this type of bug indicates they're not paying attention to details.

Microsoft C 5.1 And Quick C

While Power C comes on a pair of diskettes, Microsoft C 5.1 arrives on (catch this) 14, two of which are 1.2

megabyte high-density disks! This is because of the support for OS/2, which necessitates duplicate versions of some of the programs. The compiler generates both MS-DOS (real-mode) and OS/2 (protected mode) executables.

In the last column, I mentioned that the NEC V-20 microprocessor can execute instructions for the Intel 80186. Many compilers (including Turbo C and Microsoft C) can generate this kind of code, which is slightly faster than that for the 8086/88.

Unfortunately, Microsoft's compiler assumes that you have an 80287 math coprocessor if you compile for the 80186. The V-20 is always installed in an 8088-based machine, and will have an 8087 rather than the 80287. This problem only occurs when using the default floating-point option (/FPi) which generates inline instructions which are used by either a coprocessor or the emulator.

So, if you have a program which contains floating point calculations, you need to compile using the /FPc option, which tells the compiler to generate CALLs to the math library as opposed to inline instructions.

There's another (though minor) bug in the printf and cprintf functions. When displaying a floating point number declared with a precision of 0 (zero), these functions show leading spaces rather than leading zeros.

Microsoft has released a patch for this bug, and I have uploaded it to the Micro Cornucopia RBBS as MSCPRINT.ARC (you'll need to use PKXARC or an equivalent to extract the patch files from the archive).

Benchmarks

Several readers have commented about the benchmarks I use to compare MS-DOS C compilers. While most of the comments have been favorable, others have not.

One reader suggested that the benchmarks could be made much better by tailoring them to each individual compilers' capabilities. He showed that by tweaking the benchmarks, Let's C's performance improved.

This is a valid point. Any program (benchmark or not) will run faster when it takes advantage of the strengths of the compiler and the hardware.

Unfortunately, there are two problems with this.

First, there's more to software development than program runtime.

Second, most programmers don't know enough about their compilers to take advantage of "faster" methods. What is fast for one compiler may be slow for another.

I have found in the past that no matter how much it looks like one hardware type and compiler will be used for a project, there is *always* the possibility of porting the code. Portable programs are written to be independent of operating system, hardware, and compiler dependencies.

Editor's note: Also, that next version of the same compiler may require a completely new set of tricks for maximizing code speed.)

As I've said before, benchmarking is an arcane art. I am examining the current benchmark suite, with an eye toward improving it. By the next issue, I should have made some minor changes and added a new benchmark program, Whetstone. Whetstone is a large floating point benchmark usually used for testing Fortran compilers. Also, a few alert readers found a bug or two in the benchmark programs, which I will fix.

For now, new tables *have* been uploaded to the Micro Cornucopia RBBS. They include results for WATCOM C, Power C, Microsoft C 5.1, and Quick C 1.01.

Did You Register?

No, I'm not talking about the draft... but about those little cards you get with software in order to tell the company you've bought their product. According to Greg Lobdell at Microsoft, only about 30 percent of the cards for programming languages are returned.

It sometimes takes me a month, but I *always* make sure the cards get sent back. Why? Because I want to know about upgrades. Often, a registered user can get a special price on upgrades. Also, upgrades are not generally available to nonregistered users.

In any case, the only disadvantage to registering your software is that you'll get occasional pieces of advertising junk mail (which at least don't have Ed McMahon's picture on them). So remember to return those cards!

Rumors, Rumors

It's not a very well-kept secret that Borland International is working on symbolic debuggers for their Turbo C and Turbo Pascal compilers. What *is* a secret is when they'll be releasing them, and what they'll look like. They won't even tell *me* (obviously hiding from the media [grin]).

Next Time

Next issue, I'll talk about C debugging techniques (which are generic to any C programming environment) and discuss the pros and cons of the current generation of "source-level, symbolic" debuggers for MS-DOS. Other subjects will include third-party libraries from Blaise, public domain libraries, and a product known as risC (a C-like macro assembler).

Elsewhere in this issue, I'm talking about building database programs using C and third-party function libraries under MS-DOS.

Talk To Me...

I look forward to hearing from the readers of this column. If you have questions, or want to see me cover a given subject, please feel free to drop me a letter, call, or post a message on the Micro Cornucopia RBBS.

May you program in interesting times...

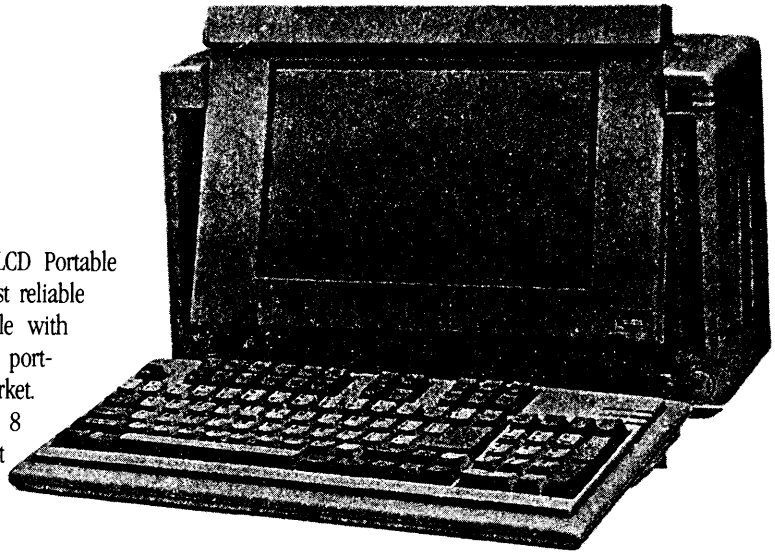
♦ ♦ ♦

McTek

LCD Portable

(Including Hard Disk Only 19 lbs.)

RABBIT 286



The McTek Rabbit-286 LCD Portable combines the fastest, most reliable AT motherboard available with most visible full-size LCD portable screen on the market. Running at a switchable 8 or MHz Ø wait state, it includes a 20MB hard disk, 3½" floppy drive, parallel & serial ports, Award 3.01 bios, 640k, turbo indicator LCD & mouse interface. The screen is a fantastically readable, electroluminescently backlit, 80-column by 25-line LCD with adjustable intensity and screen-angle. It's as readable as a CRT. You can also plug in a digital or analog color monitor or a digital or composite monochrome monitor. Included also is an external 5¼" floppy port for reading and converting to 3½" disks (5¼" external drive w/case: $179 when purchased with LCD Portable). The McTek Rabbit-286 LDC Portable comes fully assembled with our one-year parts & labor guarantee, and sells for an amazing, complete price of only ***$1799!***

3 MB On-Board AT!

Our McTek 286A is the most integrated AT-compatible to date. It utilizes the highly regarded Chips & Technology chip set, and includes memory upgradable *on board* to 3 megabytes. No more worries about speed compatibility with expanded memory cards! The 8/10 MHz, Ø-wait state McTek 286A runs at 11.5 Norton SI, and an effective 13.2MHz on the Landmark test. Serial, parallel & game ports are all standard on board. With Award 3.01 bios, 640k, 200W power supply, Samsung amber monitor with Hercules-compatible controller, locking case, AT-style keyboard, 1.2MB drive, 20MB Seagate. Assembled & fully tested, with a full one-year warranty. Get in on the most advanced AT-compatible on the market, at the lowest price ever offered! ***$1399!!***

XT Turbos & Supers

640k 4.77/8MHz and 4.77/10 switchable XT turboboards; two 360k floppy-disk drives with controller; one parallel, one serial and one game port: AT-style keyboard; clock, FCC-approved slide-case; eight slots; Hercules-compatible graphics card; amber monitor w/base; fully assembled and tested; one-year parts *and* labor warranty.

$599 **XT Turbo 4.77/8MHz Complete**

Superturbo 4.77/10 MHz Complete ***$659***

McTek Systems, Inc. • 1411 San Pablo Avenue • Berkeley, CA 94702 • 415-525-5129

McTek

*Prices subject to change without notice.

DISK DRIVES

Fujitsu 360k	$72
Fujitsu 1.2MB	$95
Teac	$79
Teac 1.2MB	$99
Toshiba 3½" 720K	$99
Toshiba 3½" 1.4MB	$129
Floppy controller	$22
20MB Hard Disk Kit	$279
30MB Hard Disk Kit	$309
ST-225	$215
ST-238	$239
ST-138 30MB	$399
ST-251 40MB	$369
ST-125 20MB 3½"	$279

PRINTERS

Citizen CD 120	$159
Citizen CD 180	$189
HPLASAR Serial2	$1699
Epson LX-800	$219
Toshiba 321 XL	$559
NEC P2000	$369
Call for prices of other brands	

MODEMS

Everex int. 300/1200	$79
Everex 2400 external	$195
Everex 2400 internal	$179

MONITORS

Samsung amber	$79
Samsung EGA color	$359
Samsung RGB color	$259
NEC Multisync	$559
Sony Multiscan	$619
HGC-compat.mono card	$49
Color graphic card	$49
EGA Paradise 480	$149
VGA Paradise	$279
Genoa Super VGA	$499

MOUSE

Logimouse C7	$69

PC/XT

640k TurboMothrbrd	$85
10MHz TurboMothrbrd	$89
Multi I/O w/disk contrlr	$59
640k RAM card	$39
2MB Expansion card	$115
RS232 2-port card	$35
4-serial port card	$79
Game I/O card	$15
384k Multifunction card	$69
FCC-app. slide XT case	$29
150W power supply	$49
XT keyboard	$42
Clock Card	$19

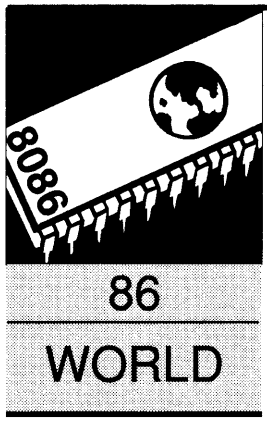
PC/AT

McTek286 6/8/10/12MHz	$279
Baby McTec 286B-AT 10/13 O-wait	$289
McTec 286A O-wait 3MB 4 ports on board	$379
3MB Multifunction card	$125
2MB Expansion card	$125
Multi I/O card	$59
Locking slide case	$59
200W power supply	$69
Enhanced keyboard	$59
WD HD/floppy controller	$129

MISC.

Kingtech CRT Portable Kits: XT/AT (power supply, case keyboard, monitor)	$380/$410
Eprom burner 4-socket	$139
LCD Portable	$799
AC power center	$25
AC power strips	$15
Diskette file box	$9
Printer or serial cable	$8
386 16MHz 2MB	$1999
286 16MHz 1MB	$999
Archive Tape Backup 40MB	$299

Reader Service Number 42



AT Block Memory Moves

And Laine Reads The Repertoire Manual Over Dinner

By Laine Stump
% Redhouse Press
Merkez PK 142
34432 Sirkeci
Istanbul, Turkey

Our boy made it again, this time in an express package all the way from a Turkish diner. His system is down, but that doesn't keep him from enjoying his dinner.

I'm totally freaked out to be sitting in front of a screen, looking at multiple windows of text, and dragging things around with a mouse again. I've just been on the road for nearly a month, with at least a week and a half in the middle where I was sleeping in tents and under the stars (and for the rest I was miles away from my computer, anyway). The idea of having 10 Megs of free space is suddenly alien to me. I can't understand the significance of "Mouse Driver Installed" anymore.

Boatingwise, we didn't spend as many days on the river or go to as many rivers as I had originally planned. But adventurewise, I got more than I bargained for. A complete synopsis of my trip would take up too much space in the magazine (and wouldn't fit in very well with the theme anyway) (well, kind of...) but I'm sending it on disk to Micro C with this column and, if they know what's good for them, they'll put it somewhere on the Micro C bulletin board.

(Editor's note: Okay, Laine, if we must. It's in the CURRENT ISSUE area.)

So what am I going to talk about then? I had planned on discussing OS/2 and Logitech's OS/2 version of Modula 2. But unfortunately, my copy of OS/2 hadn't arrived before I left the U.S. (and although the OS/2 Modula compiler will run under DOS, the programs it compiles won't).

Instead, I'll be talking about a database - user interface - natural language recognition - list management toolbox from a company called PMI. While that's going on, who can tell what other muddled meanderings you'll get dragged into...

Hard Luck Story Of The Month

As a matter of fact, I think I'll start off with a little tale about hard luck with floppy drives. I mentioned in the last issue that I had bought a portable box from McTek and stuffed it with one of PC Tech's new X24 boards (you know, 12-16 Mhz 286, up to 4 Megs on board, etc., etc.).

Before I left the U.S. everything seemed just peachy keen with it (we were still stuffing parts in the box as I was running out the door to the airport). But after I arrived back in Istanbul, strange errors started appearing.

At first it was just one disk. When I tried to read from the disk, certain files gave read errors. Just a few in the beginning, then more and more. I suspected the diskette, but when I plugged it into the Zenith 181 it read with no problem. Okay. Must be a hardware problem. Let's take the box apart and look at the cables. Maybe something has jiggled loose, or the drive connector wasn't crimped on very well.

So I took the back cover off and removed the metal box containing the floppy and winchester drives. Fiddled around with the cables and it still didn't work. Maybe it's the floppy drive.

I removed the floppy drive from the metal box and replaced it with a spare Toshiba that happened to be sitting on the coffee table. Glory be! It works!

So I put the whole thing back together, cussing all the while about idiot suppliers who sell defective equipment. Turned it on and — "Disk Read Error on A:"!

What did I do to deserve this? I've always believed in the concept of karma ("What goes around, comes around.") I may not be any kind of a saint but, you know, every airplane I've been on since last fall was delayed due to technical difficulties. And the last two times I was on the double decker bus from Ankara to Istanbul the bus either wrecked or broke down. And I won't even mention the incident at the waterfall.

Then I took the machine back apart. It worked! Maybe it was just a loose cable; I put it back together. "Disk Read Error"! Hey, this is interesting. I did some experiments. I moved the floppy in and out of the case while doing drive accesses. Guess what? When the

floppy and winchester are too close to each other, the radio emissions from the winchester turn the data from the floppy into junk.

Next step, I turned over the floppy drive and put the two drives back into the metal frame. It worked with no problem. Almost. There were no mounting holes in the correct position to mount the floppy that way. I decided to turn the winchester over instead.

After prying the winchester out of its tight fit in the metal box, I restarted the machine. It wouldn't boot! I booted from a floppy and tried to access a few files on the winchester. "Disk Read Error on C:"! WHAT HAVE I DONE!!!!

My mind spinning wildly out of control with thoughts of hard head crashes and 300 bucks down the tubes, I tried to remain calm and analyze the situation. It was working five minutes ago and now its not. What did I do to it in the last five minutes? Well, I pried the drive out of the box and turned... Wait! What was that word? Did I say "pried"? That's the ticket!

I removed the winchester and shoved it back in facing the opposite direction. With an extra squeeze into the cramped space for good measure, I flipped the switch and, wonder of all wonders, it worked! Seems the tracks had moved out of alignment when the drive wasn't in such a tight spot.

"Great," I said to myself, "I'll just back everything up, turn the drive over, and do a low level format. Now where did I put Disk Manager?" A quick look through my collection of some 75 3.5 inch diskettes and 30 5 inch diskettes failed to turn it up. Oops.

So my wonderful little "portable" machine spent four days in a thousand pieces on the table while I waited for a copy of Disk Manager to arrive from a friend in Ankara.

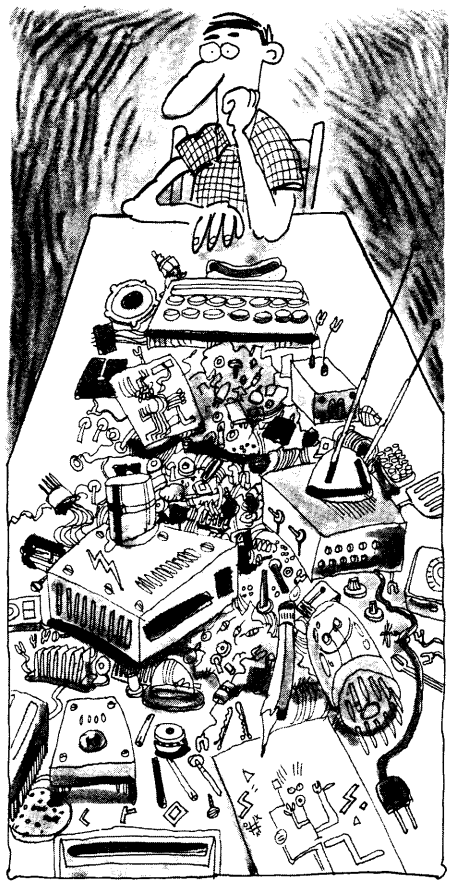
Just tonight I finally got it all put back together, but I've learned my lesson(s): Never mount two disk drives with their circuit boards facing each other. And never, never, NEVER install a winchester (or a floppy drive) in a position where it has to be torqued to fit the holes.

Oh yeah. And give offerings to Buddha next time you're at your local 1st Orthodox Temple.

Extended Memory

I'm having so much fun avoiding the so called "main topic" of the month that I think I'll continue doing so for awhile. (Besides, my mind is still confused about what to write about Repertoire. It's a toolbox of such mammoth capabilities that I don't know how I'm going to condense it all into one column.) Let's talk about using IBM AT extended memory then.

So my wonderful little "portable" machine spent four days in a thousand pieces on the table . . .



When the IBM AT was first released, there was no software (with the exception of some RAM disk and similar programs) that could take advantage of memory above the normal 640K. Eventually XENIX-286 and UNIX were released, but that still didn't solve the problem for those of us running under MS-DOS. The AT was being used as a fast XT.

What nobody told us (at least they didn't shout it from the top of the Galata Tower) was that the AT ROM BIOS has always had a function which moves data from the space above one meg down into the normal DOS memory space and vice versa.

It's not as fast or as convenient as using EMS but, what the heck, if somebody in your purchasing department went out and bought an AT with 2 megs of RAM because the dealer told him it was macho, you may as well write some software to take advantage of it. Right?

Personally, I hadn't given this much thought (seeing that I had always considered AT compatibles a waste of money). But one day during my winter apprenticeship at PC Tech, Dean came into the room and asked if I could write a program which accessed extended memory so that he could do some tests on the 16 megger board.

I told him that I didn't know squat about protected mode and that it sounded like a several-day project, trying to figure out segment descriptors and all that for the first time. But somehow I found out that INT 15h, subfunction 87h, moves blocks of data between any two arbitrary 24 bit addresses in the AT's 16 Mb address space. Perfect.

The actual program was really boring, so I won't trouble you with it here. Instead I'll just give you the macro which I wrote that will move a block of data NumWords long from Src to Dst (see Figure 1). You can use this in assembly language programs to take limited advantage of extended memory.

There are only a few points worthy of note here. First, remember that all addresses are 24 bit linear addresses! That means that if you want to use an address that's already in segment:offset form, you'll have to do some converting.

This is the formula: copy the segment register to another register and shift it right by twelve bits. This will be the high part of the address. Now make another copy of the segment register, shift that value left by four bits and add it to the offset to get the low part of the address. Finally, add the carry of this addition to the high part of the address.

Make sure that the descriptor table shown in the listing is in an active segment, otherwise you'll send INT 15h a junk descriptor and you'll get an error.

When the macro is finished executing, the CARRY flag will be set if there was an error during the move. Look in the AT Technical reference manual for a list of these errors.

A short example of using this macro appears in Figure 2.

Repertoire

When I was in college, I spent several weeks designing and writing a screen and input management package for Pascal. My only reason for writing it was so that I could use it in a cardfile type program I was writing for the University Honors Program office. If only Repertoire had existed then. If only I had known.

Repertoire is a collection of modules which take care of sophisticated screen output, input and validation of entire screens of data (including variable length scrolling text fields and virtual screens larger than the physical screen), indexed file management of true variable length records and arbitrarily complex linked lists, natural language recognition, and generalized list management, among other things.

Repertoire is written and sold by a company in Portland called PMI. I had seen their ads several times and they had caught my eye. Then one day I received their brochure.

"Blending art and science for software construction in the spirit of the Renaissance" it said on the cover. Hey! This sounds like my kind of company!

I called them and ended up talking to one of the authors. After some chit chat about new Modula compilers, OS/2, and incompatible ATs, I had a package winging its way to Lake City, hoping it would reach me before I reached the Pan Am check-in counter.

Fortunately, it arrived before I left the U.S. Fortunately I didn't put it in the bag where the bottle of chocolate sauce exploded (my *OS/2 Reference* and *Learn to Speak Minnesotan* weren't so lucky). Unfortunately, I had already sent in the column for issue 42, and I was just about to spend a month driving through the mountains and eating rice with road-kill stew. The books sat patiently on the shelf in Istanbul, awaiting my return to reality.

The first book I reached for when I got back to Istanbul (other than *The Scarlatti Inheritance* by Robert Ludlum) was *The Repertoire Reference* (that sounds like a good Ludlum title, too). The more I read, the more excited I became. In one line of code I could do things that had once taken days.

But again, I was crippled by my "bad luck." Since my machine wasn't working quite right, I wasn't able to copy the libraries onto the winchester and try it out. So I can't give you any idea of performance. I will give some comments about the conception, documentation, and capabilities, though.

First, a warning. This is a very positive review. Don't get the idea that I work for PMI (although that might be fun). I don't. I've just never before seen such a complete, well designed package for such a low price ($89). Besides, I haven't really gotten in deep yet. Knowing me, I'm sure I'll find plenty of things to bitch about as time goes on.

Figure 1 — MOVMEM Macro

```
DATA    SEGMENT PUBLIC  ;this structure is used by MOVMEM
GDT     DQ      0       ;DUMMY
        DQ      0       ;points to GDT
        DW      0FFFFh  ;limit
SRCOFS  DW      (?)     ;low word of base
SRCSEG  DB      (?)     ;high byte of base
        DB      93h     ;access rights
        DW      0       ;reserved
        DW      0FFFFh  ;limit
DSTOFS  DW      (?)     ;low word of base
DSTSEG  DB      (?)     ;high byte of base
        DB      93h     ;access rights
        DW      0       ;reserved
        DQ      0       ;points to virtual code segment
        DQ      0       ;points to virtual stack segment
DATA    ENDS
MOVMEM  MACRO   SrcHigh,SrcLow,DstHigh,DstLow,NumWords
        MOV     AX,SrcHigh
        MOV     SRCSEG,AL
        MOV     AX,SrcLow
        MOV     SRCOFS,AX
        MOV     AX,DstHigh
        MOV     DSTSEG,AL
        MOV     AX,DstLow
        MOV     DSTOFS,AX
        MOV     AX,DS
        MOV     ES,AX
        MOV     SI,offset GDT
        MOV     CX,NumWords
        MOV     AH,87h
        INT     15h     ;cassette interrupt, extended move function
        ENDM
```

Figure 2 — Example Of Calling MOVMEM

```
; Source is at ES:SI Destination is at 00200000h (linear 2Meg)
        MOV     AX,ES   ;convert ES:SI to linear address
        SHR     AX,12
        MOV     BX,ES
        SHL     BX,4
        ADD     SI,BX
        ADC     AX,0    ;now linear address in AX+SI
        MOVMEM  AX,BX,0020h,0,8000h  ;copy 8000h words (64k bytes)
        JNC     short NoFail
        SHR     AX,8
        ; note: PRINTM_NL is a routine from Earl's library
        ; It is available on the X16 source diskettes
        PRINTM_NL       <'Error %02ux while initializing'>,AX
        JMP     short Continue
Nofail1:PRINTM_NL       <'initialization sucessful'>
Continue: ...
END OF LISTING
```

Reader Service Number 36

Reader Service Number 37

Documentation

The Repertoire documentation starts with an overview of the entire package, followed by a chapter on each "system" (my name for a group of interdependent and related modules).

These chapters give many useful examples (one of them was a multiwindow text editor written in one page of Modula). They also discuss how to "glue" the various systems and modules together. For example, there is a section on moving data records from the screen display system into the indexed file system.

After this "User's Guide" section is a "Reference Manual" (these labels are mine, not PMI's) consisting of a listing of each definition module as well as an English description of what each procedure in the module does and how it is called.

Finally come the appendixes which, among other things, list the dependencies of modules. This is important when you modify one of the modules and need to recompile.

Oh. Did I forget to tell you? The $89 price includes source code. All of it. Nothing hidden. As a matter of fact, they even include suggestions in the manual on changes you might want to make, and what piece of code you should look at to make the modification.

Another reason they give for including source is to protect application programmers in case they need to move their programs to different operating systems. This sounds like implicit approval to port Repertoire to other machines.

I enjoy documentation that doesn't leave you in the dark about details, but still manages to get the main point across. This manual was enjoyable reading over a dinner of pide (an elongated pizza-like thing) and lentil soup at the local cafe. Happily, in Turkey they never bring the bill until *you* tell *them* you're ready to go.

Capabilities

I was originally interested in Repertoire for the file system which handled variable length records (nothing is more variable in length than a definition in a dictionary). I hadn't even considered the rest.

Screen System

The screen system is very complete. You define screens in a text file which contains a "picture" of each input screen as well as information about each field. Fields can be strings, integers, fixed point, or floating point. A field can also be a linked list of arbitrary length which will be scrolled within its own little window inside the main record's window. Word wrapping in these text windows is handled automatically. You can also define "choice" fields which let you construct Lotus-type bar menus.

Indexed File System

The indexed file system can truly store just about any data structure you throw at it. Not only that, but each record within a file can be constructed differently (type information is stored along with the data elements). Records can be retrieved by number or by name. There are also procedures to build a list of record names based on selection criteria passed in a human readable string.

Once you have the name of a record, it only takes a single disk access to read it into memory. This is because the entire index is kept in a linked list in memory. Of course, keeping the index

in memory puts limits on the maximum number of records in a file.

You can get around the problem by storing each record as a sublist of a smaller number of main records (made possible by the ability to store variable length, arbitrarily constructed records), but that will slow down performance.

Another solution is to buy an EMS board and link your program with PMI's EMSStorage module (a separate package that sells for $49). At any rate, the practical limit for a machine with 640K is somewhere between 10,000 to 15,000 records.

Other Modules

There are several other useful modules in Repertoire. Most of them are the things you would expect to see in a screen/database package. But one module which really stands out as something significant is the GenLists module.

GenLists contains a collection of procedures for complete management of (as always) arbitrarily structured linked lists. These procedures include NewList, ListInsert, ListDelete, SortList, CopyList, JoinLists, SplitList, ListSize, DisposeList ... Well, you get the picture.

The type of data stored in any element of a list is determined by a tag field stored along with the element. Having a tag field enables you to have a list where each element is a different data type. One of the possible types is GenList (i.e., you can have lists of lists — or lists of lists of lists). The tag field is a cardinal, so there are plenty of values left for your own (user defined) types.

Since all access to elements of a list is done through the procedures ListInsert and GetElmt, GenLists can optimize lots of things. One example is that strings stored in a GenList only take up the amount of space required for their current length (not the maximum length). Another optimization is that a pointer to the last accessed element is saved, and there will be no time wasted chaining through the list if the next access is to the same element.

PMI claims that list intensive applications can reduce their program size up to 30% by centralizing all list handling. I don't know about that. But I am certain that program complexity can be reduced by much more than that if you leave the details of dereferencing pointers in a binary tree to a central list manager. My only regret at this point is that I don't have more need for writing "list intensive" applications.

Royalties

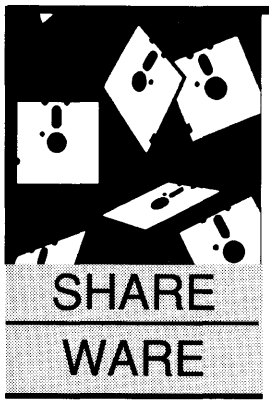
"Sure", you say. "They make the library cheap. That's just to suck you in so you'll have to pay them big royalties when your program starts selling." Not true. PMI has a strict policy of NO ROYALTIES for applications constructed using PMI modules. They even have a plan where, for a one time fee of $689, you can bundle all or part of Repertoire's UNLINKED object code with your programs.

This would be useful for people selling tool kits similar to Repertoire, or programs requiring too much modification to actual code to be distributed in EXE form. This offer is valid even if you are a "present or potential competitor."

Contact:

PMI
Cole Brecheen
4536 SE 50th
Portland, OR 97206
(503) 777-8844

♦ ♦ ♦



Program Prophylactics

Anthony Barcellos
P.O. Box 2249
Davis, CA 95617-2249
(916) 756-4866

If you think humans have all the fatal problems, you might check into the viruses that can infect your computer. After all, how would you feel if some bug destroyed your fat? (On second thought...)

Is your PC practicing safe computing? News accounts and rumors abound with reports of deadly viruses eager to corrupt your programs and destroy your hard disk's directory. Is it hype, or should you now forbid your system to pick up strange young files at its favorite bulletin board?

Sharing The Blame

The viruses are out there all right, but there aren't as many as the outcry might indicate. Viruses have become the scapegoats for all kinds of computer disasters. But scrambled directories and wiped-out hard disks are anything but new.

First of all, not every bug is a virus. Yet the consequences may be similar. Novice computer users (and some experts, even) learn the hard way that conflicting RAM-resident software can do funny things to your operating system — up to and including a complete file system wipeout.

Don't be surprised. Programs like SideKick and Metro may fight for control of your system interrupts. Sometimes it's a fight to the death. So if something funny happens to your computer, at least consider the possibility that there's a conflict induced by your TSR (Too Stupid to Run) programs.

Even programs that aren't RAM-resident can do funny things under certain circumstances. In a previous life as a California civil servant, I was the office computer guru responsible for the care and feeding of hard disks. I soon had the staff using CHKDSK/F to recover lost clusters after using a modeling program that produced half a megabyte of waste space after each work session. And that was an expensive econometric package that had presumably been carefully debugged before its release.

In brief, TSRs and program bugs can trash your system without the helpful intervention of a malevolent virus. One computer user wryly declared that the new OS/2 operating system may make viruses unneeded. Its earliest versions have already wiped out a number of hard disks.

Anatomy Of A Bug

Now that I've made the point that viruses are neither endemic nor at the root of all our directory mishaps, let's talk about the viruses that are really out there. If they proliferate, we may eventually be cut off from one of the microcomputing community's most valuable resources: the shareware and public domain programs that populate our electronic bulletin board systems.

As with biological viruses, a computer virus must be able to commandeer the resources of its host and replicate itself. A favorite virus target is COMMAND.COM on MS-DOS computers. The virus attaches itself to the executable file and thereafter copies itself into every other copy of COMMAND.COM it encounters.

If all a virus did was copy itself, that would be no big deal. However, the classic evil computer virus contains a counter that increments every time it replicates. After having created enough copies of itself, the virus destroys itself and the contents of the disk on which it resides. By then, of course, its "children" have scattered and are busy making copies of themselves on other systems.

To some degree, today's viruses are escaped combatants from the Core Wars game, the program vs. program competition made famous in the pages of *Scientific American* via A. K. Dewdney's *Computer Recreations* column. Core Wars was Dewdney's featured topic in the May 1984, March 1985, and January 1987 issues. If you're into heavy symbolism, you'll appreciate that the January 1987 issue featured a cover story on the AIDS virus.

Safe Computing

How can you protect your data from com-

puter viruses? Try the same procedures that prevent Trojan programs from getting at your hard disk. (Remember Trojans? These less-sophisticated miscreants from pre-virus days work their mischief the first time they're run and don't replicate.)

Run any new program on a floppy system a few times before trusting it to a hard disk system. After your first usage, check COMMAND.COM to see if its file size or time stamp has been altered. If so, reformat the floppy with a pristine copy of DOS to overwrite the corrupted COMMAND.COM, and discard the program that did it.

Download your shareware or public domain programs from BBSs that screen their uploads. Check with the software librarian of your local users group. If the library's master disks come directly from the authors, the programs will be less likely to have encountered viruses.

Several "vaccination" programs have now appeared to help you combat viruses. Some of them are unwieldy RAM-hungry guardians that cause about as much trouble as they cure, but others may suit your needs. Dig out the April 5, 1988, issue of *PC Week* for a handy list.

One of the most interesting examples is Sophco's Vaccinate, which is a "good" virus designed to head off bad ones before they can infect your files. (Sophco is in Boulder, Colorado, at (800) 922-3001 or (303) 444-1542.) Shades of Core Wars!

Writing Is Fundamental

There are also a couple of possibilities in the public domain. One old program, with documentation dated October 29, 1985, is CHK4BOMB. "Check 4 Bomb" has an elegant design: It examines an executable file for commands that would cause a disk-write. After all, unless it writes to your disk, a virus cannot infect your system.

Editor's note: CHK4BOMB is on the Micro C RBBS and on the issue #43 disk. To purchase the disk call 1-800-888-8087. $6 postpaid, foreign add $2.

As an example, I ran CHK4BOMB on FORMAT.COM from PC-DOS 3.1. CHK4BOMB echoes all ASCII text to the console, enabling you to read any embedded messages (unless somehow scrambled). In general, you might be suspicious of programs which contain the ASCII string "Ha ha sucker!!!"

Even more important, CHK4BOMB examines all function calls to see if a program writes to disk. Here's an abbreviated version of the utility's evaluation of FORMAT.COM:

CHECKING FOR BOMBS AND ASCII CHARACTERS IN FILE FORMAT.COM.
Note that some machine code will print as ASCII characters and appear as gibberish . . . other ASCII strings in the program will be readable. Most programs have the code first, followed by data.
CHECKING 9398 BYTES
Microsoft,Inc ibmbio com0ibmdos com0
IBM 3.1
Non-System disk or disk error
Replace and strike any key when ready
Disk Boot failure
IBMBIO COMIBMDOS COM
******WARNING**** This program uses the ROM BIOS routines for direct disk access. This program could format a disk or write to certain sectors without updating the directory or File Allocation Table. DO NOT RUN this program until checked by an expert, unless you are familiar with the author or company.**
******WARNING**** This program writes to absolute sectors. The pos-**

sibility exists to overwrite important data!
<END OF FILE> 9398 Bytes in file were read.

I routinely run CHK4BOMB on every executable program I download. Simply enter "chk4bomb" at the DOS prompt, followed by the name of the file to be examined.

If you want a copy of CHK4BOMB, look around for the archive CHK4BOMB.ARC. It should contain CHK4BOMB.EXE, a 12,032-byte file time-stamped 11-14-85, and CHK4BOMB.DOC, a 1,664-byte message from the anonymous author, who refers to the program as version 1.00.

I've never seen an update of this nifty little utility and I still don't know the identity of the author. Can anyone out there tell me?

There's also a program called PROTECT that I'm currently searching for. I recall that it's a RAM-resident utility that intercepts all attempts to write to a specified drive. Hence running "protect c:" before trying a new program can insulate your hard disk from any ill effects. What's more, a program like PROTECT will report the attempted disk-write. Some of the new vaccine defenses are based on this kind of system. If you have a lead on PROTECT, please let me know about it.

New Lines Of Defense

We shareware aficionados are among the most concerned of the virus watchers. We have the most to lose if "sharing" becomes *verboten*. Fortunately, people smart enough to be devoted to shareware are also smart enough to think of ways to preserve it.

While wandering the bulletin boards one recent night, I popped into the BBS of the San Francisco PC Users Group and checked out the SHARING conference moderated by shareware enthusiast Enric Teller. I was rewarded by a general message from Gene Catalano, who's been using his thinking cap lately.

Catalano recommends that authors incorporate CRC numbers into their programs. The "cyclic redundancy check" number is computed via a mathematical algorithm that scans all the bytes of a file. Any alteration of the file renders the CRC number invalid.

CRCs have become popular in recent years for Xmodem file transfers on BBSs. The standard Xmodem protocol uses a checksum routine that catches about 95% of all transmission errors.

Xmodem/CRC raises the error-catching rate to something above 99%. Although there's a small speed penalty in using CRC for file transfers (checksums are much easier to compute), most experienced BBS users use Xmodem/CRC whenever they have the option.

An embedded CRC number would make program files much more sensitive to tampering, especially if programs were designed to validate themselves by verifying their own CRCs before running. Theoretically, an infected program could shut down with the message "Invalid CRC number. I don't feel so good."

As Catalano puts it, "This will set a new standard of quality, by which any shareware program infected by a virus or Trojan Horse can be readily identified and more quickly eliminated."

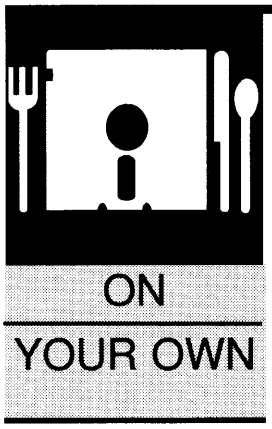
Of course, any determined program cracker (I consider "hacker" an honorable term that I refuse to sully by attaching to sociopaths) could recompute the CRC of a sabotaged program and adjust the embedded number accordingly. Catalano anticipates this problem and offers a possible solution:

"An additional means of protection is to build (and disseminate) a separate list of CRC numbers for all current shareware programs (updated regularly, of course, and available through numerous BBSs)."

To pin things down even better, continues Catalano, "The CRC statistic should also include which CRC program (such as CRCBUILD 1.01 or NSWP 1.02) was used to generate the number ... perhaps even a list of several popular CRC programs with their equivalent number check result for the author's program. (Include, also, which polynomial the CRC program uses to generate the number, i.e.: CCITT or Encyclopedia.)"

The CRC scheme strikes me as a darned good idea. Since Gene Catalano first brought it to my attention, let me give him the last word as well. He observes that the CRC standard could lend new security to shareware and establish new criteria for acceptance: "people would eventually avoid non-CRC-reported programs ... just as they now avoid copy-protected software."

♦ ♦ ♦



The Making of Micro C

By David Thompson

There we were, Larry, Cary, and I in the tech room, our feet up, our chairs balanced on two wheels, and the topic that afternoon turned to the early days of Micro C. As I wound down an hour later, Cary mentioned, "That would be a great story for 'On Your Own.'"

Okay. Here it is. Though it's about starting a magazine, it's also about finding and starting any business. The principle is the same — you keep your eyes open.

Once upon a time (August 1980) in a distant land (165 miles from Bend), a small band of engineers (seven) at Tektronix discovered a (not so) tiny new computer board called the Big Board.

The Big Board was truly unique. It was probably the first single-board computer. On board were: the video generator (80X24), parallel keyboard interface, parallel printer port, two serial ports, 8" single density floppy interface, a Z80, and 64K of RAM. That was everything. The price was $650 for a bare board and a small bag full of parts. CP/M was $150 extra. (The board cost 1/4 as much as the equivalent S-100 system.)

The reason Jim Ferguson (the designer) got so much computer into such a small space (the board fit alongside an 8" drive) was his use of the new LS (low power Shottky) ICs. These little goodies didn't turn small spaces into toaster ovens like their higher-power cousins.

After building the boards, we still had to find cabinets, power supplies, 8" floppy drives ($400 each, single-sided), keyboards, monitors, printers, and printer drivers. (It took me two months to write my first serial printer driver.)

Need For A Magazine

Building the Big Board was easy compared with what it would have taken to design our own Z80 systems and write our own CP/M BIOSs. However, it wasn't trivial, and I could see that a lot of other people were going to be looking for keyboards, cabinets, and printer drivers.

I got very little sleep during the following six months as I struggled with the decision:

"I've got to start a magazine to support all those folks who're building these systems."

"I'd be crazy to start a magazine. It'd be hundreds of hours a month on top of work, graduate school, and family. It probably wouldn't pay its own printing and postage."

"I've got to start a magazine ..."

"I'd be crazy to start a magazine ..."

I'd be crazy to start a magazine. It'd be hundreds of hours a month on top of work, graduate school, and family. It probably wouldn't pay its own printing and postage.

"I've got to start a magazine ..."

"I'd be crazy to start a magazine ..."

It was a relief when I finally decided to do it. At least I could sleep.

What Went Into The Decision

I contacted an expert on newsletters (at least his newsletter said he was an expert on newsletters). After going over all the numbers with him — size of audience, potential number of advertisers, what I could charge for subscriptions, number of issues per year, estimated time to do each issue, and so on — he told me that *Micro C* would never support itself, much less Sandy and me and two kids.

But I was also setting up some other connec-

tions, connections that made the magazine look feasible.

(1) Digital Research Computers (DRC), distributor of the Big Board, offered to send me their list of purchasers. (Made it possible to reach prospective subscribers inexpensively. Absolutely key to the success of the magazine.)

(2) *BYTE* said it was interested in a review of the Big Board. (Very important.)

(3) Sandy was interested in trying a publication. (Also very important.)

Getting Started

In January, 1981, I asked DRC to send the mailing list.

On March 1, I sent off the article to BYTE. In the article I prominently mentioned a fine new magazine called *Micro Cornucopia*, "The Journal Of The Big Board Users."

Three weeks later I bundled up several thousand *Micro Cornucopia* flyers and caught a plane for the West Coast Computer Faire. For four days, my flyers competed for table space with three zillion other flyers for computer games, newsletters, parts houses, and Timex discounters.

Back home (in Portland) Sandy and I waited for the $12 subscriptions to roll in from the Faire. We got two.

By April it was obvious that DRC wasn't going to send the list of Big Board owners. At that point things looked very bleak. However, DRC did agree to send out one of our flyers with every board. Sandy scrambled to get the flyer together.

Early June we started seeing the results of our flyer. One Monday we received eight new subscriptions, that was $96 in our mailbox. In one day!

Meanwhile, I started looking around for something to put into the magazine. I wrote up everything I'd discovered about the board, everything the rest of

the Tek group had found. Everything the neighborhood grocery had found ("A Big Board. That's a long 4 x 12?")

I ran the listing of the PFM monitor ROM which I'd received from DRC. I heard later that Russell Smith hadn't been too delighted, but it filled half of issue #1 and another half of issue #2. (May every new magazine have a PFM ROM listing.)

The flyer said we'd put out the first issue in July (1981). We almost did it, but it was August 2nd when we finished collating and binding magazines on our kitchen table and applying the 200 labels.

Actually we had only 167 subscribers, but we had to mail at least 200 pieces to get the third class rates, so my relatives got copies, Sandy's relatives got copies, and some (probably very puzzled) potential advertisers got copies.

New Subscribers

The *BYTE* review ran in September (1981) and it created a surge of orders. When we mailed issue #2 (Sep./Oct.), we had nearly 300 subscribers, and by #3 (called the December issue) it was approaching 500.

Some of the people who read the article subscribed to find out more about the computer; but even more important, it tripled the order rate for the Big Board. And that, of course, tripled the number of people who received our flyers.

At the end of the first year (July 1982), we held our first SOG. Out of 900 subscribers, 65 showed up at our house for the one-day gathering. We had no speakers, no raft trips, just an afternoon potluck that started at about 10 a.m. Saturday and finished at 3 a.m. Sunday.

We had a couple from Australia, Andy Bakkers from Holland, a whole contingent from Texas, and, of course, a number of fine folks from California. (Californians will jump at any excuse to leave their state.) It was mind boggling.

Income Goes Nuts

As for Micro C? Well, with 900 subscribers it still wasn't paying its way. At least the magazine part wasn't. However, subscribers were sending in software they'd ported over to the Big Board. There were modem programs, disk formatters, printer drivers, small-C, assemblers, disassemblers, disk utilities, you name it.

We checked out the programs, organized them into groups, and stuck them on disks. At $15 each, those disks supported both the magazine and us. (At the end of the first year, sales of subscriptions and disks were running $500 per week and rising.)

Two months before the first SOG, I quit my job at Tektronix.

Immediately after SOG, I did something really crazy. I looked around Central Oregon for a place to move *Micro C* (before it grew too big to move). Three months later (Halloween night), we were moving from the wet security of silicon forest to the crisp, dry, rarefied air of Bend. We'd put all our chips on the Big Board.

> There are more opportunities now than there have ever been, for publishers, for hardware designers, and for software gurus.

Since then, it's been a simple matter of keeping up with the changes in the market. (Do I hear chuckles from the crowd?) But that's another (long and ongoing) story.

Why Did Micro Cornucopia Work?

It worked because we were offering something that people needed and couldn't get anywhere else.

But more important, it worked because we were able to reach prospective customers inexpensively. We had $2,000 of our own money. That was our only capital and it was a lot less than I'd recommend anyone else begin with. The $2,000 meant we weren't going to advertise in *BYTE* or send a direct mail piece to zillions of hackers. But we didn't need to.

Finally, our product was better than people expected. They expected a four-page mimeographed handbill. What they got was a professionally typeset and printed labor of love.

Hindsight

I had thought about contacting other editors, but I didn't. I guess I was a little intimidated by them. Now I know they're as human as I am (or very nearly so). It's possible they would have talked me out of starting *Micro C*. It's also possible they would have connected me with the writers, printers, and other resources that I didn't find until much later.

If I were helping someone else start a similar magazine, I'd tell him to charge more. If we hadn't been selling the disks those early years, we'd have probably packed it in after a couple of years. (*Micro C* is now covering its own expenses. The disks still help, but they're more of a sideline.)

Your Product Ideas

If you have a lock on an important product, can reach your audience, and have enough money to survive while things get rolling, then go for it. You might think that those days of wide open opportunity have passed.

Bull...

There are more opportunities now than there have ever been, for publishers, for hardware designers, and for software gurus. The opportunities are there, but they aren't the same as the one I grabbed when I started *Micro C* seven years ago. Nor are they the same as the ones other people grabbed seven weeks ago.

Need more help? Read everything you can get your hands on: *Micro C*, *BYTE*, *Computer Language*, *Dr. Dobb's*, and *Programmer's Journal*. Then look around, at work, at play, ask friends what they do. Go with them to work. Get details. Be a pest.

Can you automate something at your plant? (So you could improve quality while cutting costs?) Can you trap someone's experience in an expert system? (So you can pass that expertise along to the world?) Do you have an ear for music or an eye for graphics? (Want to create graphics or music libraries for languages?)

See an application but need help with the hardware or software? Contact *Micro C* advertisers. Contact *Micro C* authors. Put together a team of local experts.

Now that you've got the idea and know where the resources are, what next? Check out the "On Your Own" Column in *Micro C* #41 and #42.

♦ ♦ ♦

 (Continued from page 6)

Baud Rate?

The article "The Mysteries of RS-232" (*Micro C* Issue #41) needs two corrections. These are somewhat pedantic corrections for two common errors.

First: the term "baud rate" is redundant. Baud, named in honor of Emile Baudot, *is* a rate. Saying that your serial port's baud rate is 1200 is similar to saying your car's speed rate is 60.

Second: bits per second (BPS) is not always the same as baud. BPS defines the speed that information is sent. But baud is defined as the number of electrical transitions per second. If the output of your RS-232 serial port is 2400 BPS, it is also 2400 baud since the electrical transitions have a one-to-one correspondence with the bits. However, this is not always true for modems.

A Bell 103 standard 300 BPS modem is also a 300 baud modem since each electrical transition transmits one bit. But starting with the Bell 212 standard (1200 BPS), baud and BPS are not the same. A telephone line has a total bandwidth of about 1200 baud, so this restricts the receive and transmit bandwidths to about 600 baud each.

Thus, a 1200 BPS modem is actually a 600 baud modem. But by using phase shifts and other black magic, each electrical transition transmits two bits. The telephone line's bandwidth also restricts a 2400 BPS modem to 600 baud, so it has four bits per transition.

I heartily agree with the author about Joe Campbell's books. In particular, *Crafting C Tools for the IBM PCs* is outstanding. It provides such a wealth of data on the PC's hardware that I use it as a reference for assembly programming.

B. H. Flusche, Jr.
3892 E. Geddes Ave.
Littleton, CO 80122

Efficient Use Of keep()

I've been using Turbo C and found Bruce's article in Issue #38 ("Magic In The Real World") so interesting that I gave the code a try. An excellent article for sure — plenty of information and motivation to play around and learn.

One thing I thought I'd pass along for what it's worth: upon compiling with the small model and running TSR.EXE, the memory left (do a CHKDSK) was much less than it should have been. Upon checking into Bruce's code, I noticed that the second parameter sent to the keep() function wants the memory amount in pages, not bytes.

So I made the changes to divide PROG_SIZE by 16 (of course, not forgetting any remainder). Sure enough, now the memory used is only one-sixteenth as much. I noticed that the following issue repeated the same oversight.

Ronald J. La Borde
LSU Medical Center
Learning Resources
1542 Tulane Ave.
New Orleans, LA 70112

Advertising Ad Nauseam

You are producing a truly delightful publication. I really appreciate your humor, tongue-in-cheek editorials, and viable, realistic technical articles. *Micro C* is actually a pleasure to read. I'm growing weary, frustrated, and angry with the glossy, pop computer magazines.

The major peculiarity of periodicals of this ilk is the increasing difficulty of following an article through a myriad of advertisement-filled pages; we've all experienced, to our mutual dismay, the insipid "continued on page ..."

(continued on page 68)

I am fully aware that advertising plus subscriptions pays the light bill. However there seems to be a Catch-22 here. Once when I felt good and disgusted, I seized one of the glossies at random. The unofficial results: nearly 70% of the rag was devoted to advertising. And this particular magazine had in excess of 250 pages.

Come now, who's fooling whom? I simply do not buy the pious crap about being uncompromisingly objective. You cannot convince me that some heavily promoted manufacturer, representing potentially vast ad revenues, is not going to be given (perhaps I'm too kind) slightly favorable treatment.

Regrettably I cannot attend your annual SOG. I'm certain that I would enjoy the atmosphere, the agenda, and the company. I find your rather detached yet realistic attitude most refreshing. Must be that quaint area in Central Oregon.

But, of course, from the photograph last issue, I can tell that you're all displaced flower children from the sixties, brought up on recreational drug use, free sex, protests, communal living, Woodstock, Alan Watts, and gaily painted VW vans.

Ron Schroeder
5105 W. Kent
Santa Ana, CA 92704

Editor's note: Ron, Ron. Objectively, I have to assume you're desperately out of work and trying to hire on at Micro C. (That's about as objective as I get this time of day.)

However, I suspect you're right about the advertising. I've found it hard to deal dispassionately with products, even when the maker isn't a friend or an advertiser or...

For instance, there are a number of products that don't get their share of editorial space simply because I don't have time to learn or install a major new database package or a fancy replacement for MS-DOS.

And speaking of advertising space, the post office requires second class magazines to limit their ads to 75% of the magazine. Some magazines stay right at the 75% limit. We're running about 33% now, hoping to get up to about 50% before adding pages.

As for the last paragraph, you're close, but both our vans are fading.

♦ ♦ ♦

CULTURE CORNER

Announcing New D Compiler

The Language With The Sweet Smell Of Success

By David Thompson
Micro C Staff

For immediate release

Ten Scents Software announces its new D language. D is beyond AI, beyond objects, beyond C++, beyond belief. D is the first whiff of sixth generation software, for it provides not only the look and feel of the latest environments, but also the smell.

"We've taken the offensive when it comes to software smell," said a corporate spokesperson. "When they get wind of this, competitors will have to get off the dime or they'll be dropping like flies."

Even programmers have been stunned by D's performance.

Willie Faint wrote: "I ran a short test loop which repeatedly called ratdroppings(). Boy, I almost passed out! Since then, the neighbors have left, and no one's tried to break into my place. (Lately, though, I've noticed the local homicide squad digging under my porch.) I'm writing a security program for a meat packing plant, so this should be perfect."

Though the package has been on the market only a few weeks, the odor library includes: evening stable, old skunk, heavy gunpowder, NBA sweat socks, used car, burning hair, pulp mill, roller derby armpits, dead fish, cooked spinach, metropolitan landfill, and secondhand Marlboro.

Those who turn up their noses at this product will soon find themselves downwind of a mushrooming market for this kind of vaporware.

For instance, we've cleared sushi bars in seconds (dead fish), reduced pushing in school cafeteria lines (cooked spinach), prevented scalp damage in salons (burning hair), and cured overbooking at resorts (pulp mill).

Product will be announced in August at the Beantown Cow Palace. A no-host deception will follow the announcement.

Remember, Ten Scents makes scents.

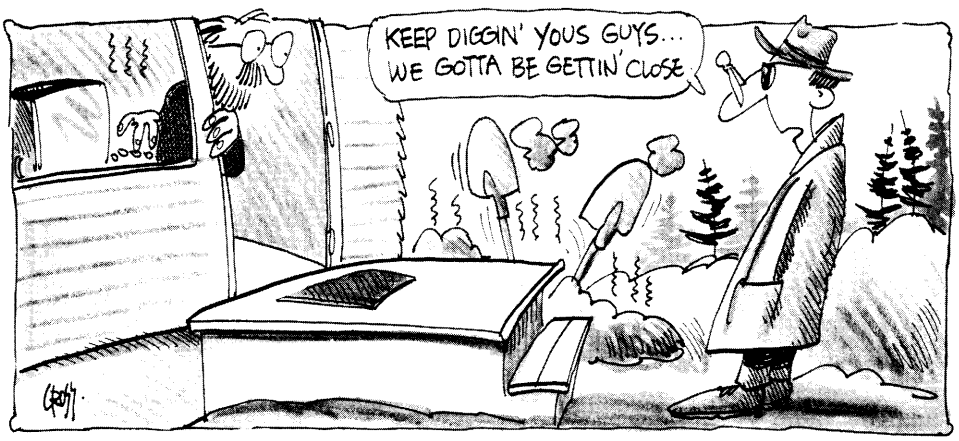
For more information, contact:

Fumie Gayshon
Ten Scents Software

P.S. Of course, we're already working on our secret seventh generation (temperature) and eighth generation (motion) products. We're doing these in response to requests from theatre owners and film makers.

For instance, a high-quality theatre fire (the combination of smoke, searing heat, screams, and the shake of falling timbers) would quickly clear patrons between films.

♦ ♦ ♦



PASCAL PROCEDURES

Creating A Pascal Database From Scratch

With A Little Help From Borland's Database Toolbox

John Paul Jones
6245 Columbia Ave.
St. Louis, MO 63139

I'm looking forward to writing a data acquisition program. It's a database that gets its data from sensors and outputs the data to the screen (with charts and graphs in fancy colors). There aren't many database packages that know how to talk directly to hardware, so I'll be forced to write the software in C or Pascal.

John's description of Borland's Database Toolbox makes Turbo 4.0 the top contender.

Since the theme of this issue is databases, I'll be talking about Borland's Database Toolbox for Turbo Pascal.

What is a database, and what's a database manager? In its most basic definition, any collection of data is a database. Obviously, a random collection of data is of little use; it's the structuring and organization of the data that turns it into a database. The software needed to access, update, manipulate, and report the data is the database manager.

A database then is a collection of related data, organized to provide efficient access, easy update, and reasonable reports.

There are numerous database managers available commercially. Generally their retail prices are high, probably because of their business applications.

In addition to high cost, there are some other disadvantages to the commercial products.

Specialized Database Languages

A database manager, for instance dBASE, is a specialized language designed for the maintenance of data files. This means learning a new language and it means you're probably going to have trouble writing new functions. This will be especially true for functions that don't manipulate data.

Many database managers have their own formats for data storage. This can increase the difficulty of using other languages for manipulation or analysis of the data.

If you're comfortable with Pascal, and have the time to do the programming, Borland's Database Toolbox can provide a cost effective alternative. The retail cost for both Turbo Pascal and the Toolbox ($100 each) is less than the discounted price of most of the commercial database managers. Of course, there will be a significant investment in learning and development time.

The Toolbox

The Database Toolbox includes both a sort and a B+ tree index file manager. These are the core functions needed to build your own database manager. These tools maintain both the data files and their index files.

Data and index files are standard Turbo Pascal files, their structure determined by your data. This means that report and manipulation utilities do not have to use the toolbox routines, though in most cases it's convenient to do so.

Indexed Files

You don't need to understand the theory behind B+ tree indexed files to use the toolbox, but I'll give you a short description.

An index file is like a book index; it's organized for quick lookup of the key (word). Each entry points to a data record (page number).

The toolbox organizes the index in a bit more complex way to allow for more efficient searches. Also, it keeps as much of the index tree in memory as possible.

For a small to medium database (under 1000 records), a disk access is rarely needed for index retrieval. For a larger database, it's uncommon to do more than two disk accesses. Generally it's zero or one for the index, then one for the data.

The records in the data file will be sequential as entered (except for reused deleted records). The index file, however, will always be structured to allow both rapid searches and easy retrieval of data in sorted order.

You can have more than one index file per data file — for instance, you could index your mail list by both name and zip code. (And, you can have duplicate keys inside a single index.) If two or more data records have identical keys, the records are retrieved in the sequence they

were entered. It's easier to manage a database if you don't allow duplicate keys.

The sort module allows you to generate sorted reports on a database without using an index. The sort field does not have to be related to the key.

The toolbox has two UNITs. TACCESS has all the low level routines; using them directly gives you flexibility and control. TAHIGH has higher level routines which are easier to use.

Developing Your Database

Developing your database application requires seven or eight steps:

(1) Read the manual. Although they weren't always easy to locate, I was able to find answers to all my questions.

(2) Design the data record — very critical, be sure to allow enough room for all the data you'll need. Definitely plan on using one of the fields as the "key" for the index file. This way you can reconstruct a corrupted index.

(3) Finalize the key field(s) for the index(es).

(4) Run the TABUILD utility that comes with the toolbox. It has a built-in editor for entry of your record and key definitions. The worksheet option allows you to see what effects modifying some of the basic parameters will have on the final files. Try to minimize the number of disk accesses within the limits of estimated database size and key size.

When you're satisfied, let TABUILD compile TACCESS and TAHIGH. They will be customized for your data structures. By the way, if you're using multiple indexes, be sure to use the longest when you run TABUILD.

(5) Design, write and debug the data entry routines. The steps needed in the routines are data entry, data file update (AddRec), and index update (AddKey). If you're using TAHIGH, these last two are performed by the procedure TAInsert. In either case, the new data record will be added either to the end of the file or at the first deleted record.

The Database Toolbox includes both a sort and a B+ tree index file manager.

(6) Design, write and debug the data access/editing routines. This is the heart of your application. The database isn't of much use if you can't get at or modify the data. The key routines in TACCESS you'll use are FindKey, SearchKey, DeleteKey, GetRec, PutRec and DeleteRec. A typical sequence for this section will be to:

- Prompt for the record to find and accept input — if the key is derived from the data, you can reuse a part of the data input routine.
- Call FindKey — if successful this will return the record number of the correct data record. If there's no match, either inform user or call SearchKey — this will return the record number for data with the nearest key <= the input key.
- Get the data with a call to GetRec.
- Display the data and query if a change or delete is desired.
- Prompt for and accept changes. The data input routine used in (5) can be written so it can also be used here.
- Update the data file with PutRec.
- If the key has been altered, first delete the old key with DeleteKey, then add the new key with AddKey.

(7) Design, write and debug the report generation routines. These may be optional depending on the application. For a mailing list manager, the label printing would go here.

(8) Relax, you're done.

WOW! Seems like a lot of work, doesn't it? Fortunately, it's not as overwhelming as it seems. Borland has even included three sample programs.

The most sophisticated of these, BTREE, is impressive, both for database manipulation and user interface. Borland includes a set of utility UNITs which have a mid-level window manager, a line editor for text, and a menu/help (horizontal style) utility.

If one of the example programs isn't close to your own application, you can still study the source and get a boost in the right direction.

Advantages And Disadvantages

The obvious advantage to writing your own application is that you have total control over the result. With total control goes the disadvantage of total responsibility; the integrity of your database depends on the accuracy of your code. The time needed to develop your application will likely be greater than with one of the dedicated database managers.

Overall, Borland's Database Toolbox provides all you need to develop simple or complex custom database managers.

♦ ♦ ♦

Intelligence Revisited;

Or Deep Into The Heart Of Raphael's Puzzle

By Gary Entsminger
1912 Haussler Dr.
Davis, CA 95616

This is a crazy group. The first day of the West Coast Computer Faire had just gotten underway when a beaming person approached. I reached out my hand.

"I solved it," he said.

"It?" I asked.

"Yeah, you just write 'one' in each blank. Of course that's cheating but it solves Gary's puzzle," he said.

I was more than a little puzzled myself. The latest issue hadn't even arrived at Micro C and here was someone with a solution. (Shades of negative runtimes?) Ah well, he did eventually shake my hand.

As a test of your intelligence, I asked you (in *Micro C* #41 May-June 1988) to send solutions to the following grand and glorious puzzle —

In this sentence, the number of occurrences of 0 is _, of 1 is _, of 2 is _, of 3 is _, of 4 is _, of 5 is _, of 6 is _, of 7 is _, of 8 is _, of 9 is _.

Just fill in the blanks, I said, leaving the precise rules of the puzzle unstated, because I wanted the determination of the rules to be part of the puzzle.

Over 30 of you have responded so far; most intelligently.

What Was Intelligence?

Your approaches to the problem, your algorithms, and your solutions ran a very interesting gamut — from philosophic and textual to numeric.

Those choosing a philosophic route to intelligence indeed took the determining of rules to be part of the puzzle. You asked first —

(1) Does the solution have to be true?

(2) And what "type" of answers can fill in the blanks?

Fortunately (I think), you decided almost unanimously that you wanted true solutions.

A minority (five or so) chose to accept "strings" for blank fillers, so a true answer could be —

```
0 is ''one'', 1 is ''one'', 2 is ''one'', 3
is ''one'', etc..
```

But —

```
0 is ''two'', 1 is ''two'', etc..
```

wasn't acceptable.

All but one of you chose to pursue a numeric solution. But your approaches differed considerably. I received a dozen or so "intuitive" solutions which began —

```
0 is ''1'', 1 is ''1'', 2 is ''1'', 3 is
''1'', etc..
```

In other words, you put 1 in every blank and then looked at the results. You were wrong, but you noticed it right away. Particularly, you had more 1s; so you upped the 1s (usually to either 9 or 11).

If 9, you likely updated the 9s next (to 2), which required you to update the 2s (usually to 2, but that meant you had to update immediately to 3). Which meant updating the 3s (to 2). Then you recounted the 1s (got 7). Changed the 7s (to 2); and changed the 9s *back* to 1; and you had it.

Or as changes in state —

```
0s  1s  2s  3s  4s  5s  6s  7s  8s  9s
1   1   1   1   1   1   1   1   1   1
1   9   1   1   1   1   1   1   1   1
1   9   1   1   1   1   1   1   1   2
1   9   2   1   1   1   1   1   1   2
1   9   3   1   1   1   1   1   1   2
1   9   3   2   1   1   1   1   1   2
1   7   3   2   1   1   1   2   1   1 (True!)
```

In other words, the puzzle (or function) reaches equilibrium in seven steps or changes of state.

Then most of you puzzled over the possibility of more than one solution. One of you used brute force (a program) to check all solutions of possible values for 32-bit integers. Time consuming and exhaustive, but definitely not intelligent.

Many of you then tackled the intriguing (but somewhat difficult) option of developing an algorithm to solve the problem in a few itera-

Figure 1 — Requires Turbo Prolog 2.0 Or Later

```
PREDICATES

compare(INTEGER)
count(CHAR)
test
make_one_str(INTEGER,STRING)
next(INTEGER)
not_sol
parse(STRING)
process
solution
set_random(INTEGER,INTEGER)
write_try

DATABASE - defs

maximum(INTEGER)
iteration(INTEGER,INTEGER)

DATABASE - notes

how_it_is(INTEGER,INTEGER)
try(INTEGER,STRING)
temp_str(STRING)

CLAUSES

process:-                              % Run with random sds
        maximum(Max),                  % Max value for random gener.
        set_random(0,Max),
        write("\n"),
        test,                          % Test random seeds
        iteration(_,Maxits),           % Reset system
        retractall(iteration(_,_)),
        assert(iteration(0,Maxits)).

set_random(Num,Max):-                  % Computer sets seeds 1 by 1
        Num < 10,
        random(Max,H),
        str_int(S,H),
        assert(try(Num,S)),
        assert(how_it_is(Num,1)),
        Next = Num + 1,
        !,set_random(Next,Max).
set_random(Num,Max).

test:-
        iteration(C,Max),              % How many iterations?
        not(C = Max),                  % Are we done?
        make_one_str(0,""),            % Convert many strings to 1
        temp_str(S),                   % Get the new (1) string
        parse(S),                      % Look at each char & count
        write_try,                     % Show where we are
        compare(0),                    % Is the new string true?
        next(0),                       % Yes; Next state
        N = C + 1,                     % Update counter
        retractall(iteration(_,_)),
        assert(iteration(N,Max)),
        !,test.
test:-                                 % String false; Next state
```

tions. The algorithms and their implementations differed wildly.

I received code in BASIC, Pascal, Trilogy, and C. Execution times varied from a few seconds (I presume) to six minutes to find a solution.

I programmed a solution in Prolog which has found (I believe) the only three solutions to the puzzle (if you find others, let me know).

First, I'll give you the solutions; then I'll discuss them a bit; and finally I'll show you the algorithm. The code in Figure 1 finds the solution after generating random seed values for each of the blanks.

I've put a much more complete (compiled) program, "Raph," on the Micro C bulletin board. (It's menu-driven and lets you interactively set seeds, iterations, and maximum values for the random generator. You can either trust the machine, or try out your own intelligent guesses.)

I don't intend to claim the best or "most intelligent" implementation. But in my defense — this code finds each solution (from any seed set there is only 1) in less than an eyeblink of a generic AT.

The Solutions

I know of two 1-cycle solutions —

```
0s  1s  2s  3s  4s  5s  6s  7s  8s  9s

1   7   3   2   1   1   1   2   1   1
(A)
1   11  2   1   1   1   1   1   1   1
(B)
```

and one 2-cycle solution —

```
0s  1s  2s  3s  4s  5s  6s  7s  8s  9s

1   7   4   1   1   1   1   1   2   1
(A')
1   8   2   1   2   1   1   2   1   1
```

By 1-cycle I mean — one unchanging state (or solution).

By 2-cycle I mean — a modulation between 2 almost true states (or solutions). In other words, if you continue to iterate the function indefinitely, you'll forever get two slightly untrue (or approximate) solutions, one just after the other. The solution (A') is always almost A (or true), but never quite.

You can actually predict when you'll get solution B (or A or A'). But I'm getting ahead of myself. Let me show you the algorithm before we discuss the results.

The Algorithm

All but one of you agreed that a solution had to be true, or in other words the values in the sentence had to be in a true state. I went with the majority.

I assumed two states.

A correct one and one to test for correctness. A correct state is merely one that's true (i.e., contains no logical contradictions). That's the one we want. And the one to test is the one we start with. It might be true; but most likely it's not.

So, we have two problems —

(1) Where to start;

(2) How to proceed;

Although I feel (as many of you do) that many 1s is somehow (intuitively) correct, I didn't feel it offered a general starting point for a computer (oops, I forgot to mention; one of my rules was to tap the edge of the iceberg of machine intelligence).

It would be nice to have a beginning that could be applied to other problems as well.

I can't begin to explain why nine or eleven 1s is appealing (in a general sense, I mean), and finally can't think of any place to start that's better than any other.

So I reconsidered the reasoning of one of my favorite writers, Stanislaw Lem. In his novel *Fiasco*, a "last" generation computer always begins to look for solutions depending on what it knows. If it knows nothing, it begins anywhere; for everyone knows, a random beginning is the best beginning when you don't know enough to determine a better one.

So, the program in Figure 1 begins with a random seed set.

How To Proceed

From that beginning, the how-to-proceed algorithm is very, very simple.

(1) Test the current state.

(2) Is it true? If so, we have a solution.

(3) If not, determine what the current state is (i.e., count the current number of 0s, 1s, 2s, etc.).

(4) Loop (i.e., test the current state).

That's all there is to it.

I used Prolog (specifically Turbo Prolog) because —

(1) it's naturally recursive (my algorithm is recursive);

(2) it's great for pattern matching (comparing states is merely comparing patterns);

(3) it's very fast.

(continued from page 73)

```
        iteration(C,Max),
        not(C = Max),
        next(0),
        N = C + 1,
        retractall(iteration(_,_)),
        assert(iteration(N,Max)),
        !,test.
test.

make_one_str(N,S2):-                    % Make one string
        N < 10,                         % Set for 10 strings
        try(N,S1),
        concat(S1,S2,S3),
        retractall(temp_str(_)),
        assert(temp_str(S3)),
        Next = N + 1,
        !,make_one_str(Next,S3).
make_one_str(N,S2).

parse(S):-                              % Look char
        S = "".                         % Done?
parse(S):-
        frontchar(S,F,R),               % Not done. Look at char
        count(F),                       % Count it
        !,parse(R).                     % Get another

next(Num):-                             % Tranform states
        Num < 10,                       % Set for 10
        how_it_is(Num,Val),             % Last try -> New how_it_is
        str_int(S,Val),
        retractall(try(Num,_)),
        retractall(how_it_is(Num,_)),
        assert(how_it_is(Num,1)),
        assert(try(Num,S)),
        Next = Num + 1,
        !,next(Next).
next(Num).

compare(N):-                            % Does try = how_it_is?
        N < 10,                         % in other wds, is try true?
        how_it_is(N,N2),                % Try is true, if all 10
        str_int(N3,N2),                 % trys are true
        try(N,N3),
        Next = N + 1,
        !,compare(Next).
compare(N):-
        N = 10,
        solution.
compare(N):-
        not_sol.

write_try:-                             % Where are we?
        try(N,S),
        write(N,":",S,","),
        fail.
write_try.

solution:-
        write("Eureka!\n").

not_sol:-
```

```
        write("Nope.\n"),fail.

count(Ch):-                          % Update char counts
        str_char(S,Ch),
        str_int(S,Num),
        how_it_is(Num,N),
        One_more = N + 1,
        retractall(how_it_is(Num,_)),
        assert(how_it_is(Num,One_more)).

Goal
        assert(maximum(17)),         % Set random generator
        assert(iteration(0,10)),     % Set # iterations
        process.                     % Do it!
```

Solutions

Using this algorithm I always get a solution in 7 or fewer iterations. (For comparison, a brute force algorithm, trying all the possibilities of all integers would require something on the order of 9!, or 362,880 iterations.) Usually one of the solutions pops up between 3 and 5 iterations, so checking them all would hardly be intelligent.

When I first wrote about the puzzle (#41), I had no idea which results were possible.

Now I think I can predict at least this much —

If the test solution (or state) becomes uniform (i.e., 0 is N_1; 1 is N_2; 2 is N_3) at any time during the iterative process, we'll get B —

```
1,11,2,1,1,1,1,1
```

Else, we'll get either A or A'.

I can't tell, yet, when we'll get one or the other, or why, but let's keep working on it.

Meanwhile, thanks to the following folks who submitted correct (or at least interesting) solutions —

WR Ayers	Millard Murphy
Don Boose	Amos Newcombe II
R. Broberg	George Richards
Alison Brody	Michael Salmon
Gary Carter	Foster Schucker
Leon Davidson	Alan Stretton
Greg English-Loeb	Jason VandenBerghe
Guy Gallant	Wayne Vucenic
Donald Halford	Tom Wilheit
Dan Kelly	Frederick Winyard
Greg Lowenberg	Michael Yam
Larry Maybin	

Oh, and by the way, air out your thinking caps. Next issue I'll spring (or will it be fall?) another puzzle on you.

Half-step Toodle Loo.

♦ ♦ ♦

Around the Bend (continued from page 4)

RAM Prices

I ruined my brother's whole day by asking him about RAM. Hey, RAM prices aren't funny anymore. We're talking $12 a piece for 256K dynamics, no matter what the speed. You know that stack of dusty XT boards you'd left in the closet? The ones with 512K? The ones with $200 worth of RAM? Each! The ones you're taking down to your local computer builder to cash in?

Man the RAM parts! $400 a meg is outlandish. Don thinks it could double before it gets better.

Plus, he says lots of board builders are getting really marginal parts (when they can find them). He thinks that someone is buying huge lots, keeping the best parts, and then selling the rejects back to suppliers. So, small companies (and smaller individuals) need to be particularly careful to test the parts they get.

Meanwhile, I understand that 1 meg RAM chips are down to $22 each. Let's see, 9 X $22 is $198!

If you have an XT or AT motherboard, what would it take to make it accept 1 meg chips? You'll need 18-pin sockets (unless you let the two additional pins hang out) and you'll need to multiplex two system address lines down to one RAM address line (but the XTs have the extra MUX available on the board). Unless the 1 meg parts need special refreshing, that's probably it.

Someone want to do it? Do I smell an article? Do I smell 2 or 3 meg on an XT mother board? Do I smell?

The Drowsy Clone & The Magic Ring

I just got a letter from a reader. (Don't laugh, it happens.) In it he says he has a power distribution box that listens to a standard Hayes-style external modem. When the modem gets a ring, the modem's ring detect signal turns on the power.

The theory is that the computer comes on, realizes that it was awakened by a (magic) ring, instantly turns into an RBBS, and says hello.

Thus, you can make your office computer available 24 hours a day, without leaving it on 24 hours a day. (During the day you use the computer normally without removing this package.)

The device is called Mr. Mox and it's $99.95, including the adapter which grabs the (brass?) ring from the modem.

I haven't seen it yet, so the caveat is empty (that's Greek for you know what I know).

Maybe they'll send one and I'll hook it up at the house. That way I can call home and talk to the computer instead of Jennifer, or Erin, or Sandy... Should be popular.

Mr. Mox
Kenmore Computer Technologies
30 Suncrest Dr.
Rochester, NY 14609
(716) 654-7356

Friendly Finder

This is a small, easy to use search program for dBASE files. I fired it up and in 5 minutes was scanning through databases, large and small.

The key is that it will search for any data in one or more fields. Let's say you're looking for someone who lives in Minnesota, Montana, or Michigan who has placed an $18 order

and has a lastname that sounds like Jones.

Easy.

Friendly Finder will give you hard matches and soft matches. You select the order field, enter $18, and make it a hard match. In the State field you enter, say, Montana but leave it soft. In the lastname field you enter Jones, again soft.

Friendly will then display all the records, one line per record, that have $18 orders, states resembling Montana, and lastnames resembling Jones. (You'll get Bones, Phones, Mines, Clones, Joe's, Moes, and so on.) How it decides what matches and what doesn't isn't immediately obvious.) If you had entered just "J" under lastname, you'd have gotten only names that started with J.

The matches appear to be in a reasonable order. The closer the match, the higher they are on the list. The program highlights perfect matches.

Friendly begins the search the instant you start typing. That's a bit disconcerting. You're typing and it's looking. On smaller databases (100 to 1,000 records) it'll be displaying records before you finish entering the search string(s). As you add or delete characters from, say, Jonathan, you'll see the list of matches change.

Go to a second field, and as you enter characters, you'll see the list shrink. Go back and delete the first field and the list will expand because you're keying only on the second field.

The program uses no indexes, everything's done in memory; it just reads the records, keeps track of what's where, and then displays matches. Speed is super for smaller files, it's not bad (30 seconds) for searching 15,000 records (2 megs) on an XT clone.

It's trivial to use, and just about the only way you're going to find really obscure matches. Super program to purchase for clients. Not copy protected.

Package includes a file futzer so you can use Friendly with non-dBASE files.

Friendly Finder $99
Proximity Technology, Inc.
3511 NE 22nd Ave.
Fort Lauderdale, FL 33308
(305) 566-3511

Missing Connections

More bad news about CP/M. A newcomer, CP/M Connection, produced its first catalog, mailed about 15,000 copies, and then folded.

Robert Ward, founder of CP/M Connection, is also publishing the *C Users Journal*. The *Journal* has been growing (with all the resultant headaches), so he's focusing on it.

I asked him about the CP/M market.

"There are enough end users to support a couple of distributors. But the market's very fragmented. *S-100 Microsystems* used to serve a major chunk of the CP/M programmers. But none of the magazines now reach more than a tiny fraction of the audience.

"The people still using CP/M are using it because they have something that works, they're not buying a lot of stuff and they're not writing new software."

I mentioned that I've been unable to contact Echelon (Z System) via their South Lake Tahoe number (it's been disconnected).

"That's really disturbing to hear. They offered the only unconfigured CP/M so their demise would close the doors on the innovator or the engineering student who wants to build his own system."

Echelon

It was after getting reports from readers that all was not well with Echelon that I dug out the most recent (March 14) copy of Z-News.

At the very end of the newsletter, I found a rambling statement titled "Of Angels and Eagles." The following are two excerpts:

"Everybody knows that something is eternal, must be! It's not Earth or even Sun or Stars... when they are gone, we are left with, well... see you down the lines..."

And,

"Toto, I've got a feeling we're not in Kansas anymore."

And perhaps, not in South Lake Tahoe, either.

Alpha Systems Non-CP/M

I contacted Ampro Computers (the big company with the little computer); after all, if anyone knows what's going on in CP/M it's Ampro. They told me that Joe Wright, author of some of the Z-System utilities, was now supporting the Z-System.

I called Joe:

"We're CP/M compatible, but that's the end of that story. We're new, alive, 8-bits, a rich operating system, not dead like CP/M. However, we support CP/M users.

"We're still reaching our old audience on bulletin boards and customer lists. We'll sell them new products, but the object is to broaden our audience. The Z-System was always been difficult to get running, people had to create their own environment. Most users couldn't do it and only a limited number even tried."

So he wrote two new versions of the Z-System.

"NZ-COM is sort of an autoloading Z-System. Z3PLUS is an autoloading system for CP/M-3 machines. Neither requires assembly of anything. The user gets to tailor his system simply by choosing which features he wants. He can save memory by eliminating features."

Joe mentioned that you must have a CP/M BIOS. If you're building a new system then you need to write a BIOS. Unlike ZCPR, this system doesn't require space in the BIOS, instead, it takes one additional K out of RAM.

All the utilities remain on disk. There are over 200 of them. He says, at a minimum, you have to have another 4 or 5 utilities at 2 to 4K each.

Joe Wright
Alpha Systems
711 Chatsworth Place
San Jose, CA 95128
(408) 297-5594

Thieving Formatters

I just received a release from Xerox Corporation about their new 3 1/2" floppies.

"ROCHESTER, N.Y. June 14 — A microdiskette for IBM Personal System/2 personal computer that provides 2-megabyte data-storage capacity is being offered with a lifetime

warranty by Xerox Corporation."

Great, I thought. When's IBM coming out with a 2 meg floppy drive?

"...use with IBM PS/2 model 40,50,60,70, and 80."

Sure. Not even IBM is claiming to get 2 megs in their high-density floppies. Finally I got down to the small print:

"Formatted capacity is 1.44 Mb."

Ah ha! And they wonder why I don't run press releases.

But hold on a minute, maybe they know something. After all, 3 1/2s are thicker than standard floppies so manufacturers could cheat by stacking bits.

And if there are really 2 megs, what are the formatters doing with the .56 meg? Are they saving it up and then modeming it back to the disk manufacturers? (Might explain my phone bill.) Are they writing secret messages in that space readable only by other computers? ("Hi Pussycat. Thanks for the letter on the Mac disk. Things are fine here, too. And by the way, here's the other half of the Encyclopaedia Britannica.") And you wondered why disk I/O was so slow.

It's a puzzle. I can understand why Xerox or IBM might put up with this kind of waste, they're large corporations. But the rest of us? There must be something we can do.

Newsstands

There are now 20,000 copies of *Micro C* on bookstore racks across the world.

Hooray!

Now, if all of you (U.S.) rack jockeys would tear out the subscription card and mail it in with $18, we'd be rich (moderately) and down right delighted.

You'd get your magazine faster (two to four weeks faster). You'd get it for less, $3 per copy instead of $3.95. You'd pay no sales tax. (Oregon has no sales tax.) And, you'd really be part of the group.

When you subscribe, you really contribute to our irresponsibility: to funnier articles, to sillier cartoons, and to longer editorials. (Okay, if you subscribe, you choose: longer editorials, or shorter editorials, write it on the subscription form. I'll go with the majority: especially if the majority chooses longer.)

Western Digital Hotline

I was delightfully surprised when I discovered that Western Digital had an 800 technical support line. It's one of those automatic, "wander through the maze" sorts of things, but its automated message machine has information and you can (eventually) reach real people.

Call (800) 777-4787 if you have questions about such niceties as WD's AT or XT controllers.

Finally

I'm outa here. There are ancient planes awaiting. The Stinson and I are scheduled to join the state-wide jaunt of antique aircraft. It starts in three days and we'll be flying over the coast, across the valleys, and around the mountains (not over, you fool, not over).

We'll wear goggles and scarves (even in the cabin craft) and sleep under wings (not under the engines, that's where the oil drips). We'll fly into backwoods grass strips as well as municipal airports, and aircraft clubs in the local areas will be holding barbeque dinners in our honor.

The Oregon Antique Airplane Club holds one of these

every two years and this'll be my first. (Weather permitting, anyway. Everything you do with an antique airplane is "weather permitting.")

Now, if I could put together one of Bill Davidson's (PC Tech) fancy flight computers with a Loran, boy, I could fly the old bird just like they fly those 737s. (Sans the pop top.)

And that's all from greater Bend.

David Thompson
Flight Line Editor

P.S.

Boy, when you go away for a few days, they really start saving up for your return. I was finished with the editorial, ready to do a perfunctory final blessing on this issue when Carol meandered up trying to look casual.

"The editorial is too short."

April fool.

"No, it really is."

Use the three pages you cut out of my last editorial.

"We're already using them, they're catching the drips under the waxer."

So now I've got add something though I've already covered absolutely everything even vaguely interesting. Everything, that is but the airplane tour. Now, that was interesting but I can't write about it yet.

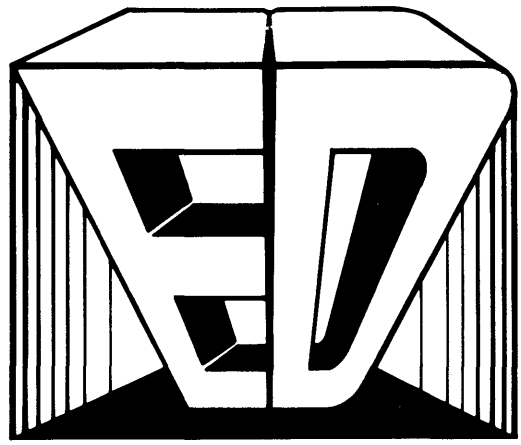
You see, some things have to sit for a while, simmer in the juices a bit before I can write about them. Sure, I used to write for a daily newspaper. And, sure I could knock out a piece an hour after the event but those were simple things: a ribbon cutting, a check passing, Richard Nixon's presidential campaign.

Writing about something I love, something with real significance is entirely different. The words have to be carefully selected, chosen for their color and meter, for their easy flow or cutting edges. The English language is the richest in the world, borrowing the best from all the others. There's no way I'll use such a powerful tool on such an incredible subject and do it halfway.

Plus, I'm just not in a creative mood. There's something about a blank half-page and a 3-hour deadline that kills the feeling.

So, if I'm a little short with you (not with Carol, she's short enough already) you'll have to understand. I'll write a truly great wax catcher next time.

Promise.



TM

EDITING SYSTEM

Solidity, Speed, and Power.

ED is an object-oriented open architecture system for editing programs and manuscripts and manipulating data files. Flexibility, extensibility, and programmability are realized by providing access to the objects which ED itself manipulates. All aspects of ED's appearance and operation can be controlled by the programmer. ED, a restricted or enhanced form of ED, or any of ED's objects, can be embedded in the programmer's applications with no royalties. Objects such as dynamic arrays, windows, data entry windows, macros, menus, browsers, popup directories, and regular expressions are manipulated through normal C function calls. Functions for creating manuscripts are part of ED's design. Stream blocks, word-wrap, tabs, intelligent paragraph formatting, block justification, and pagination behave correctly and execute instantly. Finally, ED can be used to create sophisticated sorts and filters. Macros, search and replace, column blocks, and block sorts allow records to be sorted and selectively removed, and fields to be added, rearranged, and removed.

ED • NUMBER OF BUFFERS LIMITED ONLY BY AVAILABLE RAM • NUMBER OF WINDOWS LIMITED ONLY BY SCREEN SIZE • **POPUP DIRECTORY FACILITY** • COPY • RENAME • DELETE • BROWSE • FILE STATUS • EDIT • EXECUTE • SORT • **SEARCH AND REPLACE** • FORWARD/BACKWARD • CASE SENSITIVE/INSESITIVE • FULLY PARENTHESIZED REGULAR EXPRESSIONS • INCREMENTAL/GLOBAL REPLACE ACROSS ALL BUFFERS • **BLOCK COMMANDS** • COLUMN/LINE/STREAM BLOCKS • SAVE • CUT • DELETE • BLANK • SEARCH-FOR • FORMAT • JUSTIFY LEFT/RIGHT/CENTER • UPPER/LOWER CASE • REMOVE • OVERLAY • REACTIVATE • TAB • DRAG • SORTS • **MACROS** • MENU DRIVEN • VIEW • AUTO-EXECUTE • TIME-DELAY • NESTED • RECURSIVE • SIZE LIMITED BY AVAILABLE MEMORY • **MORE FUNCTIONS** • VIEW/MANIPULATE BUFFERS THROUGH MENU OR KEYSTROKE • INTELLIGENT DIAGRAM/BOX DRAW • ADJUSTABLE SCROLL VALUE • TRANSPOSE LINES/CHARACTERS/WORDS • MOVE-TO-POSITION STACK • VISUAL TAB CREATION • COLUMNAR/ENTIRE-LINE TABS • TIME AND DATE STAMP • RESTORE TYPED-OVER CHARACTERS • CHANGE DIRECTORY • BRACKET/BRACE/PARENTHESIS MATCHING • ENTER GRAPHICS CHARACTERS • EXCUTE DOS COMMANDS • POPUP ASCII TABLE • USER DEFINED MENUS/POPUP FILES • KEYBOARD CONFIGURATION • PRINT BUFFER WITH PAGE NUMBERS, ADJUSTABLE SPACING AND MARGINS • INTERACTIVE TUTORIAL • ALL DOCUMENTATION ON-LINE • WRITTEN IN C AND ASSEMBLER• **EXTERNAL UTILITIES** • INTELLIGENT CHANGE DIRECTORY COMMAND • POPUP DIRECTORY FACILITY • STRING TRANSLATOR • **HARDWARE REQUIREMENTS** • DOS • PC/AT, PS/2, 386 • CGA, MDA, EGA, HERCULES, WYSE 700/AMDEK 1280 • RUNS IN ALL VIDEO MODES, NO FLAGS, NO DRIVERS • 256K • **FOR LITERATURE, A COMPLETE LIST OF FUNCTIONS, AND A DEMO DISK, CALL 201-450-4545.**

VERSION 1.0

PRICE: $265.00

THE AMERICAN
COSMOTRON 80 HOLMES ST • PO BOX 128 • BELLEVILLE, NJ 07109

Reader Service Number 107

CPM COLUMN

Wordstar Mods

And, Fixing The Kaypro BIOS

Happy New Year

In response to Fred Horton's CP/M question (*Micro C* Issue #42, p. 83) about getting Perfect Filer to accept dates later than 1988: the answer has been in *Profiles* a couple of times. It first appeared in October of 1985, suggested by David Porritt.

With DDT in drive A and Perfect Filer in drive B, enter:

```
DDT B:SETUP <return>
```

At DDT's "-" prompt, type:

```
S0715 <return> (that's a zero)
```

DDT will display:

```
58 (hex for 88, naturally)
```

You'll type:

```
63 <return> (good 'til 99)
. <return>
^C
```

And at the system prompt, type:

```
SAVE 16 B:SETUP
```

Be warned that I haven't tried this yet. I have six more months, after all.

Deloss Brown
444 Central Park West, #5A
New York, NY 10025

WordStar Printer Solution

Regarding Mr. Elrod's problems with a printer driver for his Canon PW-1156A dot matrix printer and WordStar 4.0 (*Micro C* Issue #42, p. 83): there's really no need to have a special printer driver. All printer controls are available immediately from within any WordStar document if he just installs WordStar so that the ^PE command is a "wild card" — the <ESC> character, 1B hex.

Then, any characters which follow the ^PE command will be interpreted as printer controls. Up to five characters of printer controls may be entered. So, with this one command, he has available all of the printer controls that can be used by the printer. Unless, of course, they exceed a total of six bytes. Install the ^PE via WSCHANGE. The input is: 01 1B (the 01 says there's one byte used).

I don't know why this seems to be such a little known secret about WordStar, but it's the only way to go if you want to use a lot of special printer controls. There just isn't enough room in WordStar to assign dedicated controls to all the functions you might want to use for a given printer.

As for the printer drivers that come with WordStar, these are real horror shows. If you haven't figured this out yet, certain drivers ignore some of the controls you've laboriously installed with WSCHANGE. And it isn't obvious which do and which don't. My suggestion is to install the custom driver — one of the few that accepts all the controls. Then rename it to match your printer.

I, too, have the Canon printer, and it's a very nice machine with more capabilities than most people realize. It has a socket where you can install a RAM (8K X 8 static, eg. Hitachi 6264) to hold downloaded fonts.

For instance, I download the complete IBM character graphics set from my MS-DOS computer. If anyone is interested, the full code to define the character graphics is given on page 385 of *PC Magazine*, Vol. 7, No. 4, Feb. 1988.

Delete the condensed print mode and 1/8" line spacing from the listing and it can then be sent directly to the printer as characters 176 through 223. Of course, this won't help most CP/M computers as they don't have provision for addressing more than the ASCII characters 0 — 127.

Norman L. Donaldson
17360 Firma Ct.
Granada Hills, CA 91344-1902

Kaypro Mods Revisited

I have an early Kaypro II and follow the Kaypro column with interest. There are several items which may be of use to your

readers. The first relates to the use of Z80 instructions.

The Kaypro II/IV BIOS doesn't preserve the alternate registers of the Z80. After careful study of the code, I found only one subroutine that causes problems. Calls to the ROM use the DE' register for temporary storage. I've rewritten this routine to avoid use of DE'. The actual change involves only six bytes and can be done using SYSGEN, DDT, etc., or by altering the system tracks with DU or some other disk editor. See Figure 1 for the altered code.

Figure 1 — Kaypro II-83 ROM Call That Preserves DE'

```
FBAC  00           NOP
FBAD  DB 1C        IN    A,[1CH]
FBAF  CB FF        SET   7,A           ;SELECT ROM
FBB1  D3 1C        OUT   [1CH],A
FBB3  ED 73 DC FB  LD    [0FBDCH],SP   ;STORE PREVIOUS STACK POINTER
FBB7  31 00 FC     LD    SP,0FC00H     ;SET UP LOCAL BIOS STACK
FBBA  E5           PUSH  HL            ;VECTOR TO BIOS FUNCTION
FBBB  21 C2 FB     LD    HL,0FBC2H     ;LOAD RETURN ADDRESS
FBBE  E3           EX    [SP],HL ;PUT RETURN ADDRESS ON TOP OF STACK
FBBF  26 00        LD    H,0
FBC1  E9           JP    [HL]

Note: Underlined bytes are the changed locations

END OF LISTING
```

The other items relate to the 7 MHz speedup article in Issue #33. Looking at the schematic (from Micro C, of course), I concluded that the U66 mod (CAS/MUX timing) could be improved. As shown, the CAS and MUX signals are generated by delaying the RAS signal. Since RAS is generated for both memory R/W and refresh, there are unnecessary CAS and MUX signals during refresh. (Also, it's best to avoid long wire runs over the printed circuit board.)

The Z80's MREQ is actually responsible for these memory control signals. MREQ (gated in U56 with the refresh (RFSH) signal to avoid extra MUX and CAS during memory refresh) is available at U66, pin 9. By sending pin 9's output through two gates of a 74LS04, MUX can be generated. The delay through U56 takes the place of the third gate used in the original mod. CAS is generated the same as before — two gate delays after MUX.

To implement this mod, I'm using a small daughter board that carries U66 and the LS04 and plugs into the U66 socket. No bent-out pins, pin to pin wiring, or soldering to the board is needed. I made the simple single-sided board myself.

I didn't like the U86 mod for some of the same reasons — long wires carrying high frequencies and bent-pin soldering. Instead, I made another daughter board to plug into U86's socket. This board has the original U86 (74LS293), a six section DIP switch, and a 7497.

The 7497 is a synchronous 6-bit binary rate multiplier. The DIP switches provide an adjustable binary input to the 7497. The 7497 takes this input (0-63), divides it by 64, then multiplies it by the input clock frequency (20 MHz). The output of the 7497 is a short pulse which is turned into a square wave (note Z80 timing requirements) by the LS293.

The clock output is half of the pulse frequency. The result of all this is that the clock rate is adjustable from 0 to 10 MHz in increments of 0.15625 MHz. Thus, you can select the highest workable clock speed for your machine.

I've traced these signals from the Z80 to the RAM and changed all critical chips to 74Fs. The following replacements were made: bus drivers (U49, U59, U62), logic chips (U39, U48, U60, U73), and multiplexers (U33, U34) to F parts, and Z80 parts to Z80B.

My Kaypro II with the original ROM (a 450 nsec part, so both 5 and 7 MHz are out of the question) now runs at 3.75 MHz. This is a significant improvement over the standard 2.5 MHz.

Alan Voth
RR1 Box 90
Valley Center, KS 67147

♦ ♦ ♦

KAYPRO COLUMN

Dynamic Mapping Of The Kaypro Keypad

Dr. Jack W. Crenshaw
1220 E. Idlewild Ave.
Tampa, FL 33604

Okay, you're wondering how the keypad configuration works? You'd like to customize it for WordStar or Perfect or anything else? Well, listen up to Dr. Jack. You'll be a keypad pro before you finish this one.

As any user of the CP/M Kaypros knows, one of their nicer features is the numeric keypad. This keypad, and the cursor keys, can be mapped in software so that the keys generate different ASCII codes — very handy for some applications.

For example, I love WordStar, but I'm not too enamored with all its control-key commands. No problem. I map those commands to the numeric keypad, and use it instead.

But WordStar has a special set of control codes that are supposed to be used for moving the cursor — the familiar diamond pattern of ^S, ^D, ^E, ^X. These codes are set up for people whose keyboards don't include cursor control keys.

The normal Kaypro cursor keys, however, generate a different set (^K, ^J, ^H, and ^L). So how do they work with WordStar? The answer is that every copy of WordStar shipped with a Kaypro has been patched to recognize *two* sets of codes: the standard WordStar diamond, and the Kaypro set.

As it turns out, I found that mapping the cursor keys to the standard WordStar codes works better. Now I can generate many "quick" and "block" commands, like ^K^X (exit) or ^Q^S (beginning of line), as combinations of a keypad key plus a cursor key.

The Problem

There's only one fly in the ointment. A quick survey of Kaypro users shows that few take full advantage of the configurable keycodes. It's not hard to figure out why. Kaypro provided CONFIG for setting up the keypad as well as baud rates and printer. But CONFIG is awkward to use.

What's worse, CONFIG changes the maps by writing permanent changes to data stored on the system tracks of the disk. Because of the way CP/M works, these changes only become active when the disk is cold booted, either by a power-up or by a hard reset.

For baud rates, that's reasonable. But for keypad mapping, it's the wrong solution. Aside from the bother of having to cold boot in order to get the mappings to take effect, it's a pain to configure one keypad mapping for one program, and a different one for another.

CONFIG won't do it unless:

(1) You are willing to put each such application program on a different disk, or

(2) You are willing to rerun CONFIG each time you change applications.

If you use a hard disk, option (1) is out. There are commercial products on the market, such as QuickKey, that solve the problem, but there's also a very simple, free solution.

The Solution

What we really need is a way to *dynamically* map the cursor and keypad keys, quickly and without altering the disk's system tracks. Wouldn't that be nice? Yes. Is it possible? Yes, and it's so incredibly easy you'll wonder why you haven't been doing it all along.

The codes generated for each of the keypad and cursor keys are contained in a translation table near the beginning of the BIOS. This table is loaded from disk to RAM, along with configuration flags and the rest of the BIOS, at cold boot. Once loaded, it remains there until updated by the next cold boot: warm boots don't affect it.

The solution, then, is simple. Why bother with writing to the system tracks of the disk at all? Why not just modify the RAM translation table?

Now, you could get fancy with this. In fact, you could make a new version of CONFIG, just as awkward to use as the old one. But stop and think about it a bit: how many sets of different functions do you want?

I only have two sets of mappings, and they're not likely to change. So I don't want to

have to remind the Kaypro what these are every time. For me, at least, the simplest solution is also the nicest: write a fixed block of codes to the translation table.

There's nothing fancy about the way the maps are stored. For most systems it's simply a list of 18 codes — four for the cursor keys and 14 for the numeric keypad. Then we poke the 18 bytes to a specified place in memory. Not exactly the most complex program ever written!

I wrote MAP and UNMAP to do the job. You must edit these programs so they hold the correct keycodes, and then reassemble them.

There's only one complication. We're doing delicate work here. We're not just running an application program, we're going to be poking into a very sensitive area: the very innards of CP/M. It's kind of like brain surgery — better get it right.

And, the table can lie just about anywhere, what with different versions of CP/M and the great number of aftermarket goodies, such as replacement ROMs, hard disks, and RAMdisks.

That's why the source files for MAP and UNMAP must also be edited so that they poke to the right address. Once you have MAP and UNMAP set up for your system, you probably can't use it on another one without redoing the address.

So you need to locate the character translation table in *your* system. I can't tell you exactly what address that will be, but I can give you some general guidelines that should help. I will also show you a method that's guaranteed to find the maps, regardless of how bizarre your system is. I speak from experience — I've used this technique on about five configurations so far, none of them "plain vanilla" Kaypros.

Locating The BIOS

First of all, you need to know which version of CP/M you have. This is important because different versions implement the translation differently. As far as I know, all versions but 2.2u permit only a single character for each key. Version 2.2u, though, allows you to generate many characters with a single keystroke, so the mapping must be done differently. The programs in this article cover both cases.

Next, you need to know where your particular BIOS is located in RAM. The location of the BIOS depends upon the size of your CP/M. There are a number of ways to determine your CP/M size, all pretty easy. The first one you might try is to (are you ready for this?) ask your computer!

Some systems, such as my old 4-83, display an opening banner that says, "63k CP/M v 2.2." If you're one of the lucky ones with that display, note the address of your BIOS from Figure 1, and go to "Locating The Translation Table."

If you have one of the newer Kaypros, the chances are good that your version is 2.2u1, like mine. In that case, go to the section called "Version 2.2U." The rest of you will have to dig a little deeper.

Figure 1 — BIOS Addresses

```
CP/M Size          BIOS Address

   60K                EA00
   61K                EE00
   62K                F200
   63K                F600
   64K                FA00

END OF LISTING
```

The next easiest way to find your BIOS is to use one of the public domain programs such as TELL.COM. Check CompuServe or a bulletin board. These programs examine your version of CP/M and give you a summary of its key addresses.

Running TELL on my system gives the printout shown in Figure 2. We're only interested in one line of this printout, the one that says, "BIOS jump table starts ... ." That's the address of your BIOS.

If you don't have access to a TELL.COM, all is not lost. You can easily find the address using DDT. The CP/M BIOS begins with the jump table, which CP/M uses to execute BIOS functions. A jump to the second of these entries, the warm boot entry, is maintained by CP/M in addresses 0000-0002. So just run DDT and look at the first three bytes by typing, "-pL0".

The first line of output should be something like:

```
0000 JMP EE03
```

Since the warm boot entry is the second entry in the jump table, it begins three bytes beyond the beginning of the BIOS. Therefore, you should drop off the 3 to get, for this example, an address of EE00 for the BIOS.

Locating The Translation Table

We now have the address of your BIOS. Just to double-check (remember, we're doing brain surgery here!), you can use DDT again to verify that the BIOS is really there. From DDT type, "-LEE00" (substituting your address). You should see a series of JMPs:

```
EE00 JMP xxxx
EE03 JMP xxxx
EE09 JMP xxxx
EE0C JMP xxxx
EE0F JMP xxxx
EE12 JMP xxxx
EE15 JMP xxxx
EE18 JMP xxxx
EE1B JMP xxxx
EE1E JMP xxxx
```

Those are the first 11 entries in the BIOS jump table. (There are 17 such entries in all.) Pay no attention to the actual addresses following the JMPs. We don't need them. If you find this jump table at the address you've gotten for the BIOS, then we can proceed. If not, your address is wrong, and you need to redo the previous steps.

Okay, now that we know for sure where the BIOS is, we need to locate the translation table. It will be easier to find if you're working with a CP/M that has not been mapped by CONFIG. That is, the keypad and cursor keys perform their normal functions. For most systems other than 2.2u, the 18 bytes are all stored together, in the order:

```
up down left right 0 1 2
3 4 5 6 7 8 9 - , enter
```

Figure 2 — Output of TELL

```
CCP starts                  D800
BDOS entry address is       E006
BIOS jump table starts      EE00
cold start routine          F193
warmstart routine           EF35
console status routine      EE45
console Input routine       EE4C
console out routine         EE56
list device out routine     EE5F
punch out routine           EE68
reader in routine           EE71
home disk routine           EE78
select disk routine         EE7C
the set track routine       EE80
set the sector routine      EE84
set the dma routine         EE88
read disk routine           EE8C
write disk routine          EE93
list status routine         EE9A
sector translate routine    EE9E

END OF LISTING
```

In almost all Kaypros, you'll find this table at address BIOS + 35H. From DDT type, "-DEE00" (again, your address). On the fourth and fifth lines of the output, you should see:

```
EE30 C3 ... 36 .........0123456
EE40 37 ... 33 789-,...t..2..:3
```

See those distinctive characters "0123456789-," on the right? Those are the codes for the keypad. The translation table begins at the cursor codes, four bytes to the left of that. If you find the translation table where you expected it, then you're home free.

Configuring MAP.ASM

Now you know exactly where the translation table is. If you've gotten this far, you've also found that the table is stored as 18 consecutive bytes. All you need to do now is to set up MAP (see Figure 3) and UNMAP (see Figure 4) to poke the info.

To make life easy, I've computed the beginning of the BIOS from the size of your CP/M in K. If you were one of the lucky ones that got your BIOS address from Figure 1, and if you've verified this to be correct with DDT, all you have to do is to change the parameter SIZE in the listing.

If your BIOS address isn't one of those in the table (some aren't; Kaypro has shipped some versions with a non-standard "K and a half" offset), then you can't use the program as listed. No problem. You already know the address of your BIOS, so just change the line that begins BIOS EQU to reflect the correct address.

Now edit the codes in the translation table. The codes in Figure 3 correspond to the codes I use for WordStar. You, of course, can use any keycodes you like.

So that you'll be able to undo the effects of MAP, you should also copy UNMAP.ASM from Figure 4, and make the same changes to the address. This program returns the map to its normal state.

Assemble the programs with your favorite assembler. They are deliberately written in plain vanilla 8080 assembler language, so ASM will do nicely.

Using The Programs

Using MAP and UNMAP is simple. When you want to map the keypad to the new values, just type MAP at the command line prompt. Now type a few characters on the keypad. Did you get funny control characters? Then MAP did its job. To put the keypad right again, type UNMAP.

Remember, these programs change the BIOS dynamically. There's no need to alter the system tracks of your disk and no need to do a cold boot to make the changes effective. The keypad will remain mapped until you UNMAP it or until the next cold boot.

CP/M 2.2U

If you have CP/M 2.2u, you're different. CP/M 2.2u is Kaypro's Universal version, so called because it supported all their late model boards: the 1, 2, 2X, and 10. This version has several differences from earlier versions.

Among these differences: it permits a single keystroke to generate multi-character strings. Clearly, the MAP program that I've shown you won't work for 2.2u. To find out what will, I did some poking around with DDT.

In 2.2u, the BIOS begins at EE00H. In other words, it's a 61K system. To support the ability to generate multi-character strings, the translation table is *indirect*. There is a table of lengths and offsets, which begins at address F044H. The data appear in pairs, which for the standard setup look like this:

```
01 1C
01 1D
01 1E
01 1F, etc.
```

Figure 3 — MAP.ASM

```
;*********************************************
;*                                           *
;*THIS SMALL PROGRAM GENERATES A CHARACTER    *
;*TRANSLATION TABLE FOR THE KAYPRO CURSOR     *
;*KEYS AND THE NUMERIC KEYPAD. IT INSTALLS THE *
;*TABLE INTO THE CP/M BIOS.  THE TRANSLATION  *
;*WILL REMAIN IN FORCE UNTIL THE NEXT COLD    *
;*(NOT WARM) BOOT, OR UNTIL THE COMPLEMENTARY  *
;*PROGRAM 'UNMAP' IS RUN.                     *
;*                                           *
;*********************************************

; *** CHANGE THIS TO MATCH THE CP/M SIZE ***
SIZE    EQU     63      ;CPM SIZE IN K
CBASE   EQU     SIZE*1024-1C00H;BEGINNING OF CCP
FBASE   EQU     CBASE + 800H  ;BEGINNING OF BDOS
; *** OR CHANGE THIS TO MATCH YOUR BIOS ***
BIOS    EQU     CBASE + 1600H ;BEGINNING OF BIOS
VTAB    EQU     BIOS+35H;BEGINNING OF PARAMETERS

        ORG     100H

MAPKEY: LXI     H, MAP;ADDRESS TRANSLATION TABLE
        LXI     D, VTAB ;ADDRESS CP/M BIOS TABLE
        MVI     B, 18   ;SET BYTE COUNT
LOOP:   MOV     A, M    ;COPY TABLE INTO BIOS
        STAX    D
        INX     H
        INX     D
        DCR     B
        JNZ     LOOP
        RET             ;RETURN TO CCP
;
; TRANSLATION TABLE TO BE LOADED
;
MAP:    DB      'E'-'@' ;UP ARROW( ^E )
        DB      'X'-'@' ;DOWN ARROW( ^X )
        DB      'S'-'@' ;LEFT ARROW( ^S )
        DB      'D'-'@' ;RIGHT ARROW( ^D )
        DB      'Q'-'@' ;NUM KEY '0'( ^Q )
        DB      'Z'-'@' ;NUM KEY '1'( ^Z )
        DB      'C'-'@' ;NUM KEY '2'( ^C )
        DB      'B'-'@' ;NUM KEY '3'( ^B )
        DB      'A'-'@' ;NUM KEY '4'( ^A )
        DB      'L'-'@' ;NUM KEY '5'( ^L )
        DB      'F'-'@' ;NUM KEY '6'( ^F )
        DB      'W'-'@' ;NUM KEY '7'( ^W )
        DB      'R'-'@' ;NUM KEY '8'( ^R )
        DB      'T'-'@' ;NUM KEY '9'( ^T )
        DB      'Y'-'@' ;NUM KEY '-'( ^Y )
        DB      'G'-'@' ;NUM KEY ','( ^G )
        DB      'K'-'@' ;NUM KEY 'ENTER'( ^K )
        DB      'V'-'@' ;NUM KEY '.'( ^V )
        END

END OF LISTING
```

Each of these pairs gives the length and location of one of the character strings to be substituted for the corresponding key. The first byte is the byte count (always 1 for the normal setup). The second byte is an offset to the code or code string. The BIOS adds this offset to some base address to get the address of the desired string.

The base address of the table is F044H. So, for the first code the BIOS adds the offset 1CH to the address F044H to get the address of the code(s), F060H. Kaypro's documentation says that you can make each string any length you like, as long as the total length is less than 180 bytes. Actually, there is room for 223 bytes, beginning at F060H.

CP/M 2.2u does not allow multi-byte codes for the cursor keys. So they get their own map. The cursor map is at F143H, a simple set of four bytes.

My first couple of tries at writing MAPU were attempts at being very clever: reading a table of variable-length strings, counting the length of the strings, and stuffing both the table of pointers and the byte string areas with the correct values. But then I realized that this is not at all necessary.

Regardless of the data being written to the BIOS, by the time it gets there it's simply a series of contiguous bytes. So all I needed to do was copy a fixed array of data bytes, just as before. Only this time, the number of bytes and their values will vary from version to version of MAPU (depending upon the character strings you want to generate). But they are known values when the assembler gets them, and any good assembler can do hex arithmetic better than you or I.

In MAPU.ASM (see Figure 5), each keycode now has a label, one for each keypad key. The labels give the assembler enough information so that it can generate the correct table entries, addresses, and counts. All you have to do is alter the character data.

I haven't included an UNMAPU.ASM, but it should be pretty obvious how to do this one.

To use MAPU, just copy it into your editor and change the table of characters to represent the strings you want each key to generate. Remember, you don't have to alter any of the count or pointer values. This is taken care of automatically by the assembler. The sample version I've given uses only single-byte keycodes, but you can have

Figure 4 — UNMAP.ASM

```
;***********************************************
;*                                             *
;*THIS SMALL PROGRAM RESTORES THE CHARACTER    *
;*TRANSLATION TABLE FOR THE KAYPRO CURSOR      *
;*KEYS AND THE NUMERIC KEYPAD TO THE STANDARD  *
;*VALUES.                                      *
;*                                             *
;***********************************************

; *** CHANGE THIS TO MATCH THE CP/M SIZE ***

SIZE    EQU     63      ;CPM SIZE IN K

CBASE   EQU     SIZE*1024-1C00H;BEGINNING OF CCP
FBASE   EQU     CBASE + 800H  ;BEGINNING OF BDOS

; *** OR CHANGE THIS TO MATCH YOUR BIOS ***

BIOS    EQU     CBASE + 1600H;BEGINNING OF BIOS

VTAB    EQU     BIOS+35H;BEGINNING OF PARAMETERS

        ORG     100H

UNMAP:  LXI     H, MAP;ADDRESS TRANSLATION TABLE
        LXI     D, VTAB ;ADDRESS CP/M BIOS TABLE
        MVI     B, 18   ;SET BYTE COUNT
LOOP:   MOV     A, M    ;COPY TABLE INTO BIOS
        STAX    D
        INX     H
        INX     D
        DCR     B
        JNZ     LOOP
        RET             ;RETURN TO CCP
;
; TRANSLATION TABLE TO BE LOADED
;
MAP:    DB      11,10,8,12,'0123456789-,',0DH,'.'
        END

END OF LISTING
```

Figure 5 — MAPU.ASM

```
;***********************************************
;*                                             *
;*THIS SMALL PROGRAM GENERATES A CHARACTER     *
;*TRANSLATION TABLE FOR THE KAYPRO CURSOR      *
;*KEYS AND THE NUMERIC KEYPAD.  IT INSTALLS THE*
;*TABLE INTO THE CP/M BIOS.  THE TRANSLATION   *
;*WILL REMAIN IN FORCE UNTIL THE NEXT COLD     *
;*(NOT WARM) BOOT, OR UNTIL THE COMPLEMENTARY  *
;*PROGRAM 'UNMAPU' IS RUN.                     *
;*                                             *
;*THIS IS A SPECIAL VERSION FOR CP/M 2.2U      *
;*                                             *
;***********************************************

SIZE    EQU     61      ;CPM SIZE IN K
CBASE   EQU     SIZE*1024-1C00H;BEGINNING OF CCP
FBASE   EQU     CBASE + 800H ;BEGINNING OF BDOS
BIOS    EQU     CBASE + 1600H;BEGINNING OF BIOS
KTAB    EQU     BIOS+244H;BEGIN OF KEYPAD TABLE
VTAB    EQU     BIOS+343H;BEGIN OF CURSOR TABLE

        ORG     100H

MAPKEY: LXI     H, MAP;ADDRESS TRANSLATION TABLE
        LXI     D, VTAB;ADDRESS CP/M CURSOR TABLE
        MVI     B, 4    ;SET BYTE COUNT
LOOP1:  MOV     A, M    ;COPY TABLE INTO BIOS
        STAX    D
        INX     H
        INX     D
        DCR     B
```

strings as long as you like. Just don't exceed the total byte limit of 223.

Cloak And Dagger

You may have been wondering how I found out all this about 2.2u. Did I have some inside info from Kaypro? No, I just found out how to get CONFIG to tell me what it was doing. The method can be used on any version of CP/M, no matter how bizarre your system is, and I guarantee that it will help find all the CONFIG generated changes.

The concept goes like this: CONFIG changes the character codes that correspond to keypad and cursor key codes, right? So all we have to do is to let CONFIG do its thing, and then look and see what changed. Simple, huh?

Now, it would be pretty tedious to compare two versions of CP/M, byte by byte. But there are programs that can compare two files. Mine is called DI.COM, an excellent program written in C that comes with the BDS C compiler. There are several other similar tools available in the public domain.

DI and its kin are designed to compare files, not BIOSes. So before we can compare the BIOSes, we need to turn them into files. There are a couple of ways to do that. The most straightforward is to use SYSGEN, a utility that comes with almost all CP/M systems.

One of the functions of SYSGEN is to provide a RAM copy of CP/M in low memory, suitable for capture to a file with SAVE. Unfortunately, that "almost all" phrase above doesn't include later model Kaypros, in particular those with 2.2u. On to plan B.

What we need is a program that will copy the BIOS into low RAM, preferably beginning at 100H, where it can be SAVEd to a file. Such a program is useful in other areas, too. For example, I've used one for years to capture the BIOS for disassembling.

The program I use is shown in Figure 6. What it does takes place in two stages. First, it relocates a portion of itself into high RAM, at 8000H, so it won't be overwritten by the BIOS image. Then it executes this relocated code, which copies the BIOS into low RAM at 100H, the normal CP/M TPA. Then the program just returns to CP/M's command level.

At this point, the BIOS is still there at 100H, and can be written to disk with a SAVE 18. Actually, you don't need to save this many pages — three or four will do. But 18 pages saves everything between EE00H and the top of RAM.

(continued from page 87)

```
        JNZ     LOOP1
        LXI     D, KTAB ;ADDRESS CP/M BIOS TABLE
        MVI     B, COUNT;SET BYTE COUNT
LOOP2:  MOV     A, M    ;COPY TABLE INTO BIOS
        STAX    D
        INX     H
        INX     D
        DCR     B
        JNZ     LOOP2
        RET             ;RETURN TO CCP
;
; TRANSLATION TABLE FOR CURSOR KEYS
;
MAP:    DB      'E'-'@' ;UP ARROW( ^E )
        DB      'X'-'@' ;DOWN ARROW( ^X )
        DB      'S'-'@' ;LEFT ARROW( ^S )
        DB      'D'-'@' ;RIGHT ARROW( ^D )

; TABLE OF COUNTS AND OFFSETS

; THIS TABLE BEGINS THE DATA LOADED INTO THE BIOS.
; DON'T CHANGE THIS TABLE EXPLICITLY. ITS VALUES
; WILL BE ADJUSTED TO MATCH THE CHARACTER TABLE.

KMAP:   DB      CNT1
        DB      OFF1
        DB      CNT2
        DB      OFF2
        DB      CNT3
        DB      OFF3
        DB      CNT4
        DB      OFF4
        DB      CNT5
        DB      OFF5
        DB      CNT6
        DB      OFF6
        DB      CNT7
        DB      OFF7
        DB      CNT8
        DB      OFF8
        DB      CNT9
        DB      OFF9
        DB      CNT10
        DB      OFF10
        DB      CNT11
        DB      OFF11
        DB      CNT12
        DB      OFF12
        DB      CNT13
        DB      OFF13
        DB      CNT14
        DB      OFF14

;***********************************************
;*                                             *
;* FOLLOWING DATA MAY BE CHANGED FOR THE KEYPAD *
;* TRANSLATIONS DESIRED.                       *
;*                                             *
;***********************************************

; BYTE DATA FOR THE MAPS
; CHANGE THIS DATA AS DESIRED FOR THE APPLICATION
; DATA MAY BE ONE OR MORE BYTES PER KEY, AS LONG
; AS TOTAL NUMBER OF BYTES DOES NOT EXCEED 223.

K1:     DB      'Q'-'@' ;NUM KEY '0'( ^Q )
K2:     DB      'Z'-'@' ;NUM KEY '1'( ^Z )
K3:     DB      'C'-'@' ;NUM KEY '2'( ^C )
K4:     DB      'B'-'@' ;NUM KEY '3'( ^B )
K5:     DB      'A'-'@' ;NUM KEY '4'( ^A )
K6:     DB      'L'-'@' ;NUM KEY '5'( ^L )
K7:     DB      'F'-'@' ;NUM KEY '6'( ^F )
K8:     DB      'W'-'@' ;NUM KEY '7'( ^W )
K9:     DB      'R'-'@' ;NUM KEY '8'( ^R )
K10:    DB      'T'-'@' ;NUM KEY '9'( ^T )
K11:    DB      'Y'-'@' ;NUM KEY '-'( ^Y )
K12:    DB      'G'-'@' ;NUM KEY ','( ^G )
K13:    DB      'K'-'@' ;NUM KEY 'ENTER'( ^K )
K14:    DB      'V'-'@' ;NUM KEY '.'( ^V )
FIN:    DS      0       ;DUMMY FOR ADDR CALCS

COUNT   EQU     FIN - KMAP      ;#BYTES TO COPY
;
; BYTE COUNTS FOR TRANSLATIONS
;
CNT1    EQU     K2 - K1
```

So here's the procedure. (Remember, you don't have to do this. Only if you want to satisfy your curiosity.) Starting with a disk with normal keypad mappings, type the following commands:

```
A0>SNAP
A0>SAVE 18 UNMAPPED.COM
```

You now have a file called UNMAPPED.COM, with an image of your BIOS on it. Run CONFIG, setting up any mappings you like. Exit CONFIG and press the reset button to activate the changes. Now run SNAP and SAVE, as before, to create MAPPED.COM.

Now, use the file compare program of your choice to look for differences. For DI, the command is:

```
A0>DI MAPPED.COM UNMAPPED.COM
```

Don't be surprised if you see many differences. There are several things, such as uninitialized data or input buffers, that aren't really part of the BIOS. Look for the differences that correspond to your keypad mappings. And you probably should concentrate on differences in the lower addresses.

One small item: the file compare program doesn't know where your BIOS was originally stored in RAM. As far as it knows, it's dealing with an ordinary COM file that begins at location 100H. So the addresses that it reports won't be the actual addresses in the BIOS, but will be offset by a value (address of BIOS - 100H).

To get the actual addresses in the real BIOS, just add this value to the addresses reported by your file compare program. Using this approach will let you find any changes that CONFIG or any similar program makes to your BIOS.

Conclusion

I've been using MAP and UNMAP on two machines now for over four years. They've always worked fine for me, even though both of my Kaypros have different ROMs and aftermarket goodies that require modified BIOSes.

For the purposes of this article, I went through the process again for CP/M 2.2u, and these programs work fine also. I haven't used CONFIG in months, and I'll bet you won't either. Enjoy.

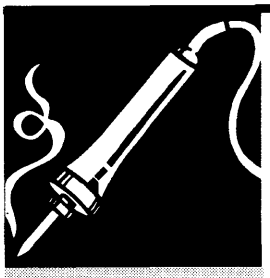
♦ ♦ ♦

(continued from page 88)

```
CNT2    EQU     K3 - K2
CNT3    EQU     K4 - K3
CNT4    EQU     K5 - K4
CNT5    EQU     K6 - K5
CNT6    EQU     K7 - K6
CNT7    EQU     K8 - K7
CNT8    EQU     K9 - K8
CNT9    EQU     K10 - K9
CNT10   EQU     K11 - K10
CNT11   EQU     K12 - K11
CNT12   EQU     K13 - K12
CNT13   EQU     K14 - K13
CNT14   EQU     FIN - K14
;
; OFFSETS FROM BEGINNING OF TABLE
;
OFF1    EQU     01CH
OFF2    EQU     OFF1 + CNT1
OFF3    EQU     OFF2 + CNT2
OFF4    EQU     OFF3 + CNT3
OFF5    EQU     OFF4 + CNT4
OFF6    EQU     OFF5 + CNT5
OFF7    EQU     OFF6 + CNT6
OFF8    EQU     OFF7 + CNT7
OFF9    EQU     OFF8 + CNT8
OFF10   EQU     OFF9 + CNT9
OFF11   EQU     OFF10 + CNT10
OFF12   EQU     OFF11 + CNT11
OFF13   EQU     OFF12 + CNT12
OFF14   EQU     OFF13 + CNT13
        END
```

Figure 6 — SNAP.ASM

```
;***********************************************
;*                                             *
;* THIS PROGRAM MAKES A COPY OF THE CP/M BIOS  *
;* IN LOW RAM, SUITABLE FOR CAPTURE WITH SAVE. *
;* AFTER RUNNING SNAP, TYPE                    *
;*                                             *
;* SAVE 18 NAME.COM                            *
;*                                             *
;***********************************************

TPA     EQU     100H    ;ADDRESS OF TPA
TARGET  EQU     8000H   ;ADDRESS TO RELOCATE TO
BIOS    EQU     0EE00H  ;ADDRESS OF BIOS

        ORG     TPA

SNAP:   LXI     H, IMAGE;POINT TO CODE TO MOVE
        LXI     D, TARGET       ;POINT TO TARGET
        MVI     B, FIN - IMAGE  ;SET BYTE COUNT
LOOP1:  MOV     A, M            ;GET A BYTE
        STAX    D       ;STORE IT
        INX     H       ;BUMP POINTERS
        INX     D
        DCR     B       ;DECREMENT COUNT
        JNZ     LOOP1   ;LOOP TILL DONE
        JMP     TARGET  ;JUMP TO RELOCATED CODE
;
; IMAGE OF CODE TO BE RELOCATED
;
IMAGE:  LXI     H, BIOS ;POINT TO BIOS
        LXI     D, TPA
        LXI     B, - BIOS       ;BYTE COUNT
LOOP2:  MOV     A, M    ;GET A BYTE
        STAX    D       ;STORE IT
        INX     H       ;BUMP POINTERS
        INX     D
        DCX     B       ;DECREMENT COUNT
        MOV     A, B    ;SEE IF WE'RE DONE
        ORA     C

; THE NEXT CONSTANT IS A PHONY JUMP ADDRESS TO
; FAKE ASM INTO DEALING WITH THE RELOCATION
PHONY   EQU     LOOP2 + TARGET - IMAGE
        JNZ     PHONY   ;JMP TO RELOCATED LOOP
        RET
FIN:    DS      0       ;DUMMY FOR ADDR CALC
        END

END OF LISTING
```



TECHTIPS

Techtips

LPT2 Problems

When your clone boots, the INIT line to LPT1 (pin 31) goes low, then high. That does the initialization on the printer (it burps). But the BIOS doesn't initialize LPT2. On this printer interface, the INIT line stays low so the printer may never go on line.

Epson and Star printers won't fly as long as INIT stays low. The Panasonics ignore that line so they work. You *can* make Epsons and Stars work on LPT2 by cutting the INIT line. But the better fix would be to write a routine to raise INIT so you have a proper initialization.

Dick Towle
P.O. Box 779
Alta, CA 95701

Beyond 360K Floppy Drives

Starting with DOS version 3.2, clones have had the capability to access higher capacity floppy drives. If you hang one of these babies on your system, you'll need to install a device driver to let everyone talk intelligently.

Assuming DRIVER.SYS lives in the root directory (where else would you put it?), complete syntax for the new line in CONFIG.SYS goes like this:

```
DEVICE = DRIVER.SYS /D:d /F:f /T:t /S:s /H:h
/C /N
```

d specifies the physical drive number, 0 through 2 for drives A through C. The f parameter sets up the kind of drive. Use zero for up to 360K, or two for 720K, 3 1/2" drives. These two drive types are plain old vanilla double density so your PC or XT can run them.

You'll need a 286 or better to use the higher density drives. For 1.2M drives set f to 1, and for 1.44M f should be 7. If not specified, this parameter defaults to 2, the 720K drive.

Next comes the number of tracks. t defaults to 80 tracks — the correct value for all drives greater than 360K. Since you won't use DRIVER.SYS to install extra 360s, you shouldn't have to set t.

Both 360K and 720K drives have 9 sectors per track, the default value for s. For 1.2M use 15 sectors and for 1.44M use 18. The last parameter, h, should be set to 1.7 for best results. Just kidding, just kidding: any floppy drive you install will likely have 2 heads (one for guests). Two is the default.

The /C switch tells DRIVER.SYS that the drive can sense a change of disks. Only 286s and up have this capability. Finally, /P indicates nonremovable media — a hard disk. So don't use this switch.

Let's roll it all together for a couple of examples. To install a 720K drive B in an XT clone, enter the following line in CONFIG.SYS:

```
DEVICE = DRIVER.SYS /D:1 /F:2
```

Actually, you don't even need the "/F:2". All defaults are correct. But your CONFIG.SYS will make more sense if you at least set the drive type.

And for a 1.2M drive B, 286 combo enter:

```
DEVICE = DRIVER.SYS /D:1 /F:1 /S:15 /C
```

According to an article in the May, 1987, issue of *PC World* (p. 294), you can accomplish the same thing as the first example using this CONFIG.SYS line:

```
DRIVPARM=/D:1/F:2
```

Whichever method you use, be sure to install this driver before any other disk DEVICEs. For example, a RAM disk installed with VDISK.SYS will take the next available drive label — say B on a single floppy system. If you then try to install a 720K as drive B, the fur will fly. Setting up the 720K first ensures that the RAM disk will become drive C (or D if a hard disk exists).

Micro C Staff

(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.

Ventura Graphics Tip

I recently encountered what I thought was a minor catastrophe within Ventura Publisher. I called Xerox technical support. The lady I spoke to was extremely courteous and well versed. We worked for 10 minutes or more, to no avail. She finally contacted a "specialist" who informed her that what I needed to undo could not be undone!

I had done the unthinkable. At the eleventh hour of a lengthy project, I had used Ventura's graphics function to draw a very detailed and annotated full page drawing.

Unfortunately, I had accidentally drawn it on the "underlying page." It had permeated my entire document from beginning to end. Worst of all, I could not find a way to remove it without destroying it.

My problems began when I created a frame for my drawing. The option to have text flow around the frame was on. With the frame drawn, I now had a clear page for my drawing.

Somewhere I must have got distracted. My entire drawing and all of its extensive labels ended up on the underlying page rather than in the frame I had opened. In a state of panic I searched the Ventura manuals. I found great assurances like, "Graphics tied to the underlying page will automatically repeat on every page."

The solution: if you find that you've placed graphics directly onto one of Ventura's underlying pages, don't panic! It can be moved into a frame without too much difficulty. Just follow the steps below:

(1) Make sure your entire document is saved.

(2) Save your work again, but under a different name. (Joe Smith is a good name.)

(3) Go to the page where your drawing is.

(4) If you haven't done so already, draw a frame around your drawing. Make sure that your frame does not extend to the very edges of the page. If your drawing does, then draw the frame slightly inside the edges of your drawing. You'll be able to expand it later.

(5) While in the FRAME function, select the underlying page. To do so, just move to the edge of your page and press your mouse button, or other selector. Black markers should appear on the outermost edges of the selected page.

(6) Select the GRAPHIC DRAWING function.

(7) Choose the GRAPHIC menu. Select the SELECT ALL choice. This will mark everything you have drawn.

(8) Choose the EDIT menu. Select the COPY GRAPHIC option. When you do this, nothing will appear to have happened. Nevertheless, your drawing is now in the graphics "clipboard."

(9) Go back to the FRAME function. Move your marker inside of the frame you drew in step 4 above. The frame you want to move your drawing into should now have dark black markers at the edges.

(10) Go back to the GRAPHIC function.

(11) Choose the EDIT menu. Select the PASTE GRAPHIC option. Again, nothing will appear to happen. But you have now copied your drawing from the clipboard onto your frame.

(12) Return to the FRAME function. Move your frame. You will notice that you now have the drawing on the underlying page and in the frame.

(13) Go to the edge of your page and select the underlying page.

(14) Return to the GRAPHICS function.

(15) Choose the GRAPHIC menu to SELECT ALL of your old drawing.

(16) Choose the EDIT menu. Select the CUT GRAPHIC option. The drawing on the underlying page will be erased.

(17) Return to the FRAME function.

(18) Move your frame into place. You are now back to where you should have been in the first place.

If anyone knows of a simpler method to move a drawing done in Ventura's GRAPHICS mode from the underlying page, or from an undesired frame, to a new frame, I would appreciate hearing from you.

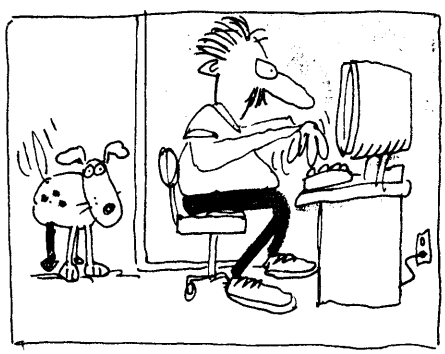
Kenneth J. Grymala
P.O. Box 273
Nokesville, VA 22123

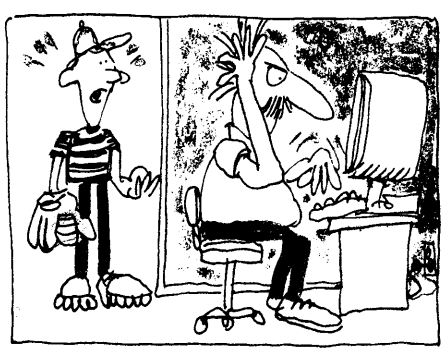
♦ ♦ ♦

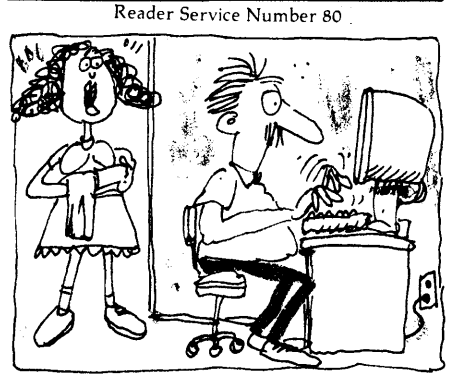
Reader Service No. 46

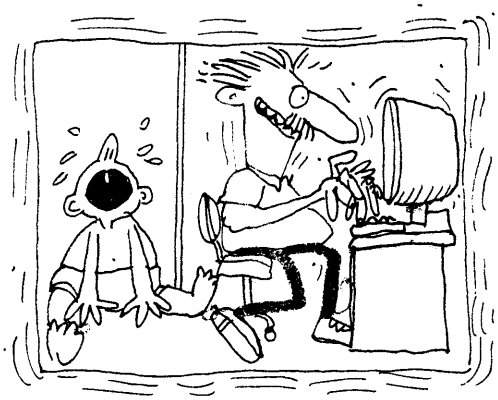
Reader Service Number 54

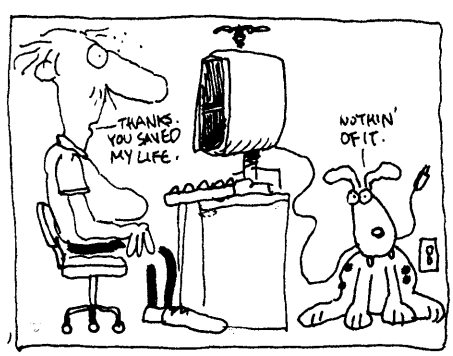
Reader Service Number 80











Is There A Gap In Your Info?

Fill in your Back Issues of Micro C today!

ISSUE #1 (8/81)
Power Supply
RAM Protection
Video Wiggle
1/2 PFM.PRN
16 pages

ISSUE #2 (10/81)
Parallel Print Driver
Drive Motor Control
Shugart Jumpers
Program Storage Above PFM
1/2 PFM.PRN
16 pages

ISSUE #3 (12/81)
4 MHz Mods
Configuring Modem 7
Safer Formatter
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
Keyboard Encoder
20 pages

ISSUE #5 (4/82)
Word Processing
Two Great Spells
Two Text Editors
Double Density Review
Scribble, A Formatter
20 pages

ISSUE #6 (6/82)
BBI EPROM Programmer
Customize Your Chars
Double Density Update
Terminal In FORTH
24 pages

ISSUE #7 (8/82)
6 Reviews Of C
Adding 6K Of RAM
Viewing 50 Hz
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
Bringing Up WordStar
Cheap RAM Disk
32 pages

ISSUE #10 (2/83)
SOLD OUT

ISSUE #11 (4/83)
SOLD OUT

ISSUE #12 (6/83)
256K for BBI
Bringing Up BBII
dBase II
Look at WordStar
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
Nevada COBOL Review
BBI Printer Interface
Kaypro Reverse Video Mod
44 pages

ISSUE #14 (10/83)
BBII Installation
The Perfect Terminal
Interface To Electronic Typewriter
BBI Video Size
Video Jitter Fix
Slicer Column Begins
Kaypro Color Graphics Review
48 pages

ISSUE #15 (12/83)
Screen Dump Listing
Fixing Serial Ports
Playing Adventure
SBASIC Column Begins
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
Adding Joystick To Color Graphics
Recovering Text From Memory
52 pages

ISSUE #17 (4/84)
Voice Synthesizer
820 RAM Disk
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
Kaypro Joystick
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
In the Public Domain Begins
88 pages

ISSUE #23 (4/85)
Automatic Disk Relogging
Interrupt Drive Serial Printer
Low Cost EPROM Eraser
Smart Video Controller
Review: MicroSphere RAM Disk
Future Tense Begins
86 pages

ISSUE #24 (6/85)
C'ing Into Turbo Pascal
8" Drives On The Kaypro
48 Lines On A BBI
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)
Pascal Runoff Winners
Rescuing Lost Text From Memory
Introduction To Modula-2
First Look At Amiga
Inside The PC
104 pages

ISSUE #29 (4/86)
Speeding Up Your XT
Importing Systems From Taiwan
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
256K RAM For Your 83 Kaypro
PC-DOS For Non-Clones
104 pages

ISSUE #31 (8/86)
RAM Resident PC Speedup
Practical Programming In Modula-2
Unblinking The PC's Blinkin' Cursor
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)
SOLD OUT

ISSUE #34 (2/87)
SOLD OUT

ISSUE #35 (4/87)
SOLD OUT

ISSUE #36 (6/87)
Build A Midi Interface For Your PC
Designing A Database, Part 2
Interrupts On The PC
Hacker's View of MS-DOS Vs 3.X
Digital To Analog Conversion, A Designer's View
96 pages

ISSUE #37 (9/87)
Desktop Publishing On A PC
Build Your Own Hi-Res Graphics Scanner For $6, Part 1
Designing A Database, Part 3
Uninterruptable Power Supply For RAM Disks
96 pages

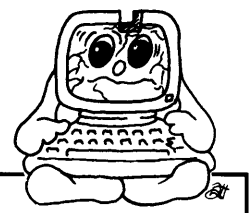
ISSUE #38 (11/87)
Parallel Processing
Laser Printers, Typesetters And Page Definition Languages
Magic In The Real World
Build A Graphics Scanner For $6, Part 2
Writing A Resident Program Extractor In C
96 pages

ISSUE #39 (1/88)
PC Graphics
Drawing The Mandelbrot And Julia Sets
Desktop Graphics
Designing A PC Workstation Board
Around the TMS-3410
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 (6/88)
Maintaining PC's
Keeping Your Hard Drives Running
Troubleshooting PC's
XT Theory of Operation
Simulating A Bus
Ray Tracing
96 pages



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.

Advertisers Index

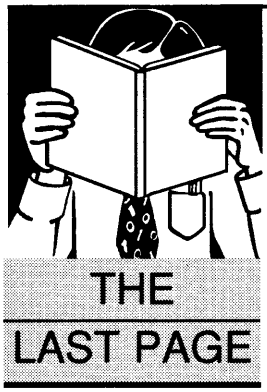
Issue 43

Reader Service	Page Number
72	Acquired Intelligence 39
107	American Cosmotron 81
**	Andsor Research 62
04	Austin Codeworks 13
**	Autotime 59
87	Brown Bag Software . 46,65,75,79
104	Bytel 5
15	Cascade Electronics 34
**	CC Software 75
71	Complete Logic Systems 11
105	Computerized Processing Unltd. 16
07	CompuView . . Inside Front Cover
32	Digital Research Computers . 68, 78
16	Dreamtech 17
09	Ecosoft Inc 40
38	E.M. Enterprise 12
10	Emerald Microware 41
93	Erac Company 33
11	Halted Specialties 2
22	Integrand 21
88	Koala Computers 25
12	Logitech Inc 26,27
17	Manx Software Systems 76
103	Max Software 46
42	McTek Systems 55
**	Micro Cornucopia 78,80
37	Microprocessors Unltd 59
24	microSOLUTIONS 63
02	Microsphere 01,67
59	National Advancement Corp 79
85	Oregon Software 60
03	PC Tech . . . . . Cover
68	Programmers Paradise 7
60	Scientific Software Solutions . . 65
27	Serengeti Software 68
**	Sofsolutions 29
19	Slicer Control/Computer 77
40	Star-K Software Systems 54
101	United Products 35
62	V Communications 47
39	Xenosoft 60
70	Zortech, Inc. . . Inside Back Cover

** Contact Advertiser Directly.

Coming in Issue #44, Object Oriented Programming

- **Actor: An Object Oriented Development System for MicroSoft Windows**
- **Smalltalk: State of the Art Object Oriented Programming?**
- **C++: Objects for C'ers**
- **Designing (and Building) a PC Robot**
- **Review of the PT68K-2 Computer**
- **Debugging Tools for C Programmers**



Turbo Prolog 2.0 - Cresting The New Wave

By Gary Entsminger
1912 Haussler Dr.
Davis, CA 95616

While many of you have been basking on the shores of some outstanding new Cs (Turbo, Quick, Datalight, Mark Williams, and the like), Gary's been beta-testing the latest version of Turbo Prolog. And, to put it mildly, he likes what he sees.

Turbo Prolog 2.0 starts quickly, and goes into a slick, easy-to-handle user interface of pulldown and popup menus. It drops into a better-than-average Turbo editor so you can enter your program, and then gets you results (via the interpreter) faster than any compiler I know.

There's plenty of help on line (including a listing of *all* the built-in predicates). And compile to EXE is quite fast. Execution times are excellent.

If you've been thinking about learning Prolog, Turbo P 2.0 is to date your best chance. The two manuals (Turbo C size) have enough examples, ideas, and wisdom to bring any serious programming hacker in. That is — any serious *and logical* programming hacker. (I'm still not sure where the *ill* and *nonlogical* fit in.)

If you've been using Turbo Prolog 1.0 or 1.1, you'll find much that's been improved or added to 2.0. For starters —

(1) A better user interface (you can redefine hot & command keys; and use Pick files, etc.);

(2) An external, indexed, database system which supports EMS memory;

(3) BGI graphics (identical to Turbo C's and Pascal's and very powerful);

(4) Error trapping & exception handling (which makes it possible to add useful error correction at RUNTIME);

(5) Extended error control in EXE files (you can set the error level);

(6) Foreign language calls to Turbo Prolog (In Turbo P 1.1, you could call functions written in other languages, but they couldn't call TP. Now other languages, like Turbo C, can call functions written in Turbo Prolog. In fact, you can call the functions in the Turbo Prolog library from other languages. Very interesting!);

(7) High resolution text modes (the video system now supports 43 or 50 rows by 90, 120, or 132 columns);

(8) Improved code generation (Programs compiled in 2.0 tend to be 20-25% smaller than those compiled in 1.1. The smallest Turbo Prolog program is now about 12K);

(9) Memory allocation control (So you can specify how much memory your EXE application should allocate from the operating system. This opens the door for TSRs at a very high level!);

(10) And many new predefined domains and predicates.

As you can tell, I'm very excited about the new Prolog, having watched it grow better and better during the past year of beta-testing.

Out Of Here

And speaking of beta-testing, I believe Bruce Eckel is going to have a lot to say about this strange art and a hot new C++ compiler next issue, when we tackle objects and object-oriented programming.

This new wave of higher level programming languages will certainly make it easier for other-than-whizzes to develop sophisticated applications.

Programmers will still have to know a lot, but they won't have to know as much. And a lot of the programmers I know (no names, please) will be glad to hear it.

So stay tuned to *Micro Cornucopia*, where we either ride the waves or crash into them.

Cheers.

♦ ♦ ♦

C++

FROM ZORTECH INC.

The change to a pure language

Now, C programmers can move over to C++ with Zortech C++ – the world's first 'true' C++ compiler for MS-DOS machines.

Zortech C++ is a 'true' compiler and fully conforms to Bjarne Stroustrup's specification as outlined in his book 'The C++ Programming Language'.

Previous implementations of C++ were actually 'translators' – only able to translate C++ source code into C. Of course, this was unacceptable due to the long translating and compiling times.

Now, C++ comes of age with the introduction of the world's first true C++ compiler – from Zortech!

■ *Object Oriented Programming*

C++ is to C what Modula 2 is to Pascal. C++ brings 'classes' to C, so you can create separate modules that contain their own data and data-related operations. These 'classes' then become new types that can in turn be used to create further modules – this allows you to practically create your own language.

■ *ANSI C Superset*

You don't have to throw away your existing C programs – C++ is a superset of ANSI C. Now, you can take your Microsoft C or Turbo C compatible programs and easily migrate to C++ to take full advantage of the new C++ features.

■ *'Codeview' Compatible*

Zortech C++ is compatible with 'Codeview' – Microsoft's industry standard source code debugger.

■ *Improved Program Structure*

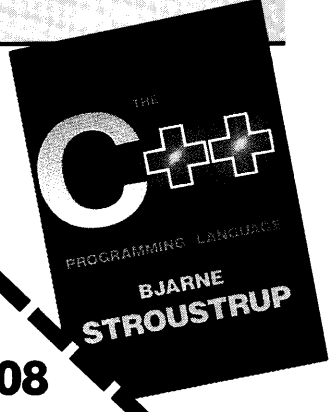
As stated in 'The C++ Programming Language', by using C++ "It would not be unreasonable for a single person to cope with 25,000 lines of code".

■ *Other benefits*

Here's just a few: Operator overloading, overloading function names, default arguments to functions and better type checking.

ESSENTIAL READING!

This 325 page book 'The C++ Programming Language' by Bjarne Stroustrup contains the original definition of C++. All the examples shown in this book have been successfully compiled and executed with the Zortech C++ Compiler.



ZORTECH
BOSTON · LONDON · FRANKFURT · GENEVA

YES!
Rush me C++ as shown below:

☐ *Zortech C++* *$99.95* ☐ *C++ Book* *$29.95*

VISA/MC/COD/CHECK ACCEPTED

Name.....

Address.....

..... *Phone*.....

VISA or MC..... *Exp. Date*.....

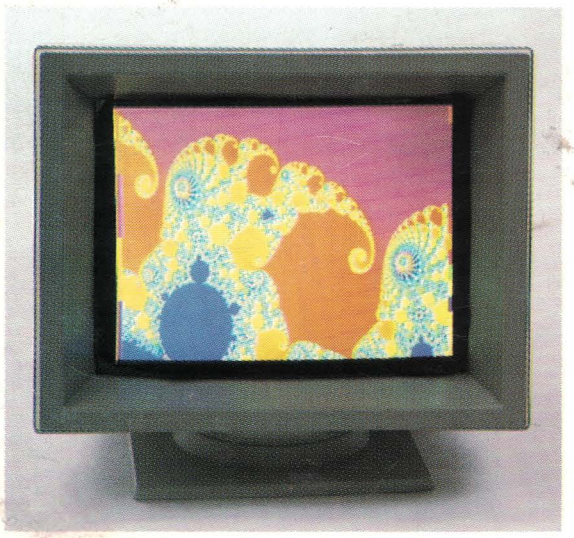
To: ZORTECH INC. 361 Massachusetts Ave., Arlington, MA 02174.
Tel: 617-646-6703. Fax: 617-648-0603.

CALL THE ORDER HOTLINE 1-800-848-8408

Reader Service Number 70

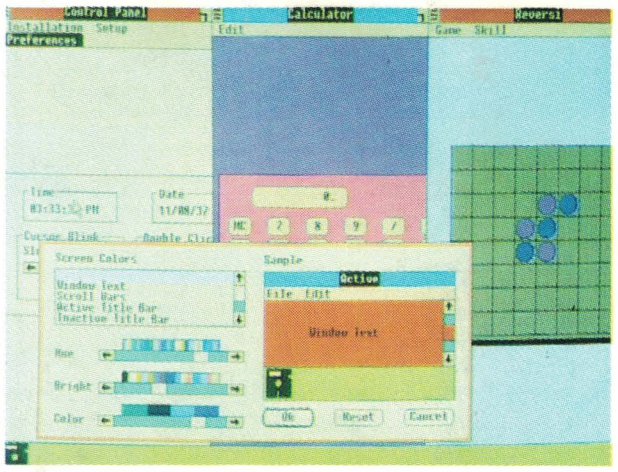
VERY HIGH RESOLUTION

The PC Tech COLOR and MONOCHROME video processor boards employ the TMS 34010 high performance graphics co-processor to insure the best possible video performance at reasonable prices.



Color 34010 Video Processor:

- Featured on the cover of Micro Cornucopia.
- From 800 x 512 through 1024 x 800 resolution (depending on monitor and configuration).
- 8 Bits per pixel for 256 simultaneous colors
- Hardware support for CGA/MDA emulation.
- PC, XT, and AT compatible



The PC Tech Color 34010 video processor is a superior 34010 native code and DGIS development tool. We support up to 4 megabytes of program (non-display) 34010 RAM as well as up to 768K bytes of display RAM. ***Compare our architecture and prices to any other intelligent graphics board. Then choose the PC Tech Color 34010 Video Processor for your development engine and your production requirements as well.***

Color 34010 Video Processor $1,195.00

Price includes 512K display RAM, 1024K program RAM, and utility software. Monitor not included.
Also available: DGIS, 34010 C compiler, assembler, 34010 fractal software, additional display and program memory, and various monitor options.

PC Tech Monochrome 34010 Video Processor and Monitor

- 736 x 1024 resolution (other options available)
- 2 bits per pixel for 4 hardware gray shades
- Hardware support for CGA/MDA/Hercules emulation
- PC, XT, and AT compatible
- Full page 66 line text editing with many popular editors
- Excellent windows 2.0 application development system

The graphics and bit manipulation capabilities of the TMS 34010 make the PC Tech Monochrome 34010 Video Processor 66 line full page text and graphics display faster than many 25 line systems. The video processor is available separately or with the high resolution white phosphor monitor shown above.

Monochrome 34010 Video Sub-System $1,295.00

Price includes Monochrome Video Processor and monitor pictured above.
Also available: DGIS, TI 34010 C compiler, TI assembler.
Monochrome 34010 Video Processor also available separately.

Designed, Sold and Serviced By:

PTech

904 N. 6th St.
Lake City, MN 55041
(612) 345-4555
(612) 345-5514 (FAX)

Reader Service Number 3