$2.50

REMark®

Volume 5, Issue 2 · February 1984
885-2049

Official magazine for users of HEATH ZENITH computer equipment.

DISK DRIVES DISK DRIVES DISK DRIVES DISK DRIVES DISK DRIVES DISK DRIVES DISK DRIVES DISK DRIVES

PRICE BREAKTHROUGH

Super Sale on New Disk Drives

Starting at **$169.00!**

Tandon — Siemens — Remex — MPI — Teac — Shugart — Tabor

40 or 80 Tracks — Single or Dual Head — New 3½″ Drivette™
Our Disk Drives Are Capable Of Single **And** Dual Density Operation

The **NEWEST** Technology Capable Of Operating On Most Popular Computers

Drive a Hard Bargain!!™ — 5 M.B.-20 M.B. Complete Systems ...... from $999.95

Diskette Breakthrough — 10 Pack in Library Case — $1~~8.95~~ NEW LOW PRICE

Since We Are Always Finding Ways To Save You Money, Please CALL For Our Most Current Pricing.

TOLL FREE ORDERING 1-800-343-8841

GENERAL AND TECHNICAL 1-617-872-9090

Model I/III/IV Drives (0 1 2 3)	starting at $169.00
Color Computer Drive (0 1 2 3)	$ Call Toll Free
Apple/Franklin Compatible Add-On Drives with Case & Cable	Now 40 Tracks ~~$219.95~~ NEW LOW PRICE
Apple/Franklin Compatible Drives with Controller	$259.95
Model I/III/IV Memory Upgrade	Call Toll Free
Printers — Daisywheel/Dot Matrix	$ Call Toll Free
Diskettes in Library Cases	$1~~8.95~~ New Low Price
Cases and Power Supplies — (Single-Dual-1/2 Height)	starting at $44.95
Printer **Buffers** 8K to 512K	starting at $143.95
Percom Double Density Controller (Model I)	$ Call Toll Free
Holmes Model I/III Speed-up Mod-VID/80	starting at $90.00
Color Computer Printer Interfaces	starting at $29.95
Cables — Printer/Disk Drive	starting at $23.00
DOSPLUS	$ Special Prices
Repair Services Now Offered — FAST Turn-a-Round	$ Call Toll Free

Warranty on Disk Drives — 6 Months — Extended Warranty $ Call Toll Free

SOFTWARE SUPPORT, INC.

One Edgell Road, Framingham, MA 01701 (617) 872-9090
Hours: Mon. thru Fri. 9:30 am to 5:30 (E.S.T.) Sat. 10 am to 4:30 pm

DEALER INQUIRIES INVITED.

TERMS:
M.C./Visa/Amex and personal checks accepted at no extra charge.
C.O.D., please add $3.00.
Shipping: Please call for amount.
Not responsible for typographical errors.
© Copyright 1983

CANADA
MICRO R.G.S. INC.
751, CARRE VICTORIA, SUITE 403
MONTREAL, QUEBEC, CANADA, H2Y 2J3
Regular **Tel. (514) 845-1534**
Canadian Toll Free 800-361-5155

Service! Service!

All in stock products are shipped within 24 hours of order. Repair/Warranty service is performed within 24 hours of receipt unless otherwise noted. We accept C.O.D., foreign and APO orders. Schools and D&B corporate P.O.s accepted.

Heath/Zenith Users' Group

HUG Manager .... Bob Ellerton
Software Engineer .... Pat Swayne
HUG Bulletin Board
and Software Developer .... Terry Jensen
Software Coordinator .... Nancy Strunk
HUG Secretary .... Margaret Bacon
REMark Editor .... Walt Gillespie
Assistant Editor .... Donna Melland

Printers .... Imperial Printing
St. Joseph, MI

REMark is a HUG membership magazine published 12 times yearly. A subscription cannot be purchased sparately without membership. The following rates apply.

	U.S. Domestic	Canada & Mexico	International
Initial	$20	$22*	$30*
Renewal	$17	$19*	$24*

*U.S. Funds.

Membership in England, France, Germany, Belgium, Holland, Sweden and Switzerland is acquired through the local distributor at the prevailing rate.

Limited back issues are availble at $2.50 plus 10% handling and shipping. Check HUG Product List for availability of bound volumes of past issues. Requests for magazines mailed to foreign countries should specify mailing method and appropriate added cost.

Send Payment to: Heath Users' Group
Hilltop Road
St. Joseph, MI 49085

616-982-3463

REMark®

Volume 5, Issue 2 • February 1984

on the stack

Thoughts of Summer *Margaret Bacon* 5
Buggin' HUG 6
A Tutorial On Random Numbers *Kenneth Mortimer* 10
Implementing Heath Escape Sequences for CP/M and Z-DOS *William Adney* 13
Simple Printer Controls For The Z-100 *Charlie Layman* 16
Files and File Handling *David Warnick* 17
COBOL Corner IV *H. W. Bauman* 20
ZBASIC Mapping Program: BASMAPER *Ted Miller, Jr.* 23
Making the CP/M DUMP Program a Useful Utility *Charles Horn* 28
Introduction To Data Structures *Emily Yount* 29
An Introduction to 'C' Part IV *Brian Polk* 32
HUG New Products 34
Get Rid of "Echo On Delete" In CP/M-86 *Pat Swayne* 37
The Stupid Computer *Wallace Theodore* 38
Improved Error Recovery For CP/M *Pat Swayne* 40
So, Your Computer Can't Add *David Pelowitz* 42
Disk Access By Tracks and Sectors *David Vinter* 48
Getting Started With Assembly Language (#9) *Pat Swayne* 50
Train *Paul Hinson* 56
dBASE II Programmer's Notebook *Tom Huber* 57
Squeezing The Most Out of Your HDOS Diskettes *Glenn Roberts* 58
Simply Graph It! *Crawford MacKeand* 63

ON THE COVER: A Valentine design done on an H/Z-100 by John Gillespie (Walt's son) using PALETTE, a ZDOS graphics package by Dale Wilson (distributed by Software Wizardry, St. Charles, MO).

Checklist for Z-100 Computer Users:

1. Are you getting the best prices?

If you put two computer dealers in the same room, and they BOTH tell you that "they won't be undersold", then one thing you can be sure of is that at least one of them is full of it! We won't make such a silly claim, but it is true that our prices are consistently amongst the lowest — and very often ARE the lowest around!

2. Does your dealer solve problems, or create them?

When you are just learning a computer system, one thing you DON'T need is a dealer that you have to TRAIN! On our staff we have no one with a background in computer sales, but we DO have people with backgrounds in computer science and engineering! In place of hard-sell, you'll get solutions. If it doesn't sound like much, ask yourself if you'd trust your accountant to recommend a good compatible printer to you!

3. Do you receive merchandise on time?

Wizardry normally can (and WILL) ship stock items within 24 hours, and most items ARE in stock. In most cases we ship standard UPS, but we also can ship UPS Blue or even Federal Express when you're really in a pinch, for delivery within a mere 2 days to your door!

4. Is your supplier a proven performer?

We're proud of our past record, and our reputation. Many of the same people now developing products for the Z-100 got their start in a computer from us. These guys know where the best single-source for pricing, support, and service is — and they keep coming back for the same reason. Our support is legendary, and we've literally won awards for end-user support to our customers.

5. What are your dealer's references?

If we listed just the major firms and institutions that purchase their products from us, we'd fill this page with just their names. Scratch a knowledgeable computer consumer, and you're apt to find a Software Wizardry customer there.

6. Isn't $1,000 worth a few minutes of your time?

That's how much we can save you on an average system sale! All it takes is a letter or phone call to receive our FREE catalog. In it you'll find over 450 compatible Z-100 products, probably many you didn't even know existed! We LOVE the Z-100 computer, and think the Zenith users are the best in the world! That's why we know our future is secure, the best users sooner or later find their way to the best dealers. We wouldn't have it ANY other way!

ZENITH | data systems
AUTHORIZED SALES & SERVICE

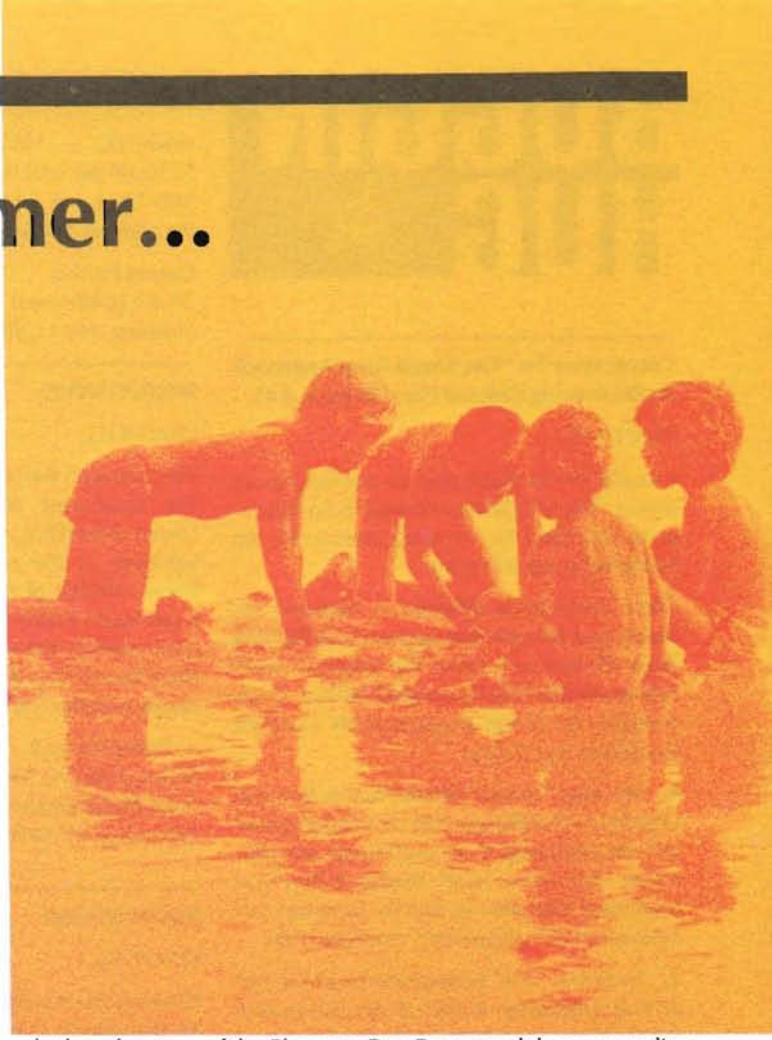
Software Wizardry
THE MAGIC TOUCH

MasterCard VISA

122 Yankee Drive
St. Charles, MO 63301
(314) 946-1968

Thoughts of Summer...

Margaret Bacon
HUG Secretary



The groundhog is just about to pop out his head and we don't dare wait any longer to start preparations for the International Heath/Zenith Users' Group Conference. Yes, this year we must recognize that HUG is International.

As good as the 1st HUG Conference was, most of what we have heard about the 2nd HUG Conference indicates that those who attended felt that it was much improved. We are going to try again! The general format we are considering turns the normal convention scheme upside down. We are going to turn you loose on the vendors long before we start tying up your time with general meetings, seminars, etc.

After two very successful years at the Hyatt Regency O'Hare, the Conference is moving. We don't want you to get bored. The 1984 Conference will be held at Pheasant Run Resort in St Charles, Illinois, the weekend of July 27-29. We have arranged a weekend package (like those get-away weekends you hear about) so, plan to arrive Friday and stay until Sunday afternoon.

Pheasant Run and St. Charles are nestled in the Fox River Valley about 45 minutes from O'Hare Airport in Chicago. The Hotel is very experienced in various ways of getting you there, but more about this in the March Issue of REMark. The DuPage County Airport is located adjacent to Pheasant Run for those of you who are pilots. Pheasant Run was originally a farm. The original farmhouse is still there and the exterior of the barn can be seen in the buildings. As the resort has expanded and added accommodations, several room choices have become available. More about this next month.

Many of you have commented that while you were having a great time, there was nothing for your guest(s) to do. That is one of the reasons we are moving to Pheasant Run. Now your spouse (family) can have a vacation while you are "bit-bashing". The facilities include abundant recreational opportunities. Pheasant Run offers an 18 hole golf course, tennis courts, the largest health club in the Mid-West, and shops. The night life available is famous in Chicago. The model of Bourbon Street is a delight with live music and that Bourbon Street atmosphere (I wanted to hold the entire conference there but Bob said it wasn't big enough). The Dinner Theater is the pride of the resort. There are six eating areas available. St. Charles and the surrounding area are rich with quaint shops, antiques around every corner, and turn-of-the-century and earlier buildings to see. There could be a riverboat trip down the Fox River, a side trip to Long Grove Village (I think they should have called it Walnut Grove) or you might enjoy wandering through neighboring Geneva, Illinois. It's beginning to sound like you should plan to come for a week.

The general theme for the Conference this year is "Adventure". Not only the adventure of the Pheasant Run Resort and the surrounding area, but the adventure of the Heath/Zenith Computer World. We have many ideas about subject matter that would be an adventure for you, but we would appreciate your suggestions. We are hoping that time will be available for special interest groups to gather. Pheasant Run has the space that will make this possible. If you have a group that wants to meet, let us know as soon as possible. We will see if we can help. Try to estimate the number of HUGgies that might want to join you. We have a bit more space for Vendors this year so the opportunity to see what is new, or just what is available, will be good.

We would like to take this opportunity to thank each of you who attended the First and/or Second National HUG Conference for your support and confidence in the Heath/Zenith Users' Group. For those of you who have not yet attended a Conference, we welcome you to participate in what is hoped to be an even bigger and better chance to get to know fellow HUGgies from all over the United States and the world.

Watch for the Official Registration Form in coming issues of REMark along with additional details as they become available. Please wait for the Official Registration Form before contacting us or Pheasant Run. There will be a special card for arrival and accommodation information for Pheasant Run. Please DO NOT contact the hotel until you have this card. This card is to help the personnel at Pheasant Run make your arrival as smooth as possible.

Pheasant Run
RESORT

BUGGIN' HUG

Corrections To "The Single Step Approach To Recovering Deleted Files" in Issue #45

Dear HUG,

I would like to thank you for publishing my article "The Single Step Approach To Recovering Deleted Files". It really does make me feel good to see it in print.

There were some differences between my listing and the one that appeared in REMark Issue 45 that will make the program work differently than I had intended. In fact, it will probably fail to run properly at all. I will list the differences below.

1. My label "syntax" should follow the "jnz chkarg" so that the example of proper syntax will be displayed if no arguments are provided. The label "exit" should follow immediately thereafter so that the program can be entered with the proper command line.

2. The label "cfu" is misplaced and the rest of that subroutine ended in my subroutine labeled "continu" where a "scall .close" and a "jmp exit" should be. The result of this is an error message and exit when "cfu" is called (a failure every execution). Also, confusion and no close of the restored file at the end of the "continu" routine, although that will never be reached.

3. I misspelled the word "Usage" in my syntax message.

C. F Webber
34 Mills Street
Morristown, NJ 07960

COLD HUG Now Has Bulletin Board System

Dear HUG,

Please place a notice in your magazine stating that COLD HUG now has a Bulletin Board System. The hours are 9 p.m. to 7 a.m. (Alaska Time Zone). The phone number is 907-895-3284. Other hours are by prior arrangement only. System contains useful programs for CP/M & HDOS written in BASIC and Assembler and soon will have some Apple programs as the majority of computers in this area are Apple/Franklin.

Stan Lockhart
P. O. Box 229
Fort Greely, AK
APO Seattle, WA 98733

H8's Wanted

WANTED — H8's. I will pay up to $150.00 per unit. My phone number is 212-380-1004, ask for Gerald Pindus or write to the address below.

Gerald Pindus
78-40 164th Street
Flushing, NY 11366

SMHUG NEWS

Dear HUG,

Effective with the January 14, 1984 meeting, the Southwest Michigan Heath Users' Group (SMHUG), will meet on the second Saturday of the month at 10:00 a.m. in Room 4010D of the main building of the Kalamazoo Valley Community College. The new contact person is Bob Hamel, 1054 Blanchard S.W., Wyoming, MI 49509. The phone number is 616-532-3875.

A. Robert Hamel
Editor, SMHUG News
1054 Blanchard S.W.
Wyoming, MI 49509

BOOM BOOM!

Dear HUG,

Have you ever watched a two year old playing with blocks? First he (or she, but hereafter "he" for clarity—and to save my poor fingers the extra writing), what was I saying? Oh, yes. First, he builds a tower, block by carefully placed block, teetering this way and that, until it reaches the sky (or so it seems to him). He stands back to admire his creation, sucking his thumb in silent wonder. Then with a wild squeal of glee, he rushes forward to kick, knock asunder, and otherwise demolish the short-lived tower.

My husband was in a two year old frame of mind the other day. *(I was not!—Bob.)* I was struggling with a computer program for Data Structures class when he sneaked up behind me, tapped me on the shoulder, and grinned into my unenthusiastic face. "Forget the homework!" he crowed. "Do you want to see a computer program that blows itself up?" Naturally, I thought he was referring to **his** attempts to solve the Data Structures problem — we are taking the same course, you see. The thing that bothered me was the fact that he seemed so **cheerful** about the failure of his program. Perhaps he was finally going off the deep end?

"This is something I have to see!" was my mumbled reply as I left my drudgery with combined relief and dread. (Bob's programs always work, eventually. Why is he so excited now?) He ushered me into the computer sanctum and politely pulled out my chair, something I've been trying to get him to do in restaurants for some time now. Without another word, he pointed to the screen, which said "RUN" — my cue to hit the carriage return. I did.

The screen cleared and the word "B O O M" appeared, dead center. I looked at Bob in puzzlement. He told me to list the program. I tried, it wasn't there. It had self-destructed. Imagine that: a computer program that blows itself up so completely that you have to rewrite it to run it again. If you ever feel like a bored two year old in

```
1180 '***************************************************************
1190 '                                                              *
1200 'PS: JEAN DIDN'T KNOW IT AT THE TIME, BUT THIS PROGRAM SAVES   *
1210 'ITSELF TO DISK AS A FILE CALLED "TEMP.DAT"                    *
1220 '                                                              *
1230 '***************************************************************
1240 '
1250 READ N
1255 N=INT(SQR(N*8)+.1)
1260 FOR I=1 TO N
1270    READ A
1275    A=INT(SQR(A*8)+.1)
1280    PRINT CHR$(A);
1290 NEXT I
1300 READ N
1305 N=INT(SQR(N*8)+.1)
1310 FØR I=1 TO N
1320   READ P
1325   P=INT(SQR(P*8)+.1)
1330   P$=P$+CHR$(P)
1340 NEXT I
1350 SAVE P$
1360 NEW
2000 DATA 21.125, 91.125, 595.125, 91.125, 990.125, 231.125, 561.125
2010 DATA 544.5, 128, 780.125, 128, 780.125, 128, 741.125
2020 DATA 8, 882, 595.125, 741.125, 800, 264.5, 578
2040 DATA 528.125, 882
```

need of something to do (or undo), here's the listing for you to play with.

Jean (and Bob) Hall
Rt. 10 Box 478B
Moore, OK 73165

Comments To Two Previous Buggin' HUG Letters

Dear HUG,

I just finished reading issue 45; another great issue! I am writing to comment on two of the letters in the Buggin' HUG column.

To Mr White: HDOS absolutely, positively DOES read NULLs from the disk. A NULL is nothing more than 8-bits of binary zero. If HDOS couldn't read binary zeros from the disk, then it would be next to impossible to run assembler programs. For example, the following assembler instructions generate binary zeros:

```
Instruction      Octal Representation

LXI    H,10      041 012 000
MVI    B,0       006 000
SCALL .EXIT      377 000
```

I believe the problem with NULLs that Walt was referring to is that the TT: device driver deletes NULLs entered from the keyboard. This is why PIE can't process them. If you have a copy of the HDOS 2.0 source listings, check the first listing in Volume 1, page 84, around line 3877; this code shows how NULLs and Line Feeds are deleted by TT: interrupt service.

To Mr. Bronosky: Yes, it is possible to eliminate labels with relative addressing, but it's not a good habit to get into. There must be a bunch of good reasons to not use relative addressing in JMP instructions, two come to mind right now.

First, you can't count on the assembler to catch type-o's for you. For example....

```
MOVE    EQU     *
        MOV     A,M
        STAX    D
        INX     H
        INX     D
        DCR     B
        JNZ     MOBE
```

....would generate an error because 'MOBE' is an undefined label, while ....

```
MOV     A,M
STAX    D
INX     H
INX     D
DCR     B
JNZ     *-6
```

....would not be caught by the assembler at all, and would most certainly cause hard-to-catch run-time errors (it should be *-5).

Second, consider the maintenance nightmare you would have if you decided to insert a CALL to a routine to convert lowercase characters to uppercase characters after the MOV A,M instruction. You'd have to remember to add three to the relative jump address. This type of house keeping is best left to the assembler to do for you.

Don't worry too much about running out of label space in the assembler. The largest assembler program I have written on my H8 is 151 pages long and just littered with labels. There are 1682 labels defined, and I would guess that in my system there is room in the symbol table for 600-700 more.

David A. Shaw
469 N. Howard
Elmhurst, IL 60126

Changes To Morse Code Program

Dear HUG,

I have amended my copy of Robert Horn's Morse Code program and found the changes to be correct but incomplete. Make the following additional changes to complete the corrections.

Change line 440 to read:

```
440 FOR T=1 TO N1:NEXT T
```

Change line 450 to read:

```
450 FOR T=1 TO N2:NEXT T
```

Insert line 455 as follows:

```
455 NEXT J
```

Change line 460 to read:

```
460 FOR T=1 TO N3:NEXT T
```

I am using CP/M MBASIC 80 Rev. 5.21. Using Dr. Milam's changes with the changes listed above, the program runs well. I have additional changes for those people who are working on the higher class licenses. The following two lines will allow you to run the program at approximately 18 words per minute code speed.

Add the following:

```
125 PRINT "D---> XFAST ( 18WPM ) "
175 IF CH$="D" OR CH$="d" THEN PRINT
    "XFAST":PRINT:N1=20:N2=70:
    N3=320:N4=1:GOTO 200
```

Are there any printing or other errors in the revised CHEAPCALC program in the article "CHEAPCALC Another Look" on page 35 of Issue 44? I have been unable to get it to run. I have had the previous CHEAPCALC partially running but some of the functions did not work.

I am also looking for anybody who is interested in using their computer for Astronomy (specifically optics and the Amateur Space Telescope) for an exchange of information and programs.

I think you are doing a great job with REMark. Keep up the good work.

James Hauser
29732 Taylor
St. Clair Shores, MI 48082

Feedback On the Updated MAPLE

Dear HUG,

The same month that my letter containing comments on MAPLE (Modem Applications Effector) was printed (Oct. '83), an updated version was offered through HUG. I'm writing to give my initial impressions of the updated version.

First I want to say that the items which caused frustration for me in the first version have been effectively dealt with. Dr. Parke has done a fine job of taking off the rough edges present in the first version. With the prior version, the keypad was left in the alternate keypad mode when MAPLE was exited. The new version allows you to select which mode the terminal will be put on an exit from MAPLE. No more problems going directly to another program which expects the keypad in a different mode. Another nice feature is the ability to send a file with CR-LF pairs, just line feeds or just CR's to designate the end of lines. No longer are double spaced files a problem when sending files to the local campus computer system.

The new version has other expanded communications options, a nicer disk directory (including size of files), and the ability to control MAPLE from a remote system. To get a good feel for the expanded abilities of MAPLE you need to read the manual, which has also been improved. Thanks to HUG and Dr. Parke, it was well worth the $10 update fee.

Daniel Gilbertson
24 N. Fourth St.
P. O. Box 158
Platteville, WI 53818

Problems With The LPMX80 Driver

Dear HUG,

I recently upgraded my MX80 to Graftrax and encountered the problem described by Mr. White in REMark Issue #45; the unwanted 'G' or 'H' in the first position of output upon opening the device.

I cured the problem by issuing the SET LP: commands to specify PAGE 66 and LENGTH 66. The escape sequence generated: <ESC> C 0 apparently has the zero byte dropped on output, and the following <ESC> G or H to set or reset. DOUBLE

STRIKE has the <ESC> used as form length and the G or H then prints.

There are other problems with the LPMX80 driver if you attempt to use 8 line-per-inch output, which I use for assembler output of software programs.

1. The SET processing for PAGE and LENGTH rejects as INVALID any values over 66. Standard 11 inch paper has 88 lines.
2. The SET LP: LPI 8 has no affect.

I changed the compare constants for PAGE and LENGTH to accept 88 as valid, but this had no affect on the output.

To correct this situation, I created a patched version of LP: specifically for COMPRESSED, 8 LPI printing the following:

1. The <ESC> 2 for 6 LPI was changed to <ESC> 0.
2. The constant moved into the <ESC> C LENGTH sequence was changed to octal 130.
3. The <ESC> F sequence was patched to 000,017 to set COMPRESSED mode printing.

I have not included specific patch commands for these changes, because I made them using the HUG dump program.

Robert C. Mann
11260 Alger
Warren, MI 48093

Answer To Letter On 'C'

Dear HUG,

Concerning Mr. Pepper's article, "Using 'C' For Fast Action Games", in REMark Issue #42, and Mr. Gillies letter, "More on 'C' Language", in REMark Issue #44, I would like to make a small suggestion. Instead of entering in the program non-printing characters, such as the ESCape character, which may cause trouble when printing, editing and/or compiling, it is better to use the 'C' language conventional escape sequences. These are described in appendix A, section 2.4.3 of "The C Programming Language" by Ritchie and Kernighan, and in section 7.6 of the "Manual for C/80, Version 2.0".

Thus, instead of using the statement

```
printf("[E[x5[x1");
```

to clear the screen, erase the cursor and enable the 25th line, where [is PIE's representation of the ESCape character, one could use

```
printf("\033E\033x5\033x1");
#define ESC '\033'
printf("%cE%cx5%cx1", ESC, ESC, ESC);
```

Here, the backslash "\" introduces the escape sequence, and 033 is the ASCII code (in octal) for the ESCape character.

Dr. W. Luis Mochan
Instituto de Fisica, UNAM
Apdo, Postal 20-364
01000 Mexico, D. F.
MEXICO

PeachCalc Patch

Dear HUG,

Since my recent purchase of an H-100 system and of the PeachText 5000 program package, I can heartily endorse the recent article and letters praising both. However, Peachtree Software, like many other houses, pursues a policy of minimal support which is primarily to their advantage, not the users'. This manifests in the 90-day support service which comes with the package: questions are readily and rapidly answered for each individual user, but the problems/answers of other users are not generally distributed. Thus, each user can have software with severe 'bugs', but if he has not yet utilized a particular feature, their existence remains undiscovered. 'Time bomb' is a much used, but accurate description.

I would be very much in favor of a HUG forum to disseminate purchased software package changes/patches resulting from each users' experiences. It should be to everyone's advantage (including software vendor!) and should, perhaps, be addressed separately from the general letters column. As a start, the following defines a PeachCalc patch.

Program: PeachCalc, Version 1.01
Problem: Unable to Save/Load spreadsheet by Values, as defined in documentation.
Symptoms: Loading by values (while combining spreadsheets) obtained only partial copy of original values, and often resulted in a spreadsheet with many incorrectly blank entries.
Action: Called Peachtree, and was immediately given a patch. Apparently others have had the problem, and the information was readily available to the Peachtree software expert. Was told that problem occurred randomly, as a function of session history and file activity. True!
Solution: Modify two program modules (PC.OVL and PC.PGM) using DEBUG utility.
Setup: 1) Place ZDOS disk in drive A. 2) Place PeachCalc disk in drive B.

Patch #1:

```
DEBUG B:PC.OVL <cr>
E2C9 <cr> (address location 2C9 hex)
9F.50 <cr> (change value 9F hex to 50 hex)
W <cr>
Q <cr>
```

Patch #2:

```
DEBUG B:PC.PGM <cr>
E4035 <cr> (goto address 4035 hex)
08 C0 F9 74 <cr> (change data to that shown)
W <cr>
Q <cr>
```

All told, the Peachtree people were most helpful once they returned my calls, and the changes worked perfectly. Now, if only the EDIT and PRINT functions utilized a 'current file' rather than requiring constant re-entry of the same file name during composition and checking of text files....

If anyone else has more data, please let us all know. Eventually we will be able to debug the entire program package!

Richard A. Pabst
18 McAdams Road
Framingham, MA 01701

Help Needed To Get PeachText 5000 To Run On an H29

Dear HUG,

First, let me pass out the flowers. I just received my second H/Z-100. I had built some major Heathkits in the past and Heath's technical literature and factory backup are super. Before buying a PC, I studied as many systems as I could and finally chose Heath/Zenith for service, economy, expandability, quality, and compatibility. I was not disappointed!

I bought my first one in July for my small business and took it home to get acquainted with. My wife (a writer) and twelve year old son soon let me know there was no way I could be inconspicuous about taking it to my office. So three months later, my Two Year Plan for a second system is here. My wife became permanently hooked on the PeachText 5000 and my son is a real ZBASIC fan.

I also bought an H29 terminal for my secretary to use. I finally got the H29 tied to my new H/Z-100 using the SWAP program from REMark Issue #38 by Marc Aagenas.

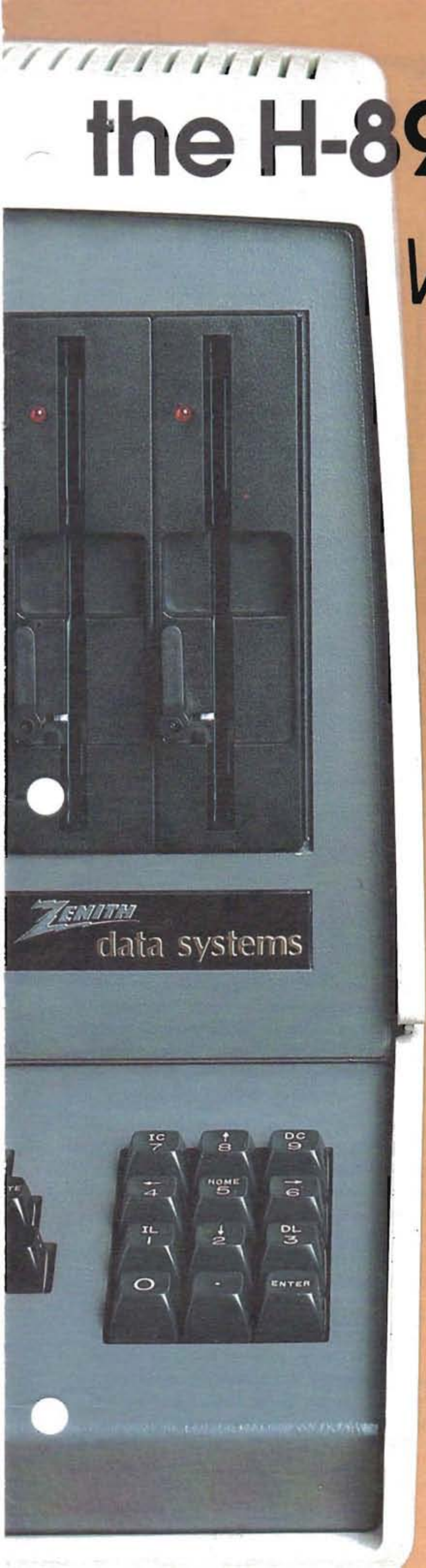
I had to overcome a few problems getting the terminal system operating. I am using Condor file manager and it works great for our production control, material control, and literature files control. Also the PeachCalc works fine by using CTL U, R, L, & D for cursor movement.

I have one remaining problem: The PeachText won't work with my H29 because the function keys send an **ESC**ape <letter> code to the computer and the PeachText responds with the escape function instead of the "F" function. I called both Peach and Heath and they say they don't

Vectored to 65 ☞

NEW FROM FLOPPY DISK SERVICES, INC.

the H-89 TWOET SYSTEMS



We've got a great idea for your H-88, 89 or 90. It's a dual *internal* half height drive system. Two of our half height 5¼″ drives can replace your built-in disk drive, doubling your information storage capacity.

Floppy Disk Services provides you with everything you need. That's two double-sided, double or single density, half height drives in either 48 or 96 tpi format, all hardware, cables and power connector adaptors. And most important, you get easy, step-by-step instructions, in the Heath/Zenith tradition of good, clear documentation.

We've thoroughly tested the TWOET/Heath set-up. Remember that a double sided 48 tpi will work perfectly as a single sided drive right out of the box! Hard or soft sectored—so you can even use this system with your H-17 controller. And of course we have the software drivers (additional cost) to run 48 or 96 tpi double sided, single density drives on the H-17.

Model TWOET 455
2 Shugart SA-455 half height
48 tpi double sided
All hardware
Metal, shielded mounting plates
Data cable with chassis connector
Power 'Y' connector
Complete documentation
Price .... $605.00 complete

Model TWOET 465
2 Shugart SA-465 half height
96 tpi double sided
All hardware
Metal, shielded mounting plates
Data cable with chassis connector
Power 'Y' connector
Complete documentation
Price .... $755.00 complete

Wondering what to do with your internal drive if you buy this system? Here's the solution. If you purchase a dual half height system for your Heath computer from Floppy Disk Services, just include an extra $50.00 plus shipping and receive a single 5¼ case with power supply *and* data cable ready to receive your SIEMENS internal drive! The case with data cable is normally a $70.00 item. And the cable that comes with your TWOET system includes the external chassis disk I/O connector.

Due to production deadlines, prices in this ad are 2 months old, so we encourage you to call us for current prices and new product info. Prices and specs subject to change without notice.

Dealer inquiries invited.

PAYMENT POLICY — We accept MasterCard, VISA, personal checks and Money Orders. We reserve the right to wait 10 working days for personal checks to clear your bank before we ship. All shipping standard UPS rates plus shipping & handling. NJ residents must add 6% tax.

Toll Free Order Line:
(800) 223-0306
Tech Help or Info:
(609) 799-4440

H-88/89/90 are registered trademarks of Heath Corp.

FLOPPY DISK SERVICES™ INC.

741 Alexander Rd. Princeton, NJ 08540

A Tutorial On Random Numbers and The Random Function

Kenneth Mortimer PE
352 Green Acres Drive
Valparaiso, IN 46383

Random Numbers — What Are They?

Everyone who has played a game of cards, dice, or a board game has used a random number generator. When one casts a pair of dice, one never knows (if the dice are honest - and they had better be or I'll send some of my friends to break your leg) what side will come up. In other words, a die is a random number generator capable of generating in random sequence the integers 1, 2, 3, 4, 5, and 6. Similarly when one draws a card from a deck of ordinary playing cards, one is not sure what card he will draw. A deck of "bridge" cards is therefore a random generator capable of generating randomly and one of 52 values in a rather peculiar numbering system. When one participates in a lottery or drawing, the cards or balls and the drum make up a much larger random numbering system.

Since these historic random number systems are used in so many games, it is not unusual that a random number generator of some sort should be used in making the random decisions required in a computer game and even in more serious business or scientific simulation. Fortunately for those of us who play with computers there is a random number generator built into most BASICs. It is the RANDOM FUNCTION and you might want to read a bit about it in your manual before you proceed further.

The Random Function

In most BASICs there is a Random function generator and most of them have the same basic form. (Not all, but both the Benton Harbor BASIC and the Microsoft BASIC follow the same format.) The random number function in most eight bit BASICs is of the form:

RND(narg)

The function returns a pseudo random number which generates a random number by some manipulation of a seed or the previous random number. The algorithms for generating the pseudo random (this is the last time I will be accurate and use the word pseudo) are subject to much discussion among computer scientists and mathematicians. The argument (narg) provides both a seed and a control number for the Random function generator. If narg is a positive real number, the function returns the next number in the series of random numbers. If narg is equal to zero, the function returns the last previously generated random number. If narg is a negative real number, the function uses that number as a seed to start a new sequence of random numbers. This is all in your manual but let us write a simple program to ask for the seed and/or control number and write the output of the random number generator on the screen. This program is listed as RNDTEST1.BAS and should be self-explanatory. The output is short enough so that the results of three run throughs will be displayed on the screen at one time.

```
00010 REM RNDTEST1.BAS   A TEST PROGRAM FOR THE RANDOM NUMBER GENERATOR
00020 REM PRINTS OUT A LIST OF 20 RANDOM NUMBERS RESULTING FROM A SEED S
00100 REM S=SEED
00105 REM S1=CONTROL NUMBER FOR REPEATED RANDOM NUMBERS
00110 REM R=RANDOM NUMBER
00120 REM I=COUNTER
00200 INPUT "ENTER SEED NUMBER  ";S
00210 INPUT "ENTER CONTROL NUMBER FOR LATER RANDOM NUMBERS  ";S1
00220 R=RND(S)
00240 PRINT "FIRST RANDOM NUMBER  ";R
00330 FOR I=1 TO 5
00340 R=RND(S1)
00350 PRINT R
00360 NEXT I
00370 GOTO 200
```

RNDTEST1.BAS

Enter this program into your computer and run it. When the computer asks for the seed number enter 1. When the computer asks for the control number for the later (succeeding) random numbers, again enter 1. The terminal will now display six different numbers that are larger than 0.000000 and less than 1.000000. Repeat the process and you will see six more random numbers. Repeat the process with any positive number and the same thing will happen.

Now enter a positive number when the computer asks for the seed number and zero when the computer asks for the control number for the later random numbers. This time the terminal will display the same number six times. Repeat this process and you will find that the zero control number will always cause the function to return the previous "random number". This is a very handy feature for use in testing programs because you will always know that you will be repeating the last previous "random number".

This time enter a negative number when the computer asks for the seed number and a positive number when the computer asks for the control number. You will see six different numbers on the screen. Rerun the program using the same negative number as a seed and a different positive number as the control number for the later random numbers. You should see the same set of six numbers on the screen. The numbers entered may be integers or real (decimal) numbers. Repeat the process with a different seed.

You should get a new set of six "random" numbers which you can repeat. I say you should get a new set of random numbers but the way that the algorithm uses the seed may cause two different seeds to generate the same "random" number. My experience has been that any seed number that is twice the previous number will return the same random number in Benton Harbor BASIC. Any integer seed will return the same random number in Microsoft BASIC. This now gives you an even more elaborate method of generating either a single random number or a series of random numbers for testing.

Continue to play with this program until you are confident that you know how the control or seed number works.

Testing For Linearity

When one rolls a die, the probability that one will roll a three is one in six. When one draws a card from a bridge deck, the probability of drawing an ace of spades is one in fifty two. The probability of drawing any card is exactly the same. If the random number generator is to be used in most forms of gaming and simulation, there should be exactly the same probability of any number being returned by the function. How can you test the random number generator for linearity? It would take a long time and a great deal of storage to check for the generation of all of the random numbers within the range. A simpler method would be to generate one thousand random numbers, divide them into ten ranges and see if the number of random numbers in each range is approximately the same. The method used will also serve as an introduction to converting the output of the random number into a series of random integers. What we would like to do is count the number of values returned that fall between .000000 and .099999, between .100000 and .199999, etc.

```
00010 REM RNDTEST2.BAS A BASIC PROGRAM TO TEST THE LINEARITY OF THE
00020 REM RANDOM NUMBER GENERATING SUBROUTINE BY DEVIDING THE RESULTING
00030 REM OUTPUT INTO TEN EQUALY SPACED GROUPS
00040 REM FOR 1000 RANDOM NUMBERS
00100 REM S=SEED NUMBER
00110 REM R=GENERATED RANDOM NUMBER AND ITS MODIFICATIONS
00120 REM S1= SUM OF ORIGINALY GENERATED RANDOM NUMBERS
00130 REM M=ARITHMETIC MEAN OF GENERATED RANDOM NUMBERS
00140 REM G(I)=NUMBER OF GENERATED RANDOM NUMBERS IS THAT DECILE
00200 DIM G(12)
00210 REM ZERO SUM
00220 S1=0
00230 REM ZERO ALL GROUP COUNTERS
00240 FOR I=0 TO 12
00250 G(I)=0
00260 NEXT I
00270 INPUT "ENTER SEED NUMBER  ";S
00300 REM GENERATE AND SORT THE NUMBERS
00310 FOR I=1 TO 1000
00320 R=RND(S)
00330 S1=S1+R
00340 R=INT(R*10)
00350 G(R)=G(R)+1
00360 NEXT I
00400 REM PRINT OUT RESULTS
00410 M=S1/1000
00420 PRINT "ARITHMETIC MEAN OF DISTRIBUTION = ";M
00430 FOR I=0 TO 12
00440 PRINT I,G(I)
00450 NEXT I
00460 END
```

RNDTEST2.BAS

Program RNDTEST2.BAS has been written to do this checking. If you multiply the output of the random number generator by ten you will get a number greater than .000000 and equal to or less than 9.99999. If you take the integer value of that result, you will get an integer value in the range between and including 0 and 9. This is done in statement number 340 of RNDTEST2. In this program I have used the output of statement 340 as an index of a vector G. Every time a number between .10000 and .199999 is generated, the value of G(1) is increased by 1.0. In a lineal distribution between zero and one, the average (arithmetic mean) should be 0.50000. Let us see how the program works.

1) Set the sum of the random numbers to be used to determine the mean at zero (Stmt. 220).

2) Set each element of the vector G to be used to determine the distribution to zero (240 to 260).

3) Enter a seed or control number (270).

4) Repeat 1000 times (310 and 360).
   4a) Generate a random number (320).
   4b) Add that number to the sum of the random numbers (330).
   4c) Convert the random number to an integer (340).
   4d) Add one to the appropriate vector element (350).

5) Compute the average (410).

6) Print out the average (420).

7) Print out the number of values in each range (430 to 450).

8) End of program (460).

Note that three elements of the vector were included to check if a random number equal to or greater than one was generated.

Run the program and see how linear the distribution actually is.

Using the Random Number Generator

There are many ways to use the random number in simulation or gaming. One of the simplest is the comparing the generated random number with a known or assumed value. Let us assume that we are trying to simulate a baseball game. Our batter has a batting average of .234. From what we have seen so far, the probability of the random number generator returning a number less than .234 is 23.4 per cent. This is the same as our batter's probability of getting a hit. Therefore, if the random number generated is equal to or less than .234, we can say that the batter got a hit, if it is greater he is out. A BASIC program element could be written thus:

```
1000 REM HITTING SUBROUTINE
1010 H = RND(1.11)
1020 IF H > .234 THEN GOTO 1200
1030 REM SUBROUTINE TO HANDLE HIT
1040 -------------
1050 -------------

1090 RETURN
1200 REM SUBROUTINE TO HANDLE OUT
1210 --------------
1220 --------------

1250 RETURN
```

Of course you would want to have a much more complex program that would take into account each pitch and what would happen after each hit and that of course will result in a program as complex and taking into account as many factors as you feel your game justifies.

If you know the probability of any event happening, you can generate a simulation program involving that event.

We previously converted the output of the random number generator to a series of integers from 0 to 9 by multiplying the output of the random generator by ten and taking the integer of the product. If we had wished to generate a number from one to ten, we would have had to add one to our result. If we want to generate an integer from 1 to N, we should multiply the output of the random number generator by N, take the integer or the product and add one to the result. A program element to simulate an ordinary six sided die is as follows.

```
2000 REM SUBROUTINE FOR SIMULATING A SINGLE DIE
2010 N = RND(S)
2020 N = INT(N*6)+1.0
2030 RETURN
```

The random integer can be used in many ways in BASIC programs. Some are:

1) As one would use dice in any game.
2) As the index of a computed GOTO.
   ON N GOTO 2000,2500,2700,2200,4000
3) As the index of a computed GOSUB.
   ON N GOSUB 123,456,789,999
4) As the index in the Line Number Function LNO(iexp).

As you can see, the random number generator is a useful tool in generating random numbers or random events for gaming and simulation.

Normally Distributed Random Numbers

Some of the more mathematically sophisticated of you will now argue that there are many instances where the results of a random occurrence will not be linearly distributed. A good example of this might be a target game of some sort. If the player's aim is good, the probability of the projectile hitting the center of the target is high and the probability of the projectile missing the target completely is very small. The distribution of the hits about the center of the target would be "normal" or distributed according to the Gaussian or "bell" curve. It would be handy to have a subroutine that would change the linearly distributed random number to a normally distributed random number. One method is suggested in the National Bureau of Standards Handbook of Mathematical Functions (AMS 55) on page 953. The algorithm consists of generating two linear random numbers (R1 and R2) and manipulating them according to the equations:

N1 = (SQR(-2*LOG(R1)))*COS(2.0*P*R2)
N2 = (SQR(-2*LOG(R2)))*SIN(2.0*P*R1)
where N1 and N2 are each normally distributed random numbers with a mean of 0.0 and a standard deviation of 1.0.
SQR,LOG,SIN and COS are the regular basic functions.
P = pi = 3.1415926

Program RNDTEST3.BAS demonstrates the generation of normally distributed random numbers and RNDTEST4.BAS tests the normality of the values generated.

```
00010 REM RNDTEST3  A PROGRAM TO DEMONSTRATE THE GENERATION OF NORMALY
00011 REM    DISTRIBUTED RANDOM NUMBERS
00020 REM R1 = FIRST LINEAR RANDOM NUMBER
00025 REM R2 = SECOND LINEAR RANDOM NUMBER
00030 REM N1 = FIRST NORMAL RANDOM NUMBER
00035 REM N2 = SECOND NORMAL RANDOM NUMBER
00040 REM S = SEED FOR VERY FIRST RANDOM NUMBER
00050 REM I = INDEX
00060 P=3.1415926                :REM PI
00100 REM BASIC PROGRAM
00110 INPUT "ENTER SEED FOR INITIAL RANDOM NUMBER   ";S
00120 R1=RND(S)
00130 FOR I = 1 TO 20
00140 GOSUB 1000
00150 PRINT I,N1,N2
00160 NEXT I
00170 END
01000 REM SUBROUTINE FOR GENERATING NORMALY DISTRIBUTED RANDOM NUMBERS
01010 R1=RND(1)
01020 R2=RND(1)
01030 N1=(SQR(-2*LOG(R1)))*COS(2*P*R2)
01040 N2=(SQR(-2*LOG(R2)))*SIN(2*P*R1)
01050 RETURN
```

RNDTEST3.BAS

```
00010 REM RNDTEST4 A PROGRAM TO CHECK THE NORMALITY OF NORMALY DISTRIBUTED
00011 REM    RANDOM NUMBERS
00020 REM R1 = FIRST LINEAR RANDOM NUMBER
00025 REM R2 = SECOND LINEAR RANDOM NUMBER
00030 REM N1 = FIRST NORMAL RANDOM NUMBER
00035 REM N2 = SECOND NORMAL RANDOM NUMBER
00040 REM S = SEED FOR VERY FIRST RANDOM NUMBER
00050 REM I = INDEX
00060 P=3.1415926                :REM PI
00070 DIM Z1(10),Z2(10)                       :REM MATRIX OF DISTRIBUTION
00080 REM S1 AND S2 ARE THE SUM OF THE PAIRS OF GENERATES
00100 REM BASIC PROGRAM
00110 INPUT "ENTER SEED FOR INITIAL RANDOM NUMBER   ";S
00120 R1=RND(S)
00130 FOR I=1 TO 6
00140 Z1(I)=0
00150 Z2(I)=0
00160 NEXT I
00170 S1=0
00180 S2=0
00190 REM TABULATE AND AVERAGE
00200 FOR I=1 TO 1000
00210 GOSUB 1000
00215 PRINT I
00220 S1=S1+N1
00230 S2=S2+N2
00240 J=INT(ABS(2*N1))
00250 Z1(J)=Z1(J)+1
00260 J=INT(ABS(2*N2))
00270 Z2(J)=Z2(J)+1
00280 NEXT I
00282 S1=S1/1000
00284 S2=S2/1000
00290 PRINT "SUMS",S1,S2
00295 PRINT "Z","Z1","Z2"
00296 Z1=0
00297 Z2=0
00300 FOR I=0 TO 6
00310 J=I/2+.5
00311 Z1=Z1+Z1(I)
00312 Z2=Z2+Z2(I)
00313 Z3=(Z1+Z2)/2
00320 PRINT J,Z1,Z2,Z3
00330 NEXT I
00340 END
01000 REM SUBROUTINE FOR GENERATING NORMALY DISTRIBUTED RANDOM NUMBERS
01010 R1=RND(1)
01020 R2=RND(1)
01030 N1=(SQR(-2*LOG(R1)))*COS(2*P*R2)
01040 N2=(SQR(-2*LOG(R2)))*SIN(2*P*R1)
01050 RETURN
```

RNDTEST4.BAS

Most of you will never worry about the normally distributed random number and those of you who know enough statistics remember that the normal distribution with a zero mean and a unit standard deviation (Z form) can be converted to a normal distribution with a mean M and standard deviation S by the relationship

N3 = M + S * N1

where N1 is the Z form normally distributed variable and N3 is the shifted normally distributed variable.

Study this material carefully and you should be able to add some valuable tricks to your programming toolbox.

Implementing Heath Escape Sequences for CP/M and Z-DOS

William M. Adney
4821 Sunnybrook
Buena Park, CA 90621

If you read other microcomputer magazines regularly, you probably noticed that a well known columnist was having trouble "getting rid of the keyclick" on his new Z-100. Since it's not immediately obvious to a new Heath/Zenith user how this is done, we will take a look at implementing the Heath escape sequences using assembly language programming.

In this month's column, I have provided two short assembly language programs (one for CP/M and one for Z-DOS) which can be used to implement the Heath escape sequences. I have also included a program for CP/M which will display a 25th line ruler display for Magic Wand. Assembler commands for each program are included in each listing so that you can quickly implement these programs.

One of the best features of the Heath/Zenith line is the fact that most of the technical and programming information is available in their documentation. As a matter of fact, CP/M-85, Z-DOS, and the Z-100 are so well documented that it is sometimes difficult to find an answer to a specific question. For that reason, it is extremely important that you review the Table of Contents for each of your application programs, and it's a good idea to take a few minutes to scan each of the sections to get a feel for the overall contents.

Finding the Information

The Heath escape sequences are documented on page 11-10 of the H-89 manual. For the Z-100, the escape sequences are listed in Appendix B (page B.14) of the Z-100 User's Manual and page 10.38 of the Z-100 Technical Manual. Detailed descriptions of the escape sequences begin on page 10.42 of the Z-100 Technical Manual, if you need them to write your own programs.

Developing the Programs

One of the best techniques for programming is to write the code so that it is modular and can be used for several purposes. Since there is no reason to waste disk space with a small ASM program which can be coded to provide multiple functions, I will only show a general program which can clear the CRT display using the "ESCAPE E" function (See Listing 1). Notice that the first line under "MAIN:" in the listing moves the "CLS" line to a register, and the program will clear the screen. If you want to use this general program to turn the keyclick off, change the "CLS" to "COFF". Then change the file name from C.ASM to CLICKOFF.ASM, and modify the assembler commands accordingly. You can even control the form feeds on a printer (e.g. H/Z-25) by implementing the "FORMF" line and creating FF.COM. Additional escape sequences are included in the Heath Escape Codes section, and these may be implemented by adding the appropriate escape sequences.

```
****************************************************************
```

Listing 1

```
;Clear Screen Program for H/Z-89 and H/Z-100 terminals
;
;       For CP/M-80 and CP/M-85
;
;       This program uses the Heath ESCAPE E sequence
;       to clear the screen and home the cursor,
;
;Created:       9-30-82 by W. M. Adney
;
;Modification Date      Description
;       N/A
;
BDOS    EQU     0005H
ESC     EQU     1BH     ;Escape function
;
        ORG     100H
MAIN:   LXI     D,CLS   ;Load Heath clear screen function
        MVI     C,9     ;CP/M print string function
        CALL    BDOS    ;Call CP/M
        RET             ;Return to CP/M
;
;******************************
;                             *
;       Heath Escape Codes    *
;                             *
;******************************
;
CLS:    DB      ESC,'E','$'     ;Clear Screen--Escape E
CRTR:   DB      ESC,'z','$'     ;Reset to power up configuration--Escape z
COFF:   DB      ESC,'x','2','$' ;Key click off--Escape x 2
KSFT:   DB      ESC,'x','6','$' ;Keypad shifted--Escape x 6
BCUR:   DB      ESC,'x','4','$' ;Block Cursor--Escape x 4
HOLD:   DB      ESC,'[','$'     ;Hold screen mode--Escape [ (H-89 only)
                                ;Allows use of H-89 scroll key
                                ;Page 11-14 of H-89 operation manual
NBLK:   DB      ESC,'x',';','$' ;Nonblinking cursor--Escape x (H-100 only)
FORMF:  DB      0CH,'$'         ;Printer form feed(e.g. H/Z-25)
;
        DS      014H    ;Reserve space for 10 entries
STACK:  DS      001H    ;Top of stack is here
BUFFER:                 ;Storage begins here
        END     MAIN    ;END OF PROGRAM
;
;
;
;       Create program with editor as C.ASM
;       All files must be on drive A
;               ASM.COM
;               LOAD.COM
;               C.ASM
;
;Use the following commands to create C.COM
;       ASM C,AAZ
;       LOAD C
;       ERA C.HEX
```

If you are not familiar with assembly language, I recommend that you format a new disk for use with these assembly language programs, and back it up after you code the source files. In some cases, typos in assembly language can cause strange things to happen, and I have wiped out the directory on a disk just because of a typo during some experimenting with a program.

The Z-DOS version of the program is shown as Listing 2. The same technique has been used, and the Z-DOS assembler commands are shown in the listing.

The same programming techniques are used to write a program for CP/M which will display a ruler line in reverse video (See Listing 3). I use this program with Magic Wand and PeachText 5000 when I want to see the column positions on the edit screen. This program can be converted to 8086 assembler (Z-DOS) code by following the example in Listing 2. If you don't like the reverse video, either delete the labeled lines or place a semicolon (;) in front of the appropriate lines of code. The assembler assumes that anything following a semicolon is a comment and does not assemble it. I recommend the use of a semicolon because it is easy to delete if you want to change a feature of the program.

Assembly Lanugage and C

For the assembly language experts who will argue that all of this code is not the most EFFICIENT way to implement these features, this is not intended to be an assembly language course. If you are interested in learning assembly programming, I can recommend the Heath Assembly Language course (EC-1108) as an excellent introduction for learning 8080 coding for CP/M-80 and CP/M-85. I have taken that course, and I believe that it is an excellent way to learn the fundamentals of assembly language programming. Although I have learned some of the 8086 assembler code for Z- DOS, I have concluded that the C programming language is the language to learn. The biggest advantage of C is that you don't have to learn a new assembler every time a new microprocessor chip is developed. In general, the C language is "portable" because the source code is supposed to be standard. C compilers can be developed for each microprocessor chip, and at least theoretically, the source code does not have to be changed. In the real world however, that doesn't seem to be true since a lot of the C compilers are "non-standard" in one way or another.

For anyone interested in the standard definition of the C language, I recommend that you read "The C Programming Language" by Kernighan and Ritchie (Prentice-Hall). Another book that I particularly like is the "C Programming Guide" by Jack Purdum (Que Corporation-Indianapolis). It has a lot of good examples plus it includes some useful programs. My favorite is the file copy program on page 183. It's a lot easier to use than PIP!

Speaking of Software

Although Heath has an excellent line of software, they obviously don't handle everything that's available on the market. For that reason, you will probably want to buy some software (e.g. WordStar version 3.30) at some point that is not usually available in the Heath stores. My favorite place to buy that kind of program is 800-Software in Berkeley, CA. In addition to discount prices, they provide technical support by telephone if you need it. And even more important, they stand behind everything they sell. It's a good place to get software that the Heath stores do not carry as a stock item.

I recently bought the WordStar Pro-Pak which contains WordStar 3.30, SpellStar, Mail Merge, and Star Index. That version of WordStar is amazing in its capabilities, and I am very pleased with it. At the risk of opening Pandora's box, it is better than Magic Wand for most of the writing that I do. I primarily use Magic Wand for programming, although it has most of the same capabilities of WordStar.

```
*******************************************************************
```

Listing 2

```
;Clear Screen Program H/Z-100 terminals
;
;       For Z-DOS ONLY
;
;       This program uses the Heath ESCAPE E sequence
;       to clear the screen and home the cursor.
;
;Created:       4-23-83 by W. M. Adney
;
;Modification Date      Description
;       N/A
;
        .XLIST
INCLUDE DEFASCII.ASM
INCLUDE DEFMS.ASM
        .LIST

PGMSEG  SEGMENT
        ASSUME  CS:PGMSEG,SS:PGMSEG,DS:PGMSEG,ES:NOTHING

ESC     EQU     1BH     ;Escape function
;
        ORG     100H

MAIN:   MOV     DX,OFFSET CLS   ;Load Heath clear screen function
        MOV     AH,DOSF_OUTSTR  ;Z-DOS print string function
        INT     DOSF_FUNC       ;Call Z-DOS to print message
        INT     DOSI_TERM       ;Return to Z-DOS
;
;********************************
;                               *
;       Heath Escape Codes      *
;                               *
;********************************
;
CLS:    DB      ESC,'E','$'     ;Clear Screen--Escape E
CRTR:   DB      ESC,'z','$'     ;Reset to power up configuration--Escape z
COFF:   DB      ESC,'x','2','$' ;Key click off—Escape x 2
KSFT:   DB      ESC,'x','6','$' ;Keypad shifted--Escape x 6
BCUR:   DB      ESC,'x','4','$' ;Block Cursor--Escape x 4
HOLD:   DB      ESC,'[','$'     ;Hold screen mode--Escape [ (H-89 only)
                                ;Allows use of H-89 scroll key
                                ;Page 11-14 of H-89 operation manual
NBLK:   DB      ESC,'x',';','$' ;Nonblinking cursor--Escape x (H-100 only)
FORMF:  DB      0CH,'$'         ;Printer form feed(e.g. H/Z-25)
;
PGMSEG  ENDS
        END     MAIN    ;END OF PROGRAM
;
;
;
;       Create program with editor as C.ASM
;       All files must be on drive A
;               MASM.COM
;               LINK.COM
;               EXE2BIN.COM
;               C.ASM
;
;Use the following commands to create C.COM
;       MASM C
;       LINK C
;       EXE2BIN C.EXE.COM
;       DEL C.EXE
```

If you have any version of WordStar (or even if you don't), HUG has some new programs that really make your system hum. One of the best programs that I've seen is the KEYMAP (HUG disk #885-1230) program. It allows you to configure all of the special H-89 keys and most of the H-100 keys to do just about anything. It includes a setup version for WordStar so that all you have to do is copy it to your disk. I have used KEYMAP to duplicate the Magic Wand special function keys so that I don't have to think about them when I switch to a different word processor. You can also create a version of KEYMAP to display a disk directory or execute STAT or just about anything else you want. As a side note, I was disappointed to find out that Pat McNally's 100-Star program would not work with WordStar 3.30. After fooling around with DDT, it looks like MicroPro has changed the user patch areas and increased the size of the WS.COM program.

I will look forward to seeing an updated 100-Star program which will work with version 3.30.

If you're interested in learning the C programming language, I have found that Walt Bilofsky's (Software Toolworks) C compiler is a worthwhile investment. It's hard to beat for the purchase price of $49.95. Although it doesn't have all of the standard C features, it still is an excellent value. I have converted all of the programs in this article to C because I don't have the time to learn any more of the new assemblers.

By the way, I am always interested in new software (and hardware too!) for the Heath/Zenith computers. If you have something that would be of interest to our user community, send it to me at the above address.

Books and Hardware

When I first started working with CP/M, I found that it was very difficult to make any sense of some of the commands. Because of the frustration that I experienced at that time, I have written the FlipFast Command Guide series published by S-A Design in Brea, CA. The first book in the series includes all CP/M-80 and CP/M-85 commands for the Heath/Zenith family of computers. I have also completed the Z-DOS FlipFast Command Guide which should be available in January. For anyone who does a lot of programming, I have included appendices which contain technical information on the operating system. If you have ever accidently erased a file, you can use the ERAFIX program to help you recover the erased files.

As a part of working on these books with S-A Design, I had an opportunity to test an 8" disk drive from Floppy Disk Services. I have the dual drive slimline version which very closely resembles the Heath HS-207-42 or Zenith Z-207-42. I have really put these drives to work in the last few weeks, and their performance is excellent. We did not get an instruction sheet with the evaluation units, and it did take me a few minutes to figure out how to plug in the data cable correctly. Once they are connected properly, they work quite well with both CP/M-85 and Z-DOS. Since they are a frequent advertiser in REMark, you might want to check with them if you need some disk drives. They also advertise hard disk drives for Heath/Zenith computers...more on that if I can get an evaluation unit.

Next Month

The slowest part of any computer system is the printer, and it is especially irritating when the computer is tied up during the printing of a long file. One of the solutions is to use a hardware print buffer to reduce that time. I have received the ANGEL print buffer from Ligo Research, and we'll take a look at that impressive piece of hardware next month.

****

Listing 3

```
;25TH LINE MAGIC WAND DISPLAY FOR RULER FUNCTION
;
;       This program will clear the screen and provide a display
;       on the 25th line which displays a ruler line in reverse
;       video for the Magic Wand text processor.
;
;BY WILLIAM M. ADNEY--2-07-83
;
BOOT     EQU     0000H
BDOS     EQU     0005H
CONIN    EQU     1
CONOUT   EQU     2
PLINE    EQU     9
CR       EQU     0DH
LF       EQU     0AH
ESC      EQU     1BH
;
         ORG     0100H
MAIN:    LXI     D,CLEARIT          ;CLEAR FUNCTION
         CALL    SENDLINE           ;CLEAR THE SCREEN
         LXI     D,LINE1            ;AND DISPLAY COMMENTS...
         CALL    SENDLINE
         LXI     D,LINE2
         CALL    SENDLINE
         LXI     D,LINE3
         CALL    SENDLINE
         LXI     D,LINE4
         CALL    SENDLINE
;
;Remember the cursor position
;
         LXI     D,CURPOS           ;REMEMBER CURSOR POSITION
         CALL    SENDLINE
;
;Enable the 25th line
;
         LXI     D,ENABLE25         ;25TH LINE ENABLE
         CALL    SENDLINE
;
;Position cursor at the beginning of 25th line
;
         LXI     D,BEGIN25          ;25TH LINE BEGINNING
         CALL    SENDLINE
;
;Enter Reverse video mode
;
         LXI     D,RVIDEO           ;REVERSE VIDEO
         CALL    SENDLINE
;
;Print 25th line
;
         LXI     D,LINE25           ;25TH LINE HEADINGS
         CALL    SENDLINE
;
;Exit reverse video
;
         LXI     D,RVIDEND
         CALL    SENDLINE
;
;Set cursor to previously saved position
;
         LXI     D,CURSAVE          ;CURSOR TO SAVED POSITION
         CALL    SENDLINE
         RET
;
;*********************************
;                               *
;    SUBROUTINES                *
;                               *
;*********************************
;
SENDLINE: MVI       C,PLINE
          CALL      BDOS
          RET
;
RVIDEO:   DB    ESC,'p','$'                ;ENTER REVERSE VIDEO MODE
;
RVIDEND:  DB    ESC,'q','$'                ;EXIT REVERSE VIDEO MODE
;
ENABLE25: DB    ESC,'x','1','$'            ;25th LINE ENABLE
;
BEGIN25:  DB    ESC,59H,38H,20H,'$'        ;CURSOR-25th line--ESC Y 8 SP
;
LINE25:   DB '1...+...10....+...20....+...30....+...40....'
          DB '+...50....+...60....+...70....+...80$'
;
CLEARIT:  DB    ESC,'E','$'                ;CLEAR DISPLAY
;
CURPOS:   DB    ESC,'j','$'               ';REMEMBER CURSOR POSITION
;
CURSAVE:  DB    ESC,'k','$'                ;CURSOR TO SAVED POSITION
;
;
LINE1:  DB  'This program sets the 25th line for Magic Wand.',CR,LF,'$'
LINE2:  DB  'A ruler is displayed for convenience.',CR,LF,'$'
LINE3:  DB  ' ',CR,LF,'$'
LINE4:  DB  'To reset the CRT, use the command CRTRESET',CR,LF,'$'
        DS      014H               ;RESERVE SPACE FOR 10 ENTRIES
STACK:  DS      001H               ;TOP OF STACK IS HERE
BUFFER: EQU     $                  ;STORAGE AREA STARTS HERE
        END     MAIN
*****************************************************************
```

Simple Printer Controls For The Z-100

Charlie Layman
35 Kendall Road
Sudbury, MA 01776

The lack of printer device drivers as such for the Z-100 precludes of setting the printer to its various mode characteristics easily. For instance, the driver H25.DVD in the H/Z-89 world allows you to set any one of 8 different units to be any of the 4 character densities or 2 line densities you might want using the H/Z-25 printer. Unit 1 might be set to 10 characters per inch (CPI), unit 4 might be set to 13.2 CPI, and unit 5 might be set to 16.5 CPI with 8 lines per inch for a real dense application. Then when the driver is invoked by a particular unit number, the corresponding character density will be called.

Unfortunately it is not quite that easy on the Z-100. There are no drivers to be configured to your liking. You are more or less stuck with the DIP switch default settings unless you bury escape sequences in the file itself. When using something like Multiplan, it's no problem because you can stick the escape sequence for 16.5 CPI in the set-up field of the Print option should you want to compress your spread sheet.

But what do you do if you want to print a compressed data base listing from Condor File Management System? The escape sequence cannot be buried anywhere (at least not where I have tried). What do you do if you print a text file at, say 12 characters per inch, you don't like it and now you want to print it at 13.2 CPI? You could change the switch settings on the back of the printer or better yet, just use your favorite editor and add the escape sequence for 13.5 CPI to your file. This will then preset the printer when the file is sent out with the PRINT command from Z-DOS.

Now that to me is a big waste of time! There is a far easier method to accomplish many of these simple tasks under Z-DOS. And that is to make use of the BATCH capability. Sure, there are other ways of doing the same thing but, I think this is about as easy as you can get.

I have created 8 different batch files to do the popular H/Z-25 printer conditioning. You can certainly add to these as you see fit. The files and their contents are listed in the table which follows:

File Name		Contents
10CPI	.BAT	PRINT SET10CPI
12CPI	.BAT	PRINT SET12CPI
13CPI	.BAT	PRINT SET13CPI
16CPI	.BAT	PRINT SET16CPI
6LPI	.BAT	PRINT SET6LPI
8LPI	.BAT	PRINT SET8LPI
RESETPRT.BAT		PRINT SETRESET
FORMFEED.BAT		PRINT SETFMFD/F

You can probably guess that these individual files contain the escape sequences that are necessary to set the printer according to the file descriptor name. And you would be right except that they contain some additional information also. First, let me explain that my batch file processing caused an extra line feed to be sent to the printer, and in the case of the form feed, I could not get rid of it no matter what I tried. The solution here was to simply print a null file with a form feed switch tagged on to the PRINT command. Thus the reason for the "/F" shown above. Let me show you the other files here:

File Name	Contents (1)
SET10CPI	↑ [[1w ↑ [M
SET12CPI	↑ [[2w ↑ [M
SET13CPI	↑ [[3w ↑ [M
SET16CPI	↑ [[4w ↑ [M
SET6LPI	↑ [[1x ↑ [M
SET8LPI	↑ [[2x ↑ [M
SETRESET	↑ [c ↑ [M
SETFMFD	(2)

(1) The symbol ↑ [designates an ESC character.
(2) This is an empty file.

The Escape-M sequence does a reverse index to compensate for the line feed introduced by the batch processing. This leaves the print head on the same line that it started on.

Now, when I want to set the printer to 16.5 characters per inch, I simply type 16CPI at the Z-DOS prompt followed by a RETURN. If I need a form feed, all it takes is to type FORMFEED at the prompt followed by a RETURN. By typing RESET followed by a RETURN, the printer gets reset to its power up condition.

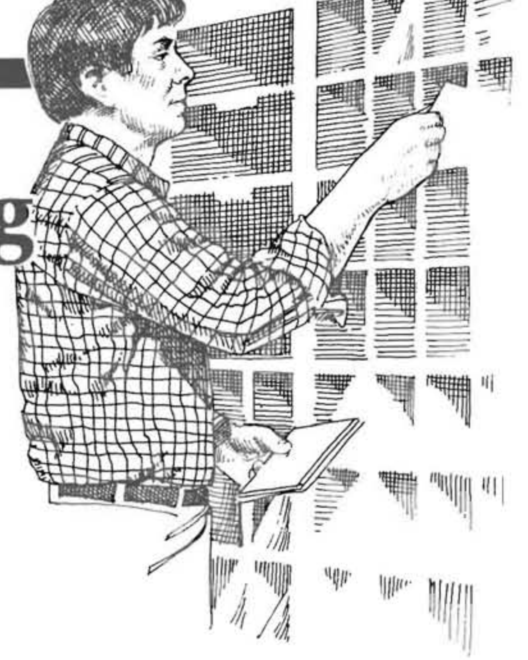
The above control codes are for the H/Z-25 printer. However the same scenario is applicable to other printers to a lesser or greater extent. Look in your printer manual for the sequences necessary to give you similar controls.

Keep in mind that batch processing can be useful in small applications as well as the big overnight jobs. There are probably many more similar problems that can be resolved with this approach.

About the Author:

*Charlie **Layman** started working with microcomputers in 1976 and has been actively engaged in the enjoyment of them ever since. He is responsible for writing the popular UD.DVD device driver for Heath/Zenith computers. Charlie is a department manager for GTE Sylvania in Needham, MA. and has a BS degree in Electrical Engineering from Northeastern University.*

Files and File Handling



David E. Warnick
RD #2 Box 2484
Spring Grove, PA 17362

Of all the things a computer can do, the handling of large amounts of data, organizing it, sorting it, and providing the information you want when you want it, is perhaps the most useful and the most powerful application you can use.

Random files. Sequential files. Records. They're all scary to the uninitiated, but with very little time and not too much effort, they become quite clear. And when they do, they become very valuable tools, tools you won't want to do without. So, don't put it off any longer. Learn about data storage and manipulation now. This is the first in a series of articles designed to make files and file handling as easy as booting up your computer. So come along for the ride. You'll be glad you did.

Just a word about the information you'll find in the series. All HUGgies are most welcome to make use of it for their personal applications, and the knowledge gained is yours for any purpose you desire. It is, however, to be considered proprietary and my property. I write the software for Applied Computing, a commercial venture, and am contemplating writing a book, but I would be very selfish and quite remiss if I failed to share these articles with the organization I turn to when I want to learn something new. 'Nuff said. Lets go on with the first article of our series.

File handling, what is it and why do it? What earthly purpose could it serve? Is it worth the time and the effort to learn how it's done? Each of these questions can have more than one answer. In this and future articles, I'll try to give you the information necessary to answer them for your particular set of circumstances. Those with firm answers will be laid out in detail for you.

To answer the last question first, yes, file handling is definitely worth the effort involved. As you learn file handling, you'll discover more and more uses for it. You'll also find that it's not at all difficult. There seems to be a lot of mystery and claims of magic with ominous titles like "SEQUENTIAL" and "RANDOM ACCESS". For now, believe me when I tell you it's quite easy. We'll take things a step at a time and, hopefully, give thorough explanations of what's going on and why we do what we're doing.

If we're going to handle files, we have to know what files are. Every separate program you store on your disk or tape is a file. It has a file name and you can copy it or make changes to it. So, in a sense, you have already been handling files. The article you're reading right now is stored on my disk with the name NUMBER8.TXT. But files go beyond this. There's much more you can do with them. Let's just say that a file is a collection of information. Information you may want to read, print, rearrange, or modify. The classic example of a file is the name and address list. But a file could just as well be the closing prices of a stock which an investor wishes to watch. For the homeowner, it could be a list of house payments with dates, check numbers, amounts, etc., all ready to be retrieved and processed or whatever you want. The ham radio operator could keep track of all the stations he's talked to and sort them by call, country, or any way he likes. Small businessmen can keep track of their inventory.

So you can see that uses for files are endless. I'm sure you can envision many applications you'd like to try for yourself. By the time this series is finished, it'll be a piece of cake and you'll be making some very practical use of your computer.

The items I pointed out above all contained similar pieces of information, repeated over and over within a file. The stock market quotes, for example, would record a date and a price. There may even be an opening, closing, high and low price with the date. The same information would appear every time we make an entry into our file and would be included when we retrieve data from our file. We should have a name for this item, so we'll call it RECORD. We now have two definitions.

FILE. A file is information which can be stored, retrieved, changed, sorted, and processed. It consists of several related records arranged systematically.

RECORD. A record is the smallest complete collection of data to be entered into or retrieved from a file.

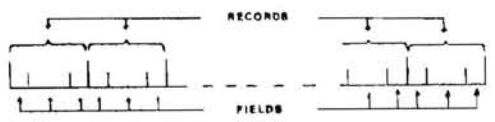
Notice I said that a record is the "smallest complete collection". In other words, a record is a collection of data. Each item of that data would be meaningless by itself, but when grouped together it gives very meaningful information. What do we call these pieces of data which make up a record? FIELDS. Each record within a file must be arranged the same way as every other record. In the example of the stock prices above, if the date is the first piece of data or "field" in one record, it must be the first field in every record within that file. If the opening price is the second field in one record, it must be the second field in every record within that file. In a separate file the records could be arranged differently, but, within a given file, the records must all be arranged the same way. Thus, if we wanted to print a chart of all the closing prices for our imaginary stock, all we'd have to tell the computer is:

1) Read each record in the file.
2) Print the first field within each record (date).
3) Print the nth field within each record (closing price).

Now we have a third definition.

FIELD. A field is a sub-division of a record. It is usually the lowest distinct order of data within the record.

Believe it or not, once you understand the concept of fields, records, and a file, you've made it well on your way to file handling. The rest of the way you'll just be applying these concepts to practical uses. Figure 1 shows pictorially how these elements interrelate to form a file.



Structure of a Data File

Figure 1

Ok, so you understand that you can pick items like date, check number, amount, interest, and principal and call each of them a field within a record. Then every time you make a house or car payment, you can type this information and save it in a file. But how will the computer know where anything is on the disk? How will it know where one record ends and the next record begins, or even more amazing, where the fields within a record begin or end?

There are two ways to know these things and each is quite easy. In the example above, we could say that each of the fields is 10 characters long, thus making each record 50 characters long. If the information to be placed in any of the fields requires less than 10 characters, just fill the rest with spaces. Then to print out the dates, amount paid and principal, just instruct the computer to read 50 characters and print characters 1 to 10, 21 to 30, and 41 to 50. Just repeat this process of reading 50 characters at a time until all data is exhausted. Not only would we know where every field is within a record, but also where every record is within a file. We could print every 5th record by telling the computer to skip 200 characters, read 50 and print, skip another 200, read 50 and print, etc. This would allow us to get into and out of the file and select records at random, just because we know where every one must be. Did you catch that word random? You've just been through a RANDOM FILE. The secret to finding anything in this type of file system is that everything has an assigned space and must be there.

But there's a price we must pay for this simplicity. While none of the information in the fields of our last example is likely to be 10 characters long, we allowed that much space. Each field could have been assigned a different length within a record, but all records must be the same. If we assign 7 spaces to the first field of one record, it must be 7 spaces in every record. Thus we waste space where there's no information. In large files with many records, this can become a considerable amount of waste and be quite objectionable. What can we do to conserve this space? If we just write one item after another, there's no way to tell where one ends and the next begins.

Obviously I wouldn't have babbled on through the last paragraph if there wasn't a way to do what we want. The easiest way to tell one field from another is to pick a character that won't appear in any field and insert it between each field of the record. An asterisk (*) is one good choice. We could write our record like this:

Date*Check Number*Amount*Interest*Principal

Now we can instruct the computer to read from the file, checking each character until it comes to an * and put the information into the first field of our record, then read to the next * and call that information the second field, and so on. So how can we tell when the whole record is finished? It should be obvious by now that we'll use another character, say the split vertical line (|). Actually, we could have used a double asterisk (**) as well. When the | is read, we stop because the record is complete. We'll call the special symbols we assign to indicate the limits of records within a file or the limits of fields within a record - DELIMITERS. Thus we've added one more word to our file- handling vocabulary.

By using delimiters to show where one piece of data ends and the next piece begins, we can pack items against each other, in sequence, without wasting any space in memory or on our storage media, be it tape or disk. Again, note that all data items are in sequence. This is the way a "SEQUENTIAL FILE" is arranged. It puts the most data in the least space. The price we pay for this memory savings is the inability to go directly to just any record. Because not all records are the same size, we have no idea what memory address is assigned to the record we want. To find it, we must start at the beginning of the file and read each record till we get to the right one.

Thus far we have discussed and you should know the meaning of the following six terms.

FILE
RECORD
FIELD
DELIMITER
RANDOM FILE
SEQUENTIAL FILE

The terms random and sequential file will gain more meaning as we go along, but you've been introduced to them and have some background to build on. Each type of file is handled differently by MBASIC and we must learn a different set of programming instructions for each. By taking them one at a time and drawing parallels between the two, we should progress quite easily and get the most out of this subject.

Most authors present sequential files first, a throwback to the days when tape storage was all we had and things could only be done that way, and then just present random files as an afterthought. I feel that random files are easier to work with and understand so I will present them first. Both types will be covered thoroughly before this series is over as each has specific uses, advantages, and disadvantages. Before we get into actual file operations, let's look at these file types and some of the characteristics of each.

RANDOM FILES

Advantages

1) Very fast access of any record.
2) Records can be added more easily.
3) File may be opened for both read and write operations.
4) Permits use of key files (to be discussed later).

Disadvantages

1) Takes more storage space on disk or tape or in RAM.
2) Must be sorted to allow random access.

SEQUENTIAL FILES

Advantages

1) Takes least space on disk or tape or in RAM.
2) Processes fastest for list print-out type jobs.

Disadvantages

1) Requires addition and removal of delimiters.
2) File may be opened for Read or Write—not both.
3) Modification of file requires use of a temporary file.

That should be enough to digest this month. Next month we'll look at the MBASIC operations which are used with random files. For those of you who wish to look ahead, they will be:

OPEN
FIELD
LSET
RSET
PUT
GET
MKI$
CVI

See you next month.

ARE YOU MOVING?

Don't leave your REMark behind.

Send your change of address in now to:

Heath Users' Group
Hilltop Road
St. Joseph, MI 49085

NEW *FOR YOUR Z-100*

ZBERT is a video arcade game that will challenge the entire family.
Package includes "ARTIST", a graphics editor.
Requires ZBASIC & full Video Ram.
ONLY $29.95

DUALPORT will intelligently parallel your Z-100* & Z-29* or Z-19* Terminal either direct or through a modem. Requires: ZDOS
ONLY $39.95

* Z-100, Z-29, Z-19 ARE ZENITH Trademarks

Available at Heathkit Electronic Centers or when ordering direct add $2 S/H. Kansas orders Add 4%. VISA & MASTERCARD ACCEPTED.

SUNFLOWER SOFTWARE, INC. (913) 631-1333
13915 Midland Drive • Shawnee, KS 66216

Controlled Data Recording Systems Inc.

ANNOUNCING THE FDC-H8

DOUBLE DENSITY 8" AND 5.25" CONTROLLER FOR THE H8 COMPUTER

Has all of the capabilities of our popular FDC-880H controller, with the added features of;

- Direct memory access (DMA) data transfer.
- Hard sectored controller (H17) incorporated on the board.
- Runs with the standard 8080 CPU card and with Z80 CPU upgrades.
- Accesses both hard sectored disk formats and soft sectored disk formats through the same drives attached to the FDC-H8 without hardware additions. **Price $495.00**

NEW PRODUCTS FOR THE FDC-880H

DM-1 DUAL BOARD MODIFICATION KIT **$29.95**
Allows for both the FDC-880H and the H88-4 controller cards to interface with the same 5.25" drives. Drives will run as both hard sectored format and soft sectored format depending upon the logical drive letter.

CDR BIOS by Livingston Logic Labs **$60.00**
Enhanced version of Heath/Zenith CP/M 2.203 BIOS with ZCPR. Supports all Heath/Zenith disk formats through the FDC-880H and the H17 controllers.

CDR DVD by Livingston Logic Labs. **$40.00**
HDOS driver for running double density HDOS through the FDC-880H

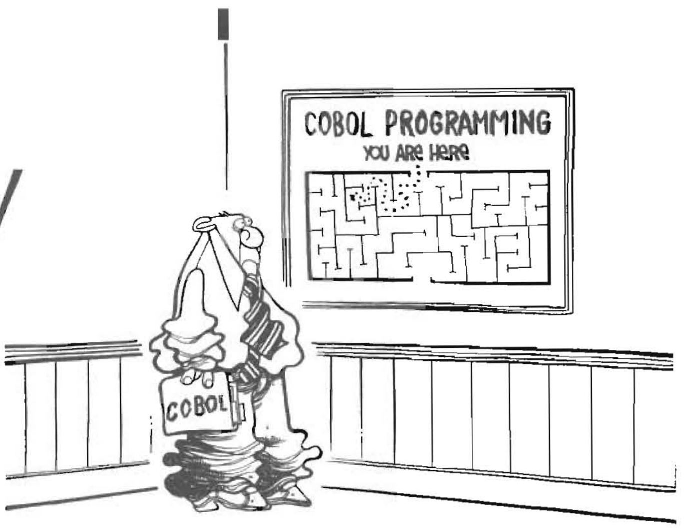
Shugart Slimline 5.25" 40 track double sided drives **$275.00**

Shugart Slimline 8" double sided drives **$525.00**

Contact: **C.D.R. Systems Inc.**
7210 Clairemont Mesa Blvd., San Diego CA 92111
Telephone: (619) 560-1272

5-20 day delivery-pay by check, C.O.D., Visa, or M/C

COBOL Corner IV



H. W. Bauman
493 Calle Amigo
San Clemente, CA 92672

Introduction

Welcome back to "COBOL Corner". I hope that you have obtained your HUG COBOL Corner Disk-I. If you have not, you can still work along with this article, but will need it by the next "COBOL Corner". We will not dwell on the previous articles. I will assume that everyone is up and running with their COBOL systems and knows how to FORMAT a COBOL Program Structure, as well as how to compile and run a COBOL program. If not, review past articles NOW!

We will start our COBOL programming by learning how to develop the program Phase by Phase!

Four (4) Phases To Develop A COBOL Program

Phase I — Specification Phase including illustrative layouts of the Input and Output Records.

1 — Print Chart. Diagram the output format.
a) Grid-like form with 132 columns used to define an OUTPUT Record.
b) Serves as a preview of how the report will look.

2 — Record Chart. Diagram the input format.
a) Modeled after an 80 column "punch-card".
b) Programmer describes the position and size of each field.

3 — General Specification of the Program. English Narrative.

4 — System Flowchart. Graphic description of INPUT, PROCESSING and OUTPUT flow.

PHASE II — Design Phase (Program Design Tools)

1 — Structure Chart. Graphic Hierarchy of tasks to be performed.

2 — Pseudocode. English-like documentation of Program.

3 — Program Flowchart. Graphic showing of Program logic flow.

4 — Structure Walkthrough. Review of design by a colleague.

***(Note:** Most Programmers do either step 2 or 3, not both.)*

Phase III — Coding Phase (Do not confuse this to mean just Keying!)

1 — Write Program Code on COBOL coding forms.

2 — Key the Source Code with your Editor from the coding forms. This is NEVER done until step 1 has been checked with Phase I and II program tools! NEVER key a program until you have a "Good" design that will compile and execute without errors, produce the correct output format, and be capable of UPGRADING and be MAINTAINABLE.

3 — Compile your Source Code. Remember the compiler will only catch syntactical errors (use of COBOL Language) and typos. Keep correcting the ERRORS until you obtain a "Clean Run" ("NO ERRORS OR WARNINGS").

Phase IV — Testing Phase

1 — Link and Execute your compiled program with a set of test data, called a Transaction File.

2 — Remember, just because the compiler shows no errors that does not mean your program will RUN or MEET Phase I Format and Specifications.
a) Your design may have LOGIC ERRORS—Used wrong COBOL Verbs to solve the program.
b) Your design may not produce the correct OUTPUT FORMAT.

3 — Go back to PHASE II, after reviewing PHASE I, and redesign, recode, and key-in the corrections.

User-Define Names (Created by the programmer for use in the program.)

1 — Data Names. Group of contiguous characters, each data-item must be assigned a unique data-name.

2 — Procedure Names. Paragraph or section names.

3 — Condition Names. Assigned to an item that may have various values or set of values or range of values, used in the PROCEDURE DIVISION to specify certain conditions for branching.

Rules For Assignment of Names

1 — Must be composed of only digits, alphabetic characters, and hyphens.
2 — Must contain at least one (1) letter.
3 — Cannot exceed 30 characters.
4 — Cannot begin or end with a hyphen.
5 — Cannot have imbedded blanks or periods.
6 — Cannot be a COBOL reserved word.
7 — Should be descriptive, meaningful, and readable (self- documenting)!

Spacing and Punctuation of COBOL Words

1 — There must be one (1) or more spaces between words.

2 — Periods ARE important in COBOL. REQUIRED in many places. You MUST remember these places.

3 — Period, comma, semicolon MUST be followed by one (1) or more spaces.

Developing Sample Program #1

*(**Note:** We do not have sufficient space on REMark's pages to show complete Print Charts, Record Charts, and Coding Forms; so, I suggest that you obtain a pad of each from your Computer Supply Store. The Print Chart has 132 columns and the Record Chart has 80 columns (a computer Punch Card can be used). The COBOL Coding Form is also a standard form at your supply store. We will show only a portion of the form below.)*

Phase I — Step 1 Print Chart for "PRGM01"

```
1234567890123456789012345678901234567890123456789012345678901234567890
  :                    : :                     : :            :: :
  :    CUSTOMER        : :       CUSTOMER      : :  CUSTOMER  ::ST:ZIP
  :      NAME          : :        ADDRESS      : :    CITY    :: :
  :                    : :                     : :            :: :
  :XXXXXXXXXXXXXXXXXXXXX: :XXXXXXXXXXXXXXXXXXXXXX: :XXXXXXXXXXXXX::XX:XXX
  :                    : :                     : :            :: :
  :XXXXXXXXXXXXXXXXXXXXX: :XXXXXXXXXXXXXXXXXXXXXX: :XXXXXXXXXXXXX::XX:XXX
  :                    : :                     : :            :: :
```

*(**Note:** We show where the print-out will occur with an "X" and blank spaces between. We also show three (3) lines of print-out to show that we want double spaced output. Because we lacked line space we did not show the complete ZIP CODE field of five (5) "X"'s on right side of the chart.)*

Phase I — Step 2 Record Chart for "FILEL1.DAT"

```
1234567890123456789012345678901234567890123456789012345678901234567890
  CUST.     CUSTOMER            CUSTOMER           CUSTOMER   ST ZIP
  ACCT.        NAME             ADDRESS              CITY
  99999XXXXXXXXXXXXXXXXXXXXxxxxxxxxxxxxxxxxxxxxxxXXXXXXXXXXXXXxx99999
```

*(**Note:** This Transaction File has some other data that we will use later, but this covers the Fields we will use with this program.)*

Phase I — Step 3 Programming Specifications

```
--------------------------------------------------------------------
Program Name: Customer List              Program ID:  PRGM01.DOC
--------------------------------------------------------------------
```

Program Description

This program reads a customer account, name, and address data file (Transaction File—more about this in later articles) and prints a customer name/address list.

Input File
Customer account number-name-address disk file.

Output File
Customer name and address list.

List of Program Operations

1 — Read each customer account number-name-address record from disk—FILEL1.DAT.
2 — For each record, print the following fields on the customer list in accordance with the Print Chart:
    Customer Name
    Customer Address   Customer City-State-Zip
3 — Double space each printed line.
4 — COBOL will be the programming language.

Phase I — Step 4 System Flowchart

```
--------------------------------------------------------------------
                 SYSTEM CHART--CUSTOMER LIST PROGRAM
--------------------------------------------------------------------


                          -----------
                          : ACCOUNT :
                          :  NAME   :
                          : ADDRESS :
                          :  FILE   :
                          -----------
                               :
                               :
                          -----------
                          :CUSTOMER :
                          :  LIST   :
                          : OUTPUT  :
                          -----------
                               :
                               :
                          -----------
                          :CUSTOMER :
                          :  LIST   :
                          :PRINT :OUT
                          -----------
--------------------------------------------------------------------
```

Phase II — Step 1

*(**Note:** We are going to use MODULES (like sub-routines in BASIC) with our Structured COBOL Programming.)*

```
--------------------------------------------------------------------
Hierarchy Chart^—^Customer List Program
--------------------------------------------------------------------

                        ------------
                        :  MAIN-   :
                        : CUSTOMER :
                        :  -LIST   :
                        ------------
                             :
                             :
         ------------------------------------------
         :                                        :
         :                                        :
   -------------                            -------------
   : INITIALIZE:                            :  PROCESS- :
   : -VARIABLE :                            : CUSTOMER  :
   :  -FIELDS  :                            :   -LIST   :
   -------------                            -------------
--------------------------------------------------------------------
```

Phase II — Step 3 Program Flowchart

*(**Note:** We are going to choose Phase II—Step 3 Program Flowcharts over the Phase II—Step 2 Pseudocode in our early programs. As the programs get more complex we will use Pseudocode. With complex programs, the flowcharts get too elaborate and are hard to follow. Remember it is the programmer's choice.)*

The FLOWCHARTS are shown separately in this article.

Phase II — Step 4 Structure Walkthrough

At this point, if possible, you should have your program design reviewed by a colleague or friend to check your Logic and Structure with the above programming tools. This is where you want to find your errors, on paper, not with the computer!

Data File (Transaction File)

If you have your HUG COBOL Corner Disk-I you can copy the file, FILEL1.DAT, from the HUG disk to your Disk A for this Sample Program #1 using the same name. Now, print out FILEL1.DAT with your printer. Can you find the fields on it that we will use with this program? It has some fields that we do not use this time but will use with a later program. It should MATCH the PHASE II— STEP 2 RECORD CHART.

If you do not have the HUG COBOL Corner Disk-I, the FILEL1.DAT is shown below:

*(**Note:** This line of numbers is just to show the field columns! Do not include them in your work!)*

```
1234567890123456789012345678901234567890123456789012345678901234567890123
L100002ABBOTT, ANTHONY     14255 CAVENDISH PLACE SAN FRANCISCOCA94122  00014489
L100005SIMMONS, RONALD     12 WALNUT AVENUE      ATHERTON     CA94025  00033250
L100010WHEELER, EDMOND     1240 ALDERBROOK LANE  SAN JOSE     CA95101  00120056
L100011PENITENCIA, JOSE    101 GISH ROAD         SAN JOSE     CA95129  0000110N
L100012STEWART, JANET      1002 MARSHALL LANE    LOS GATOS    CA95030  00030075
L100015JOHANSON, PAUL      14901 SOBEY ROAD      LOS GATOS    CA94402  00004421
L100019JAMES, JACQUELINE   42 WARREN AVENUE      SAN MATEO    CA94402  00050356
L100022COLLINS, BRUCE      1200 RACE STREET      SAN JOSE     CA95114  00033994
L100026REILLY, JACK        14101 JUNIPER LANE    SARATOGA     CA95070  00004498
L100031COOPER, ROXANNE     10305 LOVELAND COURT  SARATOGA     CA95070  00022165
L100035CASHMAN, BARBARA    11 PENNSYLVANIA AVENUELOS GATOS    CA95030  0000458R
L100040WILMER, PATRICE     6612 LAKEWOOD TERRACE SANTA CLARA  CA95050  00002300
L100042YANG, CYNTHIA       1282 BENTON AVENUE    SANTA CLARA  CA95051  00010203
L100047MACINTYRE, JOHN     10200 TERESITA BLVD   CUPERTINO    CA95014  00090054
L100048BROWN, BRUCE        1104 EL CAMINO REAL   LOS ALTOS    CA94022  00003200
L100050HUMPHREY, STAN      133 HILLVIEW ROAD     LOS ALTOS    CA94022  00033221
L100053LOGAN, MATHILDA     40221 ROSS ROAD       PALO ALTO    CA94043  00009943
L100054ROGERS, WALTER      1600 ORCHARD AVENUE   MENLO PARK   CA94025  00023559
L100060SCHULTZ, CHARLES    998 TASMAN DRIVE      SUNNYVALE    CA94086  00004479
L100062CARVER, GEORGE      6621 WILLOW ROAD      MENLO PARK   CA94025  00113680
L100063CARTOZIAN, ARAM     1550 BIRCHTREE LANE   REDWOOD CITY CA94084  00016260
L100066FORSYTHE, DOUGLAS   220 MAGNOLIA DRIVE    HILLSBOROUGH CA94010  00018320
L100081HARRIS, CLIFFORD    50102 CYPRESS STREET  SAN JOSE     CA95130  00011990
L100083ALVAREZ, ELENA      1060 PALOS VERDES ROADLOS ALTOS    CA94022  00212100
L100088KNOLL, DONALD       330 ANDERSON LANE     MORGAN HILL  CA95037  00013210
L100100XAVIER, FRANCIS     100 CORPUS CHRISTI WAYMOUNTAIN VIEWCA94942  00314340
L100103LANDUSKY, FRED      8823 CRESTVIEW DRIVE  CUPERTINO    CA95014  00015620
L100111QUINTANA, VINCENT   402 CANDY LANE        BURLINGAME   CA94010  00015670
L100113HELM, SIGRID        11005 VIA GRANDE DRIVESARATOGA     CA95070  06012750
L100116THOMPSON, CHARLES   62 GLEN BRAE DRIVE    SARATOGA     CA95070  00017290
L100133PICCATA, BEVERLY    222 MAUDE AVENUE      SAN CARLOS   CA94070  00012350
L100149WESTOVER, LEE       1077 NOTRE DAME ROAD  BELMONT      CA94002  00015964
L100188PAPPAS, IRENE       4420 WHIPPLE ROAD     REDWOOD CITY CA94084  00019130
L100200WHELAN, ROD         1100 WINCHESTER ROAD  CAMPBELL     CA95008  00011970
L100201CORVINO, GERALD     5504 SAN TOMAS ROAD   CAMPBELL     CA95008  00010420
L100205WAGNER, GUY         11 WASHINGTON SQUARE  SAN BRUNO    CA94086  00012310
L100210HAMILTON, BRUCE     881 PEMBERTON PLACE   BURLINGAME   CA94010  00013510
L100213WHITNEY, DAVID      10002 COLUMBINE AVENUEREDWOOD CITY CA94084  02014630
L100214PARSONS, LAURETTE   335 STONYDALE DRIVE   BURLINGAME   CA94010  00002489
L100222SNOW, MEREDITH      411 OAK RIDGE DRIVE   MILLBRAE     CA94030  00006735
L100227CARSON, JAMES       992 LOMITA AVENUE     SAN JOSE     CA95128  00004252
L100244WHITE, BERNARD      2200 THE ALAMEDA      SAN JOSE     CA95131  00008908
L100271JOHNSON, SARAH      2009 PARK PLACE       MILLBREA     CA94030  00002111
L101002PAULSEN, PATRICK    122 NIGHTINGALE LANE  SAN CARLOS   CA94070  00008114
L101004JACKSON, ANDREW     2133 TILLMAN PLACE    SAN MATEO    CA94402  00011970
L101101STERLING, RONALD    42 WEST 27TH STREET   REDWOOD CITY CA94084  00004415
L101108GUSTAFFSON, HARRY   665 MCALLISTER STREET BELMONT      CA94002  00023312
L102162CALLAHAN, KATHEEN   286 STATE STREET      LOS ALTOS    CA94022  00082213
L102225GOLDMAN, RACHEL     133 MASON STREET      SUNNYVALE    CA94087  00056719
L102294SANDERS, GARY       10204 37TH STREET     SUNNYVALE    CA94086  00062610
L102407ENDICOTT, PAMELA    885 HILLVIEW TERRACE  MOUNTAIN VIEWCA94043  00069512
L105252WOO, CELIA          754 GLEN BRAE DRIVE   SARATOGA     CA95070  00001411
L106280CHAVEZ, RAFAEL      1220 KINGMAN ROAD     LOS GATOS    CA95030  00080913
L108209STEWART, JANET      605 BUBB ROAD         CUPERTINO    CA95014  00003016
L108231YARBOROUGH, JEFFREY 12076 CAROLINA STREET MOUNTAIN VIEWCA94042  00012185
L108722ABINGTON, CARL      886 PICKWICK COURT    SANTA CLARA  CA95050  00005174
L110215ADOLPH, CHARLOTTE   1402 DISNEY LANE      SAN JOSE     CA95130  00028513
L110222ASHTON, WILLIAM     99722 LAKEVIEW WAY    SANTA CLARA  CA95051  00002827
L110273YOUNG, RODGER       122 CONSTITUTION ROAD MILPIAS      CA95035  00053268
L110334FREDERICKS, PATRICIA3212 FOOTHILL BLVD    CUPERTINO    CA95014  00012350
```

Now, for practice in preparing a Data File, use Disk A, your editor, and key-in the above file putting only the fields required by our program as diagrammed on the RECORD CHART making sure that each field is in the specified columns! When completed, SAVE it on Disk A with the name FILE01.DAT. Prepare a print-out and compare your work with the file FILEL1.DAT.

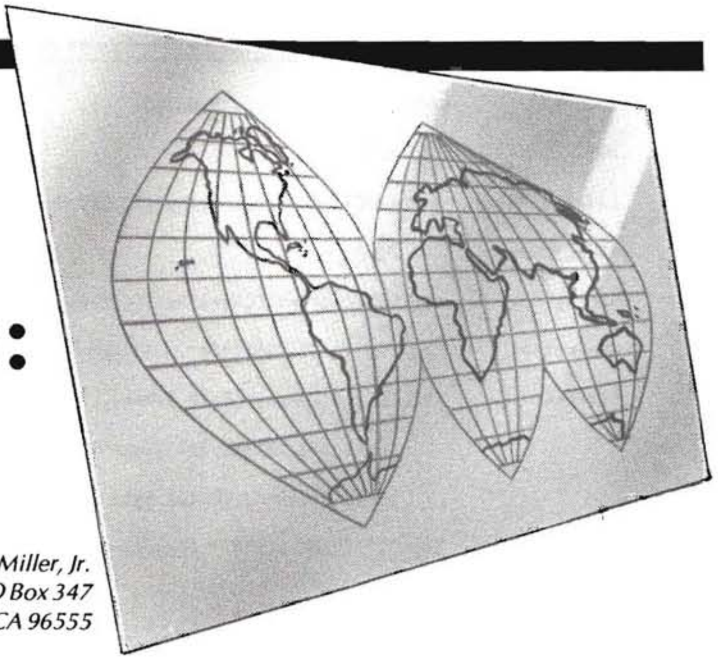
Closing

In the next "COBOL Corner" we will start with Phase III — Coding! We will write it out on coding forms and describe each COBOL Division. For your "Homework" please review the Select Entry, FD Entry and Working-Storage Section in your COBOL-80 Reference Manual.

In our future Sample Programs I will not do all the Phases for you. I will supply Phases to specify what the program will be required and leave the rest for you to work out.

This might be a good time to mention that we will be dealing with Line Sequential Files for this and many of the coming programs. Random Files are really advanced COBOL; thus, we will not work with these programs for many months. We have a lot of COBOL Language, Structured Programming, and "Good" Programming to work with first.

ZBASIC Mapping Program: BASMAPER



Ted W. Miller, Jr.
PO Box 347
APO San Francisco, CA 96555

The program BASMAPER generates a reference map of statement numbers, variable names, literal values, and constant values for a ZBASIC program. Output to a disk file, to the line printer or to the screen is optional, as is report selection. BASMAPER is written in ZBASIC, for ease of update or user modification. It is executed by first MERGEing and then RUN 65500 (defaults all reports to the printer). Limitations on the program being mapped: Length = 29899 characters, and maximum statement number = 65499. Soft limits: number of statements = 400, number of referenced elements = 1000, and number of references = 3000. (See the dimension statement at 65001 of the listing.)

After four years of programming in APPLESOFT on my APPLE II computer, I was delighted with ZBASIC on my new Z-100. The full screen editing, the many new functions, variable types, and graphics control gave programming a new dimension. However, the absence of programming support tools made the overall task of program development and checkout a chore. When in checkout, performing program documentation, and most especially when revising a program, it is mandatory for me to know what references what. So, after suffering a while trying to put together a data file management set of programs, I took out the time to write a reference mapper. In doing so, I learned some things about the internal storage of a ZBASIC program which I found useful. I hope you will too.

Figure 1 is the syntactic definition of the internal form of a ZBASIC program to the extent analyzed by BASMAPER. The symbol <=> is read "is defined as". The program is stored as a chain of tokenized strings which is common to all versions of BASIC. The link

Figure 1

```
                 ZBASIC Mapper: ZBASIC Definition - Internal form

ZBASIC Program   <=> [Basic Statement]|[Basic Statement],[ZBASIC Program]

Basic Statement  <=> [Numbered String]|[Numbered String],[Remark]

Numbered String  <=> [Link Address],[Stmt Number],[Token String]|

                     [Link Address],[Null]

Link Address     <=> [Unsigned Integer]

Unsigned Integer <=> "0"|"1"..."65535"

Stmt Number      <=> [Unsigned Integer]

Token String     <=> [String Variable]|[String Variable],[Token String]

String Variable  <=> [Map Element]|[Spaces]|[Other]

Map Element      <=> [Stmt Reference]|[Variable Name]|[Function Name]|

                     [Literal Value]|[Constant]

Stmt Reference   <=> [Statement Token],[Stmt Number]|[Link Token],[

                     [Link Address]

Statement Token  <=> "14"

Link Token       <=> "13"

Variable Name    <=> [Name]|[Name],"("

Name             <=> [Letter]|[Letter],[Alphanumeric]|[Name],[Type Char]

Letter           <=> "A"|"B"...|"Z"

Alphanumeric     <=> [Letter]|[Digit]|"."

Digit            <=> "0"|"1"...|"9"

Type Char        <=> "%"|"!"|"#"|"$"

Function Name    <=> [Function Token],[Spaces],[Name]

Function Token   <=> "216"
```

```
Spaces           <=> " "¦" ",[Spaces]¦[Null]

Null             <=> ""

Literal Value    <=> [ASCII Quote],[String],[ASCII Quote]

ASCII Quote      <=> "34"

String           <=> [Character]¦[Character],[String]

Character        <=> "32"¦"33"¦"35"¦"36"...¦"126"

Constant         <=> [Digit Token]¦[Byte Token],[Byte]¦[Integer Token],

                     [Unsigned Integer]¦[Single Token],[Four Byte Real]¦

                     [Double Token],[Eight Byte Real]

Digit Token      <=> "17"¦"18"...¦"27" (Represents "0-9")

Byte Token       <=> "15"

Integer Token    <=> [Octal Token]¦[Hex Token]¦[Decimal Token]

Octal Token      <=> "11"

Hex Token        <=> "12"

Decimal Token    <=> "28"

Single Token     <=> "29"

Four Byte Real   <=> "0"..."1.701412E+/-38"

Double Token     <=> "31"

Eight Byte Real  <=> "0"..."1.70141183460469 2D+/-38"

Other            <=> [Character]¦[Other Token]

Other Token      <=> "0"¦"1"...¦"10"¦"16"¦"30"¦"127"¦"128"...¦"255"

Remark           <=> [REM Token],[Other]

REM Token        <=> "143"

Note: "<=>" is "is defined as"; "¦" is a logical OR; "," is "followed by.
```

address at the start of each string points to the beginning of the next token string. The statement number of the string follows the link address. The remainder of the string is a codified representation of the statement entered. It consists of tokens representing reserved words, operators, punctuation, and possibly a REMark in addition to the map elements of interest.

I was impressed when I found that ZBASIC uses the token "13" to identify a direct reference to another statement, which eliminates the scan from the top of program on all transfers as performed in APPLESOFT.

The major difference is the use of tokenized constants by ZBASIC. In APPLESOFT, it is not good programming practice to use constants. They are stored as ASCII strings and converted at execution time (faster to use variables). In ZBASIC, the opposite is true. All constants and statement references are stored as tokens or in their internal form.

Finally, on the negative side, spaces are not tokenized. So, while the ZBASIC documentation espouses the structuring of your code, if you do, you are wasting memory. Maybe on a later revision.

Now to the program. You'll have to reference the listing (see Figure 2) during the following discussion. The statements 65500 through 65005 are for program initialization. The first of three phases of operation is displayed on screen, a dummy buffer is set up to hold temporary strings (this avoids the horror of garbage collection), arrays are dimensioned, functions are defined, and pointers are set to the user program. In statement 65501, you will find "I" initialized to the integer value at PEEK 343. This is the address used by ZBASIC to identify the start point of the user program. "K" is initialized as the address of the next numbered statement and "J" is the statement number of the first statement. Phase 1 is performed at 65522, after which all the statement numbers are found in the string array "X$", ready for display. Phase 2 is the workhorse performed by statements 65523 through 65526. During this phase, the statement number of the statement being parsed will be displayed. This is done so you'll know your machine really didn't go south, it's just busy. Each statement is parsed, bytewise, for map elements. Bytewise parsing is required because constants, in their internal form, may take the value of any ASCII character or token. While this task would be speeded up considerably if it were programmed in machine language, the trade off was ease of modification, operation, and understanding. Since you will probably not map your program more than two or three times, execution speed was a secondary consideration.

The statement being parsed is set up as the string "J$" in statement 65523, this allows access either by PEEKs or the ZBASIC string operators. Statement 65524 checks for map elements, using the subroutines at 65510 through 65521 to isolate them and "move" them to "I$". This "move" for variable names and literals is simply pointing the "I$" descriptor to the element within the program (saves string space). The subroutine at 65507 is then used to find the element, or where it should be, in "X$". This is an indirect lookup through a pointer list in "Y%". It is a very quick method of finding an element in a sorted array. Finally, if the element is not found, the subroutine at 65507 inserts it at its proper sort position and the number of the statement being parsed is added to the reference chain for that element in "X%".

Phase 3 is performed by statements 65526 through 65529, which by the way, is the highest statement number allowed by RENUM. Phase 3 is the report generation phase. "RPRT$" is used to select the reports to be produced, and "FLNM$" identifies the device and/or filename to use for output. The subroutine at 65008 is used for line printer initialization. It is written for an EPSON MX-80 printer, so you might have to change it to be compatible with your line printer. The spacing on the reports is set for a 132 character line length (or more properly, a line length divisible by 6 with no remainder). If you can't set your printer up for that format, you may want to take the time to change the spacing so that your reference statement numbers will not wrap around.

Installation: Simply key in the program, modifying it as you want, and then type

Figure 2

```
ZBASIC Mapping program: BASMAPER

65500 CLS: LOCATE 12,24: PRINT "WORKING PHASE 1";: OPEN "COM1:" AS #2 LEN=80:
      FIELD #2,6 AS I$: DEF FN XX(I)=256*PEEK(I+1)+PEEK(I):
      IF RPRT$="" THEN RPRT$="1234"
65501 DIM X$(1000),Y%(1,400,3),X%(1,3000),IX(3): I=FNXX(343): J=FNXX(I+2):
      K=FNXX(I): J$=" ": IF FLNM$="" THEN FLNM$="LPT1:"
65502 DIM L$(3): L$(0)="Statement Number Reference Map":
      L$(1)=" Variable Names Reference Map ":
      L$(2)=" Literal Values Reference Map ":
      L$(3)="Constant Values Reference Map "
65503 DEF FN FDC(I,J)=INSTR(I,J$,CHR$(J)): DEF FN PINT=J+65536!*(J>32767):
      DEF FN CLTH=ABS((X>14 AND X<28)+(X=15)+3*(X>10 AND X<13 OR X=28)+5*(X=29)
      9*(X=31))
65504 DEF FN STMT=ABS(X=14)*FNXX(I+N+1)+ABS(X=13)*FNXX(FNXX(I+N+1)+3):
      DEF FN NAM=X>64 AND X<91: DEF FN DEC=X%(1,NA+I)-65536!*(X%(1,NA+I)<0):
      DEF FN CHAR=X>48 AND X<58 OR FNNAM OR X=33 OR X>34 AND X<38 OR X=46
65505 DEF FN NTE=I$<>X$(Y%(0,JX,MX)): DEF FN LESS=I$<X$(Y%(0,JX,MX)):
      DEF FN MORE=I$>X$(Y%(0,JX,MX)): GOTO 65522
65506 LX=0: JX=IX(MX): WHILE FNNTE AND JX>LX:
            KX=JX: WHILE FNLESS AND JX>LX: KX=JX: JX=JX-INT((KX-LX)/2+.5): WEND
                   WHILE FNMORE AND JX>LX: LX=JX: JX=JX+INT((KX-LX)/2): WEND:
      WEND: RETURN
65507 IF FNNTE THEN JX=JX+1: FOR LX=IX(MX)-(IX(MX)=0) TO JX STEP-1:
      SWAP Y%(0,LX,MX),Y%(0,LX+1,MX): SWAP Y%(1,LX,MX),Y%(1,LX+1,MX): NEXT:
      IX(MX)=IX(MX)+1: NL=NL+1: Y%(0,JX,MX)=NL: X$(NL)=I$: RETURN
      ELSE RETURN
65508 PRINT#1,CHR$(27);"N";CHR$(6);CHR$(12);CHR$(15);CHR$(14);TAB(10);: RETURN
65509 J=Y%(1,JX,MX):I=0: WHILE J>0: I=I+1:X%(1,NA+I)=X%(1,J): J=X%(0,J): WEND:
        WHILE I>0: PRINT#1,USING"######";FNDEC;: I=I-1: WEND: RETURN
65510 MX=2: Z=FNFDC(N+2,34): IF Z<N THEN Z=FNFDC(N+2,58): IF Z<N THEN Z=L
65511 Y=Z-N: POKE P,Y: POKE P+1,(I+N) MOD 256: POKE P+2,INT((I+N)/256): RETURN
65512 WHILE NOT FNNAM: N=N+1: X=PEEK(I+N): WEND: Z=N: MX=1: WHILE FNCHAR:
      Z=Z+1: X=PEEK(I+Z): WEND: Y=Z-N: FIELD #2,Y+3 AS I$:
      LSET I$="FN "+MID$(J$,N+1,Y): RETURN
65513 Z=N: MX=1: WHILE FNCHAR: Z=Z+1: X=PEEK(I+Z): WEND:
      Z=Z-(X=40): GOTO 65511
65514 MX=3: Y=Z: FIELD #2, 13 AS I$: ON Z GOTO 65517,65518,65519
65515 IF Z=5 THEN X=VARPTR(V): FOR Z=0 TO 3: POKE X+Z,PEEK(I+N+Z+1): NEXT:
      RSET I$=STR$(V): RETURN
65516 X=VARPTR(V#): FOR Z=0 TO 7: POKE X+Z,PEEK(I+N+Z+1): NEXT:
      V=V#: RSET I$=STR$(V): RETURN
65517 V=X-17: RSET I$=STR$(V): RETURN
65518 V=PEEK(I+N+1): RSET I$=STR$(V): RETURN
65519 V=FNXX(I+N+1): IF X=11 THEN LSET I$="&O"+OCT$(V)
                ELSE IF X=12 THEN LSET I$="&H"+HEX$(V)
                ELSE RSET I$=STR$(V)
65520 RETURN
65521 FIELD #2, 6 AS I$: RSET I$=STR$(Z): MX=0: Y=4: RETURN
65522 WHILE K>0 AND I<32767 AND J<65500!: NS=NS+1: Y%(0,NS,0)=NS:
      RSET I$=STR$(J): X$(NS)=I$: I=K: K=FNXX(I): J=FNXX(I+2): WEND:
      IX(0)=NS: NL=NS: LOCATE ,38: PRINT "2": LOCATE ,24: PRINT "PARSING ";
65523 I=FNXX(343): WHILE SN<NS: SN=SN+1: P=VARPTR(J$): K=FNXX(I): J=FNXX(I+2):
      N=4: L=K-I-1: POKE P,L: POKE P+1,I MOD 256: POKE P+2,INT(I/256):
      P=VARPTR(I$): X=PEEK(I+N): LOCATE ,32: PRINT J;
65524 WHILE N<L AND X<>143: MX=4: Y=1: IF X=34 THEN GOSUB 65510
      ELSE IF X=216 THEN GOSUB 65512 ELSE IF FNNAM THEN GOSUB 65513
      ELSE Z=FNCLTH: IF Z>0 THEN GOSUB 65514
      ELSE Z=FNSTMT: IF Z>0 THEN GOSUB 65521
65525 IF MX<4 THEN GOSUB 65506: JX=JX*ABS(NOT FNNTE OR MX>0):
      ON ABS(MX>0) GOSUB 65507: IF Y%(1,JX,MX)=0 OR X%(1,Y%(1,JX,MX))<>FNPINT
      THEN NA=NA+1: X%(1,NA)=FNPINT: X%(0,NA)=Y%(1,JX,MX): Y%(1,JX,MX)=NA
65526 N=N+Y: WHILE PEEK(I+N)=32: N=N+1: WEND: X=PEEK(I+N): WEND: I=K: WEND:
      OPEN FLNM$ FOR OUTPUT AS #1: LOCATE 12,38: PRINT "3":
      MX=0: WHILE MX<4: LOCATE ,20: PRINT L$(MX);
65527 IF VAL(MID$(RPRT$,MX+1,1))<>MX+1 THEN JX=IX(MX)
          ELSE GOSUB 65508: PRINT#1,L$(MX):PRINT#1," ": JX=0:
          IF Y%(1,0,MX)>0 THEN PRINT#1,"UNDEFINED <:";: GOSUB 65509: PRINT#1," "
65528 WHILE JX<IX(MX): JX=JX+1: KX=Y%(0,JX,MX): X=LEN(X$(KX)): IF Y%(1,JX,MX)>0
      THEN PRINT#1,X$(KX);TAB(16+6*INT((X-4)*ABS(X>15)/6));" <:";: GOSUB 65509:
      PRINT#1," "
65529 WEND: MX=MX+1: WEND: CLEAR: END
```

SAVE "BASMAPER",A to save it as a MERGEable file.

Operation: Load the program to be mapped then type **MERGE "BASMAPER"**.

If you want all reports listed on the line printer, type **RUN 65500**.

If you want all reports sent to a disk file, type **FLNM$="<filespec>": GOTO 65500.**

If you want specific reports, type **RPRT$="ABCD": GOTO 65500** where:

INSERT LISTING

and:

A is the Statement number reference report (1),

B is the Variable names reference listing (2),

C is the Literal values reference listing (3), and

D is the Constant values reference listing (4) and a zero aborts the report.

Example: To generate a variable names map in file VARNAMES on disk B, Type **RPRT$="0200": FLNM$="B:VARNAMES.LST": GOTO 65500.**

A sample of each of the reports follows the listing as Figures 3 through 6.

Problems: Write to T. W. Miller, Jr., PO Box 347, APO San Francisco, CA 96555

ZBASIC is a trademark of Zenith Data Systems, Chicago, Il. APPLE II and APPLESOFT are trademarks of Apple Computer Co., Cupertino, Ca.

Figure 3

```
   Statement Number
     Reference Map

     65506         <: 65525
     65507         <: 65525
     65508         <: 65527
     65509         <: 65527 65528
     65510         <: 65524
     65511         <: 65513
     65512         <: 65524
     65513         <: 65524
     65514         <: 65524
     65517         <: 65514
     65518         <: 65514
     65519         <: 65514
     65521         <: 65524
     65522         <: 65505
```

Figure 4

```
                 Variable Names Reference Map

AS             <: 65504 65512 65514 65521 65526
FLNM$          <: 65501 65526
FN CHAR        <: 65504 65512 65513
FN CLTH        <: 65503 65524
FN DEC         <: 65504 65509
FN FDC         <: 65503 65510
FN LESS        <: 65505 65506
FN MORE        <: 65505 65506
FN NAM         <: 65504 65512 65524
FN NTE         <: 65505 65506 65507 65525
FN PINT        <: 65503 65525
FN STMT        <: 65504 65524
FN XX          <: 65500 65501 65504 65519 65522 65523
I              <: 65500 65501 65503 65504 65509 65511 65512 65513 65515 65516 65518 65519 65522 65523 65526
I$             <: 65500 65505 65507 65512 65514 65515 65516 65517 65518 65519 65521 65522 65523
I%(            <: 65501 65506 65507 65522 65527 65528
J              <: 65501 65503 65509 65522 65523
J$             <: 65501 65503 65512 65523
J%             <: 65505 65506 65507 65509 65525 65527 65528
K              <: 65501 65522 65523 65526
K%             <: 65506 65528
L              <: 65510 65523 65524
L$(            <: 65502 65526 65527
L%             <: 65506 65507
M%             <: 65505 65506 65507 65509 65510 65512 65513 65514 65521 65524 65525 65526 65527 65528 65529
N              <: 65504 65510 65511 65512 65513 65515 65516 65518 65519 65523 65524 65526
NA             <: 65504 65509 65525
NL             <: 65507 65522
NS             <: 65522 65523
OUTPUT         <: 65526
P              <: 65511 65523
RPRT$          <: 65500 65527
SN             <: 65523
V              <: 65515 65516 65517 65518 65519
V#             <: 65516
X              <: 65503 65504 65512 65513 65515 65516 65517 65519 65523 65524 65526 65528
X$(            <: 65501 65505 65507 65522 65528
X%(            <: 65501 65504 65509 65525
Y              <: 65511 65512 65514 65521 65524 65526
Y%(            <: 65501 65505 65507 65509 65522 65525 65527 65528
Z              <: 65510 65511 65512 65513 65514 65515 65516 65521 65524
```

Figure 5

```
Literal Values Reference Map

 " "             <: 65501 65527 65528
 " <:"           <: 65528
 " Literal Values Reference Map "        <: 65502
 " Variable Names Reference Map "        <: 65502
 ""              <: 65500 65501
 "######"        <: 65509
 "&H"            <: 65519
 "&O"            <: 65519
 "1234"          <: 65500
 "2"             <: 65522
 "3"             <: 65526
 "COM1:"         <: 65500
 "Constant Values Reference Map "        <: 65502
 "FN "           <: 65512
 "LPT1:"         <: 65501
 "N"             <: 65508
 "PARSING "      <: 65522
 "Statement Number Reference Map"        <: 65502
 "UNDEFINED <:"  <: 65527
 "WORKING PHASE 1"         <: 65500
```

Figure 6

```
                    Constant  Values  Reference  Map

          0   <: 65502 65504 65505 65506 65507 65509 65515 65516 65521 65522 65524 65525 65526 65527 65528
          1   <: 65500 65501 65502 65504 65507 65508 65509 65511 65512 65513 65515 65516 65518 65519 65522 65523 65524 65525 65526
65527 65528 65529
          2   <: 65500 65501 65502 65506 65510 65511 65512 65514 65521 65522 65523
          3   <: 65501 65502 65503 65504 65512 65514 65515
          4   <: 65521 65523 65524 65525 65526 65528
          5   <: 65503 65515
          6   <: 65500 65508 65521 65528
          7   <: 65516
          9   <: 65503
         .5   <: 65506
         10   <: 65503 65508
         11   <: 65519
         12   <: 65500 65508 65519 65526
         13   <: 65503 65504 65514
         14   <: 65503 65504 65508
         15   <: 65503 65508 65528
         16   <: 65528
         17   <: 65517
         20   <: 65526
         24   <: 65500 65522
         27   <: 65508
         28   <: 65503
         29   <: 65503
         31   <: 65503
         32   <: 65523 65526
         33   <: 65504
         34   <: 65504 65510 65524
         38   <: 65504 65522 65526
         40   <: 65513
         46   <: 65504
         48   <: 65504
         58   <: 65504 65510
         64   <: 65504
         80   <: 65500
         91   <: 65504
        143   <: 65524
        216   <: 65524
        256   <: 65500 65511 65523
        343   <: 65501 65523
        400   <: 65501
       1000   <: 65501
       3000   <: 65501
      32767   <: 65503 65522
      65530   <: 65522
      65536   <: 65503 65504
```

Changing your address?

Let us know. We don't want you to miss a single issue of **REMark**, send your change of address to:

Heath Users' Group
Hilltop Road
St. Joseph, MI 49085

ARE YOU PREPARED FOR TAX TIME?

TAXTYME will assist you in preparing and calculating your 1983 Federal Income Tax forms. **TAXTYME** simplifies filling out the forms by leading you through a series of formatted questionnaires. **TAXTYME** includes formatted questionnaires for IRS Forms 1040, Schedules A, B, and C. Schedule C is processed separately so that several Schedule C business activities can be calculated and combined on Form 1040. Information from Schedules D, E, F, and the many numbered Forms are calculated separately and entered on the questionnaires. Forms G, W, and 2210 are calculated automatically when required. **TAXTYME** makes decisions on Income Averaging, Marital Deduction, and penalty Form 2210 if necessary. **TAXTYME** then processes the information you enter, performs the necessary calculations, and produces a computer printout for direct transfer onto IRS forms. Requires a printer.

ONLY $59.95!

CP/M version for H89, H8, requires CP/M and MBASIC
ZDOS version for Z-100, requires ZDOS and ZBASIC

NEWLINE SOFTWARE

P.O. Box 402, Littleton, MA 01460 (617) 486-8535

ORDER NOW . . . SHIPMENTS BEGIN JANUARY 15, 1984

NAME ____________________
STREET ____________________
CITY ____________________
STATE ____________ ZIP ________

CHECK ONE
☐ H89, H8, CP/M
☐ Z-100, ZDOS

Send me ______ TAXTYME program(s) at $59.95 each.
Check one: ☐ payment enclosed ☐ send COD (add $3.00)
Send order to:
NEWLINE SOFTWARE, P.O. BOX 402, LITTLETON, MA 01460
Foreign orders: add $3.00 Airmail, $10.00 for non-U.S. checks

CP/M is a trademark of Digital Research, Inc.
MBASIC is a trademark of Microsoft, Inc.
ZDOS, Z-100 are trademarks of Zenith Data Systems, Inc.

Making the CP/M DUMP Program a Useful Utility

Charles E. Horn, P.E.
Horn Engineering Assoc.
1714 Patricia Lane
Garland, TX 75042

The DUMP program that is distributed as source code with the Digital Research CP/M operating system offers some good examples of programming techniques. It illustrates techniques for handling command line entries, opening and reading files, character conversion to hex representation, and exiting to CP/M without a warmboot, to name a few. However, as a useful utility it has little to offer.

The addition of a small amount of code can make DUMP a very useful utility. The additions that follow will provide true addresses, at least for files that originate at 100H, and will create an ASCII field similar to the DDT dump display. The resulting program, which we call ADUMP.COM, only occupies 1K bytes on disk, and is much easier than DDT to use if you only need a quick look to locate ASCII strings or to find the address of a particular instruction.

It is assumed that your DUMP program is Version 1.4 or later (earlier versions may not sign on and will require additional work). If so, the following additions will work:

1. Just ahead of the GLOOP label, change the LXI H instruction to:

```
        LXI  H,100H      ;Start at beginning of normal TPA
```

2. In the GLOOP routine, insert the following code just ahead of the comment "PRINT LINE NUMBER", following the instruction JNZ NUNUM.

```
;
;       PRINT ASCII BUFFER
;
;       This code prints the 16 character ASCII line
;       buffer located near the end of the program.
;
        PUSH H! PUSH D! PUSH B  ;Save environment
        LDA     ASCCNT          ;Get character position
        ANA     A               ;New line?
        JZ      LINENO          ;Yes - jump over this
        LXI     D,ASCBUF        ;Else line full - print it
        MVI     C,PRINTF        ;Print line function
        CALL    BDOS
        XRA     A               ;Reset
        STA     ASCCNT          ;..position count
LINENO: POP B! POP D! POP H     ;Restore environment
;
;
```

3. In the NONUM routine, after the MOV A,B instruction and ahead of the CALL PHEX instruction, insert this:

```
;
        CALL    ASCII           ;go build the ASCII buffer
;
```

4. Now add the following subroutine below the NONUM routine, following the instruction JMP GLOOP, just ahead of the FINIS routine:

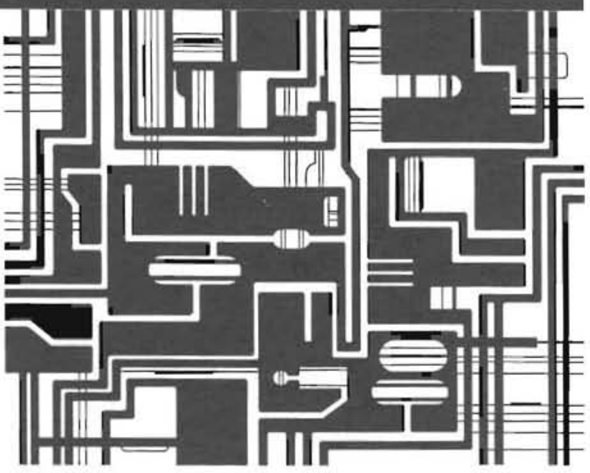
```
;
;       BUILD THE ASCII BUFFER
;
ASCII:  PUSH H! PUSH D! PUSH PSW  ;Save environment
        LXI     H,BUF16         ;Our ASCII line buffer
        LDA     ASCCNT          ;Character position in line
        MOV     E,A
        MVI     D,0
        DAD     D               ;Current position address in HL
        INR     A               ;Bump count
        STA     ASCCNT
        POP     PSW             ;Get current byte
        PUSH    PSW             ;Save again
        ANI     80H             ;Negative number?
        JNZ     PERIOD          ;Not ASCII
        POP     PSW             ;Get byte again
        PUSH    PSW             ;Save again
        CPI     ' '             ;ASCII?
        JC      PERIOD          ;If not
        CPI     'z'+1           ;Be sure of ASCII
        JNC     PERIOD          ;If not
        CPI     '$'             ;Don't allow false
        JZ      PERIOD          ;..PRINTF terminator
        MOV     M,A             ;Save ASCII in buffer
        JMP     ASCDONE
PERIOD: MVI     A,'.'           ;Get a period
        MOV     M,A             ;Store non-ASCII marker
ASCDONE: POP PSW! POP D! POP H  ;Restore environment
        RET
;
```

5. Near the end of the program, just below the label OPNMSG and above the "VARIABLE AREA", add the following:

```
;
;       ASCII BUFFER AREA
;
ASCCNT: DB      0               ;Line position counter
ASCBUF: DB      '    '          ;Four leading spaces
BUF16   DS      16              ;16 character line
        DB      '$'             ;PRINTF terminator
;
```

That's all you need to do. Note that this addition places a period in the ASCII field for each non-ASCII character. Also, note that one can not permit a '$' character to be printed because the BDOS PRINTF function would interpret it as the end of the string and would truncate the ASCII line display.

Introduction To Data Structures



Emily A. Yount
RR 1, Box 408
Danville, IN 46122

Data structures are organized ways to store and retrieve data. Generally, data structures are used in a program to make the program as efficient as possible. In particular, data structures are designed to enable a program to quickly retrieve the correct piece of data.

You need to understand and be able to use arrays in order to construct other more complex data structures, so before we go further, we'll review arrays. Arrays are ordered lists of data items. The entire list is identified by an array name, and the individual items on the list can be referenced by the array name followed by one or more subscripts. Different languages have different ways of implementing arrays, but usually space must be reserved for the array before the array is used. This is done by "declaring" the array. In BASIC, a DIM statement is used to set up an array, e.g. DIM <array name>(<integer expression>). In MBASIC the array name is any valid variable name and the integer expression is any valid integer expression that yields a positive integer when evaluated. This integer is then the maximum subscript value for the array. For example, DIM THISARRAY(20) sets up an array called "THISARRAY" that contains at most 21 elements, i.e. THISARRAY(0),...,THISARRAY(20). The default value for the minimum subscript value is zero, but an OPTION BASE 1 statement just before the DIM statement would cause the minimum subscript value to be 1.

Other languages have other means of declaring arrays. FORTRAN uses DIMENSION statements. In PASCAL, a type declaration statement is used, e.g. TYPE THISARRAY = ARRAY[0..20] OF INTEGER would set up an array of 21 integers called THISARRAY. Most languages also allow multidimensional arrays.

Stacks and queues are data structures that are frequently used. Many of you may have already encountered these structures, but their properties are worth reviewing before we go on to more complex structures.

A stack is a list of data in which the only data item that can be retrieved is the item that was most recently added to the list. A stack is also called a LIFO list (last in, first out). Adding an item to the stack is called "pushing" the item onto the stack, while retrieving an item is called "popping" an item from the stack. The classic example of something like a stack in everyday life is a stack of trays in a cafeteria. The problem for many novice programmers is just how to implement a stack in a high-level language. Perhaps this example in MBASIC will help.

First an array must be declared. This statement sets aside memory for use by the stack. We must estimate the maximum size of the stack. In our example we will use "N" as the maximum size of the stack. In practice N will be some constant.

```
10 REM ROUTINE TO SET UP A STACK
20 OPTION BASE 1
30 DIM STACK (N)
40 LET STACKTOP = 0
50 REM THE STACK IS INITIALLY EMPTY.
60 REM STACKTOP IS THE NUMBER OF ITEMS CURRENTLY IN THE STACK.
```

We also need routines for adding and deleting items from the stack.

```
1010 REM SUBROUTINE TO PUSH "ITEM" ONTO STACK
1020 IF STACKTOP >= N THEN GOTO 2010
1030 LET STACKTOP = STACKTOP + 1
1040 LET STACK(STACKTOP) = ITEM
1050 RETURN

1110 REM SUBROUTINE TO POP "ITEM" OFF OF STACK
1120 IF STACKTOP <= 0 THEN GOTO 2110
1120 LET ITEM = STACK(STACKTOP)
1130 LET STACKTOP = STACKTOP - 1
1140 RETURN

2010 REM ROUTINE TO HANDLE PROBLEM WHEN STACK IS FULL
.
.
2110 REM ROUTINE TO HANDLE PROBLEM WHEN STACK IS EMPTY
.
.
```

Queues are another common form of ordered list. In a queue, only the least recently added item can be retrieved. That is why queues are also called FIFO (first in, first out) lists. Items are added to the rear of the queue and retrieved from the front of the queue, just like a queue of customers in a cafeteria line (another classic example from everyday life). To implement a queue in MBASIC, one would begin by reserving space by declaring an array. Again, N is the maximum number of items in the queue, but this time it will be more convenient to have zero as the minimum subscript. You'll see why shortly.

```
10 REM ROUTINE TO SET UP A QUEUE
20 DIM QUEUE(N-1)
30 LET FRONT = 0
40 LET REAR = 0
50 REM THE QUEUE IS INITIALLY EMPTY

1010 REM SUBROUTINE FOR ADDING AN ITEM TO A QUEUE
1020 LET REAR = (REAR + 1) MOD N
1030 IF FRONT = REAR THEN GOTO 2010
```

```
1040 LET QUEUE(REAR) = ITEM
1050 RETURN
1110 REM SUBROUTINE FOR REMOVINGS AN ITEM FROM A QUEUE
1120 IF FRONT = REAR THEN GOTO 2110
1130 LET FRONT = (FRONT + 1) MOD N
1140 LET ITEM = QUEUE(FRONT)
1150 RETURN

2010 REM ROUTINE FOR HANDLING PROBLEM WHEN QUEUE IS FULL
.
.
2110 REM ROUTINE FOR HANDLING PROBLEM WHEN QUEUE IS EMPTY
.
```

The routines used to implement a queue are a little more complex than those used for a stack. The reasons for setting up a queue in this fashion may not be obvious at first, but they aren't really that difficult to understand. If we simply added items to the rear of the queue and removed items from the front, the queue would move through memory until we added an item in the last position we had allotted to the queue. Then the queue would seem "full" even though there might be a lot of space left in front of the queue, space created by deleting items from the front of the queue. We want to be sure that this space is not wasted. So by using integer arithmetic, we program the queue so that after filling the space at QUEUE(N-1), the next item is placed at QUEUE(0) if that space is free, i.e. FRONT > 0. If you are not familiar with the use of MOD in integer arithmetic, X MOD Y is equal to the remainder left when X is divided by Y.

You may have noticed that the test for a full queue in the insertion subroutine is the same as the test for an empty queue in the deletion subroutine, i.e. does FRONT = REAR? However, in the insertion subroutine, when FRONT = REAR, there is really one free space since FRONT points to the space before the first element in the queue. Why can't we use this space? Well, we could if we wanted to make things more complicated. As things stand, if we inserted an item into this space, we wouldn't be able to tell whether or not the queue was empty or full.

Another, less commonly used data structure, is a deque (pronounced "deck"), or double-ended queue. In a deque, data may be added and removed at both ends of the list, but not in the middle.

The data structures described above have all been examples of sequential data allocation, which means the data is stored in a sequence of memory locations. A much more flexible data allocation method is known as linked allocation. The simplest example of linked allocation is a linked list. Think of a linked list as a list of nodes containing two fields, one an information field (DATA) and the other a link field (LINK%), see Figure 1. DATA may be defined as a string or real number but LINK% will be an integer. Although in our examples there will be a "%" at the end of every integer variable, you should remember that using DEF statements not only decreases the amount of storage needed, but diminishes the number of times you must use the shift key when typing, thus making it easier to type in your program. Information is stored in the DATA fields and each LINK% field contains the subscript or address of the next node in the list, i.e. the LINK% field "points" to the next node in the list, see Figure 1. Here is an example of how to implement a linked list in BASIC.

The first step would be to set aside space for the list. We will use dimension statements to set up arrays. As before, N is the maximum length of the list.

```
10 OPTION BASE 1
20 DIM DATUM(N)
30 DIM LINK%(N)
```

The next step would be to initialize the storage pool, i.e. the list of nodes that are available for use in our list. We set the variable AVAIL% to point to the first available node and then link all the nodes together in order. They may not stay "in order" but they are usually linked in order at first. Also, we link the last node to zero; this way we can test whether or not we have reached the end of the list. The last node is the only one that will ever be linked to zero.

```
110 REM INITIALIZATION OF LINKED LIST
120 FOR I = 1 TO N - 1 STEP 1
130 LET LINK%(I) = I + 1
140 NEXT I
150 LET LINK%(N) = 0
160 LET AVAIL% = 1
```

Now to use our list, we must have a subroutine for getting a new node from the storage pool. We will call this new node NEWNODE%. NEWNODE% will be the first node on the storage pool list if there is a node available.

```
1010 REM SUBROUTINE FOR GETTING A NEW NODE
1020 IF AVAIL% = 0 THEN GOTO 1100
1030 LET NEWNODE% = AVAIL%
1040 LET AVAIL% = LINK%(AVAIL%)
1050 RETURN

1100 REM ROUTINE FOR HANDLING PROBLEM WHEN THERE ARE NO MORE
NODES ON THE LIST
```

We also need a subroutine for returning nodes to the list of available nodes. USEDNODE is a node we don't want to use any more right now, but we want to keep it on our list of available nodes in case we need it later.

```
1210 REM SUBROUTINE FOR RETURNING USED NODES TO THE LIST OF
AVAILABLE NODES
1220 LET LINK%(USEDNODE%) = AVAIL%
1230 LET AVAIL% = USEDNODE%
1240 RETURN
```

The observant reader will have noticed that our list of available nodes is a stack. We "pop" nodes off the stack when we get new nodes and "push" nodes onto the stack when we return them. So we are already using two other data structures (arrays and a stack) to implement our linked list. Now the question is, just what do we do with our nodes and how do we do it? Well, we can use our list to store data in order (e.g. alphabetical or numerical order) when the data is not received or entered in order. First we need a routine for creating a list. The following routine will create a new list. The first node will be the "BEGINNING%" and the next node will be the "ENDNODE%". We use our subroutine at line 1010 to create these nodes. "BEGINNING%" and "EN-

I	LINK%(I)	DATUM(I)
1	4	0
2	0	30
3	7	20
4	3	10
5	2	50
6	5	40
7	6	30

Figure 1

DNODE%" are labels that tell our program where the list starts and where it ends. Initially, the "BEGINNING%" node is linked directly to the "ENDNODE%" node.

```
210 ROUTINE FOR SETTING UP A NEW ORDERED LINKED LIST
220 GOSUB 1010
230 LET BEGINNING% = NEWNODE%
240 GOSUB 1010
250 LET ENDNODE% = NEWNODE%
260 LET LINK%(BEGINNING%) = ENDNODE%
270 LET LINK%(ENDNODE%) = 0
```

Now we want to create a program to search the list to find out if an item (ITEM) is in the list. If it is in the list, we will do nothing, if it is not in the list, we will insert it so that each DATUM in the list is still in proper order. We need not enter the items in order however. Before looking at the code below, why don't you try to program this yourself?

```
310 REM SEARCH AND INSERT SUBROUTINE
320 LET B% = BEGINNING%
330 LET A% = LINK%(B%)
340 LET DATUM(ENDNODE%) = ITEM
350 WHILE DATUM(A%) < ITEM
360 LET B% = A%
370 LET A% = LINK%(B%)
380 WEND
390 IF (DATUM(A%) = ITEM AND (A% <> ENDNODE%) THEN GOTO 460
400 GOSUB 1010
410 LET C% = NEWNODE%
420 REM INSERT C% BETWEEN A% AND B%
430 LET DATUM(C%) =  ITEM
440 LET LINK%(C%) = A%
450 LET LINK%(B%) = C%
460 RETURN
```

As you can see, B% and A% are pointers to nodes. These pointers move down the list with A% always one node farther down the list than B%. You may be wondering why we have line 340 in our program. The reason is that we now use the same expression (i.e. is DATUM(A%) < ITEM?) to determine if we are at the end of the list and to determine whether or not we have found either ITEM itself or the place to insert ITEM. The WHILE loop is used to traverse, i.e. move down, the list. When DATUM(A%) <= ITEM we stop. If DATUM(A%) = ITEM, we do nothing so we return, unless we have gone through the entire list without finding ITEM. In that case we insert ITEM just before ENDNODE%, i.e. at the end of the list. We are using ENDNODE as a "sentinel" node, that is, a node that contains a special value that we use to tell us when we are at the end of the list.

For some applications we will want to delete items from our list. We need one more subroutine, one that deletes an ITEM.

```
510 REM DELETION SUBROUTINE
520 LET B% = BEGINNING%
530 LET A% = LINK%(B%)
540 LET DATUM(ENDNODE%) = ITEM
550 WHILE DATUM(A%) < ITEM
560 LET B% = A%
570 LET A% = LINK%(B%)
580 WEND
590 IF (DATUM(A%) > ITEM) OR (A% = ENDNODE%) THEN 650
600 LET LINK(B%) = LINK(A%)
610 LET USEDNODE% = A%
630 GOSUB 1210
640 GOTO 660
650 PRINT ITEM "is not in the list."
660 RETURN
```

Now you should have a good idea about how to create a linked list. Think of a program that you could use to apply your knowledge. At this point a simple program to practice what you've learned will do. Later in this series, we'll examine other data structures that are more efficient than a linked list.

Those of you who use PASCAL will probably realize by now that using pointer variables would make things much simpler. Also, standard PASCAL has two library procedures, "NEW" and "DISPOSE", that take the place of our subroutines at lines 1010 and 1210 respectively. In PASCAL we would start by declaring our list as shown below. We no longer have to set up the storage pool, since PASCAL's procedure NEW will take care of getting available nodes for us. It is assumed that DATATYPE has already been declared.

```
TYPE LISTPTR = ^LISTNODE;
     LISTNODE = RECORD
                     DATA : DATATYPE;
                     LINK : LISTPTR
                 END;
     BEGINNING, ENDNODE = LISTPTR;
     ITEM = DATATYPE
```

We would set up our list with the following statements.

```
NEW(BEGINNING);
BEGINNING^.LINK := ENDNODE;
ENDNODE^.LINK := NIL;
```

In PASCAL, NIL is a null pointer, i.e. the value to which a pointer points when the pointer doesn't point to anything. We could use this as a way of testing whether or not we are at the end of our list, but using ENDNODE as a sentinel is slightly faster.

You may be wondering what the advantages of linked allocation are. There are several. The main advantages are ease of insertion and deletion, and ease of merging two lists or separating one list. However, linked allocation uses more memory than sequential allocation since the LINK fields take up space. Also, in order to find something in a linked list, the list must be traversed sequentially; and one would have to traverse the entire list to find the last item. Other more complex data structures do not have this disadvantage.

Linked lists are a good introduction to linked structures in general. In a future article we'll study some of these other structures and learn some of their uses. In the meantime, those of you who wish to study data structures in more detail should try to find one or more of the references given below. Spracklen's book will be particularly useful to assembly language programmers while Wirth's book will be best for PASCAL programmers.

Bibliography

1. Horowitz, E., and Sahni, S., Fundamentals of Data Structures, Computer Science Press, Rockville, Maryland, 1982.

2. Knuth, D. E., The Art of Computer Programming, Vol. 1, Fundamental Algorithms, 2nd Ed., Addison-Wesley, Reading, Mass., 1973.

3. Spracklen, K., Z-80 and 8080 Assembly Language Programming, Hayden Book Co., Inc., Rochelle Park, N.J., 1979.

4. Standish, T.A., Data Structure Techniques, Addison-Wesley, Reading, Mass., 1980.

5. Wirth, Niklaus, Algorithms + Data Structures = Programs, Prentice-Hall, Englewood Cliffs, N.J., 1976.

Part IV

An Introduction To 'C'

Brian Polk
86-02 Little Neck Parkway
Floral Park, NY 11001

This is the fourth in a series of articles designed to introduce the 'C' programming language.

We are going to continue analyzing the 'more' program presented in the prior article. I have to explain more of the program. We will also modify the program in order to be even more useful.

If you went through the exercise of compiling the program, you would have had a not very useful program which took input from the terminal and echoed it back, twenty three lines at a time. There are two ways to alleviate this problem. We will look at both ways.

The first way involves no changes to the program. The C/80 compiler makes available one of the most useful aspects of the UNIX operating system — I/O redirection. This allows us to temporarily change the standard input and output device from the terminal to another file or device. In our case, we want to change the standard input from the terminal to a file to be printed. The redirection symbol is '<' for input and '>' for output, followed immediately (no spaces) with the file or device name. Therefore, in order to list out a file at the terminal using our 'more' program, we could say:

```
more <filename
```

where 'filename' is the name of the file to be listed. Notice that we can direct the output to another file, or to the printer, by saying:

```
more <filename >lp:
```

This is not a very useful way to print a file, because our program will still pause after every 23 lines, even though the output is going to the printer. Notice that in this case the 'More' prompt also is sent to the printer, the result depending on the printer.

The second way to specify the file name to be printed will end up being much more useful, but it involves changing the program. What we want to do is pass a parameter to the program so that it will know the name of the file to be opened for input. 'C' provides a built in mechanism which makes available to us any parameters specified on the command line. These parameters are made available by way of an integer, which counts how many parameters were entered, and a character array, which holds the parameters. In order to add this logic to our program we have to talk a little about arrays.

We can define an array as a sequence of integer or character variables by putting a number in braces ('[' and ']') immediately following the variable name. The number indicates how many positions the array contains. One thing to note: 'C' arrays start with position '0' and end with the position one less than the number specified. Therefore:

```
char x[10];
```

declares a ten position character array starting with x[0] and ending with x[9]. We also need to understand the concept of 'pointers' in order to pass a parameter to our program. A pointer is an address of a variable in storage. Arrays are passed by specifying a pointer to the address of the location of the first element of the array. In 'C', a pointer variable is indicated by an asterisk ('*') immediately preceeding the variable name. For a more detailed explanation of arrays and pointers, check your 'C' programming language book. We will go into more detail in future articles.

Now let's modify our program to incorporate the concept of parameters.

```
#include "printf.c"
#define EOF -1
#define NULL 0
main(argc, argv)  /* here we indicate the parms */
int argc;         /* 'argc' is the number of command-line arguments */
char *argv[];     /* 'argv' is an array of pointers to the arguments */
{                 /*       entered on the command line */
int c, line_number, channel;

channel=fopen(argv[1],"r");  /* let's open the file name specified */
if (channel == NULL)         /* if file not found we will exit */
   {
   printf("File Not Found.\n")
   exit(4);
   }
for (line_number=0; (c=getc(channel)) != EOF; putchar(c))
   {                        /* use 'getc' with channel number */
   if (c == '\n')
      line_number++;
   if (line_number == 23)
      {
      putchar(27);
      putchar(106);  /* save the cursor position */
      putchar(27);
      putchar(120);
      putchar(49);   /* turn on the 25th line */
      putchar(27);
      putchar(89);
      putchar(56);
      putchar(32);   /* position cursor on the 25th line */
      putchar(27);
      putchar(112);  /* enter reverse video */
      printf("--More--");
      putchar(27);
      putchar(113);  /* exit reverse video */
      while (getchar() != '\n');  /* wait for a carriage return */
      putchar(27);
      putchar(107);  /* return cursor to saved position */
      putchar(27);
      putchar(121);
      putchar(49);   /* turn off 25th line */
      line_number = 0;
      }
   }
}
```

Now we can run our program by simply saying:

```
more filename
```

'argc' and 'argv' are the names used by convention for the parameter count and array, but any variables declared the same way can be used. Notice the '*' in 'char *argv[]' and the lack of a number between the braces. The asterisk indicates that we are declaring an array of POINTERS to characters as opposed to an array of characters. The reason we don't need to put a number in brackets is because the array is being passed into the program and has already been declared in the calling program (which in this case is the HDOS-C/80 interface). This fact is true for any array which is passed. Note that the passed variables must be declared BEFORE the opening bracket of the program.

Notice two things where we refer to the 'argv' array. The first is that we reference position number one. Why not position zero? 'argv[0]' refers to the command name ('more' in this case). Therefore, 'argv[1]' refers to the first parameter entered. The second thing is why we didn't put the '*' before 'argv'? Remember that we declared an array of pointers to characters by putting the asterisk before the variable. Conversely, we refer to the actual character by removing the asterisk. The opposite is also true. If we declared an array of characters (e.g. char argv[10]), we could refer to the pointer to a character by adding an asterisk before the variable name (e.g. *argv[1]). If this all seems a little confusing, join the club. I've probably gone into a little more detail than necessary here. If you are interested, check your manual. If not, we will return to this topic in the near future.

The 'exit(4)' statement closes all open files and returns to the calling program (in this case, the operating system). We return the value '4' to indicate an abnormal condition.

I still have to comment on other facilities used in the program. The first is '#define'. We can use this to set up global constants used throughout the program. By convention, these global constants are indicated in upper case. These are often collected into a file which is included into every program written. We may do that as an exercise in a later article. Here, I have defined two very useful constants: 'EOF' which is the end- of-file value returned from all input routines, and 'NULL' which is returned from many routines (such as 'fopen') when an error has occurred. We will add to our list of 'defines' as we go along.

In part two of the 'for' statement, I constructed what might seem like unusual syntax. What this illustrates is a quite general rule in 'C' — in any context where it is permissible to use the value of a variable of some type, you can use an expression of that type. What this means is that since the variable 'c' and the command 'getchar()' are both integers, we can combine them into '(c = getchar())' and still yield an integer. Its value is simply the value being assigned to the left hand side. Also in the 'for' statement, it should have been obvious that '!=' means 'not equal'. In a following 'if' statement we use '==' which means 'equal to'. The double equal sign differentiates it from a normal assignment statement.

The 'line number' 'if' statement used 23 instead of 24 to provide for an overlapping line on the screen. This is often very helpful when looking at long listings in order to provide consistency from screen to screen and to compensate for inadvertently missing the last line on the prior screen.

Be careful to distinguish between the single (') and double (") quotes. I ran into some trouble when I mistakenly used the single quote in the 'printf' statement. As a general rule, when you are enclosing a single character, use the single quote; otherwise, use double quotes. Note that '\n' is a SINGLE character which is why we enclosed it in single quotes in a previous program.

Also be careful when sending escape sequences to the terminal. Make sure you are sending the decimal equivalent, NOT the character constant. I spent a lot of time trying to figure out why the sequence 27-120-1 would not turn on the 25th line. The proper sequence is 27-120-49 where '49' is the decimal equivalent of the character '1'. This can also be avoided by specifying the character constant in the 'putchar' statement, as in:

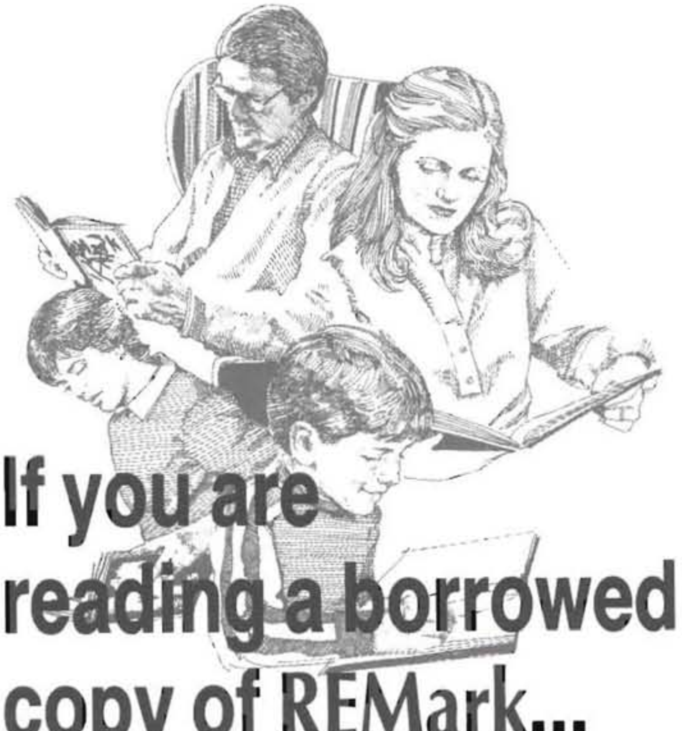
```
putchar(27);
putchar('x');
putchar('1');
```

By now you probably figured out why I used 'getc(0)' instead of 'getchar()' in the 'while' statement. This was because when we used I/O redirection, the normal terminal input had been redirected to the specified file name. Therefore, we had to specify that we wanted the character to come from the terminal by using channel 0, which is always reserved for terminal I/O, even if redirection has been used. When we passed the name of the file directly to the program, we could use the normal 'getchar()' sequence.

Once you feel comfortable with the program, try modifying it to allow a way to exit if another character (such as 'q') is entered at the point where we are currently waiting for a carriage return. This would allow us to stop without having to go the end of a long listing.

Next time we will show how to pass parameter switches to a program and also how to use the formatted input equivalent of 'printf'.

'C' you later!



If you are reading a borrowed copy of REMark...

maybe now is the time to join the National Heath/Zenith Users' Group. You will receive:

- *a copy of **REMark** filled with new and exciting articles and programs each month*
- *access to the HUG library filled with a large variety of programs*
- *discounts on a variety of Heath/Zenith computer products (see REMark January, 1984 issue for more details)*

And remember, your local HUG is an excellent source of information, support and comradery. A membership package from the National Heath/Zenith Users' Group contains a list of current local HUG clubs as well as other interesting information.

HUG NEW PRODUCTS

885-1234[-37] CP/M

HAMHELP $16.00

Introduction: HAMHELP makes use of the personal computer to calculate the MUF (Maximum Useable Frequency) for each half-hour throughout the day. Input data for the program can be obtained each hour from listening to WWV. HAMHELP is a valuable tool for the amateur radio operator.

Requirements: HAMHELP requires the CP/M operating system, version 2.2.03 or higher, an H8/H17 or H/Z-89 with one drive and at least 48K of memory. The soft-sectored version of this program (885-1234-37) will run on the H/Z100 under CP/M-85. The program itself is compiled MBASIC (MBASIC is not required).

The following programs and files are included on HUG Disk P/N 885-1234[-37]:

README .DOC
HAMHELP .COM
FILEFIX .COM
FOREIGN .LOC
NOAMER .LOC
HAMHELP .DOC

Author: Ray (Raymond S.) Isenson, N6UE

Program Content: HAMHELP calculates the MUF (Maximum Useable Frequency) each half-hour throughout the day for the date and path between two stations specified by the user. The output of these calculations is then presented in a graphic chart which is easy to read. Along with the chart, it prints additional information including the antenna azimuth, optimum elevation beam angles, the expected path attenuation, and an estimate of the likely propagation conditions as a function of the existing electromagnetic environment. If the two stations are more than some 4000 kilometers apart, the program will optionally calculate the exact times of Sunrise and Sunset for each of the locations. Also, the program will check for any unusual possibilities such as "Grayline" longpath openings or preferred paths to take advantage of certain bad or good polar cap propagation conditions.

FILEFIX.COM is included in order to make the HAMHELP program as "user friendly" as possible. Two lists of cities, each containing the names of 36 cities, along with their respective latitudes and longitudes, are stored in the files called:

FOREIGN .LOC
NOAMER .LOC

The computer presents one of these lists on command to the operator who then selects the remote city by number. The program, FILEFIX.COM, is used to modify the two data files so that the lists contain cities that are of major interest to the individual. HAMHELP will allow the user to enter the names and locations.

Comments: The HAMHELP program is limited by the lack of some math functions which result in only slight errors in final calculations. These limitations are well documented for the user in the HAMHELP.DOC included with the disk. The program is valuable and handy for the amateur radio operator.

TABLE C Rating: (0),(9)

885-8026 HDOS

SPACE DROP $16.00

Introduction: Space Drop is a video action game for Heath/Zenith microcomputers. The game uses the H/Z-19 graphic capabilities. The game itself involves an attack by hostile aliens. You are to stop the aliens while trying to avoid being hit.

Requirements: Space Drop is game which runs under the HDOS 2.0 operating system. The program makes use of either the H8/H17/H/Z-19 or the H/Z-89 with 16K of memory. Only one disk is required.

The game is an executable program, ready to play.

Note: The game makes extensive use of the features available in the H/Z-19 graphics terminal.

The following files are included on the HUG Disk P/N 885-8026:

README .DOC
SPACDROP.DOC
SPACDROP.ABS
SPACDROP.ASM

Note: The source code is included.

Author: Bruce W. Markell

SPACDROP — This graphic game gives the player three "Artillery Ships" to engage in a battle with hostile aliens. The player uses the keypad to move the ship right or left in an effort to get a shot at destroying the enemy which increase in numbers as the game progresses. The four (4) key moves the player's ship to the left. The five (5) key stops the player's ship. And, the six (6) key moves the player's ship to the right. Firing at the aliens is accomplished by using the "A" key.

Other special functions include the space key and the ESCape key on the terminal. The space bar is used to "freeze" the action. The ESC key is used to terminate the game in process and return to HDOS.

SPACDROP.ASM — The source code for the SPACDROP program is included on the disk. This code can be used by the experienced programmer to modify the existing SPACDROP program.

Comments: Space Drop is similiar to other video action games with horizontal movement of the player's ship and a combination of horizontal and vertical movement from the attacking aliens. A larger attack alien appears randomly to keep the pace of the game moving. This larger enemy must be destroyed as it appears or the alien will destroy the player's ship.

TABLE C Rating: (0),(1),(2),(9)

885-8025-37 CP/M-85/86 FAST EDDY Text Editor and BIG EDDY $20.00

Introduction: FAST EDDY is a text file screen editor that was written for everybody. It was written using the basic commands and keypad keys, so that anyone, even with no experience with an editor, can learn to use it while reading the instructions.

For those files that are too large for your computers memory, BIG EDDY will handle the breaking up of the text for editing with FAST EDDY.

Requirements: This disk requires the CP/M-85 or CP/M-86 operating systems on an H/Z-100 computer. A printer is not required, but both FAST EDDY and BIG EDDY have printer options. Only one disk drive is required.

BIG EDDY can be used with large files. A second drive (or high density drives) may be required to break up large files which cannot fit into memory. The original file is not changed or deleted.

The following files are included on the HUG P/N 885-8025-37 CP/M-85/86 FAST EDDY Text Editor and BIG EDDY File Handling Utility:

EDITOR	.COM
BIGED	.COM
BIGED	.DOC
INSTRUCT	.DOC
TUTOR1	.DOC
TUTOR2	.DOC
TUTOR3	.DOC
RETURN	.COM

Author: Hubert L. Reeder

FAST EDDY — This text file screen editor and its documentation have been designed for anyone not familiar with using an editor. The program uses commands and keys that are easy to remember and use.

The editor contains a limited number of commands, however, the commands are designed to provide a useful, easy to use editor. It does not have complex options that require time and effort to use. The editor contains a command mode and edit mode. The following are a brief list of the options:

COMMAND MODE

Typed Commands:

LOAD filename.ext	(load file)
SAVE filename.ext	(save file)
SAVE XX filename.ext	(save XX number of lines)
MERGE filename.ext	(merge two files)
PRINT	(print enter file, NN lines per page)
PRINT NN	(print double spaced)
FIND anyword	(find the first occurrence of a word)
MARGIN nn xx	(set left margin, nn, right margin, xx)
CMPRESS	(replace spaces with tabs in new text)
EXPAND	(cancel the CMPRES command)
BYE	(exit to CP/M)

Key Commands:

Up arrow - enter EDIT mode at first line of text
Down arrow - enter EDIT mode at last line of text
HOME - enter EDIT mode at pointer (last cursor location)
DELETE - cancel partial commands or stop printer
F0 - erase all text

EDIT MODE

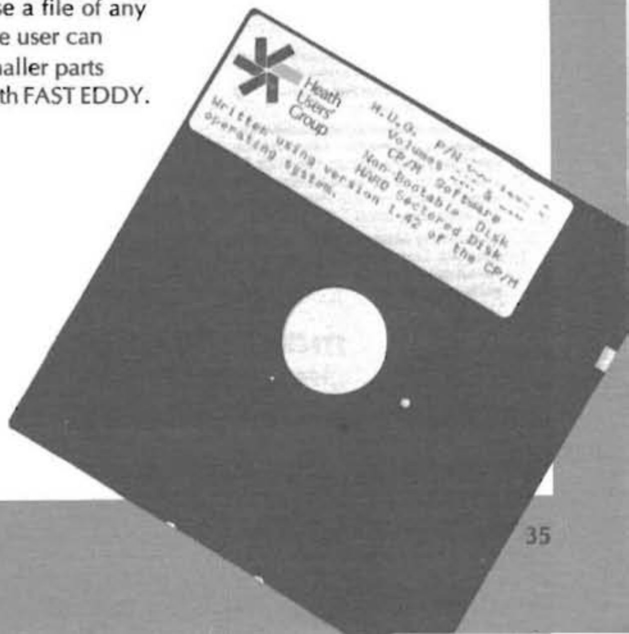
Key Commands:

Up arrow - move cursor up one line
Down arrow - move cursor down one line
Right arrow - move cursor to the right one character
Left arrow - move cursor to the left one character
HOME - return to COMMAND mode
IL - insert line
DL - delete line
IC - insert character
DC - delete character
f0 - block erase
f1 - align paragraph within left and right margins
f2 - justify right
f3 - indent on/off
f4 - margin off
f5 - split line
f6 - find next occurrence of word, after FIND of COMMAND mode
f7 - move backward in text
f8 - move forward in text
f9 - center line
f10 - tab set/release
f11 - jump left
f12 - jump right

These are most of the basic commands of FAST EDDY. Please note that it has the ability to align paragraphs to new margin settings and then the option of right justifying the paragraph text.

Details of how to use these options are contained in the documentation. The TUTOR1, TUTOR2, and TUTOR3 documentation files are included with the disk to give the user experience in using FAST EDDY while reading the doc files.

BIG EDDY — This program is a utility to work with text files which are too large to be edited by FAST EDDY directly because of memory limitations. BIG EDDY can be used to browse a file of any size which the user can break into smaller parts for editing with FAST EDDY.



BIG EDDY asks for the input filename and an output filename. It keeps track of the subfiles and names them accordingly.

BIG EDDY has some useful options to aid the user in preparing the text for smaller files. The BROWSE mode is similiar to the EDIT mode of FAST EDDY, except that no editing can be done to the file.

The following are a list of the commands of BIG EDDY:

SAVEALL - save the entire text in memory to the disk
SAVEPART - save part of the text in memory to disk
NOSAVE - discard part of text
PRINT - same as FAST EDDY's print commands
BYE - exit to CP/M

With the SAVEPART command, the user can save the text by subject or modules of his choice. Using the CP/M PIP program, the subfiles can assemble the files into any order.

Comments: This version of FAST EDDY, with the editing features e.g. align paragraph and right justify, allow formatting features that make it a powerful, easy to use editor.

TABLE C Rating: (1),(3),(10)

The following two HDOS products are available in soft sectored format beginning this month:

885-1030[-37] Disk III, Games II
885-1096[-37] MBASIC Action Games

Refer to the HUG Software Catalog for descriptions of the products.

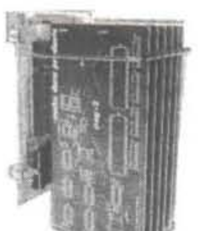
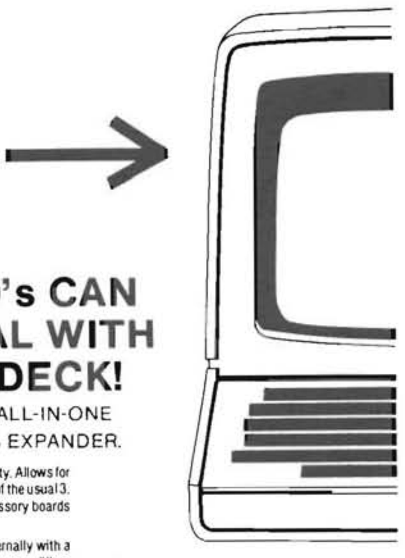
Please remember the "-37" indicates that you want a soft sectored disk. If you do not include the "-37", you will receive hard sectored.

NOTICE

Heath/Zenith Support Vendor

This could have been your advertising space to reach over 20,000 HUG members!

Call Donna at HUG today for more information (616-982-3794).

H89/Z90's CAN NOW DEAL WITH A FULL DECK!

THE ORIGINAL ALL-IN-ONE ACCESSORY BUS EXPANDER.

MH89+3 doubles expansion capacity. Allows for for 6 right-hand type cards instead of the usual 3. Room at last to run those neat accessory boards you've seen advertised!

Piggyback motherboard installs internally with a screwdriver in just minutes – with no modifications! 3 slots exactly duplicate the originals. The 3 added slots occupy unused addresses and eliminate previous conflicts. 100% compatible with all accessory boards!

No overheating problems! Simple design draws little power. Leaves plenty of overhead for the minimal load of most accessories. Full technical information provided.

The best news about this "No-hassle" design is the price — **ONLY $150.** About 1/3 the price of other solutions!

Price includes assembled and tested MH89+3 expander, complete instructions and one (1) year warranty. CA residents add 6% tax. USA include $5 shipping. Foreign add $10. Telephone and COD orders accepted.

mako data products
1441-B N. RED GUM, ANAHEIM, CA 92806
PHONE (714) 632-8583

FLOPPY DISK CONTROLLER

Controls Any Combination Of Up To Four
8" and 5¼" Drives

This easy to install plug in board can control any combination of single or double sided, single or double density drives.

Designed especially for H88/H89 users.

- Fully compatible Bios supplied for your CP/M 2.2 operating system
- Easy to follow instructions
- Contains controller board with boot prom
- Order cables for connection $15 (HFDC-110)
- ***Introductory Offer $395,*** *Order HFDC-100*

NORTH COAST INTELLIGENCE INC.
1201 Cherokee Trail
Willoughby, Ohio 44094
Phone: 216-946-7756

Check, COD, VISA or MC — 90 Day Warranty

Get Rid Of "Echo On Delete" In CP/M-86

Pat Swayne
Software Engineer

Well, here we go again, folks! A new version of CP/M is out, and like all of its predecessors, it has the "feature" of echoing a character if you delete it with the DELETE key instead of the BACK SPACE key. As we did with CP/M-85 in REMark Issue #41 (June 1983, page 41), we can "fix" the DELETE key so that it deletes, using DDT to make a patch. In this case, you need DDT86 which is supplied with CP/M-86. Run it and load in CPM.SYS as in this example:

```
A>DDT86
DDT86 1.1
-RCPM.SYS
  START      END
1681:0000 1681:4EFF
```

Now, use the L command to disassemble the area where the patch goes. It should look like this:

```
-L1560
1681:1560 OR     CH,CH
1681:1562 JZ     154A
1681:1564 DEC    CH
1681:1566 MOV    AL,[22E5]
1681:1569 MOV    [22E3],AL
1681:156C JMPS   15BF
1681:156E CMP    AL,7F
1681:1570 JNZ    157E
1681:1572 OR     CH,CH
1681:1574 JZ     154A
1681:1576 ES:    MOV      AL,[BX]
1681:1579 DEC    CH
```

If you have Heath/Zenith CP/M-86 version 1.1, Release 1.10, the code should look just like this. If you have a different version and the code is similar, but the numbers are different, try the patch anyway (but not on your only system disk). If the code looks nothing like this (as it might in a non-Heath/Zenith version), use the L command to look around for code like this. When you are satisfied that you have found the right place in the code, use the A command to replace the second occurrence of OR CH,CH with a jump as follows:

```
-A1572
1681:1572 JMPS 1560
1681:1574 .            (type a period)
```

If you are patching a version with the code in a different place, the destination of the JMPS instruction should be the line with the first OR CH,CH. After you make the patch, save the patched code with the W command, and exit DDT86 with control-C.

```
-WCPM.NEW
-^C
```

Next, rename the old and new files so that the new one replaces the old one:

```
A>STAT CPM.SYS $R/W        (remove write protect)
A>REN CPM.OLD=CPM.SYS
A>REN CPM.SYS=CPM.NEW
```

Now you can re-boot your system and test the patch. If it works OK, you can delete CPM.OLD and set the Read-Only and System attributes on your new CPM.SYS:

```
A>ERA CPM.OLD
A>STAT CPM.SYS $R/O
A>STAT CPM.SYS $SYS
```

And that's all there is to it!

The ILLUSTRATOR

The **ILLUSTRATOR** is a full-featured graphics drawing program for use with the Z-100 pixel graphics (w/wo color) or the H89/H19 IMAGINATOR pixel graphics option. No need for a lightpen.

Features Include:

- Turtle Graphics
- Rubber Banding
- Line Drawing
- Box Drawing
- Circle Drawing
- Ellipse Drawing
- Diamond Drawing
- Screen Print
- Dot Cursor Control
- Area Fill
- Box Fill
- Circle Fill
- Diamond Fill
- Copy Area
- Erase Area
- Text Mode
- Invert Mode
- Help Display
- 64 Colors
- Color Define
- Color Pattern
- Screen Save
- Screen Restore
- Area Save
- Area Restore
- Compacted Files
- more. . .

ONLY $89.95!

HDOS version for H89, H8, requires IMAGINATOR
ZDOS version for Z-100, color memory optional
Supports many dot graphics printers. Call for info.

NEWLINE SOFTWARE
P.O. Box 402, Littleton, MA 01460 (617) 486-8535

NAME ____________________ CHECK ONE
STREET ____________________ ☐ H89, H8/H19, HDOS
CITY ____________________
STATE __________ ZIP __________ ☐ Z-100, ZDOS
Send me ______ "The ILLUSTRATOR" program(s) at $89.95 each.
Check one: ☐ payment enclosed ☐ send COD (add $4.00)
Send order to:
NEWLINE SOFTWARE, P.O. BOX 402, LITTLETON, MA 01460
Foreign orders: add $3.00 Airmail, $10.00 for non-U.S. checks

The Stupid Computer You Should Be So Smart!

Wallace M Theodore
PO Box 2488
Hammond, Indiana 46323-0488

Somewhere, sometime, somehow, some "very wise" person made the statement that computers are stupid - implying, of course, they are only capable of performing tasks when instructed by human beings of far superior intelligence. Well, that may be so, but I'LL never believe it.

Other humans of equally grandiose wisdom immediately and blindly latched on to the erroneous line and through the years have repeatedly put down the intelligence of the defenseless computer. Such nonsensical chatter has bugged me for many moons and is probably enough to bring a tear to my Heath/Zenith CRT.

True, the computer IS a machine and, in its infancy (the stage before it acquires the bulk of its intelligence) is of little value to the user or anyone else, save for occasional use as a door stop, paperweight, or sinker for deep sea fishing.

But to label the computer as stupid, dumb, or merely a pile of plastic, silicon, glass, and metal or other equally descriptive insults is an INSULT to the computer! I, for one, think the time has come for someone to stand up for the computer so, at the risk of being banished from my big blue marble, I will attempt to compare the computer with the unmistakable and indisputable superiority of the human.

A baby, at birth, knows virtually nothing save for some instincts that come with it at delivery. The baby cannot talk, eat alone, sit, stand or perform many other seemingly menial tasks, nor does it obey when you tell it to settle down for the night and get its much needed rest. As the months and years progress, the infant begins to gain knowledge from those around it who work in many ways to TEACH the baby what it must know to survive.

The computer, as the baby, also arrives with precious few capabilities. You, the superior adult human being, might smugly say you ARE superior to the computer as well, at that stage, you MAY be - and then you again may NOT. You, incidentally, also have a couple of years of learning on the computer.

At its worst, the computer is STILL one up on the human, even at delivery. While it took you years of trial and error to gain your not-necessarily-correct knowledge, the computer is instantly more ACCURATELY knowledgeable than you the minute the operating system is installed. And YOU had to "teach" it nothing - even if you could, which you probably could not.

Even BEFORE the operating system is installed, the computer will still outperform you, hands down! Don't believe it? Try checking the RAM by hand and, if you make it, match the time it took you against the time it takes the infant computer.

Of course, there is the argument that a baby is born with some intelligence while the computer has none. Wrong. Long before the final assembly of your computer, the read only memory (ROM) chips are loaded and waiting for installation: instant knowledge to the fledgling computer!

As a matter of fact, when my daughter was born she cried when she wanted attention because there was nothing else she knew how to do. She has since progressed to "goo goo", "Da Da", "Can I have a dollar?", and "Can we go to Hawaii some time? Can we? Huh? Can we?". This, of course, is the normal progression of the human species (the last question, of the human teenage FEMALE species).

When my computer was first plugged in and turned on, it went "Beep Beep" which, as you know, it will continue to say until its final days: clear, concise, ADULT computer talk at its very first utterance! Possibly this borders on 'instinct' that is inborn in living things. And at the end of a long, busy day of taxing its mother and terminal logic boards, I can pull the plug and my H-89 will instantly settle down to a few hours of R&R (reset and relaxation) before the next day's busy schedule - something my daughter would never do!

The intelligence of the computer is fortified with one trait which humans have not been able to conquer: the ability to build on knowledge it has acquired in the past.

What's more, the computer has drawn its intelligence from many persons of many walks of life, each specializing in a specific field of expertise, all contributing to an ever increasing storehouse of knowledge and operations that tend to utilize efficiency to the fullest. As one works with a computer, it becomes increasingly apparent that each of these select persons are among the most knowledgeable in their field in the WORLD.

The human species (carbon units, if you will) feel we can start from scratch with each new arrival and learn all there is to know from day one until day "X", whenever that may be.

As the superior being, we have relied only on our one or two parental instructors, along with haphazardly selected others who may or may not know what they are talking about, to gain intelligence - each randomly teaching, rightly or wrongly, something new to be retained and used through life.

Unfortunately, at the end of that life all knowledge gained is lost forever to the individual's descendants or the rest of the world, for nowhere, in all of man's infinite wisdom, has anyone ever devised a system for transferring the knowledge of a lifetime from one being to another. Computer knowledge, on the other hand, is instantly transferable from unit to unit, intact, unaltered, and quickly accessible to both present and future generations.

If you are one of the fortunate individuals who has both a baby and a computer in the house, you might want to perform some ADDITIONAL comparisons of the superior human and the so-called 'stupid' computer:

1. Does your newborn have the ability to accurately test its own random access memory? (The computer does. One for the computer.)

2. If your baby WERE able to test its own memory, could it tell you exactly where a problem lies should there be a problem? (The computer can. Two for the computer.)

3. Can your baby tell you if it will be able to calculate and/or compute problems, complex or otherwise, now or in later years? (Needless to say, three for the computer.)

4. Within minutes of arriving (assuming neither computer nor baby are ordered in kit form), can your baby print symbols, numbers, and the entire alphabet? (Make it four.)

5. Referring to test four: in the language of ANY nationality? (Five.)

Bear in mind that tests one through five are performed BEFORE an operating system or language is installed in the computer.

6. If your baby passed test four, can it do all of test four forward OR backwards? (Six ...)

7. Ask your baby to test its internal memory storage speed, as you would ask your computer to test its disk access speed. (Seven.)

8. Can your baby map its own memory for maximum efficiency? (Another one for the computer.)

Since, if my meager memory serves me correctly, only about two percent of the human brain is currently being utilized, I would have to confess the computer is already stomping holes in the "superior human being" syndrome. What's more, even with one-hundred percent of its memory in use, the computer additionally rearranges its memory for the most efficient, effective use.

9. How about outputs? Does your baby know how and where? (Ok, ok! But it's still eight for the computer.)

10. Is your baby able to, instantly at birth, work with the most efficient system to do the job at hand? (Make it nine for the computer.)

'At birth' capabilities comparisons between the baby and the computer could go on and on - but I think you get the drift.

In the thousands of hours following the acquisition of knowledge by your "stupid computer", YOU (oh, human of super intelligence) sit at the keyboard typing and trying, digging through manuals, reading 'REMark', asking questions about how this or that is done and what the correct syntax is for "that particular instruction". The computer, on the other hand, waits patiently (and probably somewhat bored) for you to figure out its complexities.

I have never seen a school for 'stupid' computers but there are, however, thousands of schools for the far superior humans who spend much hard earned wampum and hundreds of hours trying to find out simply how to TALK to their computer.

The computer, by the way, is instantly able to give you a reply in YOUR language as soon as YOU find out how to talk to IT. That's quite an accomplishment for such a dumb machine.

Through the years, thousands of highly intelligent people have contributed to the knowledge your computer has the moment it is given its disk or tape full of intelligence:

... your computer is instantly able to perform complex calculations far beyond the capabilities of most of its users,

... able to write documents and then check YOUR (not its) spelling errors,

... offer shortcuts in job performance (which the user must find hidden in the maze of printed pages from which hundreds of companies are deservedly making big bucks),

... sort through thousands of bits of information and then send them back to you just the way you want it, in virtually seconds (at least, minutes),

and the list, as with all computer lists of this type, goes on and on. What has been mentioned here is only a drop in the proverbial bucket.

I would venture to bet that not many computer users today, if any, can honestly say they understand and are able to use ALL the features of even ONE operating system or language. And even if you could, the moment the next package arrives, YOU begin learning again. The computer HAS its knowledge, all of it, the moment you insert the disk or start the tape. You, my friend, are back to square one: trying to figure out how to talk to your computer in a manner it can understand. It already knows how to reply. It knew it all along. It also knows how to do what you want it to do, and MORE. YOU are the one with too little knowledge to tell it what you want it to do.

Stupid? Not on your life! Even the trashiest computer on the market (whichever one that may be) is infinitely smarter than you are as soon as you insert the disk or fire up the tape. Your Heath/Zenith computer (being one of the smartest, if not THE smartest, on the market) will have you buffaloed for years.

True, computers were invented by humans. But today humans are relying on computers to perpetuate the state of the art by designing bigger, better, and more complex computers that are far beyond the limitations of the individual human mind. Note that I said "the individual" human mind: thousands of persons working together MAY be able to compete with the speed and accuracy of the computer. But how long would it take? How much expense would be involved in making the same thing the computer makes with ease and speed unequalled by the 'superior' human species?

True, also, that the human is the one who keeps the computer alive and running. May I remind you of hospital computers that are helping keep HUMANS alive and running? I guess one good turn deserves another.

As for the idiot who first said computers are stupid: well, I guess that's the first clue to the intelligence of the human as opposed to the computer. My H-89 has never called ME names, although it has on many occasions made me FEEL stupid, and probably with plenty of justification.

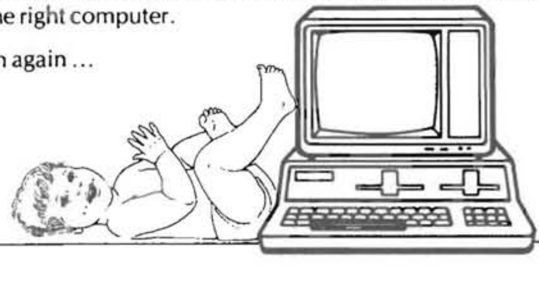
The next time you hear the phrase "a computer is really a stupid machine ...", suggest that the person who said it take the "baby test".

In all fairness to the humans though, I must sheepishly admit that even the Heath/Zenith variety of 'desktop' computers cannot clean house or open doors or cut grass. They do not walk and talk and laugh and watch TV. They do not go fishing or play tennis or go on dates. They do not know how to casually crack jokes - or even how to love. But on the other hand, HERO-1 has arrived in all his glory and who knows what wonders THAT will lead to ...!

Odds bodkins!!! Is it possible the ignorant computer really DOES possess capabilities beyond that of the "superior" human being? Can it be possible we have been overrating US and underrating THEM!? Is it possible the "highly intelligent" human newborn is on its way OUT!!??? Nawwww!!! I certainly can't believe that! Well, hardly...

With all that I have said FOR the computer, I will give this much to the human, though: "finding" the baby takes less running, less figuring, less effort, less frustration, and is probably more fun than trying to find the right computer.

And then again ...



Improved Error Recovery For CP/M

Pat Swayne
Software Engineer

Of the three operating systems used on Heath/Zenith computers, HDOS, Z-DOS, and CP/M, the one with the worst disk error recovery method is CP/M. In HDOS, which has the best error recovery, any disk error causes an error condition code to be returned to the running program, so that the programmer can process any error as he sees fit. In CP/M, only directory and space error (disk or directory out of space) conditions are returned to the program. All others are captured by the system, and that horrible "Bdos Err On ..." message is displayed, and the system waits for you to type a character. If the error is a Bad Sector, you are permitted to abort and warm boot with control-C, or to re-try the operation by typing any other character. For all other errors, any character typed causes a warm boot, and absolutely no recovery is permitted. There are three error types in this no recovery group: improper drive select, an attempt to write over a read only (write protected) file, and an attempt to write on a read only disk.

This last kind of error is the one that probably gives people the most trouble, because if you change a disk in a drive without resetting the disk system or warm booting, the new disk is automatically marked as Read Only. One solution to the problem of non-recoverable errors, which was discussed in passed issues of REMark, is to create a null directory entry (SAVE 0 GO.COM), and execute it to re-start your program whenever you get "booted out" by the system.

A better solution to the problem is to modify CP/M so that you have the option of warm booting, re-starting the program, or returning control to the running program whenever a Select or Read Only error occurs. In this article, I will present a patch to do just that. The only drawback is that the running program is not equipped to handle a return from such an error, and so the third choice will result in unpredictable behavior. It was included for the benefit of the "hacker" who likes to experiment.

To install the patch, you first need to create an image of the CP/M system on your disk. You can do it with SYSGEN as follows:

```
A>SYSGEN
SYSGEN VER 2.0.03
SOURCE DRIVE NAME (OR RETURN TO SKIP):A
SOURCE ON A, THEN TYPE RETURN                (type RETURN here)
FUNCTION COMPLETE
COPY BIOS.SYS (Y/N):N
DESTINATION DRIVE NAME (OR RETURN TO REBOOT):  (type RETURN)

A>SAVE 33 CPM.COM
```

This example is for Heath/Zenith CP/M version 2.2.03, and will vary with other versions. In particular, non-Heath/Zenith versions may require that a different amount be SAVEd. If you are not sure how much to save, run your version of MOVCPM before doing this procedure, and it will tell you.

Next, you must load the saved CP/M image into memory with DDT by entering

```
A>DDT CPM.COM
NEXT  PC
2700 0100
-
```

Now, locate the BDOS error portion of CP/M with DDT's D command by entering

```
-D1000
```

Examine the ASCII part of the resulting display (the part to the right) and look for the words "Bdos Err". If you do not see them, enter D again, this time without an address, hit RETURN, and keep looking. Continue this process until you see something that looks like this:

```
1230 D5 21 DC D5 CD E5 D5 C3 1F 12 42 64 6F 73 20 45 .!........Bdos E
1240 72 72 20 4F 6E 20 20 3A 20 24 42 61 64 20 53 65 rr On  : $Bad Se
```

Note the address of the letter B in "Bdos" (123A in this example, but it could be different in your system), and subtract 3 from it with DDT's H command as follows:

```
-Haaaa,3
xxxx bbbb
```

Notice that I have shown the address of the letter B as aaaa, and the result of the subtraction as bbbb. I will follow this procedure from here on, and use xxxx to indicate an insignificant address that does not effect the patch. Now, use the L command to disassemble from address bbbb to aaaa. It should look like this:

```
-Lbbbb,aaaa
  bbbb  JMP  0000
  aaaa  MOV  B,D
  xxxx
```

Using the A command to enter assembly code, replace the jump to zero at bbbb with the following code:

```
-Abbbb
bbbb  SUI 3
xxxx  JZ 100
xxxx  INR A
xxxx  RZ
xxxx  RST 0
xxxx  .              (type a period, then RETURN)
```

Add 37 (hex) to the address aaaa.

```
-Haaaa,37
cccc xxxx
```

Examine the code at the new address cccc.

```
-Lcccc
  cccc  LXI  B,dddd
  xxxx  CALL xxxx
  - - -              (rest of listing not shown)
```

Add 5 to the address dddd, and replace it with the result.

```
-Hdddd,5
eeee xxxx
-Acccc
cccc  LXI B,eeee
xxxx  .
```

Now, type control-C to exit DDT, and SAVE the patched CP/M image as follows:

```
-^C
A>SAVE 38 CPM.COM
```

Since the CP/M image contains an image of the SYSGEN program, it can be run as a file to re-install the new CP/M:

```
A>CPM
SYSGEN VER. 2.0.03
SOURCE DRIVE NAME (OR RETURN TO SKIP):                (hit RETURN)
DESTINATION DRIVE NAME (OR RETURN TO REBOOT):A
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT):         (hit RETURN)
A>
```

Your system will now be running with the patched BDOS, unless you are using CP/M-85. In that case, you will have to re-boot to load the patched system into memory. With the patch installed, the following changes will be in effect. The "Bdos Err On ..." message is shortened to "Err On ..." to make room for the patch. When you get a Select or Read Only error, you can type control-C to restart your program (jump to 100H), control-B to simply return from the error handler (with unpredictable results), or any other character to warm boot CP/M.

When you are running MBASIC, the best choice of the above is to restart the program with control-C. Then you must enter RESET before attempting the file operation again. Try experimenting with control-B with various errors while running various programs. In some cases, you can recover from a Select error with control-B.

USE ALL YOUR SPECIAL FUNCTION KEYS WITH WORDSTAR™

WSKEY™: Now you can take the mystery out of WordStar™ with SKILL DATA's WordStar enhancement, which implements all TWENTY-ONE of the H/Z89-19 function/pad keys or all TWENTY-THREE Z-100 labeled key commands.

Function key commands are labeled by a twenty-fifth line banner, which can be toggled on and off by you during your session.

$29.95 — Specify H/Z89-19 or Z-100

dBASE II™ ZIP™ SuperCalc™

DBKEY™ for dBASE II,
ZPKEY™ for ZIP SCREEN, SCKEY™ for SuperCalc

Just type your favorite SKILL DATA KEY command. All function key commands are labeled by a twenty-fifth line banner, which can be toggled on and off by you during your session. Pad keys also function and send multiple key inputs with a single stroke. All previous command key sequences are still available for the old and painful ways.

$29.95 - keys to SuperCalc, SCKEY. For H/Z 89-19 only.

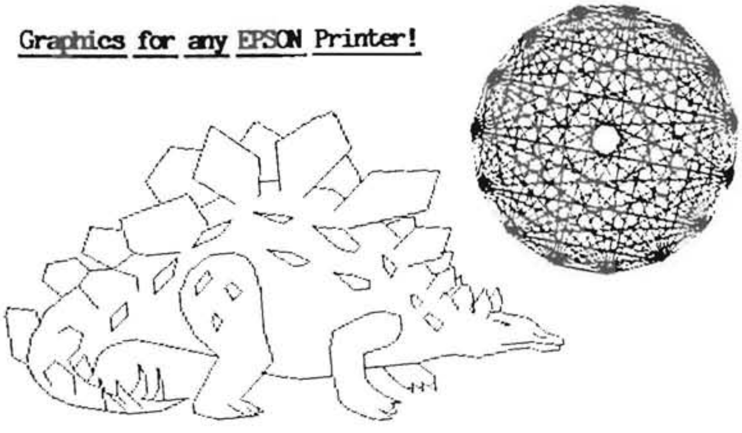
$29.95 - keys to dBASE II: DBKEY and ZPKEY.
As reviewed in Sept. 83 dNEWS
Specify H/Z89-19 or Z-100 (CP/M-85 or Z-DOS).

H/Z89-19 diskettes are 5.25" 10HS, requires CP/M 2.2.03
Z-100 diskettes are 5.25" SS, specify CP/M-85 or **Z-DOS.**

All orders shipped by first class mail. Include $3.00 per order for shipping and handling, $5.00 for overseas orders. Payment by check or money order. Allow 2-4 weeks for delivery. Visa/Master Card. 206/352-0669 (evenings).

SKILL DATA
P.O. Box 1943, Olympia, Washington 98507

Graphics for any EPSON Printer!



*** MXGRAPH** will turn your EPSON printer into a graphics plotter for personal or business needs! Make your EPSON plot an unlimited number of artistic sketches, geometric designs, and business graphs, which would not be possible without MXGRAPH.

MXGRAPH runs under **CP/M-80** or **HDOS** operating systems ONLY on the H/Z-89 and H/Z-90 computer systems. Requires an EPSON printer equipped with bit plot capabilities. Specify HDOS or CP/M, hard or soft sectored 5.25" disk. **MXGRAPH is ONLY ~~$79.95~~ $49.97!**

*** DON'T DELAY - ORDER NOW! ***

*** EFC - The Electronic Filing Cabinet** is a screen-oriented hierarchical database. Use EFC as a **PERSONAL FILING SYSTEM** for all types of information!! Runs under Z-DOS ONLY on the H/Z-100. EFC is ONLY $39.97!

*** MXPRINT** generates almost **LETTER-QUALITY** printing on an EPSON printer equipped with bit plot capabilities! Prints 10 char per inch by 6 lines per inch (9x24 matrix). MXPRINT is ONLY $19.97!

*** NOVA The WordStar™ Enhancer!** Adds ALL keypad AND function keys to WordStar. Greatly speeds text editing. 25th line prompts you. NOVA REQUIRES HEATH/ZENITH DISTRIBUTED WORDSTAR. NOVA is ONLY ~~$39.95~~ **$24.97**

MXPRINT and **NOVA REQUIRE** Heath/Zenith distributed **CP/M-80** (Version 2.2.03 or later) for operation on an **H-8, H/Z-89,** or **H/Z-90;** or **CP/M-85** for operation on an **H/Z-100.** Specify hard or soft sectored 5.25" disk.

MICRO INNOVATIONS
2455 Sylvania Ave., Toledo, Ohio 43613
VISA and MASTERCARD Orders Welcome!
CALL NOW: (419) 471-1245

So, Your Computer Can't Add

David G. Pelowitz
11614 So. 35th Street
Omaha, NE 68123

Have you ever had one of those days where nothing goes right? Was it so bad that even your trusty computer seemed to have forgotten how to add? Or worse yet, that program you just spent two months writing and debugging refused to recognize the contents of one of its key variables? Take heart, all is not lost. There are some simple explanations why a number apparently doesn't equal itself. Once the cause of this kind of problem is understood, writing your programs to avoid it is simple. The way a computer internally represents numeric quantities is the crux of the problem and all computers share the malady to one extent or another. I recently came across a major top of the line mini- computer which had a dickens of a time handling the value zero! The company fixed the bug, but it goes to show you how wide spread this type of problem is.

The following few paragraphs are for those readers that don't have a good understanding of binary number systems. If you are intimately familiar with common internal representations of signed and unsigned integers, skip down to the section on floating point numbers.

If we are going to investigate how computers internally represent numeric values, I'll have to lay some foundation. First, not all computers have the same size of memory word. Currently, most personal computers are eight bit machines. The bit is to the byte as a letter is to a word. It is the smallest unit of information a computer can store and can only represent two states. The two states are on and off, and are usually referred to as "1" and "0" respectively. Calling a computer an eight bit machine means the Central Processing Unit (CPU) handles information eight bits at a time and the computer's memory is organized with eight bits at each address. The size of a memory location is referred to as the system's word size, whereas a byte is universally accepted as consisting of eight bits. A true 16 bit machine uses a 16 bit word both inside the CPU and in memory. Its CPU will fetch two bytes from memory at a time.

Some of the recently developed microprocessers are hybrids of 16 and 8 bit machines. In the CPU, they handle data 16 bits at a time, but externally they use an 8 bit wide memory. Other word lengths are also common. Machines have been designed to use 12, 18, 24, 32, 60, or 64 bit words. Some machines are even more radical and use a variable word length. For the most part, you need not worry about the size of the word because the concepts are still the same. Either way, read the documentation on your machine. You will need to know the word size of your computer to apply this discussion. Further, some programs use what is called multiple precision. I will explain later what that means and how it affects the representation of numeric information, but for now you should simply be aware of its use.

Binary Numbers

There is a difference between how a computer stores the character "2" and how it stores the value two. Each character is represented by a predefined set of bits. Each of these bits is either on or off. The on condition is usually represented by a "1" and the off condition by a "0". One of the most common character representation standards is called the American Standard Code for Information Interchange (ASCII). It is almost universally used and employs seven bits to define its entire cast of characters. For example, the character "1" is represented by the bit pattern "0110001". A character is usually stored in the low order seven bits and the high order eighth bit is set off. Typically the right hand bit is considered the low order bit and the left the high order bit. To refer to the bits in a byte as "low order bit", "next to low order bit", "second from the low order bit", and so on is awkward at best. Let's number them from right to left starting with zero. Consequently, the key "76543210" can be used to refer to each of the bit positions in a byte. I suspect this brings two questions to mind. First, "Why right to left?" and secondly, "Why start with zero?" Right to left is used because that's what we use in our decimal system. Its roots are in the Arabic language which is read from right to left. The number 426 means six ones, two tens, and four hundreds. The ones are the lowest order and in this case the hundreds are the high order. Why start with zero? Read on!

```
THE DECIMAL SYSTEM

 5373059
 |||||||
 |||||||---  = 9                                 =       9
 ||||||----  = 5 x 10                            =      50
 |||||-----  = 0 x 10 x 10                       =     000
 ||||------  = 8 x 10 x 10 x 10                  =    8000
 |||-------  = 7 x 10 x 10 x 10 x 10             =   70000
 ||--------  = 3 x 10 x 10 x 10 x 10 x 10        =  300000
 |---------  = 5 x 10 x 10 x 10 x 10 x 10 x 10   = 5000000
                                                   -------
                                           total = 5373059
```

Figure 1.

The contents of a byte can represent a quantity as well as a character. Because a bit can only be in one of two conditions, it is called a binary system. If we remember back to our decimal system, each successively higher order position stood for a quantity 10 times the previous position (figure 1). The low order position was the quantity of ones, the next up was the quantity of 10s, the third was the quantity of 100s, and so on. We can do the same thing with our binary system. Because this is a binary system instead of decimal, we will increase by two times the previous position's value instead of 10 times. Consequently, the first bit tells us how many ones, the second bit tells us how many twos, the third how many fours, and so on. We can extend this pattern indefinitely. Figure 2 shows the values for each of the first eight bits in this system. Now let's see what significance the "76543210"

```
 THE BINARY SYSTEM

11011011 (binary)
||||||||
||||||||---  = 1                                 =  1 x   1  =  1
|||||||----  = 1 x 2                             =  1 x   2  =  2
||||||-----  = 0 x 2 x 2                         =  0 x   4  =  0
|||||------  = 1 x 2 x 2 x 2                     =  1 x   8  =  8
||||-------  = 1 x 2 x 2 x 2 x 2                 =  1 x  16  = 16
|||--------  = 0 x 2 x 2 x 2 x 2 x 2             =  0 x  32  =  0
||---------  = 1 x 2 x 2 x 2 x 2 x 2 x 2         =  1 x  64  = 64
|----------  = 1 x 2 x 2 x 2 x 2 x 2 x 2 x 2  =  1 x 128  =128
                                                            ------
                                                         total =219
```

Figure 2.

key has in this system. Each bit position represents a value. The value is two raised to the power of value in the position in the key. We now have a method of representing quantities and characters.

Addition & Subtraction

Using our binary system we can show that a single byte can contain representations for quantities up to 255. Let's examine the rules for simple binary addition and subtraction. Examine figures 3 and 4. Each of the additions are very straight forward with the exception of "1 + 1" which generates a carry to the next higher order column.

```
          BINARY ADDITION

     Rules:    0        1        0        1    Example:  01101010
             + 0      + 0      + 1      + 1             + 01101100
             ---      ---      ---      ---             ---------
              00       01       01       10              11010110
```

Figure 3.

```
          BINARY SUBTRACTION

     Rules:    0        1        1        0    Example:  10010110
             - 0      - 0      - 1      - 1             - 01101100
             ---      ---      ---      ---             ---------
              00       01        0       *1              00101010
                                        * = borrow
```

Figure 4.

The subtractions are equally simple with the exception of a borrow from the higher order neighbor. Both the carry out of the addition and the borrow from the subtraction are done in exactly the same way as in our familiar decimal system. There are three points to remember when doing binary math. First, there are only two binary values, 0 and 1. Secondly, a carry is the same as an addition to the remaining higher order bits. Thirdly, a borrow is the same as a subtraction from the remaining higher order bits. But what happens if we are subtracting a larger number from a smaller? The result is supposed to be negative!

Signed Binary Numbers

With the exception of zero, values can be either positive or negative. But the binary system we have been examining only contains positive numbers. If we take a hint from the fact that there are two states the sign of a value may assume, using one of the bits of a byte as the sign bit seems logical. The high order bit is commonly used for the sign flag. If it is on, we have a negative number and conversely, if it is off we have a positive number. This limits the maximum value a byte may store to +127 and the minimum to -127. If you aren't sure how these numbers are formed, add up the bits to verify them.

There is a problem with storing values in this manner. Look at figure 5. If we subtract eight from nine, we get a good result. If we add -8 to nine, we get a -17! Addition and subtraction must work correctly in all cases for our system to be useful. Let's try something interesting. If we complement the bits of a negative number and store it that way, we are much closer to having our sample problem work correctly. Complementation is simply turning on the off bits and off the on bits. Complementing 10000001 changes it to 01111110. This is called one's complement. We can see in figure 6 this comes much closer to giving us the correct answer.

```
SIMPLE SIGNED MAGNITUDE ADDITION

                 00001001 =  9         00001001 =  9
               - 00001000 =  8       + 10001000 = -8
                 --------              --------
                 00000001 =  1 ok      10010001 = -17 bad (should be 1)
```

Figure 5.

```
 ONE'S COMPLEMENT ADDITION

                    7 = 00000111        00001010 = 10
      complemented 7 = 11111000      + 11111000 = -7
                                        --------
                                        00000010 =  2 bad (but close)
```

Figure 6.

It looks like all we have to do is add one to the answer in figure 6 to get the correct value. In fact, that is exactly what we need to do, except instead of adding the one to the result of the addition, we will add it to the complemented seven. This form of representation is called the two's complement and is used extensively to represent negative numbers. The -7 in twos complement notation is 11111001. Figure 7 contains some examples of two's complement numbers.

```
 TWO'S COMPLEMENT

  -128 = 10000000
  -127 = 10000001
    -4 = 11111100
    -3 = 11111101                 8       00001000
    -2 = 11111110              +(-7)      11111001
    -1 = 11111111              -----      --------
     0 = 00000000                 1   =   00000001 (ok)
     1 = 00000001
     2 = 00000010
     3 = 00000011
     4 = 00000100
   127 = 01111111
   128 = invalid
```

Figure 7.

Try adding one to 127. What happened? If you did it right, you ended up with a -128! One obvious answer to this problem is to add more significant bits to the location that stores the value.

Multi-Precision Signed Integer

Most of us find the single precision two's complement limitation of -128 to +127 too limiting for most applications. The common remedy for this is to use more than one byte to store a value. If multiple bytes are used, only the high order bit of the high order byte is used as the sign bit. All the lower order bits behave exactly like the low order bits of the single precision value. Figure 8 contains the limitations of two's complement notation for various length multi-precision values. If you check your Microsoft Basic-80 documentation, you will find integer constants are limited to the same as a two byte two's complement multi-precision values' limitations in figure 8. But what about those really big numbers?

TWO'S COMPLEMENT
MULTI-PRECISION LIMITATIONS

total bytes used	minimum value	maximum value
1	-128	+127
2	-32768	+32767
3	-8388608	+8388607
4	-2147483648	+2147483647

Figure 8.

Floating Point Notation

As computers grew in speed and complexity, more and more applications relied on their unique capabilities. Some of these applications didn't need any more accuracy than a multi-byte two's complement number. But some dealt with very large numbers. To satisfy these requirements a new internal representation was developed. Mimicking scientific notation, this new representation consists of two parts, the mantissa and the exponent. A number in scientific notation is expressed with one non-zero digit to the left of the decimal point and all of the remaining significant digits to its right. An exponent is added to the right of the number to indicate how many times the number should be multiplied or divided by ten. Figure 9 contains some examples of values expressed in scientific notation.

SCIENTIFIC NOTATION

Value	Mantissa	Exponent	Scientific Notation
826	8.26	2	8.26×10^{2}
1024	1.024	3	1.024×10^{3}
52,907,000,000	5.2907	10	5.2907×10^{10}
1,000,000,000,000,000	1.0	15	1.0×10^{15}
0.00000000000000567	5.67	-15	5.67×10^{-15}

Figure 9.

Floating point notation, is very similar to scientific notation in that there are two parts, also called the exponent and the mantissa. The major difference is that floating point notation insists on having the first non-zero significant digit to the immediate right of the decimal point. The function to insure this occurrs is called normalization. So far we haven't discussed the existence of a decimal point. The decimal point is not really there. It is called an implied decimal point. By definition, all bits of the mantissa are to the right of the implied decimal point. Just to make things more difficult, each bit has a different value than it had as a single or multiple precision integer. Remember how the value in each successive bit position increased by a factor of two? In this case each bit on the right of the implied decimal point decreases by a factor of two. The first bit on the right is worth one-half, the next one-fourth, then one-eighth, and so on. Notice how each denominator is two raised to a power? Figure 10 shows the values of the first 16 bits.

```
      BIT VALUES
RIGHT OF THE DECIMAL POINT

     |------------------------>1/2     = .5
     |                         1/4     = .25
     |                         1/8     = .125
     |                         1/16    = .0625
     |                         1/32    = .03125
     |                         1/64    = .015625
     v                         1/128   = .0078125
    .1111111111111111          1/256   = .00390625
      (binary)      ^          1/512   = .001953125
                    |          1/1024  = .0009765625
                    |          1/2048  = .00048828125
                    |          1/4096  = .000244140625
                    |          1/8192  = .0001220703125
                    |          1/16384 = .00006103515625
                    |          1/32768 = .000030517578125
                    |---------->1/64536 = .0000152587890125

      Example:

      .10010001(binary) = 1/2 + 1/16 + 1/256
                        = .5 + .0625 + .00390625
                        = .56640625
```

Figure 10.

The exponent and mantissa are combined in a number of ways. One of the common methods is to define a floating point variable as using four bytes. The first byte is the exponent in single precision signed integer format. This type of format limits the size of the exponent between -128 and +127 inclusively. The remaining three bytes are multiprecision signed format with an implied decimal point before the most significant bit. In scientific notation, the exponent indicates the number of times to multiply or divide the mantissa by ten; whereas in floating point notation we are working in binary, not decimal, so it is the number of times to multiply or divide by two. If the sign is positive, multiplication is indicated, otherwise division is the desired operation. Some examples of floating point numbers can be seen in figure 11.

```
 FLOATING POINT VARIABLES


   Format:
               seeeeeee smmmmmmm mmmmmmmm mmmmmmmm

                     seeeeeee = signed exponent
                 smmmmmmm mmm... = signed mantissa


 Examples:

 0.5 = 00000000 01000000 00000000 00000000 = 1/2
 1.0 = 00000001 01000000 00000000 00000000 = 1/2 mult. by 2 once
 1.5 = 00000001 01100000 00000000 00000000 = (1/2 + 1/4) mult. by 2 once
 2.0 = 00000010 01000000 00000000 00000000 = 1/2 mult. by 2 twice
0.25 = 11111111 01000000 00000000 00000000 = 1/2 divided by 2 once
```

Figure 11.

Floating Point Error

If the mantissa uses 24 bits as in the examples of figure 11, then the smallest increment between expressible values is 1/8388608. Therefore, the fractional portion of all floating point numbers will be a multiple of this value! What happens if the number we are trying to express is not a multiple of 1/8388608? Usually, it is stored as close as possible with the remaining error simply dropped. Let's look at some examples of this type of error. If we divide one by three, we have one-third. When we express this as a decimal value it is .33333... and continues indefinitely. The decimal system cannot express it exactly. Usually we select a de-

```
A FLOATING POINT ERROR EXAMPLE

                         (1 / 3) * 3 = ?

                        seeeeeee smmmmmmm mmmmmmmm mmmmmmmm
   1/3 = .333333... = 11111111 01010101 01010101 01010101

       = (1/2 + 1/8 + 1/32 + 1/128 + 1/512 + 1/2048 + 1/8196 +
          1/32784 + 1/131136 + 1/524544 + 1/2098176 + 1/8392704) / 2

         .5
       + .125
       + .03125
       + .0078125
       + .001953125
       + .00048828125
       + .0001220703125
       + .000030517578125
       + .00000762939453125
       + .0000019073486328125
       + .000000476837158203125
       + .00000011920928955078125
       ---------------------------
         .66666662693223681640625 / 2  = .333333313466118408203125

         .333333313466118408203125 * 3 = .999999940398355224609375
```

Figure 12.

sired number of significant digits and then either truncate or round off the remaining digits. In this case let's assume five significant digits are all that is required. Therefore, one-third will be expressed as .33333. Here in lies our problem. Multiply .33333 by three and we have .99999 and not the 1.0 which is desired. We have now proven that three times one third is not one! Well, not actually. All we really have done is demonstrate that three times our representation of one-third is not equal to one.

An identical problem exists when computers represent numbers in floating point format. We can see in figure 12 how one-third is stored. Follow through the math in this example. Obviously, one-third times three is one, but we can see the results are not computed to be one! You were absolutely right, your computer not only can't add, but it can't multiply or divide either! I chose this particular example because it is easy to demonstrate. There are many numbers which cannot be exactly represented this way. They can be broken into three classes. The first group contains those numbers with infinitely repeating digits. One-third is a member of this class. In decimal it repeats "3" infinitely. When expressed in binary it repeats "01" forever. A second class are the numbers which simply need more bits to represent them. If 1/8388608 is the smallest increment we can represent in a typical floating point system and a number requires half that value more to express it exactly, then we would need one more bit past the least significant bit. The number of significant digits can be increased by increasing the size of the mantissa. A very common double precision floating point length is 64 bits. Although increasing the size of the mantissa increases the accuracy, and can therefore eliminate one of the classes of non- representable numbers, there are many numbers which cannot be exactly represented with anything short of an infinitely long mantissa. The third class contains the numbers which are non- repeating and never end. The ratio of the circumference of a circle to its diameter, pi, is an example of one of these numbers. They too cannot be exactly represented in a typical floating point system.

There is another type of floating point related error which should be considered. Most single precision floating point variables contain about seven significant digits. What do you think would happen if the least significant digit of a variable was units of thousands and we attempted to add a one to it? The value one would have to be added to bits below the least significant bit. As you can guess, the addition will not occur because the one is comparatively insignificant. Part of the program in the listing demonstrates this foible. All is not lost. Even though your computer can't add, it can learn.

```
10 CLEAR 100
20 DEFSNG D
30 DEFINT A
40 PRINT CHR$(27);"E"
50 PRINT
60 PRINT @
        "There are two functions to this program.  ";@
        "The first demonstrates the folly of"
70 PRINT @
        "adding a relative small number to a ";@
        "large one."
80 PRINT
90 PRINT "In this case we will attempt to ";@
        "add 1 to 1000000, 100 times."
100 GOSUB 550
110 FOR I = 1 TO 50
120 D9 = 1E+06
130 PRINT USING"########\      \";D9;" + 1 = ";
140 D9  = D9 +1
150 PRINT USING"########";D9
160 NEXT
170 PRINT "Notice the addition never occurred."
175 PRINT "If it had, the answer would have been 1000100."
180 GOSUB 550
190 PRINT CHR$(27);"E"
200 PRINT "Floating Point Breakout"
210 PRINT
220 PRINT @
        "This program peeks into memory to get ";@
        "the bytes of a floating point variable."
230 PRINT @
        "It then prints out the floating point ";@
        "representation in both hex and binary."
240 PRINT @
        "Page E-6 of the Microsoft BASIC Software ";@
        "Reference Manual describes the format."
245 PRINT @
        "If you are as inquisitive as I am, try -1.7014117E38"
250 PRINT
260 PRINT "Enter decimal value to be converted..."
270 A1=0:A2=0:A3=0:A4=0
280 D1=0.0#
290 PRINT
300 INPUT "Decimal ";D1
310 Q = VARPTR(D1)
320 A1=PEEK(Q)
330 A2=PEEK(Q+1)
340 A3=PEEK(Q+2)
350 A4=PEEK(Q+3)
360 A1$=HEX$(A1):IF LEN(A1$)<2 THEN A1$="0"+A1$
370 A2$=HEX$(A2):IF LEN(A2$)<2 THEN A2$="0"+A2$
380 A3$=HEX$(A3):IF LEN(A3$)<2 THEN A3$="0"+A3$
390 A4$=HEX$(A4):IF LEN(A4$)<2 THEN A4$="0"+A4$
400 PRINT "    Hex = "A4$" "A3$" "A2$" "A1$
410 PRINT " Binary = ";
420 AA=A4:GOSUB 490
430 AA=A3:GOSUB 490
440 AA=A2:GOSUB 490
450 AA=A1:GOSUB 490
460 PRINT
470 GOTO 270
480 REM Strip out binary
490 FOR I = 7 TO 0 STEP -1
500 IF (AA AND 2^I) = 0 THEN PRINT "0"; ELSE PRINT "1";
510 NEXT
520 PRINT " ";
530 RETURN
540 REM Wait for keystroke
550 PRINT"Hit any key to continue..."
560 A$=INPUT$(1)
570 RETURN
```

Listing 1.

Conclusion

Dealing with the intrinsic error of floating point notation has been the topic of numerous books. There are as many techniques to handle this type of error as there are manufacturers creating microcomputers. A common technique is to store values in a format called Binary Coded Decimal, BCD. This technique uses four bits for each number. Groups of BCD numbers are used to represent larger values. For example, 1024 could be represented as "0001 0000 0010 0100". A modified floating point system can easily be created using BCD. A system of this nature allows for large numbers to be represented but still suffers from the infamous repeating mantissa. Some systems round off the fractional portion while some simply truncate it. Microsoft Basic-80 rounds off both single and double precision floating point. Use the program in the listing to prove it.

There are a number of simple things you can do to minimize this type of problem. As a rule of thumb, use integers whenever possible. You can force a variable to be an integer in Microsoft BASIC by using the DEFINT statement. Even money can be manipulated as an integer value. Keep in mind, money can be counted by cents as well as dollars and fractions of dollars. Another good rule is to always control the type of storage a variable uses. Although not a problem in strongly typed languages like Pascal, this is very important in languages like BASIC. Probably the most effective preventative medicine you can take is to know the language you are using. Run programs similar to the listing in each and study the results. If all else fails, you can even read the documentation.

We have just scratched the surface of the numeric representation problem. Consider a series of computations all involving floating point numbers. The error from each of them can accumulate and completely wipe out any meaningful results. There are techniques to handle them and being able to apply these techniques is an absolute must for any programmer.

H/Z-100 COLOR GRAPHICS SOFTWARE

Want to explore **Z-BASIC** color graphics the easy way? Try these packages from **MICROSERVICES.** All you need is a color configured computer with 128K memory, a color monitor, and **Z-BASIC.**

ZANIMATE will help you rapidly draw and paint each frame in an animation sequence. Select frame sizes, vary location and time between each frame. Draw the background too, with the graphics editor **$64.95**

ZPALETTE, the original H/Z-100 palette program, contains a graphics editor for point-plotting images which can be painted in 92 hues! File your images for access by other **Z-BASIC** programs, and transfer imagery from one file to another.
NEW PRICE **$59.75**

ZPATTERN helps you determine the quality of your color monitor and if it needs maintenance.
NEW PRICE **$24.95**

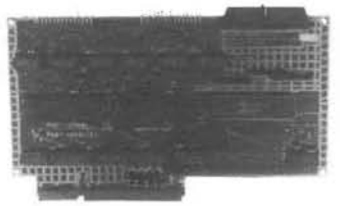
Include 3% handling/postage ($2.50 minimum) — California add 6½% tax. Send check or money order to:

MICROSERVICES
P.O. BOX 7093
MENLO PARK, CA 94026
PHONE: (415) 851-3414

H/Z89 PERIPHERALS from SECURED COMPUTER SYSTEMS

PORT SERIAL
2/3 —— CARD —— I/O
PORT PARALLEL

" . . . not your typical vanilla-flavored serial and parallel interface . . . "



Features:

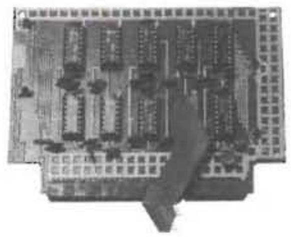
Chip independent design • Reduces computer data buss loading from 3 to 1 • Choice of Centronics or Epson parallel drivers for HDOS or CP/M • Complete documentation and installation instruction.

- 2 Serial Ports
- Supports: Ring Input, External Clock, Auto Dialer
- 3 Port Parallel with 2 Level Interrupt Control
- Fully compatible with all models of H/Z 88, 89, 90 using CP/M or HDOS.
- Fully tested, 90 day warranty, two serial cables and a parallel cable (internal to computer) and software driver.

PRICE $199.00
Shipping & Handling $10.00

16K RAM EXPANSION CARD

Expands your H/Z89 RAM Memory capacity to a FULL 64K!



Fully compatible with:
H/Z 89 • H/Z 88 • Magnolia Microsystems
CP/M and disk drive I/O interface cards

NOW INCLUDING SUPPORT MOUNTING BRACKET

Featuring:
Complete installation instruction • 90 day Warranty
Field reliability record now entering its 21st month

Now Only $65.00

HDOS is a registered trademark of Heath Company
CP/M is a registered trademark of Digital Research

Shipping & Handling $5.00

PRICES ARE LESS SHIPPING AND TAX IF RESIDENT OF CALIFORNIA
MAIL ORDER: 12011 ACLARE, CERRITOS, CA 90701 (213) 924-6741
TECHNICAL INFO/HELP:
8575 KNOTT AVE., SUITE D, BUENA PARK, CA 90620 (714) 952-3930
Terms and specifications subject to change without notice.

ZENITH data systems
SERVICE CENTER

Zenith Owners!

HUSKER SYSTEMS OF NEBRASKA, INC. PRESENTS
WINTER SOFTWARE SPECIALS THAT YOU CAN'T BEAT FOR PERSONAL COMPUTING ENJOYMENT!!

Video*Professor Computer-Based Training Systems help you learn quicker and with greater enjoyment than with conventional self-study courses. They are designed to let the computer interactively teach you rather than requiring hours of boring textbook study.

Currently there are 3 courses available: MBasic Programming I, Assembler Programming I, and How To Program, A Structured Programming Methodology. All three are available under HDOS and CP/M-80 for the H/Z89-90 (on hard sector diskettes) and under ZDOS and CP/M-85 for the Z100 (on soft sector diskettes). The regular retail price of each Video*Professor product is $29.95, but...

By referencing this special offer you can get one Video*Professor for $22.00, two for $42.00, and ALL THREE FOR $62.00!! This offer is good through March 31, 1984, and you should keep your eyes open for other additions to the Video*Professor product line.

ZMAG is the **Software Subscription Service** which provides you with a continuing stream of valuable software over a year's membership. Like a magazine, ZMAG provides you with interesting articles and personal computing tips, but with each issue you also get a diskette with **up to 15 PROGRAMS ON IT!** These programs include everything from games to business and personal aids to system utilities. ZMAG is available under CP/M-80 (on hard sector diskettes) and under ZDOS and CP/M-85 (on soft sector diskettes).

There are 8 issues (8 diskettes) to a ZMAG subscription year. The regular annual subscription rate is $240.00, which is $5.00 per issue under the "news stand" price. For this special period ending March 31, 1984, you can obtain a ZMAG subscription for **ONLY $177.00! THAT'S A 25% DISCOUNT.**

BETTER STILL, ORDER BOTH A ZMAG SUBSCRIPTION AND THE 3-PRODUCT VIDEO*PROFESSOR SERIES AND GET THE WHOLE PACKAGE FOR JUST $227.00! (It's a $329.70 value!)

Just use our handy order blank provided below or call us at **(402) 558-5702.** By the way, we have **over 40 other fine software products for Zenith computers,** all reasonably priced, and we can fulfill your media and hardware needs economically as well. Be sure to ask for our free price list, and please specify the operating system you desire for each product. Thank you.

Husker Systems of Nebraska, Inc. -- Fine Heath/Zenith Microprocessor Products

Method of Payment: ☐ CHECK ☐ MC/VISA #__________ EXPIRES______

QUANTITY	PRODUCT DESCRIPTION (INCL OPERATING SYSTEM)	UNIT PRICE	TOTAL PRICE*
		TOTAL	

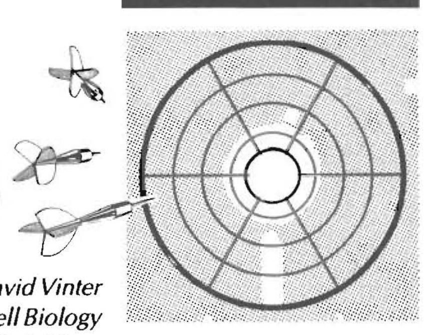
NAME__________ TEL__________

STREET__________ CITY______ ST____ ZIP______

Send Order To:
Husker Systems of Nebraska, Inc.
5208 Hamilton, Omaha, NE, 68132
Check Here For Free Catalog ☐

#1212

Disk Access By Tracks and Sectors



David Vinter
Dept of Anatomy & Cell Biology
University of Michigan
Ann Arbor, MI 48109

One of the advantages afforded by an operating system like CP/M is the ability to write and read files on disks without reference to the actual locations of the data on the disk itself. But sometimes you may wish to directly access data on the disk without using the file conventions - this is what the "un-erase" utility programs do when they recover a file, for instance, by changing a byte in the disk directory. This article will first discuss the logical organization of the Heath CP/M H17 disk, then demonstrate how to access the disk by track and sectors, and finally show you how to use a hidden 11 sectors on the disk that are not accessible by any other means. Direct track/sector disk I/O will also permit you to use HDOS-formatted disks as well, a technique that could be developed into a program that would permit direct transfer of files between the two operating systems.

Disk Organization

CP/M reads and writes disks in 128-byte units called records, regardless of the disk size or format. The BIOS determines how these records are mapped onto the physical structure of the disk itself; in this article we will only be concerned about the hard- sectored disk used by the H17, but the principles apply to the soft sectored 5-inch disks as well as the standard 8-inch disks. The disk consists of 40 concentric tracks (numbered 0-39), and each track in turn is made up of 20 records (numbered 1-20). Thus, the entire disk can contain 800 records (20x40). (Do not confuse the physical sectors, of which there are 10, with the logical records, which are 128-byte units. Each of the ten physical sectors making up a track has 2 logical records mapped onto it. Unfortunately, many people refer to the logical records as 'sectors', as well as the actual physical sectors.)

The first three tracks (0-2) are reserved for all of the operating system except the BIOS: a cold start loader, the CCP, BDOS, and the BIOS loader (the BIOS itself is stored like any other file). Track 3 contains the disk directory and the start of the file space; the remaining tracks (4-39) contain the rest of the file space. When a program running under CP/M reads or writes to a file, it finds the actual physical sectors containing each of the file records by looking at the directory for the named file. The directory is thus much like the table of contents of a book, listing the pages (sectors) on which the chapters (files) are written.

Before looking at the directory entries themselves, a little more detail about how CP/M writes disk files is necessary. Although the smallest unit which can be read or written is a record (128 bytes), to minimize the directory size, CP/M actually read/writes 8 records, called a group, at a time. By using calls to the BDOS, as almost all standard programs (like BASIC) do, file access is always by groups. Later on we shall see that the BIOS contains routines to access a single record, though, which is how the utility programs operate, and which you may use as well. For now, it is enough to know that all files are actually stored in 8-record units called groups.

Each file is listed in the directory in 32-byte entries, called extents, each one of which contains a list of up to 16 groups that the file data are written in. If a file is longer than this (16×8=128 records, about 16K), then a second extent must be entered in the directory for it, and so on, until all the file's records are listed. A file will have as many entries (extents) in the directory as needed, and they are given extent numbers (starting with 0) to keep them in order. A map of the 32-bytes for a single directory entry is shown below:

```
 byte      contents
   0     user number or deleted-file flag
           (on a one-user system, user number = 0)
           (if file is deleted, byte set to 0E5H)
  1-8    file name
  9-11   file type (extension); plus the
                highest bit can be set:
            byte 9, high bit=1: R/O file
            byte 10, high bit=1: SYS file
   12    extent number (first=0)
 13-14   blank
   15    extent size, in sectors
 16-31   group numbers
```

A specific example, showing the directory entry of the file PIP.COM, looks like this:

```
 byte:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
  HEX: 00 50 49 50 20 20 20 20 20 43 4F 4D 00 00 00 3A
ASCII:  0  P  I  P sp sp sp sp sp  C  O  M  0       58

 byte: 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  HEX: 07 08 09 0A 0B 0C 0D 0E 00 00 00 00 00 00 00 00
ASCII:  7  8  9 10 11 12 13 14
```

Reading from above, the file PIP.COM is composed of 58 sectors (byte 15), stored in groups 7-14 (bytes 16-23); it is neither an R/O nor a SYS file, since bytes 9 and 10 are not set with the high bit = 1; this is the first extent (byte 12), and the last, because it is not full (bytes 24-31.) To erase a file, CP/M simply writes E5H in byte 0, in place of the 0 there now; the rest of the entry stays intact, but is ignored by the system in subsequent file operations. Groups 7-14 would be re-assigned as needed for any newly created files; until this happens, you could change byte 0 back to 0 at any time and 're-create' the file.

The group numbers in the last 16 bytes of the directory entry provide the actual map of the disk for CP/M. Obviously, if anything should happen to the directory entry for a file, such as changing even one of the group numbers, your computer would not be able to reconstruct the file. The function of the FORMAT program can be appreciated as well. It inserts E5H's all through the disk; most importantly, it puts them into the first byte of every directory entry, indicating no file is there. Any other number in the first byte would cause CP/M to try to construct a file name and map for each entry.

Let us look for a moment at the disk in terms of groups. Tracks 0-2, containing the CCP, etc., are always in the same place and don't have group numbers associated with them. Tracks 3-39 are completely divided into groups as follows: groups 0 and 1 contain the directory, groups 2-91 contain file space. If you have SYSGENed your disk just after FORMATting it, then the BIOS.SYS file will be in groups 2-6, since these are the first open groups after the directory space.

A little multiplication will show you that the 92 groups in tracks 3-39 don't occupy all the space in those tracks; 4 sectors in track 39 (numbers 7, 8, 15, and 16) are not assigned to groups, and therefore are never written on by files. This occurs because the directory groups reference 8 sectors, and these 4 are the remainder left on the disk which don't make up enough space to be included in a group by themselves. There are also unused sectors in track 2 (sectors 14-20), which aren't occupied by any of the CP/M system components that are present in the first three tracks, and which don't get included in the group maps in the directory. Together, these are the 11 sectors I mentioned at the start of the article which are never used in CP/M files or system areas - and these could be used by you if you were to access them directly through the BIOS, instead of by the usual BDOS operations that read/write files.

Absolute Sectors

When CP/M uses files, it really looks at the disk as a collection of groups; any sectors not in groups are ignored in file operations. But we can instead view a disk as simply 800 records and access them individually, without using file conventions at all. A major advantage of this is that we can read and write to unformatted disks (like an HDOS disk!). The BIOS has primitive routines that can be called from assembly level programs to do just this; they are described in the Alteration Guide of the infamous CP/M documentation. It is done by first using commands to position the disk head at the track and sector desired, then reading or writing to that sector with another command. This is almost identical to doing a disk read or write from the BDOS, except that you must position the disk head yourself before calling the read or write routine. The routine shown below will allow you to perform absolute sector I/O by first putting the track/sector values in the <B> and <C> registers, respectively, CALLing HEADSET, then either CALLing READ or WRITE as needed. In either case, the buffer used in memory for the disk I/O is defaulted to the DMA, which runs from 080H to 0FFH.

```
BIOS    EQU     F000H
*
* THIS IS THE BIOS START FOR AN H8-CP/M 64K SYSTEM
* FOR OTHER SYSTEMS, THE START OF THE BIOS IS IN
* MEMORY ADDRESS 0002H-0003H
*
SETTRK  EQU     BIOS+01EH
SETSEC  EQU     BIOS+021H
READ    EQU     BIOS+027H
WRITE   EQU     BIOS+02AH
*
HEADSET:
* CALL WITH TRACK NUMBER IN <B>
* AND SECTOR NUMBER IN <C> REG'S
        PUSH    PSW             ;SAVE SECTOR NUMBER
        MOV     C,B             ;PUT TRACK NUMBER IN <C>
        MVI     B,000H          ;ZERO HIGH BYTE
        CALL    SETTRK          ;SET TRACK NUMBER
        POP     B               ;GET SECTOR BACK IN <C>
        MVI     B,000H          ;ZERO HIGH BYTE
        CALL    SETSEC          ;SET SECTOR NUMBER
        RET
*
* SUBSEQUENT "CALL WRITE" WILL WRITE MEMORY CONTENTS
*       IN DMA (FROM 080H TO 0FFH) TO THE DISK SECTOR;
* SUBSEQUENT "CALL  READ"  WILL READ THE SPECIFIED
*       SECTOR INTO THE DMA AREA
*
```

What if you want to read a file using the above method, how do you find which sectors correspond to which groups in the file directory entry? The answer is a little complex, since the relationship between group number and track/sector numbers is not simple. To minimize access time during I/O, the BIOS does not assign group numbers to adjacent sectors; instead, it skips 7 sectors between each pair of sectors. The chart below uses an algorithm to permit you to calculate the sector and track numbers given the group number (or the reverse), but it takes a little study to understand!

Track	Sectors	Group	Track	Sectors	Group
(2y-1)	01-02	(5x)	(2y)	01-02	(5x+2)
	09-10			09-10	
	17-18			17-18	(5x+3)
	05-06			05-06	
	13-14	(5x+1)		13-14	
	03-04			03-04	
	11-12			11-12	(5x+4)
	19-20			19-20	
	07-08	(5x+2)		07-08	
	15-16			15-16	

""x" and ""y" above are found from the equation:

y = x + 2

e.g. for group 0, x=0 and y=2; for track 3, y=2 and x=0

Don't be too horrified by this chart. If you just want to use the sectors that CP/M doesn't normally use that I mentioned earlier, you only need the assembly routine. This translation from group to track/sector is only necessary if you want to find each sector for a file given in a directory entry.

The information in this article should give you a start on constructing your own utilities to 'un-erase' files, or even recover data from a disk with a damaged directory, since the disk can be accessed for I/O by the routine above even if it has not been FORMATted. HDOS disks can also be read as well as corrupted CP/M disks, but you will need to find out how HDOS organizes it's disks (it isn't the same!) to read the directory so that you can locate the sectors containing the file contents.

About the Author:

David Vinter *is a cell biologist at the University of Michigan and is just now completing his PhD in anatomy. His research involves blood vessels and vascular surgery. David's interest in computers goes back seven years, when he began using a mainframe, then an IBM-360 at the University. He purchased an H-8 about a year ago, and has been hooked on micros ever since.*

Getting Started With Assembly Language

(Or, It's Your Turn To Help Me, Now)

Pat Swayne
Software Engineer

In the last three installments of "Getting Started", we covered disk file operations in HDOS and CP/M. The first two of those three parts covered elementary disk file reading in HDOS and CP/M, and in the last installment, I presented the first really useful "mini-program" in this series to demonstrate more advanced disk operations in HDOS. Now it's the CP/M user's turn to get a useful demo program, as we cover...

Part IX — Disk Files in CP/M, Part 2

For those of you who are new to HUG, this series on assembly language started in the April 1983 issue. If you want to read the entire series, you should buy the bound volume of all the 1983 REMarks, which is part no. 885-4004.

In our first discussion of CP/M disk files, we examined a program that transferred a file from the disk to the screen by reading one record (128 bytes), printing it to the screen, and repeating the operation. I mentioned previously that there are basically two ways of handling disk files. You can handle one record at a time as we have already done, or you can work with the whole file or with all available memory, which ever is smaller. The second method is more efficient when you want to copy files, so it is the method we will use in this month's demo program. The demo program is a file copy utility called COPY that is run with the following syntax.

```
A>COPY (source drive:)FILENAME.TYP TO (destination drive:)
```

If the source drive is the currently logged drive, it may be left off. For example, if you wanted to copy ASM.COM from A: to B:, you could enter:

```
A>COPY ASM.COM TO B:
```

As with the HDOS version of this program, you are invited to share it with your friends who are newcomers to computers and who are confused with the "destination = source" syntax of PIP. This program is limited in that it does not understand "wild cards" or allow you to change the name of a file while copying it, but its ease of use may make it useful to beginners.

The COPY program first checks for a command line argument by looking at the first name character in the default FCB area. If no argument was given, a message explaining the program is printed. I like the TYPTX routine that is built in with HDOS so much that I often put my own TYPTX routine in CP/M or Z-DOS programs, including this one. The TYPTX routine prints any text that follows the call to it until a character with the 8th bit set is found, and then it returns to the address after that character.

If a command line argument is found, the program sets up things by finding out how much memory is available. It does this by getting the BDOS start address from location 6 and subtracting from it the CCP size (800H), the CP/M serial number size (6), and the size of one record (80H), for a total of 886H bytes. Then it determines if any memory space is left, and if so, goes on with the program. Otherwise, an error message is printed, and the program quits.

After the available memory space is computed, COPY moves the file name information from the default FCB to another FCB that will be used for the destination file. Then it searches the default DMA area (remember that command line arguments are placed there "as is" by CP/M) for the string "TO ". If it can't find the string, it prints the usage information message and quits. If it does find the string, it looks for the next non-space character and converts it to a number by subtracting the value of "@" from it, so that A is changed to the value 1, B to 2, etc. Then it makes sure that the destination drive is not the same as the source drive and prints an error message if it is.

You may remember that we did not perform the same drive check in the HDOS version of the program. That is because HDOS always writes to unused parts of a disk even if the file being written has the same name as one already on the disk. If the write is successful, it will then delete the old file with the same name. However, CP/M will just write over an existing file, so the program handles things differently. It does not allow the source and destination drives to be the same so that there is no chance of damaging the source file, and if a file with the same name is found on the destination drive, COPY will ask the user if he/she wants to delete it. It tests for the duplicate named file by trying to open it.

If the destination file does not already exist, or if the user has chosen to delete it, the COPY program makes a new directory entry for the destination. If there is no directory space, the user is notified, and the program exits. If everything is OK so far, the program enters a copy loop. Since CP/M can read only one 128-byte record at a time, I set up a read loop and a write loop in the program to simulate the ability of more advanced operating systems to read as much as possible at each pass. In each loop, the DMA address is placed within the reserved memory area and adjusted upward 128 bytes after each record until the read or write pass is finished. During the read loop, the number of records read is counted so that the same amount will be written in the write loop. The entire copy loop can be executed any number of times so that a file of any size can be copied.

This may seem like a lot of work just to copy a file, but it is the most efficient way to do it. If you want to do a little homework, try writing a version of this program that alternately reads and writes only one 128-byte record at a time until the file is copied, and see how much longer it takes than this method.

If the end of the file is detected during a read loop, a flag is set so that after the next write loop, the destination file will be closed. If the close operation goes OK, the program exits. If not, an error message is printed. There is also an error trap and message in the write loop, in case something goes wrong there.

Following the main body of the program are two subroutines that are used in it. The first is used to skip over spaces while examining the command line, and the second is the TYPTX routine mentioned before. After the subroutines comes the data area that holds the memory top address, the record counter, the end of file flag, the output FCB, the stack, and the copy buffer.

A Little Contest

Now that the CP/M users are caught up with HDOS users in this series, it's time to have the contest I mentioned in the last installment of "Getting Started". The contest is to create a program called RENM that will rename a file on the disk using the following syntax:

```
RENM (drive:)OLDFILE.NAM TO NEWFILE.NAM
```

One winner will be selected for HDOS and one for CP/M. Since we have not covered "cracking" filenames in CP/M yet, I will allow the CP/M version to use this syntax:

```
RENM (drive:)OLDFILE.NAM NEWFILE.NAM
```

With this syntax, CP/M will decode the filenames itself into the two default FCB areas. But try to do it using the first syntax.

The winners in each category will get their choice of any Heath/Zenith software product and any HUG software product. Submit your entry on a disk, if possible. I will judge the entries personally, and judgment will be based on how well the program works and its usefulness in a future "Getting Started" article as a tutorial program. The program should be well commented, and any accompanying text explaining it will be appreciated. So get started, and best of luck!

```
*       COPY - FILE TRANSFER PROGRAM
*
*       THIS PROGRAM DEMONSTRATES READING AND WRITING
*       DISK FILES UNDER CP/M.  FILES ARE READ AND
*       WRITTEN IN LARGE BLOCKS.
*
*       THIS PROGRAM COPIES FILES USING THE FOLLOWING
*       SYNTAX:
*
*       A>COPY d:FILENAME.EXT TO d:
*
*       WHERE d: IS A DRIVE DESIGNATION.

*       BY P. SWAYNE, HUG  14-NOV-83

*       CP/M DEFINITIONS

CONIN   EQU     1
CONOUT  EQU     2
OPEN    EQU     15
CLOSE   EQU     16
DELETE  EQU     19
READ    EQU     20
WRITE   EQU     21
MAKE    EQU     22
SETDMA  EQU     26
CURDSK  EQU     4
BDOS    EQU     5
DFCB    EQU     5CH
DMA     EQU     80H

        ORG     100H

*       MAIN PROGRAM

START   LXI     H,0
        DAD     SP              ;LOCATE STACK
        LXI     SP,STACK        ;SET NEW ONE
        PUSH    H               ;SAVE OLD ONE
        LDA     DFCB+1
        CPI     ' '             ;CHECK FOR ENTRY
        JNZ     GOTFILE         ;IF SO, CONTINUE
ERREX   CALL    TYPTX
```

```
          DB      13,10,'The correct use of this program is',13
          DB      10,10,'A>COPY d:FILENAME.TYP TO d:',13,10,10
          DB      'where d: is a drive designation '
          DB      '(A:, B:, etc.),',13,10
          DB      'and FILENAME.TYP is file you want to COPY.'
          DB      13,10+80H
EXIT      POP     H                   ;GET OLD STACK
          SPHL                        ;SET IT
          RET                         ;RETURN TO CP/M
GOTFILE   LHLD    BDOS+1              ;GET BDOS ADDRESS
          LXI     D,-886H             ;SUBTRACT CCP, SER., AND REC. SIZE
          DAD     D                   ; TO FIND MEMORY LIMIT
          SHLD    MEMTOP              ;SAVE MEMORY TOP
          MVI     A,BUFFER/256        ;GET BUFFER ADDR HIGH
          SUB     H                   ;ANY MEMORY AVAILABLE?
          JC      GOTMEM              ;YES, GO AHEAD
          CALL    TYPTX
          DB      13,10,'ERROR - Not enough memory.',13,10+80H
          JMP     EXIT                ;RETURN TO CP/M
GOTMEM    LXI     H,DFCB+1            ;POINT TO FCB
          LXI     D,OUTFCB+1          ;AND OUTPUT FCB
          MVI     B,15                ;MOVE 15 CHARACTERS
MOVFCB    MOV     A,M
          STAX    D                   ;MOVE THE FILE NAME
          INX     H
          INX     D
          DCR     B
          JNZ     MOVFCB
          LXI     H,DMA+1             ;POINT TO COMMAND ARGUMENT
          CALL    SOS                 ;SKIP OVER SPACES
FNS       MOV     A,M                 ;GET A CHARACTER
          CPI     ' '                 ;SPACE?
          JZ      FNDSP               ;FOUND SPACE
          INX     H                   ;ELSE, INCREMENT POINTER
          JMP     FNS                 ;AND FIND NEXT SPACE
FNDSP     CALL    SOS                 ;SKIP OVER IT
          CPI     'T'                 ;LOOK FOR 'TO'
          JNZ     ERREX               ;EXIT IF NOT FOUND
          INX     H
          MOV     A,M
          CPI     'O'
          JNZ     ERREX
          INX     H
          MOV     A,M
          CPI     ' '
          JNZ     ERREX
          CALL    SOS                 ;FOUND "TO", SKIP TO DRIVE NAME
          SUI     '@'                 ;REMOVE ASCII FROM DRIVE CODE
          STA     OUTFCB              ;AND SET UP OUTPUT FCB
          MOV     B,A                 ;SAVE DRIVE CODE
          LXI     D,DFCB
          LDAX    D
          ORA     A                   ;INPUT DRIVE DEFAULT?
          JNZ     NOTDEF              ;NO
          LDA     CURDSK              ;ELSE, GET CURRENT DISK
          INR     A                   ;MAKE IT START WITH 1
NOTDEF    CMP     B                   ;COMPARE INPUT AND OUTPUT DRIVES
          JNZ     NOTSAM              ;NOT THE SAME
```

```
        CALL    TYPTX
        DB      13,10,'ERROR - Same drives.',13,10+80H
        JMP     EXIT
NOTSAM  MVI     C,OPEN
        CALL    BDOS                ;OPEN INPUT FILE
        INR     A
        JNZ     GDOPEN              ;OPEN OK
        CALL    TYPTX
        DB      13,10,'ERROR - File not found.',13,10+80H
        JMP     EXIT                ;RETURN TO CP/M
GDOPEN  LXI     D,OUTFCB
        MVI     C,OPEN
        CALL    BDOS                ;TRY TO OPEN OUTPUT
        INR     A                   ;ANY FILE?
        JZ      GDOUT               ;NO, GO AHEAD
        CALL    TYPTX
        DB      13,10,'File exists, erase? (Y/N) <N>'
        DB      ' '+80H
        MVI     C,CONIN
        CALL    BDOS                ;GET INPUT
        CPI     'Y'                 ;CHECK FOR 'Y'
        JZ      ERAFIL              ;GOT IT, ERASE FILE
        JMP     EXIT                ;ELSE, RETURN TO CP/M
ERAFIL  LXI     D,OUTFCB
        MVI     C,DELETE
        CALL    BDOS                ;DELETE OLD FILE
GDOUT   LXI     D,OUTFCB
        MVI     C,MAKE
        CALL    BDOS                ;MAKE NEW DIRECTORY ENTRY
        INR     A                   ;TEST
        JNZ     LOOP                ;OK
        CALL    TYPTX
        DB      13,10,'ERROR - No directory space.',13,10+80H
        JMP     EXIT

*       COPY LOOP

LOOP    LXI     H,0
        SHLD    RECCNT              ;CLEAR RECORD COUNTER
        LXI     D,BUFFER            ;POINT TO BUFFER
READLP  PUSH    D                   ;SAVE POINTER
        MVI     C,SETDMA
        CALL    BDOS                ;SET DMA ADDRESS
        LXI     D,DFCB
        MVI     C,READ
        CALL    BDOS                ;READ FROM FILE
        POP     D                   ;RESTORE POINTER
        ORA     A                   ;TEST READ OPERATION
        JZ      NOTEND              ;END NOT FOUND YET
        STA     LASTFLG             ;ELSE, SET LAST PART FLAG
        JMP     WRFILE              ;AND WRITE THE FILE
NOTEND  LXI     H,80H
        DAD     D                   ;UPDATE POINTER
        XCHG                        ;RETURN IT TO DE
        LHLD    RECCNT
        INX     H                   ;COUNT RECORD READ
        SHLD    RECCNT
        LHLD    MEMTOP              ;GET MEMORY TOP
        MOV     A,L
        SUB     E                   ;SUBTRACT POINTER FROM IT
        MOV     A,H
        SBB     D
        JC      WRFILE              ;NO ROOM, WRITE FILE
        JMP     READLP              ;ELSE, READ MORE
WRFILE  LHLD    RECCNT
        MOV     A,H
        ORA     L                   ;EMPTY FILE COPIED?
        JZ      DONE                ;IF SO, WE'ER DONE!
        LXI     D,BUFFER            ;RESET POINTER
WRITLP  PUSH    D                   ;SAVE POINTER
        MVI     C,SETDMA
        CALL    BDOS                ;SET DMA ADDRESS
        LXI     D,OUTFCB
        MVI     C,WRITE
        CALL    BDOS                ;WRITE FILE
        POP     D                   ;RESTORE POINTER
        ORA     A                   ;GOOD WRITE?
        JZ      GDWRIT              ;YES
        CALL    TYPTX
        DB      13,10,'ERROR - No disk space.',13,10+80H
        JMP     EXIT
GDWRIT  LXI     H,80H
        DAD     D                   ;ELSE, UPDATE POINTER
        XCHG
        LHLD    RECCNT
        DCX     H                   ;DECREMENT RECORD COUNTER
        SHLD    RECCNT
        MOV     A,H
        ORA     L                   ;ALL DONE?
        JNZ     WRITLP              ;IF NOT, CONTINUE
        LDA     LASTFLG
        ORA     A                   ;END OF FILE?
        JZ      LOOP                ;IF NOT, GET MORE

*       COPY DONE, CLOSE OUTPUT FILE

DONE    LXI     D,OUTFCB
        MVI     C,CLOSE
        CALL    BDOS                ;CLOSE FILE
        INR     A
        JNZ     GDCLOSE             ;GOOD CLOSE OPERATION
        CALL    TYPTX
        DB      13,10,'ERROR - Can''t close file.',13,10+80H
        JMP     EXIT
GDCLOSE CALL    TYPTX
        DB      13,10,'DONE!',13,10+80H
        JMP     EXIT

*       SUBROUTINES

*       SKIP OVER SPACES

SOS     MOV     A,M                 ;GET A CHARACTER
        CPI     ' '                 ;SPACE?
        RNZ                         ;IF NOT, RETURN
        INX     H                   ;ELSE, MOVE TO NEXT CHAR
```

```
         JMP     SOS               ;TRY AGAIN

*        TYPE TEXT FOLLOWING CALL

TYPTX    XTHL                      ;SAVE HL, GET TEXT ADDR
TYPTX1   MOV     A,M               ;GET CHARACTER
         PUSH    H                 ;SAVE POINTER
         MVI     C,CONOUT
         ANI     7FH               ;STRIP MARKER BIT
         MOV     E,A               ;CHARACTER TO E
         CALL    BDOS              ;PRINT IT
         POP     H                 ;RESTORE POINTER
         MOV     A,M               ;GET CHARACTER AGAIN
         INX     H                 ;INCREMENT POINTER
         ORA     A                 ;TEST FOR END
         JP      TYPTX1            ;NOT THERE, YET
         XTHL                      ;ELSE, FIX STACK, GET HL
         RET

*        DATA AREA

MEMTOP   DW      0                 ;MEMORY TOP
RECCNT   DW      0                 ;RECORD COUNTER
LASTFLG  DB      0                 ;LAST SEGMENT FLAG
OUTFCB   DB      0,'              ',0,0,0,0
         DS      16
         DB      0
         DS      32                ;RESERVE STACK SPACE
STACK    EQU     $
BUFFER   EQU     $                 ;BUFFER STARTS HERE

         END     START
```

Thinking about your next vacation?

Plan now to attend the Third International HUG Conference!

The Third International HUG Conference will be held at a family resort facility from July 27 through July 29, 1984. Activities for the family include golf, tennis, softball, horseshoes, racquetball, horseback riding, river boat excursion, volleyball, and health club facilities. More than adequate night life is available for your enjoyment. Historic settings, found in the immediate area, provide an abundance of antique and gift shops. More information will be available later, so watch REMark.

Plan your vacation, bring the family and join the fun with Heath/Zenith computer users from all over the world!

Train

Paul R. Hinson
1359 W. Harvard Place
Ontario CA 91762

My four year old developed a recent urge to 'do his numbers' which means he wants me to write down a number with him verbally responding with the answer. After developing a fairly good number recognition, I introduced him to simple addition using the fingers. Again showing good progress and having shown past interest in the computer prompted development of this program.

The idea: Draw a presentation of a train on the screen and advance it to the right as a reward for correct answers.

The program takes advantage of some of the smart terminal commands as listed in the front of the program. The text contains 3892 bytes, symb 176 and strn 125. Benton Harbor BASIC 10.06.00 was used on an H8 computer and H-19 terminal. To play, the computer requests the names of the opponents so it can display their names on the side of the 'ENGINE' and to prompt the user as to who's turn it is. The question is displayed and upon receipt of a correct answer, the terminal rings three times to signify a correct answer, the correct answer register is updated, the train moves down the track, and another sequence is generated. If the answer is incorrect, the correct answer is momentarily displayed and another sequence is generated. At the end of the game (20 questions), the computer displays the winners name, the correct percentage for each player, and queries about additional games.

Special note on the poke commands. The BASIC program at lines 1010, 1030, and 1040 is changed except when division is selected. If there are any modifications in these lines, adjustments may have to be made.

In addition to these special features, the maximum number generator is requested to limit the highest random number presented to the players. Although some adult supervision is required for younger users, a personal touch and that feeling of togetherness will also be rewarding to all participants. It even helps out mom, school teachers, and possibilities of ill feelings of the time spent at the computer.

```
1ØØØ GOTO 1Ø9Ø
1Ø1Ø PRINT SPC(S);A:PRINT SPC(T);"+";B
1Ø2Ø PRINT SPC(T+2);:FOR I=1 TO 7-T:PRINT "-";:NEXT I
1Ø3Ø PRINT ZØ$:INPUT ;X:PRINT Z$;:IF A+B=X THEN 17ØØ
1Ø4Ø PRINT "Sorry, the answer is";A+B:PAUSE 75Ø:RETURN
1Ø5Ø REM  ***************************************************************
1Ø6Ø REM *                                  TRAIN                              *
1Ø7Ø REM * BASIC 1Ø.Ø6.ØØ                                          by PAUL HINSON *
1Ø8Ø REM  ***************************************************************
1Ø9Ø E$=CHR$(27):        REM        ESCAPE KEY
11ØØ C$=E$+"E":          REM        CLEAR SCREEN
111Ø R$=E$+"p":          REM        ENTER REVERSE VIDEO
112Ø RØ$=E$+"q":         REM        EXIT REVERSE VIDEO
113Ø G$=E$+"F":          REM        ENTER SPECIAL GRAPHICS
114Ø GØ$=E$+"G":         REM        EXIT SPECIAL GRAPHICS
115Ø Y$=E$+"Y":          REM        DIRECT CURSOR ADDRESSING
116Ø Z$=E$+"x5":         REM        CURSOR OFF
117Ø ZØ$=E$+"y5":        REM        CURSOR ON
118Ø D$=E$+"J":          REM        ERASE PAGE AFTER CURSOR
119Ø DIM N$(1):DIM G(1):PRINT C$
12ØØ REM ************************   INSTRUCTIONS   **************************
121Ø PRINT "   This is a game of  'MATH'  for two players.  I will randomly"
122Ø PRINT "display two numbers between Ø and your selected highest. Answer"
123Ø PRINT "correctly and I will move your ENGINE down the track and update"
124Ø PRINT "the correct question `register'.  Twenty (2Ø) questions will be"
125Ø PRINT "given and there will be  NO  second chances.  GOOD LUCK!!"
126Ø REM ************************   GET OPPONENTS   *************************
127Ø PRINT :LINE INPUT "Who is playing? ";N$(Ø)
128Ø A$="Please use a name with 1Ø letters or less.  Re-enter!!!"
129Ø IF LEN(N$(Ø))>1Ø THEN PRINT :PRINT A$:GOTO 127Ø
13ØØ PRINT :LINE INPUT "And who is your opponent? ";N$(1)
131Ø IF LEN(N$(1))>1Ø THEN PRINT :PRINT A$:GOTO 13ØØ
132Ø REM *********************   SET HIGHEST NUMBER   **********************
133Ø PRINT :INPUT "What is the highest number allowed (Ø-1ØØØØØ)? ";N
134Ø IF N>1ØØØØØ THEN PRINT :PRINT "Your number is too large!!!":GOTO 133Ø
135Ø REM ***************************   SET THE GAME   **********************
136Ø PRINT :PRINT "What type of math would you like?":PRINT
137Ø PRINT "     1)  Addition":PRINT "     2)  Subtraction"
138Ø PRINT "     3)  Multiplication":PRINT "     4)  Division":PRINT
139Ø INPUT "Choose 1,2,3 or 4 ==> ";S1:IF S1>4 OR S1<1 THEN 136Ø
14ØØ IF S1=1 THEN POKE 19157,43:POKE 192Ø9,43:POKE 19247,43
141Ø IF S1=2 THEN POKE 19157,45:POKE 192Ø9,45:POKE 19247,45
142Ø IF S1=3 THEN POKE 19157,12Ø:POKE 192Ø9,42:POKE 19247,42
143Ø REM **********************   PRINT THE TRAIN   ************************
144Ø PRINT C$;Z$
145Ø X=G(Y):PRINT X
146Ø PRINT SPC(3*X);"  ";G$;R$;"|";RØ$;"zz";R$;"}";RØ$;"      ";R$;"r_"
147Ø PRINT RØ$;SPC(3*X);"  ";R$;GØ$;" ";N$(Y);G$;
148Ø FOR L=1Ø TO LEN(N$(Y)) STEP -1:PRINT " ";:NEXT L
149Ø PRINT :PRINT RØ$;SPC(3*X);"~wwwwwwwwwwwwww"
15ØØ PRINT SPC(3*X);" (^^)     (^) >":Y=Y+1:IF Y=2 THEN Y=Ø
151Ø IF R>1 THEN PRINT Y$;"- ";D$;:GOTO 154Ø
152Ø A$="bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb":A$=A$+A$:PRINT A$
153Ø R=R+1:IF R<2 THEN 145Ø
154Ø PRINT GØ$:REM  *********  RANDOM NUMBER GENERATOR AND PRINTER  ********
155Ø A=INT(N*RND(1)):B=INT(N*RND(1)):R=R+1:IF R=43 THEN 175Ø
156Ø IF A<B THEN W1=A:A=B:B=W1:REM **********  REVERSE RAND #'S  ***********
157Ø PRINT :PRINT N$(Y);", here is your question #";INT((R-1)/2):PRINT
158Ø REM *********************   MARGIN JUSTIFICATION   *********************
159Ø J=1:S=7:K=1:T=6
16ØØ IF A/J >= 1Ø THEN J=J*1Ø:S=S-1:GOTO 16ØØ
161Ø IF B/K >= 1Ø THEN K=K*1Ø:T=T-1:GOTO 161Ø
162Ø IF S1 <> 4 THEN 166Ø: REM  ********  GO AROUND IF NOT DIVIDE  *********
163Ø PRINT SPC(1Ø);:FOR I=1TO 5+(7-T):PRINT " ";:NEXT I:PRINT
164Ø PRINT SPC(S);A;")";A*B:PRINT ZØ$;:INPUT ;X:PRINT Z$;:IF X=B THEN 17ØØ
165Ø PRINT "Sorry, the answer is";B:PAUSE 75Ø:GOTO 168Ø
166Ø GOSUB 1Ø1Ø:REM ***  GET THE BASIC LINES FOR MOD IN FRONT OF PROGRAM ***
167Ø REM ****  TO ELLIMINATE LOCATION ERRORS DURING TYPING IN OF PROGRAM ***
168Ø PRINT Y$;"- ";D$:Y=Y+1:IF Y=2 THEN Y=Ø
169Ø GOTO 155Ø
17ØØ PRINT CHR$(7);:PAUSE 1ØØ:PRINT CHR$(7);:PAUSE 1ØØ:PRINT CHR$(7);
171Ø G(Y)=G(Y)+1
172Ø IF Y=Ø THEN PRINT Y$;"  ":GOTO 145Ø
173Ø IF Y=1 THEN PRINT Y$;"& ":GOTO 145Ø
174Ø REM  ***********  END OF GAME RESULTS AND NEXT GAME QUARRY  ***********
175Ø IF G(Ø)>G(1) THEN PRINT N$(Ø);" has won the game!!!  CONGRATULATIONS!!!"
176Ø IF G(Ø)<G(1) THEN PRINT N$(1);" has won the game!!!  CONGRATULATIONS!!!"
177Ø IF G(Ø)=G(1) THEN PRINT "We have a tie game!!!"
178Ø PRINT :PRINT N$(Ø);" got";INT((G(Ø)/2Ø)*1ØØ);"% score.":PRINT
179Ø PRINT N$(1);" got";INT((G(1)/2Ø)*1ØØ);"% score.":PAUSE 75Ø
18ØØ PRINT ZØ$:LINE INPUT "Do you wish to play again without changes? ";A$
181Ø IF LEFT$(A$,1)="Y" OR LEFT$(A$,1)="y" THEN R=Ø:G(Ø)=Ø:G(1)=Ø:GOTO 143Ø
182Ø PRINT :LINE INPUT "Do you wish to start a new game? ";A$
183Ø IF LEFT$(A$,1)="Y" OR LEFT$(A$,1)="y" THEN R=Ø:G(Ø)=Ø:G(1)=Ø:GOTO 126Ø
184Ø PRINT :PRINT "Thanks for playing ";N$(Ø);" and ";N$(1);".":END
```

A quick look at... dBASE II Programmer's Notebook

Tom Huber
Related Products Editor

dBASE II Programmer's Notebook is a little booklet (46 pages, 8-1/2 x 11") by Steven G. Meyerson that is just what it says it is: a programmer's notebook. It is full of useful hints and techniques for using Ashton Tate's dBASE II relational data based management program. Since it is small (in pages, not content), this review is also small.

For those of you who are not familiar with dBASE II, it is a powerful data base handler that has its own "programming language", similar to a job control language (but not nearly as confusing or hard to use as some). Hence, the need for a book like this to supplement Ashton Tate's own manuals.

What It Contains

There are three parts to this book: Hints and Techniques, Routines, and S-Mail. The first two parts contain the real meat of the material. There are good hints on how to get better performance out of dBASE II, how to avoid some of the pitfalls inherent in this powerful program, and several practical routines for expanding its use. The routines are: Two-Column Printing, Menu Operations, Error Checking, File Name Check, and Flashing Display. The third part of the booklet is a real bonus: S-Mail, a complete mailing list handler with an archiving feature for purged records.

Comment

Mr. Meyerson has had obvious experience with dBASE II and his skill and knowledge show through in the material contained in this book. The writing is clear and lucid; the author presents his information in an informal, straight forward manner, something that is often missing in many books. There is no attempt at humor; the material is all business. Although the booklet is produced from a letter-quality printer (not typeset), it is easy to read.

At first glance, the price seems a bit steep for the number of pages; $12.95 for 13 pages of notes, 8 pages of routines, and 24 pages of instructions and listings for S-Mail. However, considering the price of commercial mailing list programs, the booklet suddenly becomes a bargain and a couple of the hints alone could well save the user more than thirteen dollars of time and frustration.

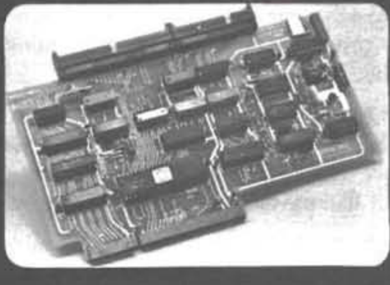
This book is not for everyone. For instance, I don't recommend it to anyone who does not have dBASE II. However, if you do own dBASE II and want to expand your expertise in programming it for yourself and others, then I strongly recommend it.

One final note: if you don't want to type in all the listings for S-Mail, the author is offering a disk copy for a paultry $7.00 more.

Vendor: CompuTech
P.O. Box 2027
Poquoson, VA 23662
(804) 868-8055

Price: $12.95 book alone
+ $7.00 for disk of S-Mail
(VA residents add 4% sales tax)

1 CONTROLLER



FOR 8" & 5.25" DRIVES

Now be able to run standard 8" Shugart compatible drives and 5.25" drives (including the H37 type) in double and single density, automatically with one controller.

Your hard sectored 5.25" disks can be reformatted and used as soft sectored double density disks. The FDC-880H operates with or without the Heath hard sectored controller.

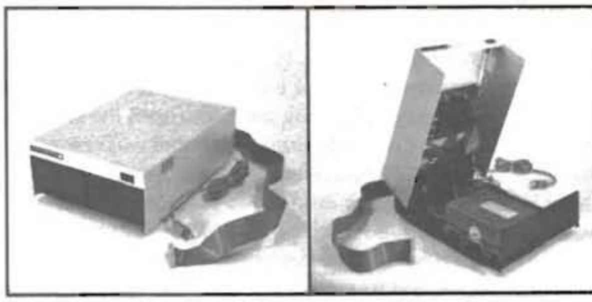
NEW PRICE $495

Includes controller board CP/M boot prom, I/O decoder prom, hardware/software manuals BIOS source listing. HDOS driver now available for $40.00.

5-20 day delivery–pay by check, C.O.D., Visa, or M/C.

Contact:
C.D.R. Systems Inc.
7210 Clairemont Mesa Blvd.
San Diego, CA 92111
Tel. (619) 560-1272

NOW 12 MEGABYTE
(CDR-10M) $2995



WINCHESTER SYSTEM
For the Heath/Zenith Computer

Systems complete with software, case, power supply, signal cable and interface.
Runs with CP/M, on the H/Z89 & H8 (with Z80 card).

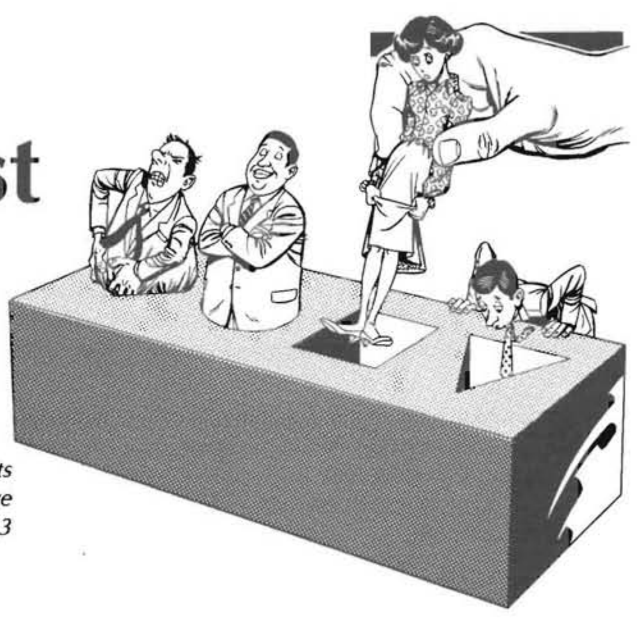
- Switching power supply
- Expansion for backup installations
- Auto attach BIOS
- Hard disk utilities
- Formatting program
- 1 year parts & workmanship warranty

CP/M is a trademark of Digital Research. Heath, H8, H89 are trademarks of Heath Corporation. Zenith, Z89, Z90 are trademarks of Zenith Data Systems.

5-20 day delivery. Pay by check, C.O.D., Visa, or M/C.

Contact:
C.D.R. Systems Inc.
7210 Clairemont Mesa Blvd.
San Diego, CA 92111
Tel. (619) 560-1272

Squeezing The Most Out of Your HDOS Diskettes



Glenn F. Roberts
9035 F Countrywood Drive
Knoxville, TN 37923

The intent of this article is to explain ways in which one can reduce the amount of floppy disk space which must be reserved for system files in the Heath Disk Operating System (HDOS). This is a particularly important issue to H-8 and H/Z-89 users with only a single hard-sectored type floppy disk drive. To a large extent this article pulls together ideas from past articles in REMark, Sextant, and Microcomputing, however, some of the ideas presented here have not been described elsewhere. Even readers who don't feel they need any more storage capacity (if such people exist) should be interested in the disk concepts presented here.

Note: Some of the techniques presented here can lead to essentially irreversible damage to the disk indexing structure if not performed properly. It is recommended that you test them out on unused disks first or else back up everything before starting.

Introduction

When I finished constructing my H-8 several years ago, one of the first programs I wanted to try out was "Adventure" (HUG part no. 885-1010). I was recalling playing the game until the wee hours of the morning during my undergraduate years and was anxious to pick up my adventure where I had left off, somewhere in the twisty little maze (or was it the maze of little twisty ...?). It was then that I got my first real lesson in the realities of HDOS: it is big. After INITing and SYSGENing a new disk, I deleted all of the files that didn't have the L (lock) flag set and found myself with 240 free sectors on the new disk. A quick CATalog of the Adventure disk showed that I apparently needed at least 246 (35 for the game itself, 188 for the main data file, and 23 for the game parameters file). It took me a day or two of fooling around until I discovered I could simply delete the supposedly "locked" file DK.DVD to gain an extra 16 sectors, more than enough to let me get my Adventure program up and running!

Since then I have learned a great deal about how HDOS works and what tricks and techniques can be used to increase the user's portion of the disk space allocation. I thought it would be fun to go back now and see just how many of those system sectors I could have recovered for my own use had I known then what I know now. I ended up recovering 56 more sectors, enough to let me SAVE three Adventure games and still have room to spare. In the sections that follow, I will explain the steps I took to recover these sectors. Much of this information is taken from past articles in REMark, Sextant, and Microcomputing. The articles are referenced in the text by listing the author and year of publication; refer to the listing at the end of the article for the complete article citation. I urge the reader to look up the original articles whenever possible. I will limit my discussion to the "hard sectored" (H-17) type floppy disk formats which have been standard on the H-8s and H/Z-89s, since these are the ones most likely to exhibit space limitation problems. Much of what I say however applies to HDOS in general and thus is also applicable to the newer "soft sectored" disks.

Minimal SYSGENing

Much of my original frustration with the Adventure game could have been bypassed if I had simply taken the time to carefully read through the HDOS manuals. Had I done so, I would have found that the SYSGEN program has an option switch called "/MIN". SYSGEN is the program which copies the various system programs which are needed to make a disk bootable. By typing SYSGEN /MIN one can request that a minimal set of these system programs be copied to the new disk. The files copied in this minimal configuration are described below:

HDOS.SYS — Contains the main HDOS code including the TT: device driver and the resident SCALLs (system routines) such as .SCIN, .SCOUT, .EXIT, etc. This code must be resident in high memory at all times except during initial boot-up.

HDOSOVL0.SYS — This file contains the principal overlaid SCALLS such as .OPENR, .CLOSE, .POSIT. These are "overlaid" since HDOS has the ability to swap them out to disk when a user's program requires a large amount of memory.

HDOSOVL1.SYS — This file contains less frequently used SCALLS such as .MOUNT, .RESET, and .DAD. Like HDOSOVL0, this file is also overlaid.

SYSCMD.SYS — This is the command processor program which is loaded into low memory whenever the user is at the command level. This is the program which generates the familiar ">" command prompt. Its job is to process user commands either by calling system routines (SCALLs), by linking to PIP, or by linking to a user's program. SYSCMD is loaded and takes control whenever an .EXIT SCALL is issued by a running program.

PIP.ABS — This is a general file and device utility (Peripheral Interchange Program) which copies, lists, renames, and deletes files and provides the user access to the disk directories. Some of the system commands (e.g. COPY, TYPE, CAT, etc.) explicitly call PIP but it is loaded and executed only as needed.

SY.DVD — This is the device driver for the H-17 disk drives. Its job is to interpret specific device driver calls (e.g. read, write, mount, abort, open, close, etc.) and control the disk hardware accordingly. The important thing to remember about device drivers is that the driver calls are the same regardless of the type of hardware being accessed. This "device independence" allows the application programmer to use software to interface almost any type of I/O device to the system. For more information on device drivers and driver calls, see "The HDOS Device Driver Programmer's Guide", (Dallas, et al., 1981), or refer to the source files on the "Device Drivers" disk distributed with HDOS.

RGT.SYS — This is the Reserved Group Table. Its primary function is to allow HDOS to lock out disk sectors which are to be flagged as unreadable.

GRT.SYS — This is the Group Reservation Table. HDOS stores files in pieces which can be strung out on random tracks and sectors throughout the disk. The GRT is a crucially important file since it contains the pointer information which allows HDOS to piece files back together into their original form. If the GRT file is damaged in any way, HDOS may be unable to read any of the files on the disk.

DIRECT.SYS — This file contains the disk directory. The information in this file, combined with that in the RGT and GRT tables, allows HDOS to locate and modify any file catalogued on the disk, either through a utility program like PIP or via a user program's SCALLs.

Note: For more information on the RGT, GRT, and DIRECT files, the interested user is strongly urged to read Tom Jorgensen's article "Dissecting the HDOS Diskette" (Jorgenson, 1981) and Herb Friedman's article "Understanding HDOS" (Friedman, 1983).

The above files will require a total of 129 sectors and the CAT command will report that there are 256 free sectors, for a total of 385 sectors. We know however that the disk is capable of storing 400 sectors (40 tracks at 10 sectors per track), so what happened to the other 15 sectors? Ten of these "missing" sectors are accounted for by track 0, sometimes called the boot track. When a new disk is INITialized, a "bootstrap" program is written on the first nine sectors of this track. When the computer is booted, the routines in ROM seek out track zero, load these nine sectors, and execute this bootstrap program. In order to prevent HDOS from storing files here, the entire first track is flagged as unusable by locking it out in the Reserved Group Table (RGT) file. Thus, as far as HDOS is concerned, these 10 sectors are unusable. The sectors on track 0 are normally read only during the process of mounting, dismounting, or booting the disk.

The remaining five "missing" sectors are hidden in five of the nine system files listed above. This occurs because HDOS allocates file space in groups (also called clusters or extents) of two sectors at a time, thus in reality all files require an even number of sectors on the disk. Even though the CAT command may show that a file has an odd length (e.g. HDOS.SYS is 31 sectors long), the user must remember that an extra sector is reserved for use by such files and that this expansion space will not be counted as being available for other uses. If you issue the CAT command with the /ALL flag, you will see all allocated sectors shown and a total of 390 sectors (i.e. all but the locked out track 0).

Resetting SY0:

Before going any further, I should take some time to discuss how and when you may replace the main "boot" disk in drive 0. This is important on single disk systems where you may not always need the system files to be present. When a diskette is MOUNTed in HDOS either directly, via the MOUNT or RESET command, or indirectly by "booting" up with the disk, HDOS stores certain critical information regarding the diskette in system RAM. This information includes the volume number of the disk and the absolute track and sector locations of various system and directory related files on the disk. The primary benefit of this scheme is that it speeds up disk access time. The price we pay is that we cannot arbitrarily swap disks without first informing HDOS via the RESET command (or equivalently the /RES option of PIP).

Before RESETting the boot drive (SY0:) one should normally set the "STAND-ALONE" flag using the SET program (Cohn, 1983; Pinkston, 1983). The stand-alone mode of HDOS is a frequently used yet undocumented feature of HDOS which forces all "swapped" files to remain resident in system RAM (these files include the system overlays and device drivers). To set the stand-alone mode merely type:

SET HDOS STAND-ALONE

Normal swapping mode can later be restored by SETting NO-STAND-ALONE. The advantage of this feature is that SY0: can be RESET to a disk containing only SYSCMD.SYS and PIP.SYS. The disadvantage is that the swapped out files now use up a chunk of RAM (usually about 6K).

You can also change the disk in SY0: if you are running an application program which is written to explicitly anticipate this situation. The HUG DUMP program (HUG part #885-1062) does this by revectoring one of the system ROM routines to a new routine which does not check the disk volume number when it reads a sector from the disk. Thus when you are running DUMP you may swap diskettes arbitrarily in any of the drives so long as you put everything back where it was before exiting from the program! Other ways to get around these safety features of HDOS are mentioned in the article "Disk Programming Without HDOS" (Smith, 1982).

I mention these points now since they may be of use in making the various patches and changes discussed in the remainder of this article, especially for readers with single drive systems.

Patch History Tables

If the MINimal SYSGEN does not give you the file space you need, the first thing you might want to consider is eliminating the Patch History Table (PHT) sector from all of your system files. The PHT was designed as a special feature of HDOS to allow the PATCH program to maintain a log of patches made to system programs (see Swayne, 1982a). In practice the PHT feature is rarely, if ever, used and can easily be eliminated by simply removing the PHT sectors appended to each system file. Note however, in light of my previous comments, that removing the PHT sector will only be useful on files which originally had an odd number of sectors. Such files will be reduced to an even number of sectors and you will have recovered two sectors.

The technique for removing the PHT is documented in the article "Losing Weight with HDOS 2.0" (Swayne, 1981a). In this article, Pat Swayne presents an assembler program which automatically strips the PHT sector off system files. *(Editor' note: Be sure to correct the program as shown in REMark #21, page 4.)* The following files can be reduced by two seconds each: EDIT, PATCH, INIT, SYSGEN, TEST47, ASM, XREF, DBUG, PIP, HDOS, and HDOSOVL1. Reducing these last three would raise our free sector total to 262 in the minimal configuration described previously.

A Compact SY: Driver

If 262 sectors is not enough user space for you, you might next consider creating a compact SY: device driver. This is another trick first pointed out by Pat Swayne in his brief article "A Tiny SY.DVD" (Swayne, 1981b). The SY: device driver is actually two programs joined together; the driver itself, and the initialization code. The designers of HDOS decided to include the device specific portion of the initialization code in the driver itself. This makes it possible to use the same INIT program to initialize various disk types (e.g. the 5-1/4" H-17 or the 8" H- 47), but it means we must carry along this initialization code in all our drivers, even those not used for INITialization.

The "Device Drivers" disk distributed with HDOS contains the file SYDVD.ASM which is the assembler source for the device driver portion of the SY: driver. This was designed to be combined with SYINIT (the initialization portion of the SY: driver) using the utility program MAKMSD, however, the device driver portion also functions fine by itself as long as you don't want to initialize any disks using it. If you assemble SYDVD.ASM, delete the original SY.DVD file (it can be deleted even though it is locked), and rename SYDVD.ABS as SY.DVD you will have an SY: device driver which occupies only 4 sectors. This represents a savings of 6 sectors. Total free space on our compact system disk is now 268 sectors! (**Note:** if you are working on a single drive system, be sure to reboot after doing anything with any device driver on your boot disk. This is necessary because HDOS looks for device driver addresses only once, at boot-up time, and moving the track and sector location of any device driver without informing HDOS can be disastrous.)

Trimming Fat DIRECTories

The HDOS directory (file DIRECT.SYS) is normally 9 groups (18 sectors) long. Each group can store 22 file entries for a total of 198 file entries. As Tom Jorgensen points out (Jorgenson, 1981), this is actually 22 more entries than the number of files it is currently possible to write on the diskette. In surveying my own disks, I have found that most of my diskettes contain no more than 30 or 40 files, and some have fewer than 20. This means that I usually need only 2 or 4 sectors allocated to the DIRECTory and thus can recover 14 to 16 sectors. There are a number of "public domain" programs which allow one to shorten the length of the directory. These may be available through a local HUG or in the HUG area on Compuserve. *(Editor's note: There is also REDUCDIR on HUG disk 885-1120.)* If you don't have access to one of these programs, you can recover these sectors using the HUG's DUMP program (disk #885-1062) and a little knowledge of the diskette's file structure.

The technique I will describe shortens the file DIRECT.SYS to only 2 sectors which will allow a maximum of 22 files to be stored on the diskette. If you perform this modification and you subsequently try to store more than 22 files, HDOS does not panic but merely informs you that it has run out of directory space. You should perform the following steps immediately after INITializing a new diskette (i.e. before SYSGENing or COPYing any files to the new disk). Since the technique is rather tricky, you might want to save a copy of the final disk so that you can make more "mini-directory" disks at a later time using an absolute sector by sector copy program such as DUP (also on HUG utility disk #885-1062).

The directory file DIRECT.SYS is created by INIT.ABS and is normally located on the 18 sectors starting at track 13, sector 0. The only time it will not be located here is if you indicate that one or more of the sectors in this area should be locked out because of some sort of damage to the media. In order to speed up access time, the groups in the directory are normally stored in an interleaved fashion starting at track 13, sector 2. Table 1 shows the normal locations of each of the 18 sectors in the standard DIRECT.SYS file.

A freshly INITialized disk will have only three files on it: RGT.SYS, GRT.SYS, and DIRECT.SYS. The directory entries for these files will normally be located at the end of the 4th directory sector, that is on track 13, sector 7. In order not to disturb these entries, the best way to make a 2-sector directory is to relocate the start of the file DIRECT.SYS to be track 13, sector 6, and the end to be track 13, sector 7. There are three places on the disk where patches must be made to accomplish this: GRT.SYS, DIRECT.SYS, and the boot track.

Directory Sector	Track	Absolute Sector
1-2	13	2-3
3-4	13	6-7
5-6	13	0-1
7-8	13	4-5
9-10	13	8-9
11-12	14	2-3
13-14	14	6-7
15-16	14	0-1
17-18	14	4-5

Table 1. Absolute track and sector locations of the 18 sectors which comprise the file DIRECT.SYS on a diskette which is initialized via INIT.

The easiest patch to make is the boot track patch. As I mentioned before, sectors 0-8 on track 0 are reserved for a 9 sector bootstrap program. Sector 9 on track 0 is reserved for storing various pieces of system information about the disk including the disk name and volume (see Swayne, 1982c). The byte definitions for this sector are contained in the file LABDEF.ACM on the "Software Tools" disk distributed with HDOS 2. The fourth byte of this sector (LAB.DIS) is the pointer to the directory sector. This byte is needed by the bootstrap program to find the directory and subsequently find the file HDOS.SYS. This byte is normally hexadecimal 84 (decimal 132), thus it normally points to track 13, sector 2. Since we want the new directory entry to be track 13, sector 6, we must change this byte to hexadecimal 88 (decimal 136). This is easily done using the HUG utility DUMP. Using DUMP, look at track 0, sector 9 of the disk to be modified. The fourth byte should be changed from 84 to 88.

The next file to be patched is DIRECT.SYS. Since the bootstrap must be able to scan through the directory before the normal HDOS file handling routines are available, a simple chained structure is incorporated in the file itself to logically connect the directory sectors during boot-up. Each 512 byte group of the directory can store 22 file entries at 23 bytes each for a total of 506 bytes. The 507th byte is always 0; the 508th byte is the number of bytes in each file entry (normally hex 17); the next two bytes contain the address of the current group; the last two bytes contain the address of the next group (or zero to indicate the end of the directory).

The first patch we must make to DIRECT.SYS is to change these last two bytes to 0 in what is to be the last group of our new directory, in this case track 13, sector 7. Using DUMP to look at this sector, you will see that the next to last byte will be hexadecimal 82 (indicating that the next group in the directory is at track 13, sector 0). By changing this byte to 0, you will inform the bootstrap that this is now the end of the directory.

If any of the above patches are done incorrectly, the disk will probably not be bootable. The remaining patches are to tell the normal HDOS file handling routines the information it needs about the file DIRECT.SYS. If any of these are performed incorrectly, you will probably get the "Disk Structure Corrupt" message when you try to boot or mount the disk.

The next patch is also made to DIRECT.SYS. In this patch you must change the entries for the file "DIRECT.SYS" itself, in particular you must change the values for the starting and ending groups. Refer to Table 2 which is taken from the file DIRDEF.ACM (on the "Software Tools" disk that comes with the HDOS distribution package) and shows the layout of each 23 byte directory entry.

```
        ORG     0

DIR.NAM DS      8       File Name
DIR.EXT DS      3       Extension
DIR.PRO DS      1       Project
DIR.VER DS      1       Version

DIR.CLU DS      1       Cluster Factor
DIR.FLG DS      1       Flags
        DS      1       (reserved)
DIR.FGN DS      1       First Group Number
DIR.LGN DS      1       Last Group Number
DIR.LSI DS      1       Last Sector Index
DIR.CRD DS      2       Creation Date
DIR.ALD DS      2       Last Alteration Date
```

Table 2. The layout of each directory file entry.

The 16th and 17th byte of each directory entry is the first group number and last group number for that file, respectively. Since these are group numbers, we must multiply by the number of sectors per group (two) to get the track and sector. For the file DIRECT.SYS, the first group number is normally hexadecimal 42. Multiplying by 2 we get hexadecimal 84 which is decimal 132, or track 13, sector 2, just as we showed in Table 1. In the new "mini" DIRECT.SYS, the first group and last group will both be track 13, sector 6 (hexadecimal 88), thus the new values for entries DIR.FGN and DIR.LGN will be 88/2 = 44.

Using DUMP you can now modify the first and last group numbers for the file DIRECT.SYS to both be 44. These should normally be found in bytes DC and DD (hexadecimal) of track 13, sector 7.

The last patch which has to be made before the mini-directory is in place is a patch to the file GRT.SYS. This file contains the group reservation table which is nothing more than a series of pointers which tell HDOS how to reconstruct the files on the diskette. To see how a file can be strung out over many portions of a disk, you may examine a file's structure as follows. Using DUMP and the information in Table 2, locate the file's first and last group numbers in the file DIRECT.SYS, then DUMP the single sector file GRT.SYS (normally on track 14, sector 8) onto the screen. Beginning with the first group number, look into the GRT at the corresponding entry, for example if the first group number is hex 42 (as is normally the case for the file DIRECT.SYS), then look at the byte at location 42 in the GRT. The entry at this location will be the number of the next group in the file. If you look at the GRT entry corresponding to this number you will find the next group number, and so on. When you find an entry in the GRT which is zero, you know you are at the end of the chain. The cell containing this terminating zero should correspond to the last group number as found in the directory. If you reconstruct the sequence of groups for the file DIRECT.SYS in this manner, you will normally obtain the following: 42, 44, 41, 43, 45, 47, 49, 46, 48. If you convert these 9 numbers to decimal track and sector locations, you will get the values in Table 1.

Now with this information you can easily make the patch to the GRT. The new file DIRECT.SYS will start and end at group number 44 (hexadecimal). This means that we don't have to chain any groups together, we simply need to indicate that group 44 is the last group. We do this by using DUMP to place a zero in byte 44 in the GRT. Note that when you make changes to the GRT table, the disk being changed should NOT be MOUNTed since when it is later DISMOUNTed or RESET, HDOS will copy back the version of GRT which it maintains in RAM. You can do the patch using DUMP since DUMP allows you to change the disks at random, just be sure and put all disks back where they were before Control-C'ing out of DUMP.

You should now be able to mount this diskette and find that the file DIRECT.SYS is only 2 sectors long. If you get the "corrupt" message you did something wrong. Try and find your mistake using DUMP. If you have made no mistakes, then you now have a diskette with 284 free sectors.

Application and Turnkey Systems

Making the changes outlined so far is about as far as you can go and still retain all of the features of HDOS. Very often however, one needs to set up an application disk which will only be used to execute some pre-specified set of application programs. On most such disks you can delete the file PIP.ABS. (For help deleting "locked" files, see "Recovering a Deleted File", Harton, 1981, "Recovering Deleted Files In HDOS and CP/M", Swayne, 1982b, or "PATCH Mysteries Revealed", Swayne, 1982a.) After you have deleted PIP, you will no longer be able to use the commands HELP, TYPE, LIST, DELETE, RENAME, CAT, DIR, IND, or INDEX since all of these require PIP to perform their tasks. If you try to use one of these commands, HDOS will simply inform you that it needs PIP to perform the command. If you delete PIP you will have recovered 20 sectors, bringing the total to 304 free.

Going one step further, you may want to set up a "turnkey" system such that your application program is automatically executed on boot-up and the disks are automatically dismounted upon program termination. My ADVENTURE disk is a good example of such a case. HDOS has a built in feature which automatically scans the directory for a file called PROLOGUE.SYS upon booting up. If such a file is found, execution is passed to it as soon as the normal HDOS system files are loaded. If no such file is present, control is passed to SYSCMD.SYS and the user is prompted for a command in the usual manner. This feature allows you to set up such a turnkey system by renaming your program to "PROLOGUE.SYS". In a turnkey system, you won't normally need to have the file SYSCMD.SYS since all interaction with the user should be handled by your program. You could simply delete SYSCMD.SYS, but nasty things will happen if your program tries to return to HDOS via the .EXIT SCALL. A better way is to create a new version of SYSCMD.SYS which simply dismounts the system disks and returns to boot level. Listing 1 shows such a program. When you assemble this program and rename it to SYSCMD.SYS, then rename your application program to PROLOGUE.SYS, you will have a turnkey system which will run only your specified program and will dismount the disks upon program termination.

Going Further

One rather sophisticated trick which the ambitious programmer might want to try is to recover some of the sectors on the boot track (track 0). I mentioned previously that the first 9 sectors on track 0 are reserved for the bootstrap program. These are "locked out" to HDOS by flagging them as unusable in the Group Reservation Table (GRT.SYS). Much of this space is not needed, in fact I have written a bootstrap program which resides in only 2 sectors. I did this by disassembling the original 9 sector version, eliminating unneeded code, reassembling, and storing the code on track 0. A surprisingly large amount of the original code can be eliminated, for instance: 1) most of the first two sectors are reserved for use on DK: type devices where the basic disk access software is not in the H-17 ROM; 2) much of the code is for accessing the older type cassette serial interface (H-8-5); 3) part of the code is used to determine the baud setting (which I always leave at 9600); 4) a big piece of the code is used for computing sector checksums.

If you do create a custom bootstrap as I did, you will have up to 7 more sectors free on track 0. You might think that you could simply "unlock" these sectors in GRT.SYS to make them available for general use by HDOS. Unfortunately the boot track is initialized with a volume number of zero which will be different from the volume number on the other tracks (it must be between 1 and 255). This is done so that the bootstrap program itself can initially be read in, however, it means that if you try to read it through the HDOS SCALLs, you will get an error. One way around this is to read and write to this sector using the ROM routines directly as described in "Disk Programming Without HDOS" (Smith, 1982). You can also read track 0 using the undocumented "read regardless" call to the SY driver (Swayne, 1982c). There are some nice things that can be done with these extra sectors. Because they are not "readable" as normal HDOS files, they are the perfect place to store passwords, deciphering keys for encoding files, and any other sensitive information.

This is about as far as you can normally go toward reducing the HDOS overhead on your system disks. If you need even more space, about the only thing you can do is run without HDOS altogether. This involves calling the H-17 ROM disk driver directly to do reading and writing, but it entirely eliminates all HDOS system files from the disk. It also requires the user to provide the means to maintain a file directory as well as any other services previously provided by HDOS. This technique is discussed in Smith (1982).

Conclusion

In this article, I have attempted to summarize some of the "tricks" which can be used to reduce the amount of disk space which must be set aside for HDOS system use. As a minimum, most users can easily retrieve 30 disk sectors for their own use, however, one can go further, even to the point of eliminating HDOS altogether! The articles I have cited represent some of the best technical articles I've seen on HDOS and I urge interested readers to refer to them as well as the excellent Heath manuals for more in-depth treatment of some of these topics.

Articles Cited

Cohn, C. E. Summer 1983. "Squeeze More Disk Space Out of HDOS", Sextant, Issue #6.

Dallas, A., Lamm, D., and Jorgenson, T. September 1981. "The HDOS Device Driver Programmer's Guide", REMark, Issue #20.

Friedman, H. Spring 1983. "Understanding HDOS, Parts 1, 2, and 3", Sextant, Issue #5.

Harton, D. August 1981. "Recovering a Deleted File", REMark, Issue #19.

Jorgenson, T. July 1981. "Dissecting the HDOS Diskette", Microcomputing, Vol. 5, No. 7.

Pinkston, W. June 1983. "Out In The Boonies With a Single Drive H/Z-89", REMark, Issue #41.

Smith, R. E. Spring 1982. "Disk Programming Without HDOS", Sextant, Issue #1.

Swayne, P. August 1981a. "Losing Weight with HDOS 2.0", REMark, Issue #19.

Swayne, P. December 1981b. "A Tiny SY.DVD", REMark, Issue #23.

Swayne, P. May 1982a. "PATCH Mysteries Revealed", REMark, Issue #28.

Swayne, P. October 1982b. "Recovering Deleted Files In HDOS and CP/M", REMark, Issue #33.

Swayne, P. December 1982c. "What's In A Name?", REMark, Issue #35.

```
***     TINYCMD - Tiny replacement for SYSCMD.SYS
*
*       Purpose: To deny the user access to HDOS
*       system commands in "turnkey" applications.
*
*       G. F. Roberts               9/25/83
*
*       This piece of code may be used to replace
*       SYSCMD.SYS on version 2 of HDOS.  It auto-
*       matically dismounts all disks when any user
*       program tries to return to HDOS command level
*       via an EXIT SCALL. To install it first delete
*       the old SYSCMD.SYS, then assemble this code and
*       rename it to SYSCMD.SYS. The XTEXT files
*       can be found on the HDOS distribution disks.
*
        XTEXT   HOSEQU          HDOS equates
        XTEXT   ASCII           ASCII equivalences
        XTEXT   HOSDEF          HDOS definitions
        XTEXT   TYPTX           Text typing routine
        XTEXT   OVLDEF          Overlay definitions

        ORG     USERFWA         ORG at start of user RAM

        LON     C               List all XTEXT code
*
*       Entry point
*
START   CALL    $DOS            Dismount all disks
        JMP     ROMBOOT         and jump to bootup

        XTEXT   DOS             Dismount disks routine
        XTEXT   RCHAR           Read character routine

        END     START           End of program.
```

Listing 1.

C and **PASCAL**

C86 . . . . by COMPUTER INNOVATIONS. The ultimate for UNIX compatibility, portability and additional features to exploit the power of your Heath/Zenith computer system. Free update for Optimizing**C86. Fastest execution time in Byte Benchmark. Ver. 1.33. Specify either CP/M86 or ZDOS. $395 (or write for discount)

BDS C . . . by Leor Zolman. Fast and efficient. Quick, easy compile and link. Very popular in CP/M database community. Healthy subset of K&R de facto standard with some additional features. Ver. 1.5 CP/M 80/85. $150 (or write for discount)

C/80 and MATH PACK . . . by THE SOFTWARE TOOLWORKS. Very popular in Heath/Zenith community. Fast and efficient. Reasonable subset of K&R de facto standard. Ver. 3, floats and longs. Specify HDOS or CP/M 80/85.
Both packages for $79.90 (or write for discount)

LUCIDATA PASCAL and P-CODE TRANSLATE . . . by POLYBYTES. Sophisticated, powerful, p-code convenience as well as native-code speed and efficiency. Can be assembled and linked using Microsoft's M80/L80. Specify HDOS or CP/M 80/85.
Both Packages for $80 (or write for discount)

All of the above compilers consistently rate high in published reviews and benchmark evaluations. They include superior documentation and have demonstrated outstanding user support. They are all very mature implementations and are royalty free.

The following EIGENWARE software packages are customized for Heath/Zenith computer systems:

GETTING STARTED IN C . . . structured library of indexed examples and utilities for learning the C programming language. Includes macros and functions for H/Z features. Specify BDS or C/80 for CP/M 80/85; CP/M 86 or ZDOS for C86. $24.95

DISCOVERING PASCAL . . . programming tutorial and source code library for the financial calculator. Specify IBM for ZDOS or LUCIDATA for CP/M 80/85. $24.95

FINANCE . . . "what-if" on-screen analysis of time-valued financial transactions. Interactive selection of parameters. Solve for any one of the financial variables. Print amortization schedules. Specify CP/M 80/85 or ZDOS. $24.95

ARCHIVE . . . avoid disk clutter, save space by putting many ASCII (text) files in a single file. A very neat way to organize source code libraries and backup files of several different packages on a single disk. CP/M 80/85. $19.95

EIGENWARE TECHNOLOGIES
13090 LA VISTA DRIVE
SARATOGA, CA 95070

Specify disk format: All 5" Zenith disk formats are supported except 96 tpi. Send check or money order. Include $2.50 for shipping and handling; 6% tax for California residents.

Simply Graph It!

Crawford MacKeand
115 South Spring Valley Rd.
Greenville, DE 19807

It is all very good to have written a program that does a neat calculation, but many of the most interesting routines are still to come when the math is all completed. How do you tell the user all about the good things that he just calculated? And what about the possibility that tables of numbers are not the best way to present the data? Maybe there are graphical methods of doing it. I personally feel more comfortable with numbers than I do with pictures, especially when I sit at a keyboard. However, there are times when the old saying that a picture is worth a thousand words comes true, even from my biased point of view!

Those of you who have tried to write a scientific or an engineering program of any sort will have gone through the experience of setting up the equations and finding with delight that the answer comes out of the sausage machine, after debugging of course, just as it was supposed to. And then you realize ...... but that's just one answer, and to do any real good with it I am going to need a raft of them. So you sit down and soon you have a nifty table of all that good information, and you have learned something about formatting and the sun is shining and everything is good.

```
100 DIM B(20,9): B$="Length, ft.": CS$=CHR$(27)+"E"
110 OPEN"I",#1,"SY1:MDATA.BAS"
120 FOR N = 1 TO 19
130 INPUT#1, B(N,0),B(N,1),B(N,2),B(N,3),B(N,4),B(N,5)
140 NEXT N
150 N=N-1
5000 REM ************* Graphic presentation sub-routine (ver2)**************
5010 CLOSE: PRINT: PRINT
5020 PRINT"Any calculated variable can be plotted against the ranged variable."
5030 PRINT"             Input Resistance (1)  Input Reactance (2)
5040 PRINT"             Input SWR        (3)
5050 PRINT"             Nominal Atten.   (4)  Actual Atten.   (5)
5060 INPUT " Enter code for required plot.   If none required, enter <0> "; H%
5070 PRINT: NM=N: IF H%=0 THEN 5300
5080 INPUT " Enter plot max. and min. values of selected variable "; UX, UN
5090 IF UX <= UN THEN PRINT " Max. must be greater than min.!!": GOTO 5080
5100 REM Graphic construction starts.
5110 UD=(UX-UN)/4: FOR I3 = 0 TO 4: U(I3) = INT(100*(UN+I3*UD))/100: NEXT I3
5120 PRINT CS$: PRINT TAB(26); "VERTICAL CO-ORDINATE  ";B$
5130 W% = ABS(N): IF (N+1)/2=INT((N+1)/2) THEN PRINT" ";B(W%,0),; ELSE PRINT,;
5140 G$ = CHR$(42): J = INT(15 + 60*(B(W%,H%)-UN)/(UD*4))
5150 IF J>75 THEN J=76: G$=CHR$(62): GOTO 5170
5160 IF (B(W%,H%)-UN)<0 THEN J=0: G$=CHR$(60)
5170 PRINT TAB(J); G$: N=N-1: IF N>0 THEN 5130
5180 PRINT TAB(12);"---^--------------^--------------^--------------^--------------^----"
5190 S%= -LEN(STR$(U(1)))/2: PRINT TAB(S%+15); U(0); TAB(S%+30); U(1);@
                TAB(S%+45); U(2); TAB(S%+60); U(3); TAB(S%+75); U(4)
5200 PRINT: ON H% GOTO 5210,5220,5230,5240,5250
5210 PRINT TAB(32);"INPUT RESISTANCE ohms."   ;:GOTO 5260
5220 PRINT TAB(32);"INPUT REACTANCE ohms."    ;:GOTO 5260
5230 PRINT TAB(32);"INPUT STANDING WAVE RATIO";:GOTO 5260
5240 PRINT TAB(32);"NOMINAL ATTENUATION dB."  ;:GOTO 5260
5250 PRINT TAB(32);"ACTUAL ATTENUATION dB."   ;
5260 PRINT TAB(60);"More plots <Y><N>";
5270 INPUT C$: N=NM: IF C$="y" OR C$="Y" THEN PRINT CS$: GOTO 5020
5300 END
```

Figure 1

Well, maybe. What do you do with those results? Five times out of ten, or even more, you promptly get a piece of paper and make a rough graph to see what they look like. That is the point that I had arrived at when I decided: if the H19 can tabulate it, then it can graph it, too. Seemed like a simple task (it is, too), but the canned routines that I found in any references at hand were complicated. Unduly complicated I thought, for the construction of a simple quick and dirty graph ..... and long. Look, I just wanted a little subroutine to tack on the end of a program that was already getting too close to the limits of 64K in the H8.

The result of the next fit of headscratching was a reasonably short routine, which did some of the things I wanted it to do, in a very simple minded way. It was grand, if you could wait! But since it is easy to explain, lets start with it first. The root of the idea was to set up a loop something like this:

```
1000  READ X
1010  A=A+1 : IF A<X THEN PRINT " ";: GOTO 1010
1020  PRINT "*"
1030  A=0: GOTO 1000
```

The operation was very simple. You read the value of the variable X and then go to the first line of the graph. It checks to see whether the value of A is less than that of your variable X, and if not, it adds one to A and tries again. It also steps the cursor one place across the screen by printing a space with the PRINT " ".

When eventually A reaches the value of X, the IF statement says go to the next line (1020) and print an * or whatever you want to use to denote the points of the line. It also says it is done with this line of the graph (there's no ; after the PRINT statement this time) and then sets the counter A back to zero and goes round again. Great, as long as the variable X knows that it is only allowed to vary between 0 and 80, and as long as you don't want any labels on the axes, and as long as you don't want the axes marked in any way with a line or some divisions, or in fact any of the good things which will make your graph useful. And also as long as you don't care that it is as slow as ditchwater.

But the principle is there. Obviously moving the cursor like that is not an efficient use of the system, and equally obviously there has to be some way of putting in the labels that are missing, and we most certainly have to provide a method of handling all of the different ranges of variable that the routine may encounter. And finally, the READ statement is not going to be an adequate way to introduce all the information from an actual program situation. So there are a number of factors to be dealt with. But in the end, it proved to be quite tractable and resulted in the short program seen in figure 1. The meat is in lines 5110 to 5190, and the rest just selects the variable we want to display and sees that the graph has the proper title blocks.

The first essential is to be able to read the data conveniently from an external source. The program I was working with was easily induced to put its data into an array format from which the graphic routine could recover the data. The chosen format for a variable is X = B(N,M) where X is the dependent variable to be plotted on the horizontal axis and N is a number which tells you where you are in the list of the key variables. M tells you which variable it is, if there is a choice of more than one dependent variable to plot, and M=0 specifies the key variable, as you can see in figure 2. This sounds quite formidable but the examples below using the sample data file in figure 2 will clarify the usage. There is one restriction on the key variable. It must change regularly. That is to say, it can go 7, 9, 11, 13, 15, 17, 19, 21 etc., or 6.3, 6.6, 7.2, 7.5 etc. But not 21, 23.2, 26.1, 28.7, 30.1 etc. Since it is most likely that you will have generated its various values with a FOR...NEXT loop, which will automatically satisfy this condition, it is unlikely to present a problem.

The next change you can see is in line 5170. Instead of pushing the cursor gradually across the screen until it has arrived at the right place, now I calculate the position in line 5140 (variable J defines the place) and I go straight to the spot with a TAB. Much easier. But let's go through the program on a blow by blow basis. Lines 5020 through 5060 find out which variable we want to graph, and H% retains that information. Next we are asked to choose the maximum and minimum limits of the horizontal axis. At first I was going to make this automatic, but soon realized that a manual entry was easier and more useful. This is especially true if you want to compare variables from several runs on a common basis. At line 5110 the intermediate values for the labels on the horizontal axis are calculated for the half and quarter points. (There is only room for five on an 80 column screen if it is not to be crowded, and it looks about right this way.) The next line does a screen clear and inserts the name for the vertical axis, and line 5130 inserts the values for the vertical axis. It does this for each alternate line, using the tab comma to set the beginning of the graphic space at column 16, whether or not it is writing a number label. Forget the ABS for the moment and note that W% = N (most often anyway!). Line 5140 is where the work is done. Here the variable is called by its name B(W%,M%) which is B(N,M), except that I am using M% instead of M to save some space. Then the number is scaled in 60ths. That is to say that if I have chosen the range for the horizontal scale to be 50 to 100, and the value of B(N,M) is 75, I will get the answer (30). The point (75) is half way between 50 and 100, and the point (30) is half way between 1 and 60. Then add 15 to get into my chosen graph space which is from 16 to 75, and you have the TAB argument. If the value were 100, I would get 60, and if it were 50, I would get 0. Adding 15 again, we arrive at the required TAB position for the printed *. (The initial TAB of 15 leaves space for the vertical numbers.) I also specify the character to print the graph line as the string G$, and ASCII 42 is the * that I chose.

One of the problem areas is handled by the next two lines. If you elect to set the graph's upper and lower bounds manually, then you will obviously have to deal with the situation where the line runs off one side or other of the graph. It is also very desirable to show that it did so, and in which direction. If J is greater than 75 then, I show a > in the line 76 position, the > being selected by the new value of CHR$(62) for G$. If on the other hand the line 5140 calculation wants to TAB off the graph in the left hand direction, then I show a < in the zero (column 16) position. Line 5170 prints the *, the <, or the >, decrements the row counter by one, and goes around again if there are any rows left to print. The ABS in line 5130 comes in at this point, where in some implementations the routine would slide a negative N into the B(N,M) array bringing the whole thing to a shuddering halt. The ABS function ensures that the array subscript cannot be negative and it has stayed there for safety!

Then finally, 5180 and 5190 print the horizontal axis, the interval marks and associated values, and the ON GOTO selects the correct name for this axis using the selected value of H%. One last question, "Do you want to do it again?", and it's all over. Oh yes, a final note. NM in lines 5070 and 5270 saves the original value of N for any reruns you do.

As I mentioned above, the program source was induced to put its data into an array from which the graphic routine could read the data, but to test the program, I wanted to take the information in from a data file on the disc. So now let's look at the data in the data file in figure 2.

```
  M  =   Ø    1           2             3          4         5

N=1     1Ø   27.8324   11.9Ø9          1.97      .Ø7Ø71    .Ø8811
N=2     15   31.Ø35    17.854          1.96      .1Ø6Ø6    .13193
N=3     2Ø   35.875    23.671          1.95      .14142    .1756
..4     25   42.9Ø2    28.963          1.94      .17677    .21912
..5     3Ø   52.7594   32.77           1.93      .21213    .26249
...     35   65.7331   33.139          1.92      .24748    .3Ø572
...     4Ø   8Ø.4443   27.Ø37          1.9       .28284    .34881
        45   92.Ø478   12.3Ø9          1.89      .31819    .39175
        5Ø   93.8662   -7.771          1.88      .35355    .43457
        55   84.6879  -24.697          1.87      .3889     .47725
        6Ø   7Ø.4399  -33.211          1.86      .42426    .5198
        65   56.9593  -34.4Ø1          1.85      .45961    .56222
        7Ø   46.468   -31.397          1.84      .49497    .6Ø452
        75   38.9766  -26.5Ø2          1.83      .53Ø33    .64669
        8Ø   33.8742  -2Ø.923          1.82      .56568    .68875
        85   3Ø.5717  -15.189          1.81      .6Ø1Ø4    .73Ø68
...     9Ø   28.6488   -9.5            1.8       .63639    .7725
..18    95   27.8497   -3.9Ø6          1.8       .67175    .81421
N=19   1ØØ   28.Ø533    1.6            1.79      .7Ø71     .8558
```

Figure 2

(For anyone who is interested, the data is real, and comes from a transmission line calculation giving length, Rin, Xin, SWR, and attenuation in two flavors.)

The data was put on the disc just as you see it in figure 2, but without the line and column number notations, and of course the spacing has been tidied up to make it pretty! In the test format, the program reads the data from DATA.BAS into the arrays that are set up at line 130, using the OPEN instruction for file #1

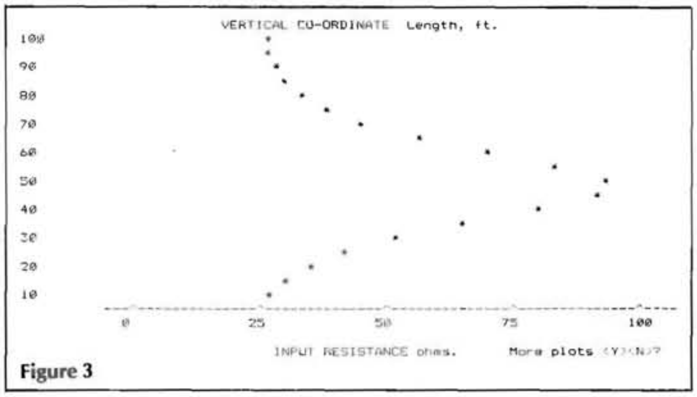


Figure 3

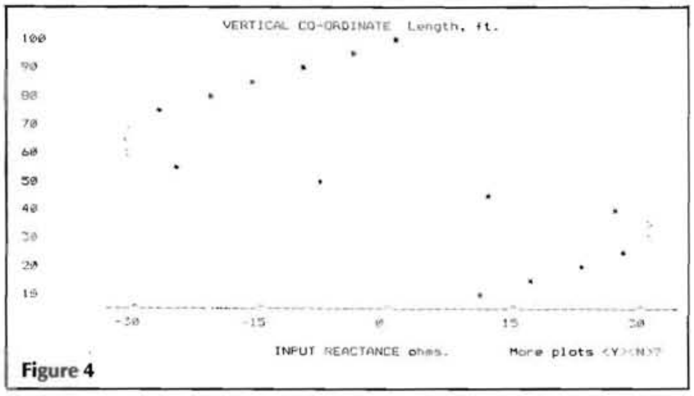


Figure 4

which starts the system off at line 110. A minor point in using the display is the number of key variables which can be handled. If the graph is to be displayed to the best advantage on the 25 line H/Z19 or similar screen, then a count of 18 will just fill it and allow space for titles etc. This is also approximately square. If a longer graph is required, then it is necessary to print it as it is generated, in which case there is no real limit other than string storage space.

In figures 3 and 4 you can see what the resulting graphs look like. Adequate for most in-work calculations, but obviously not a graphics package quality. That's it. Twelve lines of BASIC and you have a simple effective graph generator. And with a screen dump or an echo, you can print it in hard copy. I use a screen dump as the most effective method for this sort of quick graphic, but a small change to the code would also enable you to print directly from a BASIC program. I hope you find it as useful as I have, and the next stage is the simultaneous printing of two variables. But there is always a next stage just in sight down the pike, and a bunch of other projects clamoring for attention. So that will just have to wait!

Vectored from 8

support the use of PeachText with a terminal. I have tried every combination of Baud, Handshake, Stop Bits, etc.

Can someone out there help me?!

Also, here are a pair of Z-DOS assembler programs for turning on and off the Interlace Mode on an H/Z-100. I am using this with an Amdek Amber monitor and really like the display quality much better than the non-interlaced mode. I don't know if this will work on other brands of monitors. To use this program, write it up using Edlin, then use MASM and LINK to assemble it. Then just type the "Interlac" command. I hope these programs are helpful to someone else.

```
;---INTERLAC--- puts the screen in the interlace mode
        .XLIST
INCLUDE DEFASCII.ASM
INCLUDE DEFMS.ASM
        .LIST
STKSEG  SEGMENT STACK
        DB      100H DUP(?)
STKSEG  ENDS

DATASEG SEGMENT
RTADDR  DD      0
MESG    DB      '------THE SCREEN IS NOW IN THE INTERLACE MODE------',
        DB      CC_CR,CC_LF,
        DB        'there are dots interlaced between the normal',
        DB      'dot pattern',CC_CR,CC_LF,
        DB      'you may notice a slight shimmering or fuzziness',
        DB      CC_CR,CC_LF
        DB      'if this is objectionable you may return to the normal',
        DB      'mode by',CC_CR,CC_LF,
        DB      'typing -   KILLINTR',
        DB      CC_CR,CC_LF,'$'

DATASEG ENDS

PGMSEG  SEGMENT
ASSUME  CS:PGMSEG,SS:STKSEG,DS:DATASEG,ES:NOTHING

START:
        MOV     AX,DATASEG
        MOV     DS,AX
        MOV     WORD PTR RTADDR+2,ES    ;save program headder address
        MOV     DX,OFFSET MESG          ;get message address

        MOV     AL,8
        OUT     0DCH,AL
        MOV     AL,13
        OUT     0DDH,AL

        MOV     AH,DOSF_OUTSTR          ;get print string function code
        INT     DOSI_FUNC               ;PRINT STRING
        JMP     RTADDR                  ;terminate program
PGMSEG  ENDS

        END     START
```

```
;---KILLINTR---returns the screen to the normal mode

        .XLIST
INCLUDE DEFASCII.ASM
INCLUDE DEFMS.ASM
        .LIST
STKSEG  SEGMENT STACK
        DB      100H DUP(?)
STKSEG  ENDS

DATASEG SEGMENT
RTADDR  DD      0
MESG    DB      '------THE SCREEN IS NOW IN THE NORMAL MODE------',
        DB      CC_CR,CC_LF,
        DB        'to put it back in the interlaced mode',
        DB      CC_CR,CC_LF,
        DB      'type-   INTERLAC','$'
DATASEG ENDS

PGMSEG  SEGMENT
ASSUME  CS:PGMSEG,SS:STKSEG,DS:DATASEG,ES:NOTHING

START:
        MOV     AX,DATASEG
        MOV     DS,AX
        MOV     WORD PTR RTADDR+2,ES    ;save program headder address
        MOV     DX,OFFSET MESG          ;get message address

        MOV     AL,8
        OUT     0DCH,AL
        MOV     AL,12
        OUT     0DDH,AL

        MOV     AH,DOSF_OUTSTR          ;get print string function code
        INT     DOSI_FUNC               ;PRINT STRING
        JMP     RTADDR                  ;terminate program
 PGMSEG ENDS

        END     START
```

Stan Gray
Manufacturing Innovations
10210 E. 50th Street
Tulsa, OK 74146

New Info On FT.HUG

Dear HUG,

I would like to correct the information you have on file concerning the FT.HUG club.

Club name: FT.HUG Fort Collins Heath Users' Group
Club Address: 3317 Buckskin Trail, Laporte, CO 80535
Contact: Charles McJilton   Phone: 493-2987
Alternate Contact: Bob Strieby
Phone: 221-3984/482-3896 (work)
Group Size: 30

Support and More

From the Heath/Zenith Compatibility Leaders

H8 PRODUCTS

The Most Extensive Line of Hardware Support for the H8®

- **DG-80/FP8**
  Z80® based CPU including the powerful FP8 monitor — both only $199.00. The acclaimed FP8 monitor package is included with the DG-80 CPU.
- **DG-64D/64K RAM Board**
  Reliable, Low Power, High Capacity Bank-selectable RAM
  Priced from $233.00 (0K) to $299.00 (64K)
- **DG Static 64**
  Fully Static, High Capacity, Bank-selectable RAM. Also can be used as EPROM/PROM board (2716 type EPROMS). Priced from $199.00 (0K) to $499.00 (64K).
- **DG-32D/32K RAM Board**
  Low cost, Dependable RAM for the H8 32K Version Only $179.00.
- **DG-ADP4**
  H17-4 MHz disk adaptor — $19.95

THE SUPER 89

The DG SUPER 89 is a replacement central processor board for the Heath/Zenith 88-89 series of computers. The DG SUPER 89 offers advanced features not available on the standard Heath/Zenith 88-89 such as 4 MHz operation, real-time clock, optional AM9511A arithmetic processor, up to 256K of bank selectable RAM with parity check, and HDOS, CP/M and MP/M compatibility. By incorporating current state-of-the-art technology available for the Z80, the DG SUPER 89 offers the user increased speed and system reliability for years to come. Full compatibility with all Health/Zenith software and hardware products is designed into the DG SUPER 89. Electronic Disk Software included. Priced from $829.00 (128K) to $989.00 (256K).

HEARTBEAT

The DG Heartbeat is a compact computer system designed to be hardware and software compatible with the popular Heath/Zenith Z89/90 computer product line. The Heartbeat offers advanced features not found on the standard Heath/Zenith computer such as 4 MHz operation, real-time clock/calendar, two RS-232 serial ports, five peripheral expansion slots, 128 Kbytes (expandable to 256 Kbytes) parity checked RAM and provisions for an optional AM9511 Arithmetic Processor. Compatible with HDOS, CP/M and MP/M II (Multi-user) operating systems. Electronic Disk Software included. The Heartbeat may be used with most popular video terminals on the market although the Heath/Zenith H/Z19, H/Z29 and ZT-10/11 video terminals are recommended for full Heath/Zenith software compatibility. The Heartbeat cabinet design provides for inclusion of hard and/or floppy disk drives as well as other desired peripheral interfaces and is color-coordinated for use with the Zenith Z29 and ZT-10/11 video terminals. Priced from $1350.00 (Basic Unit).

CP/M®, MP/M and MP/M II® are registered trademarks of Digital Research of Pacific Grove, California.
H88/89®, Z89/90®, H17®, H77®, H/Z 47®, Z67® and H-88-1® are registered trademarks of the Heath Company and Zenith Data Systems.
Z80® and Z80A® are the registered trademarks of Zilog Corporation.

D·G ELECTRONIC DEVELOPMENTS CO.

Ordering Information: Products listed available from DG Electronic Developments Co., 700 South Armstrong, Denison, Tx 75020. Check, Money Order, VISA or MasterCard accepted. Phone orders call (214) 465-7805. Freight prepaid. Allow 3 weeks for personal checks to clear. Texas residents add 5%. Foreign orders add 30%. Prices subject to change without notice.

Meetings: Held in members' homes, first Tuesday of the month
Time: 7:00 p.m.
Bulletin board: Under construction, expected early '84
Newsletter: Mailed monthly to members and exchange clubs
Dues: $6.00 annually

I would like to thank you for the fine job you have done to make REMark a fine source of information for all Heath/Zenith users'.

FT.HUG
%Charles McJilton
3317 Buckskin Trail
Laporte, CO 80535

Patch Needed for Peachtext

Dear HUG,

How about some info for patching Peachtext for the capabilities of more sophisticated dot matrix printers, such as my "OKI" 92?

Bradford C. Meyers
46-146 Kiowai Street, #2612
Kaneohe, HI 96744

HELP!!

Dear HUG,

Having been involved in Ham Radio for many years and having operated RTTY almost exclusively, I have become quite interested in the AMTOR system. I wonder if anyone can help me put my H-89 on this mode? Any help would be appreciated very much.

Gorden Weiler
4843 N. 90th St.
Milwaukee, WI 53225

"My Favorite Subroutines"

Dear HUG,

How about a column entitled "My Favorite Subroutines"? You could limit the size to say, 10 lines or less. With remarks included in the line limit, it wouldn't take up much space in the magazine. It would provide an easy way for readers to input entertaining and educational material with a minimum of journalistic effort. I think it would provide "one place to look" without having to glean through a lot of material, looking for that one idea that you saw somewhere... An ideal way to build a library for future reference!

Just in case you go for this, I am including a two liner to kick it off.

```
10 N=100:PRINT "Pulse BELL without moving text";
20 FOR I=1 TO N:PRINT CHR$(7);:FOR J=1 TO N:PRINT CHR$(0);:
   NEXT J:N=N-3:NEXT I
```

Bob Moskus
2511 Alpine Trail
Huron, OH 44839

ED NOTE) *What do you say HUGgies? If you have any favorite subroutines, let's hear from you.*

Heath/Zenith Users' Group

Index of Advertisers

Software Support	*2*	*Skill Data*	*41*
Software Wizardry	*4*	*Micro Innovations*	*41*
Floppy Disk Services	*9*	*MicroServices*	*46*
Sunflower Software	*19*	*Secured Computer Systems*	*46*
Controlled Data Recording Systems, Inc.	*19,57*	*Husker Systems of Nebraska, Inc.*	*47*
Newline	*27,37*	*Eigenware Technologies*	*62*
Mako Data Products	*36*	*D-G Electronics Developments Co.*	*66*
North Coast Intelligence, Inc.	*36*	*Micro-Systems Software, Inc.*	*68*

Changing your address? Be sure and let us know since the software catalog and REMark are mailed bulk rate and it is not forwarded or returned.

== CUT ALONG THIS LINE ==

HUG MEMBERSHIP RENEWAL FORM

When was the last time you renewed?

Check your ID card for your expiration date.

IS THE INFORMATION ON THE REVERSE SIDE CORRECT? IF NOT, FILL IN BELOW.

Name ________________________________

Address ________________________________

City-State ________________________________

Zip ________________________________

REMEMBER - ENCLOSE CHECK OR MONEY ORDER

CHECK THE APPROPRIATE BOX AND RETURN TO HUG

	NEW MEMBERSHIP RATES	RENEWAL RATES
US DOMESTIC	$20 ☐	$17 ☐
CANADA	$22 ☐	$19 ☐ US FUNDS
INTERNAT'L*	$30 ☐	$24 ☐ US FUNDS

* Membership in France and Belgium is acquired through the local distributor at the prevailing rate.

"They shouldn't have touched my MTERM"

"I only left my keyboard for a few minutes ... when I returned, I found Stamitz from accounting and Miss Sashshay from the secretarial pool fondling my MTERM. Now if I've told them once, I've told them a hundred times ... use my coffee cup. Borrow my key to the employee lounge. Bend my paper clips but, leave my MTERM alone!! Did they listen? Nooooo! Well, I guess I lost my head. Both Stami and Sashshay are doing fine. They should be released from the hospital any day now. For me, it's an entirely different story."
Signed, 0076697

Why MTERM creates such fanatically touchy users is not so hard to understand. MTERM turns the Z-100 into something altogether better. MTERM lets your Z-100 work a lot faster, with more accuracy, efficiency and dependability. MTERM also has a lot of features that users such as 0076697 find positively endearing. For instance, the ability to translate any ASCII character to another, and an easy to read operating guide that makes using MTERM as easy as ... well, bending a paper clip.

So to increase productivity, to increase the speed and accuracy of your Z-100. MTERM Only $79.95! Still the best program for the cost!
P.S. "Get your own MTERM" 0076697.

MICRO-SYSTEMS SOFTWARE, INC.

4301-18 Oak Circle, Boca Raton, Florida 33431, Telephone: (305) 983-3390
Toll Free: 1-800-327-8724

Heath Users' Group

Hilltop Road
Saint Joseph, Michigan 49085

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

Volume 5, Issue 2

POSTMASTER: If undeliverable, please do not return.

885-2049