June 1991

REMark®

The Official Zenith Data Systems Users Magazine

Share the Knowledge!

Have you done something interesting with your computer lately? Found a piece of software or hardware you don't know how you got along without? Designed a new product, be it software or hardware, for your system? By submitting this information in the form of a major article, you can share with others the knowledge of a particular subject. REMark magazine is currently looking for authors (novice or professional) to write articles. Even if you have never written before, give it a try! As a REMark author, you will recieve up to $400 for each article accepted and published. (For more information on current policies, call Lori Lerch at 616-982-3794.)

So... Let's get to it and

Share the Knowledge!

June 1991

REMark®

The Official Zenith Data Systems Users Magazine

Advertising

Reader Service No.		Page No.
104	FBE Research Co., Inc.	48
193	QuikData, Inc.	32
149	WS Electronics	31

Resources

Managing Editor
Jim Buszkiewicz
(616) 982-3837

Software Engineer
Pat Swayne
(616) 982-3463

Production Coordinator
Lori Lerch
(616) 982-3794

Secretary
Lisa Cobb
(616) 982-3463

COM1 Bulletin Board
(616) 982-3956
(Modem Only)

ZUG Software Orders
(616) 982-3463

Contributing Editor
William M. Adney

Printer
Imperial Printing
St. Joseph, MI

Contributing Editor
Robert C. Brenner

Advertising
Rupley's Advertising Service
Dept. REM, 240 Ward Avenue
P.O. Box 348
St. Joseph, MI 49085-0348
(616) 983-4550

	U.S. Domestic	APO/FPO & All Others
Initial	$22.95	$37.95*
Renewal	$19.95	$32.95*

* U.S. Funds

Limited back issues are available at $2.50, plus 10% shipping and handling - minimum $1.00 charge. Check ZUG Product List for availability of bound volumes of past issues. Requests for magazines mailed to foreign countries should specify mailing method and include appropriate, additional cost.

Send Payment to: Zenith Users' Group
P.O. Box 217
Benton Harbor, MI 49023-0217
(616) 982-3463

Software

PRODUCT NAME	PART NUMBER	OPERATING SYSTEM	DESCRIPTION	PRICE
	H8 - H/Z-89/90			
ACTION GAMES	885-1220-[37]	CPM	GAME	20.00
ADVENTURE	885-1010	HDOS	GAME	10.00
ASCIRITY	885-1238-[37]	CPM	AMATEUR RADIO	20.00
AUTOFILE (Z80 ONLY)	885-1110	HDOS	DBMS	30.00
BHBASIC SUPPORT PKG	885-1119-[37]	HDOS	UTILITY	20.00
CASTLE	885-8032-[37]	HDOS	ENTERTAINMENT	20.00
CHEAPCALC	885-1131-[37]	HDOS	SPREADSHEET	20.00
CHECKOFF	885-8010	HDOS	CHKBK SOFTWARE	25.00
DEVICE DRIVERS	885-1105	HDOS	UTILITY	20.00
DISK UTILITIES	885-1213-[37]	CPM	UTILITY	20.00
DUNGEONS & DRAGONS	885-1093-[37]	HDOS	GAME	20.00
FLOATING POINT PKG	885-1063	HDOS	UTILITY	18.00
GALACTIC WARRIORS	885-8009-[37]	HDOS	GAME	20.00
GALACTIC WARRIORS	885-8009-[37]	CPM	GAME	20.00
GAMES 1	885-1029-[37]	HDOS	GAMES	18.00
HARD SECT SUPPORT PKG	885-1121	HDOS	UTILITY	30.00
HDOS PROG. HELPER	885-8017	HDOS	UTILITY	16.00
HOME FINANCE	885-1070	HDOS	BUSINESS	18.00
HUG DISK DUP UTILITY	885-1217-[37]	CPM	UTILITY	20.00
HUG SOFTWARE CATALOG	885-4500	VARIOUS	PROD TO 1982	9.75
HUGMAN & MOVIE ANIM	885-1124	HDOS	ENTERTAINMENT	20.00
INFO SYS AND TEL. & MAIL SYS	885-1108-[37]	HDOS	DBMS	30.00
LOGBOOK	885-1107-[37]	HDOS	AMATEUR RADIO	30.00
MAGBASE	885-1249-[37]	CPM	MAGAZINE DB	25.00
MISCELLANEOUS UTILITIES	885-1089-[37]	HDOS	UTILITY	20.00
MORSE CODE TRANSCEIVER	885-8016	HDOS	AMATEUR RADIO	20.00
MORSE CODE TRANSCEIVER	885-8031-[37]	CPM	AMATEUR RADIO	20.00
PAGE EDITOR	885-1079-[37]	HDOS	UTILITY	25.00
PROGRAMS FOR PRINTERS	885-1082	HDOS	UTILITY	20.00
REMARK VOL 1 ISSUES 1-13	885-4001	N/A	1978 TO DEC '80	20.00
RUNOFF	885-1025	HDOS	TEXT PROCR	35.00
SCICALC	885-8027	HDOS	UTILITY	20.00
SMALL BUISNESS PACKAGE	885-1071-[37]	HDOS	BUSINESS	75.00
SMALL-C COMPILER	885-1134	HDOS	LANGUAGE	30.00
SOFT SECTOR SUPPORT PKG	885-1127-[37]	HDOS	UTILITY	20.00
STUDENT'S STATISTICS PKG	885-8021	HDOS	EDUCATION	20.00
SUBMIT (Z80 ONLY)	885-8006	HDOS	UTILITY	20.00
TERM & HTOC	885-1207-[37]	CPM	COMMUN & UTIL	20.00
TINY BASIC COMPILER	885-1132-[37]	HDOS	LANGUAGE	25.00
TINY PASCAL	885-1086-[37]	HDOS	LANGUAGE	20.00
UDUMP	885-8004	HDOS	UTILITY	35.00
UTILITIES	885-1212-[37]	CPM	UTILITY	20.00
UTILITIES BY PS	885-1126	HDOS	UTILITY	20.00
VARIETY PACKAGE	885-1135-[37]	HDOS	UTILITY & GAMES	20.00
WHEW UTILITIES	885-1120-[37]	HDOS	UTILITY	20.00
XMET ROBOT X-ASSEMBLER	885-1229-[37]	CPM	UTILITY	20.00
Z80 ASSEMBLER	885-1078-[37]	HDOS	UTILITY	25.00
Z80 DEBUGGING TOOL (ALDT)	885-1116	HDOS	UTILITY	20.00
	H8 - H/Z-89/90 - H/Z-100 (Not PC)			
ADVENTURE	885-1222-[37]	CPM	GAME	10.00
BASIC-E	885-1215-[37]	CPM	LANGUAGE	20.00
CASSINO GAMES	885-1227-[37]	CPM	GAME	20.00
CHEAPCALC	885-1233-[37]	CPM	SPREADSHEET	20.00
CHECKOFF	885-8011-[37]	CPM	CHKBK SOFTWARE	25.00
COPYDOS	885-1235-[37]	CPM	UTILITY	20.00
DISK DUMP & EDIT UTILITY	885-1225-[37]	CPM	UTILITY	30.00
DUNGEONS & DRAGONS	885-1209-[37]	CPM	GAMES	20.00
FAST ACTION GAMES	885-1228-[37]	CPM	GAME	20.00
FUN DISK I	885-1236-[37]	CPM	GAMES	20.00
FUN DISK II	885-1248-[37]	CPM	GAMES	35.00
GAMES DISK	885-1206-[37]	CPM	GAMES	20.00
GRADE	885-8036-[37]	CPM	GRADE BOOK	20.00
HRUN	885-1223-[37]	CPM	HDOS EMULATOR	40.00
HUG FILE MANAGER & UTILITIES	885-1246-[37]	CPM	UTILITY	20.00
HUG SOFTWARE CAT UPDT #1	885-4501	VARIOUS	PROD 1983 TO 1985	9.75
KEYMAP CPM-80	885-1230-[37]	CPM	UTILITY	20.00
MBASIC PAYROLL	885-1218-[37]	CPM	BUSINESS	60.00
NAVPROGSEVEN	885-1219-[37]	CPM	FLIGHT UTILITY	20.00
SEA BATTLE	885-1211-[37]	CPM	GAME	20.00
UTILITIES BY PS	885-1226-[37]	CPM	UTILITY	20.00
UTILITIES	885-1237-[37]	CPM	UTILITY	20.00
X-REFERENCE UTIL FOR MBASIC	885-1231-[37]	CPM	UTILITY	20.00
ZTERM	885-3003-[37]	CPM	COMMUNICATIONS	20.00

Price List

PRODUCT NAME	PART NUMBER	OPERATING SYSTEM	DESCRIPTION	PRICE
H/Z-100 (Not PC) Only				
CARDCAT	885-3021-37	MSDOS	BUSINESS	20.00
CHEAPCALC	885-3006-37	MSDOS	UTILITY	20.00
CHECKBOOK MANAGER	885-3013-37	MSDOS	BUSINESS	20.00
CP/EMULATOR	885-3007-37	MSDOS	CPM EMULATOR	20.00
DBZ	885-8034-37	MSDOS	DBMS	25.00
DUNGN & DRAGONS (ZBASIC)	885-3009-37	MSDOS	GAME	20.00
ETCHDUMP	885-3005-37	MSDOS	UTILITY	20.00
EZPLOT II	885-3049-37	MSDOS	PRINTER PLOT UTIL	25.00
GAMES (ZBASIC)	885-3011-37	MSDOS	GAMES	20.00
GAMES CONTEST PACKAGE	885-3017-37	MSDOS	GAMES	25.00
GAMES PACKAGE II	885-3044-37	MSDOS	GAMES	25.00
GRAPHIC GAMES (ZBASIC)	885-3004-37	MSDOS	GAMES	20.00
GRAPHICS	885-3031-37	MSDOS	UTILITY	20.00
HELPSCREEN	885-3039-37	MSDOS	UTILITY	20.00
HUG BKGRD PRINT SPOOLER	885-1247-37	CPM	UTILITY	20.00
KEYMAC	885-3046-37	MSDOS	UTILITY	20.00
KEYMAP	885-3010-37	MSDOS	UTILITY	20.00
KEYMAP CPM-85	885-1245-37	CPM	UTILITY	20.00
MATHFLASH	885-8030-37	MSDOS	EDUCATION	20.00
ORBITS	885-8041-37	MSDOS	EDUCATION	25.00
POKER PARTY	885-8042-37	MSDOS	ENTERTAINMENT	20.00
SCICALC	885-8028-37	MSDOS	UTILITY	20.00
SKYVIEWS	885-3015-37	MSDOS	ATRONOMY UTILITY	20.00
SMALL-C COMPILER	885-3026-37	MSDOS	LANGUAGE	30.00
SPELL5	885-3035-37	MSDOS	SPELLING CHECKER	20.00
SPREADSHEET CONTEST PKG	885-3018-37	MSDOS	VARIOUS SPRDST	25.00
TREE-ID	885-3036-37	MSDOS	TREE IDENTIFIER	20.00
USEFUL PROGRAMS I	885-3022-37	MSDOS	UTILITIES	30.00
UTILITIES	885-3008-37	MSDOS	UTILITY	20.00
ZPC II	885-3037-37	MSDOS	PC EMULATOR	60.00
ZPC UPGRADE DISK	885-3042-37	MSDOS	UTILITY	20.00
H/Z-100 and PC Compatibles				
ADVENTURE	885-3016	MSDOS	GAME	10.00
BACKGRD PRINT SPOOLER	885-3029	MSDOS	UTILITY	20.00
BOTH SIDES PRINTER UTILITY	885-3048	MSDOS	UTILITY	20.00
CXREF	885-3051	MSDOS	UTILITY	17.00
DEBUG SUPPORT UTILITIES	885-3038	MSDOS	UTILITY	20.00
DPATH	885-8039	MSDOS	UTILITY	20.00
HADES II	885-3040	MSDOS	UTILITY	40.00
HEPCAT	885-3045	MSDOS	UTILITY	35.00
HUG EDITOR	885-3012	MSDOS	TEXT PROCESSOR	20.00
HUG MENU SYSTEM	885-3020	MSDOS	UTILITY	20.00
HUG SOFTWARE CAT UPD #1	885-4501	MSDOS	PROD 1983 - 1985	9.75
HUGMCP	885-3033	MSDOS	COMMUNICATION	40.00
ICT 8080 - 8088 TRANSLATOR	885-3024	MSDOS	UTILITY	20.00
MAGBASE	885-3050	VARIOUS	MAG DATABASE	25.00
MATT	885-8045	MSDOS	MATRIX UTILITY	20.00
MISCELLANEOUS UTILITIES	885-3025	MSDOS	UTILITIES	20.00
PS' PC &Z100 UTILITIES	885-3052	MSDOS	UTILITIES	20.00
REMARK VOL 8 ISSUES 84-95	885-4008	N/A	1987	25.00
REMARK VOL 9 ISSUES 96-107	885-4009	N/A	1988	25.00
REMARK VOL 10 ISSUES 108-119	885-4010	N/A	1989	25.00
REMARK VOL 11 ISSUES 120-131	885-4011	N/A	1990	25.00
SCREEN DUMP	885-3043	MSDOS	UTILITY	30.00
UTILITIES II	885-3014	MSDOS	UTILITY	20.00
Z100 WORDSTAR CONNECTION	885-3047	MSDOS	UTILITY	20.00
PC Compatibles				
CARDCAT	885-6006	MSDOS	CAT SYSTEM	20.00
CHEAPCALC	885-6004	MSDOS	SPREADSHEET	20.00
CLAVIER	885-6016	MSDOS	ENTERTAINMENT	20.00
CP/EMULATOR II & ZEMULATOR	885-6002	MSDOS	CPM & Z100 EMUL	20.00
DUNGEONS & DRAGONS	885-6007	MSDOS	GAME	20.00
EZPLOT II	885-6013	MSDOS	PRINTER PLOT UTIL	25.00
GRADE	885-8037	MSDOS	GRADE BOOK	20.00
HAM HELP	885-6010	MSDOS	AMATEUR RADIO	20.00
KEYMAP	885-6001	MSDOS	UTILITY	20.00
LAPTOP UTILITIES	885-6014	MSDOS	UTILITIES	20.00
PS' PC UTILITIES	885-6011	MSDOS	UTILITIES	20.00
POWERING UP	885-4604	N/A	GUIDE TO USING PCs	12.00
SCREEN SAVER PLUS	885-6009	MSDOS	UTILITIES	20.00
SKYVIEWS	885-6005	MSDOS	ASTRONOMY UTIL	20.00
TCSPELL	885-8044	MSDOS	SPELLING CHECKER	20.00
ULTRA RTTY	885-6012	MSDOS	AMATEUR RADIO	20.00
YAUD (YET ANOTHER UTIL DSK)	885-6015	MSDOS	UTILITIES	20.00

This Software Price List contains all products available for sale. For a detailed abstract of these products, refer to the Software Catalog, Software Catalog Update #1, or previous issues of REMark.

Now Available!
ZUG software is now available on 2" disks. Just put a "-90" at the end of the part number (i.e., 885-6014-90). Also add $3.00 to the purchase price of the software (i.e., $20.00 + $3.00 = $23.00).

LAPTOP OWNERS . . . don't feel left out! All of ZUG's MS-DOS software is available on 3-1/2" micro-floppies too! When ordering, just add a "-80" to the 7-digit ZUG part number. For the standard 5-1/4" floppy, just add a "-37".

Make the no-hassle connection with your modem today! HUGMCP doesn't give you long menus to sift through like some modem packages do. With HUGMCP, YOU'RE always in control, not the software. Order HUG P/N 885-3033-37 today, and see if it isn't the easiest-to-use modem software available. They say it's so easy to use, they didn't even need to look at the manual. "It's the only modem software that I use, and I'm in charge of the HUG bulletin board!" says Jim Buszkiewicz. HUGMCP runs on ANY Heath/Zenith computer that's capable of running MS-DOS!

ORDERING INFORMATION

For VISA, MasterCard, and American Express phone orders, telephone the Zenith Users' Group directly at (616) 982-3463. Have the part number(s), description(s), and quantity ready for quick processing. By mail, send your order, plus 10% postage and handling ($1.00 minimum charge, up to a maximum of $5.00) to: Zenith Users' Group, P.O. Box 217, Benton Harbor, MI 49023-0217. VISA, MasterCard and American Express require minimum $10.00 order. No C.O.D.s accepted.

Questions regarding your subscription? Call Lisa Cobb at (616) 982-3463.

ZENITH
data systems

Groupe Bull

REMark Magazine Subscription & ZLink/COM1 Bulletin Board Information

Your subscription entitles you to receive REMark, our monthly magazine containing articles specific to Zenith Data Systems computer and generally to other PC Compatible computers. All articles in REMark are submitted by readers like you. We welcome YOUR articles, and will pay you for any we accept!

A REMark subscription also allows you full access to the ZLink-COM1 bulletin board system (COM1, for short) described in detail in the brochure. There are many, many megabytes of free and shareware software available for downloading to registered COM1 users. Full access also lets you order products from the "Bargain Centre" section of COM1. The money you can save in the Keyboard Shopping Club will pay for decades of REMark subscriptions.

Last, but definitely not least, your subscription puts you in touch with thousands of other Zenith Data System computer users, from whom invaluable information can be exchanged.

REMark subscriptions, currently $22.95, can be obtained in one of three ways. First, by ordering one on the COM1 bulletin board (see the Keyboard Shopping Club section); second, by phone with VISA, MasterCard, or American Express; and third, through the US Mail using a credit card, money order or check made payable to: Zenith Data Systems. Our address is:

Zenith Data Systems Users' Group
P.O. Box 217
Benton Harbor, MI 49023-0217
(616) 982-3463

Once you receive your ID number, registration on the COM1 BBS is NOT automatic. It requires that you log on, enter your first name and last name EXACTLY as they appear on your REMark mailing label, and then enter your ID number as your password. The FIRST time you access the board, you must elect to start a NEW ACCOUNT and answer the various questions. Once you've done this, our automated scanner will compare the system's database against the subscription database. If you made no mistakes, you will be verified and given full access, within 24 hours.

Once you've been authorized as a full member, several important things happen. First, you're given full downloading privileges of up to one megabyte per day. Secondly, you'll have full access to the message boards. And finally, you'll be able to take full advantage of the Bargain Centre product savings.

Detach this form, enclose your check, money order or credit card information (no cash please).

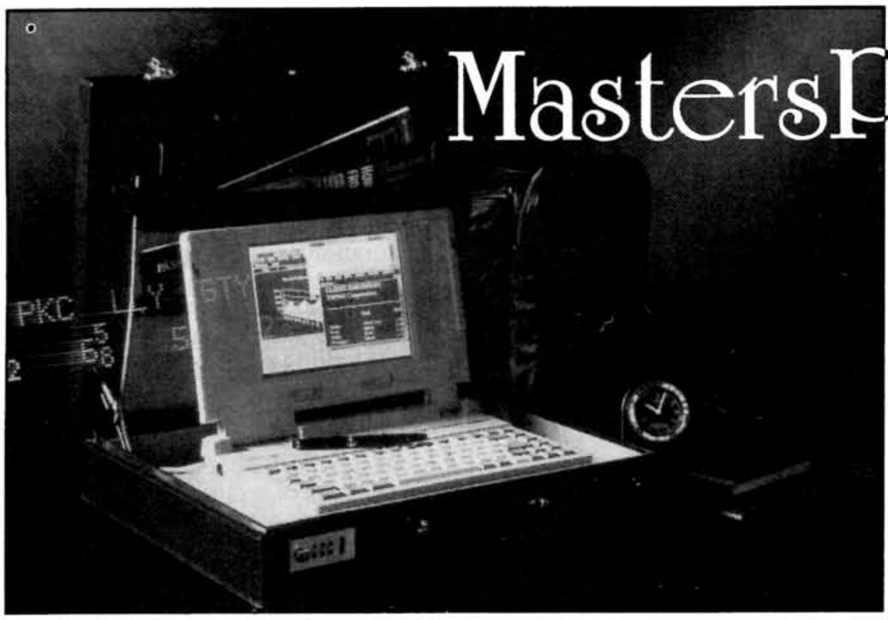
REMark Subscription / Renewal Form

New Member: ☐ Yes ☐ No   Credit Card # ______________________

ID Number: ______________________   Exp. Date ______________________

Address Change? ☐

			Renew	New
Name: ______________________	U.S. Bulk Mail		☐ 19.95	☐ 22.95
Address: ______________________	U.S. First Class		☐ 32.95	☐ 37.95
City, State, Zip: ______________________	APO/FPO Surface Overseas		☐ 32.95	☐ 37.95
Daytime Phone #: ( ) ______________________	Air Printed Overseas		☐ 52.95	☐ 57.95



MastersPort 386SL

Evan R. Hicks
Zenith Data Systems
Technical Writer

The MastersPort 386SL is Zenith Data System's latest entry in the notebook computer field. Introduced at Comdex as one of seven new computers, it is the first computer to use the 80386SL microprocessor. BYTE magazine named the MastersPort 386SL as the best portable of COMDEX '91.

The MastersPort 386SL comes standard with 2MB of system memory, 64KB cache memory, 60MB hard disk drive, and a 1.4KB 3.5-inch floppy disk drive. The LCD panel supports resolution up to 640x480 and has 256KB of video display memory. The normal operating speed is 20MHz. Slow operating speed reduces the CPU clock to 5MHz.

This article will describe the features that are new for ZDS computers. The new features that have been built into the MastersPort 386SL are:

* Rest/resume
* Standby
* Flash ROM upgrade
* Disaster recovery
* Pop-up Setup program
* Multiple configurations
* Dual passwords

Rest/Resume

Rest mode provides about three weeks of storage without turning the computer off. Most computer circuits are shut off, including the CPU clock. About the only circuits still working are the memory and memory controller. This feature is useful if you use large programs that take a long time to load. The computer can resume from rest mode quicker than it can load a large program.

You can enter rest mode one of two different ways: with the built-in timer or by pressing the power button. Before you can enter rest mode with the power button, the button must be configured for rest/resume operation in the Setup program. Set the timer in the "Power Control" menu of the Setup program to automatically enter rest up to 180 minutes after the last detected system activity (keyboard, mouse, modem, etc.).

You can configure the power button to behave one of two different ways: turn the computer on/off, or enter rest mode. If the power button is set for rest/resume operation, all system information is stored in a special memory location called System Management RAM (SMRAM). Press the power button while in rest mode, the computer reads the information in SMRAM and restarts the computer right where you left off. If the computer enters rest mode while an application program was running, the program is resumed in the same place it was with no loss of data. The whole process of entering rest mode only takes a few seconds to complete.

Standby

Standby mode does not shut the computer down as far as rest mode does. The benefit of the standby mode is you can enter standby almost instantly.

You can enter standby automatically with a timer in the Setup program just like rest/resume, or you can use the keyboard hot-key (FN-F6). The computer can maintain standby for about twelve hours.

Standby mode is for the user who needs to stop computing very quickly without any data loss, but only needs to stop for a short period of time. An example of this is you are using your computer while you are waiting in an airport terminal for your flight to be called. When the boarding call is made for your flight, you simply press the FN-F5 key combination, close your computer and board the plane. Once seated, you just open the lid, press any key and you are right back where you were before the interruption.

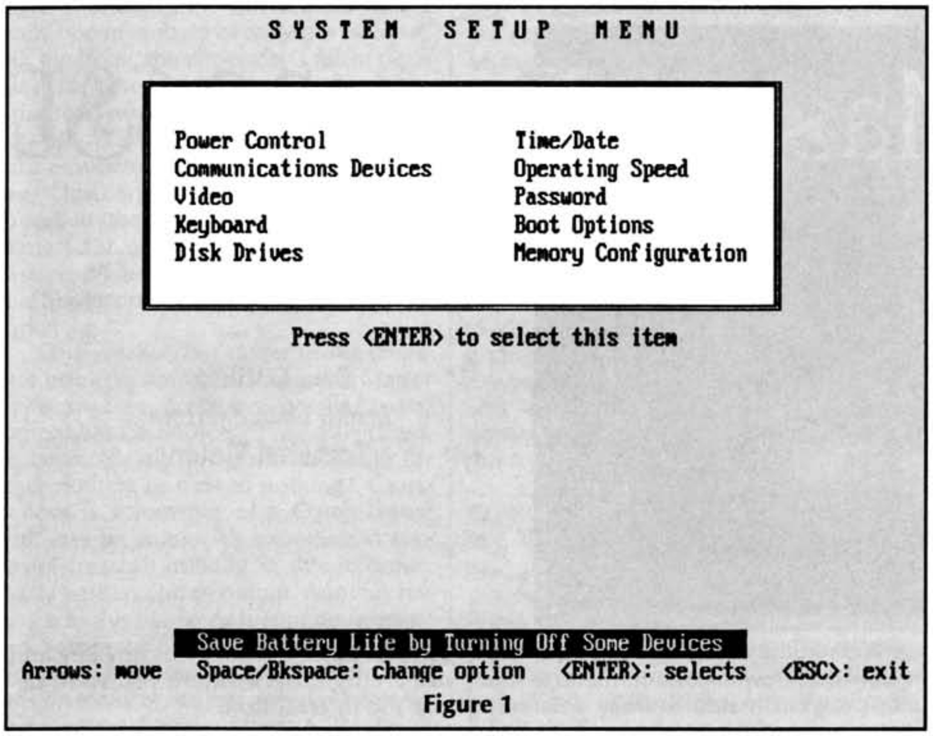
Flash ROM Upgrade

Periodically, new system BIOS code is released for existing computers. Sometimes the new code fixes a 'bug' and sometimes it adds support for a new product. On previous computers, getting the new code into your computer required some disassembly and replacing the EPROM(s) that contained the BIOS code. This computer allows new BIOS code to be copied into the existing EPROM with no disassembly.

The 386SL contains two 128K EPROMs. The first is a standard EPROM that has been around for a long time. This EPROM is called the bootstrap ROM. It contains enough code to partially initialize the computer so that it can execute the code in the second EPROM. The bootstrap ROM also contains the disaster recovery code.

The second EPROM, called the BIOS ROM, contains all the system BIOS, Setup, PMI, and video code needed to operate the computer. This EPROM is a flash ROM device. A flash ROM adds in-circuit erasability to reprogrammability. The MastersPort 386SL contains a 12-volt voltage regulator and the utility program to completely reprogram the flash ROM without removing it from the computer.

The utility, called NEWBIOS, is part of the Monitor Program and appears as the last item in the Command Summary. Before you execute NEWBIOS, you need to buy an upgrade disk containing the new BIOS code for the computer. NEWBIOS assumes that the new code is named ROMIMAGE.BIN and is on the disk that is in drive A.



```
                    S Y S T E M   S E T U P   M E N U

         Power Control                    Time/Date
         Communications Devices           Operating Speed
         Video                            Password
         Keyboard                         Boot Options
         Disk Drives                      Memory Configuration

                   Press <ENTER> to select this item




          Save Battery Life by Turning Off Some Devices
Arrows: move   Space/Bkspace: change option   <ENTER>: selects   <ESC>: exit
```

Figure 1

Don't worry, the NEWBIOS utility will not erase your existing code until it has checked the new code and determined that it is valid code for your computer.

Disaster Recover

During power-up, the computer checks the flash ROM code to determine if it has been corrupted. If not, then the computer powers up normally and slushes the flash ROM code to sections of the upper memory block. If the flash ROM is corrupt, the computer automatically executes a copy of the NEWBIOS utility in the bootstrap ROM.

Pop-up Setup

The MastersPort 386SL has a pop-up setup program that can be invoked with a hot-key (FN-F5) while you are running an application program, from the DOS prompt, or by typing SETUP at the Monitor prompt. When you exit the Setup program, your application continues execution at the same place.

Note: Be careful what setting you change if you run the setup program from within an application program. If you make changes to any of the following areas, the computer will automatically reboot:

* Internal floppy disk drive type
* External floppy disk drive type
* Memory configuration
* Serial port configuration
* Modem port configuration
* Parallel port configuration

Enter the Setup program and you are greeted by the menu, in Figure 1, of ten subsystems that can be user configured.

Select any of the main menu choices and you are presented with another menu listing all the features that can be changed along with the default settings.

Multiple Configurations

This computer gives you the option of having up to three different configurations saved in the system and the ability to load and use any of them whenever you want. Press <ESC> at the main setup menu to exit the Setup program. If you made any changes, you are asked if you want to discard them or save them to user's configuration or to user's and supervisor's configuration. In order to save to supervisor's configuration, you need to know the supervisor's password (more on that later).

You cannot save a setting to the supervisor's configuration without also saving it to user's configuration. You can, however, save a user's configuration by itself. One other option you have is to save the setting to the current session. Any settings saved here will disappear when the computer is turned off.

The current session settings are stored in DRAM memory. The user's configuration settings are stored in the RTC RAM, commonly called CMOS RAM, and the supervisor's configuration settings are stored in the password EEPROM along with two passwords.

Dual Passwords

Two passwords are contained in the MastersPort 386SL, each giving a different level of security. Assuming that both passwords are set, the user or supervisor password must be entered before you can boot DOS or enter the Monitor program. The supervisor also allows you to save a configuration setting to the supervisor's configuration space.

The idea here is that the computer owner, private owner or a library, can set the system configuration the way he wants it, save it in the supervisor's configuration space and set the supervisor's password. The computer can then be loaned to others who can change the configuration to suit their needs. The user can also protect his data by setting the user password. When the computer is returned, the owner can restore the previous settings by loading the supervisor's configuration from the Exit menu of the Setup program.

In Conclusion

I have attempted to give you a fairly thorough look at the innovative ideas that are being incorporated into the new breed of computers from Zenith Data Systems. The MastersPort 386SL is just the beginning...

LetterPerfect

Less Filling, Writes Great

Mark Haverstock
6845 Colleen Drive
Youngstown, OH

First, let me tell you that I've been a WordPerfect user for about four years. It would be almost impossible to convince me to use another word processing program on a regular basis. In fact, I've twisted a few arms among my PC user friends and won them over to WordPerfect. So when I bought my first laptop computer, there was no question about which program to use.

But as I pulled out the pile of master disks, one thing became painfully obvious. Stuffing WordPerfect 5.1 onto a computer with a single 720K disk just wasn't going to work. It was like telling my 11 year-old son to cram his feet into those Nikes we bought him a mere three months ago, instead of making that inevitable trip back to the shoe store for the next size. The program had just grown too big to fit on a single floppy system.

In desperation, I dug out an old copy of WordPerfect 4.2 and found that it fit nicely on a 3.5" 720K floppy, with about 100K or so to spare for file storage. Fortunately there was also room left for the dictionary. But working with both the old and new versions required some mental "gear shifting" due to the differences between the two versions. Text files also had to be converted to WordPerfect 5.1 format or vice-versa when transferring them between computers. This amounted to a lot of extra time and inconvenience.

Soon afterward, LetterPerfect 1.0 appeared on the market. It was advertised as a word processor for new or occasional PC users. LetterPerfect also claimed to deal with the problem of computers with limited storage capacity — especially laptop computers. You could have a spell checker and a respectable amount of features using a laptop with a single 720K floppy drive. LetterPerfect promised all this, and file compatibility with WordPerfect 5.1. It looked almost too good to be true.

There's no question that the program works, and it works well — with complete file compatibility. I wrote this article on my laptop computer using LetterPerfect 1.0 and a single floppy drive. Final editing was done in WordPerfect 5.1 on a PC AT compatible desktop computer.

Features

The big temptation is to compare the features in LetterPerfect to those in WordPerfect 5.1. It's hard not to draw some parallels between them, but LetterPerfect isn't exactly a scaled down version of WordPerfect 5.1. While sharing many features of its more powerful sibling, it stands on its own as a somewhat different approach to word processing — and it's simpler to operate.

LetterPerfect has taken the best of the "skeleton" of WordPerfect 5.1's functions. It has all the essentials, including a full range of page formatting features. Other important features are listed in Table 1.

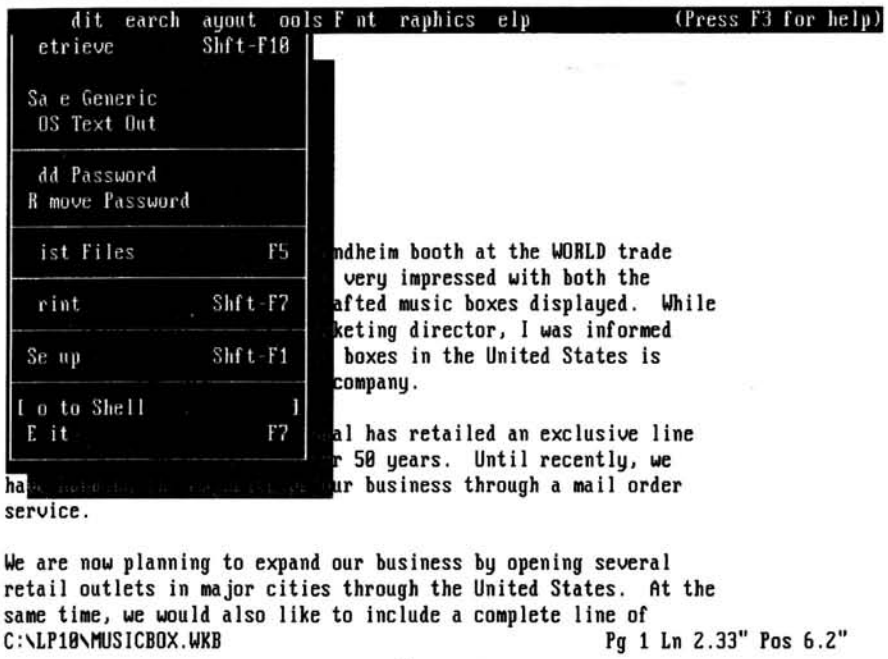
Features in WordPerfect 5.1 not included in LetterPerfect are Tables, Columns, Sort, Styles, Footnotes, Equations, Spreadsheet Import, and the ability to create a table of contents. There are some other compromises. For example, the spelling dictionary is a bit smaller than its WordPerfect counterpart.

LetterPerfect's look and feel are a bit different from WordPerfect 5.1. WordPerfect users will find most of the familiar function key combinations are still active. But LetterPerfect emphasizes a pull-down menu interface to access its features (Figure 1). This is a departure from WordPerfect's "clean screen" philosophy. The menus are also larger, less cluttered, and now can be accessed with a mouse.

Editing documents is as easy, if not easier, in this package. Gone are the menu screens that often go two or three levels deep. You can work with documents cre-

Block	Line Spacing
Bold	Locked Documents
Center	Margins
Context-sensitive	Merge
Help	Mouse Support
Date	Outline
Endnote	Page Numbering
Fast Keys	Pull-down Menus
Fonts	Search
Forms Selection	Speller
Graphics	Subscript/Superscript
Headers/Footers	Tabs
Italics	Thesaurus
Justification	Timed Backup
Labels	Underline
Line Draw	View Document

Table 1
LetterPerfect Main Features



```
    dit  earch  ayout  ools F nt  raphics  elp          (Press F3 for help)
 etrieve          Shft-F10
 Sa e Generic
  OS Text Out
  dd Password
 R move Password
  ist Files             F5   ndheim booth at the WORLD trade
                              very impressed with both the
  rint            Shft-F7    afted music boxes displayed.  While
                             keting director, I was informed
 Se up            Shft-F1     boxes in the United States is
                             company.
[ o to Shell             ]
 E it                   F7   al has retailed an exclusive line
                             r 50 years.  Until recently, we
ha                           ur business through a mail order
service.

We are now planning to expand our business by opening several
retail outlets in major cities through the United States.  At the
same time, we would also like to include a complete line of
C:\LP10\MUSICBOX.WKB                             Pg 1 Ln 2.33" Pos 6.2"
```

Figure 1
Pull-Down Menus

ated in 5.1 without any loss of formatting codes — even those that WordPerfect supports and LetterPerfect doesn't.

No Macros, But Better Help

The Macro feature in 5.1 and earlier versions of WordPerfect allow you to record a series of keystrokes and play them back, like a phone with programmable numbers. WordPerfect power users are likely to have a large collection of keystroke-saving macros, which they won't be able to utilize in LetterPerfect.

But LetterPerfect offers an interesting twist. The available "fast keys" that make finding and remembering a particular function easy. For example, CTRL-B produces bold type, CTRL-R reveals codes and CTRL-S starts a spell check. After a few hours with LetterPerfect, you'll be tempted to throw out the function key template.

Help screens in this program really help. They not only give the usual explanations, but also will perform the function if you wish. For example, if you chose the topic "Minutes Between Backup" in the help menu, you could have LetterPerfect automatically jump to the setup menu. It would place the cursor in position for you to enter the value for Minutes Between Backup.

LetterPerfect also contains graphic features that let you incorporate images from many sources into your documents. Text and graphics can be mixed, making it easy to produce flyers, instructional materials or other documents that need figures, diagrams or pictures.

Installation

Installation is easy. First, put the install disk in drive A, and type at the command prompt: A:INSTALL. The install program will prompt you to choose the type of monitor you're using and whether to install the program on floppy or hard disk. The files are in compressed format, so it takes some time to uncrunch and copy them to the appropriate drive. LetterPerfect won't run directly from the floppies supplied.

When asked, choose the two disk version of LetterPerfect, since it contains more of the program's features. Next, you'll be asked which files to copy. If you're using a portable computer with one or two floppy drives and no hard drive, copy only the program files and the spelling dictionary to conserve space. There will be several disk swaps before the floppy version is ready to run. As you might expect, hard disk installation will go a bit faster.

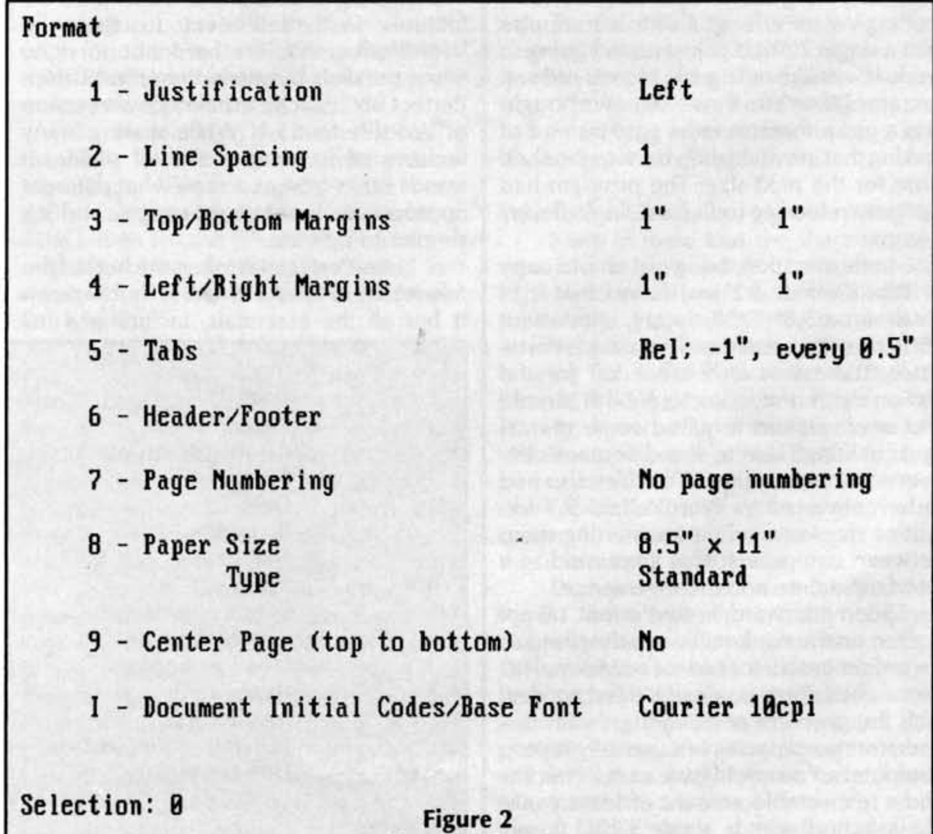
Owners of computers with hard drives will appreciate an additional built-in program, Wordperfect Shell 3.0. Shell acts as a menu, program integrator and memory manager. While running a program from Shell, you can switch to another program without exiting the current one. You can keep as many programs resident as your memory and hard disk space will allow.

Shell also includes a clipboard. The clipboard is a temporary buffer you can use to transfer information between programs running under shell. Shell takes approximately 48K of RAM to operate.

Using LetterPerfect

When you load LetterPerfect on a laptop computer with only floppy drives there are some compromises. The one disk version has the basics needed for word processing, but omits the spelling dictionary.

My personal choice for a floppy system is the two disk version. First, it loads the main LetterPerfect files into memory from disk 1. You're then prompted to load disk 2, which includes the spelling dictionary.



```
Format

     1 - Justification                          Left

     2 - Line Spacing                           1

     3 - Top/Bottom Margins                     1"      1"

     4 - Left/Right Margins                     1"      1"

     5 - Tabs                                   Rel: -1", every 0.5"

     6 - Header/Footer

     7 - Page Numbering                         No page numbering

     8 - Paper Size                             8.5" x 11"
              Type                              Standard

     9 - Center Page (top to bottom)            No

     I - Document Initial Codes/Base Font       Courier 10cpi

Selection: 0
```

Figure 2
Format Menu

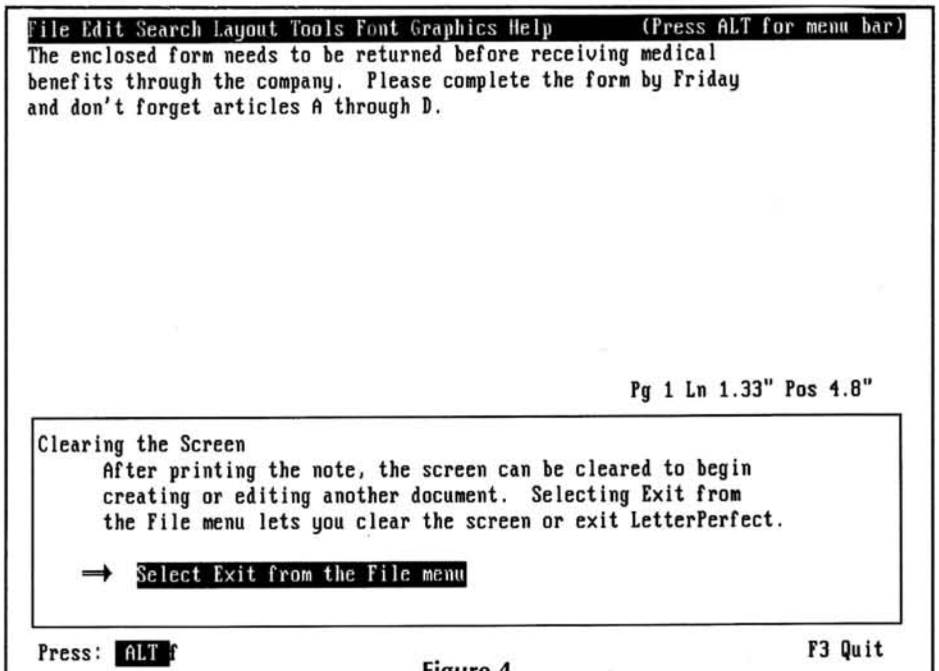

```
File Edit Search Layout Tools Font Graphics Help          (Press ALT for menu bar)
The enclosed form needs to be returned before receiving medical
benefits through the company.  Please complete the form by Friday
and don't forget articles A through D.

                                              Pg 1 Ln 1.33" Pos 4.8"

Clearing the Screen
     After printing the note, the screen can be cleared to begin
     creating or editing another document.  Selecting Exit from
     the File menu lets you clear the screen or exit LetterPerfect.

   ➡  Select Exit from the File menu

Press: ALT f                                                   F3 Quit
```

Figure 4
On-Line Tutorial

You have almost 200K of free disk space left for temporarily storing text files. The thesaurus and help files can be put on a third disk and swapped if needed. Of course, hard disk users won't have to worry about this.

After the program is loaded, you'll see a menu bar across the top, and a status line in the lower right corner. The status line indicates the position of the cursor and the page number.

To activate the menu bar and use the pull-down menus, you press the ALT key. The left and right arrow keys move to the menu bar headings. The down arrow or enter keys open the pull down menus so you can make a selection. To activate a command, press the first or highlighted letter, or use the quick key command listed on the right.

Initial formats, such as line spacing, margins and tabs have already been set. Of course, you can change these at any time and as many times as you like throughout the document (Figure 2).

As you type and make changes in the document's format, LetterPerfect "hides" codes in the text. These codes tell LetterPerfect how the document should look on the screen. They also tell the printer how to print the final copy. These codes can be searched for, deleted and replaced just as text can. You'll find codes for bold, underline, spacing, and a number of other page formatting parameters.

When you're ready to print, LetterPerfect has a preview feature — View Document. This shows you how a document will appear before you actually print it (Figure 3). This is an easy way to check the spacing, margins and page format. LetterPerfect supports over 550 different printers, including popular dot-matrix, daisywheel, inkjet and laser printers.

LetterPerfect also supports built-in fonts for printers, font cartridges, and downloadable soft fonts. To use soft fonts, you may need to order a printer driver from WordPerfect's Orders department. If you already have printer drivers from WordPerfect 5.1, you also can use them with LetterPerfect.

Documentation and Support

The clearly-written 440-page manual will teach you the most important aspects of the program. In addition, there is an excellent on-line tutorial (Figure 4). The tutorial consists of five lessons, starting with a simple letter. By the time you're done, you'll create a short report, complete with endnotes and page numbers.

LetterPerfect shares WordPerfect's toll-free phone support facilities during normal business hours, Monday through Friday. An after-hours support number (not toll free) is also available between 6 p.m. and 7 a.m. Either way, you have access to one of the best support organizations in the computer industry.

System Requirements

LetterPerfect 1.0 runs on IBM PC, XT, AT, PS/2, Zenith and other true compatibles. DOS 2.0 or higher, 330K of free memory and a minimum of one 720K or two 360K disk drives are required; a hard drive is optional. A graphics adapter and CGA, EGA or Hercules compatible monitor are also needed to display the graphics features. If you have a VGA monitor, LetterPerfect displays graphics in the EGA mode.

Conclusions

LetterPerfect is well suited for battery-powered laptop PCs. Since most of its routines are stored in memory, there is little need to access the disk drive for additional files. The less time your drive is active, the longer your battery life.

You can tailor LetterPerfect to meet your personal needs and system limitations. I found that the two disk system worked nicely with my own laptop. As you might guess, it was even more convenient to use on my PC AT compatible with a hard drive — no disk swaps at all.

When you need a "WordPerfect Lite"

Continued on Page 36

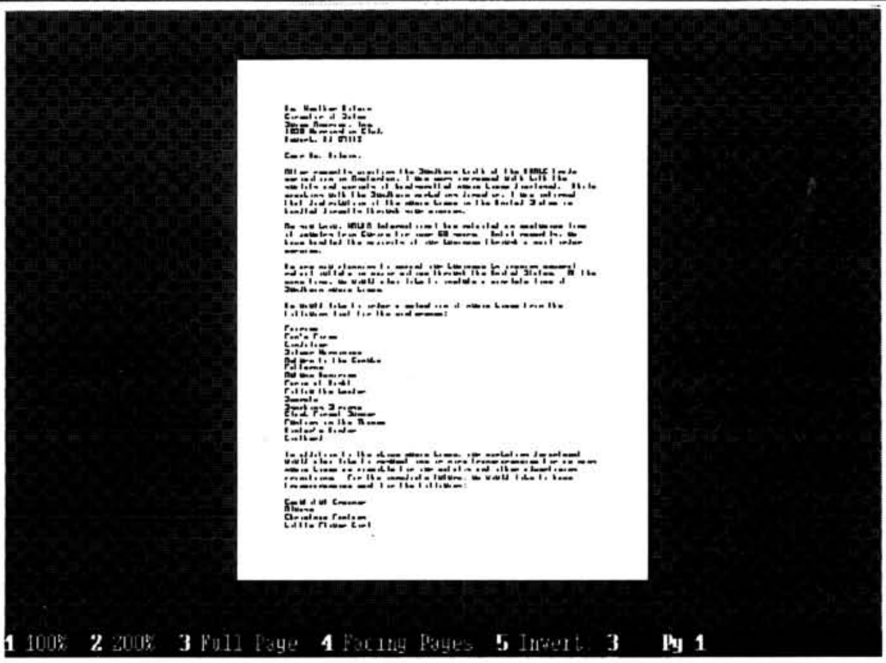
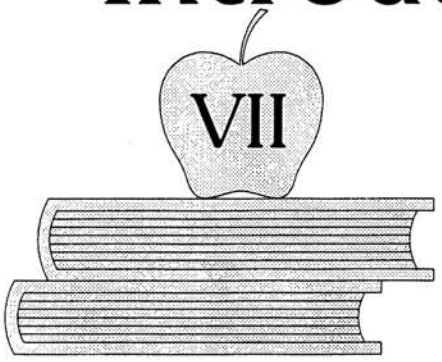


Figure 3
View Document Screen

Introduction To C++



Seventh Installment

Lynwood H. Wilson
2160 James Canyon
Boulder, CO. 80302

The new semester of my C programming class has just started and I am using the Turbo C++ compiler with "C Programming Using Turbo C++" by Robert Lafore as the text. Most of my students seem as excited about learning C++ as I am about teaching it.

I plan to integrate the C++ features into the whole class rather than leaving them for the last as Lafore does. He is presenting C with C++ as a bit of an afterthought. I feel that C++ is going to be the big language of the 1990's and I will be teaching C++ with a few asides on the differences between C and C++, as I have done in this series of articles. In fact, I plan to use these articles as supplementary material for the class.

The pressure of work has eased a bit, and I have had a chance to put in some time on my AI research project, a simulation of the evolution of intelligence in simple organisms. I am doing the programming in C++, of course, and I will be using some of the code in later articles to illustrate the use of C++ objects in simulation.

My goals for this project are to improve my understanding of Object Oriented Programming, particularly the selection and inheritance of objects in a large program, and to learn more about machine learning and the representation of knowledge.

Arrays

As we saw when we were discussing loops, anything worth doing is worth doing repeatedly. However, if you do very much very often you will need lots of variables to do it to. Fortunately C++ provides a convenient way to declare and process groups of variables. These groups are called arrays.

As an example, if you wanted to grade the author of a series of articles you could write a program like this:

```
#include <IOstream.h>
main()
{
  int grade1, grade2, grade3,
     grade4, grade5, grade6,
                     grade7;
   cout << "Enter grade for article
      1.";
   cin >> grade1;
   cout << "Enter grade for article
      2.";
   cin >> grade2;
// and so on and on and on
}
```

As you see it will take two statements to read in each variable. If you want to print it out or do anything else with it each variable will again need to be dealt with separately. Fortunately, there is a better way.

```
#include <iostream.h>
main()
{
  int grades[7];
  int sum = 0;
  for(int i = 0; i 7; i++) {
    cout << "Enter grade for
     article " << i+1 << ". ";
    cin >> grades[i];
  }
  for(i = 0; i 7; i++)
     sum += grades[i];
  cout << "Average is " << sum / 7;
}
```

Look first at the definition of the variables. Sum is a simple integer variable. Grades is an array of 7 integer variables. It's that simple.

The seven integer variables which make up the array are grades[0], grades[1], grades[2], and so forth through grades[6]. The array starts with grades[0] since 0 is the first number. Users may think that 1 is the first number, but we programmers know better. However, that does create a slight potential for confusion. This array of seven elements does not contain an element called grades[7]. The last element in an array is numbered one less than the size of the array.

The elements of such an array are perfectly benign integer variables just like sum, except that their names have numbers in them. You can do anything with them that you can do with any other integer variable. X = grades[4] * 3; is a perfectly legitimate statement, as is grades[0]++;. In addition, you can use variables in the square brackets to select an array element as I did in the program above. This is one of the great advantages of using arrays, because it means you can process all the elements of an array of any size with a simple loop. But the elements of the array are still simple integer variables.

In the program above, in the first iteration of the for loop i is 0, so the variable (array element) grades[0] gets the data. Next time around the element grades[1] gets the input, and so on. Note the code size reduction from doing all this in a loop instead of one variable at a time.

Note that in the prompt to the user the program asks for the grade for number i+1. This shifts the array element numbers to 1 through 7. Another way to do this is to declare an array of eight elements, zero through seven, and then use just one through seven. This wastes two bytes of memory but the data structure more closely matches the data to be stored. The program might be a bit more readable.

Note also that the number of elements in the array, 7 in this example, must be a constant expression. It can be a #define or a const as well as a number, but it must be known at compile time. You cannot declare an array and later decide how long it is, no matter how convenient that might be.

The above example uses an array of integers but you can declare arrays of floats or chars or any other datatype, including datatypes you invent yourself for special purposes. All the elements of an array must be the same datatype, however. For example, the definition

```
float scores[10];
```

creates an array of ten floating point numbers called scores. The first word in the definition is the datatype, the second is the name of the array, and the number in the square brackets is the number of elements in the array.

The compiler knows how much memory is needed for each datatype and allocates enough for the array. Normally the elements of an array are stored together, in order, in the computer's memory.

There is a danger here. C does not check array bounds. As in many other cases, the compiler assumes that you know what you are doing, and submissively does as it is told. If you include the statement

```
cout << "Array[7] is " < array[7];
```

in the above program, it will print something. There is no way to be sure what it will print, but it will print something. In many systems it will print the current value of sum, since that variable is declared right after the array and so is stored in the next two bytes of memory. The Turbo C++ compiler apparently does not do that. I put that line into the program above and it printed 1040 no matter what grades I entered.

Writing to memory outside the bounds of an array is much more dangerous. Sometimes it will work fine. Other times the data will be different when you read it back. And sometimes it will cause strange things to happen somewhere else in the program. And worse yet, you can get all the above effects including random crashing in the same program, depending on the input to the program. This can be a difficult bug to find, and it is best dealt with by an ounce of prevention.

You must ensure that your programs neither read nor write outside array boundaries. The compiler won't help you.

So, what if you don't know in advance how much data your array will need to handle? You must declare an array big enough for the most data you could possibly get and use only part of it if you don't get that much. Here is an example.

```
#include <iostream.h>
const int ARRAY_SIZE = 10;
main()
{
 int grades[ARRAY_SIZE];
 int sum = 0, i = 0;
 do {
  cout << "Enter grade for article
   " << i+1 << ". ";
  cin >> grades[i];
 } while(grades[i] 0 && ++i
   < ARRAY_SIZE);
 for(int j = 0; j i; j++)
   sum += grades[j];
 cout << "Average is " << sum / i;
}
```

I used a do loop since it had to run once in any case. The size of the array is now a constant. The for loop gets the input and ends when the array is full or the user enters a negative number. Upon exiting from the loop, i holds one more than the number of the last valid entry.

Note that we do not want to include the negative number entered to end the data entry in our average. In the test for the while loop, if the input is less than zero the test fails (because the and cannot be true if one of its elements is false) so the second test is not evaluated and thus i is not incremented. However if the last entry is positive the second part of the and is evaluated and i is incremented. Then if the loop ends because i is equal to ARRAY_SIZE, i is still one more than the number of the last valid entry.

In the summing loop, all array elements from number one to number i-1 are added in, and then divided by i. Note that i is the number of valid elements and it points to the first invalid element.

The best way to see this clearly is to work it through with some test data, pretending that you are the computer. This is also one of the best ways to find logic bugs in your programs. One of the more common problems is being one off at the end of a loop, and it is hard to see unless you plug in some values and step it through. I do it on paper, since my mind is all too ready to see what it expects and if I try to do it in my head it always comes out right. Modern debuggers are a great help here, letting you run your code on test data and watch it work.

Initializing an Array

Arrays may be initialized when they are defined, just as other variables can. The syntax is a little different, as you can see below.

```
int a[5] = { 5, 4, 3, 2, 1 };
```

This does just what you would expect. It creates an array of 5 integers and fills it with the numbers given in the order given.

Some of the books say that an array cannot be initialized unless it is global or static but that is not true of the compilers I use. And that brings up a good point.

Your compiler is the final arbiter. Not the language books, not the compiler books, not the language standard (if you are so fortunate as to work in a language which has one) and certainly not me. If you run across something which doesn't seem to make sense write a tiny program and test it. Try it with several compilers if you have access to them, and figure out how it works in the setup you are using. That is what counts.

There are some other tricks to initializing arrays. For example, you can leave out the size of the array if you initialize it.

```
int a[] = { 5, 4, 3 };
```

This will create an array of three elements with 5, 4, and 3 in them.

```
int a[5] = { 5, 4, 3 };
```

This will create an array of 5 integers with 5, 4, 3, 2, and 1 in them.

```
int a[3] = { 5, 4, 3, 2, 1 };
```

This will create an error message because there is no place to put the extra numbers.

Remember, if you don't initialize an array it contains whatever junk happened to be in the memory. The books claim that static and global arrays will be automatically initialized to all zeros but I have been burned in the past and I don't count on it. Modern compilers are much better than those available a few years ago. Perhaps it is time for me to be a little less paranoid.

Passing an Array to a Function

An array is passed to a function as a single parameter. Here is an example.

```
#include <iostream.h>
const int ARRAY_SIZE = 10;
int max(int arr[], int size);
main()
{
 int ar[ARRAY_SIZE];
 for(int i = 0; i ARRAY_SIZE; i++)
 cin >> ar[i];
 cout << "The largest is " <<
max(ar, ARRAY_SIZE);
}

int max(int arr[], int size)
{
 int max_num;
 max_num = arr[0];
 for(int i = 1; i size; i++)
 max_num = arr[i] max_num? arr[i]:
max_num;
 return(max_num);
}
```

In the main function we declare an array of integers and fill it from the keyboard. Note that this routine will not handle bad input properly, but we are just playing. Neither you nor I would use an input routine like this in a real program.

The call to the function max() is embedded in the output statement. We could have declared a variable and filled it with the returned value from max() and then printed the value of the variable, but it would require another variable and an extra line of code.

When we call max() from main() the array is passed by merely writing its name. Max() knows it is getting an array of integers rather than some other kind of variable named ar by the type declaration in the definition of max(). This is consistent with the way we pass other parameters.

Note that we passed the length of the array to the function. As I said above the language furnishes no indication of the boundaries of an array, so the function must know the length of the array so as not to exceed it. Of course we could have hard-coded the length of the array into the function which would have saved a parameter, but then the function could only be used with arrays of length 10. We could have written the function so it would use the global constant ARRAY_SIZE, but then the function would only be usable in programs which used that particular name for the length of the array. Passing in the length is the most portable and flexible way to do it.

There is one way in which passing an array to a function is a little different than other parameters. Remember, Dennis Ritchie was working on machines with a lot less memory than ours and he felt that creating a new copy of an array when it was passed to a function might use up too much of that memory, particularly since arrays can be quite large and use a lot of memory. So when an array is passed to a function as a parameter the array itself is passed rather than a copy. Therefore, if you change the array in the function, the main function will have access to the changes. There is only one copy of the array. Here is an example.

```
#include <iostream.h>
const int ARRAY_SIZE = 10;
void times_2(int arr[], int size);
main()
{
 int ar[ARRAY_SIZE];
 for(int i = 0; i < ARRAY_SIZE; i++)
   cin >> ar[i];
 times_2(ar, ARRAY_SIZE);
 for(i = 0; i < ARRAY_SIZE; i++)
   cout << "\n" < ar[i];
}

void times_2(const int arr[], int
size)
{
 for(int i = 0; i < size; i++)
   arr[i] *= 2;
}
```

Once again we define an array in main() and fill it with integers from the keyboard. Then we pass the array to the function times_2(), along with its size. In the function each element of the array is multiplied by 2. The function does not return anything. Then back in main, we print the array to the screen and find that each of its elements is twice what we entered.

Note that the function does not need to know the size of the array for storage purposes, as the memory to store the array is allocated by the function which defines the array. The function needs the size only to avoid running off the end.

There is a way to ensure that an array cannot accidentally be changed in a function. You can declare the function to be a constant in the function definition. Here is an example.

```
#include <iostream.h>
const int ARRAY_SIZE = 10;
void times_2(const int arr[], int
      size);
main()
{
 int ar[ARRAY_SIZE];
 for(int i = 0; i < ARRAY_SIZE; i++)
   cin >> ar[i];
 times_2(ar, ARRAY_SIZE);
}

void times_2(const int arr[], int
size)
{
 for(int i = 0; i < size; i++)
   cout << "\n" << arr[i] * 2;
}
```

Note that the array must be declared a const both in the function definition and in the declaration (prototype) at the head of the file.

Since we cannot change the array, we must print the new values in the function. There is a subtle difference between this definition and most of those we have seen. Normally when we define or declare the parameters to a function we are talking about the data type which the calling code must pass to the function. In this case, as you can see above, we declare that this function takes a first parameter which is a constant array and then pass it an array which is clearly not a constant. In this case the const in the definition refers to the way the parameter is to be treated once it is in the function, not the data type to be passed in.

The const in the definition has the effect that you cannot change the array once it is in the function. This is valuable for two reasons. One is that you cannot modify the array, by accident or on purpose. Not even six months from now, when you need to make a few changes and you don't remember exactly what you had in mind when you (or someone else) wrote the code. The other value is that you can easily see from the prototype that the function does not modify the array. This means that you can more easily understand the program without reading the code of the function.

One of your primary goals should be to write programs which can be understood at a high level without reading each line of code in the lowest function. Appropriate declarations help a great deal.

Multidimensional Arrays

I said earlier that you can declare arrays of any datatype, and that includes arrays of arrays. This is the construct which we call a two dimensional array and it can be quite useful for holding data which is inherently two dimensional, such as the contents of a screen. Here's how it works.

```
#include <iostream.h>
const int MAX_ROW = 3;
const int MAX_COL = 4;
void fill(int arr[][4], int
     rowsize, int colsize);
main()
{
 int ar[MAX_ROW][MAX_COL];
 fill(ar, MAX_ROW, MAX_COL);
 for(int row = 0; row MAX_ROW;
                     row++) {
   cout << '\n';
   for(int col = 0; col MAX_COL;
                     col++) {
     cout << ar[row][col] << ' ';
   }
 }
}
void fill(int arr[][MAX_COL], int
            rowsize, int colsize)
{
 int col;
 for(int row = 0; row rowsize;
                        row++)
  for(col = 0; col colsize; col++) {
   cout << "\nEnter value for row "
    << row << " col " << col << ":
                                 ";
   cin >> arr[row][col];
 }
}
```

Note that when passing a two dimensional array to a function, the size of the second dimension must be available to the function. Two dimensional arrays are stored in one dimensional memory by storing (in the above case) ar[0][0] through ar[0][3] followed by ar[1][0] through ar[1][3] and so on. In order to access a particular element, ar[2][1] for example, the function must know how far to step in memory for each increment in the first index, and this is the size of the second dimension of the array.

Two dimensional arrays can be initialized when they are defined. If you think of it as an array of arrays, it seems consistent. Here's how.

```
#include <iostream.h>
const int MAX_ROW = 3;
const int MAX_COL = 4;
main()
{
 int ar[MAX_ROW][MAX_COL] = { { 1,
                          2, 3, 4 },
                    { 5, 6, 7, 8 },
                  { 9, 0, 1, 2 } };
 for(int row = 0; row MAX__ROW;
                      row++) {
  cout << '\n';
  for(int col = 0; col MAX_COL;
                         col++) {
    cout << ar[row][col] < ' ';
 }
 }
}
```

Arrays of three or more dimensions are possible, but seldom used. The syntax follows just as you would expect from that of two dimensional arrays.

Strings

A string is an array of characters terminated by a NULL. It is a simple but useful idea.

The NULL character, sometimes written '\0', has a numerical value of 0. In this system the identifier NULL is predefined as 0.

Here's how it works.

```
#include <iostream.h>
main()
{
 char str[20];
 cout << "Please enter your name: ";
 cin >> str;
 cout << "You're quite a programmer,
                                  ";
 cout << str << '.';
}
```

The array str[] is an ordinary array of char. The input function fills str from the keyboard and replaces the carriage return with a NULL. This means that the contents of str comprise a string, an array of char terminated by a NULL. This program does not protect against writing past the end of the array, but after all it's just a demo.

Always remember that a string is always one character longer than it looks like it is because of the NULL.

The advantage of using a string instead of a simple array of char is not apparent

Continued on Page 26

Breaking the Churlish Bounds of DOS

Part 4

David W. Lind
RR1, Box 3114
Bar Harbor, ME 04609

This article is the last in a series which discusses means of overcoming the memory limitations of the Microsoft Disk Operating System (MS-DOS) and Personal Computer Disk Operating System (PC-DOS). In it, extended memory is discussed. Extended memory is that portion of system memory whose address is above one megabyte. This type of memory is available for computers using the Intel 80286, 80386, or 80486 microprocessors. IBM AT and the higher numbered IBM PS/2 series computers or compatibles use these microprocessors and may use extended memory. The 80286 microprocessor can address up to 16 megabytes of memory, 15 megabytes of which is extended memory. The 80386 microprocessor can address up to 4 gigabytes (over 4 billion bytes) of memory. All but one megabyte is extended memory. The 80486 microprocessor is essentially a faster 80386 with a built-in coprocessor. For the purposes of this article, the 80386 and 80486 microprocessors will be considered equivalent.

When the Intel Corporation engineers designed the 80286 and 80386 microprocessors, they gave these microprocessors the capability to emulate the operation of the 8086 and 8088 microprocessors. This emulation capability is called the "Real-Address Mode" or, more commonly, the "real mode." The microprocessor is physically prevented from addressing more than one megabyte of memory by the deactivation of the signal to the A20 line connection to the microprocessor. If this line is activated, the microprocessor can address more than one megabyte of memory.

The 80286 and 80386 can also operate in another mode called the "Protected Mode." In this mode, the microprocessors can address the entire range of extended memory and acquire capabilities not available in real mode, e.g., multi-tasking. During protected mode, different programs are "protected" from accessing each other's memory space or otherwise interfering with one another. The 80386 has a third mode of operation called the "Virtual 8086 Mode" in which it can mix execution of real and protected mode programs by rapidly switching between the two modes. The 80286 microprocessor does not have this capability.

When any of these microprocessors are first activated or reset, they begin in the real mode of operation. Protected mode operating systems, e.g., OS/2, must subsequently switch to protected mode which is accomplished by changing the Machine Status Word (MSW) in the 80286 microprocessor or Control Register 0 (CR0) in the 80386. There are a number of other actions which must be accomplished before the MSW or CR0 can be changed to enter protected mode. Entering protected mode also stops access to the real mode interrupt vector table. Essentially, one must establish a new operating system once in protected mode. It is for this reason that one cannot easily leave MS-DOS or PC-DOS to continue execution in protected mode and return to real mode. There are existing programs which accomplish this feat, e.g., new versions of Windows, some applications programs, and programs which were produced with DOS extenders. In this article, some basic techniques of using extended memory through interrupt 15h and the Extended Memory Specification (XMS) are described. The direct entry to protected mode and the use of DOS extenders are beyond the scope of this article.

Addressing Memory Above One Megabyte - High Memory Area (HMA)

Every byte in the computer's conventional and extended memory has a physical address which the microprocessor can use to obtain the information in that byte. Note that expanded memory is not included in this statement because the microprocessor does not address expanded memory. The amount of memory which a microprocessor can address depends upon the memory bus design, specifically, a register which will be called the "memory address register." The memory address register is usually a physical device in the computer which holds the address of the byte in memory currently accessed. For the purposes of this article, the memory address register will be considered an abstract register capable of containing the physical address of any byte in memory. This distinction must be made because the addressing mechanism of some Intel microprocessors is very complex and a physical memory address register is difficult to identify.

The 8088 and 8086 microprocessors have memory address registers that are 20 bits long. These microprocessors can address up to 2^20 = 1,048,576 bytes or one megabyte of memory. The 80286 microprocessor has a memory address register that is 24 bits long which means it can address up to 2^24 = 16,777,216 bytes or 16 megabytes of memory. The 80386 has two memory models, flat and segmented. The flat model is the model normally used by systems programmers and corresponds to the physical capabilities of the microprocessor. The flat model has a 32-bit memory address register which means it can address up to 2^32 = 4,294,967,304 bytes or 4 gigabytes of memory. The segmented model uses a memory address register with 48 bits of which only 46 are used to address up to 7 x 10^13 bytes or 64 terabytes of memory. The segmented model still uses a 4-gigabyte address space, but the applications programmer assumes a segmented 64-terabyte address space. The 64-terabyte space is an abstract (not physical) space. Therefore, all further references to the 80386 address space in this article are based upon the flat model.

Now that the memory capabilities of the microprocessors are established, one should examine how the memory addressing scheme of MS-DOS and PC-DOS affect these capabilities. The disk operating systems use a segment:offset addressing scheme. Both the segment and the offset are 16-bit registers which means they both can contain numbers from 0 to 65,535. Normally these numbers are written as hexadecimal numbers. More about that in a moment. The segment register actually identifies memory paragraph numbers. Recall from previous articles that the operating system allocates memory in 16-byte blocks called paragraphs. The offset register identifies byte numbers starting at the paragraph identified by the segment register. For example, suppose the segment register contains the decimal number 23,552 and the offset register contains the decimal number 256. The paragraph in this example would begin at the 1+(16x23,552) = 376,833th byte or physical address 376,832 (remember memory locations and paragraph numbers begin with the number 0, not 1). The segment:offset physical address would be 376,832+256 = 377,088. The memory address register would contain the number 377,088. Thus, the memory address register number or actual physical address in memory is formed by multiplying the segment address by 16 and adding the offset address. The microprocessor accomplishes this action by placing the segment register information into the equivalent of a physical memory address register, then shifting the bits four places to the left (which is equivalent to multiplying the number by 16) and adding the number in the offset register to the result.

Usually, the numbers in the registers are expressed in hexadecimal form. In the example, 16 = 10h, 23,552 = 5C00h, 376,832 = 5C000h, and 256 = 100h. Note that 10h x 5C00h = 5C000h which is equivalent to shifting the binary equivalent of 5C00 four bits to the left and filling with zeros. The actual physical address would be 5C000h + 100h = 5C100h. Readers who follow this discussion may wish to check that 5C100h = 377,088.

These details are necessary to an understanding of a useful quirk of the disk operating system. The largest paragraph number that the segment register can contain is 65,535 which places the beginning of the paragraph at 16 x 65,535 = 1,048,560. The larget offset which can be placed in the offset register is 65,535. Therefore, the largest segment:offset address is 1,048,560 + 65,535 = 1,114,095. The disk operating system can address 1,114,096 bytes! But, the 8088 or 8086 microprocessor can only address 1,048,576 bytes, a difference of 65,520 bytes. What happens when one places 65,535 (FFFFh) in the segment register and 65,535 (FFFFh) in the offset register? The memory address register of the 8088 or 8086 "wraps around" and the physical address 65,519 (0FFEFh) appears in the register. But, the memory address registers of the 80286 and 80386 microprocessors are much larger and no wrap around need occur! The physical address 1,114,095 (10FFEFh) could appear in these registers. Therefore, the disk operating system could address 65,520 bytes of extended memory. This block of memory is called the "High Memory Area." There is one catch, the A20 line must be enabled to permit the 80286 and 80386 microprocessors to recognize the address and not emulate the 8088/86 microprocessor. Fortunately, there are ways of enabling the A20 line. More about that later. CAUTION: Some programs take advantage of the wrap around feature of real mode processing. The A20 line should not remain enabled for longer than is necessary.

Addressing Memory Above the High Memory Area

Unfortunately, the disk operating system cannot be forced to address extended memory above the HMA. One must place the microprocessor in the protected mode to achieve this objective. Reference 2, the 80386 Programmer's Reference Manual, explains how to enter protected mode and return to real mode for the 80386 microprocessor - a nontrivial task. A test instruction in the 80286 microprocessor, called "LOADALL" (opcode 0Fh 05h) will permit access to extended memory. But, this instruction is not supported by most assemblers or compilers and must be used with care.

Fortunately, the systems engineers anticipated the need to address extended memory from DOS and provided two ways of accomplishing this objective. The first way is through interrupt 15h. The second means is through a driver program called the Extended Memory Manager (XMS).

Interrupt 15h - The Basic Input/Output System Door to Extended Memory

There are three extended memory functions available through interrupt 15h, one of the Basic Input/Output System (BIOS) interrupts. The first is Function 88h - Get Extended Memory Size. One simply places 88h in register ah and calls interrupt 15h. The function returns the amount of extended memory in kilobytes through register ax. The second function is Function 87h - Move Extended Memory Block. This function moves blocks of extended memory data to or from conventional memory. One places 87h in register ah, the number of words to move in register cx, sets es:si to point to a Global Descriptor Table (GDT), and calls interrupt 15h. The Global Descriptor Table is described in Table 1. The third function is Function 89h - Switch Processor to the Protected Mode, something one does not wish to do unless one is willing to program operating system functions into the applications program. This function is beyond the scope of this article.

A few words about the GDT are necessary. Note that the length of the source and destination segments are in bytes, not words. The number in register cx when the interrupt is called is in words. The segment lengths must be greater than the number of words to be moved. The data to be moved will always begin at the source address and be moved to the beginning of the destination address. These addresses are physical addresses, not segment:offset addresses. The source and destination addresses may begin at any byte. It is necessary to convert any segment:offset addresses to physical addresses. That is accomplished by multiplying the segment address by 16 and adding the offset address. The result will be in two words. Put the least significant word in the least significant two bytes of the physical address. The least significant byte of the other word is placed in the remaining byte. The address is only three bytes or 24 bits to ensure compatibility with both the 80286 and 80386 microprocessor. Of course, Function 87h has a maximum address space of 16 megabytes - the limit of the 80286 microprocessor. Few 80386 microprocessor based computers have more than 16 megabytes of memory, so this limitation is

Offset	Size	Contents
00h	16 bytes	zeros (0)
10h	1 word	Length of source segment in bytes
12h	3 bytes	Source physical address
15h	1 byte	93h (access rights)
16h	1 word	zero (0)
18h	1 word	Length of destination segment in bytes
1Ah	3 bytes	Destination physical address
1Dh	1 byte	93h (access rights)
1Eh	18 bytes	Zeros (0)

Table 1. Global Descriptor Table.

seldom a problem. Note that no restriction is placed upon the location of the source and destination addresses. One could use this function to move blocks of memory in conventional memory. But, keep in mind that the addresses are physical addresses from 0 to 16,777,215 (FFFFFFh) and segment:offset addresses have no meaning above the HMA.

The access rights byte, which is always 93h, and the zeroed areas in the GDT are used by the function to prepare maps required by the protected mode. The movement of the data occurs in the protected mode and the function returns to the real mode. The program READEXT.EXE listed at the end of this article demonstrates the use of INT 15h-Functions 87h and 88h. READEXT.EXE moves blocks of extended memory to conventional memory and displays the printable characters. One can move data to extended memory, but this program does not do so. The reason is a problem with the INT 15h process – it reads and writes absolutely, that is, without regard to how the memory space is used. Thus, one must ensure by some means that the process will not interfere with another usage of the area. In other words, the applications programmer must manage the entire range of memory.

The Extended Memory Specification

The memory management problems with INT 15h extended memory functions encouraged the development of a system similar to the Expanded Memory Specification. This extended memory management system is called the EXtended Memory Specification (XMS). Table 2 contains the descriptions of the XMS functions. Table 3 lists the error codes returned in register bl. Like the EMS system, the XMS system is normally installed through the CONFIG.SYS file when the computer is booted. It also uses an interrupt number, interrupt 2Fh. This interrupt is used by a number of drivers, e.g., print spoolers. So, why is it used? The reason is that the only times one can use the interrupt is to determine if the driver is present and where the entry address is located. At other times, the XMM is accessed by a far jump to the driver. The demonstration program, XMMPROG.EXE, listed at the end of this article shows how this jump is accomplished.

If the XMM is used to manage extended memory, other techniques and programs that are independent of the XMM should not be used. For example, if one installs a virtual disk in extended memory, the XMM should not be used. One may establish the upper limit of the virtual disk and use interrupt 15h to read and write to the remainder of memory. But, if one uses the XMM, the virtual disk area could be overwritten.

The use of interrupt 15h to access extended memory out of the context of the

Function	Description
00h Get XMS Version	Returns the XMS version in register ax, the driver version in register bx, and 0 in register dx if no High Memory Area is allocated or 1 if allocated.
01h Allocate High Memory Area (HMA)	Register dx must contain the number of bytes needed. The function will return 1 in the ax register if allocation was succesful. Normally, the number of bytes needed is 0FFFFh. If the program is a resident program, only the required number of bytes should be requested.
02h Free HMA	If successful, the function returns 1 in register ax.
03h Global Enable A20	If successful, the function returns 1 in register ax. This function permits the HMA to be addressed. The HMA should be allocated before this function is called to ensure the HMA is managed properly.
04h Global Disable A20	If successful, the function returns 1 in register ax. This function returns the microprocessor to real mode addressing emulation.
05h Local A20 Enable	If successful, the function returns 1 in register ax. This function provides access to extended memory for programs which have not allocated the HMA.
06h Local A20 Disable	If successful, the function returns 1 in register ax.
07h Determine A20 Address Line Status	Returns 1 in register ax if the A20 line is enabled or 0 in register ax if the A20 line is disabled.
08h Find Amount of Free Extended Memory	Returns the size in kilobytes of the largest free extended memory block in register ax and the total amount in kilobytes of all free extended memory in register dx.
09h Allocate Extended Memory Block	Register dx must contain the number of kilobytes of extended memory requested. If successful, the function returns a 1 in register ax and the handle number in register dx.
0Ah Free Extended Memory Block	Register dx must contain the handle number of the block to be released. If successful, 1 is returned in register ax.
0Bh Move Extended Memory Block	Registers ds:si must contain the segment: offset address of the following parameter block: **Offset** **Contents** 00h Length of block in bytes (dword) 04h Source block handle 06h Source offset (dword) 0Ah Destination handle 0Eh Destination offset (dword) For conventional memory the handle number is 0 and the offset is the usual segment: offset address format. The offset for extended memory is the number of bytes from the beginning of the extended memory block. If successful, 1 is returned in register ax.
0Ch Lock Extended Memory Block	Register dx must contain the block handle number. The function returns the physical (linear) address of the beginning of the block in registers dx

Table 2. XMS Functions (Cont'd.)

	(high bits) and bx (low bits). This function prevents dynamic relocation of the memory block and provides the address of the beginning of the block. If successful, 1 is returned in register ax.
0Dh Unlock Extended Memory Block	Register dx must contain the handle number of the block to be unlocked. If successful, 1 is returned in register ax.
0Eh Get Handle Information	Register dx must contain the handle number. If successful, the function returns a 1 in register ax, 0 in register bh if unlocked or the lock count if locked, the number of handles still available in register bl, and the block size in register dx.
0Fh Change Extended Memory Block Size	Register bx must contain the new block size in kilobytes and register dx must contain the handle number. If successful, 1 is returned in ax.
10h Allocate Upper Memory Block (640kb to 1mb)	Register dx must contain the requested size in paragraphs. If successful, register ax will contain a 1, register bx will contain the segment base, and register dx will contain the block size in paragraphs.
11h Deallocate Upper Memory Block (640kb to 1mb)	Register dx must contain the segment base of the block. If successful, 1 is returned in ax.

Table 2. XMS Functions.

XMM should also be avoided. The XMM may move blocks of memory it is managing to improve efficiency. Therefore, data which was found at an address once may not be at that address next time one wishes to access it. The XMM process assumes that the user will move blocks of memory using XMS functions. If one wishes to access extended memory directly, that is, not through the XMM; the extended memory blocks must be "locked" by the XMM first. That is, the XMM establishes a physical address for the block which will not change (until the XMM is directed to "unlock" the block) and the XMM returns the physical address of the block through the lock function. Then the user may access the block through interrupt 15h processes or even directly reference the area after the A20 line is enabled.

Normally, blocks of memory are moved in and out of extended memory by XMM functions. Handles are used to identify the blocks of extended memory. A handle is nothing more than a number assigned by the XMM to a block of extended memory at the time the block is allocated. Thereafter, the handle number is used to refer to the block. As with block moves with interrupt 15h, the user is not required to enable the A20 line when using the XMM move block function. The only time the user is required to enable the A20 line is when direct access to extended memory is necessary. Another fact worth mentioning is that the HMA is not allocated as part of an extended memory block by the XMM. For the purposes of the XMS, the HMA is separate from the remainder of extended memory.

The XMS provides two sets of A20 line enable/disable functions. The "global" A20 functions are intended for use by programs which were allocated the HMA. The "local" A20 functions should be used by programs which were not allocated the HMA but wish to directly access the extended memory area. Some memory allocated by the XMM may be directly accessed regardless of the status of the A20 line.

The last two functions in Table 2 refer to "upper memory blocks." These blocks are areas of free memory between 640K and one megabyte. For example, the area reserved for the expanded memory Page Frame Segment may be available for use. Once the XMM has established these areas, they may be used for data or code and accessed by far calls, jumps, or block moves without enabling the A20 line. Upper memory blocks are referenced by segment addresses provided by the XMM not by handles.

Using Extended Memory

Extended memory may be used to store code or data. However, executing code in extended memory is not a trivial process. Code can be transferred from extended memory to a segment in conventional memory and then executed. The DOS access to extended memory is generally limited to transferring blocks of memory. Anything more requires entering protected mode unless the activity is confined to conventional memory and the high memory area.

Regardless of the means one uses to ac-

Error Code	Description
80h	No such function
81h	Virtual disk present
82h	A20 address line problem
8Eh	Driver error
8Fh	Driver error
90h	HMA does not exist
91h	HMA in use
92h	Requested HMA size is less than allowed
93h	HMA not yet allocated
94h	A20 address line is enabled
A0h	All extended memory is in use
A1h	No more handles available
A2h	Inactive handle
A3h	Source handle invalid
A4h	Source offset invalid
A5h	Destination handle invalid
A6h	Destination offset invalid
A7h	Block length invalid
A8h	Overlap in block move
A9h	Parity error
AAh	Block unlocked
ABh	Block locked
ACh	Lock count too large
ADh	Lock attempt failed
B0h	Upper Memory Block request is too large Available memory is in register dx
B1h	Upper Memory Blocks are not available
B2h	Upper Memory Block segment number is not valid

Table 3. XMS Error Codes.

cess extended memory, there are some rules that prudent programmers should follow. First, ensure extended memory exists with interrupt 15h - Function 88h - Get Extended Memory Size or with the XMM process. If the XMM exists and was called, Interrupt 15h - Function 88h may indicate that no extended memory exists. In that case, one can use the XMM to determine the size of extended memory. When using the XMM process, first ensure that the XMM exists.

Second, determine if a program, e.g., a virtual disk or terminate-but-stay-resident, is already using extended memory. If extended memory is already in use and not under control of the XMM, the use of the XMM may not be viable. However, one can always use interrupt 15h block moves. Just ensure that existing data or code is not overwritten.

Finally, if the XMM can be used, allocate the HMA and/or extended memory and move the data as required under the control of the XMM. Be sure to release handles, HMA, extended memory, and upper memory blocks before terminating the program unless one intends to transfer data to another program through extended memory. The A20 line, if enabled by the program, should be disabled by the program before termination. If these rules are followed, the use of extended memory can be rewarding and trouble free.

The Demonstration Program

The demonstration programs listed at the end of this article, are intended to show the application of the concepts discussed in this article. The extensive comments explain the processes in detail and need not be transcribed should the reader wish to assemble and execute the programs. All the major techniques of accessing extended memory from MS-DOS or PC-DOS are demonstrated short of entering protected mode. The readers are encouraged to modify the programs to suit their needs.

The READEXT.EXE program can be a useful tool in that it reads all of extended memory and displays the printable data in 1,024 byte pages. Note that the Global Descriptor Table in this program is set up by using offsets vice labels. The second program uses labels in a similar table. Both methods are sound and were included for instructional purposes.

The XMMPROG.EXE program is not a useful tool as written mainly because extended memory is saved and restored which defeats the purpose of the XMS. These extra block moves are necessary to help prevent the disruption of programs already using extended memory. The reader may wish to delete saves and restores if the reader is certain that extended memory is free. Of course, in a real application, these saves and restores would not be used.

Careful study of both these programs should reward the reader with enough knowledge to use extended memory in many applications.

Summary

In this article two ways of accessing extended memory from MS-DOS and PC-DOS were discussed - interrupt 15h functions and using the Extended Memory Specification. When combined with the techniques discussed in previous articles, the power and utility of real mode disk operating systems can be vastly increased. The reader's knowledge should be vastly expanded and extended opening new avenues for innovative problem solving. The best aspect of this knowledge is the joy of being able to use the full memory capabilities of modern personal computers.

References

1. Duncan, Ray, Extending DOS, 1990, Addison-Wesley Publishing Company, Inc., Reading Massachusetts.
2. Intel Corporation, 80386 Programmer's Reference Manual, 1986, Intel Corporation, Santa Clara, California.
3. Microsoft Corporation, Microsoft Macro Assembler 5.0 Microsoft CodeView and Utilities, 1987, Microsoft Corporation, Redmond, Washington.
4. Microsoft Corporation, Microsoft Macro Assembler 5.0 Programmer's Guide, 1987, Microsoft Corporation, Redmond, Washington.
5. Microsoft Corporation, Microsoft MS-DOS Operating System Programmers Reference, 1984, Microsoft Corporation, Bellevue, Washington.
6. Phoenix Technologies Ltd., System BIOS for IBM PC/XT/AT Computers and Compatibles, 1989, Addison-Weseley Publishing Company, Inc. Reading, Massachusetts.
7. The Waite Group, The Waite Group's MS-DOS Developer's Guide, second edition, 1989, Howard W. Sams & Company, Indianapolis, Indiana.

Trademarks

IBM, AT, PC-DOS, and PS/2 are trademarks of the International Business Machine Corporation.

Intel is a registered trademerk of the Intel Corporation.

Microsoft, MS-DOS, and Windows are registered trademarks of the Microsoft Corporation.

Listing

```
        COMMENT * READEXT.EXE
                  This program determines the amount of extended memory using
                  the Basic Input/Output System (BIOS) Interrupt 15h - Function
                  88h, reads and displays 1,024 byte blocks of extended memory
                  using Interrupt 15h - Function 87h, and terminates when all
                  extended memory is read and displayed.  Only printable ASCII
                  characters are displayed, all other characters are displayed
                  as blanks.  The program may be terminated by CTRL-C.  This
                  program is not intended for use after an Extended Memory
                  Manager (XMM) is invoked.

                  Interrupt 15h - Function 87h will write to extended memory.
                  This program does not do so because there is no analysis
                  performed which determines what part of extended memory is
                  available for use.  To write to extended memory, make
                  conventional memory the source and extended memory the
                  destination in the Global Descriptor Table. Ensure that
                  any portion of extended memory written is not in use by
                  another program.

                  Copyright (C) 1990 by David W. Lind

                  No expressed or implied warranties are made for this program
                  with respect to merchantability or fitness for a particular
                  purpose.  The user assumes all responsibility for any damages
                  resulting from the use of this program. *

data    SEGMENT                     ;Program Data Segment
ermess1 DB      ' ERROR - Extended memory not found.',10,13,'$'
mess0   DB      10,13,' Extended memory starting address: ','$'
mess1   DB      10,10,13,' Depress any key to continue or CTRL-C to '
        DB      'terminate program.',10,13,'$'
mess2   DB      ' Expanded Memory limit reached.  Program terminated.','$'
vmode   DB      ?               ;Video display mode
outstr  DB      10 DUP(1)       ;Display string
hundsq  DW      10000           ;Conversion constant
hund    DW      100             ;Conversion constant
temp    DW      ?               ;Temporary storage
tempa   DW      ?               ;Temporary storage
lows    DW      0               ;Low 16 bits of extended memory address
highs   DW      10h             ;High 8 bits of extended memory address
highlim DW      10h                 ;High 8 bits of memory address limit
sixfour DW      64                  ;Conversion constant
linefeed    DB  10,10,13,'$'        ;Double line feed and carriage return
dest    DB      1028 DUP(0)         ;Destination block
data    ENDS
esdata  SEGMENT
```

```
gdt     DB      48 dup(0)               ;Global Descriptor Table (GDT)
esdata  ENDS

code    SEGMENT                         ;Program Code Segment
        ASSUME  cs:code,ds:data,es:esdata
start:                                  ;Program Entry Point
        mov     ax,data                 ;Move address of data segment to ax
        mov     ds,ax                   ;Initialize ds register
        mov     ax,esdata               ;Move address of extra segment to ax
        mov     es,ax                   ;Initialize es register
        mov     ah,0Fh                  ;Move get video mode function number
                                         to ah
        int     10h                     ;Get current video display mode
        mov     vmode,al                ;Store video display mode

        COMMENT * The following code determines the amount of extended memory
                  in kilobytes.  If extended memory does not respond, an error
                  message is printed.  NOTE:  If an Extended Memory Manager
                  (XMM) was used before this program, the amount of extended
                  memory returned by Function 88h will be zero and this
                  program will terminate.  To use this program under these
                  circumstances, either reset (warm boot) the computer and
                  run this program before another program uses the XMM or
                  eliminate the memory limit code that follows and set the
                  highlim variable to the amount of memory in the computer. *

        mov     ah,88h              ;Put Extended Memory Size function # in ah
        int     15h                 ;Get extended memory size
        jnc     himem               ;If extended memory, go to himem
nohimem:
        lea     dx,ermess1          ;Put message offset in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Display string
        jmp     xit                 ;Terminate program
himem:
        cmp     ax,0                ;Check the amount of extended memory
        je      nohimem             ;If none, go to nohimem

        COMMENT * Hereafter, the addressing of memory blocks will be in the
                  form of 24 bit physical addresses - NOT segment:offset.  The
                  memory limit is the amount of extended memory just determined
                  plus the basic system memory.  This physical address requires
                  24 bits i.e. a byte to hold the high order bits and a word to
                  hold the low order bits.  Normally, one manipulates two words
                  to make the process easier.  The word containing the lower
                  order bits represents up to 64 kilobytes.  Therefore, the
                  word containing the higher order bits is the number of 64
                  kilobyte blocks of memory.  The following code divides the
                  number of 1 kilobyte blocks of extended memory by 64 to
                  obtain the number of 64 kilobyte blocks of extended memory
                  and then adds the result to 10h (i.e. 16 which is the number
                  of 64 kilobyte blocks in conventional memory or one megabyte)
                  yielding the total number of 64 kilobyte blocks of memory.
                  This number in the high order bits word/byte and zeros in the
                  low order word represent the upper limit of memory. *

        xor     dx,dx                   ;Clear the dx register for division
        div     sixfour                 ;Divide number of 1K blocks by 64
        add     highlim,ax              ;Add to 10h to get high word limit

        COMMENT *  If the memory limit code is deleted, do not delete
                   the code below this line. *

        COMMENT * The following loop moves a block of extended memory to
                  the data segment.  The source is extended memory and
                  the destination is the data segment.    The addresses for
                  both the source and destination must be 24 bit physical
                  addresses.  If one wished to move data from conventional
                  memory to extended memory, one simply makes the source
                  conventional memory and the destination extended memory.
                  This program does not write to extended memory because
                  an analysis of the contents of extended memory would be
                  necessary to ensure other programs or data would not be
                  erased.  Such an analysis is difficult without knowledge
                  of the resident programs. *

        COMMENT *  The following code initializes the Global Descriptor Table
                   for the destination data block which is the data segment.
                   The Global Descriptor Table must be in the extra segment
                   at the offset in register si. The segment:offset address
                   of the destination data block (dest) must be converted to
                   a 24-bit address before placing the result in the table.
                   Note that the addition of the segment and offset addresses
                   is accomplished in the dx:ax register pair using the add
                   followed by the add carry (adc) instruction. This process is
                   a good example of how to add large integers.

                   The size of the block to be moved is 1024 bytes.  The number
                of bytes to be transferred should be an even number greater
                    than 0.

                   Be sure to store the access rights number in the GDT.
                   This number is always 93h.     *

        lea     si,gdt          ;Put OFFSET of GDT in si
        mov     ax,ds           ;Put data segment address in ax
        mov     dx,16           ;Place 16 in dx
        mul     dx              ;Convert ds segment to physical address
        lea     cx,dest         ;Put OFFSET of dest in cx
        add     ax,cx           ;Add the OFFSET to segment address
        adc     dx,0            ;Add carry, if any, to upper word
        mov     cx,1024         ;Define number of bytes to transfer

        COMMENT * The segment lengths must be greater than the number of bytes
                  to be transferred. *

        mov     es:[si+18h],1028          ;Put segment length at gdt+18h
        mov     es:[si+1Ah],ax            ;Put lower part of address at gdt+1Ah
        mov     es:[si+1Ch],dl            ;Put higher part of address at gdt+1Ch
        mov     byte ptr es:[si+1Dh],93h  ;Store access rights in gdt

        COMMENT *  Next the source portion of the GDT is completed starting with
                   the source length.  When the interrupt is called, register
                   cx must contain the number of words to be moved.  Therefore,
                   the number of bytes to be moved, which is already in
                   register cx, is converted to the number of words to be
                   moved by dividing cx by 2.  Shifting the cx register one
                   byte to the right is equivalent to division by 2. *

loop00:

        mov     es:[si+10h],1028 ;Define source segment length
        shr     cx,1             ;Convert from # of bytes to # of words
        mov     ax,lows          ;Put the low part of the address in ax
        mov     word ptr es:[si+12h],ax    ;Store lower source address in GDT
        mov     dx,highs         ;Put high source address bits in dx

        COMMENT *  The physical address is now in registers dx:ax.  The
```

```
                procedure "ascstr" converts this address to an ASCII
                string and stores the result in "outstr" for subsequent
                display. *

        call ascstr                 ;Convert address to ASCII string

        COMMENT * Finish the GDT. *

        mov     ax,highs            ;Put the high address bits in ax
        mov     byte ptr es:[si+14h],al   ;Store upper source address
        mov     byte ptr es:[si+15h],93h  ;Store access rights in GDT

        COMMENT * Call the Move Block function. *

        mov     ah,87h              ;Put function number in ah
        int     15h                 ;Call move block subroutine

        COMMENT *  The following code displays the starting extended memory
                address. *

        mov     ah,00h              ;Put Set Video Mode function number in ah
        mov     al,vmode            ;Put video display mode in al
        int     10h                 ;Clear screen and set video mode
        lea     dx,mess0            ;Put message offset in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Call Display String
        lea     dx,outstr           ;Put message offset in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Call Display String
        lea     dx,linefeed         ;Put message offset in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Call Display String

        mov     bx,0                ;Set index variable to 0
loop0:

        COMMENT * The data moved from extended memory into the data segment
                is displayed by the following code.  Nonprintable characters
                are replaced by blanks (20h).  *

        mov     dl,dest[bx]         ;Move ASCII character at offset dest[bx] to dl
        cmp     dl,32               ;Compare value to 32
        ja      skip0               ;If above, go to skip0
        mov     dl,20h              ;Else, replace with a blank character
skip0:
        cmp     dl,127              ;Compare value to 127
        jb      skip1               ;If below (printable), go to skip1
        mov     dl,20h              ;Else, replace with a blank character
skip1:
        mov     ah,02h              ;Put Display Output function number in ah
        int     21h                 ;Display ASCII character found in dl
        inc     bx                  ;Increment the index variable
        cmp     bx,1024             ;Compare the index to 1024
        jb      loop0               ;If below, continue the display

        COMMENT * The user is asked to depress any key to continue.  The input
                CTRL-C will terminate the program. The low bits of the
                extended memory address are incremented by 1024 and control
                is passed to the move block process until the low address
                word over flows.  Then, the high byte of the address is
                incremented, the lower address word zeroed and control is
                passed to the move block function unless the memory limit
                is attained in which case the program terminates. *

        mov     ah,09h              ;Put Display String function number in ah
        lea     dx,mess1            ;Put message offset in dx
        int     21h                 ;Call Display String
        mov     ah,08h              ;Put STDIN Input function number in ah
        int     21h                 ;Call STDIN Input (terminates program for CTRL-C)
        add     lows,1024           ;Else, add 1024 to the 24-bit address low bits word
        jc      nextseg             ;Jump to nextseg if the register rolls over
        jmp     loop00              ;Move next block
nextseg:
        inc     highs               ;Increment the high bits of the source address
        mov     ax,highlim          ;Move the high limit to ax
        cmp     highs,ax            ;Compare the present address with memory limit
        jae     xitend              ;If above or equal to limit, terminate program
        mov     lows,0              ;Else, reset low address bits
        jmp     loop00              ;Move new block

        COMMENT * Program termination code. *

xitend:
        lea     dx,mess2            ;Put offset of message in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Display exit string
xit:
        mov     ah,4Ch              ;Put function number in ah
        int     21h                 ;Terminate program

        COMMENT * The following PROCedure converts an unsigned binary integer
                to ASCII code and stores the result in a ten byte string
                labeled "outstr."  The binary integer must be in the dx:ax
                register pair or in ax alone when the PROC is called.  If
                the register dx is not used, it must be cleared (set to 0).
                The ax, bx, cx, and dx registers are used by the PROC.  The
                following code must be in the data segment:
                                outstr   DB  10 DUP (1)
                                hundsq   DW  10000
                                hund     DW  100
                                temp     DW  ?
                                tempa    DW  ?
                The stored string is terminated by an ASCII "$" to permit
                use of the MS-DOS display string subroutine. Leading zeros
                are replaced by ASCII blanks. *

ascstr  PROC    NEAR                ;Convert to ASCII & store in outstr
        xor     bx,bx               ;Zero bx

        COMMENT * Convert binary to ASCII and store in the stack segment. The
                numbers are pushed onto the stack in reverse order. This code
                uses the "aam" (ASCII Adjust After Multiply) microprocessor
                function to shorten the code and execution time. *

ascst1: nop                         ;Begin loop
        div     hundsq              ;Divide by 10,000
        mov     tempa,ax            ;Put quotient in tempa
        mov     ax,dx               ;Put remainder (<10,000) in ax
        xor     dx,dx               ;Clear dx
        div     hund                ;Divide by 100
        mov     temp,ax             ;Put quotient in "temp"
        mov     ax,dx               ;Put remainder in ax
        aam                         ;Convert from binary to BCD
        add     al,48               ;Convert from BCD to ASCII
        inc     bx                  ;Increment index register
        add     ah,48               ;Convert from BCD to ASCII
        push    ax                  ;Push the two digits onto the stack
        inc     bx                          ;Increment index register
        xor     dx,dx                       ;Clear dx
        mov     ax,temp                     ;Put next two digits in ax
        aam                                 ;Convert from binary to BCD
        add     al,48                       ;Convert from BCD to ASCII
        inc     bx                          ;Increment index register
        add     ah,48                       ;Convert from BCD to ASCII
        push    ax                          ;Push the two digits onto the stack
        inc     bx                          ;Increment index register
        xor     dx,dx                       ;Clear dx
        mov     ax,tempa                    ;Put remaining number (>10,000) in ax
        cmp     ax,0                        ;Does quotient exist?
        jz      ascst0                      ;If not, go to ascst2
        jmp     ascst1                      ;If yes, go to beginning of loop
ascst0: nop                                 ;End of function
        COMMENT * Prepare registers for next function. *
        dec     bx                          ;Decrement index register
        mov     cx,bx                       ;Move bx to cx, save old index
        xor     bx,bx                       ;Clear the index register
ascst2: nop                                 ;Store numbers in outstr
        COMMENT * Pop the stack and store the number in "outstr" in the
                  proper order *
        pop     ax                          ;Put two most significant digits in ax
        mov     outstr [bx],ah              ;Put most significant digit in "outstr"
        inc     bx                          ;Increment index register
        sub     cx,1                        ;decrement the counter
        mov     outstr [bx],al              ;Put next digit in "outstr"
        inc     bx                          ;Increment the index register
        sub     cx,1                        ;decrement the counter
        cmp     cx,0                        ;Compare counter to zero
        jge     ascst2                      ;Go to next numbers if bx is + or 0
        mov     outstr [bx],36              ;Put ASCII "$" in outstr
        COMMENT * Blank leading zeros *
        cmp     outstr,48                   ;Is most signifficant digit 0?
        jne     ascst3                      ;If not, return
        mov     outstr,32                   ;If so, put an ASCII blank in outstr
        mov     bx,1                        ;Set bx to 1
        cmp     outstr [bx],48              ;Is next digit 0?
        jne     ascst3                      ;If not, return
        mov     outstr [bx],32              ;If so, put an ASCII blank in outstr
        inc     bx                          ;Increment bx
        cmp     outstr [bx],48              ;Is next digit 0?
        jne     ascst3                      ;If not, return
        mov     outstr [bx],32              ;Else, put an ASCII blank in outstr
ascst3: nop                                 ;Transfer to return
        ret                                 ;Pop stack and return
ascstr  ENDP                                ;End of PROCedure

code    ENDS

stack   SEGMENT stack                       ;Program strack segment
        DW      256 DUP(?)          ;Define stack space
stack   ENDS

        END     start                       ;Mark end and define start


        COMMENT * XMMPROG.EXE
                  This program demonstrates the use of the High Memory Area
                  (HMA) and the remainder of extended memory using the Extended
                  Memory Specification (XMS) Extended Memory Manager (XMM).
                  The program checks for the presence of the XMM, allocates
                  the HMA, writes data to the HMA, displays the data in the
                  HMA, displays the XMS version numbers, and demonstrates
                  block moves.

                  Copyright (C) 1990 by David W. Lind

                  This program is for pedagogical purposes only.  No expressed
                  or implied warranties are made for this program with respect
                  to merchantability or fitness for a particular purpose.  The
                  user assumes all responsibility for any damage resulting from
                  the use of this program or the concepts.  WARNING:  Failure to
                  save and restore the HMA data could interfere with other
                  programs and destroy data. *

data     SEGMENT                            ;Program Data Segment
lfcr     DB     10,13,'$'
mess0    DB     ' ERROR - Extended Memory Manager is not installed.',10,13,'$'
mess01   DB     ' ERROR - Could not allocate High Memory Area.','$'
mess1    DB     ' The Extended Memory Specification version number is ','$'
mess2    DB     10,13,' The Extended Memory Manager version number is ','$'
mess3    DB     ' The High Memory Area exists.',10,13,'$'
mess4    DB     ' The High Memory Area does not exist.',10,13,'$'
mess5    DB     ' High Memory Area allocated. ',10,13,'$'
mess6    DB     ' High Memory Area deallocated. ',10,13,'$'
hmatest DB      ' HMA test OK. ',10,13,'$';HMA test message
xmal     DB     ' Expanded Memory Block Allocated.',10,13,'$'
xmdeal   DB     ' Expanded Memory Block Deallocated.','$'
xmms     DB     ' Block Move test OK. ',10,13,'$',1024 DUP(?)
xmmout   DB     1048 DUP(?)         ;Output area for block move data
savehma DB      17 DUP(?)           ;HMA save area
savexmm DB      1048 DUP(?)         ;XMM save area
xmm      DD     ?                   ;Far address of Extended Memory Manager
majvers  DB     ?                   ;Major version number of XMS
minvers DB      ?                   ;Minor version number of XMS
majverm DB      ?                   ;Major version number of XMM
minverm DB      ?                   ;Minor version number of XMM
hmaind   DB     ?                   ;HMA existence indicator
handle   DW     ?                   ;XMS handle
         COMMENT * The following data is the XMS parameter block to move data to
                   extended memory. *
eblklen  DD     1024                ;Length of block to be moved in bytes
eshandle DW     ?                   ;Source handle
esoffset DD     0                   ;Offset address of source
edhandle DW     ?                   ;Destination handle
edoffset DD     0                   ;Destination offset
         COMMENT * The following data is the XMS parameter block to move data to
                    conventional memory. *
cblklen  DD     1024                ;Length of block to be moved in bytes
cshandle DW     ?                   ;Source handle
csoffset DD     0                   ;Offset address of source
cdhandle DW     ?                   ;Destination handle
cdoffset DD     0                   ;Destination offset
data     ENDS
esdata   SEGMENT
esdata   ENDS

code     SEGMENT                    ;Program Code Segment
         ASSUME cs:code,ds:data,es:esdata
start:                              ;Program Entry Point
         mov    ax,data             ;Put address of data segment in ax
         mov    ds,ax               ;Initialize data segment
         mov    ax,esdata           ;Put address of extra segment in ax
         mov    es,ax               ;Initialize extra segment

         COMMENT * The following code determines if the Extended Memory Manager
                    is installed.  If not, the program terminates. *
        mov     ax,4300h            ;Put installed status function number in ax
        int     2Fh                 ;Call installed status function
        cmp     al,80h              ;Compare result to 80h
        je      continue1           ;If present continue
        lea     dx,mess0            ;Else, put message offset in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Call Display String
        jmp     xit                 ;Terminate program

continue1:

        COMMENT * The following code finds the entry address of the Extended
                  Memory Manager and stores the address in the data segemnt at
                  xmm. *

        push    es                  ;Save address of extra segment
        mov     ax,4310h            ;Put function number in ax
        int     2fh                 ;Get XMM driver entry point
        mov     ax,es               ;Move XMM segment address to ax
        pop     es                  ;Restore extra segment
        mov     WORD PTR xmm+2,ax     ;Put XMM segment address in xmm
        mov     WORD PTR xmm,bx       ;Put XMM offset address in xmm

        COMMENT *  The following code obtains the XMS  and XMM version
                   number.  It also determines if a HMA exists.  IMPORTANT
                   NOTE:  Although the XMM presence is determined through
                   interrupt 2Fh, it is called by a FAR call to the address
                   just loaded at xmm, not through interrupt 2Hh.  All
                   subsequent calls to the XMM will be through the code:

                                call DWORD PTR xmm

                   DWORD PTR indicates that the call is to a double word
                   address (FAR address) stored at location xmm. *

        mov     ah,00h              ;Put Get XMS Version function number in ah
        call    DWORD PTR xmm       ;Call Get XMS Version
        mov     majvers,ah          ;Move the XMS major version number to majvers
        mov     minvers,al          ;Move the XMS minor version number to minvers
        mov     majverm,bh          ;Move the XMM major version number to majverm
        mov     minverm,bl          ;Move the XMM minor version number to minverm
        mov     hmaind,dl           ;Move the HMA indicator to hmaind
        cmp     dl,1                ;Check HMA indicator
        je      continue2           ;If HMA exists, go to continue2
        lea     dx,mess4            ;Else, put offset of message in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Display string
        jmp     continue3           ;Go to continue3

continue2:

        COMMENT * Now allocate the HMA through XMM Function 01h - Allocate
                  HMA. *

        lea     dx,mess3            ;Else, put offset of message in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Display string
        mov     ah,01h              ;Put Allocate HMA function in ah
        mov     dx,0FFFFh           ;Put size of allocation in dx
        call    DWORD PTR xmm       ;Call Allocate HMA
        cmp     ax,01h              ;Was allocation successful?
        je      continue3           ;If yes, go to continue3
        lea     dx,mess01           ;Else, put message offset in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Call Display String
        jmp     xit                 ;Terminate program

continue3:

        lea     dx,mess5            ;Put message offset in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Display string
        lea     dx,mess1            ;Put offset of message in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Display string
        mov     dl,majvers          ;Put XMS major version number in dl
        add     dl,30h              ;Convert to ASCII
        mov     ah,02h              ;Put Display Character function number in ah
        int     21h                 ;Call Display Character
        mov     dl,'.'              ;Put period character in dl
        mov     ah,02h              ;Put Display Character function number in ah
        int     21h                 ;Call Display Character
        mov     al,minvers          ;Put XMS minor version number in al
        aam                         ;Convert to binary coded decimal
        mov     cl,al               ;Save least significant digit
        mov     dl,ah               ;Put most significant digit in dl
        add     dl,30h              ;Convert to ASCII
        mov     ah,02h              ;Put Display Character function number in ah
        int     21h                 ;Call Display Character
        mov     dl,cl               ;Put least significant digit in dl
        add     dl,30h              ;Convert to ASCII
        mov     ah,02h              ;Put Display Character function number in ah
        int     21h                 ;Call Display Character

        lea     dx,mess2            ;Put offset of message in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Display string
        mov     dl,majverm          ;Put XMM major version number in dl
        add     dl,30h              ;Convert to ASCII
        mov     ah,02h              ;Put Display Character function number in ah
        int     21h                 ;Call Display Character
        mov     dl,'.'              ;Put period character in dl
        mov     ah,02h              ;Put Display Character function number in ah
        int     21h                 ;Call Display Character
        mov     al,minverm          ;Put XMS minor version number in al
        aam                         ;Convert to binary coded decimal
        mov     cl,al               ;Save least significant digit
        mov     dl,ah               ;Put most significant digit in dl
        add     dl,30h              ;Convert to ASCII
        mov     ah,02h              ;Put Display Character function number in ah
        int     21h                 ;Call Display Character
        mov     dl,cl               ;Put least significant digit in dl
        add     dl,30h              ;Convert to ASCII
        mov     ah,02h              ;Put Display Character function number in ah
        int     21h                 ;Call Display Character
        lea     dx,lfcr             ;Put offset of message in dx
        mov     ah,09h              ;Put Display String function number in ah
        int     21h                 ;Display string

        cmp     hmaind,1            ;Check HMA indicator
        je      continue4           ;If HMA exists, go to continue4
        jmp     continue5           ;Otherwise, go to XMM processing

continue4:

        COMMENT *  The following code enables the A20 line, places the extra
                   segment at the top of conventional memory (FFFFh), and
                   sets the offset of register es, register si, to 10h (16).
```

```
                    This process has the effect of placing es:si at the
                    beginning of the HMA.  That is, FFFF0h + 10h = 100000h or
                    physical address 1,048,576. *

        mov     ah,03h             ;Put enable A20 function number in ah
        call    DWORD PTR xmm      ;Enable A20 line
        push    es                 ;Save the es register
        mov     ax,0FFFFh          ;Put FFFFh in ax
        mov     es,ax              ;Place extra segment at top of memory
        mov     si,0010h ;Put offset of 16 in si
        mov     bx,0               ;Put zero in bx

        COMMENT * The next loop is important.  It saves the data in the HMA
                  where subsequent code will write.  This process helps prevent
                  the disruption of another  program which is using the HMA.
                  Normally, one would not use the HMA unless nothing else is
                  using it.  This demonstration program should not require
                  reconfiguration of the system; so, anything in the HMA is
                  saved and restored. *

saveloop:
        mov     al,BYTE PTR es:[si]       ;Put HMA data in al
        mov     ds:savehma[bx],al ;Save data in data segment
        inc     si                        ;Increment si
        inc     bx                        ;Increment bx
        cmp     bx,17                     ;Compare bx to 17
        jb      saveloop                  ;If below, go to saveloop
        mov     si,0010h                  ;Else, data saved - reset si
        mov     bx,0                      ;Reset bx

        COMMENT *  Now use HMA. *

hmaloop:
        mov     al,ds:hmatest[bx] ;Move data at hmatest[bx] to al
        mov     BYTE PTR es:[si],al       ;Move data to HMA
        inc     si                        ;Increment si
        inc     bx                        ;increment bx
        cmp     bx,17                     ;Compare bx to 17
        jb      hmaloop                   ;Jump if below to hmaloop
        pop     es                        ;Restore es

        COMMENT *  Now that the test data is in the HMA, the data segment
                   is made the HMA to display the test message. *

        push    ds                 ;Save ds
        mov     ax,0FFFFh          ;Put FFFFh in ax
        mov     ds,ax              ;Put data segment at top of memory
        mov     dx,0010h           ;Put offset of 16 in dx
        mov     ah,09h             ;Put Display String function number in ah
        int     21h                ;Display string
        pop     ds                 ;Restore ds

        COMMENT *  The HMA is now restored. *

        push    es                 ;Save es
        mov     ax,0FFFFh          ;Put FFFFh in ax
        mov     es,ax              ;Set extra segment to top of conven. memory
        mov     si,0010h           ;Set offset to 1 byte above conven. memory
        mov     bx,0               ;reset bx
restoreloop:
        mov     al,ds:savehma[bx] ;Move saved data to al
        mov     BYTE PTR es:[si],al        ;Put saved data back into HMA
        inc     si                         ;Increment si
        inc     bx                     ;Increment bx
```

```
        cmp     bx,17                   ;Compare bx to 17
        jb      restoreloop             ;Jump if below 17 to restoreloop
        pop     es                      ;Restore es

        COMMENT *  The next code disables the A20 line.  Normally, nothing
                   is damaged by leaving the A20 line enabled. Unfortunately,
                   there are programs which use the wrap around memory feature
                   of real mode.  Therefore, it is good programming practice
                   to disable the A20 line. *

        mov     ah,04h             ;Put disable A20 function number in ah
        call    DWORD PTR xmm      ;Disable A20 line

        COMMENT * The HMA area is also released to make it available to other
                  programs. *

        mov     ah,02h             ;Put Free HMA function number in ah
        call    DWORD PTR xmm      ;Free HMA
        lea     dx,mess6           ;Put message offset in dx
        mov     ah,09h             ;Put Display String function number in ah
        int     21h                ;Call Display String

continue5:

        COMMENT * The following code demonstrates the use of the XMM to
                  transfer blocks of memory to and from extended memory.
                  The objective is to move data from the data segment to
                  extended memory and from extended memory back to another
                  section of the data segment. Then, that data is displayed
                  to demonstrate that the process works. Dáta or code may be
                  transferred in this manner. However, one should not attempt
                  to execute code in extended memory because the physical
                  address of the extended memory block is not the same as the
                  offset.  Also, real mode interrupts are not available in
                  protected mode.  The linear address of the extended memory
                  block is available through Function 0Ch - Lock Extended
                  Memory Block which will permit direct access to extended
                  memory if the A20 line is enabled.  The use of this linear
                  address is beyond the scope of this demonstration program.

                  First, the amount of free extended memory is determined
                  using XMM Function 08h.  The size, in kilobytes, of all free
                  extended memory is returned in register dx and the size of
                  the largest contiguous free block of memory is returned in
                  register ax.  The program checks to ensure there is at least
                  10 kilobytes of memory in the largest block. *

        mov     ah,08h             ;Put size of extended memory function # in ah
        call    DWORD PTR xmm      ;Find amount of free extended memory
        cmp     ax,10              ;Compare largest block size to 10
        jae     continue6          ;If memory sufficient, go to continue6
        jmp     xit                ;If memory insufficient, terminate program

continue6:

        COMMENT * Next 10 kilobytes of extended memory is requested through
                  Function 09h. The XMM returns a handle number in register
                  dx. *

        mov     ah,09h             ;Put allocate extended memory function # in ah
        mov     dx,10              ;Request 10 kilobytes
        call    DWORD PTR xmm      ;Allocate extended memory
        mov     handle,dx          ;Store handle
        lea     dx,xmal            ;Put message offset in dx
```

```
mov     ah,09h              ;Put Display String function number in ah
int     21h                 ;Display string

COMMENT * Normally, the XMM is able to move extended memory blocks in
          memory to accommodate other programs and make memory
          management efficient.  Therefore, the location of the
          extended memory block just allocated may change during the
          execution of this program.  To prevent this dynamic
          relocation, the "Lock Extended Memory Block" function may
          be called.  This function returns the physical address of
          the extended memory block permitting the use of interrupt
          15h to move data or direct access after the A20 line is
          enabled.  But, most importantly, it stops dynamic relocation
          of the expanded memory block associated with the specified
          handle.  Expanded memory blocks of any or all handles may be
          locked.  But, each lock requires a separate call to the XMM.

          In this program, the extended memory block is locked to save
          and restore the extended memory area the program uses.  This
          process is usually not necessary because the use of the XMM
          to manage extended memory precludes the need for other
          processes.  However, if a random access memory disk, e.g.
          VDISK, is using extended memory or some other program is
          using extended memory without XMM control; data could be
          overwritten. To avoid this problem in this demonstration
          program, the extended memory to be used will be saved and
          restored.  Therefore, this program should not disturb other
          programs using extended memory. There may be cases where
          this program will still interfere. The user should ensure
          that all critical processes using extended memory are
          terminated and critical data in extended memory is saved
          before executing this program.  *

mov     ah,0Ch              ;Put lock block function number in ah
mov     dx,handle           ;Put handle number in dx
call    DWORD PTR xmm       ;Lock extended memory block

COMMENT *  The following code sets up the parameter block for Function
           0Bh - Move Block.  Handles for conventional memory are
           always 0 and the logical address (called the "offset" here)
           is a segment:offset address.  The extended memory block
           handle was previously assigned by the XMM.  The extended
           memory block offset is relative to the beginning of the
           allocated memory block and can have a value ranging from
           zero to the size of the allocated block.

           THIS PARAMETER BLOCK MOVES DATA FROM CONVENTIONAL TO
           EXTENDED MEMORY. *

mov     eshandle,0              ;Source is data segment, handle = 0
lea     ax,xmms ;Put data offset in ax
mov     WORD PTR esoffset,ax    ;Put data offset in soffset
mov     ax,ds                   ;Put address of data segment in ax
mov     WORD PTR esoffset+2,ax  ;Put segment address in soffset
mov     ax,handle               ;Put handle number in ax
mov     edhandle,ax             ;Put handle number in dhandle
mov     WORD PTR edoffset,0     ;Block offset is 0
mov     WORD PTR edoffset+2,0

COMMENT * Set up parameter block to move data from EXTENDED MEMORY TO
          TO AN AREA IN THE DATA SEGMENT. Note that the source and
          destination roles are reversed compared to the previous
          block move. *

mov     ax,handle           ;Put XMS handle number in ax
mov     cshandle,ax         ;Source is extended memory
mov     WORD PTR csoffset,0       ;Block offset is 0
mov     WORD PTR csoffset+2,0
mov     cdhandle,0          ;Put handle number in dhandle
lea     ax,savexmm                ;Put data destination offset in ax
mov     WORD PTR cdoffset,ax      ;Put data destination offset in doffset
mov     ax,ds                     ;Put address of data segment in ax
mov     WORD PTR cdoffset+2,ax    ;Put segment address in doffset

COMMENT * The extended memory area to be used is saved in savexmm *

lea     si,cblklen          ;Put offset of parameter block in si
mov     ah,0Bh              ;Put move extended block function number in ah
call    DWORD PTR xmm       ;Move block

COMMENT * Call Function 0Bh - Move Block to move test data to extended
           memory. *

lea     si,eblklen          ;Put offset of parameter block in si
mov     ah,0Bh              ;Put move extended block function number in ah
call    DWORD PTR xmm       ;Move block

COMMENT * Move data from extended memory to data segment xmmout *

lea     si,cblklen          ;Put offset of parameter block in si
lea     ax,xmmout           ;Put offset of xmmout in ax
mov     WORD PTR cdoffset,ax  ;Change destination offset in parameter
                            ;block
mov     ah,0Bh              ;Put move extended block function number in ah
call    DWORD PTR xmm       ;Move block

COMMENT * Restore extended memory block. *

lea     si,eblklen          ;Put offset of parameter block in si
lea     ax,savexmm          ;Put offset of saved data area in ax
mov     WORD PTR esoffset,ax  ;Change source offset in parameter block
mov     ah,0Bh              ;Put move extended block function number in ah
call    DWORD PTR xmm       ;Move block

COMMENT * Unlock the extended memory block *

mov     ah,0Dh              ;Put unlock block function number in ah
mov     dx,handle           ;Put handle number in dx
call    DWORD PTR xmm       ;Unlock extended memory block

COMMENT * Display data. *

lea     dx,xmmout           ;Put offset of xmmout in dx
mov     ah,09h              ;Put Display String function number in ah
int     21h                 ;Display string

COMMENT * Free XMM handle and memory with Function 0Ah - Free
          Extended Memory Block. If this function is not called, the
          extended memory and handle remains active until the
          computer is reset. *

mov     ah,0Ah              ;Put free memory function number in ah
mov     dx,handle           ;Put handle number in dx
call    DWORD PTR xmm       ;Release handle and memory block
lea     dx,xmdeal           ;Put offset of message in dx
mov     ah,09h              ;Put Display String function number in ah
int     21h                 ;Display stringxit:
```

Continued on Page 26

What is it, REALLY?

Brief Descriptions of the Zenith Data Systems User's Group Software

Ed Mosher
ZUG Staff

Most of us have looked at the ZUG Software Price List selections at one time or another, and noticed some interesting sounding programs. But the Price List does not give you a real definition of what the program is, or what it is intended to do. If you ever wondered what some of them really are, this article is for you.

Included here is a partial list of ZUG Software from the latest Price List, to which I have added a short description of what the offering is and/or does. This is not intended to give you all of the specifics of the program; just a basic description to help you decide if you need further information, or if the program will fill your needs. I think you will find the descriptions helpful. Note that the remaining sections of the list will be covered in future REMark issues.

The programs are listed in alphabetical order. Please consult the latest Price List for current prices. Unless otherwise noted, all programs require MS-DOS 2.0 or higher, and less than 256K of RAM.

H/Z-100 & PC COMPATIBLE SOFTWARE

Adventure
The Z-100 (non-PC) and PC compatible versions of Adventure, the cave exploration computer game. Requires Z-DOS or MS-DOS.

Both Sides Printer Utility
This printer utility lets you print disk text files that are formatted into pages, on both sides of the paper, thereby using half the normal amount of paper for those .DOC files.

CXREF
CXREF is a C language cross-reference utility that allows complete control over what is being cross-referenced. You can, for example, cross reference only variables, or just library functions, and redirect the output to any valid device.

Debug Support Utilities
This disk contains three utilities: Processor Window, which lets you look inside your microprocessor while it is running; Breakout, which lets you run a program under DEBUG and then break out of the program back into DEBUG; and Anti-Paranoid, a utility which lets you debug "paranoid" commercial programs that otherwise cannot be debugged. Requires any version MS-DOS/Z-DOS. Anti-Paranoid requires 512K RAM; others less than 256K.

DPATH
This is an MS-DOS utility that provides a data directory path search facility. Once loaded into memory, DPATH remains resident and provides directory searching for data and overlay files in a similar way that the DOS Path command searches for executable files.

HADES II
HADES is an acronym for HUG'S Absolute Disk Editing System. In short, HADES is a screen-oriented byte (or disk) editor with file recovery and attribute modifying capabilities. For beginners, HADES is a window to the secrets of disk formats and file structures. For experts, it's a very powerful tool.

HEPCAT
Handy Engineers and Programmers CAlculation Tool. HEPCAT is both a scientific floating-point calculator AND a programmers binary calculator. It is a memory resident pop-up calculator that allows you to use it without leaving your current program. An automatic installation program senses if you have a Z-100 non-PC or a PC compatible, and the PC version supports CGA, VGA, EGA and Hercules video modes. And HEPCAT pops-up over programs that others cannot.

HUG Background Print Spooler
HBPS lets you assign from 4K to 512K of your computers memory for use as a printer buffer, depending on RAM available. When loaded, files being sent to the printer are first directed to the allocated memory which then sends the characters to the printer. This frees your computer for immediate use, instead of having to wait for the printer to finish.

HUG Editor
A fast command mode character editor originally derived from a public domain CP/M User's Group editor. This is not a screen editor, and uses no function or arrow keys. It is mainly designed for writing source code for assemblers and compilers. It makes an excellent replacement for EDLIN, the DOS line editor.

HUG Menu System
The HUG Menu System is a set of programs that lets you perform all normal computer operations from menus. The menu choices are made by using the arrow keys and selecting the item with the ENTER key. Includes the menu system, a sophisticated file manager, and a console setup utility.

HUG Software Catalog Update #1
Describes all of the HUG programs added to our collection during the years 1983 through 1985. A listing of programs added after 1985 was included in each January issue of REMark; i.e., the January 1989 REMark had all software additions for 1986, 1987, and 1988.

HUGMCP
HUG Modem Communications Package. This is a simple to use, low cost communications program that works on either Z-100s or PC compatible systems. It can be used for file transfers using RS-232 ports directly or modems. HUGMCP provides for uploading or downloading with ASCII or XMODEM protocols, a variable length buffer, concurrent printing capabilities, and the ability to send pre-defined messages/commands.

ICT 8080-8088 Translator
Interactive Code Translator. The programs on this disk can translate the assembly source code for CP/M programs into 8088 assembly language, for operation under MS-DOS or Z-DOS. They have the ability to prompt the user to make decisions that will optimize the code during the translation process.

MagBase
This is a database system designed specifically for keeping track of magazine articles. It was produced to create a database of REMark articles, but can be used for any magazine. It keeps track of articles by author, title, date, and up to 64 classifications and lets you sort by any of these. It also includes search capabilities that allow more than one parameter.

Matt
MATT is a Turbo Pascal program designed to facilitate operations on one and two-dimensional matrices. It is fully menu-driven and uses a spreadsheet-type of display to make matrix entry and editing very fast and easy.

Miscellaneous Utilities
A generous collection of utilities for MS-DOS and Z-DOS users. Includes some programs for the Z-100 only, and others for both systems. Some of the utilities include HSORT for sorting text lines, REPRINT for redirecting printer output to a disk file, ATTRIB which lets you change the attributes of any disk file, and versions of the HUG File Manager for MS-DOS/Z-DOS version 1, or for MS-DOS versions 2.0 and above.

PS' PC & Z-100 Utilities
This utility disk contains a number of handy programs developed for both types of computers by Pat (Mr. Software) Swayne, the HUG Software Engineer. The disk includes programs that test disks (hard and floppy) and computer operating speeds, as well as utilities to support batch menu systems.

REMark Volume 8: Issues 84-95
REMark issues for 1987.

REMark Volume 9: Issues 96-107
REMark issues for 1988.

REMark Volume 10: Issues 108-119
REMark issues for 1989.

REMark Volume 11: Issues 120-131
REMark issues for 1990.

Screen Dump
SCDMP is a utility that allows reproduction of a complete video screen on a dot matrix printer, including both text and graphics, without having to exit the current program. Comes with drivers for many different types of printers, including Epson, IBM, Star, NEC, Okidata, etc.

Utilities II
Another utilities disk that includes a font editor, a screen clock that can be turned off or on at will, and SEE, a replacement for the DOS TYPE command that lets you browse through text files one line or one page at a time.

Z-100 WordStar Connection
This is a set of utilities that lets you run PC compatible versions 4.0, 5.0, 5.5, and 6.0 of WordStarR and run it on your Z-100 (non-PC) computer. The Word Finder thesaurus program (found in version 4.0) is also supported by this program. Requires 320K if Word Finder is used, and Z-DOS/MS-DOS 2.0 or higher.

PC COMPATIBLES

CardCat
This program permits you to organize information in a manner similar to that found in a library's card catalog. Entries are stored on disk and can be edited and searched using built-in routines. Detailed help screens are included, as are several versions designed for RAM level up to 640K. Requires MS-DOS 2.11 or higher.

CheapCalc
CheapCalc is a minimal but very useful spreadsheet program that can introduce you to these types of programs with little cost. With CheapCalc your computer screen becomes a window to a large worksheet where you can enter mathematical problems and see almost instant solutions.

CP/EMulator II & ZEMulator
CP/EMulator is a program that lets you run standard 8-bit (8080) CP/M programs under MS-DOS on any DOS-compatible computer, and does not require an 8-bit processor. ZEMulator provides most of the Z-100's keyboard functions, escape code functions, and block graphics (ala H-19) characters on any PC-compatible computer, including the Z-100 series PC compatibles, but does not require the Z-319 video card. These two programs greatly expand the types of programs you can run on a PC compatible machine. Runs on Z-DOS/MS-DOS 1.25 or higher.

Dungeons & Dragons
D & D is the ZUG rendition of the popular game of the same name, played in real time. This version displays graphic representations of the rooms, halls, and doors in the areas you enter. The object of D & D is to locate the lord master of the dungeon on one of the 50 dungeon levels. Requires GW-BASIC 2.0 or higher.

EZPlot II
This program is a user-friendly high resolution graphics function plotting program for engineers, scientists, and just about anyone who would like to have their printer plot curves from disk data. It is menu driven, can plot as many as three functions [i.e., f1(x), f2(x), f3(x)] on the same set of axes, and can plot contours or path functions. EZPLOT will also sort data if necessary, and can use data from BASIC, FORTRAN, PASCAL, C, or COBOL language data files.

Grade
This "teacher's aid" program lets a teacher enter class grades into a computer "gradebook" in random order, just as they were in the old fashion paper and pencil type. It provides a printout suitable for permanent records, as well as scores and summaries to use for display to a class. This menu-driven program can handle up to 100 students and 60 scores for each student. Requires GW-BASIC 1.0 or higher, 320K RAM.

Ham Help
This program uses your PC compatible computer to calculate the MUF (Maximum Usable Frequency) for a path between two geo-graphical locations, using data provided by hourly broadcasts from NBS station WWV. It will calculate the MUF for each 30 minute period of one day, and display the data in chart form on the screen. A great program for all of the DX-chasers.

Keymap
KEYMAP lets you designate the responses produced by your computer's function keys. It works like the KEY command in GW-BASIC except that more keys can be defined. In turn, those keys become part of the "system" and can be used with any program or operating system. For example, you can redefine key F1 do produce the command "DIR A:" (including the return!) so that a single keystroke lists the files on the disk in drive a. Requires MS-DOS 1.25 or higher.

Laptop Utilities
This disk contains a group of utilities designed to make life with your laptop more enjoyable and productive. Included on the disk are CURSOR.COM, a program that makes your cursor more visible; CAPCON, a program that lets you swap the functions of the CAPS LOCK and CONTROL keys on your keyboard; HFM, the HUG File Manager that makes it easy to keep up with your disk files; and SEE, a program that lets you scan text files one line or one page at a time.

PS' PC Utilities
This disk contains a number of handy utilities developed by our own Pat Swayne (ZUG Software Engineer) for use with PC com-patible computers. Included are a character undelete utility for WordStar, a fast alphabetizing directory program, a screen clock, and a collection of drawing libraries for use with four different CAD/drawing programs.

Powering Up
A collection of the first series of articles by the same name, written by William Adney, REMark Contributing Editor. This is a

"beginner's manual" for using your PC compatible computer, and includes many tips for using DOS commands more effectively. The book contains 15 chapters dealing with everything from set-ting up and maintaining your computer system to understanding video hardware and selecting utility and application software. This is a MUST HAVE book for those new to computers, or those who want to expand their knowledge of PC compatibles.

Screen Saver Plus
This disk contains four DOS utilities for the H/Z-100 PC series computers: ScreenSaver, DualScreen, ChangeSpeed, and a Print SCreen utility for Text Modes. ScreenSaver blanks your screen after a preset time of keyboard and screen inactivity; DualScreen lets any text appearing on the color graphics screen, to appear on a Z-329 hi-res mono screen. ChangeSpeed is a utility for the H/Z-241 computers that lets you change processing speed to that of a standard 5 MHz H/Z-100 series PC and back to normal, at will. Print SCreen is used in conjunction with a graphic Print Screen utility to allow it to work in text modes.

Skyviews
Did you ever wonder just what stars were overhead on a given night? With SKYVIEWS, you can plot the positions of major stars (those with 4.0 or greater magnitude) as well as those of the sun, moon, planets and constellations. This astronomer's helper provides right ascension-declination, azimuth-elevation, and other information about the individual celestial bodies, and marks major constellations on the screen to aid in locating them.

TCSpell
This spelling checker is designed to be quick and easy to use. Dictionary size is limited only by disk space, while the document size is limited only by memory if the number of unique words exceeds the remaining space left in RAM. TCSPELL can be used with standard ASCII and WordStar file formats. Includes a tutorial and documentation.

Ultra RTTY
Another championship program for the hams. ULTRA RTTY pro-vides you with the means of using your PC compatible computer to receive and transmit RTTY using standard Baudot and ASCII codes. This program comes with excellent documentation and requires some type of RTTY Terminal Interface, such as the Heathkit HD-3030. It covers all popular Baudot speeds and ASCII 100/300 bauds, and supports COM1 and COM2 ports.

YAUD (Yet Another Utilities Disk)
The name says it all. This disk is another collection of useful utility programs for your H/Z PC compatible. Included are GD, a program that takes you directly to a directory on a given drive, without using the cd\xxx\xxx DOS command; HATCH, a program that places a cross hatch or dot pattern on your video monitor screen to help you with alignment and convergence adjustments; and revised versions of HFM.COM (HUG File Manager), DT.COM (Directory Tree), F.COM (Find, a file search utility), and D.COM (an alphabetizing directory display), among others.

Continued from Page 12

until you pass them to a function. Here is an example:

```
#include <iostream.h>
int char_cnt (char in_string[]);
main()
{
   char str[20];
   cout << "Please enter your name: ";
   cin >> str;
   cout << "Your name has " <<
      char_cnt(str) << " chars in it.";
}
int char_cnt(char in_string[])
{
   for(int i = 0; in_string[i] !=
              NULL; i++)
      ;
   return(i);
}
```

We sent the string to the function without passing its length with it. In fact, the main() function didn't know the length of the string. The function simply steps through the string until it finds the NULL. Note the run condition in the for loop is in_string != NULL. But since NULL has a numerical value of 0, you can also do it like this:

```
int char_cnt(char in_string[])
{
   for(int i = 0; in_string[i]; i++)
   ;
   return(i);
}
```

This for loop runs until in_string[i] evaluates to FALSE or 0, which occurs when in_string[i] is the NULL at the end of the string. So it counts the chars in the string and stops when it gets to the NULL. It does not count the NULL.

Initializing Strings

You can initialize a string one char at a time just like any other array. For example:

```
char bovine[] = { 'C', 'O', 'W', '\0' };
char porcine[] = { 'P', 'I', 'G', 0 };
char feline[] = { 'C', 'A', 'T', NULL };
```

will all work just fine, but there is another way which is often used, and that is to initialize it with a string constant like this:

```
char porcupine[] = { "prickly" };
```

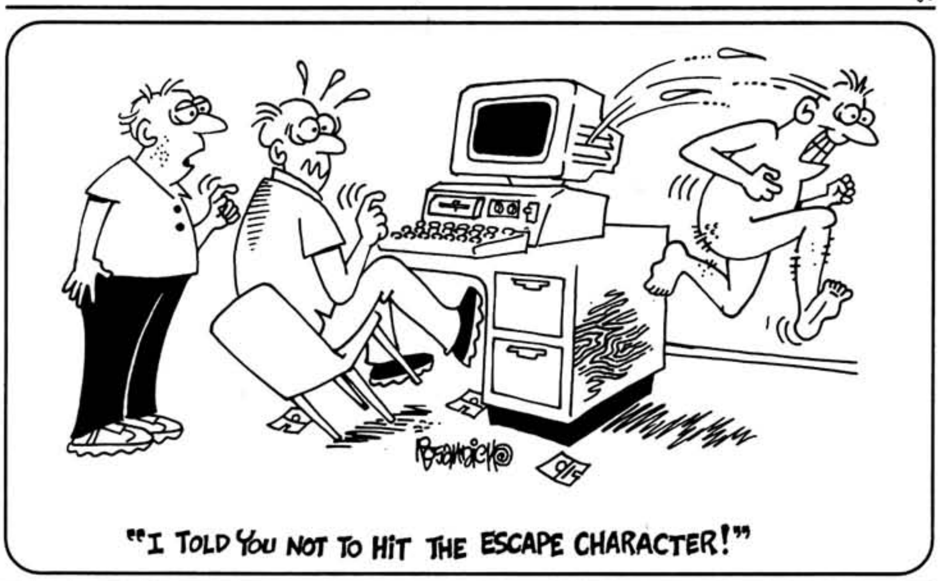
We have used string constants (also called string literals) in several of our programs already. A string constant is a group of chars with quotes around them, like "cat". The NULL is there, although invisible. For this reason, it takes an array 4 elements long to store "cat".

There are quite a few functions in the standard library which operate on strings. You should be familiar with what is available, to save yourself time writing functions which already exist. Most of the string function names begin with str.

Be sure to try a few of these new ideas between now and the next class. Remember, you learn to write programs by writing programs.

Continued from Page 23

```
xit:
        mov      ah,4Ch         ;Call terminate function
        int      21h
code    ENDS

stack   SEGMENT  stack          ;Program stack segment
        DW       128 DUP(?)     ;Define stack space
stack   ENDS

        END      start          ;Mark end and define start
```


"I TOLD YOU NOT TO HIT THE ESCAPE CHARACTER!"

Harnessing FORTRAN

or

I/O, I/O, It's Off to Work We Go (The FORTRAN Way)

D. C. Shoemaker
5828 5th Avenue NW
Seattle, WA 98107

FORTRAN is nobody's favorite language. Probably not even a second-favorite language. But it's not going to go away in your lifetime or mine. A great deal of scientific and engineering work has been and is still done in FORTRAN, both in this country and abroad. Despite the popularity of Pascal, C and other more modern tools, FORTRAN remains one of the few truly portable languages. It's actually possible to write a complex FORTRAN program that will run unchanged on an H89, a Z100, an H386, a DEC 11/782 supermini and a UNIVAC 1100/62 mainframe. Speed of execution, of course, is not a topic for discussion. (See the Historical Footnote at the end of this article.)

In my case, FORTRAN suddenly became interesting when I discovered that there is a version that is reasonably fast, standard (meaning portable) and most important, available to Z-100 users like myself. The portability part was significant, since in my work I often have to write programs that will run on a wide variety of machines. And yes, before I get some irate comments, there are other so-called portable languages, with C usually at the top of the list. This is, of course, nonsense, as you will know if you have tried to convert BDS C to Lattice C to Microsoft C to Turbo C++, and so on. The work is not pleasant.

All is not well in the FORTRAN world, however. I must now point out the main sticking point. Portability does not extend completely to file input/output. If you've tried file I/O with Microsoft BASIC, you may have a hint of the trouble to come. FORTRAN is, if anything, worse. The trouble starts, as it often does, with Microsoft.

The version of FORTRAN we'll discuss is Microsoft FORTRAN 77, version 3.10. This was commonly available in years gone by, and accompanied many of the Z-100 computers bought by various Government agencies in the mid-80's. As a result, you may find that it comes with a used Z-100 bought at a surplus sales or from a used computer dealer. I've also seen it advertised by Henry Fale at QuikData for a few dollars. Don't worry if the manual got lost or is no longer available; it's probably for the best. This was one of Microsoft's worst manuals in that it contained very few examples. Each command was explained, but you were left to figure out how each was supposed to work.

Microsoft and I have one thing in common; we both assume that you have FORTRAN, that you need to use it for something meaningful, and that you have a desire to see the results of your work appear on the CRT screen, the printer, or be written to or read from a disk file. This brief article is not intended to introduce you to FORTRAN programming; that would take a book. At the end of the article I've included some recommendations if you need help getting started.

For now, I will focus on the problems of getting information printed on the screen, the printer and to a disk file. These all fall under the category of FORTRAN I/O.

The four programming examples that follow are intended to give you a set of models that you can work with and develop for your own uses. The actual program is trivial, just a list of floating point numbers used to show when the FORTRAN compiler breaks down. It was lifted from Alan R. Miller's book, "FORTRAN Programming For Scientists and Engineers".

The first program, Listing 1, shows how to write data to the console. In most programming languages this is a simple procedure. For instance, in BASIC one just says PRINT. In Pascal, the word is WRITE. However, in FORTRAN, before you can WRITE anything, you have to tell the program where to put it (politely.) FORTRAN uses a concept of input/output called the Logical Unit Number.

The Logical Unit Number, or LUN as we shall now call it, was actually intended to help in that it made FORTRAN device-independent. The programmer only needed to know what physical device corresponded to what logical device, and use the right number. In standard ANSI (American National Standards Institute) FORTRAN 77, each LUN from 1 to 10 was supposed to correspond to a particular physical device such as a punch card reader, a tape drive or a line printer. In Microsoft FORTRAN 77, the logical units were somewhat re-

```
C          PROGRAM CONSOLE.FOR
C
C          WRITTEN IN MS-FORTRAN 77 VER 3.10.  PRINTS TO CONSOLE (*).
C
           REAL X
           INTEGER I, N
           N = 50
           X = 1.0E-4 / 3.0
           DO 10 I = 1, N
                 X = X / 10.0
                 WRITE(*, 101) I, X
10         CONTINUE
           STOP
101        FORMAT(1X, I4, 1PE15.6)
           END
```

Listing 1
An example of writing to the console, terminal or CRT.
The Logical Unit Number is replaced with the * character.

```
C       PROGRAM PRNTEST.FOR
C
C       WRITTEN IN MS-FORTRAN 77 VER 3.10.  PRINTS TO FILE CALLED 'PRN'
C         WHICH FORTRAN TREATS AS THE DEVICE LPT:  LUN FOR FORTRAN LINE
C       PRINTER I/O TRADITIONALLY IS 2.
C
          REAL X
          INTEGER I, N
C
C       NOTE THE USE OF THE CALL STATEMENT AND THE SINGLE QUOTES
C
          OPEN(2, FILE = 'PRN')
          N = 50
          X = 1.0E-4 / 3.0
          DO 10 I = 1, N
                  X = X / 10.0
                  WRITE(2, 101) I, X
10        CONTINUE
          STOP
101       FORMAT(1X, I4, 1PE15.6)
          END
```

Listing 2
In this example, LUN 2 is assigned to the printer, but DOS treats it as a file.

vised due to the widely remarked lack of tape drives and card punch machines so that LUN 1, 3, 4 or 5 all refer to the console terminal. LUN 2 is the line printer, and LUN 6, 7, 8, 9 and 10 were disk files.

Now that you've made a note of that, forget it. It turns out that Microsoft actually doesn't care where you send the output or from where you read the input, and the logical unit numbers so painfully dug out of the manual are essentially useless. In order to send the output anywhere except to the console (CRT, to you and me), you will need to master the CALL statement.

```
C       PROGRAM DISKTEST.FOR
C
C       WRITTEN IN MS-FORTRAN 77 VER 3.10. PRINTS TO A SEQUENTIAL FILE
C       CALLED DATA ON THE CURRENT DRIVE.
C       LUN FOR DISK I/O IN THIS EXAMPLE IS 6.
C
          REAL X
          INTEGER I, N
C
C       NOTE LUN 6 IS A FICTION, MAINTAINED FOR TRADITION
C
          OPEN(6, FILE = 'DATA', STATUS = 'NEW')
          N = 50
          X = 1.0E-4 / 3.0
          DO 10 I = 1, N
                  X = X / 10.0
                  WRITE(2, 101) I, X
10        CONTINUE
          STOP
101       FORMAT(1X, I4, 1PE15.6)
          END
```

Listing 3
This program opens LUN 6 as a disk file named DATA, which will need a new file on the default disk drive.

Microsoft FORTRAN will allow one useful shortcut, and that's the use of the asterisk (*) in place of the console LUN. So where standard FORTRAN would require you to read or write to the console as LUN 1, we can use the * character instead. You may use the LUN 1 if you wish, but as we'll see, this also requires the use of the CALL statement.

One other note about Listing 1 is the syntax for the WRITE statement. The asterisk must be followed by a comma, then the statement number that contains the formatting instruction. These must be in parentheses, followed by the variables to print. This follows standard FORTRAN syntax, but it looks rather odd with the asterisk.

In Listing 2, the same program has been modified so that the data will be printed on the printer. The LUN used here is 2, for the sake of tradition, but actually makes no difference, since the OPEN statement assigns the DOS file called PRN to the LUN. As we saw above, almost any number could be used, provided the OPEN statement assigns the file name PRN. Especially note the use of SINGLE quotes. Double quotes are NOT allowed.

There may be one small problem with this technique. Depending on the type of printer, the amount of buffer space, the version of DOS, the printer I/O card, the height of the tide and the phase of the moon, you may find the last line of the printed output hung up in the DOS I/O buffer and not sent to the printer. In this case, close LUN 2, or just re-open it.

In Listing 3, the same program has again been modified to send the data to a disk file called DATA. The name of the file may, of course, be any legal DOS file name, and may include a drive designator such as:

A:MYDATA
C:COST.DAT
B:STAT.FIL

Here we've used the LUN 6. This is again for traditional reasons. As before, this is irrelevant, and could be any number, since it's the OPEN statement that makes the assignment. Also, note the STATUS = 'NEW' which creates a new data file on the disk.

Finally, Listing 4 is a program to read the data file created by the program in Listing 3 above, and print it on the console. Note the LUN for the file and the console. Also note use of the STATUS = 'OLD' to specify an existing file.

Historical Footnote

It would be difficult to overstate the importance of FORTRAN to everything you do with your computer. The Spring of 1957 saw a major revolution in the way programs were created on computers, largely due to John Backus and his group at IBM. There were compilers before FORTRAN, but they were highly machine-dependent, and programs written in AUTOCODE were completely different from programs written in FLOWMATIC. Suddenly it became possible to write an algebraic program that would run on any number of different computers. The programmer's time was becoming more important than the machine time, and portability and ease of use began to count for more than sheer execution time. By 1960 the revolution was complete, and the path was open to BASIC, Pascal, C and a host of other compilers that would produce reasonably fast, efficient code economically. But FORTRAN was the first.

Suggested Reading and Sources

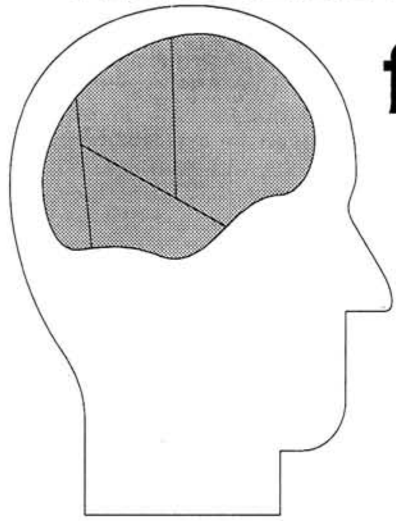
A Simplified Guide to FORTRAN Programming, by Daniel McCracken, is the hands-down best introduction to standard FORTRAN. Full of examples, exercises, illustrations, but thin on input/output. Published by Wiley, you should be able to find or order this book from any community college or university book store. About $15.00.

Illustrating FORTRAN, by Donald Alcock, is an excellent discussion on portable FORTRAN, showing techniques that insure a program will run on virtually any computer that speaks FORTRAN. Published by Cambridge University Press, ISBN 0 521 28810 X, this one will be difficult to find, but will repay the effort. About $10.00.

FORTRAN Programming for Scientists and Engineers, by Alan R. Miller. An excellent book for gaining an idea of the number-crunching power or FORTRAN, which

Continued on Page 36

Memory Transplant for Z-386/16



Mike Wolfson
1842 Mifflin Avenue
Ashland, OH 44805

Have you ever gotten a sinking feeling in the pit of your stomach when you turn on your computer and discover that your system is a "little under the weather"? That happened to me recently when one of my hard disk drives died. I turned the system on and it did not boot normally. After about 90 seconds I got a "Drive Error" message and tried to remember when I had last backed up my hard drives.

Diagnosing the Problem

The first thing I did was hit the reset button that I had installed in my Zenith 386/16 when I first got it a couple of years ago. I really hoped that the problem was just a "random glitch". No such luck. It was time to get serious. My next step was to pull out a bootable copy of DOS on a floppy disk and get into the MONITOR so that I could boot from the floppy. Fortunately, that worked and I was able to get the system up and limping along.

Once I had gotten the system running from the floppy, I tried to access both hard disks. In my case, I had a Seagate ST-225 that originally came with the Z248 that I purchased about 5 years ago. When I upgraded to a Z386 using the HUG upgrade kit, I reinstalled the drive. It was slow, but then it was paid for. My second drive was a Seagate ST-251-1 that's about 3 years old. I was able to access the C: drive (the ST-225) without any problem. When I tried to do a DIR of either D: or E: (the two partitions on the ST-251) I got "Invalid Drive" error messages.

Next, I tried to reboot the system from the hard disk and listened very carefully to the noises coming from the hard disks. It quickly became apparent during a cold boot that on one of the fixed disks the arm holding the read/write heads was not moving properly. Evidently, what was happening was that the arm was not able to move properly to initialize the drive or to access data. I discounted the possibility of a bad connection because the drive was trying to initialize and because the nature of the problem did not seem to be related to a bad connection. That left only one alternative — I had a hardware failure in the drive mechanism itself.

Alternatives

After repeatedly trying to reboot the system and access the malfunctioning drive, I decided I had better look at my alternatives. First of all, I could just take the bad drive out of the system and rely on the 20 megabyte drive as my only hard disk. That alternative was not appealing at all. Using the slow, small 225 was tolerable because I had a faster and larger drive to supplement it. Using it as my only hard disk was not an alternative I was willing to accept.

My second option was to pull out the drive and send it to one of the companies that repair hard disk drives. I had never tried this before, so I picked up a copy of "COMPUTER SHOPPER" to find out what it would cost. According to a number of advertisements, it would cost approximately $150 to repair the drive. That seemed excessive for a 3 year old drive, especially when I considered that most of the information I would loose by throwing away the drive could be replaced. Also, the guarantee provided with the repaired drive was not very good. It looked like my only reasonable alternative was to bite the bullet and buy a new drive. This third option was in my opinion, the best, but while choosing it I created a further series of decisions to work my way through.

There are currently five, different, major types of hard disk subsystems available for PC compatible computer systems. The most familiar type is a system based on the MFM (Modified Frequency Modulation or ST 412/506) standard. This type of drive has been around for years. By today's standards, it does not offer good performance or capacity. The second major type of hard disk is an enhanced form of MFM called RLL or Run Length Limited. This standard adds roughly fifty percent to the capacity of an MFM drive. A special RLL compatible hard disk controller board must be used. In some cases, a MFM drive can be used with an RLL controller. This practice is not recommended, though, because the drive is not certified to meet RLL standards.

Hard disk drives using MFM and RLL technology are generally considered to be out-of-date; especially for 386 based systems. Today, higher capacity hard disks are available with faster access times. Known as IDE, ESDI and SCSI drives, these hard disk subsystems offer higher capacity and better performance than fixed disks using older MFM and RLL technology.

Hard disks using "IDE", or Integrated Drive Electronics technology are a recent addition to the market. They offer good performance and capacity in a small package. The disk platters in an IDE drive are typically 3.5 inches in diameter. Manufacturers are developing new versions that have disk surfaces closer to 2 inches in diameter.

Another new hard disk type is ESDI or Enhanced Small Device Interface drive. These drives have better performance specifications than IDE drives. They're also more expensive. While IDE hard disks are made that can hold a maximum of about 200 megabytes, ESDI drives are readily available that hold 600 megabytes. Drive capacities of more than a gigabyte can be obtained using ESDI technology. Another difference is that ESDI drives only seem to be available with 5.25 inch platters. Some manufacturers might be developing these drives with 3.5 inch disks, but I haven't seen any advertised.

The final type of hard disk system available to the users of PC compatible computers is known as "SCSI", which is an abbreviation for Small Computer System Interface. Hard disks using this technology have been around for awhile. Apple's Macintosh has used SCSI drives for years. They've been available for PC compatibles for several years as well, but there have been problems with the way that SCSI technology has been implemented in the PC compatible world. The original SCSI specification was not strictly followed by every company adopting this "standard" for products they developed.

Because each one of these SCSI implementations was different, one of the most attractive capabilities of the SCSI interface couldn't be exploited. That capability is the ability to "daisy-chain" different SCSI compatible peripherals together so that only one connection has to be made to the computer.

This shortcoming seems to have been corrected with the development of "SCSI-II". SCSI-II is an enhancement of the original standard. It offers better performance than the original SCSI specification. SCSI drives also offer storage capabilities similar to those of ESDI drives. The downside of these drives is that because they are so new, they are more expensive. In my opinion, it is too soon to tell if SCSI-II will be more successful than the original.

Making a Decision

After looking at the alternatives, I decided to replace my entire hard disk subsystem with an IDE based alternative. I chose an IDE based drive for a number of reasons. While the easiest choice would have been to replace my ST-251 with another drive of the same type, I was also worried about the reliability of the other hard disk in the system. My second hard disk, the ST-225 that came with my Z248 was about 5 years old. As you know, and as I found out, hard disks don't last forever and I didn't want to have to replace that drive 6 months after replacing this one. Another consideration was performance. IDE drives typically have access times of about 15 to 25 ms.. My ST-251 is rated at 24 ms. and the ST 225 at 65ms. On a Z386/16, I really noticed the lag time. A hard disk with a faster access time would help me work faster and more efficiently.

Having decided to purchase an IDE hard disk system, the next question I needed to answer was — which one? Since I was replacing both of my hard disks, I needed at least 60 megabytes of capacity. I didn't find any drives of this size. There were a couple of drives that would hold 50 megabytes, but none that would hold 60 megabytes. My options were limited to 40 or 80 megabytes. I briefly fantasized about 100 megabyte drives, but decided against one because of cost, lack of need and the difficulty of backing up such a large drive to diskettes. By default, I decided to buy an 80 megabyte unit.

Since I had answers to the questions of (1) what type of drive system to buy and (2) how big of a drive to get I next needed to collect some information on how much this type of drive would cost. To answer that question, I turned to the most recent issue of "COMPUTER SHOPPER". If you're familiar with the magazine, no explanation is needed. If you've never seen a copy, buy one sometime. Its an experience you won't forget. If a piece of microcomputer equipment exists, you'll find it and at least 2 suppliers listed in "COMPUTER SHOPPER". Its a computer flea market/swap meet on paper. I ended up with a list of at least 30 mail order firms offering CONNER, QUANTUM, SEAGATE, and MAXTOR drives. While preparing this list, I paid particular attention to suppliers who offered the drive in a kit complete with a new controller card. Since IDE drives are not compatible with MFM controllers, I needed to replace my hard disk controller as well. To make things easier, I made sure that the controller would also handle floppy drives. While I was preparing the list of hard disk prices, I also started to research who made dependable IDE disk drives. I decided the best place to start was on the ZUG BBS. I save and archive all of the messages that I think are interesting and useful from the BBS each week. It turned out that just recently one of the regular participants on the board upgraded his fixed disk system with a MAXTOR drive. I read how he had done it and his reasons for selecting the MAXTOR drive over other brands. I discovered that I shared some of his concerns.

I also looked in old copies computer magazines that I keep on file. The 25 September, 1990 issue of "PC MAGAZINE" contained an extensive review of the service and reliability history of a number of computers, monitors, printers, and hard disk drives. MAXTOR, along with a number of other manufacturers, received a favorable review in the article's hard drive section. That review was another point in favor of my choosing a MAXTOR hard disk. Finally, I looked in a back issue of "COMPUTER CURRENTS". One of the regular contributors who reviews a lot of equipment for the magazine also had good things to say about MAXTOR drives.

It looked like a MAXTOR 80 megabyte IDE hard disk was in my future. All I needed to do now was decide where to buy it. Actually, that was a pretty easy decision. The firm, HARD DRIVES INTERNATIONAL, has an excellent reputation for selling good hard disks at fair prices. They also offer a 30 day money back guarantee if you decide you're not satisfied with your purchase. A quick call and the drive was on its way.

Installing the New Hard Disk

This step always bothers me. It seems so easy, but I usually manage to screw it up somehow. This time everything worked out fine. I took the case off my computer and removed the two disk drive chassis. I removed the two hard disk drives from their respective chassis and reinstalled one chassis in the computer. The second chassis I set aside for the time being.

Next, I unpacked the new hard disk and followed the directions included with the disk and checked to make sure that all of the jumpers on the drive were properly configured. Then, I attached rails to the sides of the drive with screws so that the new drive would fit in a standard 5.25 inch drive bay. Using more screws, I then installed the hard disk in the second disk drive chassis.

After installing the hard disk in the chassis, I removed the MFM controller card from the right hand most slot in the computer. When I tried to place the new hard/floppy controller card in the same slot, I found that it would not fit. After carefully looking at the situation I realized that the lugs holding the controller card to the mounting bracket were too long to fit properly. Since the card was small, I decided to remove the mounting bracket and to install the card in the slot without it. I used a spare blank bracket to cover the empty bracket opening in the back of the computer. This problem was the only one I had during the entire installation of the hard disk.

IDE drives only need a single 40-wire connection with the controller card. This cable was supplied with the hard disk and I followed the instructions that described how to make the correct connections. The drive also has a standard power connection, so feeding power to the drive was no problem.

After hooking up the new hard disk, I reinstalled my floppy drives and plugged in the floppy drive controller cable to the connection provided for it on the controller card.

Preparing the Hard Disk for Use

Once I had installed the new hard disk, it was time for the "acid test". I plugged the computer in and turned it on. Everything sounded okay and I was able to enter the MONITOR without any problems. At this point, I did experience some doubts. The ROM in my system is version 2.1E. It was produced before IDE drives were widely available and does not take this type of drive into consideration when giving drive definition types. According to the MAXTOR instruction manual, the user can define his own drive parameters or use the parameters from a similarly sized drive. On my system, I can't define my own parameters in ROM. However, drive type 35 is for an 80 megabyte hard disk. I decided to try it and see if it would work. If it didn't, I would have to order a new set of ROMs.

One of the unique features of IDE drives that I had heard about was that the IDE embedded controller had a "translation" feature that allows the user to install IDE hard disk drives in systems that do not offer the correct drive parameters. Apparently, this is indeed the case. I defined my drive as type 35 and proceeded to partition the drive using PART from DOS 3.3+. I created three partitions of roughly equal size. The total of the three partitions equalled the capacity of the drive as defined by MAXTOR. This total was less than the total as defined by the ROM. In spite of this difference, the drive has a total formatted capacity of about 77 megabytes.

Note that I did not run PREP on the drive. According to information supplied by MAXTOR, doing a low level format on the drive I bought will cause damage to the drive. I'm not particularly happy about this situation, but a lot of drives have been made this way and they seem to work satisfactorily. Only time will tell if this technique works successfully.

After I divided the hard disk into three logical drives, I formatted each partition, established a primary boot partition and transferred the system files to that part of the hard drive. Everything went off without a hitch! By now I was feeling pretty good. I then reloaded all of my files and software so that the new drive duplicated my old hard disk setup. Feeling really brave, I shut everything down and did a cold boot. Almost before I could sit back in the chair, I was looking at my familiar opening screen on a fully restored system. The patient survived the operation and was back in action; as good as new.

Conclusion

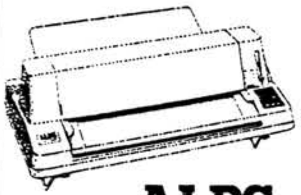
In looking back, I'm glad I took the time to fully research my alternatives. I learned a lot about hard disk drives and have a better appreciation for how they work and under what circumstances one type is preferable over another. Hopefully, I won't have to replace another hard disk anytime soon, but if I do, I'll be ready.

W S Electronics

(513) 376-4348 ** Since 1975 **** (513) 427-0287**

1106 State Route 380, Xenia, Ohio 45385

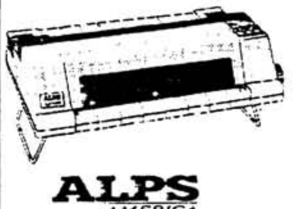
YOU AND YOUR BIG IDEAS.



* 300 cps draft speed
* Wide carriage
* 24 pin printing
* Front panel controls

ALPS *Allegro 500XT™*

THE ALPS ASP1600 PRINTER. COSTS VERY LITTLE, JAMS NOT AT ALL.



- Auto tear bar prevents paper waste. With the flick of a button, your fanfold paper advances to the perforation, then automatically returns to top-of-form position.
- Rugged 9-pin head delivers crisp output at 192 cps in draft mode, 38 cps in letter quality.
- Compact 9-lb. body makes for easy portability.
- Printer stand is built-in.
- Prints labels easily.
- Full Epson FX-85 compatibility.

ALPS AMERICA
Built by popular demand.

SPECIAL

Z-315	EMS Kit for Z-159	$ 10.00
Z-417	H D Controller Z-248	$100.00
Z-317	H D Controller Z-150	$ 75.00
ZCM-1400-1	Swivel Base for 1490	$ 5.00
CB-4364-39	Diagnostics for 184	$ 5.00
ENABLE 2.0	Word Processing	$ 99.00

Quantities limited to stock on hand

ATTENTION:

Federal Government Offices
We stock ALL ALPS Printer Models and we stock ALPS PARTS and RIBBONS for all ALPS models including your P2000's and ASP-1000's

****GOVERNMENT DISCOUNTS OFFERED****

We are looking for good dealers.
ALPS Authorized Distributor and Service Center

Reader Service #149

QUIKDATA - 15 YEARS OF H/Z SUPPORT!

YOUR H/Z RAM & DRIVE EXPERTS!

ACCELERATE YOUR PC!

From Sota Technologies, Inc., the fastest and most proven way to breath new life in your Heath/Zenith PC/XT computer, giving it -AT compatible speeds! Turn your turtle into a -286 rabbit with a 12.5 Mhz 80286 accelerator board. Complete with 16K on-board CACHE.

There are other ways to speed up your H/Z PC/XT computer, but you spend too much and get too little. Our EXP12 **286i** is the effective solution, making your H/Z150/160/150/158/159 series of computers, or any general PC/XT computer faster, in many cases, than a standard IBM AT type computer! Simple half-card plug-in installation with switchable speed control. **You won't believe your stop watch!**
EXP-12 - **$295**
EXP386 - **$469** Much faster 16Mhz 80386 SX version

MEMORY UPGRADES

Note: All memory upgrades come without memory chips. 150ns 256K DRAMs are $1.79 as of this printing.

Z150MP - $17 Will allow you to **upgrade your H/Z150/160 to up to 704K** on the main memory board, using up to 18 256K DRAM chips.

MEGARAM - $43 **Upgrades your H/Z150160 series** with up to 704K of main memory, and about 512K for RAMDRIVE memory. Includes documentation, software RAMDRIVE disk, PAL and jumper wire. For the full **1.2 megs total memory**, 45 256K DRAM chips are required.

EAZYRAM - $89 Upgrades EaZy PC from 512 to 640K

ZMF100 - $55 Will allow you to **upgrade your H/Z110/120** (old motherboards; with p/n less than 181-4918) **to 768K system RAM**. Requires 27 256K DRAM chips.

Z100MP - $55 Similar to ZMF100 above, but for new motherboards with p/n 181-4918 or greater.

Z159 EMS LOGIC UPGRADE includes the PAL chip, logic chips and one bank of 256K (9) chips. Up to two banks (18) can be installed for a total system RAM of 1.2 meg on the Z159 main board.
Z315-1 - $79!!

3MB RAM BOARD for Z241/248 computers is an excellent memory card. Will backfill your 512 to 640K, and provide both extended and expanded RAM, all can coexist. Uses 100ns M256-15 RAM chips, 36 per megabyte desired. Minimum of 18 DRAM chips required.
EVATRD - $119

Z248/12, Z286LP RAM UPGRADE Z605-1 consists of 2MB SIMM 80ns RAM kits to upgrade your H/Z systems.
Z605-1 - $169

Z386/20, Z386/25, Z386/33, Z386 EISA 2MB SIMM 80ns UPGRADE to add increments of 2MB to these systems. Two required.
ZA3600ME - $95

ZA3800MK - $389 4 megabyte SIMM upgrade for Z386/20, /25, /33, /33E. Must have 4-1 meg SIMMs installed first.

WINCHESTER UPGRADE KITS

PCW20 - $249 Complete MFM winchester setup for a H/Z150, 148, 158, 159, 160, PC etc. Includes **21 meg** formatted half-height Segate ST-124 37ms drive, controller, cable set, doc.

PCW30	$295	32 meg with 28ms Segate ST-138-1.
PCW40	$329	42 meg with 28ms Segate ST-251-1.
PCW80	$595	80-meg with Segate ST-4096 full size drive.

We also have the DTC controllers ($59) and daughter board expansions ($65) to place a hard drive in the **H/Z148** computers.

BARE WINCHESTER DRIVES/CONTROLLERS

ST-124	$209	21 meg 37ms 3.5" drive in 5" frame
ST-138	$249	32 meg 38ms autopark 3.5" drive in 5" frame
ST-251-1	$289	42 meg 28ms autopark HH drive
ST-4096	$549	80 meg 28ms autopark FH drive
WDCON	$59	PC/XT hard drive controller board
WDATCONF	$129	1:1 interleave HD/floppy controller for AT's

FLOPPY DRIVES

T28M	$159	2.8MB 3.5" floppy
MF501	$71	5" 360K DS/DD drive
MF504	$81	96 TPI 1.2 meg AT/Z100 drive
MF353	$79	720K 3.5" drive in 5" frame
MF355	$89	1.4 meg 3.5" AT drive in 5" frame
SIEM-R	$39	40 track SS refurb (H8/H89 type)
TM100-4R	$75	80 trk DS refurb (H8/H89 type)
TM100-2R	$69	40 trk DS refurb (H8/H89 PC type)

ANY DRIVE IN YOUR PC/XT/AT

With the CompatiCard, you can install up to four additional drives, of any type in your PC/XT/AT computer. Add a 1.2 meg 5" floppy, or a 1.44 meg 3.5" floppy, or any other drive, including 8" to your system. The CompatiCard (CCARD) will handle up to four drives, and the CompatiCard II (CCARD2) will handle up to 2 drives. CCARD4 has boot ROM to allow it to be used as primary boot controller in systems that allow you to remove floppy controller in some systems. Also handles 2.8MB 3.5" floppy drives. Additional cables and external enclosures may be required.
CCARD2 - $79 **CCARD** - $99 **CCARD4** - $125

ADD ANY FLOPPY TO ANY LAPTOP OR PC WITH A PARALLEL PORT easily and inexpensively. With Backpack, you simply connect the external unit to your parallel printer port (do not lose printer function), install software, and away you go! No expansion boxes needed for laptops, and no slots required. Too easy to be true, but it is. Want a 2.8mb/1.4mb/720K floppy on your MinisPort? Want to add a 1.2 meg to your laptop? Want to add an additional drive to your desktop? Plug it in and go. 2.8MB 3.5" version will read and write 2.8 megabyte format, 1.4 meg format and 720K format. 1.2 meg version will read 1.2 meg and 360K floppies, and write 1.2 meg format.

BPACK2.8	$319	2.8 Mb version
BPACK1.4	$269	1.4Meg/720K version
BPACK1.2	$269	1.2/360K version
BPACK360	$269	360K only version

8-BIT/Z100

We carry a full line of replacement boards, parts and power supplies for the H/Z89/90 and Z100. We also have some H8 boards available. We continue to fully support and carry a full line of hardware and software products for the H8/H89/90 and Z100 computers including almost the full line of Software Toolworks items. Other items we carry include diskettes of all types, printers, modems, hard drives, etc.

OTHER STUFF

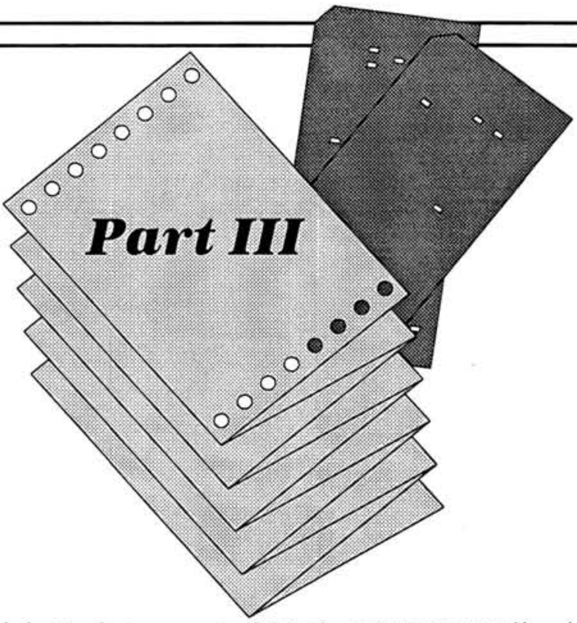
Quikdata also carries BIOS ROM upgrades for most H/Z PC/XT/AT computers, spike protection filters, backup power supplies, tape backup units, modems, printers, cables and ribbons, disk drives and diskettes of all types, external hard drive and floppy drive enclosures, cables and connectors, laptop batteries, CMOS batteries, video monitors and video cards, memory cards, memory chips and ICs, joysticks, accessory cards, serial and bus mouse, a variety of useful and most popular software and much more! **Need a PC/XT/AT computer?** Tell us what you want and we will quote you a price on one of our custom assembled QD computers made up to your exact specifications! Price is lower than you think for a high quality system.

Call or write in to place your order, inquire about any products, or request our **new** free no obligation catalog. VISA and Master Card accepted, pick up 2% S&H. We also ship UPS COD and accept purchase orders to rated firms (add 5% to all items for POs). All orders under $100 add $4 S&H. Phone hours: 9AM-4:30PM Mon-Thu, 9AM-3PM Friday. Visit our bulletin board: (414) 452-4345. FAX: (414) 452-4344.

QUIKDATA, INC.
2618 PENN CIRCLE
SHEBOYGAN, WI 53081-4250
(414) 452-4172

Reader Service #193

Do You Want to Write Another Program?



Gil Hoellerich
2617 Country Way
Fayetteville, AR 72703

Welcome to the third and final article in a series on BASIC programming, the first of which appeared in the January 1991 issue of REMark. Since you have returned, I will assume that you are definitely interested in learning more about programming, in general, or about the BASIC language syntax.

In this article, I plan to do four things:

1. Show you how to make the label programs of the previous articles more practical by using data files.
2. Give some references for continuing your programming education.
3. Give a few clues as to what it is like using a compiler (such as QuickBASIC or TurboBASIC).
4. Show you that the fundamentals are easily applied in programming within dBASE III or IV.

In the last article, we coded a BASIC program which would print the label for the name, address, city, state and zip which you entered from the keyboard. There also was some homework! I hope all of you were able to write the code which would make sure that a Y or N had been entered rather than some character other than Y or y. Adding these two lines of code will do that:

```
142 IF ANS$="N" OR ANS$="n" THEN 150
145 PRINT"You did not enter a Y or N!":
        GOTO 130
```

There are several ways to prepare the code to accomplish this task; so if your code is different but worked, don't fret! Congratulations!

While this program of Part II showed the BASIC syntax, you won't want to enter all the names and addresses each Christmas! Right? You could use a typewriter for that purpose!

So now we will store the list semi-permanently on a floppy disk. I want to stress semi-permanently because if the disk is not handled properly or deteriorates with age, the information could be lost. Having a second copy is a good practice. You can also store this on your hard disk, but in that case, the second copy on a floppy is even more important.

Anything placed on a disk (floppy or hard) must be in a file! So we will place our list in a file on a disk. We have already discussed how to place a BASIC program in a program file, with the SAVE command. Now we want to discuss data files.

BASIC syntax has two types of data files: sequential and random (sometimes called direct access). Now we will have three kinds of files to remember: sequential data, random data, and program.

Data files have two parts: records, and fields. Each different label will be stored as a record; name, street address, and city-state-zip will be stored as three fields in that record. Records are numbered; fields have names.

Let's start with two major tasks:

1. To place labels in the file (LABEL3.DAT).
2. To print labels in the file.
3. To close files and exit program.

Since files on disks should always be closed before leaving the program, we have added the third task.

We will need to find out how many records (labels) are in the file (label3.dat) when we begin. There are several ways this can be done. We will use a random data file to store the labels and, to illustrate the use of sequential data files, we will store the number of records (labels) in the sequential file (count.dat).

So our list of tasks now looks like this:

1. Find out how many labels there are (check COUNT.DAT).
2. Find out what the user wishes to do.
   a. Place labels in LABEL3.DAT.
   b. Print labels in LABEL3.DAT.
   c. Exit the program.
3. Peform 2a, 2b, or 2c, whichever is selected.
4. After 2a or 2b, return to 2.

To begin the process, we will use a small BASIC program ONCE! This will place the count of 0 in the COUNT.DAT file whenever it is run. So if you run this program after you have placed labels in label3.dat, you will not be able to print those labels, since the count will be set to 0!!!

This short program, let's call it BEGIN.BAS, is:

```
10 OPEN "O", #1, "COUNT.DAT"
20 PRINT #1, 0
30 CLOSE
40 END
```

Line 10 will prepare (OPEN) this sequential file (COUNT.DAT) on your disk for output ("O") from the computer and assign a buffer (#1) to this communication. Line 20 will place the number 0 in the file via buffer 1. Line 30 will close this file.

After placing this program in your computer and running it, you can use the `SYSTEM` command and return to the DOS prompt. Then you can use the command `TYPE COUNT.DAT` and you should see the 0 you placed there.

Now for the main program, which we can call LABEL3.BAS. The .BAS extension is stressed to show the difference from LABEL3.DAT, which is the data file to be used with LABEL3.BAS program. Using LABEL3 in both cases, helps remind the user which data files are used by which BASIC program.

Lines 20 to 40 is comparable to the small program we used to start the process, except notice the "I" for input to the computer from the file on the disk, and assigning the value in the file to variable N.

```
10 CLS
20 OPEN "I", #1, "COUNT.DAT"
30 INPUT #1, N
40 CLOSE
```

At the end of line 40, COUNT.DAT is closed and the variable N has the value of the number of records (labels) stored in LABEL3.DAT file.

Line 50 opens the random file LABEL3.DAT and sets record length at 80 characters (bytes); line 60 gives the name and length of the three fields: NAM$ length 26 characters, etc. (**Caution:** Don't improve the variable name, NAM$, by changing it to NAME$; this will give you an error message which will be hard to understand!!)

Using a random file allows us to both input from the file and output to the file without closing and re-opening the file.

```
50 OPEN "R", #2, "LABEL3.DAT",80
60 FIELD #2, 26 AS NAM$, 26 AS ADDR$, 26
AS L.ADDR$
```

The code in lines 70 to 180 should appear familiar, since we used the output keyword PRINT, input keyword INPUT, and the IF-THEN-ELSE structure in previous articles. They perform the task of learning the user's choices. If necessary, re-read the previous articles to refresh your memory. (**Caution:** When entering this program into your computer, don't confuse the ; with : .)

```
70 PRINT "What do you want to do?"
80 PRINT "  Enter 1 for adding names and
addresses"
90 PRINT "  Enter 2 for printing labels."
100 PRINT " Etner x to exit this program."
110 INPUT ANS$
120 IF ANS$="x" OR ANS$="X" THEN 340
130 IF ANS$="1" OR ANS$="2" THEN 150
140 CLS: PRINT "You did NOT enter a valid
choice.":CLS:GOTO 70
150 IF ANS$="2" THEN 260
160 INPUT "Enter name to be entered";
THE.NAME$
170 INPUT "Enter street address"; STR.ADDR$
180LINE INPUT "Enter city, state, and
zip"; CITY.STATE$
```

Notice that the names of the variables associated with the INPUT statements are different from the field names (for example: THE.NAME$ in INPUT and NAM$ as field name.

Lines 190 to 210 take the values of the INPUT variables and place them in the appropriate field of the record in the buffer #2; LSET keyword left-justifies the contents of variables within the field.

```
190 LSET NAM$=THE.NAME$
200 LSET ADDR$=STR.ADDR$
210 LSET L.ADDR$=CITY.STATE$
```

Line 220 places the contents of the buffer into the N+1 record, which in the first case is record 1. Line 230 increments the value in variable N to prepare to use record 2 the next time. Line 250 allows further entries; if no other entries are desired, the user is given the original choices again.

```
220 PUT #2, N+1
230 N=N+1
240 INPUT "Do you want to add other names?
";ANS2$
250 IF ANS2$="Y" OR ANS2$="y" THEN 160
ELSE 70
```

If the print label option was chosen by the user (line 150), the program would go to line 260. Line 270 retrieves the Y record (label) from LABEL3.DAT file; this will be record numbered 1.

Lines 280 to 310 print the labels, just as they did in previous programs. Since these lines are inside the loop (lines 260 and 320), the operation will continue with the second record in the file, since the value of Y has changed to 2. Each time the loop is performed, the value of Y is increased by 1 until the value of Y reaches the value of N, which is the number of the last record in the file.

```
260 FOR Y= 1 TO N
270 GET #2,Y
280 LPRINT NAM$
290 LPRINT ADDR$
300 LPRINT L.ADDR$
310 LPRINT:LPRINT:LPRINT
320 NEXT Y
```

Note that this is the same variable (N) we use when adding new labels. So you can decide to print labels after entering new labels. After printing is complete, line 330 returns the user to the choices given earlier in line 70.

```
330 GOTO 70
```

If the exit program option was chosen in line 120, the program jumps to 340. Since we may have added records in the program, we need to write the number of records stored in variable N to the sequential file COUNT.DAT. Lines 340 and 350 do that and line 360 closes all open files. Lines 370 to 390 should be familiar from previous articles.

```
340 OPEN "O", #1, "COUNT.DAT"
350 PRINT #1, N
360 CLOSE
370 CLS
380 SYSTEM
390 END
```

This program prints one label in each row. Sometimes you may wish to print 2 or 3 labels in a row. For those who want a challenge in BASIC programming, change our program to do 3 labels. You have the necessary knowledge with one exception.

You will need a parameter for the LPRINT command: TAB(). This allows you to begin printing in a specific column. For example:

```
LPRINT TAB(20) "Gil Hoellerich"
```

will start the printing of my name in column 20. **Hint:** If you are printing 3 labels in a row, you will want to retrieve 3 records before beginning. This will necessitate three different variables for names, three for street address, etc.

Remember that you will want to print the three names on one line, the three street addresses on the next line, and the city, state, zip on the last line! You also need to know that to print two or more variables or strings on one line, you need to connect them with a semi-colon. For example:

```
PRINT "This is";"fun."
```

instead of

```
PRINT "This is"
PRINT "fun."
```

The latter will put them on two lines.

As mentioned earlier, the purpose of these articles was to show you what BASIC programming is and hopefully to encourage you to continue to learn programming. If you are saying "I am interested, but where do I go from here?", you can, of course, refer to the Operational Manual on BASIC provided with your computer. However, most people need a book that is a tutorial.

I have used Shelley-Cashman's book on BASIC programming as a textbook, which is very helpful to beginners. Another book I have used is *BASIC Fundamentals and Style* by James S. Quasney, et al. This is more advanced, but is one which you will need to explore files more. If you find the challenge I listed above too much without further help, check on these books.

Now that I have fullfilled goals 1 and 2, let's proceed to 3. According to some information that I have read, Microsoft plans to provide QuickBASIC with version 5.0 of MS-DOS. Also some articles seem to imply that windows will be modified to allow the use of a form of QuickBASIC syntax.

While QuickBASIC is compiled BASIC instead of interpreted BASIC, like GWBASIC, the syntax is very similar; I ran this program in QuickBASIC 4.5 without any changes.

While the code is similar, the QuickBASIC compiler is somewhat different from using the GWBASIC interpreter. There are a number of good books and articles on learning to use the QuickBASIC compiler. I have found *Using QuickBASIC 4* by Feldman and Rugg very helpful. One of the major differences is the optional use of line numbers; if you wish to eliminate line numbers, you will need to learn to use labels. This should not be difficult.

Now to goal 4. Some of the concepts about programming you have learned are also transferrable. You have learned about input commands, output commands, use of variables, and loops. Other languages also have these, including C, one of the most popular ones.

dBASE III+ has an internal interpreter; dBASE IV has an internal compiler. While the languages are different from BASIC, they are very similar. I believe that you will find the transition from BASIC to dBASE programming an easy one.

I will refer to parts of Cool's dBASE, Part 8 article in the February 1991 issue of REMark. Line 5 of Listing 1 is:

```
@ 4,28 say "MONTHLY REPORT"
```

This is the dBASE output statement and similar to BASIC.

```
PRINT "MONTHLY REPORT"
```

instead of the keyword PRINT, dBASE uses the keyword SAY. Also the dBASE command includes the location of row 4 and column 28. BASIC does have a *LOCATE* command. So the equivalent to this dBASE command in BASIC would be:

```
Locate(4,28):PRINT "MONTHLY REPORT"
```

Lines 9 and 10 of Listing 1 are dBASE input statements:

```
@ 10,10 SAY "Enter start date: " get
START_DT
READ
```

The equivalent in BASIC would be:

```
LOCATE(10,10):INPUT "Enter start date: ";
START_DT
```

Since you have the BASIC interpreter with your computer, start there if you would like to learn to program. Much is transferrable to other languages. Happy programming! ✻

Henry Fale
Quikdata, Inc.
2618 Penn Circle
Sheboygan, WI 53081

Converting Z-386/16 RAM From 1MB to 4MB

The Z-386/16 was Zenith Data Systems' first entry into the 386 computer line. To make their memory fast they designed special memory boards for their computers to operate at "0" wait states. It is a true 32-bit memory card and the cards are quite unique. So unique, in fact, that most third party memory cards will not work in that system. That means you're pretty much stuck with the Zenith Data Systems cards.

Unfortunately, their 4 meg cards which once retailed for $3000, later to be dropped to $2000 are not available anymore. And with the new software being used such as Windows 3.0, memory is a must. One or two megabytes just won't do the trick anymore. Thus we are finding that many 386/16 owners are wanting to add more memory to their systems as most only have a one meg or perhaps two one meg boards. Well, your Z-505 one meg board can be modified to a true 4 meg board, so you're in luck if you have some time and patience.

The subject of this article is how to upgrade the Z-505 1MB RAM board to the Z-515 4MB RAM board. The boards are identical. There are two sets of DIP socket holes on the board for each RAM chip; one for the 16-pin 256K DRAM set and one for the 18-pin 1 Megabyte DRAM set.

In a nutshell, the modification goes like this. You must remove the 36 16-pin sockets, install the 36, 18-pin sockets and change the 256K DRAM chips for 1 meg DRAM chips. Two PALs are also changed and some resistors are moved. It is not a difficult modification, it just takes time and patience. It's not a job for those not experienced in soldering and desoldering, and although I would recommend a good (Pace) vacuum pump desoldering station for doing a job like this, it is not essential. Just keep in mind that if you mess it up, you'll lose the board and end up with nothing.

I assume absolutely no responsibility for what is presented here. I attempted to be accurate and as far as I know, it is accurate.

Be certain to observe anti-static precautions when working on this board. If you don't know what that means, you probably shouldn't be working on the board in the first place. I always keep the humidity high enough to prevent static discharges. Others use grounded wrist straps, a grounded work bench, etc.

What you will need to obtain: 36 high quality 18-pin DIP IC sockets, a 444-529 PAL, a 444-530 PAL, and 36 M1000-10 1 meg x 1 80ns or 100ns DRAM chips.

First of all, the 1MB memory chips are larger than the 256K chips that were used in the Z-505 board. The 256K chip is a 16-pin DIP while the 1 meg chip is an 18-pin DIP. Thus all the 16-pin sockets must be removed. Then the new sockets are added in the other set of socket holes. The board has two sets of holes for the two sockets. But to get at the 18-pin holes the 16-pin sockets must be removed. That's 36 sockets, for 576 points to desolder. For those who don't have a desoldering station, there are other ways. Several of these have been presented to me by some of our customers who have tried this upgrade.

You can gently pry the plastic IC memory socket housing off of the pins. One guy used a screwdriver and was successful. If doing this, you must be very careful not to damage any traces. One customer wrote that he damaged traces by doing this so he devised another way. He used a channel lock pliers (vice grips) to gently rock the housing back and forth until it came off. He cautioned to be careful again not to hit any traces with the pliers which would damage them. After this is done, you can apply heat to each pin on the solder side of the board, and remove the pin. It doesn't have to be free of solder as the other set of holes are used anyway. You just need to get the old socket out of the way. One customer said he got tired of heating each pin and removing it, so he finally used a small pair of diagonal cutters to cut them off at board level. Once they are all out of the way, check to see if the 18-pin holes are clear and free of solder. You will probably have to remove the solder in the holes in order to install the new IC sockets. This can be done with a desoldering station, or by heating each one and using solder wick or a solder sucker pump or bulb.

Now add the new 18-pin DIP IC sockets and solder them in. That will be 648 places to solder. Use a standard low wattage solder iron, 40 watts or so, not a 120 watt soldering gun or blow torch or anything like that. As mentioned earlier, don't go out and buy the cheap tin single contact sockets - get something good. After all, you want something reliable and don't want to have to tear them all out and start over again.

Now that the hard part is done. The rest of it is replacing PALs and moving resistors.

- At location U464 remove PAL 444-558 and replace with 444-529.
- At location U470 remove PAL 444-559 and replace with 444-530.
- Remove 47-ohm 1/4W resistors (yel-

low-violet-black) from R405 and R406.

- Remove 33-ohm 1/4W resistor (orange-orange-black) from R410.
- Install 47-ohm 1/4W resistors at R404 and R407.
- Install a 33-ohm 1/4W resistor at R414.

Now go and install the 36, 1-megabyte DRAMs in the 18-pin sockets. Be sure you observe pin 1 positioning. And be certain that no pins get bent out or under the chip.

That should do the trick. If you did everything properly, you should now have a 4MB RAM board from your 1MB RAM board. Total cost of the upgrade at today's prices should be about $250. So if you're good at desoldering and soldering and you want to tackle it, you'll get a lot for your money.

I assume (or hope) that you either already had a Z-515 card so you have the switch setting documentation, or that your Z-505 manual contains the necessary information. In case you don't, the factory default settings for SW401 for the four-meg board are as follows: Sections 0 and 2 are OFF, the rest are ON. (The original one-meg board defaults were all sections ON.)

For your information, we sell the PAL set as Z-515 for $21. We sell the 1 meg X 1 dynamic RAM chips, 100ns for $5.75 each, or the 80ns ones for $5.95 each. So you're better off the spend the extra 20 cents per chip and get the 80ns ones.

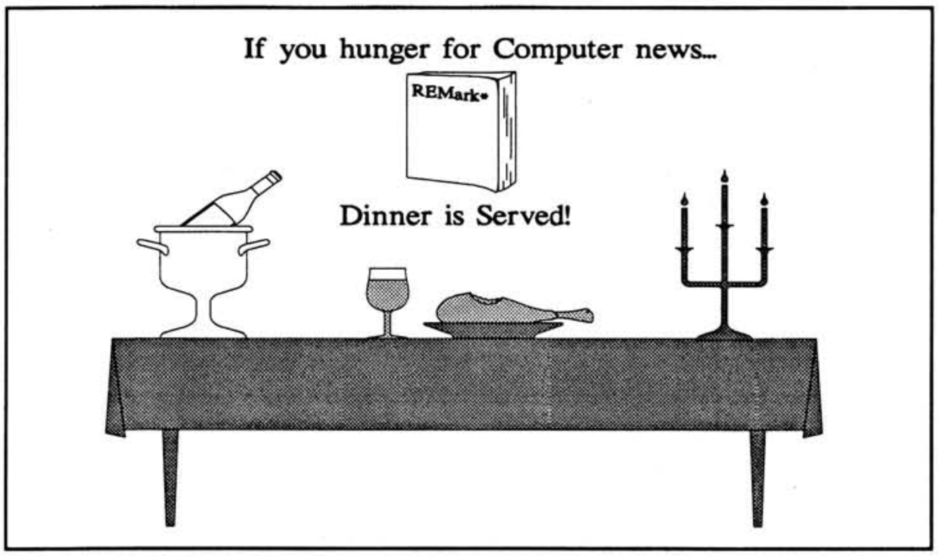
Take your time, and you'll be rewarded with 3 extra megs of RAM per board for a reasonable cost. ✻

Continued from Page 9

with a little muscle, this is the ideal program. At $229 list (considerably less at sheet prices), LetterPerfect is a great alternative because it has WordPerfect file compatibility and most of its features for about half the price. If you're intimidated by function keys you'll appreciate the pull-down menu and mouse support, as well as the logical quick key commands.

LetterPerfect is one way you can still experience most of the power and essential features of WordPerfect without a hard drive. Floppies aren't the ideal way to run the program, but they'll do a respectable job if you don't mind a disk swap or two.

LetterPerfect
WordPerfect Corporation
1555 N. Technology Way
Orem, UT 84057
801-321-4566 ✻



Continued from Page 28

is about all its good for. Sybex book number F440.

QuikData, now a book but a company run by Henry Fale, a truly great source for all the wonderful old software you thought was gone for good. Call Henry at (414) 452-4172, or try his bulletin board at (414) 452 4345 (character width of 10, one start bit, eight data bits and one stop bit, XMODEM and YMODEM supported, 300, 1200, 2400 or 9600 buad auto recognition.) At different times he's had various versions of FORTRAN at giveaway prices.

```
C       PROGRAM DISKREAD.FOR
C
C       CALLED DATA ON THE CURRENT DRIVE, THEN PRINTS FILE ON CON-
C        SOLE.
C       LUN FOR DISK I/O IN THIS EXAMPLE IS 6. THE CONSOLE IS *
C
        REAL X
        INTEGER I, N
C
C       SAME LUN AS PREVIOUSLY
C
        OPEN(6, FILE = 'DATA', STATUS = 'OLD')
        N = 50
        X = 1.0E-4 / 3.0
        DO 10 I = 1,N
           X = X / 10.0
           READ(6,101)I,X
           WRITE(*,101)I,X
10      CONTINUE
        STOP
101     FORMAT(1X,I4,1PE15.6)
        END
```

Listing 4
This program reads the data in the exisitng (old) file DATA from LUN 6 and prints it on the console, LUN *. ✻

Classified Ads

FOR SALE "Z-100 LOW PROFILE", Dual Disk, Z-DOS, Manuals, Mounds of Software, $350.00. (813) 920-7544.

Enable Revisited

The Word Processor

Part 2

George P. Elwood
1670 N. Laddie Ct.
Beavercreek, OH. 45432

Welcome back Enable Revisited. This article will focus on the word processor and the changes that have been made in Enable OA. This article is not a how to do it but rather what it can do. I will discuss some things that I have found in using Enable. Hopefully, this short article will give you the desire to look into Enable.

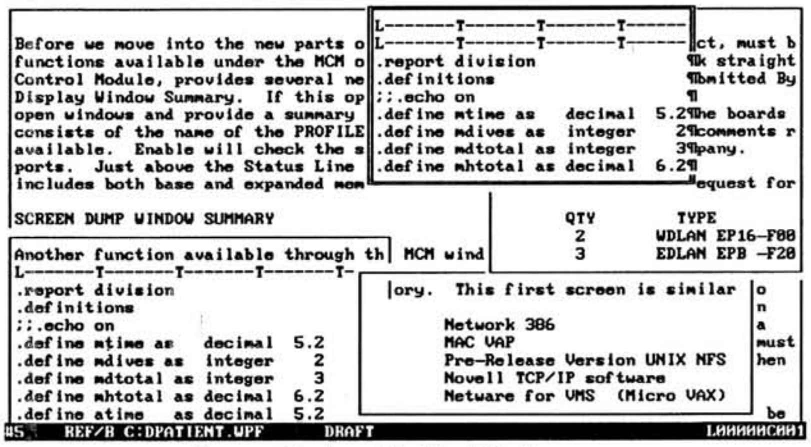
The word processor in Enable has had several enhancements. (In an up-coming article on Enable 4.0, you will see even more of these enhancements.) To access the word processor, press (U)se System, (W)ord Processing, and (C)reate. Enable will now open a block which you use to type in the name of the file to create. Enable OA has the capability to accept up to a 64-character entry in this block. These longer blocks permit you to specify the exact-DOS sub-directory the file is to be created in. If the file will be created and saved in another directory, use the DOS conventions for directing the file to the correct location. If you want to save the file in the EN300 directory, simply insert the name of the file. Like earlier versions of Enable, you do not have to insert an extension. Enable will insert the ".WPF" extension for all word processing files. If you wish to insert your own extension, Enable will not automatically search for and display the file unless you know the name or use wild cards for the search.

When you type in the name of the new file and press <Enter>, Enable will open a word processing window. Enable will prompt you to change the default rulers. The cursor is located on the ruler and by moving the cursor and typing "T" for TABs, or "L" for left margin or "R" for right margin. The maximum line length is 160 characters. When you are happy with the basic layout, press <Enter>. Enable will now display the basic word processing input screen.

The screen size is 78 characters wide by 22 lines. The window is outlined on three sides by double lines. If you create another window, the lines will change color if you have a color monitor. The window that you are in is always outlined with double lines. Inactive windows are displayed with single lines. Like the earlier version of Enable, you may have up to eight windows open at one time.

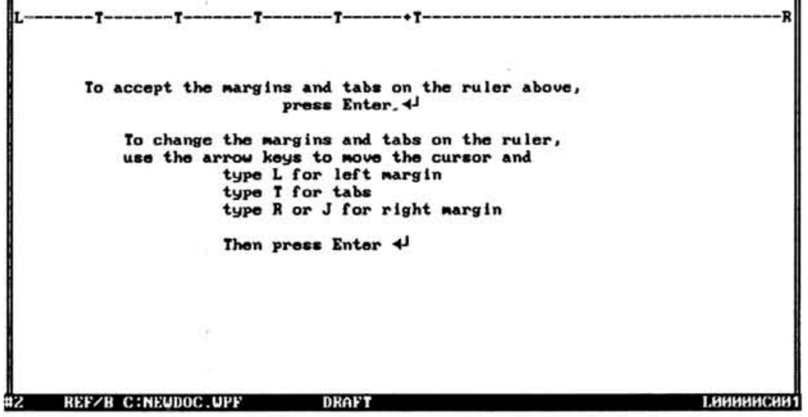
On the top of the screen is the Document Title block. This is an optional area in Enable. If you use this area you must make sure that the print options include printing the title page. This area is normally not printed. If you wish to delete it, put the cursor within the area or on one of the lines and press ALT/F3 (Enable's delete line command). You may now type your document.

Like earlier versions of Enable, every time you press the <Enter>, a paragraph marker will be displayed on the right side of the screen. You may add as many rulers as necessary to the document. To add a ruler you can use the Top Line menu by pressing F10 to display it and then selecting Layout and Insert Ruler. The "Expert" keys are ALT/F6. A new ruler will be inserted on the line were the cursor was located. In addition to indicating the right,



```
                                          L-------T-------T-------T------
Before we move into the new parts o L-------T-------T-------T------ ct, must b
functions available under the MCM o .report division              ¶k straight
Control Module, provides several ne .definitions                  ¶bmitted By
Display Window Summary.  If this op ;;.echo on                    ¶
open windows and provide a summary  .define mtime as   decimal  5.2¶he boards
consists of the name of the PROFILE .define mdives as  integer    2¶comments r
available.  Enable will check the s .define mdtotal as integer    3¶pany.
ports.  Just above the Status Line  .define mhtotal as decimal  6.2¶
includes both base and expanded mem                                equest for

SCREEN DUMP WINDOW SUMMARY                          QTY        TYPE
                                                     2         WDLAN EP16-F08
Another function available through th MCM wind       3         EDLAN EPB -F20
L-------T-------T-------T-------T-
.report division                          ory.  This first screen is similar  o
.definitions                                                                  n
;;.echo on                                      Network 386                   a
.define mtime as   decimal  5.2                 MAC VAP                       must
.define mdives as  integer    2                 Pre-Release Version UNIX NFS  hen
.define mdtotal as integer    3                 Novell TCP/IP software
.define mhtotal as decimal  6.2                 Netware for VMS  (Micro VAX)
.define atime   as decimal  5.2                                               be
#5    REF/B C:DPATIENT.WPF        DRAFT                                LAAAAACAA1
```

Screen Display with Five Windows open



```
L-------T-------T-------T-------T------+T-------------------------------------R


       To accept the margins and tabs on the ruler above,
                          press Enter.↵

           To change the margins and tabs on the ruler,
           use the arrow keys to move the cursor and
                     type L for left margin
                     type T for tabs
                     type R or J for right margin

                     Then press Enter ↵

#2    REF/B C:NEWDOC.WPF          DRAFT                                LAAAAACAA1
```

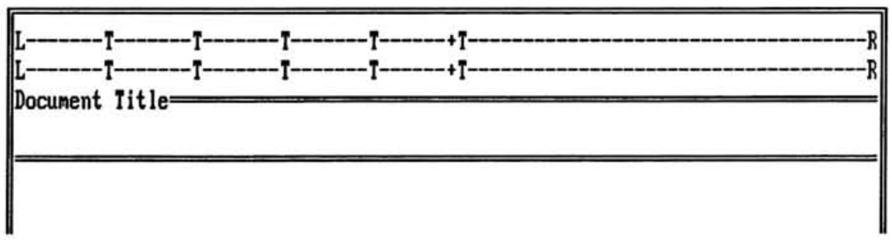
Screen Display New Word Processing Document left, and TAB setting, you may also use the "+", "C", "X", "N" and CARETS.

These additional characters, which can be added to the ruler, will make Enable even more powerful. The "+" will establish a center point on the line. This will permit you to center text based on this point by simply pressing ALT/F4. The text that is on the line will be centered on the "+".

If you work with multiple columns, using the "C" will make your work easy. To establish the column, you must insert a (R)ight or (J)ustified right margin. Then move to the right and insert a "C" to mark the location of the left margin of the next column. You may add as many as necessary but three should be maximum for appearance. As you type the text, the first column will be filled. To see what the output will look like, press "F9 O D" to move the word processor to the final mode. The text will now snake around to fill the columns. You may have to add lines to the top of the second and third columns to have the desired start locations. Note in the example that the second column starts above the first column. This is an example of where you would have to add some line to make it come out correctly.

Note in the multiple column example the amount of space between the letters. If your printer supports microjustification, this will close up the spaces and make the document look better. There is another letter that can be added to the ruler that may help with the spacing between the words. Enable OA supports a user defined hyphenation zone. To use this function, insert the letter "Z" about five spaces before the (R)ight or (J)ustify right margin marker on the ruler line. Enable will now hyphenate a word if it can break any place between the "Z" and the right margin indicator. This would help make the line even and the justification look better.

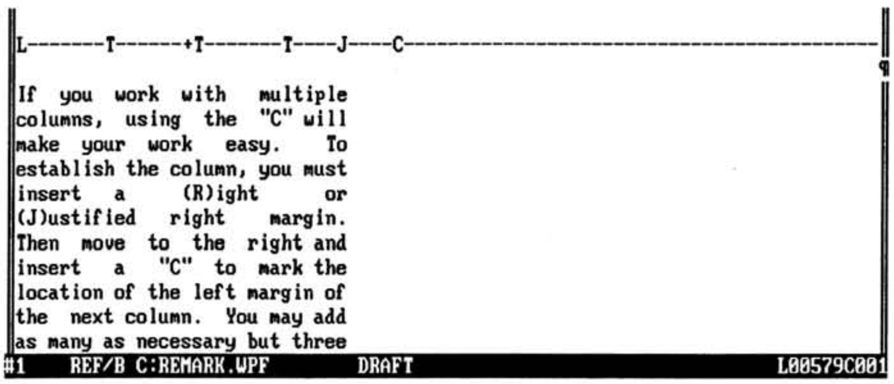
The next set of letters that can be added to the ruler are the letters "X" or "N". The "X" will define alphanumeric tabs and the letter "N" will define numeric tabs. You may insert a decimal in the ruler line to align decimals in numbers at points established using the numeric tabs.



```
L-------T-------T-------T-------T------+T---------------------------------R
L-------T-------T-------T-------T------+T---------------------------------R
Document Title
```

Document Type Page One

Using tabs in Enable OA will define a protected zone. If you use tabs, Enable will display the protected zone by using small blocks. You may not insert characters in this zone.



```
L-------T------+T-------T----J----C------------------------------------------

If  you  work  with   multiple
columns,  using  the  "C" will
make  your  work   easy.    To
establish the column, you must
insert   a      (R)ight     or
(J)ustified   right    margin.
Then  move  to  the  right and
insert   a   "C"  to  mark the
location of the left margin of
the  next column.  You may add
as many as necessary but three
#1    REF/B C:REMARK.WPF        DRAFT                             L00579C001
```

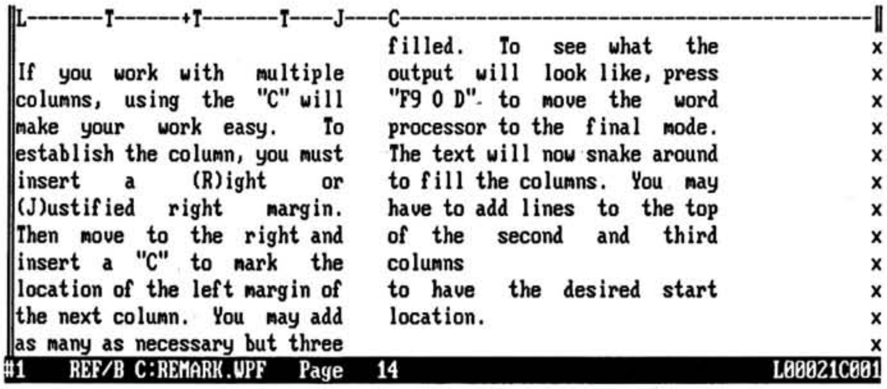
Multiple Column Format - Draft

The dictionary has been improved to where it now recognizes hyphenated words and possessives. In addition to the included dictionary, Enable Software has another dictionary available on their electronic bulletin board. This can be downloaded and put into the EN300 directory. It can then be selected during the spell check options. The dictionary is accessible using the Top Line Menu by pressing F10 or using the expert command "F9 E C".

Also included with the package is the Proximity/Merriam Websters Thesaurus. In version 2.0, this was an add on package. To access the thesaurus, press F10 for the top line menu and then "2" for the dictionary and "2" again. You can also access it by placing the cursor on the word you want to check and pressing "F9 E T". Enable will then display a list of definitions from which you can chose. It will then display the meanings and provide other words. You can move the cursor to the word you wish to select and press <Enter> to move it to the word processing document.

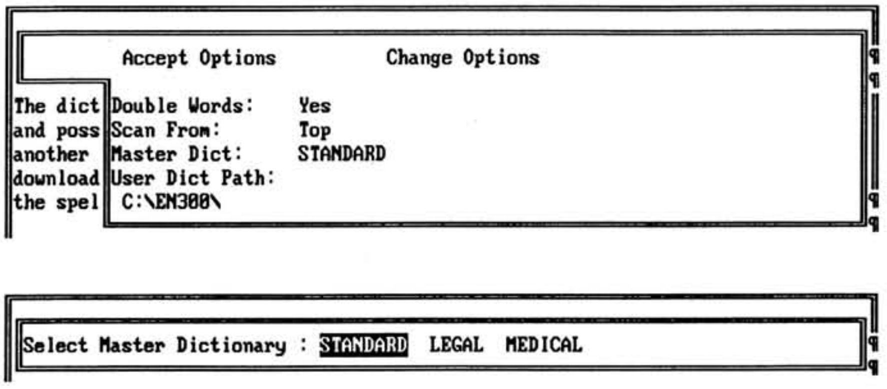
If you use footnotes, Enable OA has increased the power of this function. You can define your separator, symbol, and annotation style. The footnotes can run several pages if necessary. The settings for footnotes are found in the profile under word processing.

Enable's word processor has had the capability to do math within the document. You may now define a columnar block for calculations. This is an improvement over the calculations in a line. To use this function, highlight the block of numbers. Position the cursor in one corner and press "F7". Move the cursor to the oppo-



```
L-------T------+T-------T----J----C----------------------------------------
                                  filled.   To   see  what   the          x
If  you  work  with   multiple    output  will   look like, press          x
columns,  using  the  "C" will    "F9 O D". to  move  the   word          x
make  your   work  easy.    To    processor to the final  mode.           x
establish the column, you must    The text will now snake around          x
insert    a      (R)ight    or    to fill the columns.  You  may          x
(J)ustified   right    margin.    have to add lines  to  the top          x
Then  move  to  the  right and    of  the   second   and   third          x
insert  a  "C"  to  mark   the    columns                                 x
location of the left margin of    to  have   the  desired  start          x
the next column.  You  may add    location.                               x
as many as necessary but three                                            x
#1    REF/B C:REMARK.WPF    Page   14                             L00021C001
```

Multiple Column Format - Final Mode



```
          Accept Options          Change Options

The dict Double Words:     Yes
and poss Scan From:        Top
another  Master Dict:      STANDARD
download User Dict Path:
the spel  C:\EN300\

Select Master Dictionary : STANDARD  LEGAL  MEDICAL
```

Dictionary Menus

site corner and press "ALT/F7" to mark the column block. Now position the cursor where you want the result and press "F9 M +". The result will default with a two place decimal. This can be removed if not needed. Make sure you leave enough space for the result or Enable will over write it with text. This will only add the numbers in the column. If you need other functions you must use the long form or do it in the spreadsheet and copy it to the document using Inter Window copy.

```
3456
5413
6565
729
16,163.00
```

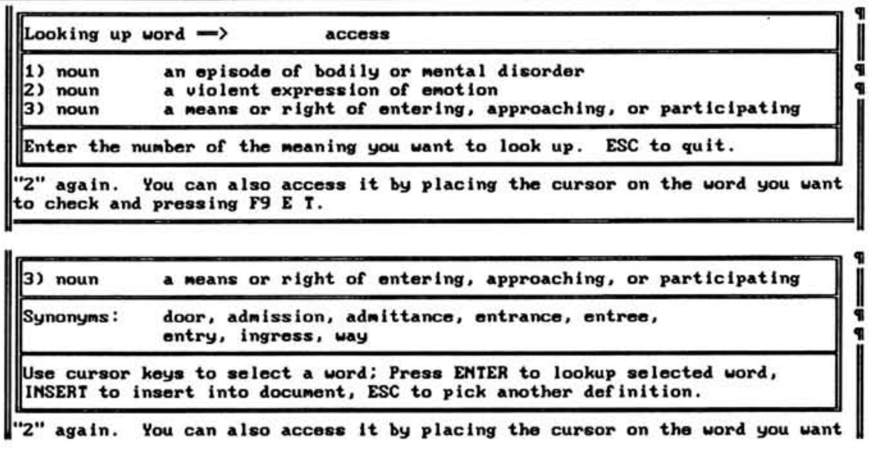
The system date function that was available in Enable is now available anyplace in the document. Before V3.0, the "percent date" would only work in the header or footer. Now it will be printed any place %DATE is inserted, like in this line. This will permit you to insert "percent date" in a letter and have the system date inserted when it is printed.

Enable has always been able to import graphics from the spreadsheet. I used this function extensively in a business plan I developed. I would generate the graphs and use the interwindow copy function to move them into the word processing document. A new capability added to OA is the ability to import Perspective charts and graphs created using MacPaint or PC Paintbrush. (.PCX extensions).

To import the PCX files, they must have been created on with the same system. If you are using an EGA card and monitor to create the files, Enable must be running on the same type system, i.e., EGA. To import these files you must press "F9 Ins D". Enable will open a window and ask for the name of the file. You may use the wild card character and can specify another directory. As an example, if I insert "C:\HG\*.pcx", Enable will display all files in the HG directory ending with .PCX. You may highlight the file and press <Enter>. Enable will now prompt for the type file you wish to import. Select Graphics and the file will be inserted into the document. Note, you must use the %density printer commands to insure the graphic will be printed correctly. You will have to experiment with this command and your printer.

After importing the graphic, you will see a gray area on the screen where it is located. In order to see the graphic, you must put Enable into the graphics mode by using "F10 M 3 4". You may type over the graphic if necessary. If you do so, Enable will make a block the size of the letters in the specified location. (See "Harvard Graphics" illustration.)

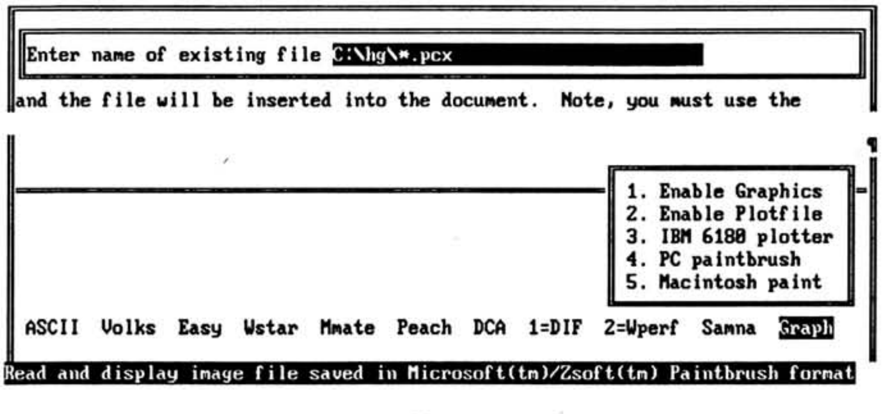
Another feature in Enable is the "%include" statement. This has been in all versions of the product but it has nice capabilities. I write instructional manuals for different software packages, or use to anyway. Rather than having one large 200K file, I break the manual into chapters. The first chapter or front matter is the basic unit. I include the Table of Contents and other related material. At the end of the file I insert the "%include" command to point to the rest of the chapters. I can now run the Table of Contents compile option in this first unit and Enable will check all of the included chapters and add the chapter TOC information to the basic matter. The index function works the same way.



```
Looking up word -->          access

1) noun       an episode of bodily or mental disorder
2) noun       a violent expression of emotion
3) noun       a means or right of entering, approaching, or participating

Enter the number of the meaning you want to look up.  ESC to quit.

"2" again.  You can also access it by placing the cursor on the word you want
to check and pressing F9 E T.

3) noun       a means or right of entering, approaching, or participating

Synonyms:     door, admission, admittance, entrance, entree,
              entry, ingress, way

Use cursor keys to select a word; Press ENTER to lookup selected word,
INSERT to insert into document, ESC to pick another definition.

"2" again.  You can also access it by placing the cursor on the word you want
```

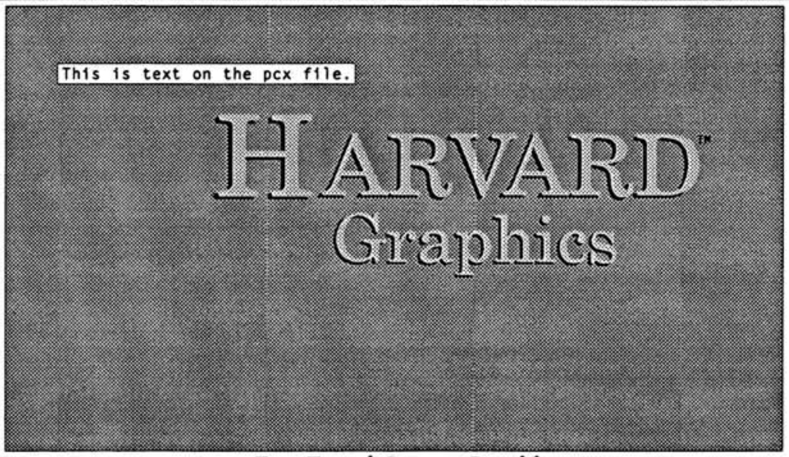
The Enable Thesaurus

One of the functions I use extensively in manual creation is over strike. I use a public domain program called "SNIPPER" for screen captures of ASCII characters. I will then bring these screens into the document using the Interwindow Copy capability of Enable. I will then use over strike to show the highlighted areas on the screen. To use the over strike, position the cursor where you want to start the over strike and press "ALT/O". Enable will display a diamond O on the screen. Next I use the ASCII character 176, which is a light shadow block. To get this character

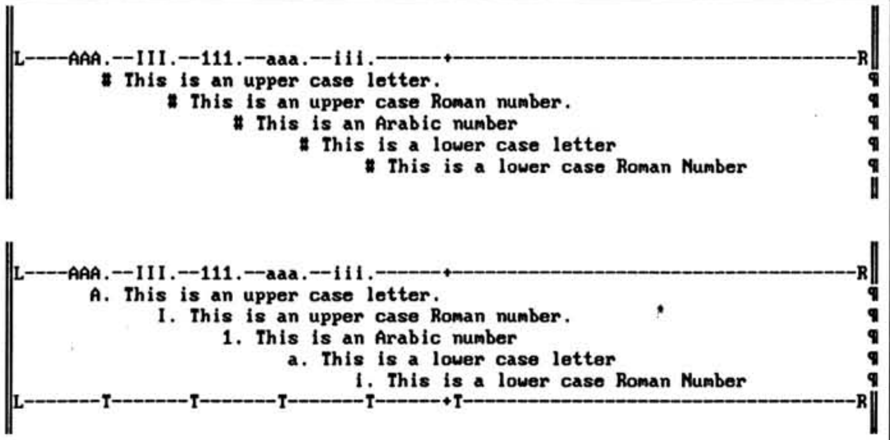


```
Enter name of existing file C:\hg\*.pcx

and the file will be inserted into the document.  Note, you must use the

                                              1. Enable Graphics
                                              2. Enable Plotfile
                                              3. IBM 6180 plotter
                                              4. PC paintbrush
                                              5. Macintosh paint

 ASCII  Volks  Easy  Wstar  Mmate  Peach  DCA  1=DIF  2=Wperf  Samna  Graph
Read and display image file saved in Microsoft(tm)/Zsoft(tm) Paintbrush format
```

Graphics Imports



Text Typed Over a Graphic



```
L----AAA.--III.--111.--aaa.--iii.------+---------------------------------R
       # This is an upper case letter.                                    ¶
              # This is an upper case Roman number.                       ¶
                     # This is an Arabic number                           ¶
                            # This is a lower case letter                 ¶
                                   # This is a lower case Roman Number    ¶


L----AAA.--III.--111.--aaa.--iii.------+---------------------------------R
     A. This is an upper case letter.                                     ¶
            I. This is an upper case Roman number.                        ¶
                   1. This is an Arabic number                            ¶
                          a. This is a lower case letter                  ¶
                                 i. This is a lower case Roman Number     ¶
L-------T-------T-------T-------T------+T--------------------------------R
```

Outline Format and Output

you can turn on Enable graphics characters "F10 L 7" and the set you need, or the expert command "F9 O C" and the character set you need. The light shadow is located in the (G)reek character set. I use this option so much that I found that by pressing ALT and 176 on the key pad, not the numbers across the top of the key board, this character will be generated. This works only on MD-DOS 3.0 and above. When you want to turn off the over strike, press "ALT/O" again. This is how all of the shadowed areas were created in this article.

One of the capabilities that exist within Enable is the outline function. This works in conjunction with the rulers. As you insert a new ruler, you may indicate where you what the outline sequence numbers and the type of numbering you wish to use. As an example, an upper case "I" will generate upper case Roman numerals. A lower case "i" will generate lower case Roman numerals. Upper case "A" or lower case "a" will generate a letter based outline. Using a "1" will generate an Arabic based outline. To generate the outline, all you do is insert a "#" where you want the number. When you print the outline or change the display mode to final, the selected numbering will be output or displayed. Note the period in the Outline Format example. If this is included on the ruler line, Enable will display or print it.

After you have finished the document and are ready to print it, you can specify several new functions in Enable OA. To access the print menu screens, press "F10 P". The first of four print screens will now be displayed.

On this first screen you can keep the default of printing the entire document or specifying the pages to print. A new feature in Enable OA is the selected page option. In earlier versions the pages to print had to be in order. In OA you can specify exactly which pages you want to print. Individual pages are separated by commas and groups of pages are separated with a hyphen.

Another new option is the Portrait or Landscape orientation. This will work ONLY if your printer will support it. Most laser printers support this function and it can be selected here. Within the document you can use the "%landscape" option to select individual pages for this orientation.

Another new option on this page is the "Print in draft mode:". This is NOT the option to select to output the document in a draft mode on a dot matrix printer. This option, if selected, will print EVERYTHING shown on the screen. This includes the rulers, comments, index blocks, etc. This option would be selected if you want to print out the menu report commands which appear in a comment block.

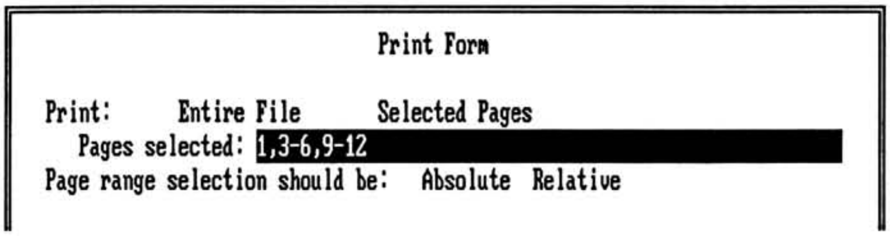
The last new option on this page is the widow/orphan option. If selected (Yes), this option will keep Enable from printing one line on the bottom or top of a page. Enable will automatically move a line to prevent these conditions.

The middle two pages in the print menus remain basically the same. The printer selection menu has changed in that all of the selected printers are not displayed on the bottom of the screen. If you want to see the available printer drivers, press F7 and they will be displayed.

The last screen has several new options. The first of these is the ability to create your own date picture. If you select this option Enable will display a block where you can insert the desired format. Whatever format is selected will be used in your document where the "% date" is shown. Note, there is no space between the percent and date.

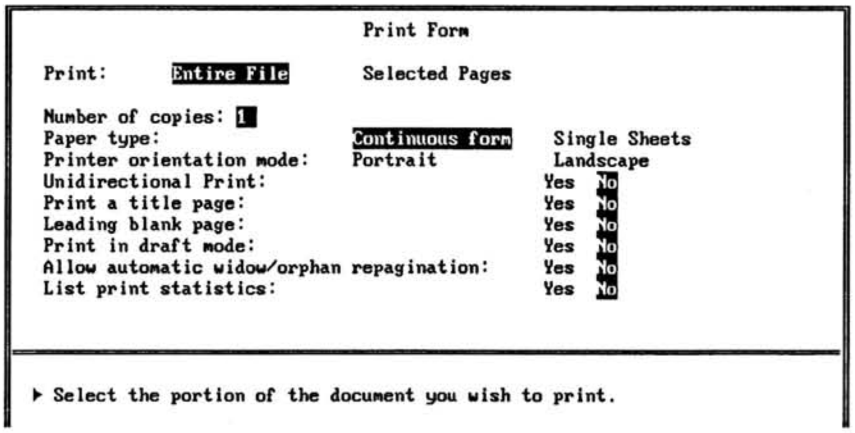
The next new option is microjustification. If your printer supports this function, Enable will output the text with fine variations between the letters. I use this in conjunction with the (J)ustification and auto hyphenation to output very good looking documents.

One of the printer drivers that is available with Enable is Postscript. Using the built-in capability of Enable, I have output



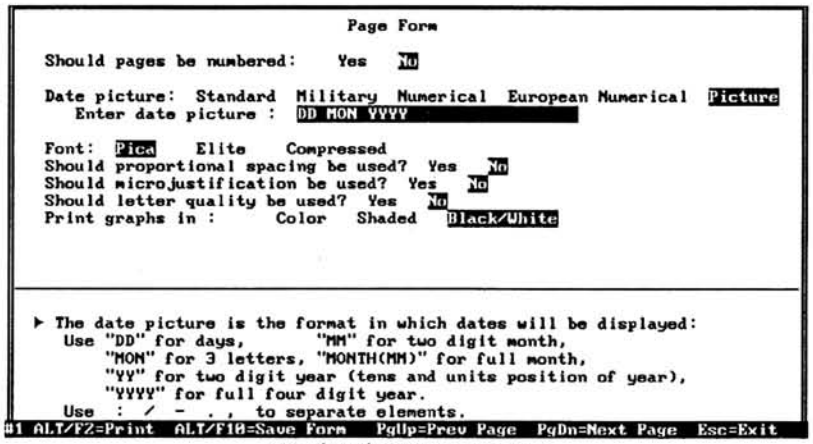
```
                          Print Form

Print:      Entire File       Selected Pages
   Pages selected: 1,3-6,9-12
Page range selection should be:   Absolute  Relative
```

Selecting Pages Option



```
                              Print Form

Print:       [✓ Entire File]         Selected Pages

Number of copies: [✓ 1]
Paper type:                    [✓ Continuous form]   Single Sheets
Printer orientation mode:      Portrait              Landscape
Unidirectional Print:                          Yes [✓ No]
Print a title page:                            Yes [✓ No]
Leading blank page:                            Yes [✓ No]
Print in draft mode:                           Yes [✓ No]
Allow automatic widow/orphan repagination:     Yes [✓ No]
List print statistics:                         Yes [✓ No]

▶ Select the portion of the document you wish to print.
```

First Print Menu Screen



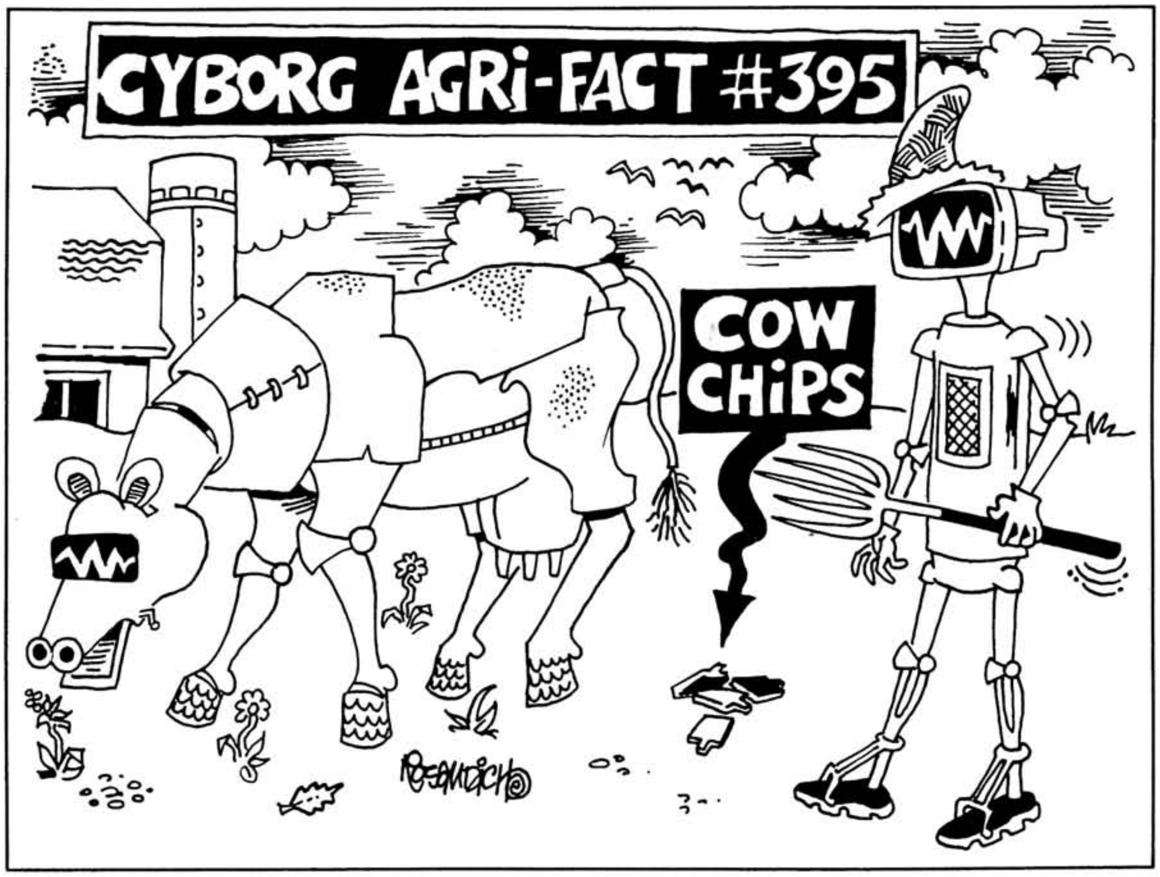
```
                            Page Form

 Should pages be numbered:    Yes [✓ No]

 Date picture:  Standard  Military  Numerical  European Numerical  [✓ Picture]
    Enter date picture :  DD MON YYYY

 Font: [✓ Pica]   Elite    Compressed
 Should proportional spacing be used?  Yes  [✓ No]
 Should microjustification be used?  Yes  [✓ No]
 Should letter quality be used?  Yes  [✓ No]
 Print graphs in :       Color   Shaded   [✓ Black/White]

▶ The date picture is the format in which dates will be displayed:
   Use "DD" for days,          "MM" for two digit month,
       "MON" for 3 letters, "MONTH(MM)" for full month,
       "YY" for two digit year (tens and units position of year),
       "YYYY" for full four digit year.
   Use  :  /  -  . ,  to separate elements.
#1 ALT/F2=Print  ALT/F10=Save Form   PgUp=Prev Page  PgDn=Next Page  Esc=Exit
```

Fouth Print Menu Screen

test pages with four different fonts and type sizes from 6 point to 30 point. The printer specification manual provides instructions on how to develop your own Postscript configuration file for other fonts. You can use the "%control" function to change fonts within a document. You must know the printer codes required for these changes. Enable OA does have what is called "Club Commands". These are unique commands for your printer to enable all of the capabilities to be used. As an example, the Alps 324 has a color capability. You could insert the "%control" above the line you wish to change. But if you wish to change a color of one word, the "%control" will not work. Using the club commands, position the cursor before the word you wish to change and press "F9 P". Enable will prompt for a letter. In the printer specification manual under Alps 324 (P080), the letter "G" will cause the printer to print green. After the word you would have to insert the club command "B" to return the printer to black.

As you can see, Enable has further increased the capabilities of the word processor. I know that you have your favorite word processor but you should consider Enable if you are looking for a change. This has been only one module, and Enable does have three more, database, spreadsheet, and telecommunications. They work together.

The next article will cover the spreadsheet and graphics changes.



The Frugal PC
(Or, Let Shareware Fill Your Hard Disk Drive)

Harold C. Ogg
357 W. Diversey Avenue
Addison, IL 60101-3508

Software is too expensive! Few persons will argue the point, but who can deny that many of those sold-at-gunpoint programs aren't often worth every penny? My Scottish heritage cringes every time I browse through the displays of data bases, word processors, and spreadsheets costing hundreds of dollars and protests, "there must be a better alternative." There is, but as the announcer says, "you gotta know where to shop."

If you are the owner of a Z-148, Z-151, or any later Zenith Data Systems PC-compatible machine, there is a treasure trove of inexpensive (and sometimes free) software waiting for you. As with any product, you need to know how the shareware system works and the names and addresses of the often lesser known sources. For this concept of shareware and public domain software to work for you, knowledge of a few facts is essential. We'll keep the classwork to a minimum and proceed into the intent of this article: stretching your software dollar and fighting inflation drain on your wallet caused by overpriced programs that usually have cheaper (and sometimes better) counterparts for a fraction of the cost of the "name brands."

The Copyright Law and the Shareware Concept

Decades ago, the United States government established the Copyright Office to protect authors' ideas from unauthorized duplication and illegal profiteering by others. Originally targeting printed works, the law has been extended by the advent of high technology to include computer software. You give your implied consent to this law every time you break the seal or the shrinkwrap on the disk(s) of a newly purchased program. The early law protected a book for twenty-seven years (and twenty-seven more with a renewal of the copyright), after which the work was considered "public domain." The coming of the fifty-fifth year meant that anyone could copy the work without paying the original author any more royalties or fees.

At first, courts had a problem with software copyright. The problem was that most judges and juries could not decide the worth of information. A number of computer crime books have been written that consider, for example, the severity of the theft of a disk pack. The classic story is the one of a disgruntled employee who left his company with the original (and only) copy of a disk pack containing numerous company trade secrets and processes. At his trial, not able to place a value on the information contained on the disks, the court found the ex-employee guilty only of a misdemeanor theft of a computer peripheral worth approximately $59.95!

But the situation has been resolved — thankfully — and to the computer users' advantage. "Public domain" has been defined and has become a mutually beneficial haven for the programmer and end user alike. You need to learn just a few rules are in order to make the concept work for you:

"Public domain" does not necessarily mean that you may take unbridled possession of a program or piece of software. On the contrary, while you may generally "copy freely" the program, you may not sell it at a profit, nor may you make alterations without the original author's permission. It is really the same with commercial software. When you "purchase" a program, you do not gain title to it (unless transfer of ownership is specifically given in the license agreement). What you "own" is the medium (floppy disk or tape) upon which the program resides; the program, i.e., the result of intellectual creativity, is still the property of the author or programmer. The same principles generally apply to the concept of shareware, and we'll separate the fine points between the latter and public domain software in the next section.

What you may do with a program tagged as public domain is distribute free copies (or copies for which you charge no more than the cost of the disk). You may also generally use it without charge, within certain restrictions (and possibly time limits) which the author usually places in a READ.ME file on the disk. The idea behind public domain software and shareware is one of marketing and user support: the author gives you a free sample so that you can determine whether the program is right for you. (S)he usually has little money for advertising, so if you like the program, you'll tell your friends. And the cost, typically for the full-fledged or registered version of the program, is almost always a fraction of cost of the brand name counterpart. You get cheap, functional programs and an aspiring programmer makes a sale. It's as simple as all that.

What's the Catch, and How do You Pay?

Keep in mind that the shareware author is a businessperson, and, as such, expects to turn a profit for his/her intellectual efforts. However, since the author's overhead is kept low because the program is being "advertised" on computer bulletin boards, in limited production catalogs, and literally, by word of mouth, you benefit with lower software prices. But, in order to perpetuate the shareware concept, you must "play fair" and abide by the honor system to induce authors to keep writing these low cost, useful programs.

Payment for shareware takes several forms. Most typically, a freely obtained shareware issue of a program is actually a demo version (or older edition) of a larger prototype. The demo contains information for ordering the latest, full fledged, supported version, and many times the demo will include a ready-to-print order blank. Many authors are set up to accept credit cards. Other times, the author requests payment in the form of a donation: the shareware you copied is actually the complete version, and the author asks for your support — usually $10.00 to $20.00 — if you enjoyed the program. For your generosity, you may be sent additional data files, program graphics, or, in the case of shareware games, supplemental puzzle files. How much (and whether) you contribute is between you and your conscience.

Some variants of shareware do exist. Completely free software, sometimes called bannerware, is a part of the public domain and includes programs that authors have written as funded projects, school theses, or simply for bragging rights. You can't beat the cost, and many bannerware programs are of high quality. Another (but somewhat negative) variant is so-called crippleware, a demo version of a larger program that is presented in such a way that the program appears more or less dysfunctional. It may take the form of an accounting program that demonstrates its features for only eleven months of the year (your registration fee buys you December's subroutines) or a graphics program that is timed to run for only, say, three iterations. It is this writer's opinion that crippleware goes contrary to the spirit of the shareware concept and doesn't show the potential buyer the full range of a program's capabilities. Crippleware may go the way of disk-based copy protection and should be avoided.

Sources

Since the shareware bonanza doesn't often hit the big national publications with a media blitz, it sometimes goes undetected. The rest of this article will reveal sources for catalogs and mailing lists so that *REMark* readers can tap this economical genre. Unless otherwise stated, the programs are for the Z-151/159, Z-248, Z-386 and IBM-PC/XT/AT compatible series of computers.

Flea Markets

Those of us fortunate to live in or near big cities have our choices of weekend exhibitions at which to go bargain shopping. On the tables are generic disks full of $1.00-$5.00 programs. The vendors usually distribute photocopied lists of their wares, and there is much redundancy from one seller to another. Typically, these programs are the "also rans" of software and are untested and unsupported. However, the fun is in the discovery and sometimes a sleeper will appear. Use flea market shareware as a benchmark to what's available on the public domain market as a whole, and for experimentation.

Newsletters and Periodicals

If it be true that the initials "IBM" actually mean "Information Becomes Money," then the few dollars' subscription fee for one of the several good shareware magazines now published is money well spent. You not only get news on the most recently authored public domain and shareware programs, but also a variety of money-saving special offers and detailed reviews (and sometimes recommendations) about the programs themselves.

Shareware News[fn1] is published by one of the largest shareware vendors, PC-SIG (PC Special Interest Group). The News is actually an advance sheet for the Group's larger publication, *Shareware Magazine*,[fn2] and gives brief descriptions of new programs with PC-SIG library numbers and ordering information. Hardware requirements are given for each program, as are the royalties or registration fees (if any) expected for the standard or latest program releases. *Shareware Magazine* has been around for several years, and contains material worth much in excess of its subscription price. In addition to detailed reviews of new PC-SIG software titles, the publication contains classified advertisements, software deals, program ratings, and useful articles such as "Make Money Writing Shareware." Further, it categorizes programs and makes available by direct order many of the commercial programs that had their roots as shareware fledglings. Get a copy of the magazine and check for special membership offers before you purchase just a subscription — many times, PC-SIG wraps *Shareware* as a package with a book and free disks for only a few additional dollars.

The Alternative Software Bulletin[fn3] exhibits a slightly different format than other periodicals. Its issues are thematic (a recent issue headlined "Cheap Disk Space") and illustrate a subject group of utilities with specific program examples, comparing and contrasting named programs and their features. The newsletter is full of screen dumps, and lists new and recent additions to commercial bulletin boards such as CompuServe and GEnie. *Alternative* lives up to its name, and is a very readable publication for those who desire a no-nonsense approach to shareware and freeware reviews.

Dr. File Finder's Shareware Newsletter[fn4] is a newcomer to the scene, the first issue having been printed in April, 1991. This newsletter is an adjunct to *Dr. File Finder's Guide to Shareware* (see below) and right now is concerned mainly with reviewing new shareware programs. It is of the same quality as its namesake paperback book, and, while it is distributed in a photocopy medium, it might be fun to watch it mature into a full-fledged offset publication.

The Library

You should check your local public library to ascertain whether it has a circulating collection of public domain software. Many libraries have collections of what they term "non-print media," and some include software as part of that assortment. Typically, the library will have master copies (or a file of duplicates of master copies) of public domain and shareware programs, which you may either borrow as you would a book and copy at home or copy in the library using in-house equipment. A charge is usually made only for the cost of a floppy disk, if you do not bring your own. Many libraries will also have public access microcomputers, upon which you may test the program before you make a copy.

Several years ago, a book and disk set was published entitled *Public Domain Software on File*, and it offered, for $195.00, a generous collection of free programs for the Apple II plus and IIe. It is now available in an IBM version,[fn5] and many libraries have purchased it. Again, check with your local library to see if a copy is available. There is a potpourri of utilities and educational programs (with documentation) in the publication, and the deluxe edition contains additional Business Satellite Diskettes featuring, as the name implies, business applications programs. These programs are royalty- and copyright free and generally require no further payments or registrations.

Books and Disks by Mail

It is becoming increasingly popular for paperback computer books to contain one or more diskettes of the programs described in the text. This, of course, makes a book slightly more expensive, but it removes the tedium of retyping source code for programs in the book with which you wish to experiment. The books on shareware make a similar provision, and there is an opportunity here to pick up some valuable software along with very detailed documentation, usually written by the shareware author him/herself.

PC Magazine DOS Power Tools[fn6] was a treasure trove when it was first published, and the second edition is an even greater bargain. A collection of utility programs which first appeared in regular issues of *PC Magazine*, *Power Tools* contains several hundred DOS utilities, applications and subroutines for the power programmer to use as standalone programs or as incorporations into other software routines. In many cases, you'll need to compile the C code or assemble the .ASM files with a macro assembler, but the result is worth it. If only two or three of the programs are to your liking, the book was worth at least what you paid for it. The source code, and some .EXE files compiled from the related source, can be found on the MagNet bulletin board (mentioned below), and updates of programs since the book was published can be found there as well.

Another recently published source of public domain programs is *Dr. File Finder's*

Guide to Shareware.[fn7] In addition to the nearly one megabyte of software included with the book, the appendix is full of deals galore for other shareware programs. One of the coupons gets you twelve diskettes of software, plus a sample copy of a shareware newsletter, for $38.00. The authors have "test driven" all the programs included in the chapters, and they offer pointers for best use of the full-featured versions of each program. Check your local bookstore for a copy of this extensive work.

One smaller, but rather useful shareware book is *Alfred Glossbrenner's Master Guide to Free Software for IBM's and Compatible Computers*.[fn8] As advertised, the book "details programs available for either nothing or next to nothing." Glossbrenner is somewhat of a CompuServe guru, and he shares his expertise in successful treasure hunting in this informal, clearly written compendium. Another reference in this category is *Finding (almost) Free Software*,[fn9] which offers an added bonus of $170.00 worth of discount coupons on many of the registered versions of the programs described therein. This book features entire chapters on some of the "heavy hitters" in the shareware realm, including PC Write version 3.03, the AsEasyAs... spreadsheet, PC-File 5.0 and PC-Type 5.0 (both further described below), the Flushot+ virus detector, PKZIP and PKUNZIP file compressors, PC Chart, PC Speech, and some games. A feature of this book is a chapter on some upcoming Windows version 3.0 shareware utilities. The work lives up to its declaration as a "guide to shareware and public domain software for conscientious PC users."

Catalogs and Trade Lists

As with any information medium, shareware can be purchased from specialty dealers. The advantage here is that the vendor carries programs with common denominators of price and functionality, and your number of choices in various categories is greatly expanded.

PC-SIG issues its catalog as *The Encyclopedia of Shareware*[fn10] in paperback form, and the volume is available in many bookstores. This is a good reference for a retrospective look at the PC-SIG library and offers the reader a chance to fill the applications and function gaps in his/her own program collection from one source.

Public Brand Software[fn11] makes available an extensive selection of shareware programs. You'll want to get a copy of their detailed, annotated catalog to browse items costing (initially) only $5.00 per disk. PBS is especially good for carrying unusual items such as cross compilers, mathematical and statistical tools, accounting packages, and graphics utilities. There is a generous complement of standard programs as well.

Other dealers exist, and the following list is compiled for the comprehensiveness of their catalogs: The Software Labs[fn12] charges $3.49 per disk, and $2.99 each in quantities of ten or more. Shareware Express[fn13] charges $4.95 per disk, and often runs sales and "bargain basement" deals along with its regular catalog offers. Computer Solutions[fn14] has 254 disks in its catalog, and will sell the collection for $3.49 for individual disks, or the entire collection either by outright purchase ($450.00) or rental ($250.00). The latter arrangement allows the shareware user to examine the entire collection and copy only what is needed before returning the consignment. All prices are quoted as of the time of compilation of this article.

CD-ROM Packages

The availability of collections of one-half gigabyte of software for pennies a program might just spur many of us to break down and spend $600-$800 for a CD-ROM drive and interface. The PC-SIG organization mentioned above makes available its entire library of programs on CD-ROM.[fn15] And you might, as I did at a local computer flea market, find a discounted collection of other shareware on compact disk such as *Phethean's Public Domain Library #1 for IBM & Compatibles*[fn16] for under $40.

DAK Industries Incorporated[fn17] used to handle only consumer electronic equipment, but lately the company has been getting into the PC business. It has also made available, in addition to a $399.00 external CD-ROM drive, various collections of CD-ROM software at sellout prices. A recent offer was for five CD-ROM reference tool programs for $300.00 (when purchased with a drive). This is a "remainder table" type of material, but still a bargain when compared with the programs' full retail prices. Another excellent source for CD-ROM based shareware programs is the Ztek Company.[fn18] Its offerings include *Software Library* (1,400 programs, $299.00), *Shareware Carousel* (500 megabytes of files, $225.00), and *C-CD ROM* ($99.00), a disk of public domain C language source code, templates, and utilities. Ztek also trades in commercial CD-ROM programs and has available for sale various drives for IBM compatible PCs.

For some CD-ROM programs, you may need the Microsoft Extensions program to break the DOS 640K memory barrier limitations. If a copy isn't included with the drive you purchase, the price is around $50 when bought from a dealer. I won't go into details of interfacing CD-ROM drives in this essay; persons requiring information on using these peripherals with Zenith Data Systems hardware are referred to my December, 1990 *REMark* article entitled "CD-ROM Setup on a Zenith Data Systems Computer."

Bulletin Boards

Remote BBS systems, such as the one the Zenith Users' Group offers to its members,[fn19] typically make available a variety of shareware programs in their downloadable files sections.These are usually of the same content and format as found in the over-the-counter disk-based offerings. There is an added advantage that the BBS' system operations manager (SYSOP) may have screened the uploaded programs for appropriateness, adherence to copyright, and functionality before opening them to bulletin board users. In addition, (s)he may have added some blurbs or comments to a "new programs" area that prevent the potential user from wasting telephone connect time on programs that prove inappropriate for one's current needs.

There is a necessary caveat here, one that has been spoken countless times but is particularly germane to shareware users. We have all heeded the caution of being wary of picking up computer viruses on public bulletin boards, but the mental deficients who create these viruses take particular delight in preying on shareware users. For example, there is a Trojan horse virus that mimics the popular virus scanning software Viruscan made available by McAffee and Associates. The fake version, usually distributed as SCANV70.ZIP, destroys all files on a PC when it is unZipped. It has been identified on several bulletin board systems. The latest (as of this writing) version of the real Viruscan is 67C and appears as SCANV67C.ZIP, and the latest version should be able to detect the presence of its evil alter ego. Just be advised that extra prudence is not wasted when hunting for public domain and shareware treasures.

Several online services have free or inexpensive software available for downloading. *PC Magazine* offers many of the utility programs featured in *PC DOS Power Tools* on its MagNet[fn20] bulletin board. CompuServe[fn21] offers a variety of public domain programs across its IBM Applications Forum, IBM Communications Forum, and IBM Systems/Utilities Forum, and GEnie[fn22] makes available files from its IBM PC Roundtable. Public Brand Software's PBS-BBS[fn23] makes available nearly all of the titles from its extensive catalog for a nominal annual fee.

Categorical Examples

It is almost easier to ask, "What kinds of software do not appear as shareware?" than "What kinds do?" Space limitations prevent offering a rundown of every possible category, so I'll take author's liberty and mention just a few that I have found to be exceptionally useful.

Author Jim Button has, for several years, written and refined a triumvirate of programs known collectively as

ButtonWare;[fn24] namely, PC-Type+, PC-Calc+, and PC-File+, a word processor, spreadsheet, and data base manager respectively. PC-File has evolved into PC-File:dB, and is compatible with the dBase III plus file format. PC-File version 5.0 has replaced both PC-File+ and PC-File:dB, retaining the features of both. Button has taken special care to ascertain that document and data files are interchangeable between programs, and the trio makes a versatile integrated package. I find the shareware versions extremely useful in teaching introductory computing classes, because the programs can be run virtually without printed documentation, using the author's help screens. For those instructors requiring a text, the three programs are available as a teaching package in *Applications Software for the IBM PC*,[fn25] and the text includes copies of the programs on DOS-based floppies including an added bonus of a disk of example files.

Shareware also contains many library disks of source code for various languages. This is particularly useful for building your own libraries of subroutines in Pascal, dBase III and IV, assembler, and C language. The advantage is that you can alter the syntax of the code if your compiler is not exactly ANSI compatible, or if you wish to customize a particular feature. One C language utility that seems ubiquitous on bulletin boards is Star Guidance Consulting's Window Boss.[fn26] This C language based windowmaking toolkit is a delightful alternative to the Programmer's Development Kit for Windows by Microsoft (and much cheaper, although you can't program directly for the Microsoft Windows platform with the Window Boss). The Window Boss shareware version comes with routines to support windowing for small model compilers for Microsoft C, Turbo C, Lattice C, and other popular C compilers. If you want the callable routines for other compiler models, you must register the program. For ease of use, versatility, and low price, the Window Boss is a bargain.

Those of us engaged in desktop publishing know how much of an investment a fully functional PC typesetting shop can be. Add the high cost of the hardware to the programs and software accessories needed to make publications look professional, and you can take a rather hefty sum from your computing budget. If you can't afford a scanner, one alternative is to use public domain libraries of clip art which usually sell for a couple of dollars for a disk of several dozen images. Type fonts are available this way, too, although you are sometimes limited to the sizes and orientations the disk vendor thinks you will need. But with a bit of shopping, you can avoid the $100-$200 per disk for the name brand soft fonts, and if you have the time, you can experiment with a font editor (also available as shareware) and get just the look you want.

The shareware market is also an excellent source for educational programs, including support tools for teachers. For businesspersons, to be found are template files for Lotus 1-2-3, dBase III and IV, Ventura Publisher, and AutoCAD. If it is a particular application for which you're searching, you just might find that someone else has already done the work and saved you hours of programming. Of course, shareware includes the usual complement of games, hobby materials, and tutorial programs. The sky is usually the limit, and what is useful is bound only by your own individual needs.

Afterword

It would be entirely possible to accomplish all of your computing activities by using programs completely within the shareware and freeware domain. What makes this unique mechanism work is the dedication and trust of thousands of competent programmers relying on your honor and trust to perpetuate the system. To reiterate, if you use shareware and there is a royalty fee involved, play fair and give the author his/her due. A final note: public domain software is one of the only computer fields with its own accrediting agency. The Association of Shareware Professionals[fn27] is a consumers' union for monitoring the quality of programs and standards of customer support and constitutes a sort of *Good Housekeeping* Seal of Approval. There is even an ombudsman to intercede in disputes between shareware authors and registered users.

Programs and Sources Mentioned

[fn1]*Shareware News*, published by PC-SIG, 1030 D East Duane Avenue, Sunnyvale, CA 94086. (408) 730-9291. Mailed to members; current membership $39.95 per year.

[fn2]*Shareware Magazine*, same publisher as in footnote 1 (above). Bimonthly, $14.95 per year.

[fn3]*The Alternative Software Bulletin*, published by Binary Press, P.O. Box 757, Brooklyn, MI 49230. Ten issues per year, $12.00 for a one year subscription.

[fn4]Dr. File Finder's Shareware Newsletter, published by FF & P Enterprises, P.O. Box 591, Elizabeth, CO 80107. Six issues per year, $12.00 for a one year subscription.

[fn5]*The Public Domain Software on File — IBM Package*. New York: Facts on File, Inc., 1990. Twelve 5-1/4" diskettes and binder, $195.00. Four additional 5-1/4" diskettes in Business Satellite Diskettes supplement, $50.00.

[fn6]Somerson, Paul. *PC Magazine DOS Power Tools: Techniques, Tricks and Utilities*. New York: Bantam, 1990. Second edition. Paper, 1,272 pages, $44.95 (includes two 5-1/4" disks; 3-1/2" disks are available by mail for an additional $7.95).

[fn7]Callahan, Mike and Anis, Nick. *Dr. File Finder's Guide to Shareware*. Berkeley, CA: Osborne McGraw-Hill, 1990. Paper, 1,019 pages, $39.95 (includes a 5-1/4" disk; coupon in book entitles purchaser to request disks 2 and 3 of the set free, by return mail).

[fn8]Glossbrenner, Alfred. *Alfred Glossbrenner's Master Guide to Free Software for IBM's and Compatible Computers*. New York: St. Martin's Press, 1989. Paper, 624 pages, $18.95.

[fn9]Schlentner, Klaus. *Finding (almost) Free Software*. Grand Rapids, MI: Abacus, 1990. Paper, 289 pages, $16.95.

[fn10]*The Encyclopedia of Shareware*. 3rd Edition, $19.95. See *Shareware News* (footnote 1, above) for mailing address.

[fn11]Public Brand Software, P.O. Box 51315, Indianapolis, IN 46251. (800) 426-3475, or (317) 856-7571 in Indiana; request a copy of their "Shareware Catalog & Reference Guide."

[fn12]The Software Labs, 3767 Overland Avenue #112-115, Los Angeles, CA 90034. (800) 359-9998.

[fn13]Shareware Express, 27601 Forbes Road, Suite 37, Laguna Niguel, CA 92677. (800) 346-2842.

[fn14]Computer Solutions, P.O. Box 354, Mason, MI 48854. (800) 874-9375, (517) 628-2943 in Michigan.

[fn15]*The PC-SIG Library on CD-Rom*. 9th Edition, $299.00. See *Shareware News* (footnote 1, above) for mailing address.

[fn16]*Phethean's Public Domain Library #1 for IBM & Compatibles*. Available from Peter J. Phethean, Ltd., 1640 E. Brookdale Avenue, La Habra, CA 90631. Single CD-ROM disc, $39.95.

[fn17]DAK Industries Incorporated, 8200 Remmet Avenue, Canoga Park, CA 91304. (800) 325-0800; (818) 888-8220 in California.

[fn18]Ztek Company, P.O. Box 1055, Louisville, KY 40201. (800) 247-1603; (502) 584-8505 in Kentucky.

[fn19]COM1 Bulletin Board, c/o Zenith Users' Group, P.O. Box 217, Benton Harbor, MI 49023-0217. (616) 982-3463 (voice).

[fn20]PC MagNet, c/o *PC Magazine*, One Park Avenue, New York, NY 10016. (800) 635-6225 (voice). $12.80 per connect hour at 1,200 or 2,400 baud, $6.30 per hour at 300 baud.

[fn21]CompuServe, P.O. Box 20212, Columbus, OH 43220. (800) 848-8990 (voice); (614) 457-8650 (voice) in Ohio. $12.50 per connect hour at 1,200 or 2,400 baud, $6.00 per hour at 300 baud, plus $0.30 per hour communication surcharges.

Continued on Page 48

Getting the Most From Your Computer

John Lewis
6 Sexton Cove Road
Key Largo, FL 33037

Part 1

I bought my first computer back in the latter part of 1982. A Christmas present to myself, so to speak. Up to that point, I had been ignoring computers in the work place, but the time came when I could no longer ignore their presence so I bought a Heath H-89 with very few frills and very little software. I admit to being a bit eccentric when it comes to learning a new skill. I bought my first airplane (a Piper Cherokee 140) when I had only three hours flying time! I now hold a commercial pilots license with many hours in the log book, but that's another story.

One of my close friends was involved in programming and I thought that I would do the same, as an instrument in the learning process. Needless to say, that was quite some time ago and I stumbled many times, but I must admit, my education in programming has been quite enjoyable. An enterprise which, by the way, is still continuing.

I became a subscriber to *REMark* soon after that and I have learned much from its pages. Lately, however, I've noticed a turn in the direction of its contents. It seems that very few readers are interested in writing their own programs. A very unfortunate turn of events for a number of reasons. One of the most compelling reasons for getting involved in programming is the satisfaction gleaned from the act of creating something of your very own that is both useful and unique.

I admit to owning quite a bit of software not written by me, but the programs used most are those that are a product of my own creativity. As an example, the editor being used to write this article was written in Turbo Pascal, by yours truly. Also on line is a TSR (Terminate and Stay Resident) utility that incorporates a calculator and a screen saver. This last program is my latest creation in Turbo Pascal. I own other more elaborate word processing software but my favorite is the one I'm using, for obvious reasons. Another piece of software that gets a lot of use is a database program (also written in Turbo Pascal) that is used extensively in the making of color prints (I run a small photographic business).

In short, my computer gets a lot of use and that brings me to the object of this article. The creation of useful programs is much easier than ever, by a factor that I would hesitate to guess at. The learning curve is much easier to traverse now, due, in large part, to the plethora of PC software and books on the subject.

OK, since you are still reading this, you are interested. Probably a bit skeptical but interested none the less. My intention is to take the reader through a series of tutorials in Pascal programming where we will create some useful programs, designed to provide a library of functions and procedures that will enhance future development efforts. I will attempt to explain the code used to produce these program segments in a manner conducive to a thorough understanding of the algorithms used.

Why Pascal instead of "C" or BASIC or even Fortran? The reasons are manifold, the most compelling are that Turbo Pascal is a software bargain (in my humble opinion) and easy to learn as well. Included in Borland's compiler package is a built-in debugger. A very important software tool which we will explore in our quest for programming achievement. Another very important advantage is Turbo Pascal's incredible speed. I better stop before I begin to sound like a Borland salesman, but in truth, I know of no better vehicle for diversified software creation.

We will first explore some of Pascal's methodology, delving into procedures and functions. Then some of the more esoteric aspects of the language will be examined. We will even convert some of the more generic program fragments into Pascal units.

If you wish to pursue this series and gain some programming skills, you will need the following tools: Version 4.0 (or later) of Turbo Pascal and an IBM PC or compatible computer.

My own tools consist of Version 5.5 of Turbo Pascal and a Zenith Data Systems Z-248 as well as a Z-386 SX (I'm a firm believer in good tools).

Each article will contain a useful program listing which will eventually lead to some interesting software.

Still on board? OK, hang on, were going on a tour of Pascal, Borland's version.

Most programs of any complexity are composed of a main body and a series of subroutines which do most, if not all, of the work.

The subroutines are often designed to be used with other programs. In other words, generic in nature.

Pascal uses procedures and functions to be called to perform various routines. A *procedure* is called to perform a task where no value is returned to the calling body of code. A *function* is used when a value must be returned by the routine, much like an *int* Function in "C".

Although a procedure can not, by its very nature, return a value, it can modify a variable. What's the difference? By using the key word "Var" on its argument list, a procedure will operate on the variable in its memory location, thus changing the value in memory. If the word "Var" is omitted from the argument list, the variable is copied into the procedure where only the copy is operated on. Let's look at a short program (Figure 1).

Copy this short example and then compile and run it. You should see a rather graphic example of the difference between the two procedures. Since we can change a program variable within a procedure,

```
Var x, y : Integer;
Procedure NoChange (a, b : Integer);
   Begin
   a:=a+b;
   end;

Procedure ChangeVal (Var a, b : Integer);
   Begin
   a:=a+b;
   end;

begin
writeln;
x:=5;y:=10;NoChange(x,y);
writeln('After nochange, x = ',x);
ChangeVal(x,y);
writeln('After ChangeVal, x = ',x);
end.
```

Figure 1

why use functions? A good question, and the answer may seem a bit trivial at first, but there are some very good reasons for using a function for modification of a variable.

Variables come in two flavors, *global* and *local*. What's the difference? We just saw an example of global variables. The first declaration after the program name was: Var x, y : Integer; This statement creates two integers (two byte numeric variables) that can be accessed by any part of the program since they were declared outside of any procedure or function, in other words – global.

Each procedure contained a parameter list, not a variable declaration. This list tells the compiler what kind of variables to expect when called. If a different type of variable is used in the calling sequence, the compiler will flag an error. For instance, if we had declared x and y as "real" variables (six byte numeric variables) the compiler would have refused to compile the code and displayed an error message informing us that the variables were of the wrong type. Turbo Pascal goes even further than that, it will place the cursor at the location (within the source code) where it encountered the first occurrence of a type mismatch.

Let's look at a function in the same context and see how it differs from the two preceding procedures. We'll create a function that returns the sum of two integers. (See Figure 2.)

You will notice that we assigned z as

```
Var
   x, y, z : Integer;

Function Product ( a, b : Integer):Integer;
   Begin
   product:=a*b;
   end;

begin
x:=6;y:=4;
writeln;
z:=Product(x,y);
writeln('After Product, z = ',z);
end.
```

Figure 2

the value returned by our function "Product". Notice also that we declared a parameter list, specifying that we would be calling the function with two integers and that an integer would be returned. Please note the line: product:=a*b; within the body of the function. This line serves to define the function. If you should substitute the line z:=a*b; the compiler will compile the code, but the function will fail to deliver the goods. In fact, the answer will be garbage, whatever value the compiler assigns to z. The reason for this is that the function is not defined within its body.

We used another very powerful statement in our short example. "Writeln" and its cousin "write" are used to output to the screen, or in one of its variations, to a disk file. We'll leave the disk I/O version for later, but let's look at the screen output version before we proceed.

This is an area where Pascal makes our work much easier to perform. We can call writeln with a string to be printed, an integer, a real or other data type and our function will dutifully print it. If we call writeln with no parameter list, it will simply write a carriage return/line feed sequence to the screen (vertical space). Its cousin, "write", requires a Parameter list if it is to accomplish anything. The statement: Write(''); will cause a screen output which consists of precisely nothing. Not a terribly useful command in this context, but as we will soon see, a very useful adjunct to writeln.

Our programming project for part one of this series is the creation of a menu "Window". We will be using cursor control commands extensively and the write(parameter list) command will be used in conjunction with the gotoxy(column, row) Pascal procedure to create the frame for our window. We will use the "write" procedure in lieu of "writeln" since, although "writeln" would still achieve the desired result, it would cause execution of a carriage return/line feed sequence with each call. An unnecessary operation since the cursor will be controlled by a "gotoxy(column, row);" command through most of the sequence. We can save a bit of time by omitting the carriage return/line feed sequence. Another advantage gained through the use of "write" instead of "writeln" is that we will use the default position of the cursor to advantage when printing the horizontal characters to the screen.

Since speed is one of our goals in the creation of any computer program, we will use every tool at our disposal to achieve that end.

You will notice that, just below the program name "DrawBox" in the enclosed listing, is the line: "uses crt;". The Turbo Pascal library "gotoxy(column, row)" procedure code is contained in that "unit". This is another area of program development that Turbo Pascal excels in. The best programs are written in a method that use a modular style. The program is broken down into a number of tasks which are then accomplished by procedures or functions. We can further modify a program by enclosing various, usually related procedures and functions, into units; thus encapsulating their code. This methodology lends itself admirably to our project since we will be breaking this job into pieces that will fit into the magazine format on a month by month basis.

Pascal units offer many advantages, one we have already mentioned, modularization, but another important advantage is the way in which memory is allocated to Turbo Pascal programs. By using units to break up our project, we can spread the memory limitations over a large area, avoiding any head to head confrontations with exceeding the limits otherwise imposed (64K is the limit to any one body of executable code under Turbo Pascal). This might sound a bit far fetched, but "Au Contraire", my darkroom program that is in frequent use here and undergoing evaluation by a major photo manufacturer, is over 120 K in its executable form. It started life as a small program that accomplished just one job, while containing under 20 K of code.

The code listing included, (DrawBox.pas) is in a regular program format. I have left it that way since it may be a bit easier to understand. Next issue we will convert it to a unit before we get into the next project – a computer calculator that features the ability to operate on a number, operator, number format and includes the capability to "back up" (using the backspace key). We will then combine the two programs to create a menu accessible calculator. We will continue to add items to the menu each issue until we have a valuable, unique piece of software. Each part of this series will also cover new facets of Turbo Pascal, a very fascinating computer language.

OK. give me the story one more time, you're reading a borrowed REMark?

DrawBox

```
uses crt;

 const
   Top    = 5;              { Alter these values to move the menu box }
   Left   = 5;              { or change its size }
   Right  = 75;
   Bottom = 22;
   MenuCol1 = 5;
   MenuRow1 = 4;

Procedure draw_box(UplftX,UplftY,LowrtX,lowrtY:integer);
   Var i : Integer;

   const
      TopLft = 201;        { These characters may be changed to alter the }
      Hor    = 205;        { appearance of the box }
      Vert   = 186;
      BtLft  = 200;
      TopRt  = 187;
      BotRt  = 188;

   begin
   gotoxy(UplftX,UplftY);write(chr(TopLft));{ draw top left corner }
   for i:= (uplftX+1) to (LowrtX-1) do{ top row  }
   begin
   write(chr(Hor));{ draw the top horizontal char }
   end;
   write(chr(TopRt));{ draw top right corner }
 for i := (UplftY+1) to (lowrtY-1) do begin
   gotoXY(UplftX,i);write(chr(Vert));{ draw both vertical columns }
   gotoXY(LowrtX,i);write(chr(Vert));
   end;
   gotoXY(UplftX,LowrtY);write(chr(BtLft));{ draw bottom left corner }
   for i:= (UplftX+1) to (LowrtX-1) do
   write(chr(Hor));write(chr(BotRt));{ draw bottom horizontal char }
   end;{ followed by the bottom right char }

 Procedure Esthetics;
 Begin
   TextColor(Black);TextBackground(LightGray);
   ClrScr;
 end;

 Procedure SetUp;
 Begin
 window(Left+1,Top+1,Right-1,Bottom-1);
 TextBackground(Blue);TextColor(Yellow);clrscr;
 end;

 Procedure Contents;
 Begin
 Gotoxy(MenuCol1, MenuRow1);write('Menu Window complete ');
 end;

 Begin
 Esthetics;
 draw_box(Left,Top,Right,Bottom);SetUp;Contents;
 readln;
 end.
```

Copy this listing very carefully, then compile and run it. If you are like me, you will have to find the inevitable typos and correct them before the program will execute. Fortunately, Turbo Pascal makes finding mistakes quite easy. Make a backup copy of the source code and park it on your disk along with Turbo's Editor/Compiler. We have just started a very interesting exploration of Pascal. ✻

Continued from Page 45

[fn22]Genie, 401 N. Washington Street, Rockville, MD 20850. (800) 638-9636 (voice). $18.00 per connect hour (prime time rate); $4.95 per month (connect charge only, if no prime time use is invoked).

[fn23]PBS-BBS, c/o Public Brand Software; see footnote 11 for address. $50.00 per year for one connect hour per day of access.

[fn24]ButtonWare, Inc., P.O. Box 5786, Bellevue, WA 98006. (206) 454-0479.

[fn25]Shuman, James E. *Application Software for the IBM PC*. Santa Cruz, CA: Mitchell Publishing, Inc. Paper, second edition, 318 pages [supplemental option package contains seven 5-1/4" floppy disks].

[fn26]The Window Boss. Star Guidance Consulting, 273 Windy Drive, Waterbury, CT 06705. (203) 574-2449; check with author for latest registration fee.

[fn27]The Association of Shareware Professionals, 325 118th Avenue S.E., Suite 200, Belleview, WA 98006. ✻

Quality Enhancements!

EaZy PC Products

EZM-128: Expand 512K base memory to 640K. Simple, plug-in installation. $95.00

EZCLOCK: Calendar/Clock. Piggy-back add-on for EZM-128. $35.00

No Slot Clock/Calendar

FBE SmartWatch: Automatic date/time on bootup. Installs under BIOS/Monitor ROM. Ten year battery. Works with all Heath/Zenith MSDOS computers. For PC's $32.00, Z-100 $33.00 Module: $25.00

H/Z-148 Expansions

ZEX-148: Adds one full-size and one half-size expansion card slot. $79.95

ZP-148: PAL chip expands existing 640K memory to 704K. CGA/MDA only! $19.95

Configuration Control

CONFIG MASTER: Menu-select active CONFIG.SYS during bootup. Software for PC/Z-100 MSDOS. $29.95

H/Z-150 Items (Not for '157, '158, '159)

VCE-150: Eliminate video card. Install EGA or VGA card. All plug in. Includes circuit board, SRAM and RM-150. $49.95

ZP640 PLUS: PAL chip to expand standard memory card to 640/704K with 2 banks of 256K RAM chips (not included). $19.95

ULTRA-PAL: Three PAL chips: MR150 for 704K + 512K RAM Disk; MR150T for 640K + 512K RAM Disk; LIM150 for 640K + 512K (32 pages) of simulated v3.2 Lotus/Intel/Microsoft Expanded Memory. With software. Install on standard memory card. No soldering. Needs 45 256K RAM chips (not included) for maximum memory configuration. $39.95

COM3: Change existing COM2 to COM3. Put internal MODEM at COM2. Don't lose serial port. With software. $29.95

H/Z-100 Modifications

ZMF100A: Expand "old" motherboard (p/n 181-4917 or less) using 256K RAM chips (not included). No soldering. $65.00

ZRAM-205: Put 256K RAM chips on your Z-205 board. Get 256K plus 768K RAM disk. Contact us for data sheet before ordering. Without RAM chips. $39.00

H/Z-89 Add-Ons

H89PIP: Parallel printer 2 port interface card. With software. $50.00 Cable $24.00

SLOT4: Add fourth expansion slot to right-side accessory bus. $39.95

Order by mail, FAX, telephone, or see your dealer. UPS/APO/FPO shipping included. VISA/MasterCard. WA residents add 8.1% tax. Hours: M-F 9-5 PST. We return all calls left on our answering machine!

FBE

FBE Research Company, Inc.
P.O. Box 68234, Seattle, WA 98168
206-246-9815 Voice/FAX TouchTone Selectable
Reader Service #104

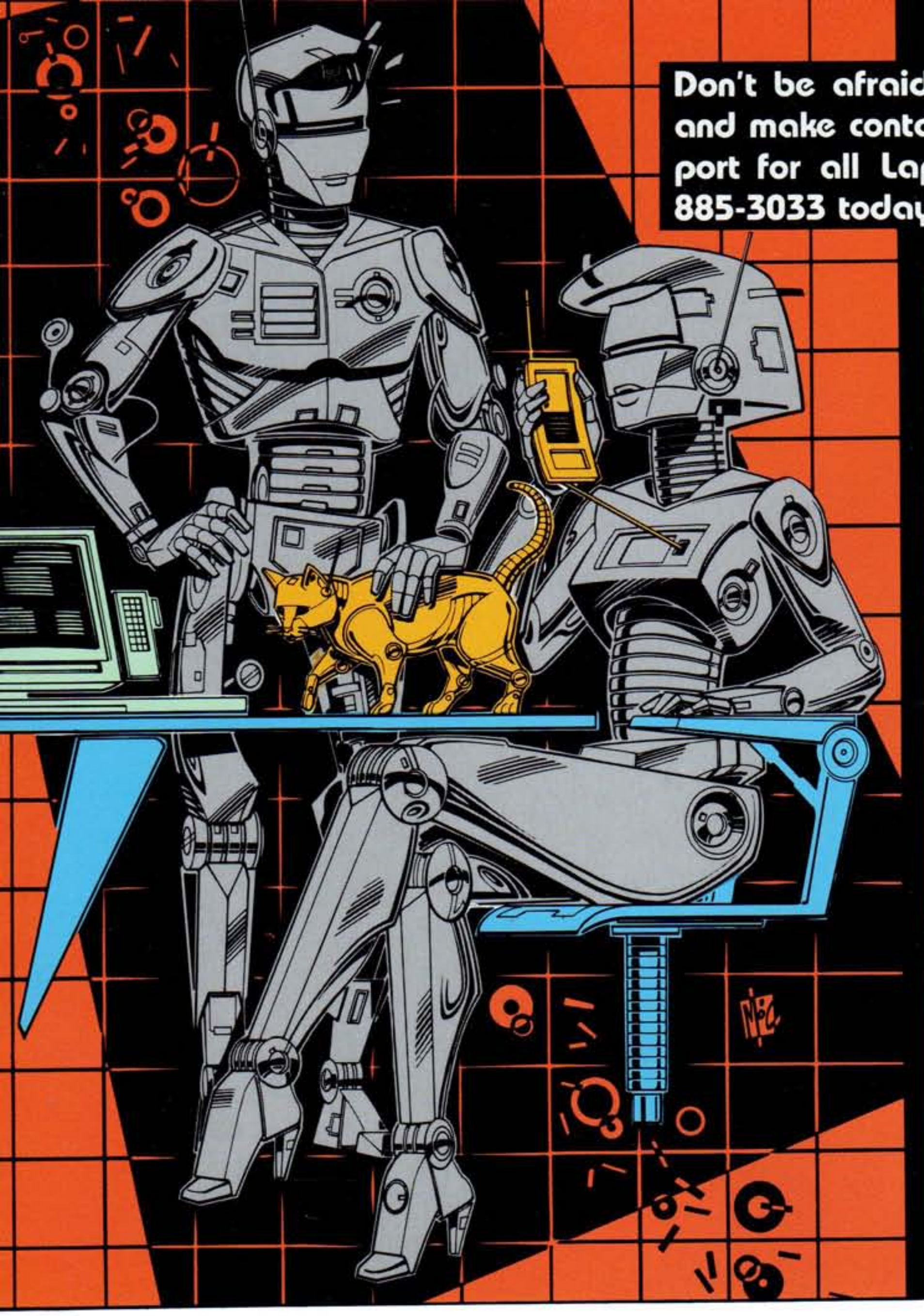
HADES II

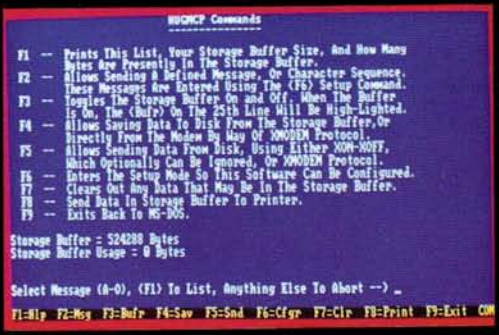
It's HOTTER than ever! Jam-packed with new features, HADES II still remains the easiest-to-use disk editor ever! Just look at some of the features:

- Sector Display/Editing
- Sector HEX/ASCII String Search
- File Display/Editing
- Physical and Logical Cluster Display
- File HEX/ASCII String Search
- Drive Parameter Display
- 512 MegaByte Drive Size Limit
- File Attribute Display/Edit
- Automatic Erased File Recovery
- Manual Rebuild File Recovery
- Works with Headerless MS-DOS Disks
- PC-Compatible or H/Z-100

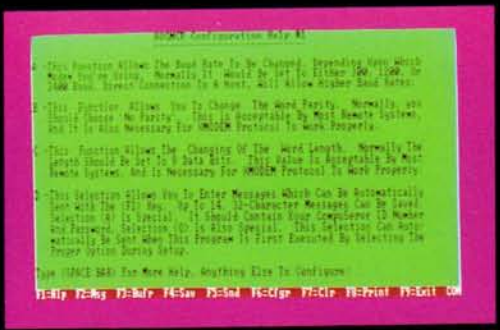
HADES II is still only $40, and original HADES owners can upgrade their distribution disk for only $15. Call HUG today at: (616) 982-3463.

Don't be afraid to communicate! Get HUGMCP and make contact the easy way. Now with support for all Laptops, order HUG Part number 885-3033 today.

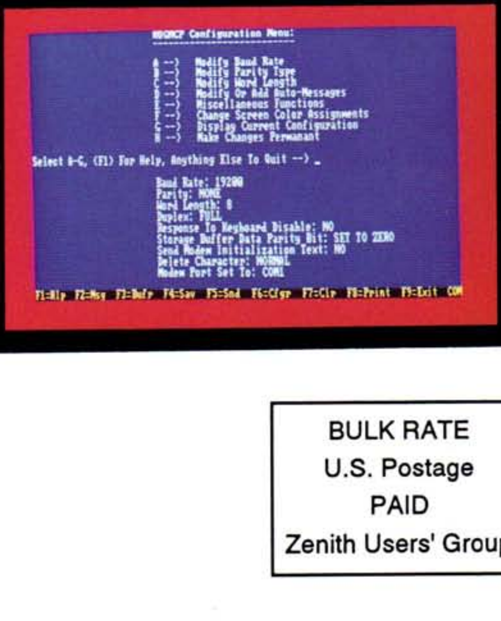




```
                     HUGMCP Commands
                     ---------------

F1  --  Prints This List, Your Storage Buffer Size, And How Many
        Bytes Are Presently In The Storage Buffer.
F2  --  Allows Sending A Defined Message, Or Character Sequence.
        These Messages Are Entered Using The (F6) Setup Command.
F3  --  Toggles The Storage Buffer On and Off. When The Buffer
        Is On, The (Bufr) On The 25th Line Will Be High-Lighted.
F4  --  Allows Saving Data To Disk From The Storage Buffer,Or
        Directly From The Modem By Way Of XMODEM Protocol.
F5  --  Allows Sending Data From Disk, Using Either XON-XOFF,
        Which Optionally Can Be Ignored, Or XMODEM Protocol.
F6  --  Enters The Setup Mode So This Software Can Be Configured.
F7  --  Clears Out Any Data That May Be In The Storage Buffer.
F8  --  Send Data In Storage Buffer To Printer.
F9  --  Exits Back To MS-DOS.

Storage Buffer = 524288 Bytes
Storage Buffer Usage = 0 Bytes

Select Message (A-O), (F1) To List, Anything Else To Abort --> _

F1=Hlp  F2=Msg  F3=Bufr  F4=Sav  F5=Snd  F6=Cfgr  F7=Clr  F8=Print  F9=Exit  COM
```



```
                 HUGMCP Configuration Help #1
                 ----------------------------

A -This Function Allows The Baud Rate To Be Changed. Depending Upon Which
   Modem You're Using, Normally It Would Be Set To Either 300, 1200, Or
   2400 Baud. Direct Connection To A Host, Will Allow Higher Baud Rates.

B -This Function Allows You To Change The Word Parity. Normally, you
   Should Choose 'No Parity'. This Is Acceptable By Most Remote Systems,
   And It Is Also Necessary For XMODEM Protocol To Work Properly.

C -This Function Allows The Changing Of The Word Length. Normally The
   Length Should Be Set To 8 Data Bits. This Value Is Acceptable By Most
   Remote Systems, And Is Necessary For XMODEM Protocol To Work Properly.

D -This Selection Allows You To Enter Messages Which Can Be Automatically
   Sent With The (F2) Key. Up To 14, 32-Character Messages Can Be Saved.
   Selection (A) Is Special. It Should Contain Your CompuServe ID Number
   And Password. Selection (O) Is Also Special. This Selection Can Auto-
   matically Be Sent When This Program Is First Executed By Selecting The
   Proper Option During Setup.

Type (SPACE BAR) For More Help, Anything Else To Configure

F1=Hlp  F2=Msg  F3=Bufr  F4=Sav  F5=Snd  F6=Cfgr  F7=Clr  F8=Print  F9=Exit  COM
```



```
                 HUGMCP Configuration Menu:
                 --------------------------

                 A --)  Modify Baud Rate
                 B --)  Modify Parity Type
                 C --)  Modify Word Length
                 D --)  Modify Or Add Auto-Messages
                 E --)  Miscellaneous Functions
                 F --)  Change Screen Color Assignments
                 G --)  Display Current Configuration
                 H --)  Make Changes Permanant

Select A-G, (F1) For Help, Anything Else To Quit --> _

                 Baud Rate: 19200
                 Parity: NONE
                 Word Length: 8
                 Duplex: FULL
                 Response To Keyboard Disable: NO
                 Storage Buffer Data Parity Bit: SET TO ZERO
                 Send Modem Initialization Text: NO
                 Delete Character: NORMAL
                 Modem Port Set To: COM1

F1=Hlp  F2=Msg  F3=Bufr  F4=Sav  F5=Snd  F6=Cfgr  F7=Clr  F8=Print  F9=Exit  COM
```

ZENITH data systems

Groupe Bull

BULK RATE
U.S. Postage
PAID
Zenith Users' Group

POSTMASTER: If undeliverable, please do not return.

$2.50
P/N 885-2137