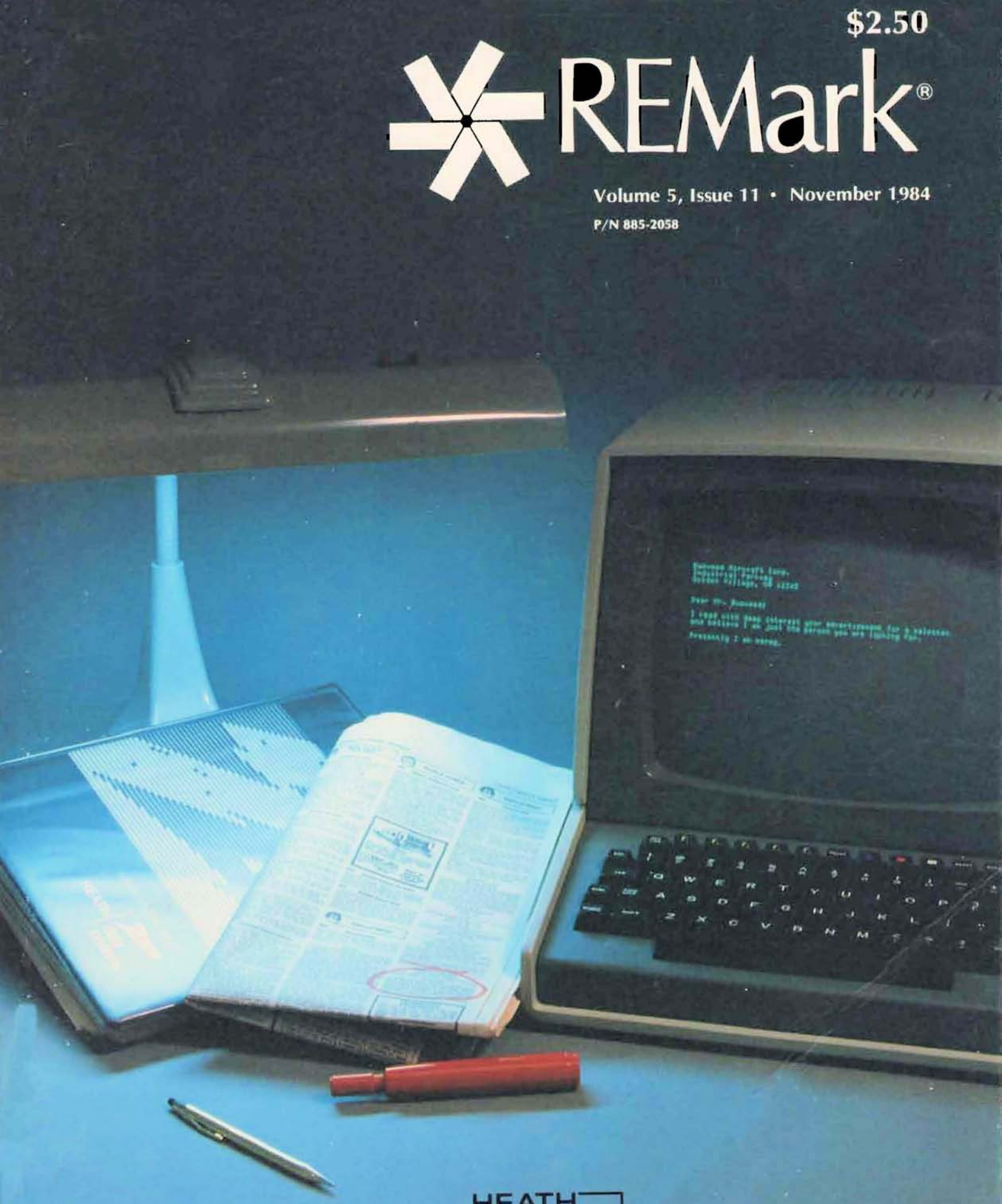
$2.50

REMark®

Volume 5, Issue 11 • November 1984

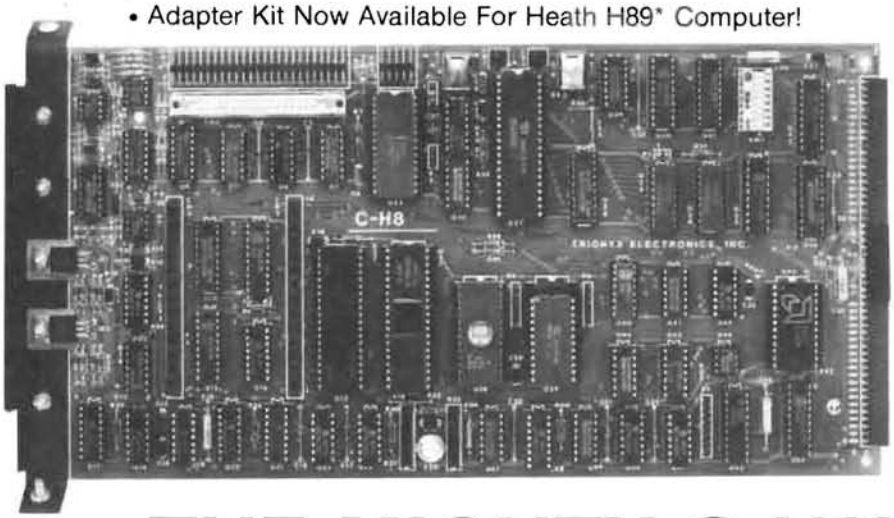
P/N 885-2058



Official magazine for users of HEATH ZENITH computer equipment.

Now! *Revitalize your* H8* COMPUTER!

• Adapter Kit Now Available For Heath H89* Computer!



with **THE MIGHTY C-H8!**

8085 On-Board Microprocessor Runs at 5 MHz

Operates as a Parallel Processor on the H8 Buss

8K Byte On-Board RAM Memory Capability

8K Byte On-Board ROM Memory Capability

Full DMA Operation (Direct Memory Access) On the H8 Buss

Uses 55 Integrated Circuits

2793 WESTERN DIGITAL Floppy Disk Controller

H17 2350 Hard Sector Floppy Disk Controller

INTEL 8219 Buss Controller

THE MOST POWERFUL FLOPPY DISK CONTROLLER IN THE WORLD!

Convert Your H8 Computer Into A State-Of-The-Art Machine!

Priced Without Integrated Circuits -
C-H8 Kit $350.00 C-H8 Assembled $450.00
C-H8 Integrated Circuit Package (52 Integrated Circuits) - $250.00

All Software Programs for Both HDOS* and CP/M** Are Included in the C-H8 Pricing
All Firmware Packages (3 ROMS) Are Also Included in the C-H8 Pricing

DMA CPU Controller Modules (One Required) Are priced Separately:
$35.00 Kit $50.00 Assembled
Specify H8 CPU Board: Heath 8080, Heath Z80, Trionyx Z80 or DG Z80

C-H8 NOVRAMs Are Software Configured from the Operator's Console
C-H8 Operation is Fully Automatic - Reads All H8 Diskette Formats

Controls up to Four 5—inch and up to Four 8—inch Floppy Disk Drives (8 total)
Hard Sector and Soft Sector - 40 Track and 80 Track (5 inch)
Single Side and Double Side - Single Density and Double Density

Plug-In Connectors Provide Multi-Purpose Interface to C-H8 Internal Buss
Use for Hard Disk Adaptor Module (SASI Buss Interface) or
for Additional Memory (64K Bytes) for 8085 Microprocessor

Fully Compatible With All Heath H8 Hardware and Software

*H8 and HDOS are registered trademarks of the Heath Company.
**CP/M is a registered trademark of the Digital Research Corporation.

Check • Money Order • VISA • MASTERCARD • C.O.D.
Phone Orders Welcome (714) 830-2092 Send For Free Brochure

TRIONYX ELECTRONICS, INC.

P.O. BOX 5131, SANTA ANA, CA 92704

Heath/Zenith Users' Group

Staff

Manager ..... Bob Ellerton
(616) 982-3867

Software Engineer ..... Pat Swayne
(616) 982-3463

Bulletin Board and
Software Developer ..... Jim Buszkiewicz
(616) 982-3463

Software Coordinator ..... Nancy Strunk
(616) 982-3838

Secretary ..... Margaret Bacon
(616) 982-3463

REMark

Editor ..... Walt Gillespie
(616) 982-3789

Editorial and Advertising
Assistant ..... Lori Latham
(616) 982-3794

Graphic and Layout
Assistant ..... Greg Martin
(616) 982-3463

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

	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 available at $2.50, plus 10% shipping and handling. 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/Zenith Users' Group
Hilltop Road
St. Joseph, MI 49085

(616) 982-3463

REMark®

Volume 5, Issue 11 • November 1984

on the stack

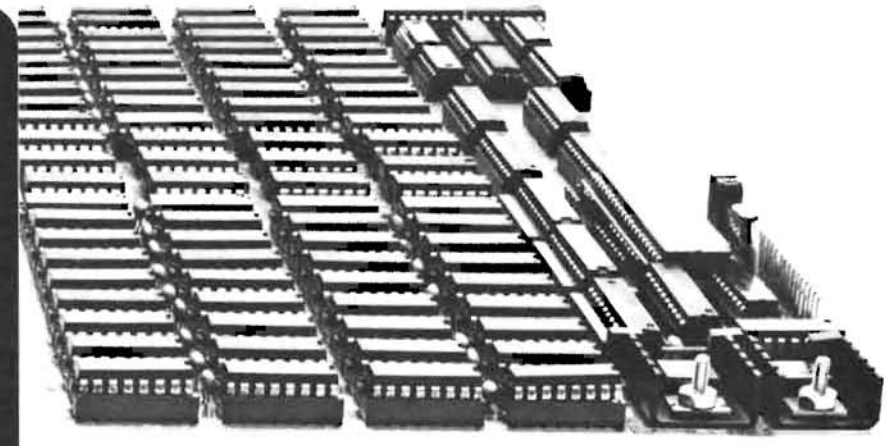
ON THE COVER: This month Harold Ogg presents us with "A Personal Job Resume Generator" (see page 35). Those of you who might be in the market or thinking of joining it just might be able to get "one-up" on the competition.

"Hey UCI, we heard you have memory boards for ZENITH and CompuPro and Lomas and..."

"This board is unique. It is the last RAM board you'll ever need. The board is a very excellent board and has been up and running in our Z110 winchester system under MP/M for some time now. No problems. We are very satisfied with that board."
Henry E. Fale
QUIKDATA, Inc.
Sheboygan, WI

"UCI boards provide better value than other memory boards. They have excellent quality, are easy to install, and have expansion features other boards don't have. We recommend this board to our customers."
Ray Massa
Studio Computers
Birmingham, MI

"For anyone interested in a very good Z100/S100 memory card, they should have their dealer contact UCI Corporation. A nice thing about the board is that is comes already configured for the Z100. UCI deserves excellent marks for support and user assistance."
Chuck Ballinger
Acme Computers
Spokane, WA



EXPANDABLE RAM *FOR S-100 SYSTEMS*

Look at these features...

- 256K up to 2MB of memory *without piggybacking* (9 bits wide)
- 8 bit or 16 bit systems (automatically selected)
- Operates with up to 8 MHz CPUs (with hidden refresh)
- On-board parity generation/detection
- Z-DOS Ram Drive Software
- Zenith System Memory and Ram Drive operational on same board
- Unique Switching Scheme — Zenith System can access entire 2MB memory

Better yet, look at these amazing prices...!

256K	$795 list	1.5MB	$3049 list
512K	$1145 list	2MB	$3995 list
1MB	$2095 list		

Also from UCI...

ZENITH 8 MHZ SPEED UP MODULE .. $49.95 list

Increase your system speed from 5MHz to 8MHz

*Contact These UCI Dealers**

Beaverton, OR
WRAM Corporation
8243 SW Cirrus Dr.
(503) 641-3550

Birmingham, MI
Studio Computers
999 S. Adams Rd.
(313) 645-5365

Detroit, MI
Merit Data Systems
19622 Plymouth
(313) 838-6026

Florence, MA
The Computer Farm
30 N. Maple
(413) 584-2038

Ft. Myers, FL
Computer Tech
3049 McGregor Blvd.
(813) 337-1888

Ft. Walton Beach, FL
Compustation
99 Elgin Pkwy.
(904) 243-1501

Houston, TX
Payload Computer Services
15006 Sun Harbor
(713) 486-0687

Huntsville, AL
Dudley-Wales Ent.
2109 Bob Wallace Ave.
(205) 534-6868

Raleigh, NC
Computopia
1275 Buck Jones Rd.
(919) 469-0267

Sheboygan, WI
QuikData, Inc.
2618 Penn Circle
(414) 452-4172

Spokane, WA
Acme Computers
1727 E. Sprague
(509) 535-4122

St. Charles, MO
First Capital Computer
1106 First Capital
(314) 946-1968

Summerville, SC
The Computer Terminal
908 Bacons Bridge Rd.
(803) 875-4727

Winston-Salem, NC
ITS Computer Center
2070 Beach St.
(919) 727-0290

*Or All Heathkit Electronic Centers

UCI CORPORATION
an affiliate of ACME-CLEVELAND CORPORATION

CALL TOLL FREE
1-800-824-2667

IN OHIO CALL
216-673-5155

948 CHERRY STREET KENT, OHIO 44240

BUGGIN' HUG

Excuse us!

In the September Issue, Page 52, the S & K Technology, Inc. advertisement for Watchword, with a HUGCON special price, was rerun by mistake.

We apologize to any REMark readers who ordered expecting to receive this special discount, and to S & K Technology, Inc. for any inconvenience it has caused them.

The HUG Staff

The ZDOS - DEBUG Difference

Dear HUG,

The other day when I was trying to debug an H-100 assembly language program, I noticed a difference in the way the ZDOS Command Processor and DEBUG handle missing parameters (arguments) in a command line.

According to Appendix P, Page 2, of the ZDOS Reference Manual, Volume II, "The file control block (FCB's) at 5CH and 6CH, in the program header area, are formatted (i.e. converted to upper case and blank filled from the first two parameters entered on the command line.)" That tells me that if my second parameter, for example, was missing (specifying two consecutive commas), I would expect the second FCB (FCB2) to be blank filled. My program that I was debugging has three parameters, of which the second and third are optional.

When I enter the command line with two consecutive commas after the first parameter and included the third parameter as follows:

PRT fname,,lc <return>

The program header area is set up differently for a direct command line call and for a call via DEBUG. Using DEBUG, the program header area gets set up as I thought I understood it to work; namely, the second FCB (FCB2) was blank filled to indicate that no second parameter was entered. However, if the call to my program is made directly to the Command Processor (command line entry), the second FCB is set up with information from the third parameter rather than blank filling it to indicate no second parameter.

It was a rather frustrating experience to fully check out my program using DEBUG, then have the program fail when I called it directly, via the command line entry. Debug code had to be inserted into the program to isolate the problem, since under DEBUG the program worked fine. My program was a .COM file and I don't think that an .EXE file would make any difference, since the program header area is supposed to be set up the same for both.

This certainly is an inconsistency between the two pieces of software, but is this the way it was designed? Or is it an actual "bug?" Has anyone else experienced this problem?

Richard L. Mueller, Ph.D.
11890-65th Ave. N.
Maple Grove, MN 55369

It's Arrived!

Dear HUG,

At long last Dos 2.0 has arrived . . . or has it? I couldn't wait to get home after stopping by the local Heathkit and buying 2.0. Well, here's one of the problems with it. The File Allocation Table has been changed . . . grievously so for those with 96 TPI drives.

```
   Fat ID              1.2   2.0   Comment
   ------------------  ---   ---   ------------------
   48 TPI SS 5.25  9S        FC    ;
   48 TPI DS 5.25  9S        FD    ;notice swap
                                   ;
   96 TPI DS 5.25  8S    FD  FA    ;with 1.2 96 TPI
   96 TPI DS 5.25  9S        F9    ;

{ this table from Zenith after talking to MS }
{                  \ /                       }

   Dos 2.0 FAT ID byte.
   Bits    7  6  5  4  3  2  1  0
   ---------------------------------
           1  1  1  1  1  48 8  DS
 Bits
 0       set if double sided
 1       set if 8 sector
 2       set if 48 TPI
 3-7     always '1'

Fat ID of FB means 96 TPI 8 Sector DS.
```

FORMAT (and everything else) really mess up when trying to access a 96 TPI 8 sector disk. The FAT says it's reading a 48 TPI 9S disk. FORMAT can't/won't format 96 TPI 8 sectored (though it will 9 sectored).

```
ie : A:DEBUG              | this patch lets 2.0 read 96 TPI 8S
     -L CS:100 0 0 10     | disks ... it also renders the disk
     -E300                | unreadable by DOS 1.2.  From 2.0
     0D21:0300    FD.FB   | you can use debug to reverse the
     -E500                | process (F8.FD).
     0D21:0500    FD.FB   |
     -W CS:100 0 0 10     |
     -Q                   |
```

Sure hope Zenith will do something about FORMAT. They say they don't support 96 TPI, but perhaps they'll be nice enough to fix FORMAT so it'll format 96 TPI 8S, as well as 9S. Well, I found my bug (an easy one, I'll admit) now where is the rest of them (hopefully not there!)

By the way, I use TURBO Pascal and a program that called interrupt 25H (absolute disk read) under Dos 1.2 and worked great under 2.0. Anybody got any ideas? Everything just locks up . . . Zenith swears there's no problem and no difference from Dos 1.2 INT 25H, there has to be some difference.

God Bless,

Dale G. Welch
16902 Creekline Dr.
Friendswood, TX 77546

Inside Microsoft Basic

Dear Hug:

The article "Inside Microsoft Basic," by D.D. Dodgen, in the August issue of REMark, is a very thorough study of this interpreter, and I now have a much more complete understanding of the internal workings of the language.

He did overlook one thing, however, the EQV function is documented on Pages 2-11 through 2-13 of the CPM version of the Basic-80 manual, although it is omitted from the list of reserved words and the index.

I have always ignored the EQV and IMP functions because, in my opinion, what little documentation is included in the manual produces more confusion than understanding. Mr. Dodgen's article aroused my curiosity enough to make me take a little time to examine these functions. Other users might be interested in my findings.

A EQV B produces a "true" (non-zero) value if both A and B are true or if both are false, otherwise it produces a "false" (zero). The number returned is the result of taking the two's complement of A and XORing it with B. This operation is commutative, that is A EQV B=B EQV A. The following program lines are equivalent:

```
10 INPUT;A,B:PRINT ,A EQV B
10 INPUT;A,B:PRINT ,-A-1 XOR B
```

For example, the program line:

```
100 IF X<N EQV N<20 THEN END
```

will end the program if X<N and N<20, or if X>N and N>20, it will continue if X>N and N<20, or if X<N and N>20. This could possibly be used to end a program when the variables X and N fell within these limits, although this operator will probably find it's greatest use in more complex expressions.

A IMP B produces a "false" (zero) value under only one condition, if A is true and B is false. The number returned is the result of taking the two's complement of A and ORing it with B. This operation is not commutative. The following program lines are equivalent:

```
10 INPUT;A,B:PRINT ,A IMP B
10 INPUT;A,B:PRINT ,-A-1 OR B
```

The program line:

```
100 IF A$="OR" IMP B$="503" THEN 200 ELSE GOSUB 1000!
```

'will branch to line 1000 only if A$="OR" and B$<>"503." Since 503 is the area code for Oregon, this line could be used in a program that scans a telephone list for incorrect area codes. The fact that the state is Oregon IMPlies that the area code is 503, because all telephones in Oregon have the same area code.

On an entirely different subject, it should be noted that the 'extra' opcodes, listed in James T. Malone's letter in the same issue, are parts of valid Z-80 opcodes with very different functions. Attempting to run a program using these opcodes on a computer, using the Z-80, could result in some very interesting problems.

The "Reference Card for the Z-80," published by Nanos Systems, contains 33 such undocumented opcodes for the Z-80, although many are duplicates. Anyone interested in such novelties should try to get a copy.

I am also glad to see that Heath is now offering the DataProducts printer. I have been using one for two years with excellent results.

Regards,

Paul Berkebile
8 River Road
Amherst, NH 03031

CROSSREF

Dear HUG:

Reference the August REMark article, "Inside Microsoft BASIC (MBASIC)," the author provided some interesting insight into MBASIC. Unfortunately, the listing of the CROSSREF program apparently incurred several errors between the author's computer and the REMark article. Also, although the author stated that some versions of MBASIC have different codes, no mention was made of the fact that the location given as the beginning of any MBASIC program (28761) does not apply to all systems.

For those interested in using CROSSREF on a different system, the following suggestions are offered. Also, I have corrected the errors in the listing and the revised listing is enclosed herewith.

The following program can be used to find the beginning location of MBASIC programs on any system. A trial and error approach will work if there is a sufficiently broad span of memory between n and n1. I used 26000 for n and 30000 for n1.

```
10 FORI=nTOn1
15 IFPEEK(I)=73ANDPEEK(I+1)=0ANDPEEK(I+2)
   =0ANDPEEK(I+3)=0GOTO20ELSE30
20 PRINTI;PEEK(I);CHR$(PEEK(I)),:STOP
30 NEXTI
```

If line 20 does not execute, then try other ranges of numbers above or below the range already tried. When line 20 executes, the (I) location printed minus 99 should equal the location where MBASIC programs begin on your system. Use the following program to test this, using a number equal to the location minus 99 as n and a number equal to the location plus 47 as n1. (Don't use any spaces when entering either program.)

```
10 FORI=nTOn1
20 PRINTI;PEEK(I);CHR$(PEEK(I)),
30 NEXTI
```

The above is the program shown as Program Listing 1 in the REMark article and should result in a printout very much like Example 1 in the article. By adding a line 15, as shown below, that program will randomly test whether the codes in a given version of MBASIC are the same as those shown in the REMark article's 3 Tables. (n and n1 are the same as used in the above program.)

```
10 FORI=nTOn1
15 IF A > B THEN C = INT(D+1):GOTO 20
20 PRINTI;PEEK(I);CHR$(PEEK(I)),
30 NEXTI
```

Line 15 is simply an example. Any executable combination of variables, mathematical symbols and keywords can be used. (Don't use a REM statement.) But, this time use spaces between virtually everything on line 15. Then, you can find the codes for whatever symbols, keywords, etc. you used by finding the location of the spaces (code 32) in the printout.

Now, some comments about CROSSREF. I sincerely hope that REMark never publishes another listing with the REMarks in between statements on a multi-statement line. It made copying the listing without error virtually impossible, especially since some errors were built into the listing. Also, some of the multi-statement lines exceeded the 256-character limit of MBASIC. And finally, it would also have been extremely helpful if the author or the editor had used RENUM to achieve some consistency in line numbering, which would have made it easier to copy by using the AUTOmatic line numbering command.

Rather than attempt to guarantee that I will discuss every problem with the original listing, I would suggest that the enclosed listing be copied.

Near the beginning of the program, the author lists VL=0, following his suggestion that defining variables early saves processing time. Throughout the rest of the program, however, the variable is LV.

Under my system (H8 with Z80) all MBASIC programs begin at location 29596. Wherever they start in a given system, that is the location that should be specified for MP= at the beginning of the listing.

Because of the method of interspersing REMarks in the listing, the author normally (but, not always) uses a colon before the REM. If the next statement is ELSE, remember that no colon should precede it.

In the author's line 65015, the statement V$(1)=V$(1)+" -" should be on a separate line because of the ELSE earlier. If it's not, under some options, line 65150 will show a syntax error because X will equal -2. Also, although I left it in my revision, there seems no purpose for the statement V$(2)="ZZZ."

Under the author's lines 65010 and 65210, some of the statements must be put on separate lines, in order to avoid exceeding the 256 character line limit. More important, however, is the absence in the list of IF statements under 65210 of one referring to the subroutine at the author's line 65180. (IF MM=34 THEN GOSUB 65180) This is the only logical place for a reference to that subroutine. There is no reference to that subroutine in the author's listing.

Under the author's line 65460, delete the -1 from the statement SW=I-1. Then put it back in under line 65480 after SW=I. In the first case, it is not needed and has no effect, but in the latter case, not having the -1 results in an improperly sorted final listing of variables and line references.

Also under line 65460 all references to V(I), V(I+1), and V(0) should be V$(I), V$(I+1), and V$(0).

Finally, under the author's line 65500, an extra closing parenthesis [)] is needed in the IF statement for each of the two VAL(Left$ parts to avoid a syntax error.

Sincerely,

C.F. Mowery, Jr.
Colonel, USAF (Ret.)
406 Van Reed Manor Dr.
Brandon, FL 33511

```
65000 END:REM
*** Program to MERGE with another to find Variables/Line #s
65010 CLEAR 5000:LV=0:MP=29596:ED$=CHR$(27)+"E"
      :DIM V$(99),V(99)
65020 PRINT ED$;"Variable/Line # Cross-Reference Utility"
65030 PRINT "1. List ALL Variables/Line # References":PRINT
65040 PRINT "2. List ALL Variable References Only":PRINT
65050 PRINT "3. List ALL Line # References Only":PRINT
65060 PRINT "4. Search for ONE Variable Only":PRINT:@
      PRINT "5. Search for ONE Line # Only":PRINT
65070 INPUT "Enter the # of your choice: ";
      CH:IF CH<1 OR CH>5 THEN 65020
65080 IF CH>3 THEN LV=1:PRINT:INPUT
      "Enter the Var/Line # to search for: ";@ V$(1)
65090 V$(1)=V$(1)+" -":J=1:V$(2)="ZZZ"
65100 PRINT:INPUT "Do you want output to go to a printer? ";
      LP$:@ LP$=LEFT$(LP$,1):IF LP$="Y" THEN OPEN "O",2,"LP:"
65110 GOSUB 65160:IF LN=65000! THEN 65340 ELSE GOSUB 65170
      :GOTO 65110
65120 X=INSTR(V$(J),"-"):X=X-2:IF AV$=LEFT$(V$(J),X)
      THEN 65130 ELSE RETURN
65130 IF STR$(LN)=RIGHT$(V$(J),LEN(STR$(LN))) THEN F=1
      :RETURN
65140 V$(J)=V$(J)+STR$(LN):J=LV:F=1:PRINT AV$"=":RETURN
65150 MP=MP+1:MM=PEEK(MP):IF MM=34 THEN RETURN ELSE 65150
65160 NL=PEEK(MP)+256*PEEK(MP+1):LN=PEEK(MP+2)+256*PEEK(MP+3)
      :PRINT:@ PRINT "Working on Line #";LN:MP=MP+4:RETURN
65170 FOR MP=MP TO NL-1:MM=PEEK(MP)
65180 IF MM=255 OR MM=15 THEN MP=MP+1
65190 IF MM=132 OR MM=143 OR (MM=58 AND PEEK(MP+1)=143)
      THEN MP=NL-1
65200 IF MM=28 OR MM=11 OR MM=12 THEN MP=MP+2
65210 IF MM=29 THEN MP=MP+4 ELSE IF MM=31 THEN MP=MP+8
65220 IF MM=34 THEN GOSUB 65150
65230 IF MM=14 THEN GOSUB 65320 ELSE IF MM>64 AND MM<123
      THEN GOSUB 65250
65240 NEXT:RETURN
65250 AV$=""
65260 AV$=AV$+CHR$(MM)
65270 MM=PEEK(MP+1):IF (MM>47 AND MM<58)
      OR (MM>64 AND MM<123) OR MM=33 @ OR (MM>34 AND MM<38)
      THEN MP=MP+1:GOTO 65260
65280 IF MM=32 THEN MP=MP+1:GOTO 65270
65290 IF MM=40 OR MM=91 OR MM=123 THEN AV$=AV$+CHR$(MM)
65300 IF CH=3 OR CH=5 THEN RETURN ELSE IF CH=4 THEN J=1
      :GOTO 65120
65310 F=0:FOR J=1TO LV:GOSUB 65120:NEXT:IF F=1
      THEN RETURN ELSE @ V$(J)=AV$+" -":LV=LV+1:GOTO 65140
65320 AV$="":X=PEEK(MP+1)+256*PEEK(MP+2)
      :MP=MP+2:AV$=MID$(STR$(X),2)
65330 IF CH=2 OR CH=4 THEN RETURN ELSE IF CH=5
      THEN 65120 ELSE 65310
65340 PRINT ED$;"Sorting":IF CH>3 THEN LV=1
      :GOTO 65410 ELSE SW=LV-1
65350 X=SW:SW=0:FOR I=1TO X:IF V$(I)>V$(I+1)
      THEN SW=I:V$(0)=V$(I):@
65360 NEXT:IF SW>0 THEN 65350
65370 FOR I=1TO LV:IF VAL(LEFT$(V$(I),1))>0 THEN SW=I-1
65380 NEXT
65390 X=SW:SW=0:FOR I=1TO X:X1=INSTR(V$(I),"-")
      :X2=INSTR(V$(I+1),"-"):@
      IF VAL(LEFT$(V$(I),X1))>VAL(LEFT$(V$(I+1),X2))
      THEN SW=I:V$(0)=V$(I):@ V$(I)=V$(I+1):V$(I+1)=V$(0)
65400 NEXT:IF SW>0 THEN 65390
65410 FOR I=1TO LV:PRINT V$(I):IF LP$="Y"
      THEN PRINT #2,V$(I)
65420 NEXT:CLOSE:END
```

FORTRAN-80 The Best, But . . .

Dear HUG:

This is in response to a letter in the August 1984 issue of REMark.

In my experience, Microsoft FORTRAN-80 (F80) is probably the best implementation of Fortran for 8-bit microcomputers. However, even the best can have a few shortcomings. One, which was mentioned by William E. Crocker in Buggin' HUG (see the August 1984 REMark) is that the CP/M version of F80 writes non-standard text files when outputting to a disk. In particular, F80 terminates each line with a carriage-return (CR) only, instead of the CP/M standard of a carriage-return/line-feed (CR-LF) combination. Thus, if you TYPE the file, the lines overprint on the screen and are unreadable. Many editors, including ED, have problems with these files, and many printers cannot print them properly.

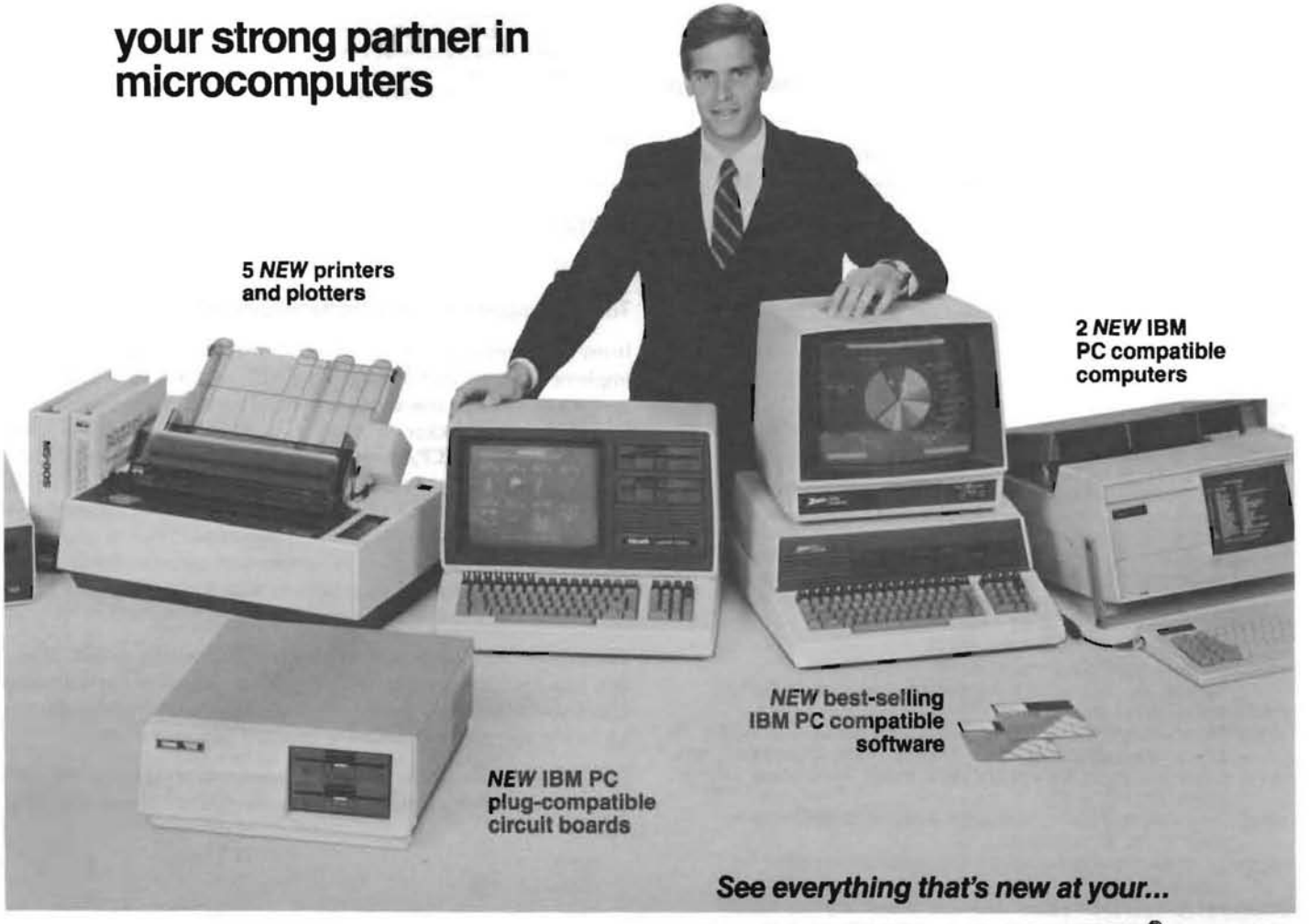
The solution is to force a CR/LF pair into the disk file output. To do this, you need to output an LF as the first character of the following line. If you think about it, this will place the LF immediately after the CR in the file, and thus create a standard CP/M text file.

One procedure, therefore, is to write every line of your disk file with an LF as the first character. Here is an example of how this might be done:

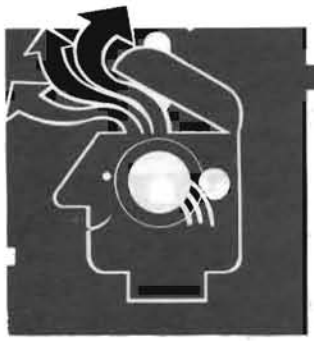
```
BYTE LF
DATA LF / 10 /
  :
  :
VALUE = 12.7
  :
  :
WRITE(8,100) LF, LF, VALUE
```

Vectored to 79 ☞

The one-stop microcomputer shopping center



Heathkit®
Electronic Center
Where you get more by doing



"My Favorite Subroutines"

Dear HUG,

Here is a possible subroutine for your "My Favorite Subroutines" section of REMark. The routine calculates the free space on any CP/M 2.2 disk. It took quite a lot of brain muscle to derive the algorithm from the CP/M manuals and might save some time for assembly language programmers.

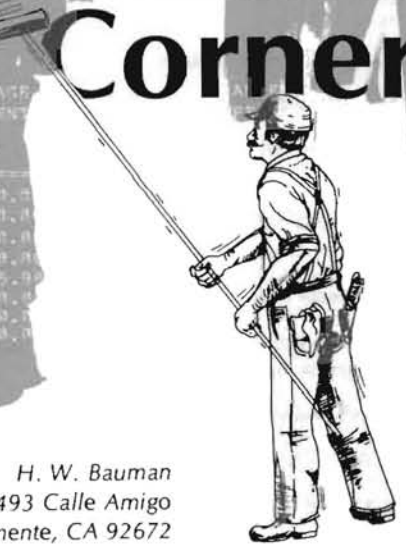
Gary Cramblitt
HQ 5th Sig Comd
P.O. Box 138
APO NY 09056

```
;       FREE - Compute free space on the disk.

;       This routine will calculate the free space on a
;       CP/M 2.2 disk. When called, the A register should
;       contain the drive letter for which free space is
;       to be calculated. For example 'A'.

;       This routine has been tested on a Z100 CP/M-85,
;       2.2.103 system, but should work for any CP/M 2.2
;       system.

BDOS    EQU   5
SELDSK  EQU   14        ;Log-in a disk
CURDSK  EQU   25        ;Returns current logged-in disk
GETALV  EQU   27        ;Get ADDR (Alloc)
GETDPB  EQU   31        ;Get ADDR (Disk Parms)

;       The following variables are returned by this
;       routine. Location FREEK contains the free space
;       on the disk in K's

DEFDRV  DB    0         ;Current logged-in drive saved
                         here
BSH     DB    0         ;Disk Block Shift Factor
BLM     DB    0         ;Number of records per disk block
DRM     DB    0         ;Max number of directory entries
FRECNT  DW    0         ;Free blocks on the disk
USDCNT  DW    0         ;Used blocks on the disk
DSM     DW    0         ;Max number of blocks on disk
FREEK   DW    0         ;Free space in "K"s

FREE    PUSH  PSW       ; Save desired drive letter on
                          stack
        MUI   C,CURDSK
        CALL  BDOS      ; Get current logged-in drive
        STA   DEFDRV    ; Save for later restoral
        POP   PSW       ; Get desired drive letter
        SUI   41H       ; Convert to binary
        MOV   E,A
        MVI   C,SELDSK  ; Log-in the desired drive
        MVI   D,0
        CALL  BDOS
        MVI   C,GETDPB
        CALL  BDOS      ; (HL) => DPB (Disk Parameter
                          Block)
        INX   H
        INX   H         ; Now pointing at BSH (Block Shift
                          Factor)
        MOV   A,M
        STA   BSH       ; Load and save BSH
        INX   H         ; Now pointing at BLM
        MOV   A,M
        STA   BLM       ; Load and save BLM
        INX   H
        INX   H         ; Now pointing at DSM (Max # of
                          blocks on disk)
        MOV   A,M
        STA   DSM       ; Load and save DSM
        INX   H
        MOV   A,M
        STA   DSM+1
        INX   H         ; Now pointing at DRM
        MOV   A,M       ; Load and save DRM (Max number of
                          directory entries...
        STA   DRM       ; ...with a maximum of 255
        MVI   C,GETALV
        CALL  BDOS      ; (HL) => ALV
        XCHG
        LHLD  DSM
        XCHG            ; (HL) => ALV  (DE) = DSM
        INX   D
        PUSH  H
        XRA   A
        MOV   H,A       ; Initialize counter to 0
        MOV   L,A
FREE2   MVI   B,8       ; 8 bits per ALV byte
        SHLD  FRECNT    ; Save free block counter
        POP   H         ; Get ALV pointer
        MOV   A,M       ; Get ALV byte
        INX   H         ; Advance to next ALV byte
        PUSH  H         ; Save ALV pointer
        LHLD  FRECNT    ; Restore free block counter
FREE3   RLC
        JC    FREE35
        INX   H         ; Increment free block counter
        JMP   FREE4
FREE35  PUSH  H
        LHLD  USDCNT    ; Increment used block counter
        INX   H
        SHLD  USDCNT
        POP   H
FREE4   MOV   C,A       ; Save current ALV byte
        DCX   D         ; Decrement DSM counter
        MOV   A,D       ; Any more blocks to check?
        ORA   E
        MOV   A,C       ;   - maybe, but first restore the
                            ALV byte
        JZ    FREE6     ;   - No more blocks to check,
                            finished counting
        DCR   B         ; Decrement bit counter
        JNZ   FREE3     ; If more bits in current ALV byte
        JMP   FREE2     ; If not more bits, go do next
                          byte
FREE6   SHLD  FRECNT
        LDA   BSH       ; BSH is # of 128-byte records per
                          block
        SBI   3         ; We want to display free "K"s
        MOV   B,A
FREE7   JZ    FREE9     ; Adjust counter to number of "K"s
        MOV   A,L
        RAL
        MOV   L,A
        JNC   FREE8
        MOV   A,H
        RAL
```

Vectored to 83 ☞

Part 5

SPREADSHEET Corner



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

The readers using Multiplan or PeachCalc/SuperCalc should have their version of the second assignment--PROFIT & LOSS--that we started in "SPREADSHEET Corner--Part 4" using 1-2-3 completed. I will show you how I completed the assignment using Multiplan and PeachCalc. Remember, there will nearly always be MORE THAN ONE WAY to complete the "model." I am choosing my methods on the basis of what I think is the easiest to understand and to use as many different features as I think that you readers are ready for. So, your "model" can and most likely should use a different procedure than mine. NO WAY IS BEST if you get the correct result and you know how you did it!

Before starting my procedure for Multiplan, I would like to offer a suggestion. LOTUS 1-2-3 comes with a "cardboard overlay" that fits above the Function Keys on the H/Z-100 with short "labels" describing each keys function. I have made myself a 3 X 5 inch file card listing the Function Keys purpose for Multiplan. I lean this card against the front panel of the 100 above the numeric key-pad. I find that this is a big help. Of course, I have a bigger problem than most of you readers because I am trying to remember three (3) different software packages. Here is a sample for such a card:

MULTIPLAN

F1---PAGE LEFT	F9---END
F2---WORD LEFT	F10--NEXT WINDOW
F3---CHAR LEFT	F11--NEXT UNLOCKED CELL
F4---CHAR RIGHT	F12--REFERENCE (@)
F5---WORD RIGHT	ESC--CANCEL (CONT-C)
F6---PAGE RIGHT	!---RECALCULATE
F7---PAGE DOWN	HELP-GET HELP (?)
F8---PAGE UP	

My Multiplan step-by-step procedure starts with the first assignment "model"--P—LTEMP1--(I will call this second assignment--P—LTEMP2). I will not go into the procedure of how to "LOAD" Multiplan or P—LTEMP1. You should know this by now! Also, I will not tell you how to SAVE or QUIT. You will have to decide when you want to do these. Most of the NOTES for 1-2-3 apply, so I will only add a few to explain "new" items. Time to start the procedure:-

1) Press F,W, type 14, press TAB, type 1, press TAB, type 18, press Return.

2) Press G,R, type 5, press TAB, type 5, press Return.

3) Press A, type 1ST QUARTER and press Return.

4) Press G,R, type 12, press TAB, type 5,press Return.

5) Type +SUM(R12C2:R12C4) and press down arrow.

6) Type +SUM(R13C2:R13C4) and press Return.

7) Press G,R, type 5, press TAB, type 6, press Return.

8) Press A, type APRIL and press right arrow.

9) Type MAY and press right arrow.

10) Type JUNE and press right arrow.

11) Press A, type 2ND QUARTER and press right arrow.

12) Type JULY and press right arrow.

13) Type AUGUST and press right arrow.

14) Type SEPTEMBER and press right arrow.

15) Press A, type 3RD QUARTER and press right arrow.

16) Type OCTOBER and press right arrow.

17) Type NOVEMBER and press right arrow.

18) Type DECEMBER and press right arrow.

19) Press A, type 4TH QUARTER and press right arrow.

20) Type YEAR TOTALS and press Return.

21) Press G,R, type 7, press TAB, type 6, press Return.

22) Type =, press left arrow twice, type * and 1.08, press down arrow.

NOTE: I do not like to type Multiplan cell names. Here is a method of writing formulas called "pointing". Watch the following steps carefully. Read your manual if you would like more information. It is the method used with the Multiplan tutorial lessons. I have used a few "SUM" functions in the steps so that you can see that the two (2) methods can be mixed. This saves me a lot of typing errors. How do you like this method?

23) Type =, press left arrow twice, type * and 1.04, press down arrow.

24) Press down arrow once.

25) Type =, press up arrow 3 times, type -, press up arrow twice, press Return.

26) Press C,R, type 2, press Return.

27) Press down arrow twice.

28) Type =, press left arrow twice, type * and 1.10, press down arrow.

29) Type =, press left arrow twice, type * and 1.17, press down arrow.

30) Press down arrow once.

31) Type =, press up arrow 5 times, type -, press up arrow 3 times, type -, press up arrow twice, press Return.

32) Press C,R and Return.

33) Press G,R, type 13, press TAB, type 7, press Return.

34) Type =, press left arrow, type * and 1.17, press up arrow.

35) Type =, press left arrow, type * and 1.10, press Return.

36) Press G,R, type 8, press Return.

37) Type =, press left arrow, type * and 1.04, press up arrow.

38) Type =, press left arrow, type * and 1.08, press right arrow.

39) Type =, press left arrow, type * and 1.08, press down arrow.

40) Type =, press left arrow, type * and 1.04, press Return.

41) Press G,R, type 12, press Return.

42) Type =, press left arrow, type * and 1.10. press down arrow.

43) Type =, press left arrow, type * and 1.17, press Return.

44) Press G,R, type 7, press TAB, type 9, press Return.

45) Type =, press left arrow 3 times, type +, press left arrow twice, type +, press left arrow once, press Return.

46) Press down arrow once.

47) Type =, press left arrow 3 times, type +, press left arrow twice, type +, press left arrow once, press Return.

48) Press down arrow twice.

49) Type +SUM(R10C6:R10C8) and press Return.

50) Press down arrow twice.

51) Type =, press left arrow 3 times, type +, press left arrow twice, type +, press left arrow once, press Return.

52) Press down arrow once.

53) Type =, press left arrow 3 times, type +, press left arrow twice, type +, press left arrow once, press Return.

54) Press down arrow twice.

55) Type +SUM(R15C6:R15C8) and press Return.

56) Press G,R, type 7, press TAB, type 10, press Return.

57) Type =, press left arrow twice, type * and 1.08, press down arrow.

58) Type =, press left arrow twice, type * and 1.04, press down arrow.

59) Press down arrow once.

60) Type =, press up arrow 3 times, type -, press up arrow twice, press Return.

61) Press C,R and press Return.

62) Press down arrow twice.

63) Type =, press left arrow twice, type * and 1.10, press down arrow.

64) Type =, press left arrow twice, type * and 1.17, press down arrow.

65) Press down arrow once.

66) Type =, press up arrow 5 times, type -, press up arrow 3 times, type -, press up arrow twice, press Return.

67) Press C,R and press Return.

68) Press G,R, type 7, press TAB, type 11, press Return.

69) Type =, press left arrow, type * and 1.08, press down arrow.

70) Type =, press left arrow, type * and 1.04, press down arrow.

71) Press down arrow 3 times.

72) Type =, press left arrow, type * and 1.10, press down arrow.

73) Type =, press left arrow, type * and 1.17, press right arrow.

74) Type =, press left arrow, type * and 1.17, press up arrow.

75) Type =, press left arrow, type * and 1.10, press up arrow.

76) Press up arrow 3 times.

77) Type =, press left arrow, type * and 1.04, press up arrow.

78) Type =, press left arrow, type * and 1.08, press right arrow.

79) Type =, press left arrow 3 times, type +, press left arrow twice, type +, press left arrow once, press Return.

80) Press down arrow once.

81) Type =, press left arrow 3 times, type +, press left arrow twice, type +, type left arrow once, press Return.

82) Press down arrow twice.

83) Type +SUM(R10C10:R10C12) and press Return.

84) Press down arrow twice.

85) Type =, press left arrow 3 times, type +, press left arrow twice, type +, press left arrow once, press Return.

86) Press down arrow once.

87) Type =, press left arrow 3 times, type +, press left arrow twice, type +, press left arrow once, press Return.

88) Press down arrow twice.

89) Type +SUM(R15C10:R15C12) and press Return.

90) Press G,R, type 7, press TAB, type 14, press Return.

91) Type =, press left arrow twice, type * and 1.08, press down arrow.

92) Type =, press left arrow twice, type * and 1.04, press down arrow.

93) Press down arrow once.

94) Type =, press up arrow 3 times, type -, press up arrow twice, press Return.

95) Press C,R and press Return.

96) Press down arrow twice.

97) Type =, press left arrow twice, type * and 1.10, press down arrow.

98) Type =, press left arrow twice, type * and 1.17, press down arrow.

99) Press down arrow once.

100) Type =, press up arrow 5 times, type -, press up arrow 3 times, type -, press up arrow twice, press Return.

101) Press C,R, and press Return.

102) Press G,R, type 7, press TAB, type 15, press Return.

103) Type =, press left arrow, type * and 1.08, press down arrow.

104) Type =, press left arrow, type * and 1.04, press down arrow.

105) Press down arrow 3 times.

106) Type =, press left arrow, type * and 1.10, press down arrow.

107) Type =, press left arrow, type * and 1.17, press right arrow.

108) Type =, press left arrow, type * and 1.17, press up arrow.

109) Type =, press left arrow, type * and 1.10, press up arrow.

110) Press up arrow 3 times.

111) Type =, press left arrow, type * and 1.04, press up arrow.

112) Type =, press left arrow, type * and 1.08, press right arrow.

113) Type =, press left arrow 3 times, type +, press left arrow twice, type +, press left arrow once, press Return.

114) Press down arrow once.

115) Type =, press left arrow 3 times, type +, press left arrow twice, type +, press left arrow once, press Return.

116) Press down arrow twice.

117) Type +SUM(R10C14:R10C16) and press Return.

118) Press down arrow twice.

119) Type =, press left arrow 3 times, type +, press left arrow twice, type +, press left arrow once, press Return.

120) Press down arrow once.

121) Type =, press left arrow 3 times, type +, press left arrow twice, type +, press left arrow once, press Return.

122) Press down arrow twice.

123) Type +SUM(R15C14:R15C16) and press Return.

124) Press G,R, type 7, press TAB, type 18, press Return.

125) Type +R7C5+C9+C13+C17 and press Return.

126) Press down arrow once.

127) Type +R8C5+C9+C13+C17 and press Return.

128) Press down arrow twice.

129) Type +R10C5+C9+C13+C17 and press Return.

130) Press down arrow twice.

131) Type +R12C5+C9+C13+C17 and press Return.

132) Press down arrow once.

133) Type +R13C5+C9+C13+C17 and press Return.

134) Press down arrow twice.

135) Type =, press up arrow 5 times, type -, press up arrow 3 times, type -, press up arrow twice, press Return.

136) Press F,C, type R7C6:R15C18, press TAB, type R, press TAB, type $, press Return.

137) Press G,R, type 6, press TAB, type 5, press Return.

138) Press C,R, type 13, press Return.

139) Press down arrow 3 times.

140) Press C,R, and press Return.

141) Press down arrow 5 times.

142) Press C,R, and press Return.

143) Press down arrow twice.

144) Press C,R, and press Return.

145) Press F,O, press TAB, press Y, press Return.

NOTE: Check your "template" formulas with your "Spreadsheet Preparation Form" carefully. Correct any errors you might find. Did you note that Multiplan expanded the column widths so that the formulas would not be truncated? Use your Function Keys to move the pointer around to help you check all the formulas. Remember, F1 will Page Left, F6 will Page Right, etc. When you are SURE that all the formulas are correct, repeat step #145 BUT press N in place of Y!

146) Press G,R, type 1, press TAB, type 2, press Return.

147) Press C,F, type R1C2:R3C4, press TAB, type R1C9, press Return.

NOTE: We used COPY to move the "model" name area closer to the middle of the "template". However, when you use the COPY command, the original text remains in place. So, we will "BLANK" the old name area. We will use the same "block range" to describe the area.

148) Press B, type R1C2:R3C4, press Return.

149) Press G,R, type 7, press TAB, type 2, press Return.

150) Press W,S,T and press Return.

NOTE: Multiplan is different from 1-2-3 and PeachCalc when we want to use Lock, Protect and Window. In some ways it has better features and in other ways it is lacking. I guess it really depends on what you get use to. We NOW have four (4) windows. We can use BORDERS to help define our "template"!

151) Press W,B, type 1, press Return.

152) Press W,B, type 2, press Return.

153) Press W,B, type 3, press Return.

154) Press W,B, type 4, press Return.

NOTE: Now you can easily see the four (4) windows. If you want to move the pointer from one window to another, you must use the F10 Function Key (H/Z-100). Now move the pointer to window #4.

155) Press L,C, type R6C1:R15C1, press TAB, type L, press Return.

NOTE: We have "LOCKED" the Labels in the window so that they cannot be accidently changed. Lets complete "LOCKING" the other Labels. Move the pointer to window #1 using the F10 key.

156) Press the down arrow once.

157) Press L,C, type R1C2:R5C1, press TAB, type L, press Return.

158) Press F10 Function Key.

159) Press L,C, type R1C2:R5C18, press TAB, type L, press Return.

NOTE: All the Labels are "LOCKED". Try changing any Label. What happened? How do you eliminate Windows, did you ask?

160) Press W,C, type 1, press Return.

161) Press W,S,V, type 5, press Return.

NOTE: We now have two (2) windows so that we can set-up to try a WHAT IF problem. We want to display the original data that we entered in our first assignment in window #1 and the Year Totals In window #2.

162) Press G,R, press TAB, type 18, press Return.

163) Press F10 Key.

164) Press G,R, type 7, press TAB, type 2, press Return.

165) Type 22456 and press Return.

NOTE: Did you watch the recalculation? If you did, you would have seen all the cell's data change. This is one of Spreadsheet's important uses! Lets do another so that you can carefully watch.

166) Type 12456 and press Return.

167) Press L,C, type R7C2:R13C4, press TAB, type L, press Return.

NOTE: We have "LOCKED" the original data. Try step #165 again. What did you find? We have completed our second assignment "model"-- P—LTEMP2 (for Multiplan). SAVE and QUIT. You should know how to do this. Have you SAVED your work several times for safe keeping? You should have. If you have ever lost a lot of work, you will learn the HARD way! ALWAYS SAVE before you QUIT!

I would like to ask you Multiplan users to make a "LIST" of the new features that you have used in this second assignment. If you did not use the "HELP" screens during the above procedure, please repeat the procedure using "HELP" screens and study the new features. Also, look up each of the new features in your Manual and study them. NOW, please repeat the second assignment until you can complete the "model" correctly with out reference to my procedure. You should only need your "HELP" screens and the Spreadsheet Preparation Form"! Repetition is the "BEST" teacher!

PeachCalc/SuperCalc are nearly the same as the LOTUS 1-2-3. Multiplan has many differences. Therefore, we will design our PeachCalc "model" using similar methods that we used with 1-2-3. Remember, there will nearly always be several ways to accomplish a "GOOD" Spreadsheet "model". Feel free to experiment! Here is a "list" of characters to remember when using PeachCalc:

CHARACTER	RESULT
/	Enter Command
=	Specify Block to Jump To
!	Recalculate
;	Change Screen Window
CONT-Z	Clear Out Current Entry Line
"	Start Text Block
'	Start Repeating Text Blocks
Any Other	Starts Formula Blocks
?	Calls Help Screens

My PeachCalc step-by-step procedure is also started with the first assignment "model"--P—LTEMP1 (I will call this second assignment--P—LTEMP2). Again, I will not repeat how to "Load" the software or P—LTEMP1 and I will not remind you when or how to SAVE. You should know these. I will only supply a NOTE when they are required to explain something "new". Here we go:

1) Press /G,F.

2) Type = and E5 and press Return.

3) Type "1ST QUARTER and press Return.

4) Type = and E12 and press Return.

5) Type +SUM(B12:D12) and press Return.

6) Type = and E13 and press Return.

7) Type +SUM(B13:D13) and press Return.

8) Type = and E5 and press Return.

9)) Press /M,C, type E, press Return, type S, press Return.

NOTE: If you would like to check what you have Moved, Type = and S1 and press Return. Now, type = and E5 and press Return. Back where we started from!

10) Type "APRIL and press Return.

NOTE: Do you remember how to set the automatic direction arrow? Look back to your last assignment to refresh your mind!

11) Type "MAY and press Return.

12) Type "JUNE and press Return.

13) Type "JULY and press Return.

14) Type "AUGUST and press Return.

15) Type "SEPTEMBER and press Return.

16) Type "OCTOBER and press Return.

17) Type "NOVEMBER and press Return.

18) Type "DECEMBER and press Return.

19) Type = and E7 and press Return.

20) Type +D7, * and 1.08, press Return.

21) Type +D8, * and 1.04, press Return.

22) Type = and E12 and press Return.

23) Type +D12, * and 1.10, press Return.

24) Type +D13, * and 1.17, press Return.

25) Type = and E7 and press Return.

26) Press /R, type E7, press Return, type F7:M7, press Return.

27) Type = and E8 and press Return (or press down arrow once).

28) Press /R, type E8, press Return, type F8:M8, press Return.

29) Type = and D10 and press Return.

30) Press /R, type D10, press Return, type E10:M10, press Return.

31) Type = and E12 and press Return.

32) Press /R, type E12, press Return, type F12:M12, press Return.

33) Press down arrow once.

34) Press /R, type E13, press Return, type F13:M13, press Return.

35) Type = and D15 and press Return.

36) Press /R, type D15, press Return, type E15:M15, press Return.

37) Type = and E5 and press Return.

38) Press /MC, type S, press Return, type E, press Return.

NOTE: Did the 1ST QUARTER column come back into its place on your "template"? We moved it from where we stored it, back to its place per our "Spreadsheet Preparation Form"!

39) Type = and I5 and press Return.

NOTE: We must add a blank column at this place on our "template" for the 2ND QUARTER column. BE SURE to note how the formulas "correct" themselves as we add a column. This is another way that Spreadsheets can save "re-typing" with its associated chances for typing ERRORS!

40) Press /I,C, type I, press Return.

41) Type "2ND QUARTER and press Return.

42) Press down arrow once.

43) Type +SUM(F7:H7) and press Return.

44) Type +SUM(F8:H8) and press Return.

45) Press down arrow once.

46) Type +SUM(F10:H10) and press Return.

47) Press down arrow once.

48) Type +SUM(F12:H12) and press Return.

49) Type +SUM(F13:H13) and press Return.

50) Press down arrow once.

51) Type +SUM(F15:H15) and press Return.

52) Type = and M5 and press Return.

53) Press /I,C, type M, press Return.

54) Type "3RD QUARTER and press Return.

55) Type = and M7 and press Return.

56) Type +SUM(J7:L7) and press Return.

57) Type +SUM(J8:L8) and press Return.

58) Press down arrow once.

59) Type +SUM(J10:L10) and press Return.

60) Press down arrow once.

61) Type +SUM(J12:L12) and press Return.

62) Type +SUM(J13:L13) and press Return.

63) Press down arrow once.

64) Type +SUM(J15:L15) and press Return.

65) Type = and Q5 and press Return.

66) Type "4TH QUARTER and press Return.

67) Type = and Q7 and press Return.

68) Type +SUM(N7:P7) and press Return.

69) Type +SUM(N8:P8) and press Return.

70) Press down arrow once.

71) Type +SUM(N10:P10) and press Return.

72) Press down arrow once.

73) Type +SUM(N12:P12) and press Return.

74) Type +SUM(N13:P13) and press Return.

75) Press down arrow once.

76) Type +SUM(N15:P15) and press Return.

77) Type = and R5 and press Return.

78) Type "YEAR TOTALS and press Return.

79) Type = and R7 and press Return.

80) Type +E7+I7+M7+Q7 and press Return.

81) Type +E8+I8+M8+Q8 and press Return.

82) Press down arrow once.

83) Type +E10+I10+M10+Q10 and press Return.

84) Press down arrow once.

85) Type +E12+I12+M12+Q12 and press Return.

86) Type +E13+I13+M13+Q13 and press Return.

87) Press down arrow once.

88) Type +E15+I15+M15+Q15 and press Return.

89) Type = and F6 and press Return.

90) Type '* and press Return.

91) Type = and F9 and press Return.

92) Type '- and press Return.

93) Type = and F14 and press Return.

94) Type '- and press Return.

95) Type = and F16 and press Return.

96) Type '= and press Return.

97) Type = and S6 and press Return.

98) Type 'space and press Return.

99) Type = and S9 and press Return.

100) Type 'space and press Return.

101) Type = and S14 and press Return.

102) Type 'space and press Return.

103) Type = and S16 and press Return.

104) Type 'space and press Return.

105) Type = and B1 and press Return.

106) Press /C, type B1:D3, press Return, type I, press Return.

NOTE: We have copied the Spreadsheet "name block" nearer to the center of the "template". Did you notice that we typed a "block

range"-- B1:D3? We only needed to use one corner of the "block range" to locate its new location! When you use the COPY command, the "original" is still left in place. We must "BLANK" it out with the the same "block range"!

107) Press /B, type B1:D3, press Return.

108) Press /G,F.

NOTE: Again, we find that PeachCalc has all of our calculations made for us right up to this step. I would like to have the month's name at the top of our columns "right-aligned". PeachCalc does not provide a "center-alignment".

109) Press /F,R, type 5, press Return, type TR, press Return.

110) Type = and A5 and press Return.

111) Press /T, type B, press Return.

NOTE: To "LOCK" BOTH our vertical and horizontal "title" areas we must locate the pointer to cell A5. Then all title blocks to the left of and above the pointer and including the pointer row and column will be "protected". Try using the arrow keys to see if you can "scroll" into this area!

112) Type = and E6 and press Return.

113) Press /W, type V, press Return.

114) Type = and Q5 and press Return.

115) Type ; and press Return.

NOTE: Do you remember what the (;) is for? Look at the chart of commands that we typed before starting this step-by-step procedure.

116) Type = and D7 and press Return.

NOTE: The screen now has two (2) windows so that we can view the two (2) original months on the left screen and the YEAR TOTALS on the right screen. Now if we do a "WHAT IF", we can watch the effect of changing an original data cell on the Total for the Year! Pretty neat, wouldn't you say?

117) Type = and B7 and press Return.

118) Type 22456 and press Return.

NOTE: Did you see the changes take place in both windows? We will try it again. Watch carefully!

119) Type 12456 and press Return.

120) Press /P, type B7:D13, press Return.

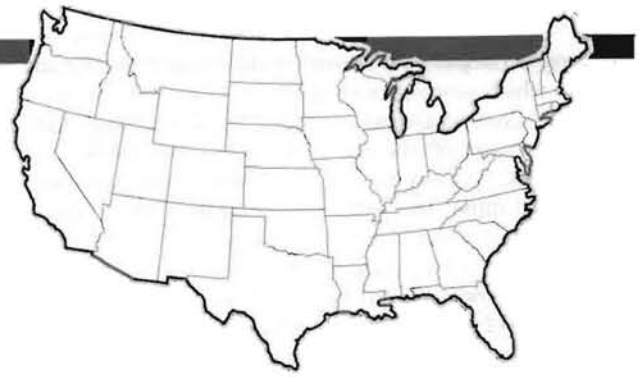
NOTE: We have used another "block range"--B7:D13--to "lock/protect" all of the original data so that this data cannot be changed. Try steps 117 and 118 again. Did it work? We have now completed our second assignment--P—LTEMP2 (for PeachCalc). SAVE and QUIT! You should know how. Have you SAVED your work several times along the procedure steps? You should have! ALWAYS SAVE before you QUIT!!!

We have covered a lot of new Spreadsheet features with this second assignment. Please make a "list" of them. If you did not use the "HELP" screens for explanations during the above steps, repeat the procedure and study these "HELP" screens as you proceed. Also, you should look each of the features up in your Manual and study them. NOW, repeat the step-by-step procedure until you can do the assignment without referring to mine. You should only need to refer to your "Spreadsheet Preparation Form" and use the "HELP" screens. When you can do this, you will be ready for the next assignment.

Closing

We will be doing one more "model" to learn a few more Spreadsheet features that we need to know in order to start our first "PROJECT". This PROJECT will be to design "templates" that can be used to prepare your Federal Income Tax Return! This will require using a large "work-screen" and most of your working RAM. In fact, the limitation on how many "forms" (A, B, Etc.) you can put in this "model" will depend on the amount of RAM that you have. If you are using an H8/H89 type of computer you will need a minimum of 64K and if you are using a H/Z-100 computer you should have at least 192K of memory! If you do not have this, save the money you would have paid someone to prepare your Return and buy more memory for your computer with it. How do you like that idea?

If you are having any problems with the first or second assignments, you MUST clear up these problems! If you cannot solve your problems, remember that this is your "SPREADSHEET Corner". Write up your problem in detail and/or send a hard-copy of your "model" to the author for help. **No Phone Calls Please** and be sure to enclose a SASE (business size) with your questions! We want ALL READERS to be up-to-date for the first "PROJECT"!



An Easy Method For Drawing The United States In ZBASIC

John E. McLaughlin
93 North 300 West
Brigham City, UT 84302

In attempting to map an area, such as the United States, on the computer monitor and accurately portray points on that map by means of a coordinate system, one is immediately confronted by two basic problems. First, exactly how much realism is necessary in the visual map--that is, how much distortion in the shape of the area is acceptable--and second, exactly how much ease in entering the coordinates is necessary? In the ideal world, a video screen exists in which all the pixels are perfectly square. This would solve both problems simultaneously, as the video representation would have zero distortion for a coordinate system in which both the vertical and horizontal coordinates were equally spaced and numbered. The ideal world, however, has nothing to do with designing the pixels in the average monitor; so we are forced to live in a world of rectangular pixels and must weigh the needs of a program in deciding whether visual accuracy or coordinate accuracy is more important.

My particular problem involves ownership of a large number of U.S. Geological Survey maps covering various parts of the West. I wanted a method of locating these on a reference map of the western United States and color coding them as to scale. The maps are of three types--large area maps (usually on the scale of 1:250,000) covering National Parks or Monuments, 15' maps covering 15' of longitude and latitude, and 7.5' maps covering 7.5' of longitude and latitude.

Because of the large number of maps, I considered the entry of accurate coordinates for each map to be more important than visual accuracy. This decision had two positive ramifications--first, since the 15' maps cover the same area as four 7.5' maps, then the depiction of these two series was greatly simplified; and second, since the distance between lines of longitude in this part of the world is less than the distance between lines of latitude, the maps are vertically rectangular, which happens to be the shape of the pixels on my video screen (a Zenith ZVM-134), thus reducing the visual distortion. While some visual distortion remains on the video display, it only involves a slight increase in height and is not overly problematic.

The basic coordinate system, which I adopted, is to consider one pixel as equal to one 7.5' map--that is, each pixel represents 7.5' of longitude and 7.5' of latitude, or, in other words, eight pixels equals one degree of longitude or latitude. By considering the upper left corner of the pixel at coordinate (1,1) to be latitude 49 North and longitude 125 West, and using the U.S.G.S. topographical map index for each state as a guide, I was able to construct the outlines of each of the western states using a series of LINE(x,y)-(x,y) commands connecting the prominent undulations of each state boundary by entering the coordinates for the 7.5' map which contained the protuberance or indentation. For some utterly arbitrary reason, I chose to draw the eleven states in alphabetical order; therefore, the program proceeds to draw Arizona in toto, then California (minus its Arizona border which is already drawn), Colorado (in toto), Idaho (in toto), Montana (minus its border with Idaho), Nevada, New Mexico, Oregon, Utah, Washington, and Wyoming. Drawing the borders consisting of mountain ranges, rivers, and coastlines, it was necessary to generalize the shape somewhat, but depending on the state this was not overly difficult. (Since Utahns tend to think of Californians, like Coloradians think of Texans, the California coastline has been more generalized than the others (lines 420-460), but this can be easily fixed by a loving Californian with the California topographical map index.)

Running the Program

This program was written on a H/Z-100 with 192K and color RAM. It requires only one disk drive, but needs some preparatory work in terms of creating the data files. This particular program requires two sequential files named "NATPARK.DOC" and "MAPS.DOC." The file "NATPARK.DOC" contains the large area National Park and Monument maps and is set up with each line containing five pieces of information separated by commas--the name of the map, the upper left corner coordinates, and the lower right coordinates (longitude first and latitude second). For example:

```
Death Valley National Monument and Vicinity,59,94,70,108
Sequoia and Kings Canyon National Parks and Vicinity,48,95,55,102
```

The second file, "MAPS.DOC" contains the 15' and 7.5' maps and each line contains four pieces of information--the name of the map, the series of the map (15 or 7.5), and the upper left corner coordinates (for the 7.5' maps this is equivalent to the coordinates for the map itself) with the latitude given first and the longitude second (I can't remember why I reversed the order between the "NATPARK.DOC" format and the "MAPS.DOC" format as to listing longitude and latitude). For example:

```
Furnace Creek CA,15,101,65
Mount Whitney CA,15,99,53
Goose Creek ID-NV-UT,15,57,87
Mount Bonneville WY,15,49,125
Crestone Peak CO,7.5,89,156
Brigham City UT,7.5,60,104
Gilbert Peak NE UT-WY,7.5,65,118
Grand Teton WY,7.5,43,114
Mount Bonneville WY,7.5,50,126
```

The program begins by drawing the outlines of the states and asking the user which running mode is desired or whether to end. The two modes are Continuous and Individual. The Continuous mode simply reads the maps off the sequential files "NATPARK.DOC" and "MAPS.DOC," colors them appropriately, and places them on the screen until done. This is the default mode. The Individual mode reads each map, colors it specially, and writes it's name at the bottom of the display. Each carriage return colors the last map according to its series and reads the next map, etc. This proceeds until the last map has been read and colored on the screen. After each mode has finished, hitting any key will clear the screen, redraw the outline map, and once again ask the user what is desired.

Lines 200-230

These lines set up a variable of 70 spaces named SPAC$, establish the color of printing and background, clear the screen, and set up a variable BOR, which is the color used in drawing the state boundaries.

Lines 250-1070

These lines draw the outlines of the eleven western states in alphabetical order--Arizona, California, Colorado, Idaho, Montana, Nevada, New Mexico, Oregon, Utah, Washington, and Wyoming. As each state is drawn, any common borders which have already been drawn are not repeated, so that drawing Wyoming involves only the single straight line which separates Wyoming from South Dakota and Nebraska.

Line 1080

This line sets up three variables which indicate the different colors that the various scales of map will be. NATPARK is the color of the large area National Park and Monument maps. FIF is the color of the 15' maps. SEV is the color of the 7.5' maps. This is a variable which sets the color of the map indicated at the bottom of the display when the individual map mode is running.

Line 1090

This line queries the user as to which type of display is desired or whether to exit the program. The default mode is Continuous.

Lines 1100-1360

This is the individual display mode. Lines 1110-1180 read the National Park and Monument maps from the file "NATPARK.DOC." If this file is not needed by a user, then these lines may be deleted. Lines 1190-1310 read the 15' and 7.5' maps from the file "MAPS.DOC." Line 1220 determines whether the map is a 15' map or a 7.5' map. If SCALE$ is "15," then each map is drawn as a filled two pixel by two pixel box (lines 1230-1260), otherwise it is a single lighted pixel (lines 270-1290).

Line 1380

This line ends the program.

Lines 1390-1580

This is the Continuous display mode. Lines 1390-1440 read the file "NATPARK.DOC" and can be deleted if not needed. Lines 1450-1530 read the file "MAPS.DOC" and draw a box for 15' maps (line 1490) and light a pixel for 7.5' maps (line 1510).

Line 1590

This subroutine simply waits for input from the keyboard to proceed.

Further Applications

While the present program simply displays the location of various maps, the use of the U.S.G.S. topographical map indexes for plotting coordinates on these outlines is nearly limitless, as anyone who has seen the detail contained on these indexes can attest. My next project is to use these same state outlines and indexes to plot dialect isoglosses within the Shoshoni speech area of California, Nevada, Utah, Idaho, and Wyoming. Since the indexes show cities, rivers, county boundaries, and other natural and man-made features, in brown underneath the black grid of the maps available, the job of locating longitude and latitude for each community and translating this into computer coordinates has already been done by the U.S.G.S. and this program. These indexes are available free of charge by writing:

Branch of Distribution
U.S. Geological Survey
1200 South Eads Street
Arlington, Virginia 22202

(For indexes east of the Mississippi River, including Minnesota, Puerto Rico, and the Virgin Islands)

Branch of Distribution
U.S. Geological Survey
Box 25286 Federal Center
Denver, Colorado 80225

(For indexes west of the Mississippi River, including Alaska, Hawaii, Louisiana, American Samoa, and Guam)

Listing 1

```
10 '    ********************************************
20 '    ********************************************
30 '    **********    USGS MAPS ver 1.01    **********
40 '    **********                          **********
50 '    **********    John E. McLaughlin    **********
60 '    **********    93 North 300 West     **********
70 '    **********  Brigham City, UT 84302  **********
80 '    **********                          **********
90 '    **********       19 May 1984        **********
100 '     ********************************************
110 '    ********************************************
200 SPAC$=SPACE$(70)
210 COLOR 7,0
220 CLS
230 BOR=1
240 REM ARIZONA
250 LINE(89,96)-(128,96),BOR
260 LINE(128,96)-(128,141),BOR
270 LINE(128,141)-(112,141),BOR
280 LINE(112,141)-(82,132),BOR
290 LINE(82,132)-(85,129),BOR
300 LINE(85,129)-(83,128),BOR
310 LINE(83,128)-(83,125),BOR
320 LINE(83,125)-(87,118),BOR
330 LINE(87,118)-(83,114),BOR
340 LINE(83,114)-(83,104),BOR
350 LINE(83,104)-(89,104),BOR
360 LINE(89,104)-(89,96),BOR
370 REM CALIFORNIA
380 LINE(7,57)-(40,57),BOR
390 LINE(40,57)-(40,81),BOR
400 LINE(40,81)-(83,112),BOR
410 LINE(85,129)-(64,132),BOR
420 LINE(64,132)-(63,129),BOR
430 LINE(63,129)-(57,123),BOR
440 LINE(57,123)-(35,116),BOR
450 LINE(35,116)-(5,69),BOR
460 LINE(5,69)-(7,57),BOR
470 REM COLORADO
480 LINE(128,65)-(184,65),BOR
```

```
490 LINE(184,65)-(184,96),BOR
500 LINE(184,96)-(128,96),BOR
510 LINE(128,96)-(128,65),BOR
520 REM IDAHO
530 LINE(65,57)-(65,39),BOR
540 LINE(65,39)-(63,38),BOR
550 LINE(63,38)-(69,27),BOR
560 LINE(69,27)-(65,24),BOR
570 LINE(65,24)-(65,1),BOR
580 LINE(65,1)-(72,1),BOR
590 LINE(72,1)-(72,9),BOR
600 LINE(72,9)-(86,19),BOR
610 LINE(86,19)-(86,29),BOR
620 LINE(86,29)-(89,27),BOR
630 LINE(89,27)-(98,37),BOR
640 LINE(98,37)-(109,35),BOR
650 LINE(109,35)-(112,37),BOR
660 LINE(112,37)-(112,57),BOR
670 LINE(112,57)-(65,57),BOR
680 REM MONTANA
690 LINE(72,1)-(168,1),BOR
700 LINE(168,1)-(168,32),BOR
710 LINE(168,32)-(112,32),BOR
720 LINE(112,32)-(112,37),BOR
730 REM NEVADA
740 LINE(40,57)-(65,57),BOR
750 LINE(89,57)-(89,96),BOR
760 REM NEW MEXICO
770 LINE(176,96)-(176,136),BOR
780 LINE(176,136)-(144,136),BOR
790 LINE(144,136)-(144,138),BOR
800 LINE(144,138)-(135,138),BOR
810 LINE(135,138)-(135,141),BOR
820 LINE(135,141)-(128,141),BOR
830 REM OREGON
840 LINE(7,57)-(4,50),BOR
850 LINE(4,50)-(9,23),BOR
860 LINE(9,23)-(16,23),BOR
870 LINE(16,23)-(18,24),BOR
880 LINE(18,24)-(18,27),BOR
890 LINE(18,27)-(23,28),BOR
900 LINE(23,28)-(43,25),BOR
910 LINE(43,25)-(65,24),BOR
920 REM UTAH
930 LINE(112,57)-(112,65),BOR
940 LINE(112,65)-(128,65),BOR
950 REM WASHINGTON
960 LINE(9,23)-(5,11),BOR
970 LINE(5,11)-(3,9),BOR
980 LINE(3,9)-(3,5),BOR
990 LINE(3,5)-(15,7),BOR
1000 LINE(15,7)-(16,8),BOR
1010 LINE(16,8)-(18,8),BOR
1020 LINE(18,8)-(18,5),BOR
1030 LINE(18,5)-(15,4),BOR
1040 LINE(15,4)-(16,1),BOR
1050 LINE(16,1)-(65,1),BOR
1060 REM WYOMING
1070 LINE(168,32)-(168,65),BOR
1080 NATPARK=5:FIF=4:SEV=6:THIS=7
1090 LOCATE 24,1:PRINT
     "<C>ontinuous or <I>ndividual or <E>nd? <C>";
     :GOSUB 1590
1100 IF A$="I" OR A$="i" THEN 1110 ELSE 1380
1110 OPEN "I",1,"NATPARK.DOC"
1120 WHILE NOT EOF(1)
1130    INPUT #1,MAP$,A,B,C,D
1140    LINE(A,B)-(C,D),THIS,BF
1150    LOCATE 24,1:PRINT SPAC$;:LOCATE 24,1:PRINT MAP$;
        :GOSUB 1590
1160    LINE(A,B)-(C,D),NATPARK,BF
1170 WEND
1180 CLOSE
1190 OPEN "i",1,"maps.doc"
1200 WHILE NOT EOF(1)
1210    INPUT #1,MAP$,SCALE$,VERT,HORIZ
1220    IF SCALE$<>"15" THEN 1270
1230    LINE(HORIZ,VERT)-(HORIZ+1,VERT+1),THIS,BF
1240    LOCATE 24,1:PRINT SPAC$;:LOCATE 24,1:PRINT MAP$;
        :GOSUB 1590
1250    LINE(HORIZ,VERT)-(HORIZ+1,VERT+1),FIF,BF
1260    GOTO 1300
1270    PSET(HORIZ,VERT),THIS
1280    LOCATE 24,1:PRINT SPAC$;:LOCATE 24,1:PRINT MAP$;
        :GOSUB 1590
1290    PSET(HORIZ,VERT),SEV
1300 WEND
1310 CLOSE
1320 LOCATE 24,1:PRINT SPAC$;
1330 LOCATE 23,1
1340 PRINT "(Hit any key to continue)"
1350 GOSUB 1590
1360 GOTO 200
1380 IF A$="E" OR A$="e" THEN END
1390 OPEN "i",1,"natpark.doc"
1400 WHILE NOT EOF(1)
1410    INPUT #1,MAP$,A,B,C,D
1420    LINE(A,B)-(C,D),NATPARK,BF
1430 WEND
1440 CLOSE
1450 OPEN "i",1,"maps.doc"
1460 WHILE NOT EOF(1)
1470    INPUT #1,MAP$,SCALE$,VERT,HORIZ
1480    IF SCALE$<>"15" THEN 1510
1490    LINE(HORIZ,VERT)-(HORIZ+1,VERT+1),FIF,BF
1500    GOTO 1520
1510    PSET(HORIZ,VERT),SEV
1520 WEND
1530 CLOSE
1540 LOCATE 24,1:PRINT SPAC$;
1550 LOCATE 23,1
1560 PRINT "(Hit any key to continue)"
1570 GOSUB 1590
1580 GOTO 200
1590 A$=INKEY$:IF A$="" THEN 1590 ELSE RETURN
```

NATPARK.DOC

Bennetts Well CA, 15, 103, 65
Big Pine CA, 15, 95, 53
Bishop CA, 15, 93, 53
Blanco Mtn CA, 15, 93, 55
Confidence Hills CA, 15, 105, 67
Coso Peak CA, 15, 103, 59
Darwin CA, 15, 101, 59
Dry Mtn CA, 15, 97, 59
Eagle Mtn CA, 15, 103, 69
Emigrant Canyon CA, 15, 101, 63
Funeral Peak CA, 15, 103, 67
Furnace Creek CA, 15, 101, 65
Haiwee Reservoir CA, 15, 103, 57
Hockett Peak CA, 15, 103, 53
Independence CA, 15, 97, 55
Keeler CA, 15, 101, 57
Kern Peak CA, 15, 101, 53
Kernville CA, 15, 105, 53
Lamont Peak CA, 15, 105, 55
Little Lake CA, 15, 105, 57
Lone Pine CA, 15, 99, 55
Manly Peak CA, 15, 105, 63
Marble Canyon CA, 15, 99, 61
Maturango Peak CA, 15, 103, 61
Monache Mtn CA, 15, 103, 55
Mount Whitney CA, 15, 99, 53
Mountain Springs Canyon CA, 15, 105, 59
Mt Pinchot CA, 15, 97, 53

New York Butte CA, 15, 99, 57
Olancha CA, 15, 101, 55
Panamint Butte CA, 15, 101, 61
Shoshone CA, 15, 105, 69
Stovepipe Wells CA, 15, 99, 63
Telescope Peak CA, 15, 103, 63
Timber Mtn CA, 15, 59, 29
Tin Mtn CA, 15, 97, 61
Trona CA, 15, 105, 61
Ubehebe Peak CA, 15, 99, 59
Waucoba Mtn CA, 15, 95, 55
Waucoba Spring CA, 15, 95, 57
Waucoba Wash CA, 15, 97, 57
Wingate Wash CA, 15, 105, 65
Ash Meadows CA-NV, 15, 101, 69
Big Dune CA-NV, 15, 99, 67
Bullfrog CA-NV, 15, 97, 65
Chloride Cliff CA-NV, 15, 99, 65
Grapevine Peak CA-NV, 15, 97, 63
Last Chance Range CA-NV, 15, 95, 59
Magruder Mtn CA-NV, 15, 93, 59
Ryan CA-NV, 15, 101, 67
Soldier Pass CA-NV, 15, 93, 57
Ubehebe Crater CA-NV, 15, 95, 61
Electric Peak CO, 15, 87, 155
Goose Creek ID-NV-UT, 15, 57, 87
Cotton Thomas Basin ID-UT, 15, 57, 89
Kelton Pass ID-UT, 15, 57, 95
Park Valley ID-UT, 15, 57, 93
Yost ID-UT, 15, 57, 91
Bare Mtn NV, 15, 97, 67
Lathrop Wells NV, 15, 99, 69
Springdale NV, 15, 95, 65
Thirsty Canyon NV, 15, 95, 67
Timber Mtn NV, 15, 95, 69
Topopah Spring NV, 15, 97, 69
Santaquin UT, 15, 73, 105
Santaquin Peak UT, 15, 73, 107
Mount Bonneville WY, 15, 49, 125
Bonnie Claire SW CA-NV, 7.5, 96, 63
Beck Mtn CO, 7.5, 89, 157
Courthouse Mtn CO, 7.5, 87, 140
Crestone Peak CO, 7.5, 89, 156
Sheep Mtn CO, 7.5, 87, 141
Uncompahgre Peak CO, 7.5, 88, 141
Wetterhorn Peak CO, 7.5, 88, 140
Almo ID, 7.5, 56, 91
Cache Peak ID, 7.5, 55, 91
Bonnie Claire NV, 7.5, 95, 64
Bonnie Claire NW NV, 7.5, 95, 63
Bonnie Claire SE NV, 7.5, 96, 64
Gold Point NV, 7.5, 94, 62
Gold Point SW NV, 7.5, 94, 61
Scottys Junction NV, 7.5, 94, 64
Scottys Junction SW NV, 7.5, 94, 63
Aspen Grove UT, 7.5, 69, 108
Brigham City UT, 7.5, 60, 104
Brighton UT, 7.5, 68, 108
Draper UT, 7.5, 68, 106
Dromedary Peak UT, 7.5, 68, 107
Garden City UT, 7.5, 57, 109
Garfield Basin UT, 7.5, 67, 117
Honeyville UT, 7.5, 59, 104
Kings Peak UT, 7.5, 66, 118
Lehi UT, 7.5, 69, 106
Mantua UT, 7.5, 61, 105
Meadowville UT, 7.5, 58, 109
Mount Emmons UT, 7.5, 67, 118
Mount Pisgah UT, 7.5, 60, 105
Mt Elmer UT, 7.5, 58, 107
Mt Powell UT, 7.5, 66, 117
Naomi Peak UT, 7.5, 57, 107
Temple Peak UT, 7.5, 58, 108
Timpanogos Cave UT, 7.5, 69, 107
Tony Grove Creek UT, 7.5, 57, 108
Wellsville UT, 7.5, 59, 105
Willard UT, 7.5, 61, 104
Bridger Lake UT-WY, 7.5, 65, 117
Gilbert Peak NE UT-WY, 7.5, 65, 118
Grand Teton WY, 7.5, 43, 114
Halls Mountain WY, 7.5, 49, 125
Horseshoe Lake WY, 7.5, 49, 124
Jenny Lake WY, 7.5, 42, 115
Moose WY, 7.5, 43, 115
Mount Bonneville WY, 7.5, 50, 126
Mt Moran WY, 7.5, 42, 114
Raid Lake WY, 7.5, 50, 125
Roberts Mountain WY, 7.5, 49, 126
Scab Creek WY, 7.5, 50, 124
Death Valley National Monument and Vicinity,59,94,70,108
Sequoia and Kings Canyon
National Parks and Vicinity,48,95,55,102

Transmitting Pages & Cursor Position Reports on the '89

Jim Feasel
1121 ECR 16
Tiffin, OH 44883

As all of us '89 fans know, our computer contains an impressive set of escape codes, giving us a great deal of control over it's many and varied functions. Most of the functions are easy to use and self-explanatory. Others, such as Direct Cursor Control, have been covered by many fine articles here in the pages of REMark. I would like to explain the usage (in BASIC) of two functions which still remain a bit unclear in the minds of many '89 users that I know. These are the Transmit Page and Cursor Position Report functions.

Transmit Page (ESC #)

The Heath Operation Manual states that to use this feature the "computer requires a special routine .." to accept or catch the information sent to it by the terminal. A lot of us users out here would like to have known what they meant by a "special routine." I always thought that it would have to be some exotic machine language affair, and indeed it may have to be to use this feature to it's fullest extent.

There is, however, a way for us "high level language" programmers, who deal mostly with BASIC, to get some use of the Transmit Page function. We can write a "special routine" in BASIC to catch the transmitted page of information like this:

```
10 DIM X$(24)
20 PRINT CHR$(27)"E"  'Clear Screen
30 .
40 .  (Input information and format screen as desired)
50 .
60 .

100 PRINT CHR$(27)"#"  'Transmit Page
110 FOR I=1 TO 24
120  X$(I)=INPUT$(79)
130 NEXT I
```

The screen, full of information, is now contained in the string array X$(1-24) and can be processed or stored, etc. just like any string data.

Why, you may ask, INPUT$(79) instead of INPUT$(80), since there are 80 characters per line. The only answer I can give is that INPUT$(80) doesn't work, possibly because of the carriage return/line feed sequence at the end of the line, and causes BASIC to sit and wait for more characters after the page has been transmitted. And yes, you will lose any characters that are in column 80. I, however, have no trouble living with a 79 column screen when I plan to use this function.

Two other points to make when using this routine. First, if there are areas of Reverse Video or Graphics Characters on the screen when the page is transmitted, the terminal will send the Escape sequences required to display such information. Depending on the display format, these unprintable characters will get into the X$() array and cause that many more characters to be lost at the bottom of the page. This is the area that needs more experimentation, and perhaps that exotic machine language routine.

The second point has to do with the way BASIC stores it's string variables in memory. This is a subject that would take another article to explain completely, but for now suffice it to say that BASIC is a rather messy housekeeper. When all the free memory is "messed up" by discarded strings and variables, BASIC will suspend operation, get everything back in order and then resume where it left off. This explains why things seem to "lock up" once in a while. Sometimes this can take more than a minute, but usually only a few seconds, depending on how many good variables are in memory. The important thing about all this is we don't want the housekeeper to show up when we are busy and don't want to be interrupted, (ie: in the middle of transmitting a page!) Therefore, we should make sure that memory is cleaned up before transmitting a page. This can be done with the FRE("") command. The statement:

```
PRINT FRE("")
```

or

```
<dummy variable> = FRE("")
```

will cause BASIC to do its housekeeping chores before continuing. In any case, it is important that at least 2000 bytes of memory be free before using the Transmit Page function. Otherwise, the buffer overflow bell will scream with anger and things can really get messed up, sometimes to the point of necessitating a cold boot.

I have no idea if this routine will work under HDOS, as I have never had an opportunity to try it out. Perhaps someone out there can give it a try and let us all know.

Cursor Position Report (ESC n)

The Operation Manual is a bit more helpful on this one, or at least tries to be, by including a short BASIC routine for using the CPR function. Their routine, however, causes characters to be printed on the screen, and even makes the user hit the RETURN key to input the cursor position information. I wanted a way to get a report on the cursor position that was completely transparent to the user of the program, and that certainly didn't cause any confusing characters to be printed on the screen.

The following simple routine accomplishes this very nicely.

```
100 PRINT CHR$(27)"n";
    ' Request report of cursor position
110 I$=INPUT$(3)
120 IF LEFT$(I$,1)<>"Y" THEN I$=RIGHT$(I$,2)+INPUT$(1)
130 ROW=ASC(MID$(I$,2,1))-31
140 COL=ASC(RIGHT$(I$,1))-31
```

The variable, ROW, now contains the row number the cursor is in and, COL, the column. Nothing was printed to the screen and the cursor didn't move.

Actually, the routine looks more complicated than it is. When ESC n is sent to the terminal it returns three characters. Thus, the INPUT$(3). The first one should always be a "Y" (Ex: Y7!). I have noticed in using this function in several programs, however, that there seems to be a one character buffer somewhere between the terminal and the computer. Sometimes an extraneous character gets into this buffer and causes four characters to come from the terminal. If we input 3 of them, that leaves the last character of the cursor position string left sitting there in the buffer. This guarantees that the next time a Cursor Position report is called for, that character is going to mess it up again. Line 120, in effect, looks at the first character to see if it is indeed a "Y" like it is supposed to be. If not, it tosses it away and gets the fourth character and tacks it onto the end of the cursor position string. Lines 130 & 140 look at the second and third characters of the string and subtracts 31 from their ASCII value, yielding the true ROW and COL position.

This function can be very useful when writing programs that are "screen-oriented." I hope you will find it helpful.

Sample Program

If you are like me, reading about a certain programming technique is one thing, but seeing it used in an actual program's context gives a better understanding of just what is going on. With that thought in mind, I set out to write a short program to demonstrate the use of these functions.

ED.BAS (Listing #1) is a two page (48 line) text editor that lets the user make full use of all the H-19's built-in editing capabilities, and then send the text to the computer for storage or printout. In order to make the program useful (such as for writing single page letters), it's got to be a little longer than it was intended to be. Although many more features could be added to it easily, it is not meant to be used for a full blown "word-processor." Hopefully though, it will give you some idea of how these functions can be used to enhance or make possible a program that you may have in mind to write yourself.

I will attempt to go through the program line-by-line explaining it as we go, in hopes that you may pick up some helpful hints in other areas along the way.

Lines 70-190: I put some instructions to use the program here so that they would be handy to review by listing the program before running. There are only a few things to remember because all functions are carried out by use of the Function Keys, which are labeled on the 25th line.

Lines 200-310: The string constants used throughout the program are defined here. String constants are string variables that will not be changed during the program. They should always be assigned at the beginning of a program, so that BASIC won't have to move them around while doing it's housekeeping chores. The X$() array is dimensioned to accept the 48 lines we will assign and reassign to it later.

Lines 400-490: Line 420 clears the screen and moves the cursor down the screen 10 lines, where line 430 asks for the filename where you may have some text stored to edit. Line 440 skips the file input routine if no file name is entered and assumes the creation of a new document. Otherwise, the designated file is opened for input in line 450. Lines 460-490 then input 48 lines, or everything up to the end of the file (EOF), whichever comes first, and puts them in the X$() array.

Now we are ready to edit the text. PG is the variable that keeps track of the page number (1 or 2), and tells the Transmit Page Subroutine (800-890) and the Display Page Subroutine (900-1020) which page they are dealing with.

Lines 500-670 contain the main body of the program and directs the flow of things from here on. Line 520 tells the Display Page Sub. to display page one. The function key labels are put on the 25th line as part of this subroutine. Now, except for certain circumstances defined in lines 540-590 or unless one of the labeled function keys is pressed, the computer will react almost like the OFF-LINE key is locked down.

Line 530 accepts one character from the keyboard. This character is scrutinized by lines 540-560. If it is not a Line feed CHR$(10) or a Tab CHR$(9) or a Delete CHR$(127), then it is printed to the screen as is.

If the character is a Line Feed, it is not printed and the program goes back to line 530 and gets the next character. The effect of this is to disable the Line Feed key.

If the character is a Tab, then the Escape "C" (cursor foward) sequence is printed eight times by line 550. This is done because when BASIC prints a Tab character it actually prints eight spaces which would in effect erase any text that was passed over. The cursor forward function does not.

If the Delete key is pressed, line 560 prints the Escape "D" (cursor backward) sequence eight times. Thus, the Delete key is used as a "backwards" Tab. (Finally, a good use for that key!)

After the character is printed, lines 580-600 check to see if it is one of three more possible special case characters.

If the printed character was a Return CHR$(13), then line 580 calls the Check for Bottom of Page Sub. This subroutine contains the Cursor Position Report routine. It checks to see if the cursor is at the bottom of the screen (row 24+31=55). If not, a Line Feed character is printed to move the cursor down one row. Otherwise, the bell is sounded to inform the user that the cursor is at the bottom and no Line Feed is printed, thus keeping the top row from scrolling off the screen.

If the last printed character was a Backspace CHR$(8), then line 590 prints a space and then another Backspace. The overall effect is to back up and erase the last character when the BACK-SPACE key is pressed.

Finally, if the last character was not an Escape CHR$(27) or E$, which was defined at the beginning of the program, the program loops back to 530 and waits for the next key to be pressed.

If, however, the last character was an Escape, then the program thinks perhaps a function key has been pressed. When a function key is pressed it sends two characters. The first is an Escape. The second is a special letter that is different for each of the function keys. If you are unfamiliar with this, check the Operation Manual for the escape codes sent by all the different function keys. This program detects (acknowledges) five of the function keys and directs program flow accordingly.

Function Key	Escape Code
f1	ESC S
f3	ESC U
Blue	ESC P
Red	ESC Q
White (Grey)	ESC R

When a function key is pressed, line 530 picks up the Escape

character. Then, the IF statements in 540-600 are processed and line 610 catches the second character. Pretty fast, isn't it! This second character is then analyzed in lines 620-660, to see if it belongs to one of the five function keys that we have designated for special purposes. If not, then the program goes back to line 570 and prints the character.

If the Blue (Output) key was pressed, the Transmit Page Sub. is called to save any changes that were made to the text on screen. Then the Output Text sub. is called. Notice the POKE statements in this routine. When running CP/M, location 3 in memory is what is known as the I/O byte. I prefer to call it the traffic cop. It's purpose is to direct the flow of data between the different ports. There have been some good articles in past issues of REMark on what values to POKE at this location to obtain different results. Here the value 105 is used to intercept characters being sent to the printer (LPRINT statement in line 1180) and direct them to the screen instead. (The value 169 is normally here if the system is setup for a printer.) Depending on the user's choice of screen or printer, the I/O byte is set accordingly and the X$() array is listed. If the output was made to the screen, then line 1200 gives the option of printing it, assuming it looked good on the screen. Otherwise, control is sent back to line 500 and into the edit mode on page one once again.

If the Red (Save) key was pressed, then the Transmit Page Sub. is called to update the X$() array once again. Then the Save File Sub. is called. This routine still remembers the name of the file that was entered when the program was started. If no file name was specified, line 1330 requests a name for the new file. If a file was input at the beginning, then line 1340 lets you know that saving under the same file name will overwrite what is currently in that file and asks if that is Ok. If line 1350 doesn't get a Yes (Y or y) answer, the program loops back to 1330 so a different file name can be entered. Then lines 1360-1410 open the file for output and writes the contents of the X$() array onto the disk. The program then returns to the edit mode on page one. Notice line 1370. I always like to include a line such as this whenever writing to a disk to let the user know just what is happening.

If the White (chg page) key is pressed, the Transmit Page Sub. is first called to update the X$() array, and then the Display Pages Sub. is called to display the other page on the screen. Line 880 toggles the page number variable (PG) back and forth between 1 & 2. Notice line 1010 of the Display Pages Sub. Besides printing P$ (the labels) on the 25th line, it also prints FRE("") at the far right of the line. This serves two purposes. It forces BASIC to do it's housekeeping each time this subroutine is called, and it lets the user know how much space is still available. Memory should be no problem, as long as the housekeeping is kept up. However, if this program were modified (as it easily could be) to use as many pages as needed, then having the available memory on screen would be a necessity.

If the f3 (Toggle Pad) key is pressed, the Toggle Keypad Sub. is called. This routine toggles the keypad back and forth between normal (numbers) and shifted (cursor control, etc.) modes. Just a handy added attraction. The variable KP lets the program keep track of which mode the keypad is currently in. Remember that when the keypad is in the shifted mode, each key sends two character escape sequences. For example, the HOME key sends the sequence ESC H. Since the program is not intercepting these codes, they are sent to the terminal and make the cursor react as it should. Due the inherent slowness of BASIC (most "high level" languages are slow), however, things will not go as planned if the REPEAT key is used with these keys.

Pressing the f1 (Quit) key causes line 660 to set the width back to normal, reset the I/O byte, shut off the 25th line, clear the screen, put the keypad back into it's normal (numbers) mode, and then Stop. Now all the modes are set to normal and everything should be as it was when we started. This should be done at the end of any program that has set things to other than normal. The GOTO 500, after the STOP instruction, is just a tricky little thing that I include in some programs. Issuing the CONT (continue) command after the program has stopped will restart the program at line 500 with all the X$() array contents still intact, just in case you really didn't want to quit after all.

A couple of other notes about the program. Notice that the WIDTH was set to 255 while in the edit mode. The default width setting (screen width) for MBASIC is 80. This means that after 80 characters are printed to the screen, even if the semicolon was used in the print statement, a carriage return/line feed sequence is sent. The purpose of this is to cause the text to wrap around onto the next line and not get printed way off in space to the right of the screen. Since this program is taking control of the cursor position into it's own hands, it doesn't want BASIC tossing in a CR/LF whenever it feels like it. Remember, BASIC would be counting all those non-printable cursor control Escape sequences too, and may decide to wrap around at what looks to be the middle of a line, since it had counted 80 characters. The WIDTH 255 defeats the character counting process and causes BASIC to assume an infinite screen width.

Notice also that the cursor is turned off during the Display Pages and Output Text subroutines when a lot of text is being printed to the screen. This keeps it from darting and flashing all over the screen while the screen is being filled. Whenever the cursor is turned off for such a purpose, it should always be turned back on when the routine is finished, as the user tends to feel lost without it.

Well, that should just about cover it. The explanation was longer than the program, so I guess everything is about normal! If you are still in the learning process (and who isn't) and some of the things we have covered here are new to you, taking the time to enter ED.BAS into your computer should prove to further your understanding of how MBASIC and your '89 work. There are many ways the program can be expanded upon and I invite you to do so. Experimenting with and improving upon a program is the best way I know of to improve one's grasp of the fine art of computer programming.

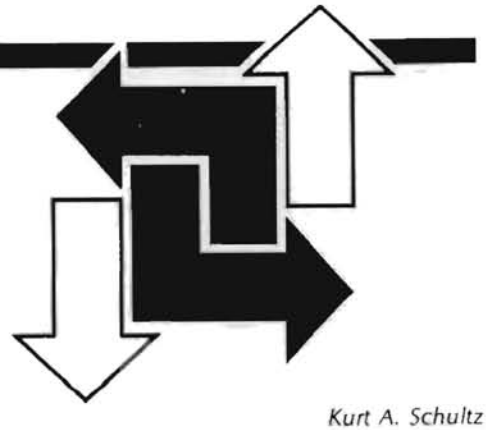
Have fun! And keep those '89s a-hummin'.....

```
10 ' ED.BAS              < Public Domain >
20 '
30 ' *** Example of a BASIC language text editor ***
                                      Jim Feasel 2/15/84
40 '                                  1121 ECR 16
50 ' For H/Z-89, Microsoft Basic 80, CP/M
                                      Tiffin Ohio 44883
60 '
70 'Usage Instructions
75 '
80 ' When in edit mode (when labels are on bottom line)
   all keys except
90 ' labeled function keys and Delete key respond as if
   Off-Line key is
100 ' depressed.
101 '
110 '   Delete key causes a backward Tab
120 '   White(gray) key toggles between pages 1 & 2
130 '   Red key saves text to disk file
140 '   Blue key outputs text to Printer or screen (use
      Ctl S to stop scroll)
150 '   f3 key toggles keypad to shifted mode and
160 '
170 '   Text is stored into memory exactly as displayed
      on screen
180 '
190 '
```

```
200 E$=CHR$(27)                 ' Escape
210 CLS$=E$+"E"                 ' Clear screen
220 CH$=E$+"H"                  ' Cursor home
230 BELL$=CHR$(7)               ' Ring bell
240 TP$=E$+"#"                  ' Transmit page
250 CRPT$=E$+"n"                ' Cursor position report
260 C25$=E$+"x1"+E$+"Y8 "       ' Cursor to 25th line
270 OFF25$=E$+"y1"              ' Erase 25th line
280 COFF$=E$+"x5"               ' Cursor off
290 CON$=E$+"y5"                ' Cursor on
300 DIM X$(48)
310 P$="f1=Quit  f3=Toggle Pad  Blue=Output  Red=Save
        White=chg page"
320 '
400 'Input file to edit
410 '
420 PRINT CLS$:FOR I=1 TO 10:PRINT:NEXT
430 INPUT
    "Enter file name to edit or RETURN for new file ---> ",
     FILE$
440 IF FILE$="" THEN 500
450 OPEN "I",1,FILE$
460 I=1
470 IF EOF(1) OR I>48 THEN CLOSE:GOTO 500
480 LINE INPUT #1,X$(I)
490 I=I+1:GOTO 470
495 '
500 'Edit and/or enter text
510 '
520 PG=1:GOSUB 900
530 I$=INPUT$(1)
540 IF I$=CHR$(10) THEN 530
550 IF I$=CHR$(9) THEN FOR I=1 TO 8:PRINT E$"C";:
    NEXT:GOTO 530
560 IF I$=CHR$(127) THEN FOR I=1 TO 8:PRINT E$"D";:
    NEXT:GOTO 530
570 PRINT I$;
580 IF I$=CHR$(13) THEN GOSUB 720:GOTO 530
590 IF I$=CHR$(8) THEN PRINT " ";I$;
600 IF I$<>E$ THEN 530
610 I$=INPUT$(1)
620 IF I$="P" THEN GOSUB 820:GOSUB 1100:GOTO 500
630 IF I$="Q" THEN GOSUB 820:GOSUB 1300:GOTO 500
640 IF I$="R" THEN GOSUB 820:GOSUB 900:GOTO 530
650 IF I$="U" THEN GOSUB 1500:GOTO 530
660 IF I$="S" THEN WIDTH 80:POKE 3,169:PRINT OFF25$;
    CLS$;E$"y6":STOP:GOTO 500
670 GOTO 570
680 '
700 ' Check for bottom of page subroutine
710 '
720 PRINT CRPT$;
730 I$=INPUT$(3)
740 IF LEFT$(I$,1)<>"Y" THEN I$=RIGHT$(I$,2)+INPUT$(1)
750 IF ASC(MID$(I$,2,1))<55 THEN PRINT CHR$(10);:
    ELSE PRINT BELL$;
760 RETURN
770 '
800 'Transmit page subroutine
810 '
820 IF PG=1 THEN X=1
830 IF PG=2 THEN X=25
840 PRINT TP$;
850 FOR I=X TO X+23
860  X$(I)=INPUT$(79)
870 NEXT
880 IF PG=1 THEN PG=2 ELSE PG=1
890 RETURN
895 '
900 ' Display pages subroutine
910 '
920 IF PG=2 THEN X=25:GOTO 950
930 IF PG=1 THEN X=1
950 PRINT CLS$;COFF$;
960 WIDTH 80
970 FOR I=X TO X+23
980  PRINT X$(I);
990 NEXT
1000 WIDTH 255
1010 PRINT C25$;P$;FRE("");CON$:PRINT CH$;
1020 RETURN
1030 '
1100 'Output text
1110 '
1120 PRINT CLS$;OFF25$::FOR I=1 TO 10:PRINT:NEXT
1130 INPUT "Output to Screen or Printer (S/P) ";I$
1140 IF I$<>"P" THEN POKE 3,105:GOTO 1170
1150 POKE 3,169
1160 PRINT CLS$;"Printing.....";:GOTO 1180
1170 PRINT CLS$;COFF$
1180 FOR I=1 TO 48:LPRINT X$(I):NEXT
1190 IF I$="P" THEN 1220
1200 PRINT CON$:INPUT "Ready to Print now? (Y/N) ",I$
1210 IF I$="Y" THEN 1150
1220 RETURN
1230 '
1300 ' Save File Subroutine
1310 '
1320 PRINT CLS$;OFF25$:FOR I=1 TO 10:PRINT:NEXT
1330 IF FILE$="" THEN INPUT
     "Enter new file name please ---> ",FILE$:GOTO 1360
1340 INPUT "Overwrite existing file? (Y/N) ",I$
1350 IF I$<>"Y" OR I$<>"y" THEN FILE$="":GOTO 1330
1360 OPEN "O",1,FILE$
1370 PRINT:PRINT "Saving as ---> ";FILE$
1380 FOR I=1 TO 48
1390  PRINT #1,X$(I)
1400 NEXT
1410 CLOSE
1420 RETURN
1430 '
1500 'Toggle Keypad subroutine
1510 '
1520 IF KP=0 THEN PRINT E$"x6";:KP=1:GOTO 1540
1530 IF KP=1 THEN PRINT E$"y6";:KP=0
1540 RETURN
```

A Problem and Its Solution

(For the special attention of MBASIC programmers...)



Kurt A. Schultz
115-1 Roxanne Court
Walnut Creek, CA 94596

The Problem

After having written, looked inside of and played so many games, I've noticed that there is a recurring problem with the use of random numbers. Most authors who deal with MBASIC understand how to fetch and manipulate random numbers. Many, however, don't understand how to avoid starting at the same location in the random number table.

One of the games I got from HUG is a good example. This game is annoying because it always puts the targets in the same series of patterns. Every time you call the game from the operating system, it always starts with the same "randomly-derived" target pattern, followed by the next pattern, and the next and so on. I learned how to do very well on the first eight or so, then I got bored.

After all, what good is a game that has predictable random numbers? An otherwise well written program can be consigned to the dust-gathering rack, just because it always plays the same game. Fear not, fellow BASIC programmers, if your random numbers seem to be stuck in a rut. Gather ye 'round and take notes...

The Background

The only 'constant' random event to be found by nearly all MBASIC programs is the length of time that it takes the user to respond at the keyboard. This is very nearly a true random, as the speed that humans are calibrated for is of little significance when compared to that of an incrementing loop. By the time that your eye has registered that the loop value has been printed on the screen, it's already a couple of steps past you. The assurance of randomness (sufficient for practical purposes, anyway) can be learned by anyone who tries to stop the loop in the same place twice in a row.

So, if you could stop a fast-running loop, snatch the value and use that value as the "seed" for the random number generator, then you should be able to run the same program twice (not twice in a row, but two "cold-starts" in a row) and yet have different random-controlled values.

The Solution

The segment of code in **Figure 1** is the approximate solution I use in nearly every game that needs it.

The Walk-Through

Line 10 initializes the loop variable to zero and askes a question. This question doesn't have to be the one that I've used here, but the one used does seem to occur frequently.

Line 20 increments the loop, tests to see if the value is within the acceptable range for that type of variable and if not, resets the loop to zero, so that it can go again.

Line 30 fetches the input from the keyboard. If there is nothing waiting as input (which will be the normal situation), the INKEY$ function returns a null (ie: the information contained between the quote marks in the expression: " "), and if so, sends control back to the previous line for another increment (thus creating the loop). If, on the other hand, there was some kind of input waiting at the keyboard, the INKEY$ function will transfer it to A$. A$ will then be tested for null-ness, fail and then fall through to the next statement (thus breaking out of the loop).

Line 40 tests the keyboard inputs for extraneous input. Any garbage detected will restart the loop from where it was exited. Acceptable inputs will completely stop the loop activity.

Line 50 tests accepted input to see if the subroutine which displays the instructions should be executed. This subroutine can be anywhere, just be sure that you're referring to its line number, not something else.

(In order to see the value of the loop counter at the exit point, add the statement: 55 PRINT RANNUM%)

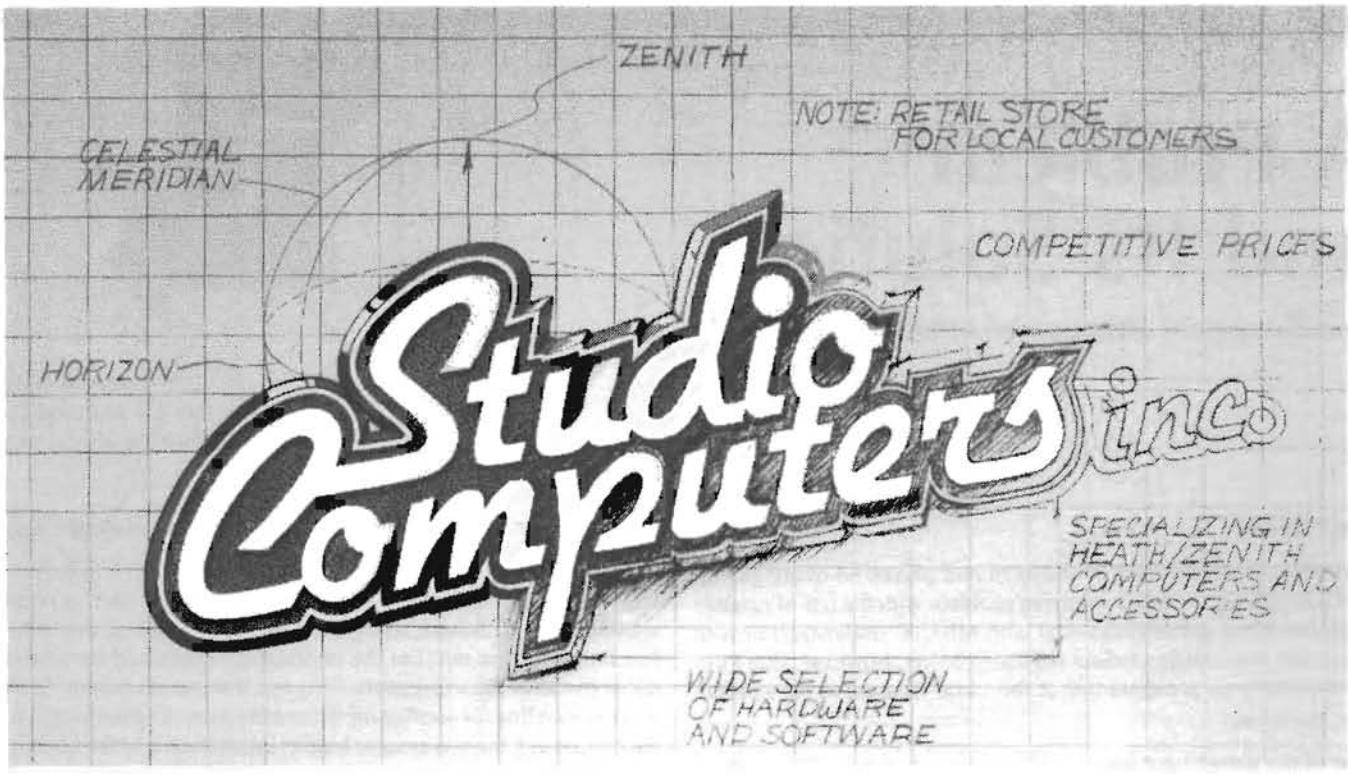
Line 60 is the whole reason that we've gone through all of the steps above. Whether or not the instructions were displayed, when line 50 is completed, line 60 will take the last value that the loop (ie: "RANNUM%") contained, and use that as the seed for the random number generator.

The Wrap-up

It is possible (in theory) to get the same results repeatedly with this scheme, if the user pre-loads their response (ie: answers the question before it's asked). This is to be avoided, if possible.

This segment of code could go anywhere before the first occurrence of an "RND" statement, but I find that it fits in best near the beginning, just after the DIMensioning declarations and intializing statements.

```
10 RANNUM%=0:PRINT "Do you want Instructions?";
20 RANNUM%=RANNUM%+1:IF RANNUM%>32764 THEN RANNUM%=0
30 A$=INKEY$:IF A$="" THEN 20
40 IF NOT(A$="Y" OR A$="N" OR A$="y" OR A$="n") THEN 20
50 IF A$="Y" OR A$="y" THEN GOSUB 10000:REM print instructions
60 RANDOMIZE RANNUM%
    .
    .
10000 PRINT "The Instructions for ...
    .
    .
    . RETURN
```



DESIGNED TO BRING YOU THE BEST IN PRICE AND PERFORMANCE

HERE ARE JUST A FEW OF THE ITEMS WE CARRY...

DOT-MATRIX GRAPHICS PRINTER

MPI Printmate 99g	$429
MPI Printmate 150 B1	$715
MPI Printmate 150 A1 w/keypad	$815
DataProducts 8050, 200 cps, Color	$1516

DISK DRIVES & ENCLOSURES

Tandon TM-100-1 48tpi, SS,DD	$189
Tandon TM-100-2 48tpi, SS,DD	$245
Tandon TM-101-4 48tpi, SS,DD	$295
Tandon TM-848-2e 8" DS,DD thinline	$419
Qume QT-592 full-height 96tpi, DS,DD	$349
Mitsubishi 1/2 height, 96tpi, DS,DD	$275
SC-100 Single 5 1/4" vertical enclosure	$75
SC-200 Dual 5 1/4" vertical enclosure	$100
SC-800 Single/Dual 8" vertical enclosure	$195
SC-900 Dual 5 1/4" horizontal enclosure	$125

MODEMS

Hayes 300 baud Smartmodem	$209
Hayes 300/1200 baud Smartmodem	$495
Novation 300/1200 External Smartcat Plus	$375
Novation 300/1200 Smartcat Plus for PC	$375
U.S. Robotics S100 300/1200 baud modem	$375

ZENITH DATA SYSTEMS

Call us for SUPER Z100 & Z150/160 PC prices, as well as accessory hardware and software.

EIGHT-INCH DRIVE SYSTEMS

Complete Dual 8" Thinline disk drive system for Z100 computers. Includes all required components and documentation. Minor assembly required.
Special Sale Price $1019

PRO DRIVER

A professionally written MS-DOS modem communication package for Z100 & Z150 computers. Features Compuserve B, Xmodem and Xon-Xoff protocols. Plus 32 auto-logons, user-defined color parameters, easy to read menus, on-line help displays, usage log reporter and more. $49

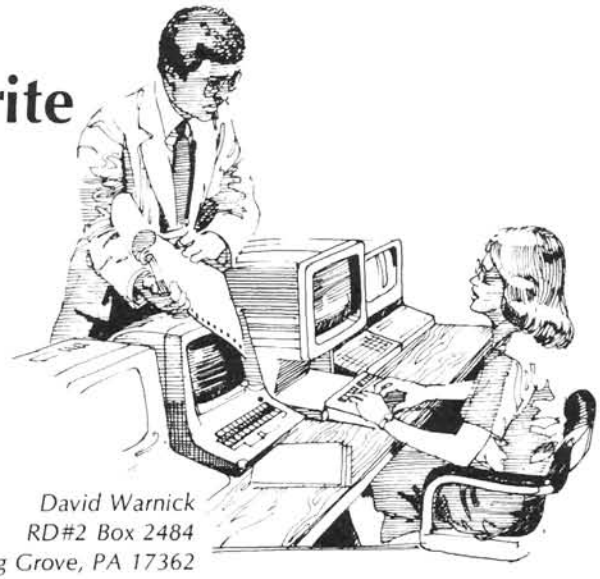
Please add 2% of the total order (minimum $2) for shipping, insurance and handling for anywhere in the continental U.S.

ZENITH data systems

999 South Adams
Birmingham, MI 48011
Phone (313)-645-5365

For a free copy of our 24 page illustrated catalog describing all our products, use the reader service card or call us anytime from 9:30 to 5:30 Eastern time.

Part 6 - We're Ready To Write Our Own Data Base



Random Files

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

Over the past 5 months we've covered a lot of information in developing our random file handling. We've discussed sorting, key files, and the details of what goes on when we read, write, or make changes to random files. At times, I pointed out that certain operations were my personal preference and showed you alternative ways to do the same thing. It is important to note that you must decide which way you prefer to do things, and write your programs that way. Don't accept a bit of programming you don't like just because someone put it in a magazine. The authors of those articles (this one included) are just like you, and they have their own little hangups. Learn by reading their work, but if part of their program doesn't suit you, rewrite it to your own tastes. I've tried to make that easier in this series by using the modular programming technique, so have at it.

That's what this month's article is all about. Changing the program we've developed to make it usable and convenient to you. The first step we'll take is to write the program in a smaller space. This is not to be taken too lightly. It must be done carefully, and should be done after the complete program has been written and tested. We'll perform the size-reducing operation by looking for routines which are written at more than one place in the program. We will then write the duplicated routine in a separate area, just as though it were another module. Then we'll send program execution to that sub-routine with a GOSUB when we need it. The theory behind all this is that a sub-routine written once is smaller than the same series of programming lines written several times. By reducing the size of our program, we are taking up less space in RAM. By taking up less space in RAM, we allow more space to handle and sort the files we're working on. This in turn allows us to create and manipulate larger files with the resources available. Buying and installing more RAM is neat. It's expensive, but it's neat. A better way of doing things is to keep your program compact where possible. This is especially important when your machine is at it's maximum memory capacity.

I'm not advocating cutting out frills or making a program any less friendly. All I'm saying is cut it's size by a conscious effort of eliminating as many repetitive lines of programming as is practical. Again, do this after you know your program works. You'll be adding more GOSUB's, and if you've stayed with me this far into the series, I'm sure you're well aware of the frustrations they add when trying to debug a program.

I'll show you three sets of subroutines which I combined in the random file handling program we've been working on. With the program, as we left it last month, loaded in my H89 with 64K RAM there were 14276 bytes of RAM available for the actual file handling operations. After extracting these three sub-routines from each module in which they appeared and adding GOSUB's to branch to them for execution when needed, I was able to use 15926 bytes of RAM for these operations. That's an additional 1650 bytes of RAM, and it's free. All you have to do is find where it was wasted in programming and eliminate that waste. What does this extra RAM mean, and what does it do for us? If we were sorting a file by using a keyfile, and if the key field was 30-bytes long, we would be able to add and sort 51 more records to our random file than we could before the size reduction effort. And that's without running out of RAM when working on or sorting the file.

Before we eliminate the duplicate lines, I must point out that there are other alternatives to reduce program size, or at least to reduce the amount of RAM required at any time while we're running the program. We could have written the menu as our program without any subroutines. The menu could then have called up other programs when needed, and each of the modules could have been written as a separate program. It's not as difficult as it sounds and I hope to cover that kind of operation in a future article. The other alternative is to compile the program. While the program would be longer than the original MBasic program after compilation, the fact that the MBasic interpreter would not be needed to run the compiled version would save about 12K of RAM. We'd also experience much faster execution times. However, there are limitations to what can and what cannot be compiled. The file-handling program we just developed cannot be compiled without some changes. This too will have to wait for a future article if fair treatment is to be given to its explanation.

Now I'll stop babbling and get on with reducing the size of our program. The first repetition of programming we find occurs when

we print part of the CRT screen. Get out a copy of your completed program and check the following lines:

2010 thru 2080
5020 thru 5090
8110 thru 8180
11020 thru 11090

In each case, the month, date, and year are added to the display. We can put the same code after all other programming at line 30000 as follows:

```
30000 PRINT FNCA$(9,21); "MONTH"
30010 PRINT FNCA$(11,21); "DATE"
30020 PRINT FNCA$(13,21); "YEAR"
30030 PRINT GM$
30040 PRINT FNCA$(10,27);"zz"
30050 PRINT FNCA$(12,27);"zz"
30060 PRINT FNCA$(14,27);"zz"
30070 PRINT GO$
30080 RETURN
```

When this has been done, go back and remove all the lines we specified above. Next, add instructions at the first line removed in each case, to GOSUB 30000. It should look like this:

```
2010 GOSUB 30000
5020 GOSUB 30000
8110 GOSUB 30000
11020 GOSUB 30000
```

You should have noticed that the first area removed, lines 2010 thru 2080, also included programming to print the word "VALUE". Therefore, we'll have to add two more lines of code to make it work.

```
2070 PRINT FNCA$(15,21);"VALUE"
2080 PRINT GM$
```

Make these changes to your program and run it. Remember to test each change as it is made. That way, if there is a problem with a change, you'll know where to look to correct it. If several changes had been made without testing after each one, a small problem which should have been easily found can become very frustrating. This series is intended to be educational, but it and all of programming should be fun. Don't make it difficult on yourself by skipping important steps in testing.

You're back, so the tests you made must have shown that the changes worked. The second change we'll make is a small one. In each case we'll replace a single line with a GOSUB. It saves space because the line is a long one. At three different locations in our program we use the red key to exit from a sub- routine. This occurs on lines 2110, 5100, and 11100. Delete these lines and add their programming instructions at line 30500.

```
30500 PRINT FNCA$(24,21);
  "PRESS THE RED KEY TO RETURN TO THE MENU";FNCA$(1,1)
30510 RETURN
```

Replace the removed lines with GOSUB instructions.

```
2110 GOSUB 30500
8060 GOSUB 30500
11100 GOSUB 30500
```

It's time to make another test. To test the latest change, you'll have to enter each of the sub-routines which uses the red key to exit to the menu. The one at line 2000 is ADDITION so enter that option from the menu, and use the red key to exit the option and return to the menu. Do the same for line 8000, DELETE, and line 11000, LOOKUP. Your flow charts should have pointed you to these routines. All should have gone well, and you should be ready to make the last and the biggest of the changes.

This change also involves some of the screen setup instructions we provided in our program. The lines to check for are:

2120 thru 2270
5110 thru 5250
11110 thru 11250
8210 thru 8340

We'll replace these lines with similar programming at line 31000. Operation will be very slightly changed as the wording will be the same no matter what operation is performed. This may well be one of those cases where you feel the savings in space is not worth the changed display. Try it and decide for yourself. Delete the lines above and enter the following lines of programming.

```
2120 GOSUB 31000
5110 GOSUB 31000
8210 GOSUB 31000
11110 GOSUB 31000

31000 PRINT FNCA$(18,21);
   "ENTER THE MONTH AS 2 DIGITS (01-12)"
31010 PRINT FNCA$(9,27);
31020 M$=INPUT$(2)
31030 IF M$=CHAR$(27)+"Q" GOTO 1000
31040 PRINT M$
31050 PRINT FNCA$(18,1);EE$
31060 PRINT FNCA$(18,21);
   "ENTER THE DATE AS 2 DIGITS (01-31)"
31070 PRINT FNCA$(11,27);
31080 D$=INPUT$(2)
31090 PRINT D$
31100 PRINT FNCA$(18,1);EL$
31110 PRINT FNCA$(18,21);
   "ENTER THE LAST 2 DIGITS OF THE YEAR"
31120 PRINT FNCA$(13,27);
31130 Y$=INPUT$(2)
31140 PRINT Y$
31150 RETURN
```

Make a test of the latest change. When you've got it working, it would be a good idea to copy the latest version to your backup disk. If you lose this program, you've lost a lot of work. Don't take any chances.

It's your turn now. Print yourself a listing of the program and look it over. There are several other repetitious instructions which can be combined as a separate sub-routine, and referred to by a GOSUB at their old location. Try it. There's nothing to lose. (You did make that backup copy, didn't you?) You can learn from this, and your own personalized version of this program will be the only one like it in the world.

Well, we've done it. We've created a data base handling program and a data base to go with it. We can add, change, delete, and lookup anything we want within that data base. It's even possible to print a nifty chart of the information contained in our file. But, you say, I'm a ham radio operator and I want to keep a file of the countries I've worked and the bands I've worked them on so I can keep track of my award program progress. Or, I'd really like to catalog my stamp collection so I can look up what I've got, or print a listing by country to take to the local flea market or stamp show. How about, I need an inventory recording program, or a name and address file to be used for a mailing list? The answer to all the above and to just about

anything else you can think of is 'Sure we can do it'. We have the basics down, and most of the work has already been done over the past 5 months. All we have to do is make a few small changes to the program we've already written, and PRESTO, a new data base.

Let's look at what must be changed. A data base is a collection of records in a file. The difference between two data bases is merely a difference in the format of those records and in what information is contained in them.

You will recall from the beginning of the series on random files that we discussed records. We said that each record is made up of several pieces of information. These pieces of information were called fields. Alone, the fields were meaningless, but when considered together as a record they conveyed information. The difference between data bases is a difference in the information we wish to store. A true data base may be used by several programs, none of which use all the fields stored in each record. Our data base is specialized and tailored to our specific application.

Two MBASIC statements set up the size and the configuration of the random file records. They are the OPEN and the FIELD statements. To write them you must first decide what information will be stored in each record, and how much space that information will require. The examples for our data bases requested above could be:

HAM RADIO

Field #	Field Contents	# of characters
1	Call Letters	10
2	Country Name	15
3	Date Contacted	6
4	Band Used	2
5	Comments	25
	TOTAL	58

```
OPEN 'R',#1,DXCARDS,58
FIELD #1,10 AS A$,15 AS B$,6 AS C$,2 AS D$,25 AS E$
```

STAMP COLLECTION

Field #	Field Contents	# of Characters
1	Stamp Number	4
2	Stamp Condition	1
3	Country	15
4	Value	4
	TOTAL	24

```
OPEN 'R',#1,STAMPS,24
FIELD #1,4 AS A$,1 AS B$,15 AS C$,4 AS D$
```

INVENTORY

Field #	Field Contents	# of Characters
1	Item Name	25
2	Item Number	10
3	Qty On Hand	4
4	Min Order Qty	4
5	Reorder When Qty Is	4
6	Order Qty	4
7	Last Wholesale	6
8	Selling Price	6
	TOTAL	63

You write this one. It's just as easy as the others. The only difference is that the field statement is a little bit longer. The whole point to all of this is that you can set up your own data file to contain any information you like. The first step is to decide what you want to store in the file. The second step is to decide how long you want to make each field for the information it will contain. Don't waste space by allowing ridiculously long fields when they're not needed, and don't cut yourself short. These two steps are the hardest part of the whole process of setting up a data base. When they're complete you can modify the OPEN and FIELD in our program to accommodate your records.

The next step is not difficult, but it does require patience and care. You must lay out the CRT screen for your records. We've done this several times in past articles. A CRT layout form is an invaluable aid for this operation. Computer shops and mail order houses have them or you can lay out a grid 80 by 25 to do your workups. Decide what you want on the screen and print it. Then enter operator prompts and take the inputs for each operation. You can use the scrolling screen most often used, but remember how much nicer it was to work with our data files when everything had a place on the CRT and was in that place. You have the ability to put that little extra effort into your program, and it's well worth it.

You may or may not require a routine which permits you to name a file from the console keyboard. If you will always work on the same file, enter it into all the OPEN statements and do away with the NEWNAME module beginning at line 14000. Don't forget to give names to the files used as keyfiles if you do this. Take your time. If you're unsure of yourself, go back over those past articles and review how we built our program the first time. Don't be afraid to try your ideas, and NEVER, NEVER, NEVER give up.

Another area affected by our new record format is the PUT and GET statements. Before we can PUT any of our new records, we must have an LSET or an RSET command for each field. If the data to be put into the record is numeric, it must have an MKI$, MKS$, or an MKD$ command to convert it from a number to a string. If we use any of these commands, then the GET statement will have to be followed by a CVI$, CVS$, or a CVD$ command to convert the string back to a number. It's not difficult, but there are rules, and they must be followed to the letter. Computers are stupid but rigidly obedient servants. They must be told exactly what you want them to do. If they are, they'll perform time and again for you, tirelessly doing what you ask.

The final area of the program you'll have to change to fit your own special needs is the part which gives the required data to the sort routine. In our program, we read a key field (the field containing the data we wanted to sort on) and its record number into an array. There was one entry for every record, then we did our sorting. If you wish to sort on the data contained in a single field, all you have to do is make sure your revised program selects the correct field to be the key for the array to be sorted.

Suppose, however, that you were running a mailing list for your club. It's not enough to sort by the last name of each member. You must sort by last name, first name, and middle initial. This isn't even enough. You found that there are advantages to sorting by zip code, too. (Check your local post office for lower mailing rates on presorted mail.) You must now find a way to sort on multiple fields. Let's assume that you've assigned information from the record to the following variables when you used GET.

LN$ = Last Name

FN$ = First Name
MI$ = Middle Initial
ZIP$ = Zip Code

The order of importance of each field must be determined. For our application we want all entries sorted by zip code. Within each zip code area, we want all entries sorted by last name, first name, and middle initial. To do this we concatenate (string together) each of the fields in their order of importance. For this example we could use:

SORT$=ZIP$ + LN$ + FN$ + MI$

We would enter SORT$ into an array and its record number into an array. This would be done for each record within the file. Then the sorting would be done and a key file generated. If we began with the first item in the key field and printed to the last, the required list would be sorted as we wanted.

Don't forget, we could sort with zip codes as above for mailing, and then sort again by just last name, first name, middle initial to create a separate key file for an alphabetical membership list. Use those key files for all they're worth.

Another kind of key field could also have been used. There could be a one-character field with an entry of O for officer of the club, M for member, L for life member, etc. Then to print just a list of the life members, read each record, check for an L in that field, and if it exists print the record. You can play all kinds of tricks with these things if you just give it some thought.

That's it. You can now build your own data base for whatever purpose you may have. I told you at the beginning of this series that your computer is a powerful tool, and that we'd unlock some of the secrets of that power. We have done it. Now it's up to you to turn it loose on your needs. Make data storage and retrieval simple. As this completes random files, we'll be able to take a look at sequential files next. They do have some important uses, and we'll begin to explore that area.

See you next month.

Make Your H89 Talk

W.C. Parham
R#5 Box 84
LaGrange, IN 46761

After being an H-89 user for about three years and seeing all the low cost computers talking up a storm, and I might say quite easily, I thought it was time to make my H-89 talk.

After looking at several different types of speech synthesizers with many different prices, I decided on the ECHO from Street Electronics Corporation, 1140 Mark Ave., Carpinteria, CA 93103. The ECHO comes in three models. My choice was the ECHO-GP because this model is interfaced via the RS-232 port, which I already have installed in my H-89. A call to Street Electronics and three days later I had my ECHO-GP.

Upon receiving my ECHO I inspected the packaging and found it to be execellent. The unit comes complete with an RS-232 type cable and power supply. The unit also has a limited one year warranty. The manual covers many different types of hardware and software configurations. The Z-100 jack (J1) wiring is included, but not the wiring or software for the H-89 serial port.

The manual also covers the different modes of speech. The Text-to-Speech Mode uses many rules to guide it in correctly pronouncing your text. English is so full of exceptions to these rules, such as compound words, that it is simply not feasible to make perfect unlimited vocabulary. The Phoneme Mode sends phonemes to generate speech. Phonemes are the smallest distinguishable sound units of a language. This allows you to have words pronounced exactly as you wish.

Making the H-89 talk turned into quite a challenge. The problem came when I realized that the 8250 UART chip in the H-89 would have to be initialized to send data out to a port when using MBASIC. The port number, baud rate, start and stop bits and parity had to be set. All this information can be found in the Heathkit manual pages 13-1 to 13-9. Even with this manual I found programming the 8250 UART a challenge. Below is my solution to making the H-89 talk.

Listing 1 is a program I named TALK.BAS. In this program the 8250 UART is initialized and you are given the choice either to send words out in word form or letter form. Notice that a control E is sent out before and after each group of words. Line 330 of TALK.BAS sends control E both times.

Listing 2 is an educational program I named SPELL2.BAS. This program also initializes the 8250 UART and lets the user input a list of words to a file then take a test over these words. The program uses ECHO to pronounce the words to the user and, if the user misspells the word, ECHO will spell the word correctly. Also listed below is the hardware wiring for the RS-232 plug at port number 330Q.

There are other features of the ECHO that I have not used, but I am sure that this will get our H-89's talking up a storm. Happy talking!

H-89 PORT 330Q(DTE)		ECHO
Pin # 2 (XMT)	to	Pin #5 (DATA IN)
Pin # 3 (RCD)	to	Pin #3 (DATA OUT)
Pin # 4 (RTS)	to	Pin #1 (STATUS IN)
Pin # 5 (CTS)	to	Pin #4 (STATUS OUT)
Pin # 6 (DSR)	to	Pin #4 (STATUS OUT)
PIN # 7 (GND)	to	Pin #2 (GND)

Listing 2

```
1 REM ********** SPELL2.BAS **********
2 REM ********** BY W.C.PARHAM **********
10 GOSUB 660:REM INIT PORT 330Q AT 9600 BAUD
20 C=0
30 DIM B$(50)
40 PRINT CHR$(27)+"E"
60 PRINT TAB(15)"0=TO EXIT"
70 PRINT TAB(15)"1=START NEW LIST OF WORDS"
80 PRINT TAB(15)"2=TO PRINT WORDS ON FILE"
90 PRINT TAB(15)"3=TO TEST YOURSELF"
100 PRINT TAB(15)"4=TO SEND WORDS TO PRINTER"
110 PRINT:PRINT:PRINT
120 INPUT "WHAT WOULD YOU LIKE TO DO ?";A
130 IF A=0 THEN END
140 IF A>=5 THEN  490
150 ON A GOTO 160,170,180,190
160 GOSUB 200:GOTO 40
170 GOSUB 350:GOTO 60
180 GOSUB 510:GOTO 60
190 GOSUB 1390: GOTO 40
200 INPUT
    "HOW MANY WORDS DO YOU WANT TO PUT IN
     (MAX NUMBER IS 50)";B
210 FOR N=0 TO B-1
220 INPUT "TYPE IN WORD";B$(N)
230 C=C+1
240 PRINT C
250 IF C=B THEN 280
260 NEXT N
270 REM PUT WORDS IN FILE SPELL.BAS
280 OPEN "O",#1,"SPELL.DAT"
290 FOR N=0 TO B-1
300 PRINT #1,B$(N)
310 NEXT N
320 CLOSE #1
330 PRINT N
340 RETURN
350 PRINT B
```

```
360 REM READ WORDS OUT OF FILE SPELL.DAT
370 OPEN "I",#1,"SPELL.DAT"
380 FOR N=0 TO 50
390 INPUT #1,B$(N)
400 IF EOF(1) THEN 420
410 NEXT N
420 CLOSE #1
430 REM PRINT WORDS ON FILE TO SCREEN
440 FOR N= 0 TO 50
450 PRINT B$(N) " ";
460 NEXT N
470 PRINT:PRINT:PRINT:PRINT:PRINT
480 RETURN
490 PRINT CHR$(27)+"p":PRINT
    "THAT IS NOT A VALID CHOICE"
500 FOR N=0 TO 700:NEXT N:PRINT CHR$(27)+"q":GOTO 40
510 REM SET UP FOR TEST WITH ECHO
520 PRINT CHR$(27)+"E"
530 PRINT TAB(10)"GET READY FOR THE QUIZ !"
540 OPEN "I",#1,"SPELL.DAT"
550 FOR N=0 TO 50
560 INPUT #1,B$(N)
570 PRINT CHR$(27)+"p":PRINT TAB(30)"SPELL THE WORD "
    :PRINT CHR$(27)+"q"
580 GOTO 940
590 INPUT "TYPE IN THE CORRECT SPELLING ";F$
600 IF F$=B$(N) THEN GOSUB 1020:FOR K=0 TO 300:NEXT K
610 IF F$<>B$(N) THEN GOSUB 1040
620 IF EOF(1) THEN GOTO 640
630 NEXT N
640 CLOSE #1
650 RETURN
660 LET P0=216
670 LET P1=P0+1
680 LET P3=P0+3
690 LET P4=P0+4
700 LET P5=P0+5
710 LET P7=P0
720 LET P8=P0+1
730 LET D1=128
740 LET D2=16
750 LET D3=3
760 D5=0
770 D4=12
780 D6=2
790 D7=1
800 D8=64
810 REM INIT PORT
820 OUT P3,0
830 OUT P1,0
840 OUT P4,D2
850 OUT P3,D1
860 OUT P7,D4:OUT P8,D5
870 OUT P3,D3
880 C1=INP(P0)
890 FOR I=1 TO 100:NEXT I
900 C1=INP(P0)
910 OUT P4,0
920 OUT P4,D6
930 RETURN
940 GOSUB 1200:GOSUB 1160
950 FOR I=1 TO LEN(B$(N))
960 C$=MID$(B$(N),I,1)
970 D=ASC(C$)
980 GOSUB 1120
990 NEXT I
1000 GOSUB 1200:GOSUB 1160
1010 GOTO 590
1020 PRINT CHR$(27)+"E":PRINT CHR$(27)+"p":PRINT:PRINT
     "GOOD JOB":PRINT CHR$(27)+"q"
1030 RETURN
1040 PRINT CHR$(27)+"E":PRINT:PRINT:PRINT
     "BETTER LUCK NEXT TIME !"
1045 GOSUB 1200:GOSUB 1160
1050 E$= "THE WORD SHOULD BE SPELLED"
1060 FOR I=1 TO LEN(E$)
1070 G$=MID$(E$,I,1)
1080 H=ASC(G$)
1090 GOSUB 1240
1100 NEXT I
1105 GOSUB 1200:GOSUB 1280:GOSUB 1320
1110 RETURN
1120 REM SUB TO OUTPUT WORD
1130 LET T=INP(P5) AND 32 :REM MASK FOR TRANS HOLD REG
1140 IF T=0 THEN GOTO 1130
1150 OUT P0,D :RETURN
1160 REM SUB TO SET WORD MODE
1170 LET T=INP(P5) AND 32
1180 IF T=0 THEN GOTO 1170
1190 OUT P0,87 :RETURN
1200 REM OUTPUT CONTROL E
1210 LET T=INP(P5) AND 32
1220 IF T=0 THEN GOTO 1210
1230 OUT P0,5:RETURN
1240 REM OUTPUT WRONG STATEMENT
1250 LET T=INP(P5) AND 32
1260 IF T=0 THEN 1250
1270 OUT P0,H:RETURN
1280 REM SET TO LETTER MODE
1290 LET T=INP(P5) AND 32
1300 IF T=0 THEN 1290
1310 OUT P0,76:RETURN
1320 REM OUTPUT CORRECT SPELLING
1330 FOR I=1 TO LEN(B$(N))
1340 J$=MID$(B$(N),I,1)
1350 D=ASC(J$)
1360 GOSUB 1120
1365 NEXT I
1370 FOR K=0 TO 1000:NEXT K
1380 RETURN
1390 REM READ FILE FOR PRINTER
1400 OPEN "I",#1,"SPELL.DAT"
1410 FOR N=0 TO 50
1420 INPUT #1,B$(N)
1425 LPRINT B$(N)
1430 IF EOF(1) THEN 1450
1440 NEXT N
1450 CLOSE #1
1460 RETURN
```

Listing 1

```
1 REM ************** TALK.BAS *****************
2 REM ************ BY W.C.PARHAM **************
10 REM INIT PORT 330Q AT 9600 BAUD
20 LET P0=216 : REM OCTAL PORT 330
30 LET P1=P0+1: REM INTERRUPT FLAG
40 LET P3=P0+3: REM LINE CONTROL REG.
50 LET P4=P0+4: REM MODEM CONTROL REG.
60 LET P5=P0+5: REM LINE STATUS
70 LET P7=P0  : REM BAUD RATE REG. LS
80 LET P8=P0+1: REM BAUD RATE REG. MS
90 LET D1=128 : REM DIVISOR LATCH ACCESS BIT
100 LET D2=16 : REM LOOPBACK BIT
110 LET D3=3  :
    REM CONFIGURATION (8BITS,1SB,NO PARITY)
120 LET D5=0  : REM BAUD RATE MS (9600)
130 LET D4=12 : REM BAUD RATE LS (9600)
140 LET D6=2  : REM RTS (RESET)
150 LET D7=1  : REM DTR
160 REM INIT PORT
170 OUT P3,0: REM CLEAR DLAB
180 OUT P1,0: REM DISABLE INTERRUPT
190 OUT P4,D2:REM SET IN LOOPBACK
200 OUT P3,D1:REM SET BAUD RATE
210 OUT P7,D4:OUT P8,D5
220 OUT P3,D3
230 C1=INP(P0)
240 FOR I=1 TO 100:NEXT I: REM DELAY TWO CHAR TIMES
250 C1=INP(P0)
260 OUT P4,0:REM TAKE OUT OF LOOP
270 REM LAST OF INIT PORT
280 OUT P4,D6
290 PRINT CHR$(27)+"E":REM CLEAR SCREEN
300 PRINT TAB(15)"1= TO PUT ECHO IN WORD MODE"
310 PRINT:PRINT TAB(15)"2= TO PUT ECHO IN LETTER MODE"
```

```
320 PRINT:PRINT TAB(15)"3= TO EXIT PROGRAM"
330 GOSUB 520 :REM SEND OUT A CONT E
340 PRINT
350 INPUT "WHICH MODE DO YOU WANT 1 OR 2";Z
360 IF Z=1 THEN E=87:GOSUB 610
    :REM CONT EW FOR WORD MODE
370 IF Z=2 THEN E=76:GOSUB 610
    :REM CONT EL FOR LETTER MODE
380 IF Z=3 THEN END
390 IF Z>3 THEN PRINT :PRINT "INVALID CHOICE ENTER "
    :GOTO 350
400 INPUT "A WORD";A$
410 FOR I=1 TO LEN(A$)
420 C$=MID$(A$,I,1)
430 D=ASC(C$)
440 GOSUB 490
460 NEXT I
470 GOTO 290
480 REM SUB TO OUTPUT WORD
490 LET T=INP(P5) AND 32
500 IF T=0 THEN GOTO 490
510 OUT P0,D:RETURN
520 LET T=INP(P5) AND 32
530 IF T=0 THEN GOTO 520
540 OUT P0,5
550 RETURN
600 REM SUB TO SET MODE
610 LET T=INP(P5) AND 32
620 IF T=0 THEN GOTO 610
630 OUT P0,E:RETURN
```

Professional Graphics at Practical Prices

Add our Imaginator™ graphics upgrade card to your H/Z-19 terminal or H/Z-89 computer. It's quick and easy. You gain intelligent, highly efficient graphic display capabilities proven in countless Heath/Zenith terminal and computer updates.

Check Imaginator's special features:

- ☐ High resolution 504 x 247
- ☐ Accessible through any high level language FORTRAN, Pascal, BASIC, etc.
- ☐ Onboard microcomputer eliminates processing load on host
- ☐ Source code available
- ☐ Rich graphics instruction set
- ☐ Mix text and graphics
- ☐ Tektronix® 4010-4014 compatibility option with GIN mode
- ☐ Comprehensive documentation includes numerous examples
- ☐ Fully buffered for asynchronous operation
- ☐ All original H/Z features remain intact

Now check Imaginator's low cost:

- ☐ Assembled complete $445.
- ☐ Kits from $215.

Also ask us about our Imaginator 2 upgrade for H/Z-29 terminals.

Call or write us today for additional information.

CLEVELAND CODONICS, INC.

18001 Englewood • Cleveland, OH 44130 (216) 243-1198

Your H-19 Can Do This...

IMAGINATOR

Tektronix® Registered trademark of Tektronix, Inc.
Codonics and Imaginator are trademarks of Cleveland Codonics, Inc.

A Personal Job Resume Generator

Harold C. Ogg
7043 Montana Circle
Hammond, IN 46323

All of us have, at one time or another, been in the position of looking for work. The process usually need not be mentally devastating, but, unless we are adept at careful planning and have ample time to perform a thorough campaign, the matter can be frustrating at best.

Many persons have little idea of how best to present themselves to a potential employer. They will borrow one of the many resume texts available at the local library or purchase a book at a local bookstore, which gives an outline idea of someone else's hunting process, but the parameters never seem to apply to the reader and the "suggested" search forms don't apply to the job desired. A person may have all the tools necessary to conduct a job search, but no idea how first to begin.

Most good jobsearch texts will exhibit one common denominator: in various permutations, they will all make mandatory the resume to "get one's foot in the door." This is fine, but what content is to be included? And, although most computer professionals and certainly most anyone who possesses a home computer will have at least one word processing package with which to create a neat, plausible document. It has been seen that most persons do not know the format or content of an "acceptable" resume that will get a potential employer's attention. Thus, the job seeker spends $25-$100 to have a resume service do the preparation (and sometimes mass mailing) of a task that could be accomplished in the home with an hour's time and a printer.

The following program, originally created for a public library job information center, was written in MBASIC. It is a highly user-friendly text editor which sets the proper spacings, columns, margins, offsets, and content of a job resume suitable for mailing. The format is general and is applicable to most levels of search, from vocational to professional, and includes elements common to the majority of such documents. The length is variable, since each user will have a different amount of experience, education, etc., to install. The document therefore will, when printed, have to be separated with scissors at the proper page break point and reproduced on a plain paper type photocopy machine. The "instant print" shops should be able to provide this service, and a hundred copies of mailable quality should not cost more than a few dollars.

The program itself "walks" the user through each section. After a few questions are answered, the program queries the user whether (s)he wishes to print the section or go back and start the particular section over again. At each milestone, the program gives the user an on-screen facsimile of what a given section will look like on paper. If the user doesn't like the content or has made a mistake, that section simply recycles until the user is satisfied.

There are error traps at various points, so the user does not necessarily have to be familiar with the program code itself. This might be handy in an office or school environment where a number of persons would have need of such a document generator. The DIM statements should be adequate to hold enough information for a two to three page resume; this is really the outside length most potential employers will tolerate in reading. The program was originally created for an Osborne I with CP/M; the only difference with that version and the one for Heath/Zenith is in the "clear screen" statements (Osborne I uses 'PRINT CHR$(&H1A)' for that purpose). The program could also be used with an Apple if the 80-column and CP/M cards were installed; be sure to add a statement to activate the printer (PR#6, or similar, depending on the slot used for the printer interface). The program could also be coded in CBASIC, although with its given length, there would probably be no appreciable savings of computer time.

```
1 '
2 '
3 '       ********************************************
4 '       *                                          *
5 '       *         PERSONAL RESUME GENERATOR        *
6 '       *                                          *
7 '       ********************************************
8 '
9 '
10 REM  RESUME GENERATOR 'RESUME.BAS'
15 DIM L$(15), S$(20)
20 PRINT CHR$(27);"E": REM CLEAR THE SCREEN
30 PRINT:PRINT TAB(20); "RESUME WRITER":PRINT:PRINT
40 PRINT
   " THIS PROGRAM CREATES A PERSONAL JOB RESUME FROM"
50 PRINT
   " YOUR TYPED INFORMATION.  YOU MAY MAKE COPIES OF"
60 PRINT " THE RESULT AND REPRODUCE IT AT WILL.":PRINT
70 PRINT
   " USE THE COMPUTER'S KEYBOARD AS YOU WOULD A REGULAR"
80 PRINT
   " TYPEWRITER.  YOU WILL RECEIVE A PROMPT (? OR --"
90 PRINT " OR :) WHERE INFORMATION IS REQUIRED.":PRINT
100 PRINT
    " HIT 'RETURN' WHEN A LINE OR ENTRY IS FINISHED."
110 PRINT
    " TO BACKSPACE, HOLD DOWN 'CTRL' AND PRESS 'H'."
    :PRINT
120 PRINT
    " DO NOT BE OVERLY CONCERNED ABOUT MISTAKES: THE"
130 PRINT
    " PROGRAM IS DIVIDED INTO SECTIONS SO THAT EVERY"
140 PRINT " FEW LINES YOU CAN MAKE CORRECTIONS BEFORE"
150 PRINT " PROCEEDING.":PRINT
160 LINE INPUT
    " ARE YOU READY (TYPE 'Y' AND HIT 'RETURN')? ";R$
```

About the Author:

Harold C. Ogg *is Director of Libraries for the Hammond Public Library system, Hammond, IN.*

```
170 IF LEFT$(R$,1)<>"Y" AND LEFT$(R$,1)<>"y" THEN PRINT
    " INVALID ENTRY; TRY AGAIN.":GOTO 160
180 '
190 '
200 PRINT CHR$(27);"E": REM CLEAR SCREEN
210 PRINT:PRINT
    " THIS SECTION PREPARES THE PAGE HEADINGS."
220 PRINT " (NOTE: YOU MAY ENTER DATA IN BOTH lower and"
230 PRINT
    " UPPER CASE LETTERS; USE 'shift' or 'alpha lock'."
240 PRINT:PRINT
    " ENTER YOUR NAME EXACTLY AS YOU WANT IT TO APPEAR."
250 LINE INPUT " --";PERSON$
260 PRINT " ENTER YOUR HOUSE NUMBER AND STREET ADDRESS."
270 LINE INPUT " --";ADDRESS$
280 IF LEN(ADDRESS$)>25 THEN PRINT
    " CAN YOU SHORTEN THAT TO LESS THAN 26 LETTERS?":
    GOTO 260
290 PRINT " AND YOUR CITY, STATE AND ZIP CODE."
300 LINE INPUT " --";CITYSTATE$
310 IF LEN(CITYSTATE$)>25 THEN PRINT
    " CAN YOU SHORTEN THAT TO LESS THAN 26 LETTERS?":
    GOTO 290
320 PRINT " ENTER YOUR HOME PNONE NUMBER WITH AREA CODE."
330 LINE INPUT " E.G., (219) 555-1212. --";HOMEPHONE$
340 PRINT
    " YOUR BUSINESS PHONE, IF YOU HAVE ONE, SAME WAY."
350 LINE INPUT " (HIT 'RETURN' IF YOU HAVE NONE) --";
    BUSPHONE$
360 PRINT CHR$(27);"E": REM CLEAR SCREEN
370 PRINT:PRINT
    " THIS IS THE WAY YOUR HEADER WILL APPEAR:"
380 PRINT:PRINT STRING$(80,"_")
390 FOR X=1 TO 5: PRINT: NEXT X: REM SPACE DOWN 5 LINES
400 PRINT TAB(33); "PERSONAL RESUME OF": PRINT
410 PRINT TAB(INT(42-(LEN(PERSON$)/2))); PERSON$: PRINT
420 IF LEN(BUSPHONE$)<7 GOTO 460
430 PRINT ADDRESS$; TAB(50); "Home Phone: "; HOMEPHONE$
440 PRINT CITYSTATE$; TAB(50); "Business:   "; BUSPHONE$
450 GOTO 480
460 PRINT ADDRESS$; TAB(50); "Home Telephone:"
470 PRINT CITYSTATE$; TAB(50); HOMEPHONE$
480 PRINT:PRINT:PRINT
    " DO YOU WANT TO LEAVE THE HEADERS THE"
490 PRINT " WAY THEY ARE <1) OR DO YOU WISH TO START OVER"
500 PRINT " AND MAKE CORRECTIONS <2>?"
510 LINE INPUT
    " ENTER '1' OR '2' AND 'RETURN' TO CHOOSE-- ";R$
520 IF LEFT$(R$,1)="2" GOTO 180
530 IF LEFT$(R$,1)<>"1" THEN PRINT CHR$(27);
    "E":PRINT:PRINT:PRINT " PLEASE ENTER ONLY '1' OR '2'.
    TRY AGAIN.": GOTO 480
540 '
550 '
560 LPRINT TAB(33);"PERSONAL RESUME OF":LPRINT
570 LPRINT TAB(INT(42-(LEN(PERSON$)/2))); PERSON$:LPRINT
580 IF LEN(BUSPHONE$)<7 GOTO 620
590 LPRINT ADDRESS$; TAB(50); "Home Phone: "; HOMEPHONE$
600 LPRINT CITYSTATE$; TAB(50); "Business:   "; BUSPHONE$
610 LPRINT:LPRINT:GOTO 670
620 LPRINT TAB(20); ADDRESS$; TAB(50); "Home Telephone:"
630 LPRINT TAB(20); CITYSTATE$; TAB(50); HOMEPHONE$
640 LPRINT:LPRINT
650 '
660 '
670 PRINT CHR$(27);"E": REM CLEAR SCREEN AGAIN
680 PRINT:PRINT
    " IN THIS SECTION YOU WILL TELL SOMETHING ABOUT"
690 PRINT
    " THE KIND OF WORK YOU ARE LOOKING FOR AND WHAT KINDS"
700 PRINT " YOU HAVE DONE.":PRINT
710 PRINT " IN ONE OR TWO LINES, STATE YOUR 'JOB OBJECTIVE',"
720 PRINT " SUCH AS 'Entry level sales representative for"
730 PRINT " a tool and die manufacturing firm'.  DO NOT"
740 PRINT " SPLIT WORDS AT THE END OF A LINE.  WHEN YOU"
750 PRINT " HAVE TYPED AS MUCH AS YOU WISH , TYPE 'ZZZZ'"
760 PRINT " AFTER THE '--'. NOW TYPE YOUR 'JOB OBJECTIVE'."
770 X=0
780 X=X+1
790 LINE INPUT " -- ";L$(X)
800 IF L$(1)="ZZZZ" OR L$(1)="zzzz" THEN PRINT:PRINT
    " YOU MUST ENTER AT LEAST ONE LINE. TRY AGAIN.":
    GOTO 790
810 IF LEN(L$(X))>50 THEN PRINT
    " SHORTEN THAT LINE TO 50 SPACES OR LESS AND REENTER.":
    GOTO 790
820 IF L$(X)<>"ZZZZ" AND L$(X)<>"zzzz" GOTO 780
830 PRINT CHR$(27);"E"
840 PRINT:PRINT
    " HERE IS HOW THE PREVIOUS INFORMATION WILL APPEAR:"
850 PRINT:PRINT:PRINT "   OCCUPATIONAL"; TAB(20); L$(1)
860 IF L$(2)="ZZZZ" OR L$(2)="zzzz" THEN
    PRINT "   OBJECTIVE": GOTO 920
870 PRINT "   OBJECTIVE"; TAB(20); L$(2)
880 X=2
890 X=X+1
900 IF L$(X)="ZZZZ" OR L$(X)="zzzz" GOTO 920 ELSE
    PRINT TAB(20); L$(X)
910 GOTO 890
920 PRINT: PRINT: PRINT: PRINT
930 PRINT " DO YOU WISH TO CONTINUE WITH THE PROGRAM <1>"
940 PRINT " OR DO YOU WANT TO REDO THE ABOVE <2>?"
950 LINE INPUT " ENTER '1' OR '2' AND 'RETURN'-- ";R$
960 IF R$="2" GOTO 670
970 IF R$<>"1" THEN PRINT
    " ENTER ONLY '1' OR '2'.  TRY AGAIN.": GOTO 930
980 '
990 '
1000 LPRINT "   OCCUPATIONAL"; TAB(20); L$(1)
1010 IF L$(2)="ZZZZ" OR L$(2)="zzzz" THEN LPRINT
     "   OBJECTIVE":GOTO 1050
1020 LPRINT "   OBJECTIVE"; TAB(20); L$(2)
1030 X=2
1040 X=X+1
1050 '
1060 IF L$(X)="ZZZZ" OR L$(X)="zzzz" GOTO 1080 ELSE
     LPRINT TAB(20); L$(X)
1070 GOTO 1040
1080 LPRINT:LPRINT
1090 '
1100 PRINT CHR$(27);"E": REM CLS
1110 FLAG=0
1120 PRINT:PRINT
     " YOU WILL NOW LIST YOUR WORK EXPERIENCES.":PRINT
1130 PRINT
     " IF YOU HAVE NO EXPERIENCE, TYPE '1', ELSE TYPE '2'"
1140 LINE INPUT " AND HIT 'RETURN'--";R$
1150 IF R$="1" THEN GOTO 1850
1160 PRINT:PRINT
     " WHAT IS THE NAME OF THE COMPANY OR ORGANIZATION"
1170 LINE INPUT " FOR WHICH YOU MOST RECENTLY WORKED? ";C$
1180 LINE INPUT " CITY, STATE IN WHICH LOCATED? ";L$
1190 PRINT:PRINT
     " IF YOU ARE CURRENTLY EMPLOYED, TYPE 'present' IN"
1200 PRINT " SMALL LETTERS; ELSE TYPE THE LAST YEAR WORKED"
1210 LINE INPUT " (AS '1982') AND 'RETURN'--";LY$
1220 PRINT:PRINT " FIRST YEAR YOU WORKED FOR THE ABOVE"
1230 LINE INPUT " ORGANIZATION? ";FY$
1240 PRINT:PRINT " WHAT WAS YOUR JOB TITLE (OR AS CLOSE AN)"
1250 LINE INPUT " APPROXIMATION AS YOU KNOW)? ";JT$
1260 PRINT CHR$(27);"E": REM CLS
1270 PRINT:PRINT " YOU WILL NOW TELL A LITTLE ABOUT YOUR WORK"
1280 PRINT " AND RESPONSIBILITIES.  THIS WILL TAKE UP ABOUT"
1290 PRINT " TWO TO FOUR LINES ON YOUR RESUME.  THE FIRST"
1300 PRINT " LINE WILL BE SHORT (YOU'LL SEE WHY ON THE"
1310 PRINT " PRINTOUT), SO YOU MIGHT WANT TO WRITE DOWN"
1320 PRINT " YOUR RESPONSES BEFORE TYPING INTO THE COMPUTER."
1330 PRINT " AGAIN, TYPE 'ZZZZ' WHEN YOU WISH TO STOP ENTRY."
1340 X=0
1350 PRINT:PRINT " FIRST LINE;";55-LEN(JT$);
     "LETTERS OR LESS"
1360 X=X+1
1370 LINE INPUT " --";L$(X)
1380 IF LEN(L$(X))>(55-LEN(JT$)) THEN PRINT:PRINT
     " ENTRY TOO LONG; TRY AGAIN.":GOTO 1370
1390 PRINT " NEXT LINE; 55 LETTERS OR LESS"
```

```
1400 X=X+1
1410 LINE INPUT " --";L$(X)
1420 IF LEN(L$(X))>55 THEN PRINT:PRINT
     " CAN YOU SHORTEN THAT A BIT?  TRY AGAIN."
     :GOTO 1410
1430 IF L$(X)<>"ZZZZ" AND L$(X)<>"zzzz" THEN GOTO 1390
1440 X=X-1:PRINT CHR$(27);"E": REM BACK UP 'X' AND CLS
1450 PRINT:PRINT
     " HERE IS THE 'EXPERIENCE' ENTRY YOU JUST TYPED:
     ":PRINT:PRINT:PRINT
1460 PRINT:PRINT:PRINT TAB(47-(LEN(C$)/2));C$
1470 PRINT TAB(47-(LEN(L$)/2));L$:PRINT:PRINT
1480 IF L$(1)="ZZZZ" OR L$(1)="zzzz" THEN L$(1)=" "
1490 IF L$(2)="ZZZZ" OR L$(2)="zzzz" THEN L$(2)=" "
1500 PRINT TAB(6);FY$;" to";TAB(20);JT$;".  ";L$(1)
1510 PRINT TAB(6);LY$;TAB(20);L$(2)
1520 IF X<3 THEN GOTO 1550
1530 FOR Y=3 TO X:PRINT TAB(20);L$(Y):NEXT Y
1540 PRINT:PRINT
1550 PRINT:PRINT
     " DO YOU WISH TO CONTINUE WITH THE PROGRAM <1)"
1560 PRINT " OR DO YOU WANT TO REDO THE ABOVE <2>?"
1570 LINE INPUT " ENTER '1' OR '2' AND 'RETURN'--";R$
1580 IF FLAG=0 AND R$="2" THEN PRINT CHR$(27);"E"
     :GOTO 1160
1590 IF FLAG=1 AND R$="2" THEN PRINT CHR$(27);"E"
     :GOTO 1800
1600 IF R$<>"1" THEN PRINT
     " ENTER ONLY '1' OR '2'.  TRY AGAIN.":GOTO 1550
1610 '
1620 '
1630 IF FLAG=0 THEN LPRINT "   EXPERIENCE";
     TAB(42-(LEN(C$)/2))
     ;C$ ELSE LPRINT TAB(42-(LEN(C$)/2));C$
1640 IF FLAG=0 THEN LPRINT "   RECORD"
     ;TAB(42-(LEN(L$)/2))
     ;L$:LPRINT ELSE LPRINT TAB(42-(LEN(L$)/2))
     ;L$:LPRINT
1650 LPRINT TAB(6);FY$;" to";TAB(20);JT$;".  ";L$(1)
1660 LPRINT TAB(6);LY$;TAB(20);L$(2)
1670 IF X<3 THEN LPRINT:LPRINT:GOTO 1720
1680 FOR Y=3 TO X:LPRINT TAB(20);L$(Y):NEXT Y
1690 LPRINT:LPRINT
1700 '
1710 '
1720 PRINT CHR$(27);"E": REM CLS
1730 PRINT:PRINT " YOU MAY LIST AS MANY JOBS AS YOU WISH."
1740 PRINT " IS THERE ANOTHER PRIOR TO THE ONE YOU LISTED"
1750 PRINT " ABOVE?  IF SO, ENTER 'Y' FOR 'YES', ELSE ENTER"
1760 LINE INPUT " 'N' FOR 'NO' AND HIT 'RETURN'--";R$
1770 IF LEFT$(R$,1)="N" OR LEFT$(R$,1)="n" THEN GOTO 1840
1780 IF LEFT$(R$,1)<>"Y" AND LEFT$(R$,1)<>"y" THEN PRINT
     " INVALID ENTRY; TRY AGAIN.":GOTO 1730
1790 FLAG=1:PRINT CHR$(27);"E":PRINT:PRINT:PRINT
1800 LINE INPUT
     " NAME OF THIS COMPANY FOR WHICH YOU WORKED? ";C$
1810 LINE INPUT " CITY, STATE IN WHICH LOCATED? ";L$
1820 LINE INPUT " LAST YEAR WORKED FOR THIS COMPANY? "
     ;LY$
1830 GOTO 1220
1840 '
1850 '
1860 PRINT CHR$(27);"E": REM CLS
1870 PRINT:PRINT
     " YOU WILL NOW LIST YOUR EDUCATIONAL QUALIFICA-"
1880 PRINT " TIONS.  START WITH THE MOST RECENT SCHOOL OR"
1890 PRINT " COLLEGE YOU ATTENDED AND WORK BACKWARD."
     :PRINT
1900 FLAG=0
1910 PRINT " NAME OF SCHOOL (AS 'Mytown Vocational"
1920 PRINT "   Technical School, Mytown, IN'"
1930 LINE INPUT " --";S$(1)
1940 IF LEN(S$(1))>55 THEN PRINT:PRINT
     " CAN YOU SHORTEN THAT A LITTLE? PLEASE REENTER."
     :PRINT:GOTO 1910
1950 PRINT:PRINT " GIVE THE NAME OF THE DIPLOMA OR DEGREE"
1960 PRINT " YOU EARNED WITH THE YEAR YOU EARNED IT"
1970 PRINT " (AS 'Diploma: Secretarial Studies, 1978')"
1980 PRINT " NOTE: IF YOU DID NOT GRADUATE, ENTER THE"
1990 PRINT " YEARS OF ATTENDANCE AS 'Attended 1976-78'."
2000 LINE INPUT " --";S$(2)
2010 IF FLAG=1 GOTO 2060
2020 PRINT CHR$(27);"E":PRINT:PRINT
     " HERE IS YOUR FIRST 'EDUCATIONAL EXPERIENCE:"
     :PRINT:PRINT
2030 PRINT "   EDUCATION";TAB(20);S$(1)
2040 PRINT TAB(20);S$(2):PRINT
2050 GOTO 2080
2060 PRINT CHR$(27);"E":PRINT:PRINT
     " HERE IS THE NEXT 'EDUCATION' ENTRY:":PRINT:PRINT
2070 PRINT TAB(20);S$(1):PRINT TAB(20);S$(2):PRINT
2080 PRINT " DO YOU WANT TO PRINT OUT THIS ENTRY <1>"
2090 PRINT " OR DO YOU WISH TO DO IT OVER <2>?"
2100 LINE INPUT " ENTER '1' OR '2' AND 'RETURN--";R$
2110 IF R$="2" THEN PRINT:PRINT
     " RETYPE YOUR CORRECTED ENTRY.":PRINT:GOTO 1910
2120 IF R$<>"1" THEN PRINT:PRINT
     " PLEASE ENTER ONLY '1' OR '2';TRY AGAIN.":PRINT
     :GOTO 2080
2130 IF FLAG=0 THEN LPRINT "   EDUCATION";TAB(20);S$(1)
     :LPRINT TAB(20);S$(2):LPRINT
2140 IF FLAG=1 THEN LPRINT TAB(20);S$(1):LPRINT TAB(20)
     ;S$(2):LPRINT
2150 PRINT:LINE INPUT " ANY MORE SCHOOLS <Y OR N>? ";R$
2160 IF LEFT$(R$,1)="Y" OR LEFT$(R$,1)="y" THEN
     PRINT CHR$(27);"E":FLAG=1:PRINT:PRINT:GOTO 1910
2170 IF LEFT$(R$,1)<>"N" AND LEFT$(R$,1)<>"n" THEN PRINT
     :PRINT " PLEASE ANSWER ONLY 'Y' (YES) OR 'N' (NO);
    TRY AGAIN.":PRINT:GOTO 2150
2180 '
2190 '
2200 '
2210 '
2220 PRINT CHR$(27);"E":PRINT:PRINT
2230 F$(2)="0"
2240 PRINT " FOR THE LAST ELEMENT, YOU WILL GIVE SOME"
2250 PRINT " REFERENCES, IF YOU WISH.  IF YOU PREFER"
2260 PRINT
     " NOT TO DO THIS, ENTER <1> TO PRINT 'References"
2270 LINE INPUT
     " furnished on request', ELSE ENTER <2>--";R$
2280 IF R$="1" THEN LPRINT:LPRINT "   REFERENCES"
     :TAB(20);"References furnished on request."
     :FOR X=1 TO 20:LPRINT:NEXT X:GOTO 2760
2290 IF R$<>"2" THEN PRINT:PRINT
     " PLEASE ENTER ONLY '1' OR '2';TRY AGAIN.":PRINT:
     GOTO 2240
2300 PRINT:PRINT
     " THIS PROGRAM WILL ASK FOR ONLY TWO REFERENCES."
2310 PRINT
     " IF YOU WISH TO ADD MORE, USE THE SAME FORMAT AND"
2320 PRINT " TYPE THEM ON THE BOTTOM OF THE PAGE.":PRINT
2330 LINE INPUT " NAME OF REFERENCE (PERSON)? ";F$(1)
2340 IF LEN(F$(1))>25 THEN PRINT:PRINT
     " CAN YOU SHORTEN THAT A LITTLE? TRY AGAIN.":PRINT:
     GOTO 2330
2350 PRINT:PRINT
     " THAT PERSON'S ADDRESS (GIVE City, State ONLY,"
2360 PRINT
     " IF YOU HAVE THE PHONE NUMBER, ELSE GIVE ONLY"
2370 LINE INPUT " THE NUMBER AND STREET FOR THIS LINE--"
     ;N$(1)
2380 IF LEN(N$(1))>25 THEN PRINT:PRINT
     " CAN YOU SHORTEN THAT A LITTLE? TRY AGAIN.":PRINT
     :GOTO 2350
2390 LINE INPUT
     " City, State OR TELEPHONE (AS (219) 555-1212)--"
     ;P$(1)
2400 IF LEN(P$(1))>25 THEN PRINT:PRINT
     " CAN YOU SHORTEN THAT A LITTLE? TRY AGAIN.":PRINT
     :GOTO 2390
2410 PRINT:LINE INPUT
     " DO YOU HAVE ANOTHER REFERENCE <Y> OR <N>? ";R$
2420 IF LEFT$(R$,1)="N" OR LEFT$(R$,1)="n" THEN GOTO 2510
2430 IF LEFT$(R$,1)<>"Y" AND LEFT$(R$,1)<>"y" THEN PRINT
     " PLEASE ENTER ONLY 'Y' OR 'N'; TRY AGAIN.":PRINT
```

```
    :GOTO 2410
2440 LINE INPUT " NAME OF OTHER REFERENCE? ";F$(2)
2450 IF LEN(F$(2))>30 THEN PRINT:PRINT
    " CAN YOU SHORTEN THAT A LITTLE? TRY AGAIN.":PRINT
    :GOTO 2440
2460 LINE INPUT " City, State or Street Address? ";N$(2)
2470 IF LEN(N$(2))>30 THEN PRINT:PRINT
    " CAN YOU SHORTEN THAT A LITTLE? TRY AGAIN.":PRINT
    :GOTO 2460
2480 LINE INPUT " AND City, State or TELEPHONE--";P$(2)
2490 IF LEN(P$(2))>30 THEN PRINT:PRINT
    " CAN YOU SHORTEN THAT A LITTLE? TRY AGAIN.":PRINT
    :GOTO 2480
2500 GOTO 2550
2510  PRINT CHR$(27);"E":PRINT:PRINT
    " HERE IS YOUR 'REFERENCES' SECTION:" :PRINT:PRINT
2520 PRINT "   REFERENCES";TAB(20);F$(1):PRINT TAB(20);
    N$(1):PRINT TAB(20);P$(1):GOTO 2600
2530 '
2540 '
2550 PRINT CHR$(27);"E":PRINT:PRINT
    " HERE IS YOUR 'REFERENCES' SECTION:":PRINT:PRINT
2560 PRINT "   REFERENCES";TAB(20);F$(1);TAB(50);F$(2)
2570 PRINT TAB(20);N$(1);TAB(50);N$(2)
2580 PRINT TAB(20);P$(1);TAB(50);P$(2)
2590 '
2600 PRINT:PRINT:PRINT
    " DO YOU WISH TO PRINT OUT THIS SECTION <1>"
2610 LINE INPUT " OR REDO IT FROM THE START <2>? ";R$
2620 IF R$="2" THEN GOTO 2220
2630 IF R$<>"1" THEN PRINT
    " PLEASE ENTER ONLY '1' OR '2';TRY AGAIN.":PRINT
    :GOTO 2600
2640 '
2650 '
2660 IF F$(2)<>"0" GOTO 2700
2670 LPRINT:LPRINT "   REFERENCES";TAB(20);F$(1)
2680 LPRINT TAB(20);N$(1):LPRINT TAB(20);P$(1):GOTO 2760
2690 FOR X=1 TO 15:LPRINT:NEXT X
2700 LPRINT:LPRINT "   REFERENCES";TAB(20);F$(1)
    ;TAB(50);F$(2)
2710 LPRINT TAB(20);N$(1);TAB(50);N$(2)
2720 LPRINT TAB(20);P$(1);TAB(50);P$(2)
2730 FOR X=1 TO 15:LPRINT:NEXT X
2740 '
2750 '
2760 PRINT CHR$(27);"E": REM CLEAR SCREEN
2770 PRINT:PRINT:PRINT
    " YOUR RESUME MAY BE LONGER THAN A STANDARD"
2780 PRINT
    " ELEVEN INCH PAGE.  THE SPACING IN THIS PROGRAM IS"
2790 PRINT
    " DESIGNED TO ALLOW YOU TO CUT AND TRIM THE FINISHED"
2800 PRINT " DOCUMENT BEFORE PHOTOCOPYING."
2810 PRINT:PRINT:PRINT
    " THE PROGRAM AND SERVICE CREATED HEREIN WERE"
2820 PRINT " MADE POSSIBLE THROUGH A LIBRARY SERVICES AND"
2830 PRINT
    " CONSTRUCTION ACT GRANT ADMINISTERED BY THE STATE"
2840 PRINT " LIBRARY OF INDIANA."
2850 END
```

Customizing WordStar Version 3.3 on CP/M

Charles W. Harper, Jr.
University of Oklahoma
830 Van Vleet Oval
Norman, OK 73019

In REMark issue 46 (Nov. '83), Bob Metz provides information on customizing WordStar version 3.3 running under Z-DOS. I'd like to discuss similar customizing of Heath/Zenith's version 3.3 for CP/M. I've restricted my comments to the Z-100, but most modifications suggested are also appropriate for the H-89.

When I first got CP/M WordStar version 3.3, I very much wanted to setup the function keys to emulate Software Toolwork's editor PIE (which I also use). Minor modifications to Patrick McNally's 89-STAR had done the job nicely for an earlier version. Several programs exist which will modify the codes sent by the Z-100 function keys. Yet, most such as Patrick's Z/STAR use the operating system to reassign codes to keys. In order to use such a program one must either reboot each time WordStar is invoked, or leave the reassigned codes installed while other programs are running on the Z-100. I don't find either alternative acceptable. One should be able to freely move from one program to another in an operating system, such as CP/M, without having to worry about which version of the operating system is currently in memory. Besides, rebooting takes time.

Patrick Swayne's KEYMAP (HUG 885-1230-37) is a whole lot better. The program can enable and disable new function key assignments without rebooting. I like the program a lot for setting up the keyboard for programs that I don't use every day, like say DBaseII. WordStar, however, is a different story. Using KEYMAP to set up the function keys, you must invoke programs external to WordStar in order to obtain settings used only *in* WordStar. At best this is inelegant; with many switches in to and out of WordStar (and PIE) it is a real nuisance. Many times a day I found myself in WordStar with the function keys set up for PIE and vice-versa.

Reassigning Key Codes Within WordStar

With key expansion enabled (the default setting), function keys <F0> to <F8>, <DCHR-ICHR> and <DELLINE-INSLINE> transmit the codes given on Page B-19 of the Z-100 user's manual and listed here in Table 1, column 2. Similarly <HOME>, <left-arrow>, <right-arrow>, <up> and <down> keys send the codes given at the same page and Table 1, column 2 here. For example, <F0> (unshifted) sends ESC-J and <F1> sends ESC-S. These codes are intercepted by WordStar and converted to control-codes meaningful to WordStar. The conversion is mapped in a look-up table at address locations 468C to 46A3 and is shown in Table 1 herein. Old WordStar codes and corresponding functions are given in columns 3 and 4 of the table. The new codes and functions that I use are provided in columns 5 and 6. The reader may, of course, insert his own codes here using DDT terminated with 'g0' command followed by 'Save 72 WS.COM'; several of my function key assignments follow the Software Toolworks' program, Pie.

Incidentally, I located the table by transferring CP/M WS.COM over to Z-DOS and using DEBUG to search for the sequence CNTL-O, CNTL-P, CNTL-Q, CNTL-K, CNTL-J. In the Zenith version of 3.3, function keys <F1> to <F5> are set to correspond to these WS codes. Furthermore, these function keys send ESC-S, ESC-T, ESC-U, ESC-V, ESC-W. It seemed that if a look-up table did exist, then the proper WS codes for ESC-S, -T, -U, -V, -W (in alphabetical order) would occur in sequence. I can't describe the delight when this conjecture worked! Kind of like killing the dragon for the first time in ADVENTURE. (Debug's search function is very handy. Too bad CP/M's DDT does not have one.)

The main menu Z-100 expansion table referred to by Bob Metz in his article on Z-Dos WS 3.3 is found in the CP/M version 3.3 at address locations 0489 to 0654. See Metz' article for an excellent discussion of this table. I made one change in the table: the code at 0563-0564 is changed from F568 to 3964. This has the effect of causing the tab key <CNTL-I> to respond as WordStar's word right <CNTL-F>. Pie-affecionados will empathize with the change. (Try making short moves to the right on a line with WS using the Tab key with Insert On.) Note that I set <F0> to respond as word-left <CNTL-A> in Pie-like fashion (actually Pie uses the Escape key for tab-left, but it didn't make sense to change the Escape key).

Having modified the look-up table(s), you will want to change the function key descriptions placed by WordStar on the 25th line. The text for the 25th line is located at address locations 46B4 to 4732. Personally, I just moved the "$" sign signifying the end of the text to the beginning of the table to 46B4, thus getting rid of the entire 25th line text. The text does include an ESC-t which sets the keypad to shifted mode. (In shifted mode, the keypad acts like that of an H-89). I reintroduced the Esc-t in the terminal initialization sequence TRMINI (see below).

Additional Patch Locations

Table 2 gives various WordStar 3.3 code locations, which may be patched together with their code names taken from the WordStar 2.2x manual (which was more helpful for patching). These may be accessed using the WordStar provided program WINSTALL.COM by typing "+" at the Winstall initial installation menu prompt. (Winstall only lists options 'A' to 'F' or 'X' at the prompt.) I got this useful 'gem' from the excellent book Proportional Spacing on WordStar (obtainable from Writing Consultants, 11 Creek Bend Drive, Fairport, N.Y. 14450.) The Table 2 code locations can of course be patched using DDT. each patch location consists of a leading byte containing the number of code bytes followed by the code. :IVON and :IVOFF locations contain the code for highlighting on/off. I put in '04-ESC-m-5-2' and '04-ESC-m-7-0' to set highlighting with foreground

Table 1

Address	Function Key Code	Old WordStar Code	Old WS Function	New WordStar Code	New WS Function
468C	ESC-@ <ICHR>	CNTL-R	Up screen	CNTL-V	Insert char.
468D	ESC-A<Up arrow>	CNTL-E	Cursor up	CNTL-E	same
468E	ESC-B<Down arrow>	CNTL-X	Cursor down	CNTL-X	same
468F	ESC-C<Right arrow>	CNTL-D	Cursor right	CNTL-D	same
4690	ESC-D<Left arrow>	CNTL-S	Cursor left	CNTL-S	same
4691	ESC-E<not used>	letter E	-	letter E	-
4692	ESC-F<not used>	letter F	-	letter F	-
4693	ESC-G<not used>	letter G	-	letter G	-
4694	ESC-H<Home>	CNTL-U*	Quick prefix	CNTL-U*	same
4695	ESC-I<not used>	letter I	-	letter I	-
4696	ESC-J<F0>	CNTL-G	Del char rt.	CNTL-A	Word left
4697	ESC-K<not used>	letter K	-	letter K	-
4698	ESC-L<INSLINE>	CNTL-A	Word left	CNTL-N	Insert line
4699	ESC-M<DELLINE>	CNTL-F	Word right	CNTL-Y	Delete line
469A	ESC-N<DCHR>	CNTL-C	Down screen	CNTL-G	Del char rt.
469B	ESC-O<ICHR>	CNTL-R	Up screen	CNTL-V	Insert char.
469C	ESC-P<F6>	CNTL-T	Del word rt.	CNTL-O	Onscrn prefix
469D	ESC-Q<F7>	CNTL-V	Insert char.	CNTL-K	Block prefix
469E	ESC-R<F8>	CNTL-B	Reform para.	CNTL-P	Print prefix
469F	ESC-S<F1>	CNTL-O	Onscrn prefix	CNTL-C	Down screen
46A0	ESC-T<F2>	CNTL-P	Print prefix	CNTL-Z	Up line
46A1	FSC-U<F3>	CNTL-Q	Quick prefix	CNTL-B	Reform para.
46A2	ESC-V<F4>	CNTL-K	Block prefix	CNTL-W	Down line
46A3	ESC-W<F5>	CNTL-J	Help prefix	CNTL-R	Down screen

* 'CNTL-U' not 'CNTL-Q'. I don't know why 'CNTL-U' works here but it does.

Table 2

Address	WordStar Function	WordStar Address	Function
0267 to 026D	:IVON	06CF to 06D3	:USR1
026E to 0274	:IVOFF	06D4 to 06D8	:USR2
0275 to 027D	:TRMINI	06D9 to 06DD	:USR3
027E to 0286	:TRMUNI	06DE to 06E2	:USR4
02B2	:DEL4	06E3 to 06E7	:RIBBON
06AE to 06B4	:PSHALF	06E8 to 06EC	:RIBOFF
06BB to 06BF	:PALT	06ED to 06FD	:PSINIT
06C0 to 06C4	:PSTD	06FE to 070E	:PSFINI

of 'Cyan' and background of 'Red.' I don't have a color monitor, but I do have all 3 color planes installed. :IVOFF restores the colors to 'white' on black. :TRMINI and :TRMUNI are the terminal initialization and de-initialization sequences; I send 'ESC-t' at :TRMINI so that I can nostalgically use my keypad from time to time like an H-89; my :TRMUNI is 'ESC-z-ESC-E-ESC-H-ESC-E.' :DEL4 sets the time that the WordStar sign-on message remains on the screen; I changed it from 40 to 1 (see J. Fallows, "Behavior Modification" in The Atlantic Monthly, January, 1984, Page 92 - his "Long delay"). :PSHALF is for non-daisy strings for printer carriage return and half line feed. :PALT and :PSTD are strings for 'set alternate character width' <CNTL-P-A> and 'set standard character width' <CNTL-P-N> respectively; I use these to set italics vs. standard type on my MX-100 (G. Milburn ("Interface Age," Nov. 1982, Page 126). :USR1 to :USR4 are user-defined functions for <CNTL-P-Q>, <CNTL-P-W>, <CNTL-P-E>, and <CNTL-P-R> print control characters imbedded in a WordStar file. :RIBBON and :RIBOFF change printer ribbon color (I use these for MX-100 continuous underline on/off). :PSINIT and :PSFINI initialize & restore printer settings.

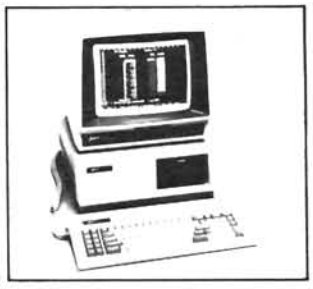
Request For Help

One detail remains. I sure would like to fix WordStar so that it will scroll up and down a full page (i.e. a full 21 lines) with CNTL-C <new F1> and CNTL-R <new F5>. Any ideas?

About the Author:

Charlie Harper *is a Professor of Geology at the University of Oklahoma. He has an article in the current issue of Computers and Geosciences entitled "A Fortran-IV Program For Comparing Ranking Algorithms in Quantitative Biostratigraphy."*

Finally, text processing that fits your computer to the letter.



Newline. Matched to the Zenith 100, 150 and the IBM-PC. If you have one of these systems, Newline has the right line of text processing software for you. Because Newline's Professional Text Processor (PTP) is matched to the keyboard and display characteristics of each of these computers.

There are no complicated key sequences to learn. And you'll be able to use labeled editing keys. Which means it's exceptionally easy to use.

New features for Newline's PTP. Even better, now PTP has more powerful text processing than ever before. There's everything from full screen text editing and on-screen bold, underline, paragraph fill and justification to cut and paste to configurable macro keys and much more. Plus, you'll be able to use our software with any printer.

Update from our old line. If you're presently using the Newline TxtPro software on your Zenith 100, now you can update to our more powerful version. It's called the PTP-100. If you currently have the ZDOS TxtPro, you can upgrade to ZDOS PTP-100. If you have the CP/M-85 TxtPro, you can upgrade to CP/M-86 PTP-100, but you'll also need to upgrade your system to CP/M-86. Just specify which one you have when you order. And if you return your old TxtPro disk to us with your order, you qualify for a special reduced price.

For Zenith 150 and IBM-PC users, there's the new PTP-PC. And it has all the same features as the PTP-100.

Software development editing. For programmers, Newline's PTP also offers auto indent and produces ASCII files for use with assemblers, interpreters and compilers. And that's not just different, it's unique.

Just give us the word. Newline's Professional Text Processor (PTP) is available right now. So place your order today. And get the text processing software that fits *your* system to the letter.

NAME _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

PTP-100 (for Z100) @ **$99. or**
PTP-100 (Z100 Upgrade) @ **$50.***

	Qty	Amount
☐ CP/M-86 (replaces CP/M-85)	____	____
☐ ZDOS	____	____

*To qualify for Z100 upgrade price of $50. you must return your TxtPro disk.

PTP-PC (for Z150 or IBM-PC) @ **Special Introductory Price of $149.**

☐ CP/M-86	____	____
☐ MS-DOS or PC-DOS	____	____
RI Residents Add 6% Sales Tax		____
Shipping ($3. per program)		____
TOTAL ENCLOSED		____

Payment in U.S. Funds Only • Allow 2 wks. for Personal Checks • CALL (401) 624-3322 FOR C.O.D. DELIVERIES

NEWLINE SOFTWARE

P.O. Box 289 • Tiverton, RI 02878 • (401) 624-3322

HUG NEW PRODUCTS

NOTE: The [-37] means the product is available in hard-sector or soft-sector. Remember, when ordering the soft-sectored format, you must include the "-37" after the part number; e.g. 885-1223-37.

P/N 885-1239-[37] CP/M
Spreadsheet Contest Disk I $20.00

Introduction: This disk contains two SuperCalc spreadsheet templates that were submitted as contest entries.

Requirements: Both programs require an H/Z-89 or H-8 with H/Z-19 terminal. WEIGHT requires a 48k system, while AMORTIZE needs a full 64k system. Both run under the CP/M operating system and SuperCalc.

The following programs and files are contained on the HUG P/N 885-1239-[37] CP/M Spreadsheet Contest Disk I:

WEIGHT	.PRN
WEIGHT	.CAL
WTABSTR	.PRN
WTQUEST	.PRN
WTAUTO	.CAL
AMORT48	.CAL
AMORTIZE	.CAL
AMORTIZE	.DOC
README	.DOC

Authors:

WEIGHT	-James H. Jones
AMORTIZE	-Clifford E. Walls

Spreadsheet Descriptions:

WEIGHT is a SuperCalc Worksheet that assists decision analysis for tangible systems comparison. It employs an analytic hierarchy technique to objectify the normally subjective process of determining the relative worth of trade-off study criteria (weighting). Useful for Systems/Engineering, Consultants or anyone interested in the use of decision support tools.

AMORTIZE is a SuperCalc template to allow amortization of any obligation through a variety of conditions of payment amounts, payment frequencies, and interest rates.

Comments: None

TABLE C Rating: (9)

P/N 885-1240-[37] CP/M
Spreadsheet Contest Disk II $20.00

Introduction: This disk contains four SuperCalc spreadsheet templates that were submitted as contest entries.

Requirements: All four templates require SuperCalc running on an H/Z-89 or H-8 with an H/Z-19 terminal and the CP/M operating system. COST requires 32k, RETIREMENT PLANNING and GRATEMP require 48k, and RETIRED requires a 64k system.

The following programs and files are contained on the HUG P/N 885-1240-[37] CP/M Spreadsheet Contest Disk II:

RETIRED	.CAL	PENSION	.DOC
RETIRED	.DOC	PENCOUP	.PRN
COST	.CAL	PENREAD	.ME
COST	.DOC	PENSING	.CAL
GRATEMP	.CAL	PENSING	.PRN
GRATEMP	.DOC	PENCOUP	.CAL
GRADEX	.CAL	README	.DOC

Authors:

RETIRED	-Henry C. Liverpool, Jr.
COST	-Jean I. Reynolds
GRATEMP	-Sanford Lamovsky
RETIREMENT PLANNING	-L. Roland Doerschug

Spreadsheet Descriptions:

RETIRED is a SuperCalc template for a military man about to retire to provide a comparison of after tax income before and after retirement. This is very useful to a retiring military man because of big differences in taxable portions of income and possible change of legal state of residence. Additionally, the military retirement survivors plan is an option used to avoid taxes.

COST is a template designed to be an efficient way to provide a business client with an accurate cost estimate or cost estimate range for a given job or task. This spreadsheet can be used as is (once the proper billing rates are inserted) for many businesses, and may be modified for conceivably any type of business.

GRATEMP is a spreadsheet template which is designed to keep track of student grades, much like a teacher's grade book. Unlike a grade book, however, it will automatically keep an updated account of the student's total points, percent correct, and calculate the average score for each set of grades.

RETIREMENT PLANNING is a template to model a retirement savings plan for Supplemental Retirement Annuities. A template for a

single savings program is included, as well as one for a husband and wife.

Comments: None

TABLE C Rating: (9)

P/N 885-1241-[37] CP/M Spreadsheet Contest

Disk III $20.00

Introduction: This disk contains two SuperCalc spreadsheet templates that were submitted as contest entries.

Requirements: Both programs require an H/Z-89 or H-8 with H/Z-19 terminal. BATSTATS requires a 48k system, while PLASTIC MONEY MANAGER needs a full 64k system. Both run under the CP/M operating system and SuperCalc.

The following programs and files are contained on the HUG P/N 885-1241-[37] CP/M Spreadsheet Contest Disk III:

PMM .CAL
PMM .DOC
BATSTATS .CAL
BATSTATS .DOC
BATSAMPL .CAL
README .DOC

Authors:

PLASTIC MONEY MANAGER -V. N. Fritz
BATSTATS -Thomas M. Goymerac

Spreadsheet Descriptions:

PLASTIC MONEY MANAGER is a SuperCalc template designed to assist in the usage of credit cards, leaving liquid assets in an interest earning position until payment is essential. Features include: 1) a running total of balances, 2) current balance of each credit card, 3) computes next statement due date from next billing date, 4) projects balance payable on next statement, and 5) computes running total of projected balances.

BATSTATS is a SuperCalc template that contains all of the necessary formats and formulas to easily maintain a softball or baseball team's vital statistics. It is designed to keep seasonal cumulative data for each player and the team as a whole, by simply entering a minimum amount of new data after each week/game. The worksheet has been setup to accommodate a maximum of 25 team members and 10 pitchers.

Comments: None

TABLE C Rating: (9)

/P/N 885-1242-[37] CP/M Spreadsheet Contest Disk IV $20.00

Introduction: These two disks contain two SuperCalc spreadsheet templates that were submitted as contest entries.

Requirements: Both programs require an H/Z-89 or H-8 with H/Z-19 terminal. ACOUSTIC requires 48k of memory, while PAYROLL needs a full 64k system. The CP/M operating system and SuperCalc are also required.

The following programs and files are contained on the HUG P/N 885-1242-[37] CP/M Spreadsheet Contest Disks IV:

Disk 1		Disk 2	
PAYROLL	.CAL	ACOUSTIC	.CAL
PAYROLL	.DOC	ACOUSTIC	.DOC
PAYROLL	.BAK	ROOM1	.CAL
PAYFORM	.CAL	ROOM2	.CAL
README	.DOC	README	.DOC

Authors:

PAYROLL -Christopher P. Pitonza
ACOUSTIC -Don Daugherty

Spreadsheet Descriptions:

PAYROLL is a SuperCalc template which allows a manager of a small department to control the most expensive portion of his business, mainly salaries of employees. It also keeps tabs on all time taken off by an employee.

ACOUSTIC is a SuperCalc template used to determine the acoustic characteristics of any room. Changes then can be made to the room for the best reproduction of sound. This program would be useful to anyone interested in the best reproduction of sound from their stereo or for a room that is used as a recording studio.

Comments: None

TABLE C Rating: (9)

P/N 885-1243-[37] CP/M Spreadsheet Contest Disk V $20.00

Introduction: These two disks contain a SuperCalc spreadsheet template that was submitted as a contest entry.

Requirements: This program requires an H/Z89 or H8 with an H/Z-19 terminal. A total of 64k of memory is needed,as well as CP/M and SuperCalc. Two disk drives are also required.

The following programs and files are contained on the HUG P/N 885-1243-[37] CP/M Spreadsheet Contest Disks V:

Disk 1		Disk 2	
ENTERLOG	.CAL	PLANES	.CAL
YEARLOG	.CAL	MTHLOG	.CAL
FLITELOG	.DOC	JULYIN	.CAL
DOYEAR	.CAL	README	.DOC
README	.DOC		

Author:

FLITELOG -Jennifer T. McGraw

Spreadsheet Description:

FLIGHT LOG is a series of overlay spreadsheets for maintaining a pilot's flight log using SuperCalc ©, Version 1.04. When all are completed, you will have a printout of daily flight legs, a summary of all planes flown, and a printout of monthly totals for the year thus far.

Comments: None

TABLE C Rating: (9)

P/N 885-1244-[37] CP/M Spreadsheet Contest Disk VI $20.00

Introduction: These two disks contain a SuperCalc spreadsheet template that was submitted as a contest entry.

Requirements: This program requires an H/Z89 or H8 with an H/Z-19 terminal. A total of 64k of memory is needed, as well as CP/M and SuperCalc. One disk drive is required, two are recommended.

The following programs and files are contained on the HUG P/N 885-1244-[37] CP/M Spreadsheet Contest Disks VI:

Disk 1

ITAXREAD	.ME
F1040	.CAL
BUSEX	.CAL
CARE	.CAL
ENERGY	.CAL
INCTAX84	.DOC
FOREIGN	.CAL
MOVEX	.CAL
README	.DOC

Disk 2

SCHA	.CAL
SCHB	.CAL
SCHC	.CAL
SCHD	.CAL
SCHE	.CAL
SCHF	.CAL
SCHG	.CAL
SCHR	.CAL
SCHW	.CAL
SCHSE	.CAL
SALE	.CAL
README	.DOC

Author:

INCTAX84 -Robert F. Hassard

Spreadsheet Description:

INCTAX84 is a set of seventeen SuperCalc templates for use next year in preparing an individual 1984 income tax return. Provision is made for the most frequently used schedules and ancillary forms.

Comments: None

TABLE C Rating: (9)

P/N 885-3017-37 ZDOS Games Contest Package (2 Disks) $25.00

Introduction: These two disks contain color graphic games that were sent in as entries to our game contest. All of these programs use the color graphic capabilities of the H/Z-100 system. Some are real time, fast action, like Worm and NYAG. Others use passive input like Bowling and Blackjack. Young and old will enjoy these disks full of fun.

Requirements: These games use the ZDOS operating system. Some of them use ZBASIC and others are compiled. The H/Z-100 with color memory and a color monitor is also needed.

The following programs are included on the HUG P/N 885-3017-37 Contest Games Disks.

Disk 1

ELEVATOR	.EXE
ELEVATOR	.BAS
ELEVATOR	.DOC
WORM	.BAS
WORM	.DOC
WORM	.EXE
WORM	.SCR
REF	.BAS
REFINST	.BAS
REF	.EXE
REFINST	.EXE
REF	.DOC
BLKJAK	.BAS
BLKJAK	.DOC
BOWLING	.BST
BOWLING	.DOC
BOWLING	.BAS
PINS	.DAT
README	.DOC

Disk 2:

DRAW-ME	.EXE
DRAW-ME	.BAS
DRAW-ME	.DOC
PLAYER1	.DGM
MAN-WB	.DGM
PLAYER2	.DGM
PLAYER4	.DGM
SIMPLE	.DGM
TITLE-PG	.DGM
MAIN-M	.DGM
PLAYER3	.DGM
THE-NW	.DGM
Z-150	.DGM
STILL	.DGM
SKIER	.DGM
ILUSADY	.DGM
MOUSCN	.DGM
NYAG	.BAS
NYAG	.DOC
DOODLES	.BAS
DOODLES1	.DOC
DOODLES2	.DOC
RDATLANT	.BST
RDATLANT	.BAS
CARS	.DAT
TRACK G	.
RDATLANT	.DOC
README	.DOC

Authors:

ELEVATOR	- Wojtek Bok
DOODLES	- Arleigh E. Luckett
BLKJAK	- Rober Sully
REF	- Michael Osborne
WORM	- Morris Proctor
NYAG	- Matt Newell
DRAW-ME	- Phil Winninghoff
BOWLING	- Joseph T. Frank
RDATLANT	- Joseph T. Frank

ELEVATOR: This is basically a two player game. The object is to move your player between different floor levels by moving according to dice rolls. Watch out however, landing on one of the many elevators can move you from the top floor to the bottom. This game can be quite frustrating.

DOODLES: This program can be used by the younger crowd to draw or "doodle" simple shapes using circles and lines. These shapes can then be painted in with different colors.

BLKJAK: This version of Blackjack uses liberal Las Vegas Strip casino rules and displays the cards using brilliant color graphics.

REF: Reflections is a logic game for one person. The object of the game is to guess the positions of the balls hidden in a square grid using a minimum number of 'probes.' This game comes with an extensive 'doc' program and playing it doesn't appear to be a 'push over.'

WORM: Actually, you start with what looks like a multi-bodied snake. The object is to eat all the apples on the screen by directing the movement of the snake with the keypad keys. The problem is, every time your snake eats an apple, it grows some new body parts making it longer. Needless to say, after a couple of screens of apples, 'snaking' your way around becomes difficult. This is an addictive fast action game.

NYAG: "Not Your Average Game" is the only way this one can be described. It appears that the object of this game is to move around a playing board and, in doing so, perform different randomly selected tasks. One of these tasks is to catch falling bricks on your head by moving your man across the video screen. As previously said, "not

your average game!"

DRAW-ME: The object of this game is to recreate a line drawing which appears on the screen. A cursor is moved, by pressing the arrow keys to points in a playing area which correspond to the end points of lines making up the displayed figure. Up to four players can participate at the same time. Scoring is kept track of by the amount of moves needed to complete the drawing. In addition, there is a full screen creator mode where individuals can create their own line drawings and save them in a disk file.

BOWLING: Watch your bowling ball travel down the alley to meet the pins in this exciting version of the ever popular sport of Bowling. Throw curves, hooks, put spin on the ball. All you'll miss is the sound of the pins crashing.

RDATLANT: This game takes advantage of the color graphics capability of the H/Z-100 computer to simulate a sports car road race at the Road Atlanta race track. You may race against a live opponent, or you may select a car for the computer to race against you. You control the speed of the car by entering acceleration values, or braking values.

Comments: None

TABLE C Rating: (0), (1), (2), (3), (7), (9)

P/N 885-3018-37 ZDOS
Spreadsheet Contest Package $25.00

Introduction: This ZDOS Spreadsheet Contest disk contains five different spreadsheets which were sent in as entries to our contest. Four of these programs are for Lotus 123 and one is for Peachtext 5000.

Requirements: Each of these spreadsheets require an H/Z-100 system with the Z-DOS operating system. The program, PERSONAL EXPENSE TRACKING requires Peachtext 5000. The other four, TAXPRO, PAYROLL, REAL ESTATE INVESTMENT, and CRYSTAL BALL need Lotus 123. All of these spreadsheets will work with a minimum of 128k of memory, except for CRYSTAL BALL, which requires a full 192k of RAM.

This package is a three disk set. Since most of the files had the same extension, they were renamed to show which files belong to which spreadsheet system. The documentation explains how to rename the files back to their original names. The following files are contained on the HUG P/N 885-3018-37 ZDOS Spreadsheet Contest disks:

Disk 1

SCHEDYJ	.WK1	LETTER	.WK3
SCHEDX	.WK1	README	.WK3
SCHEDY	.WK1	REALEST	.WK3
SCHEDYS	.WK1	INSTRUCT	.DOC
FEDTAX	.WK1	P128	.XQT
OHIOTAX	.WK1	P192	.XQT
TAXPRO	.WK1	P256	.XQT
INCOME	.WK1	FINYR	.KY1
README	.DO1	FINYR	.KY2
SUBMITTA	.DO1	FINYR	.KY3
PAYROLL	.WK2	FINYR	.DAT
README	.WK2	FINYR	.MLB
WORK	.WK2	README	.DOC

Disk 2

FIN128	.BAT	FIN128	.CAL
FIN192	.BAT	FIN192	.CAL
FIN256	.BAT	FIN256	.CAL

Disk 3

CRYSTAL	.B&W	EXPENSE5	.PIC
REPORT	.PRN	INCOME1	.PIC
CRYSTAL	.WKS	BALANCE	.PIC
EXPENSE1	.PIC	INCOME2	.PIC
CRYSTAL	.DOC	INCOME3	.PIC
EXPENSE2	.PIC	TABLE1	.PRN
EXPENSE3	.PIC	TABLE2	.PRN
EXPENSE4	.PIC	TABLE3	.PRN

Authors:

TAXPRO	- George P. Elwood
REAL ESTATE INVESTMENT	- Joel Schulman
PAYROLL	- Charles McClain
PERSONAL EXPENSE TRACKING	- Richard Donaldson
CRYSTAL BALL	- Terry Robert Groff

Program Content: The following information was taken from the authors description of each spreadsheet.

TAXPRO: This is an interactive worksheet that permits the user to input data on income, withheld taxes, interest received, dividends, donations, and interest paid to project the taxes for the current year. The program projects the Federal and Ohio State Income taxes based on the 1983 tax forms. Four of the seven modules are combined to provide the entire worksheet.

REAL ESTATE INVESTMENT: This worksheet program uses the what-if capabilities of Lotus 123 to analyze the potential for profit in a real estate investment purchase. By entering only data regarding price, proposed financing, estimated appreciation rate, and marginal tax rate, the economic viability of an investment can be determined.

PAYROLL: This template is a full service payroll module, capable of handling up to 2000 employees on a weekly payroll system under maximum RAM in Lotus 123. It is fully flexible in terms of employee information, capable of accommodating temporary or regular employees, on salary, hourly, or piecework basis. It accounts for married or single, variable dependent count, and personal deductions/advances. PAYROLL calculates FICA, SDI (State Disability Insurance), Federal and State (California) income tax for both married and single employees, working from computed gross and taxable income to yield weekly net.

PERSONAL EXPENSE TRACKING: A total of 120 classifications are provided, including two autos and a professional section. Tax deductible items are covered where they cross over into your personal life. The area for income has room for multiple incomes such as interest, husband and wife, stock, etc. Dining out is graded A thru D to study eating out habits carefully. List Manager also keeps address, telephone, and account number information for all expenses. It will also produce reports from that information for such things as applying for new credit. Tax time is made much easier with all of your expense information on the spreadsheet.

CRYSTAL BALL: This worksheet forecasts annual household financial management and cash reserve. It is entirely menu driven, and can be useful as a budgeting and financial forecasting tool.

Comments: None.

TABLE C Rating: (0), (9)

HUG Price List

Part Number	Decription of Product	Selling Price	Vol. - Issue

HDOS HARDCOPY SOFTWARE

885-1008	Volume I Documentation	9.00	
885-1013	Volume II Documentation	12.00	
885-1015	Volume III Documentation	9.00	
885-1037	Volume IV Documentation	12.00	8
885-1058	Volume V Documentation	12.00	

MISCELLANEOUS HDOS COLLECTIONS

885-1032	Disk V H8/89	18.00	8
885-1044-[37]	Disk VI H8/89	18.00	
885-1064-[37]	Disk IX H8/89 Disk	18.00	
885-1066-[37]	Disk X H8/89	18.00	10
885-1069	Disk XIII Misc H8/89	18.00	

GAMES

HDOS

885-1010	Adventure Disk H8/89	10.00	4
885-1029-[37]	Disk II Games 1 H8/89	18.00	8
885-1030-[37]	Disk III Games 2 H8/89	18.00	8
885-1031	Disk IV MUSIC H8 Only	20.00	25
885-1067-[37]	Disk XI H8/19/89 Games	18.00	12
885-1068	Disk XII MBASIC Graphic Games	18.00	10
885-1088-[37]	Disk XVII MBASIC Graphic Games	20.00	14
885-1093-[37]	D$D H8/89 Disk	20.00	16
885-1096-[37]	MBASIC Action Games H8/89	20.00	18
885-1103	Sea Battle HDOS H19/8/89	20.00	20
885-1111-[37]	HDOS MBASIC Games H8/89	20.00	23
885-1112-[37]	HDOS Graphic Games H8/89	20.00	23
885-1113-[37]	HDOS Action Games H8/89	20.00	23
885-1114	H8 Color Raiders $ Goop	20.00	23
885-1124	HUGMAN $ Movie Animation Pkg	20.00	41
885-1125	MAZEMADNESS	20.00	41
885-1130	Star Battle	20.00	47
885-8009-[37]	HDOS $ CP/M Galactic Warrior	20.00	32
885-8022	HDOS SHAPES	16.00	45
885-8026	HDOS Space Drop	16.00	49

CP/M

885-1206-[37]	CP/M Games Disk	20.00	11
885-1209-[37]	CP/M MBASIC D$D	20.00	19
885-1211-[37]	CP/M Seabattle	20.00	20
885-1220-[37]	CP/M Action Games	20.00	32
885-1222-[37]	CP/M Adventure	10.00	35
885-1227-[37]	CP/M Cassino Gamess	20.00	38
885-1228-[37]	CP/M Fast Action Games	20.00	39
885-1236-[37]	CP/M Fun Disk I	20.00	55

ZDOS

885-3004-37	ZDOS ZBASIC Graphic Games	20.00	37
885-3009-37	ZDOS ZBASIC D$D	20.00	50
885-3011-37	ZDOS ZBASIC Games Disk	20.00	52
885-3016-37	ZDOS/MSDOS Adventure	10.00	57
885-3017-37	ZDOS Contest Games Disk	25.00	58

UTILITIES

HDOS

885-1022-[37]	HUG Editor (ED) Disk H8/89	20.00	20
885-1025	Runoff Disk H8/89	35.00	
885-1060-[37]	Disk VII H8/89	18.00	
885-1061	TMI Load H8 ONLY Disk	18.00	
885-1062-[37]	Disk VIII H8/89 (2 Disks)	25.00	
885-1063	Floating Point Disk H8/89	18.00	
885-1065	Fix Point Package H8/89 Disk	18.00	10
885-1075	HDOS Support Package H8/89	60.00	
885-1077	TXTCON/BASCON H8/89	18.00	
885-1079-[37]	HDOS Page Editor	25.00	15
885-1080	EDITX H8/H19/H89 Disk	20.00	
885-1082	Programs for Printers H8/89	20.00	
885-1083-[37]	Disk XVI Misc H8/89	20.00	11
885-1089-[37]	Disk XVIII Misc H8/89	20.00	20
885-1090-[37]	Disk XIX Utilities H8/89	20.00	22
885-1092-[37]	Relocating Debug Tool H8/89	30.00	14
885-1098	H8 Color Graphics ASM	20.00	19
885-1099	H8 Color Graphics Tiny PASCAL	20.00	19
885-1105	HDOS Device Drivers H8/89	20.00	24
885-1116	HDOS Z80 Debugging Tool	20.00	27
885-1119-[37]	BHBASIC Support	20.00	29
885-1120-[37]	HDOS 'WHEW' Utilities	20.00	33
885-1121	HDOS Hard Sec Sup Pkg 2 disks	30.00	37
885-1123	XMET Robot $ Cross Assembler	20.00	40
885-1126	HDOS Utilities by PS:	20.00	42
885-1127-[37]	HDOS Soft Sector Support Pkg	30.00	45
885-1128-[37]	HDOS DISKVIEW	16.00	46
885-1129-[37]	HDOS CVT Color Video Terminal	20.00	46
885-8001	SE (Screen Editor)	25.00	28
885-8003	BHTOMB	25.00	28
885-8004	UDUMP	35.00	28
885-8006	HDOS SUBMIT	20.00	31
885-8007	EZTRANS	30.00	30
885-8015	HDOS TEXTSET Formatter	30.00	42
885-8017	HDOS Programmers Helper	16.00	42
885-8024	HDOS BHBASIC Utilities Disk	16.00	46

CP/M

885-1210-[37]	CP/M ED (same as 885-1022)	20.00	20
885-1212-[37]	CP/M Utilities H8/89	20.00	21
885-1213-[37]	CP/M Disk Utilities H8/89	20.00	22
885-1217-[37]	HUG Disk Duplication Utilities	20.00	26
885-1223-[37	]HRUN HDOS Emulator 3 disks	40.00	37
885-1225-[37]	CP/M Disk Dump $ Edit Utility	30.00	40
885-1226-[37]	CP/M Utilities by PS:	20.00	40
885-1229-[37]	XMET Robot $ Cross Assembler	20.00	40
885-1230-[37]	CP/M Function Key Mapper	20.00	42
885-1231-[37]	Cross Ref Utilities for MBASIC	20.00	43
885-1232-[37]	CP/M Color Video Terminal	20.00	46
885-1235-37	CP/M RDZDOS	20.00	54
885-1237-[37]	CP/M Utilities	20.00	55
885-5001-37	CP/M 86 KEYMAP	20.00	51
885-5002-37	CP/M 86 HUG Editor	20.00	52
885-5003-37	CP/M 86 Utilities by PS:	20.00	54
885-8018-[37]	CP/M FAST EDDY $ BIG EDDY	20.00	43
885-8019-[37]	DOCUMAT and DOCULIST	20.00	43
885-8025-37	CP/M 85/86 FAST EDDY	20.00	49

ZDOS

885-3005-37	ZDOS ETCHDUMP	20.00	39
885-3007-37	ZDOS CP/EMulator	20.00	47
885-3008-37	ZDOS Utilities	20.00	47
885-3010-37	ZDOS KEYMAP	20.00	51
885-3012-37	ZDOS HUG Editor	20.00	52
885-3014-37	ZDOS/MSDOS Utilities II	20.00	54
885-8029-37	ZDOS FAST EDDY	20.00	53

PROGRAMMING LANGUAGES

HDOS

885-1038-[37]	Wise on Disk H8/89	18.00	
885-1042-[37]	PILOT on Disk H8/89	19.00	
885-1059	FOCAL-8 H8/89 DISK	25.00	13
885-1078-[37]	HDOS Z80 Assembler	25.00	21
885-1085	PILOT Documentation	9.00	
885-1086-[37]	Tiny HDOS Pascal H8/89	20.00	13
885-1094	HDOS Fig-Forth H8/89 2 Disks	40.00	18

CPM

885-1208-[37]	CP/M Fig-Forth H8/89 2 Disks	40.00	18
885-1215-[37]	CP/M BASIC-E	20.00	26

BUSINESS, FINANCE AND EDUCATION

HDOS

885-1047	Stocks H8/89 Disk	18.00	
885-1048	Personal Account H8/89 Disk	18.00	
885-1049	Income Tax Records H8/89 Disk	18.00	
885-1055-[37]	MBASIC Inventory Disk H8/89	30.00	
885-1056	MBASIC Mail List	30.00	
885-1070	Disk XIV Home Fin H8/89	18.00	
885-1071-[37]	MBASIC SmBusPk H8/H19/H89	75.00	17
885-1091-[37]	Grade/Score Keeping H8/89	30.00	14
885-1097-[37]	MBASIC Quiz Disk H8/89	20.00	18
885-1118-[37]	MBASIC Payroll	60.00	30
885-1131-[37]	HDOS CHEAPCALC	20.00	47
885-8010	HDOS CHECKOFF	25.00	32
885-8021	HDOS Student's Statistics Pkg	20.00	44
885-8027	HDOS SCICALC	20.00	50

CP/M

885-1218-[37]	CP/M MBASIC Payroll	60.00	31
885-1233-[37]	CP/M CHEAPCALC	20.00	47
885-1239-[37]	Spread Sht. Contest Disk I	20.00	
885-1240-[37]	Spread Sht. Contest Disk II	20.00	
885-1241-[37]	Spread Sht. Contest Disk III	20.00	
885-1242-[37]	Spread Sht. Contest Disk IV	20.00	
885-1243-[37]	Spread Sht. Contest Disk V	20.00	
885-1244-[37]	Spread Sht. Contest Disk VI	20.00	
885-8011-[37]	CP/M CHECKOFF	25.00	32

ZDOS

885-3006-37	ZDOS CHEAPCALC	20.00	47
885-3013-37	ZDOS Checkbook Manager	20.00	54
885-3018-37	ZDOS Contest Spreadsheet Disk	25.00	58
885-8028-37	ZDOS SCICALC	20.00	50
885-8030-37	ZDOS MATHFLASH	20.00	55

DATA BASE MANAGEMENT SYSTEMS

HDOS

885-1107-[37]	HDOS Data Base System H8/89	30.00	23
885-1108-[37]	HDOS MBASIC Data Base Sys.	30.00	23
885-1109-[37]	HDOS Retriever ASM (3 disks)	40.00	23
885-1110	HDOS Autofile (2 disks)	30.00	23
885-1115-[37]	HDOS Navigational Program	20.00	25
885-8008	Farm Accounting System	45.00	30

CP/M

885-1219-[37]	CP/M Navigational Program	20.00	31

AMATEUR RADIO

HDOS

885-8016	Morse Code Transceiver Ver 2.0	20.00	42

CP/M

885-1214-[37]	CP/M MBASIC Log Book (64k)	30.00	23
885-1234-[37]	CP/M Ham Help	20.00	49
885-1238-[37]	CP/M ASCRITY	20.00	57
885-8020-[37]	CP/M RF Comp. Aided Design	30.00	44
885-8031-[37]	CP/M Morse Code Transceiver	20.00	57

COMMUNICATION

HDOS

885-1122-[37]	HDOS MicroNET Connection	16.00	37

CP/M

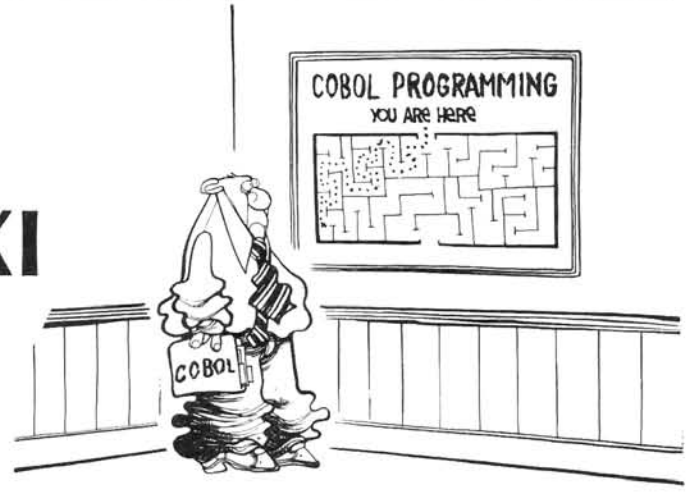
885-1207-[37]	CP/M TERM $ HTOC	20.00	26
885-1224-[37]	CP/M MicroNET Connection	16.00	37
885-3003-]37]	CP/M ZTERM (Z100 Modem Pkg)	20.00	34
885-5004-37	CPM86 TERM86 and DSKED	20.00	56
885-8005	MAPLE (Modem Appl. Effector)	35.00	29
885-8012-[37]	CP/M MAPLE (Modem Program)	35.00	34
885-8023-37	CP/M 85 MAPLE	35.00	45

MISCELLANEOUS

885-0004	HUG Binder	5.75	
885-1221-[37]	Watzman ROM Source Code/Doc	30.00	33
885-4001	REMark Volumes 1 to 13	20.00	
885-4002	REMark Volumes 14 to 23	20.00	
885-4003	REMark Volume III issues 24-35	20.00	
885-4004	REMark Vol 4 Issues 36-47	20.00	
885-4500	HUG Software Catalog	9.75	
885-4600	Watzman/HUG ROM	45.00	41
885-4700	HUG Bulletin Board Handbook	5.00	50
885-3015-37	ZDOS SKYVIEWS	20.00	55

NOTE: The [-37] means the product is available in hard sector or soft sector. Remember, when ordering the soft sectored format, you must include the "-37" after the part number; e.g. 885-1223-37.

COBOL Corner XI



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

Introduction

By now you should have Program #4 producing a readable Discount Report per the specifications. If you do not and can not work out your problems, transfer the two (2) needed files (PRGM04.COB and FILEL3.DAT) from your HUG COBOL Corner Disk-I on a NEW disk as we have described previously. Now, Compile and produce a Hard Copy Listing. Also, Link/Execute to obtain a Hard Copy Discount Report. Using these, carefully compare them with your Program #4 line-by-line. You should be able to find your ERRORS! Using your Editor, make the necessary corrections to your source code, Compile and Link/Execute. Did you get the required results? If not, review the previous COBOL Corner articles about the complete procedures, following the sequence required to RUN a COBOL program. If this does not solve your problems, send me Hard Copies of your source listing and Discount Report, along with a business size SASE. I will try to get you back on the "track"! You MUST be able to follow all the things that we have covered so far!

How do you like COBOL? The more that you learn about what COBOL can do, the better you are going to like it. We still need to add headings to our reports to make them more readable to all users of the reports. We will do that in Program #5.

Program #5 Information

PROGRAM SPECIFICATIONS

Program Name: Accts. Receiv. Report Program ID: PRGM05

Program Description:

This program will print an Accounts Receivable Report from the Customer Account Data File.

Input File:

FILEL1.DAT

Output File:

Account Receivable Report as specified below.

List of Program Operations:

A. Read each input Customer Account Record.
B. For each record, the program should do the following processing:
   1. Print an output report detail line.
   2. Accumulate the required totals.
C. Single-space each detail line.
D. Print the heading-lines on the first page, one (1) inch from the top of the page and on each additional page. Provide for 50 lines per page (not counting the total-lines).
E. Print the Accounts Receivable Page Total at the bottom of each report page and after the last report detail-line on the last page of the report. Double-space the page total down from the last detail-line on each page. Put a single asterisk after the page total.
F. After all the input Customer Account Records have been processed, print the report total-line on the report's last page. Double-space the report total-line down from the page total-line. Put two (2) asterisks after the report total.
G. COBOL will be the programming language.

Definition of Output Records in Working-Storage

In our previous programs, we have defined the Output Print Lines in the File Section. Program #5 will provide the Output Record Field Descriptions in "Working-Storage Section." (We could do the Input Records, but we will take one step at a time and do that in a later program.) We started with the File Section method because I thought that way was easier to understand. The Working-Storage method is usually the better method with the following advantages:

1. VALUE clauses (we will explain later in the article) can be used to initialize fields with data.
2. It allows the program to reference fields of a record that have been already written to output devices. "File Section" fields, within records, are not available after the WRITE statement. In other words, once a record is written, the field data can not be referenced again.
3. With the records defined in "Working-Storage", it will make the record being processed readily identifiable. This simplifies debugging, as we will see in later programs.

Defining the record-descriptions in "Working-Storage" has some disadvantages:

1. It uses more working memory space because "File Section" record-descriptions are implicitly redefined during processing.
2. It requires more data movement during processing.

The programming benefits for the "Working-Storage" definition method out-weigh the minor efficiency advantage of the FILE SECTION definition method. This will clear up as we develop new

OUTPUT REPORT LINE FORMAT

PRINT POSITIONS	FIELD NAME	COMMENTS
	DETAIL LINE	
1-5	Filler	Provide left margin
6-10	Customer Account Number	Do not zero-suppress non-significant zeros No commas or decimal point
11	Filler	
12-31	Customer Name	
32	Filler	
33-56	Customer Address	
57	Filler	
58-70	Customer City	
71	Filler	
72-73	Customer State	
74	Filler	
75-79	Customer ZIP Code	
80	Filler	
81-916	Customer Acct Balance	Zero-suppress non-significant dollar position zeros Insert commas & decimal point
92-132	Filler	
	PAGE TOTAL LINE	
1-63	Filler	
64-73		Print "PAGE TOTAL"
74-78	Filler	
79-91	Page Total	Zero-suppress non-significant dollar position zeros Insert commas & decimal point
92		Print one (1) asterisk
93-132	Filler	
	REPORT TOTAL LINE	
1-63	Filler	
64-75		Print "REPORT TOTAL"
76-77	Filler	
78-91	Report Total	Zero-suppress non-significant dollar position zeros
92-93		Insert two (2) asterisks
94-132	Filler	
	PROGRAM HEADER LINE 1	
1-5	Filler	
6-25		Print "COBOL Corner COMPANY"
26-75	Filler	
76-86		Print "REPORT DATE"
87-88	Filler	
89-90	Month	
91		Print "/"
92-93	Day	
94		Print "/"
95-96	Year	
97-132	Filler	
	PROGRAM HEADER LINE 2	
1-5	Filler	
6-25		Print "YOUR NAME, PROGRAMMER"
26-83	Filler	
84-87		Print "PAGE"
88	Filler	
89-92	Page Number	Zero-suppress non-significant zeros
93-132	Filler	
	COLUMN HEADER LINE 1	
1-6	Filler	
7-10		Print "CUST"
11-13	Filler	
14-21		Print "CUSTOMER"
22-83	Filler	
84-90		Print "ACCOUNT"
91-132	Filler	
	COLUMN HEADER LINE 2	
1-6	Filler	
7-10		Print "ACCT"
11-15	Filler	
16-19		Print "NAME"
20-35	Filler	
36-42		Print "ADDRESS"
43-59	Filler	
60-63		Print "CITY"
64-71	Filler	
72-73		Print "ST"
74-75	Filler	
76-78		Print "ZIP"
79-83	Filler	
84-90		Print "BALANCE"
91-132	Filler	

INPUT RECORD FORMAT

FIELD POSITIONS	FIELD NAME	DATA CLASS	COMMENTS
1-2	Filler		Skip Code "L1"
3-7	Account Number	Numeric	
8-27	Customer Name	Alphanumeric	
28-49	Customer Address	Alphanumeric	
50-62	Customer City	Alphanumeric	
63-64	Customer State	Alphanumeric	
65-69	Customer ZIP	Numeric	
70-71	Filler		
72-79	Customer Acct Balance	Numeric	Assumed Decimal Point between columns 77 & 78
80	Filler		

programs using this technique. Thus, we will be using the "Working-Storage" method for most of the applications.

The CODING changes for this technique follow:

```
FILE SECTION.

 FD  ACCT-REC-REPORT
     RECORD CONTAINS 132 CHARACTERS
     LABEL RECORDS ARE OMITTED.

 01  ACCT-REC-LINE.
     05  FILLER                              PIC X(132).
```

Observe that only one (1) 01-level record-description entry has been defined with no field descriptions. Only a "Filler" specification for 132 characters has been provided. Actually, ACCT-REC-LINE in Program #5 will have seven (7) different formats. They are:

1. Two (2) Column Header-Lines.
2. Two (2) Program Header-Lines.
3. One (1) Detail-Line.
4. Two (2) Total-Lines----Page and Report.

The definitions for these have been omitted from the "File Section," but they will be defined in "Working-Storage Section."

Working-Storage Section

```
WORKING-STORAGE SECTION.

 01  HEADING-LINES.
     05  H1-HEADER.
         10  FILLER          PIC X(05)  VALUE SPACES.
         10  FILLER          PIC X(20)  VALUE
                             "COBOL Corner COMPANY".
         10  FILLER          PIC X(50)  VALUE SPACES.
         10  FILLER          PIC X(11)  VALUE
                             "REPORT DATE".
         10  FILLER          PIC X(02)  VALUE SPACES.
         10  H1-MONTH        PIC Z9.
         10  FILLER          PIC X(01)  VALUE "/".
         10  H1-DAY          PIC Z9.
         10  FILLER          PIC X(01)  VALUE "/".
         10  H1-YEAR         PIC 99.
         10  FILLER          PIC X(36)  VALUE SPACES.
     05  H2-HEADER.
         10  FILLER          PIC X(05)  VALUE SPACES.
         10  FILLER          PIC X(20)  VALUE
                             "YOURNAME, PROGRAMMER".
         10  FILLER          PIC X(58)  VALUE SPACES.
         10  FILLER          PIC X(05)  VALUE "PAGE ".
         10  H2-PAGE-NBR     PIC ZZZ9.
         10  FILLER          PIC X(40)  VALUE SPACES.
 01  COLUMN-HEADERS.
     05  C1-HEADER.
         10  FILLER          PIC X(06)  VALUE SPACES.
         10  FILLER          PIC X(04)  VALUE "CUST".
         10  FILLER          PIC X(03)  VALUE SPACES.
         10  FILLER          PIC X(08)  VALUE
                             "CUSTOMER".
         10  FILLER          PIC X(62)  VALUE SPACES.
         10  FILLER          PIC X(07)  VALUE
                             "ACCOUNT".
         10  FILLER          PIC X(42)  VALUE SPACES.
     05  C2-HEADER.
         10  FILLER          PIC X(06)  VALUE SPACES.
         10  FILLER          PIC X(04)  VALUE "ACCT".
         10  FILLER          PIC X(05)  VALUE SPACES.
         10  FILLER          PIC X(04)  VALUE "NAME".
         10  FILLER          PIC X(16)  VALUE SPACES.
         10  FILLER          PIC X(07)  VALUE
                             "ADDRESS".
         10  FILLER          PIC X(17)  VALUE SPACES.
         10  FILLER          PIC X(04)  VALUE "CITY".
         10  FILLER          PIC X(08)  VALUE SPACES.
         10  FILLER          PIC X(02)  VALUE "ST".
         10  FILLER          PIC X(02)  VALUE SPACES.
         10  FILLER          PIC X(03)  VALUE "ZIP".
         10  FILLER          PIC X(05)  VALUE SPACES.
         10  FILLER          PIC X(07)  VALUE
                             "BALANCE".
         10  FILLER          PIC X(42)  VALUE SPACES.
 01  DL-DETAIL-LINE.
     05  FILLER              PEC X(05)  VALUE SPACES.
     05  DL-CUST-NBR         PIC 9(05).
     05  FILLER              PIC X(01)  VALUE SPACES.
     05  DL-CUST-NAME        PIC X(20).
     05  FILLER              PIC X(01)  VALUE SPACES.
     05  DL-CUST-ADDRESS     PIC X(24).
     05  FILLER              PIC X(01)  VALUE SPACES.
     05  DL-CUST-CITY        PIC X(13).
     05  FILLER              PIC X(01)  VALUE SPACES.
     05  DL-CUST-STATE       PIC X(02).
     05  FILLER              PIC X(01)  VALUE SPACES.
     05  DL-CUST-ZIP         PIC 9(05).
     05  FILLER              PIC X(01)  VALUE SPACES.
     05  DL-CUST-ACCT-BAL    PIC ZZZ,ZZ9.99-.
     05  FILLER              PIC X(41)  VALUE SPACES.
 01  TOTAL-LINES.
     05  PL-TOTAL-LINE.
         10  FILLER          PIC X(63)  VALUE SPACES.
         10  FILLER          PIC X(10)  VALUE
                             "PAGE TOTAL".
         10  FILLER          PIC X(05)  VALUE SPACES.
         10  PL-PAGE-TOT     PIC Z,ZZZ,ZZ9.99-.
         10  FILLER          PIC X(01)  VALUE "*".
         10  FILLER          PIC X(40)  VALUE SPACES.
     05  RT-TOTAL-LINE.
         10  FILLER          PIC X(63)  VALUE SPACES.
         10  FILLER          PIC X(12)  VALUE
                             "REPORT TOTAL".
         10  FILLER          PIC X(02)  VALUE SPACES.
         10  RT-REPORT-TOT   PIC ZZ,ZZZ,ZZ9.99-.
         10  FILLER          PIC X(02)  VALUE "**".
         10  FILLER          PIC X(39)  VALUE SPACES.
```

NOTICE the following items:

1. First heading line contains what is termed 'constant' data. 'Constant' data does not change during the processing of the program. The words COBOL Corner COMPANY, REPORT DATE and DATE VALUES are examples. They are printed on the first heading line every time the heading is printed in the report with the same data. The blanks (spaces) between the words and through to the end of the line (132 characters total) can also be called constant data. The VALUE clause is used to insert the constant data (can be used only in 'Working-Storage'). With constant data, the individual fields are not referenced, thus the fields do not require data-names but can be specified by 'Filler' entries.

2. Second heading line contains both constant and variable data. The constants are YOURNAME, PROGRAMMER, PAGE and BLANK spaces. The variable data is the page number. The variable field must have a data-name, H2-PAGE-NBR (Note the prefix H2, for second header). The page number will change during the program processing.

3. The two (2) column headers have only constant data.

4. The detail-line has constant data (BLANK spaces) and variable data. The fields (variables) that will have data moved into them have the prefix, "DL". They do not have VALUE clauses because they receive their values from the input records and Working-Storage areas. The 'filler' items represent blank spaces between the columns and they have been assigned a value by the figurative constant, SPACES.

5. The two (2) total-lines have both constant and variable data. Note how easy it is to insert the asterisks!

Cobol Value Clause

The VALUE clause format follows:

VALUE IS literal (IS optional)

The VALUE clause is used in the data-item description entry of the appropriate items in Working-Storage, where we wish to establish the initial contents of a field. When the initial value for a field is established, the programmer must make the literal specified in the VALUE clause consistent with the data-class defined by the PICTURE clause. In other words, if PICTURE clause is numeric, the literal must be numeric. Similarly, if PICTURE clause is alphanumeric, the literal must be alphanumeric. The length (size) of the literal should be consistent with the specified length of the field. The literal can not be longer than the field length as defined by the PICTURE clause. Also, when a numeric literal with a sign is used (+52 or -61), the numeric 'Picture' must contain the symbol, "S". Remember, we also have a maximum numeric literal length of 18 digits and a maximum non-numeric literal length of 120 characters. We must remember that the figurative constant, "SPACES", is a non-numeric and can not be used with a numeric PICTURE clause. However, the figurative constant, "ZEROS" may be considered either numeric or alphanumeric.

The VALUE clause can not be used in the following places:

1. In the 'File Section.'
2. With a group field data-item description entry.
3. With a data-item description entry that contains a REDEFINES clause (We will explain REDEFINES in a later article.).
4. With a data-item description entry that contains an EDITING PICTURE clause.
5. With a data-item description entry that contains an OCCURS clause (We will explain OCCURS clause when we have the occasion to use it in later programs.).

Other New Working-Storage Areas

The REPORT-CONTROL-AREA in 'Working-Storage' is used to control the following functions:

1. Sequential page numbering.
2. Page-skipping over the continuous form perforations to the top of the next report page.
3. Variable line spacing with the identifier option of the WRITE statement.

Here is some of the Report Control coding:

```
01  WS-REPORT-CONTROLS.
    05  WS-PAGE-COUNT        PIC S9(03).
    05  WS-LINES-PER-PAGE    PIC S9(02)  VALUE +50.
    05  WS-LINES-USED        PIC S9(02).
    05  WS-LINE-SPACING      PIC S9(02).
```

We will explain this when we discuss the Procedure Division coding. We will vary this area in future programs with other ideas!

Here is one (1) way, (we will use others) to CODE the DATE-AREA:

```
01  WS-DATE-AREA.
    05  WS-MONTH             PIC 9(02)  VALUE 09.
    05  WS-DAY               PIC 9(02)  VALUE 12.
    05  WS-YEAR              PIC 9(02)  VALUE 83.
```

You would, of course, pick your report date by varying the value of the numeric literals.

Closing

For your "homework" please do the following for Program #5:

1. Prepare the Print Chart.
2. Prepare the Record Chart.
3. Prepare the Structure Chart.
4. Prepare the Flowcharts.
5. Code the program through the 'Working-Storage Section' on the Coding Sheets.
6. Key-in the above code.

I will help you code the Procedure Division for Program #5 into the numbered modules in the next COBOL Corner article. If you have the above "homework" completed, you will find it much easier to follow! If you feel "daring" you might try writing the Procedure Division code on your Coding Sheets. Then, next month you will be able to compare your ideas and methods with mine. Remember, that in COBOL that there will nearly always be more than one way to write the program. We are going to use many ways in future articles. See you next month, happy COBOLing!

Update . . .

Available for 885-3010-37: ZDOS Keymap has been updated and will now run under ZDOS 1.X or MSDOS 2.X on the Z100 series computers only. Anyone wishing to obtain this update may send their original 885-3010-37 and $5.00 to Nancy Strunk, Heath Users' Group, Hilltop Road, St. Joseph, MI 49085. The label on the disk will determine if you have the updated version or not.

Two New HDOS File Management Utilities

Jack McKay
3200 19th St. NW
Washington, DC 20010

Introduction

At first inspection, I thought two new HDOS file management utilities from T & E Associates might prove useful on occasion. After two months of use, I've changed my mind, they're essential programs for the serious HDOS user.

Functions

FM and DM facilitate file management, especially copying files from disk to disk (multiple-drive systems only). There are two compelling advantages to the use of these utilities in place of the standard HDOS copy routine.

Copy Verification. FM and DM execute a check to be certain that the destination file is a good copy. FM rereads both files and does a complete byte-by-byte compare; DM runs a CRC check on the destination. HDOS, as you know, does no checking; if the copy is bad, due to a disk problem or power glitch or whatever, you discover it only when you come back later and attempt to read the copy. If, in the meantime, you've changed or deleted the original -- too bad. For the serious user who cannot afford file losses, the file-verification function is essential. It can, of course, be simulated with a compare utility, such as those marketed by Software Toolworks and Software Wizardry, but the single-step operation is quick and automatic.

Date Preservation. This may not, at first blush, seem so important, but in fact turns out to be invaluable: FM and DM preserve the file alteration date. HDOS substitutes the system date, which can create confusion; if, weeks or months later, you discover two files with the same name but different dates, you can't be sure that the "more recent" file isn't really an earlier version merely subjected to a later disk transfer operation. With the T & E utilities, the file date becomes a fixed, reliable version date; a flag indicating when the file was really last edited, not the date of the last time it was copied. I have, for example, three different versions of FM.ABS around (due to patches, as Tom Cauthen responded to my criticisms of the original); there is never any question of which is which, since the three have different dates attached, dates which will remain fixed for each version, despite any later disk-copy operations.

FM also uses the true "alteration date," distinguishing properly between the two dates, which HDOS attaches in the directory to each file but neglects to employ. Change a driver setting with SET, and the HDOS-indicated date remains the same; but copy that file with FM and the date on which the file was changed with SET will appear. Even patches applied with the Software Toolworks' Superzap will result in a new file-alteration date, exposed after an FM copy operation. This greatly helps with keeping track of which version of a program or a driver is really the most recently modified.

Operation: FM.ABS

While similar in general function, FM and DM are quite different in operation. Heavy-duty will probably prefer FM, while unsophisticated or non-typing users may favor DM.

FM begins with a request for specifications of source and destination disks, then displays the directory of the source disk. A single file, or a subset of files, is selected by typing in the file name, with the usual HDOS * and ? wildcards. Operation selection switches can also be specified:

/U = Copy & Verify	/R = Reset specified drive
/T = Copy, Verify & Delete	/F = Toggle S-flag
/V = Verify Only	/D = Toggle date
/W = Verify & Delete	/E = Exit FM
/X = Delete	

/U is the natural default value. /R permits switching source or destination disks without leaving FM. /F adds the S-flagged files to the directory listing. /E gets you out of FM.

A particularly useful switch is /D, the date switch. Its invocation selects only those files with an alteration date identical to the present system date, meaning that they have been edited or otherwise changed this day. This is for automating a daily back-up procedure, using the /D switch to copy all files edited during the day's work, while ignoring those unchanged. This is one reason for preferring FM to DM, which lacks the date-selection function; I know of no other file management program with such a date selection capability.

After each change in the specification of files, either by the /D or /S switches or by filename specification with wildcards, the displayed directory is changed to show only the files corresponding to the specifications. When this list is agreeable to the operator, typing P causes FM to execute the operations specified by the switches.

While this menu mode is useful when first learning to use FM, familiarity with the switches and the operation of FM rapidly permits one to use the direct-execution mode. For example, typing

```
>FM SY1:=SY0: /T /D *.BAS
```

will cause FM immediately to copy all files with the .BAS extension, revised today, from SY1: to SY0:, deleting the files on SY0: if the verification proves that the copy is good. Since /U (copy and verify) is the default, the usual FM call will be simply

```
>FM SY1:=SY0: FILENAME.EXT
```

Executing FM is naturally slower than the simple HDOS copy, even with verification suppressed with the /S switch; nevertheless, I now use FM exclusively in order to preserve the file alteration dates.

The byte-by-byte verification is astonishingly fast. This was tested by copying large files, then using DUMP to change single bytes and running FM with the /V (verify only) option. The artificial errors were found virtually as soon as the disk read had reached the point of the changed byte.

Operation: DM.ABS

DM is similar in the end result -- copy checking and date preservation -- but is aimed at the less sophisticated (or bad typing) user. Most operations are reduced to single keystrokes. File selection is by moving the cursor, with the arrow keys, to the file name on an alpha-sorted directory list. The operation is then selected with the function keys as follows:

Key	Label	Function
f1	VIEW	Scroll the (text) file
f2	COPY	Disk-to-disk copy
f3	RENAME	File rename
f4	LIST	Send the (text) file to LP:
f5	FLAG	Remove the write-protect flag
ERASE	DELETE	Delete the file
BLUE	DISK	Reselect source disk, or reset drive
RED	EXIT	Leave DM

Plainly DM will do some things that FM will not, like permitting examination, printing, or renaming of a file. Most of these functions simply duplicate HDOS functions, e.g. TYPE and PRINT, permitting their execution with single keystrokes. If you're a slow typist and prefer menu-selection to keyboard input, you'll like DM. But, if you're quick with your fingers and have numerous files to manipulate, you'll prefer FM, as I do. One quickly tires of having to specify the destination disk each and every time "COPY" is selected.

Problems

Use of DM revealed one limitation: It won't tolerate a write-protected source disk, even though you may intend no write operations to that disk. This was a fairly substantial problem for my system, since my 80-track outboard drives will read 40-track disks but declare them write-protected.

Testing of FM revealed a few bugs in the original version. For example, it initially didn't deal properly with STAND-ALONE operation. This and all the other problems I could find have been eliminated, as my criticisms sent Tom Cauthen back to the rewrite table.

Conclusion

FM and DM take 17 sectors each on a single-density disk. I now keep a copy of FM on every running disk I have, since I consider the date-preservation and copy-verification functions indispensable. Every serious HDOS user will, I believe, find one or the other of these programs essential. The two come as a single package from T & E Associates, Box 362, Millersville, MD 21108, for $30; you can try both and take your pick.

ZERA - An Erase Utility With A Catch For The H/Z-100

Jeff Kalis
1920 Sylvan S.E.
Grand Rapids, MI 49506

Every now and then while deep into one of those midnight programming sessions, one tends to forget which disk is in which drive. And what files are on which disk. You decide that you no longer need the ASM files you've been playing around with so you enter the command 'DEL *.ASM' only to realize (too late, of course) that your on the wrong disk and you just deleted some of your good ASM files on your master program disk. Well that has happened to me more times than I care to recall. I know, I know, where is my backup? Where's yours? We all have backups don't we?

ZERA won't allow this to happen, unless you really want it to happen. Each filename that matches the wildcards '*' or '?' is printed on the screen and you are asked if this file should be deleted. When all matching filenames have been printed and you have answered yes or no to each of them, a final query is made making sure this is what you really want. Answering yes to the filenames will cause them to be deleted while answering no will leave those files untouched.

The last utility I wrote, ZD - A sorted directory utility, seemed to generate a lot of interest in writing machine language programs under ZDOS/MS-DOS. At least that is what I gathered from the people who contacted me after the article. This program is a little shorter and a little easier to follow than ZD, so I'll try to give you a better understanding of what is going on.

First, I would suggest careful study of Appendix I and P in the ZDOS manuals. Here is where you can find most of the stuff you'll need to know. If you are unfamiliar with assemblers, you might also check out the MACRO-86 chapter in the ZDOS manual, chapter 10. It's a big chapter.

ZDOS, PC-DOS, or any MS-DOS type operating system works the same way. There are function calls a programmer can use when writing his or her own programs that will perform certain operations. For instance, sending a character to the screen. This is function call 2 (see page I.6 in the ZDOS manual). Load the DL register with the character you want output to the screen, then load the function call in the AH register and finally execute an interrupt 21 hex. If the programmer sticks to the function calls defined in the MS-DOS system, the resultant program should run on any MS-DOS machine. As a matter of fact, this program was run on an IBM PC without any modification. ZD will also run on a PC but the screen clear function does not work because the PC uses a different escape sequence than the Z-100 does. By the way, the sequence for the PC is ESC '[2J'. Put this in the ZD program and ZD will function perfectly on an IBM PC.

Looking at the source listing for ZERA, notice an area where a bunch of symbols all have the word EQU after them. What we're doing here is assigning a symbol name the value of the EQU directive. The first one, CONIN EQU 8, in effect says, whenever the symbol CONIN occurs in the source code listing, stick an 8 in there. So, the instruction MOV AL,CONIN is the same as MOV AL,8. Why use the symbol when the other way looks shorter and easier? Two reasons come to mind. First, the source code is a little more readable. Seeing CONIN rather than an 8 is more apt to remind me that I am using the console input function call. Second, what happens if you decide to change the value of a symbol? That's simple, just change the EQU directive to what ever you want the new symbol to be. The other way would result in going through the program and changing every instruction to reflect the new value. The first way, the assembler does the work and the other way, you do the work. Every function call used in ZERA is listed in the EQU statements so you can look each of them up and check out their operation. Many assembly language programs will include predefined files that contain all the system calls, interrupt vectors and such. I find it easier to just define the ones I plan to use in the program. Good ol' CP/Mers will probably agree.

The main jist of ZERA is this. The filename specified when the program was invoked is moved to a file control block (FCB). The directory is searched for a matching filename. If one is not found, an appropriate message is printed and control returns to DOS. When a match is found, the 12 character file name is moved into a storage area and a filename counter gets 1 added to it. The directory is searched for anymore matches. When they are found, they too are moved to the storage area. When no more filenames match, each name in the storage area is printed on the screen with a question asking if this file is be to deleted. If the answer is yes, then the filename is left unchanged. If the answer is no, then the first character of the 12 character filename is changed to an 'FF' hex. This is done to mark the files that are not to be deleted. After answering yes or no to each of the filenames, you are asked to answer yes or no to the double check question. A no will cause a return to DOS. A yes will cause the program to go back through the list of filenames and delete any file that does not have an 'FF' hex as the first character.

The first function call used in ZERA is PSTRNG. It is EQUated to the number 9 which is the print string function. The DX register is loaded with the starting location of a text string or escape sequence. In this case, DX is loaded with the location where the signon message starts. Notice the word OFFSET in the MOV instruction. This tells the assembler to load the indicated register with the offset value of the variable in the segment the variable is in. In effect it loads the location of the variable, in this case, the beginning of the string message. The string is terminated with a '$' character. When interrupt 21H is performed (21H is EQUated to SYS CALL), the message will be printed on the screen. Each of these function calls is discussed

in the ZDOS manual in appendix I, so I won't explain each of them in to much detail.

The next thing to do is see if a filespec was included in the invocation of ZERA. When a program is executed from ZDOS, the remainder of the command line is stored at location 80H. We'll point the SI register at location 80H for the start of the command line. Location 80H contains the length of the command. If the length is 0 then there was no filespec provided. ZERA points to a little help message, and prints it on the screen using the PSTRNG funtion. We then use interrupt 20H to return to ZDOS.

If the length of the input buffer was not 0, there was something else typed in when ZERA was started. Is should be a filespec so we will use the parse function call to move this filespec to a FCB that was set up in the storage area. Add 1 to the SI register so it's pointing to the beginning of the command line. Next, point the DI register to the dummy FCB. Those familiar with CP/M can appreciate the parse function call. It searches the buffer pointed to by the SI register and creates an unopened FCB where the DI register is pointed.

With the dummy FCB set up, we can use the search for first function call. This call will take the FCB that was set up by the previous call and check to see if there are any filenames on the disk that match it. Obviously, wild card specifiers are supported by the parse and search for first function calls. The AL register will return a 'FF' if no names on the disk match the FCB. If this is the case, ZERA will print a message stating this fact and then return to ZDOS. Otherwise, the input buffer at 80H will hold a good file name that matched the FCB. We're now down to the FOUND1 label in the program. As you can see, the function calls do a lot of work in just a few lines of code. This shows you the power found in the MS-DOS/ZDOS system, and how easily it can be put to use by anyone trying to write a program.

The loop at FOUND1 moves the 12 byte filename from the input buffer at 80H to the filename storage provided at the end of the program code. This storage will overlay (write on top of) the help message. The help message will not be needed when a legal filespec is used so we'll just reuse the space for our filename buffer. This method of reclaiming memory space is very usefull where memory is scarce. In this case, memory is not scarce but it's good efficient programming. The loop will also use the search for next call and look for any more matching file names. If they are found, they too will be moved into the storage area.

When all filenames that match have been found, control drops through to the CHKLP area. Here, each name that was found and matched is moved into a message buffer one at a time. The message buffer asks whether the file should be deleted or not. The print string function call is used to print the message and the console input function call is used to get the answer. Any input other than a 'Y', 'N' or CTRL C, will be ignored. A CTRL C will cause the program to be aborted. A 'Y' input will cause the 'Yes' message to be printed and a 'N' input will cause ZERA to print the 'No' message. When 'N' is entered, ZERA also will change the first character of the filename in storage to an 'FF'. This 'FF' will let us know that this file is not to be deleted later on when we actually delete the files.

After all the filenames have been run through the message buffer, and all the names have been marked for saving, control drops through to the DBLCHK area. Here, a confirmation message is displayed and the final O.K. is recieved. Again, CONIN and PSTRNG are used with anything other than a 'N', 'Y' or CTRL C being ignored. When that last 'Y' is typed, each filename is moved into the dummy FCB. The first byte is checked for the save flag 'FF'. If it is there, we leave that file alone and go on to check the next one. When the first byte is not an 'FF', the delete a file function call is used. The file whose name is in the FCB will be deleted. After all the filenames in storage have been taken care of, the program terminates with a function complete message, and an interrupt 20H returns control back to ZDOS.

There are many different ways to accomplish most any task. The same goes for computer programming. Any two people given a program to write with the same language and the same end result will no doubt arrive there by different means. ZERA could have been written a number of different ways and the more I look at it, I see things that I did not see earlier. The whole point is there is no right or wrong way to write programs. There are ways which are faster, more efficient or easier to follow but the end result is usually what counts. The best way to learn is just to do. The worst thing that can happen is to wipe out a disk, but what are backups for?

Anyone wishing to avoid typing the source for ZERA is welcome to send me a blank 5-1/4" disk and a dollar and I'll put the program on the disk for you. You can also just send me six dollars and I'll send you the program on a brand new disk. Feel free to contact me with any problems or questions you may have about ZERA.

```
        PAGE    60,132

        TITLE   ZERA - Z100 Erase Utility  Ver. 1.1
        SUBTTL          By Jeff Kalis   Aug. '84

;
;       ZERA is used the same way as the ZDOS
;       DEL command is used.  The differance
;       is ZERA will display the filename and
;       ask if it is to be deleted before it
;       it does so.  Typing a 'Y' or 'y' will
;       cause the file to be erased.  Typing
;       'N' or 'n' will not erase the file.
;
;       Hidden files are not deleted by ZERA.
;       The command may include '?' and '*'
;       as wildcards in the filename and ZERA
;       will ask if each matching file is to
;       be deleted.  Erasing of files is done
;       at the end of the program, after one
;       last query is made double checking
;       the files marked for deletion.
;
;   Example:  A:ZERA TEST?.*
;
;             A:TEST1    .ASM  Erase [Y or N]? No
;             A:TEST1    .OBJ  Erase [Y or N]? Yes
;             A:TEST2    .HEX  Erase [Y or N]? No
;
;               Delete as indicated [Y or N]? Yes
;
;                  Function complete
;              A:
;
;       Enter 'ZERA' without a filespec, and
;       a short help message will be displayed.
;

CODE    SEGMENT
        ASSUME  CS:CODE, DS:CODE, ES:CODE

        ORG     0100H   ;Start the program at 100H
```

```
BEGIN:  JMP     START   ;Jump over our data to the program code


;               ----- Define the Equates -----

CONIN   EQU     8       ;System call for console input, no echo
PSTRNG  EQU     9       ;System call for print string to console
LF      EQU     10      ;ASCII line feed
CR      EQU     13      ;ASCII carriage return
SRCH4F  EQU     17      ;System call for search directory for first
SRCH4N  EQU     18      ;System call for search directory for next
DFILE   EQU     19      ;System call for delete a file
RESET   EQU     20H     ;System interrupt for returning to DOS
SYS_CALL EQU    21H     ;System interrupt for initiating system calls
PARSE   EQU     41      ;System call for parsing an input for a filename
BUFFER  EQU     0080H   ;Location of input buffer

;               ----- Set up storage area -----

DFCB    DB      0, 11 DUP (' ')         ;Dummy FCB
        DB      21 DUP (0)

;               ----- Message storage -----

SIGNON  DB      CR,LF,'ZERA - Erase Utility     '
        DB      '      Version 1.1',CR,LF,LF,'$'
MBUFF   DB      '?:????????.???  Delete  [Y or N]? $'
YES     DB      'Yes',CR,LF,'$'
NO      DB      'No'
CRLF    DB      CR,LF,'$'
NOFLMS  DB      'No matching files found',CR,LF,'$'
CHKMSG  DB      CR,LF,'   Delete as indicated  [Y or N]? $'
ABORT   DB      'No',CR,LF,LF,'    Function ABORTED',CR,LF,'$'
COMPLT  DB      LF,'    Function complete',CR,LF,'$'

;               ----- Actual start of program code -----

START:

        MOV     DX,OFFSET SIGNON    ; Point to sign on message
        MOV     AH,PSTRNG           ;   then print it
        INT     SYS_CALL
        MOX     BX,BUFFER           ;
        MOV     AL,[BX]             ; If the input buffer is
        CMP     AL,0                ;   empty, print a help
        JNZ     CONTINU             ;   message then return
        MOV     DX,OFFSET HELPMSG   ;   to ZDOS
        JMP     ERLYOUT             ;

CONTINU:XOR     BX,BX               ; Clear out the file counter
        MOV     SI,OFFSET BUFFER+1  ;Point SI to input buffer
        MOV     DI,OFFSET DFCB      ;Point DI to the dummy FCB
        MOV     AH,PARSE
        XOR     AL,AL           ;If there is a filename in the
        INT     SYS_CALL        ; buffer, move it to the FCB
        MOV     DI,OFFSET FNPTR ;DI points to base of file storage
        MOV     DX,OFFSET DFCB
        MOV     AH,SRCH4F       ;Search the directory for
        INT     SYS_CALL        ; matching filename
        INC     AL
        JNZ     FOUND1          ;If a file was found, continue
        MOV     DX,OFFSET NOFLMS ;  else print no files and
ERLYOUT:MOV     AH,PSTRNG       ;  return to DOS
        INT     SYS_CALL
        INT     RESET

FOUND1: MOV     SI,OFFSET BUFFER ;Move 12 bytes of filename
        MOV     CL,12           ;  into storage
        XOR     CH,CH
        REP     MOVSB
        INC     BX              ;Add 1 to the file counter
        MOV     DX,OFFSET DFCB
        MOV     AH,SRCH4N       ;Search the directory for
        INT     SYS_CALL        ;  another matching filename
        INC     AL              ;If we found another, store it
        JNZ     FOUND1          ;  else drop through
        PUSH    BX              ;Save the file counter on the stack

        MOV     SI,OFFSET FNPTR ;Point SI to first filename in storage
CHKLP:  MOV     DI,OFFSET MBUFF ;Point DI to the message buffer
        MOV     AL,[SI]
        ADD     AL,'A'-1        ;Get the drive, make it ASCII
        MOV     [DI],AL         ;  then put it in the message buffer
        INC     SI
        INC     DI              ;Bump the buffer pointers
        INC     DI
        MOV     CX,8            ;Move the 8 byte filename from storage
        REP     MOVSB           ; to the message buffer
        INC     DI              ;Bump the pointer over the '.'
        MOV     CX,3            ;Move the 3 byte file extention
        REP     MOVSB           ; from storage to the buffer
        MOV     DX,OFFSET MBUFF
        MOV     AH,PSTRNG       ;Print the message asking if this
        INT     SYS_CALL        ;  filename is to be deleted
GTANSR: MOV     AH,CONIN
        INT     SYS_CALL        ;Get a key from the keyboard
        CMP     AL,03           ; If the key is ↑C then
        JZ      BK2DOS          ;   return to ZDOS
        OR      AL,' '          ;Force lower case letter
        CMP     AL,'n'          ;The key must be a y or an n
        JZ      MARKIT          ;  else ignore it and wait for another
        CMP     AL,'y'
        JNZ     GTANSR

        MOV     DX,OFFSET YES   ;Arive here only if input was a 'y'
        MOV     AH,PSTRNG       ; print 'Yes'
        INT     SYS_CALL
        JMP     SEIFDN          ;Check for more filenames

MARKIT: MOV     DX,OFFSET NO    ;Arive here only if input was an 'n'
        MOV     AH,PSTRNG       ; print 'No'
        INT     SYS_CALL
        SUB     SI,12             ;Mark the file with an 'FF' as the first
        MOV     BYTE PTR [SI],0FFH ; byte so it will not be deleted
      ADD     SI,12

SEIFDN: DEC     BX              ;Subtract 1 from the file counter
        JNZ     CHKLP           ; ask about another if there are more
```

```
                        DX,OFFSET CHKMSG
                        AH,PSTRNG       ;No more filenames in storage so
                        SYS_CALL        ;  ask if they are to be deleted
                        AH,CONIN
DBLCHK:                 INT             ;Get a key from the keyboard
                        CMP             ;
                        JZ              ;  If it's ^C or N then go
                        OR              ;    back to ZDOS
                        CMP             ;
                        JZ              ;  If it's not Y then ignore it and
                        CMP             ;    wait for another input
                        JNZ             ;
                        DBLCHK          ;
                        DX,OFFSET YES   ;  Print 'Yes' on the screen
                        AH,PSTRNG       ;
                        SYS_CALL        ;
                        BX              ;Restore the file counter
                        SI,OFFSET FNPTR ;Repoint to the beginning of the file space
                        DI,OFFSET DFCB
                        CX,12           ;Move the 12 byte filename to the FCB
                        MOVSB           ;  from the storage area
                        DX,OFFSET DFCB  ;Point DX to the FCB
                        AL,DFCB         ;If first byte is 'FF', don't delete it
                        AL,0FFH         ;
                        DONTDEL         ;
                        AH,DFILE        ;  Delete the file if first byte wasn't FF
                        SYS_CALL        ;
DONTDEL:                INT             ;
                        BX              ;Check for any more files in storage
                        DELOOP          ;
                        DX,OFFSET COMPLT ;Send Complete message to the screen
                        TODOS           ;
                        DX,OFFSET ABORT ;Send Aborted message to the screen
                        AH,PSTRNG       ;  then return to ZDOS
                        SYS_CALL        ;
                        RESET           ;
FNPTR:                                  ;Start of filename storage space
                                        ;  storage will overlay help message
HELPMSG                 DB              'Enter  ZERA d:filespec  where d is a drive letter'
                        DB              ' and filespec is a',CR,LF
                        DB              '                  filename which may include'
                        DB              ' * and ? wildcards.',CR,LF
                        DB              '                  Respond with a Y or an N'
                        DB              ' to the questions.',CR,LF
                        DB              '                  Type ^C to abort.',CR,LF,'$'
CODE                    ENDS            ; End of segment
                        END             BEGIN
```

Embellishment With Frames

Alison Phillips
4012 Thororoughgood Drive
Virginia Beach, VA 23455

Provides subroutines for displaying frames paging the display, and utilizing the 25th line on-again off-again as needed.

Are you turned off by routine, unformatted displays? Would you like to add a touch of class to your programming? Is it to much trouble to draw a box or frame on your screen? Dispair not, FRAMES.BAS provides a single routine for printing as many frames as you want, any where you want, and any size that the screen will accommodate.

See Figure 1 for a listing of FRAMES.BAS. Since the program is quite well commented, we can go through it in a hurry. Perhaps when you copy it, you will choose to omit the comments, at least until you get it running. You may be in for a small surprise! The program is written in MBASIC ver. 5.2. If you are familiar with a different BASIC, the changes should be easy.

Note the terminal escape codes. They are very similar to those set forth in David E. Warnick's, "Standards for Terminal Control," in Issue 46 of REMark. You will note that they are 2-letter codes. This allows you to use letter-number codes; for example, T1$, T2$ and the like with little fear of a nasty conflict with an escape code, which can bring about a most perplexing form of error.

Line 210 contains one of my favorite defined functions. It does three things at once. First, it allows the user to directly address the row for printing by substituting the row number for the variable R. Next, it determines the length of the message, X$, to be printed and from the length, directs the cursor to the correct column number for centering the message on the display. Finally, it prints the desired message. See lines 320 - 360 for an example of how the FNC$ defined function is used.

Line 220 is the prosaic direct cursor addressing defined function where the user selects both row and column. It is used in lines 690, 710, and 730. The nice thing about this function is that R and C (short for Row and Column) are place holder variables and can represent an expression or a value. Lines 710 and 730 show R and C replaced with expressions using ROW and COL.

The subroutine which prints the selected frame(s) is contained in lines 670-770. This subroutine is entered with five variables explained in the remarks in line 270. Depending on the user's needs the ROW, COL, W, H, and N variables can be immediately supplied in the program before going SUB, as we did in lines 290 and 300, or they can be read from a data line as shown in line 500.

We mentioned lines 320-360 above in connection with the FNC$ function for centering. But before leaving them for good, please note that the message is included both within the parentheses and within the quotes - a neat little bit of syntax which MBASIC requires.

In lines 380-420, we use a different form of direct cursor addressing that is somewhat more cryptic than the usual defined function. The code is harder to read, but I like it because it is short. Compare the two statements below and note that they both do the same thing.

```
PRINT FNCA$(10,10)"      (19 spaces)
PRINT Y$"))               (11 spaces)
```

The 19 space statement is awkward to use on the terminal when you wish to start printing on the 10th column, because the line which you type extends farther to the right than it will be printed when the program is run. If you are not using this space saving form, refer to your ASCII conversion chart and get used to it. It easy!

Line 440 contains a message we wish to print in the center of the 25th line. But suppose our message is short and there is a longer message on the 25th line? Remember that direct cursor addressing overwrites existing text and a trashy looking line will result if the new line is shorter than the line being overwritten. We GOSUB to 840 to solve this problem.

In line 840, BL$ rings the bell to alert the operator to do something. LO$ clears the 25th line of any existing characters and LX$ enables the 25th line anew. We still must send the the cursor to the 25th line and CHR$(56) does the job (could have used "8" just as well). Where do we get 56? (56=31+25). What is 71 doing in this line? It is the center of the screen (71=31+40). Next, we take 1/2 the length of the string we are going to print and subtract it from 71. For example, suppose our message is 40 characters long, 71 minus 1/2 of 40 is 51 and 51 minus 31 is 20, so we position our cursor for the 20th column. Finally, we print our message, M$, and send our cursor home because if we don't, it will remain there and its awfully hard to read a program on the 25th line.

Now let's look at line 500 which reads values for the frame variables in groups of four from data lines 910 and 920. After reading the data, we GOSUB to our frame printing subroutine and print frames of the size and shape specified in the data elements.

And now to the heart of the program, the subroutine that prints the assorted frames in this program and which will go on printing frames ad infinitum if we tell it what we want. Look at lines 670-770. In 670, we go to the graphics mode, and in line 760, we leave the graphics mode and send our cursor home. Have you noticed that you cannot get a full 24 line page if you leave the cursor on line 23 or line 24 when the program terminates? The printing of the OK prompt and the cursor line feed cause the screen image to jump up. The cursor home, CH$ takes care of this problem.

In line 680 we make N loops. If N=1, then only the outer frame is

printed. If N=10, the line 740 resets the variable on each iteration to print 10 concentric frames if the correct original values are entered as given in line 450. Line 690 prints the top of the frame, line 730 prints the bottom of the frame. We form an inner loop of lines 700-720 to print the sides of the frame. We must loop H times to form the sides of the proper height. Here, it might be well to examine the ASCII chart that Heath furnishes with the computer and see that cdef are the characters that form the 4 corners of each frame and that a lower case "a" forms the top and bottom lines. The character """ is used for the sides.

Line 850 contains a feature I like. Note that it insists on just one response, the carriage return, and nothing else before it disables the 25th line with the LO$ string. This puts tight control into the program. I sometimes let the input of any character suffice for paging, line clearing and the like, but if a key is struck inadvertently action will take place prematurely.

Style in programming is a personal thing. We have opportunities to style each line of code, as well as the structure and appearance of the program as a whole. First, in structuring, there is almost universal agreement that it is well to structure and code the program in coherent modules that perform a specific function. We should format our program so that each module is visibly distinct. Book publishers and advertisers are well acquainted with the power of "white space." White space will enhance the readability and attractiveness of a program.

Second, in coding, a program we can choose between long dense lines and short loose line, and all shades in between. When developing a program, it is a good tactic to code each statement on a separate line, so that MBASIC can lead us quickly to our syntax errors with its powerful and immediate error handling capability. After the program is debugged, statements can be unified in a logical way keeping associated code on a single line. As we learn more about coding, we have a choice of accomplishing an action with a synthesized compound function grouping, or of hacking away at the action an increment at a time. Judgements must be made between juvenile simplicity and arcane sophistication.

What is the price of all of the white space and comments in the program FRAME.BAS? Does it take longer to run? How much more memory does it use and how much more disk space does it require? Is it worth it? The table, which follows, shows values recorded from FRAMES.BAS with an H-89 working at 4 MHz and utilizing 96 tpi Tandon drives.

Since the size of the program and the amount of RAM used fall well below the minimum 16K available in almost all computers, this is not a limiting situation. The variable storage space is obtained by reading the free space after the program is loaded and before it is run, and then again after the program is run. The difference in these two values is the space used by the program for the storage of escape codes, variables, and for housekeeping. Of course, it is the same whether or not the program is commented. Even though it takes longer for the MBASIC interpreter to ignore the comments, the additional time is imperceptible. At 4 MHz, MBASIC will process about 1500 comment lines per second. Therefore, the run time for each version was essentially 11 seconds.

I pay $2.50 for single-sided double-density disks which are formatted to hold 782 kilobytes on the 96tpi Tandon drives. This computes to a little more than 3 kilobytes of storage for a penny. In the case of FRAMES.BAS, the disk space cost of the comments amounted to about two thirds of a cent, which I consider a small price to pay to have a program that is easy to read at some future date. I close with this comment: "Happy and cost effective computering to all!"

	With Comments	Without Comments
Size of program	3676 bytes	1891 bytes
Variable storage	2692 bytes	2692 bytes
Total RAM used	6368 bytes	4583 bytes
Amt of disk space	4 kilobytes	2 kilobytes
Run time	11 seconds	11 seconds
Cost of disk space	1-1/3 cents	2/3 cents

```
10 'FRAMES.BAS     JANUARY 26, 1981     ALISON PHILLIPS
20 '
30 'Written in MBASIC-80 ver 5.2 for H-89 computer. Prints frames.
40 '
50 '=============================================================
60 '                    Terminal escape codes
70 '   Escape      Direct curs   Set 25th      No 25th      Erase line
80 ES=CHR$(27): CA$=ES+"Y": LX$=ES+"x1": LO$=ES+"y1": LE$=ES+"K"
90 '
100 '   Home        Era Display   No Cursor     Cursor On    Bell
110 CH$=ES+"H": ED$=ES+"E": CX$=ES+"x5": CO$=ES+"y5": BL$=CHR$(7)
120 '
130 '  Graphics     Graphic Off   Curs. Save    Cursor Ret   Direct curs
140 GM$=ES+"F": GO$=ES+"G": CS$=ES+"j": CR$=ES+"k": :Y$=ES+"Y"
150 '
160 '=============================================================
170 'Define FNC$ function to center message on any line
180 'Define FNCA$ function to print message on any line
190 'Define integers and clear screen
200 '
210 DEF FNC$(R,X$)=CA$+CHR$(31+R)+CHR$(71-LEN(X$)/2)+X$
220 DEF FNCA$(R,C)=CA$+CHR$(31+R)+CHR$(31+C)
230 DEFINT A-Z:  PRINT ED$ CX$
240 '
250 '           *** MAIN PROGRAM STARTS HERE  ***
260 '=============================================================
270 'ROW=row  COL=column  W=width  H=height  N=number of frames
280 '
290 ROW=1: COL=1:  W=75: H=22: N=2:  GOSUB 670  'Print outside frames
300 ROW=6: COL=26: W=26: H=5:  N=1:  GOSUB 670  'Print inside frame
310 '
320 PRINT FNC$(4,"***  USING FRAMES TO EMBELLISH PROGRAMS  ***")
330 PRINT FNC$(7,"A BASIC TUTORIAL")
340 PRINT FNC$(8,"by")
350 PRINT FNC$(9,"Alison Phillips")
360 PRINT FNC$(10,"January 20, 1984")
370 '
380 PRINT Y$"-+     This program demonstrates how frames, as shown
390 PRINT Y$".+ here, can be used to embellish a display.  Several
```

```
400  PRINT Y$"/+ other displays follow which show other sizes and
410  PRINT Y$"0+ uses of frames.  A set of variables is used to posit-
420  PRINT Y$"1+ ion the frames and determine their sizes and shapes.
430 '
440  M$="PRESS 'RETURN' KEY TO CONTINUE ...": GOSUB 840  'new page
450  ROW=1: COL=1: W=75: H=21: N=10:          GOSUB 670  'make frames
460  PRINT FNCA$(12,30)"** KILROY WAS HERE **"
470  N=1:  GOSUB 840     'new page
480 '
490  FOR I=1 TO 9                'Make a house
500    READ ROW, COL,W,H         'one frame at
510    GOSUB 670                 'a time
520  NEXT
530 '
540  PRINT FNCA$(15,42)"o" FNCA$(13,18) GM$ "aaaa" FNCA$(13,28)"aaaa
550  PRINT FNCA$(13,60)"aaaa" GO$
560 '
570  M$="PRESS ANY KEY TO END PROGRAM .."
580  PRINT LO$ LX$ FNCA$(25,28) M$ : Z$=INPUT$(1): PRINT LO$
590  PRINT FNCA$(23,32)"**  GONE FISHING **"; CO$ FNCA$(21,0)
600  END
610 '
620 '              **  SUBROUTINES START HERE  **
630 '=============================================================
640 '      Prints a single frame, or a series of concentric frames.
650 '      Enter with variables COL,ROW,W,H,& N set.
660 '
670  PRINT GM$
680  FOR J=1 TO N
690     PRINT FNCA$(ROW,COL) "£" STRING$(W,"a") "c"
700     FOR K=1 TO H
710        PRINT FNCA$(ROW+K,COL)"" FNCA$(ROW+K,COL+W+1)""
720     NEXT K
730     PRINT FNCA$(ROW+H,COL) "e" STRING$(W,"a") "d"
740     ROW=ROW+1: COL=COL+2: W=W-4: H=H-2
750  NEXT J
760  PRINT GO$ CH$
770  RETURN
780 '
790 '=============================================================
800 '      Rings bell, erases 25th, enables 25th, directs cursor,
810 '      centers and prints message, and sends cursor home.
820 '      Responds to 'RETURN' only, then clears screen.
830 '
840  PRINT BL$ LO$ LX$ CA$ CHR$(56) CHR$(71-LEN(M$)/2) M$ +CH$
850  Z$=INPUT$(1): IF Z$<>CHR$(13) THEN 850 ELSE PRINT LO$
860  PRINT ED$: RETURN
870 '
880 '=============================================================
890 '      Read in groups of four, ROW,COL,W,H
900 '
910  DATA 4,10,60,6, 10,13,54,8, 11,17,4,4, 11,27,4,4, 11,59,4,4, 11,41,6,7
920  DATA 18,13,54,2, 2,35,6,3, 12,43,2,2
```

*

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.

ACCESS™ with Autopilot™ —

The State of the Art in Communications Programs

- Talk to bulletin boards, information utilities, main frames or other micros
- Send and receive any kind of file—text, binary, .EXE, .COM, etc.
- Transfer files using a variety of protocols, including XMODEM protocol
- Use Autopilot to turn your computer into a communications robot

NEW • Unattended auto-answer mode configures to caller's baud rate, and

NEW • Depending on caller's password, grants unlimited, limited, or no access

NEW • Logs password, duration, date and time of calls you receive

NEW • Logs calls you place—recording number dialed, duration, date & time

NEW • Compatible with all modems, even USR S100 and other board modems

Start Communicating Right Away!

You can start communicating as soon as you pull ACCESS out of its package. We don't make you learn a strange new language of names and symbols the way most programs do. Our customers proclaim it—ACCESS is the easiest and fastest to use!

Autopilot—Your Own Communications Robot

Autopilot can dial, log on, send or receive, answer prompts, even make decisions. Sit back and relax while Autopilot does your work for you.

While you eat dinner or watch TV, have Autopilot call the local bulletin boards, pick up messages to you, drop off messages to others, and collect the classified ads for you to go through later.

Have Autopilot reduce your telephone and timesharing bills by making calls for you late at night, when rates are lowest. When you get up in the morning, your computer can be chock full of fresh information, just like your morning paper.

Instant Links with Other Micros

ACCESS gives you instant phone-links with any other 8-or 16-bit microcomputer for fast, efficient file transfers or just to chat. Or you can direct-cable your computer to others.

And ACCESS answers calls and lets people with one of your passwords have either limited or unrestricted use of your system. . . including freedom to run programs other than ACCESS!

Make Those Tough Connections

Again and again our customers have thanked us for the power to talk with main frames, minis, and specialized networks, with emulators, microcontrollers, and development boards. And you'll thank us too, when ACCESS gets you into systems that defy people with other programs.

Don't Let Your Computer Act Dumb!

Other programs only let your computer act like a dumb terminal while you're on line, but ACCESS leaves the full power of your computer in your hands. Not only can you view your directories and files, you can run your other software—without leaving ACCESS or hanging up the phone.

What Makes ACCESS So Powerful?

To make ACCESS so easy to use, yet fast and flexible, took state-of-the-art programming and years of evolution. To achieve compact power, ACCESS is written in C, the language used to write the famous operating system UNIX. And ACCESS is interrupt-driven for ultimate speed with reliability.

How You Can Get ACCESS

Buy ACCESS from your local Heath or ZDS dealer. If there isn't a dealer near you, order it direct from Hilgraeve Inc.

ACCESS, H/Z-89 with CP/M (2.0 or later and 64K req'd) . . . $49.95
ACCESS, H/Z-110 or 120 with ZDOS, CP/M-85, or CP/M-86 $59.95
ACCESS, H/Z-150 or 160 with MSDOS or CP/M-86 . . . . . $69.95

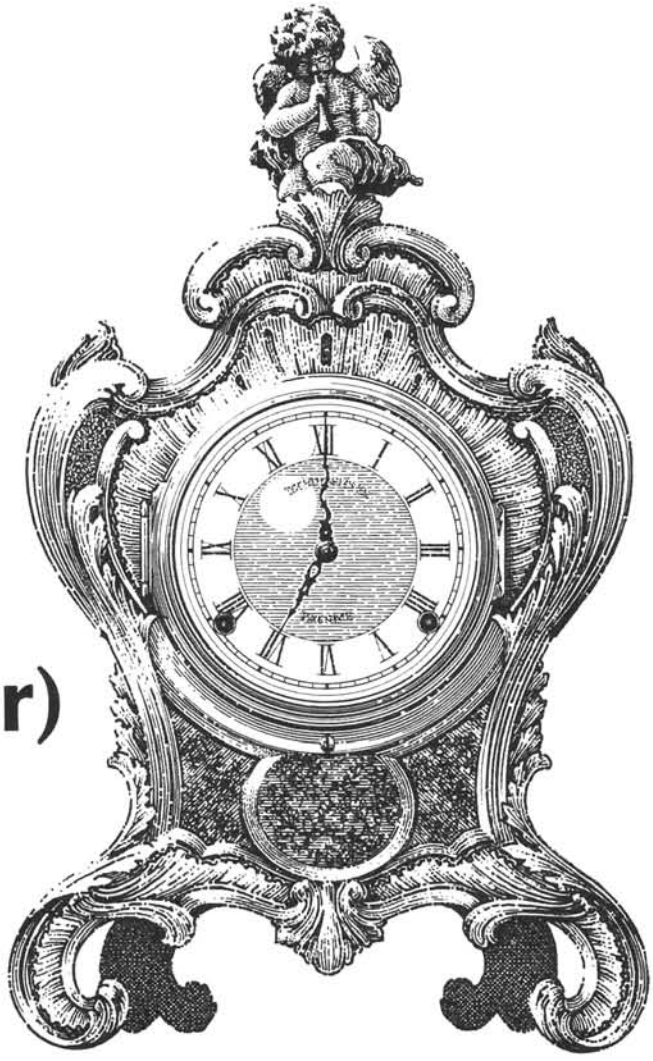
Add $3.00 shipping; MI residents add 4%

Hilgraeve Inc. P.O. Box 941 Monroe, MI 48161 (313) 243-0576

Clock Watcher's Delight

(The Final Chapter)

Pat Swayne
Software Engineer



I made an "oops" in the screen clock programs that were presented in the last two "Clock Watcher's Delight" articles in REMark. That "oops" caused the problems with top of the display "tearing" or the clock display becoming garbled while the clock program was running. It seems that I forgot that whenever you use the string operations of the 8088 processor (MOVS, STOS, etc.), you should ensure that the direction flag is set the way you want it. This oversight was pointed out by HUG member Robert G. Brasfield. You can forever solve the problem of the display messing up by adding a CLD instruction in the SCRNCLK program. For example, in the Z-100 version, where the code now has

```
PTIME:  MOV     AL,CH                   ;GET HOURS
```

you should change it to

```
PTIME:  CLD                             ;ENSURE FWD DIRECTION
        MOV     AL,CH                   ;GET HOURS
```

A similar change should be made to the Z-150 version and to the CP/M-86 version that is on HUG disk 885-5003-37.

The 25th Line

A number of people have asked me how to make the clock appear on the 25th line instead of the top of the screen. I will tell you how to do it, but first let me explain why you shouldn't do it. On a Z-100, the clock will not appear unless the 25th line is enabled, so provision to enable it must be made. Also, there are a number of programs that use the 25th line, and the clock display will over-write a portion of their display. On a Z-150, the 25th line is usually handled like all other screen lines, and a line on it will scroll upwards when a new line is sent to the screen. If you are scrolling text up the screen while you have a 25th line clock on, you will get several clock displays going up the side of the screen. I have found that the upper right hand corner, where the SCRNCLK program normally puts the clock display, is the least used part of the screen.

To modify the Z-100 SCRNCLK to display on the 25th line, locate the line that has

```
        MOV     AX,OFFSET VRAM
```

and change it to

```
        MOV     AX,(OFFSET VRAM)+(128*24)
```

and re-assemble the program. You must also modify the CLOCK program so that the last few lines look like this.

```
CLKON:  MOV     BYTE PTR ES:[BX],1      ;ENABLE CLOCK
        MOV     DX,OFFSET MSG
        MOV     AH,9
        INT     21H                     ;TURN ON 25TH LINE
EXIT:   INT     20H                     ;AND EXIT
MSG     DB      1BH,'x1$'
CLOCK   ENDS
        END     START
```

To modify the Z-150 SCRNCLK program to display on the 25th line, find the label

```
ROW     EQU     0
```

and change it to

```
ROW     EQU     24
```

and re-assemble the program. If the 25th line version works better with certain programs, you can put it on the disk with those programs, and keep the old version on your normal system disks.

We give you our finest!

WIN an H-151 PC Computer System!

HUGGIES everywhere, the perfect partners at your Heathkit Electronic Centers are really giving their best this time! One top-quality H-151 PC computer to be awarded at each of the two big HUG conferences in November.

We offer the kind of support you want. A friendly, trained staff that speaks your language. Plenty of software and great hardware. Plus qualified in-store service on Heath/Zenith equipment.

So, stop by the local stores' booth at either conference. We're showing our support with an H-151 prize donation. And we hope you're one of the lucky ones to win!

Join us at...

Capital HUG Conference,
November 3,
Arlington, VA

Western Regional HUG Conference,
November 10 & 11,
Anaheim, CA

Heathkit®
Electronic
Center *

Where you get more by doing

*Units of Veritechnology Electronics Corporation in the U.S.

VEC-856

Printing Graphics With The H-89 And MX-80A Printer Under HDOS 2.0

R. Kenneth Strum
100 Patrick Drive
Biloxi, M 39531

Ever sat down at your H/Z-89 and wanted to make use of it's graphic capabilities? Ever wanted to print those graphics? If so, this article is for you! We'll examine a program for creating graphics on the H-89 computer and an HDOS 2.0 device driver for printing them on a GRAFTRAX (TM Epson) equipped Zenith or Epson printer.

Our programs for review are: ED-A-SKETCH version 1.1 (The Software Toolworks, 15233 Ventura Blvd, Suite 1118, Sherman Oaks CA 91403), and SCREEN DUMPER (FBE Research Company Inc, Box 68234, Seattle WA 98168). This article was specifically written for HDOS 2.0 users, but if you use CP/M, the information on ED-A-SKETCH will still be beneficial to you because ED-A-SKETCH works the same under both HDOS and CP/M. Dave Brockman, President of FBE Research, Inc. showed me an experimental CP/M screen dumper which worked well. If you're interested in it, call Dave or drop him a line. HDOS users will need the programs discussed and either the Heath/Zenith MX-80A or Epson MX-80 Type III printers, or one of the older Heath/Zenith or Epson MX-80 printers with Graftrax installed.

ED-A-SKETCH is an excellent value program ($30) that gives you "graphics editing" capability on your H/Z-89. The Software Toolworks provides ED-A-SKETCH in Heath 5.25" hard- or soft- sectored HDOS or CP/M formats. The command for this program is: "DRn:SKETCH<cr>." The 64 sector program loads and asks you if you will create a file or read in a previously created file. Once that is done, you're ready to work. Using ED-A-SKETCH is easy. The cursor movements and general operation are almost the same as for PIE 1.5(d) (also by The Software Toolworks). If you use PIE as much as I do, then you'll be comfortable very quickly with ED-A-SKETCH. See Figures 1 through 4 for some examples created using ED-A-SKETCH.

About the Author:

R. Kenneth (Kent) Strum *is an Administration Management Officer for the Air Force. Currently, he is stationed at Keesler AFB Mississippi as Master Instructor, and Chief, Administration Management Officer Course. He is scheduled to move to Brooks AFB at San Antonio Texas in November. He built his HN-89A in 1983 to learn how computers work and for teaching office automation. He has also been involved in several computer projects, both for the Air Force and in his local community. Kent's interests include military history, low cost office automation, personal computing, photography, camping, and his family, including his wife, Pam, and their three boys.*

Moving the cursor around, typing text and creating graphics aren't the only things ED-A-SKETCH does. It has a cut and paste function called "pick & put." This allows you to create an image and then paste it wherever you want on the screen. If you do mess up, you can use the "oops" key to restore your picture. An unusual feature is "relative positioning." You can use this feature to animate pictures for use on the screen. Once you are finished sketching, you exit by telling ED-A-SKETCH you are through with the screen. ED-A-SKETCH will ask you which format you want to save the picture in. The default is "picture" (which is the only way of reading a picture back into ED-A-SKETCH). You can also save it as a BASIC (either Benton Harbor or MBasic) file (already numbered!), an assembly language file, or even as a C language file. ED-A-SKETCH allows you to clear the screen and then read in another file. If you choose, you can use "save" to write your work to disk every few minutes. I recommend doing this because it has helped me during power interruptions, especially those caused by my three very inquisitive sons.

ED-A-SKETCH's power lies in it's commands and ability to mix text with graphics. There are two basic modes of operation for ED-A-SKETCH: text and graphics. You can work with both in either normal or inverse video. You also have the option of using inverse video in a defined area. There is a 25th line which displays all the graphic characters and the keys needed to produce them. If you want, you can turn this line off. Cursor positioning is simple. The cursor keys move you left, right, up, or down. Pressing <SHIFT> at the same time moves you to the edge of the screen in the direction indicated. <HOME> moves the cursor to the top left position on the screen. <TAB> moves the cursor to the predetermined tabs, at eight space intervals to the right, while <ESC> acts as a backtab. <RETURN> moves the cursor down one line to the leftmost position. <DEL> moves the cursor one space to the left and deletes that character. <BACKSPACE> moves the cursor one space to the left and inserts a blank in place of the character. The distinction between <DEL> and <BACK SPACE> is important only during animation. One of ED-A-SKETCH's fine features is it's "OOPS" key, found as the <.> key on the numeric keypad. It undoes the last action. It is a real help when hitting the wrong key, but it is also helpful as an aid in comparing two different drawings. "Painting," "cutting," and "pasting" are the action categories of ED-A-SKETCH. Painting consists of the functions: fill line to right, fill line to left, fill area, fill to bottom, fill from top, erase, and invert. The "fill line" commands are useful in drawing straight lines. The fill area commands are useful in filling areas

with either ASCII or graphic characters. The "fill to bottom" and "fill from top" are specialized fill area commands that allow you to fill below or above the cursor position. Erase replaces text and graphic characters with spaces. Invert allows you to put a defined area in reverse video. The lower right corner of Figure 2 demonstrates this. All the painting functions allow "area definition." That is, you can act on a desired line, square, or rectangle. ED-A-SKETCH's cutting function is known as "Pick." It copies everything within a defined area into a buffer. Unlike physical cutting, the original area remains intact. The pasting function is known as "put." You place the cursor where you want the top left part of the image to go and press the "put" key. Presto! The image you "picked" is "put" down. Pick and Put give an option for working with blanks and spaces also.

SCREEN DUMPER costs a modest $15 and runs under HDOS 2.0 on GRAFTRAX equipped Heath/Zenith and Epson printers. If you have HDOS and one of the printers listed above, you'll want SCREEN DUMPER. SCREEN DUMPER is really three device drivers: DH.DVD (for Driver Horizontal), DV.DVD (for Driver Vertical), and QD.DVD (for Quick Dump). They are 20, 18, and 14 sectors in size, respectively. DH: uses 960 dots per inch (dual density) to print straight across the paper, while DV: uses 480 dots per inch (normal density) to print sideways (top left on the screen prints as bottom left on the paper, a 90 degree turn left), and QD: prints graphics as ASCII characters. I found QD: to be useful in a number of applications. One example is dumping a text image from the screen. See Figure 5 for an example of a SET LP: configuration I dumped for reference. SCREEN DUMPER accurately portrays each dot on the screen to a corresponding dot on paper. According to Dave Brockman of FBE Research Company, the H/Z-89 screen is 640 dots wide and 240 dots high when matched to the MX-80's GRAFTRAX OPTION. You can see from the figures the quality product SCREEN DUMPER produces. The command for SCREEN DUMPER is: "COPY TT:=SYn:FILENAME.EXT,DH:" (or DV: or QD:). FBE gives "COPY DH:=" as the command to dump a current screen, but I couldn't get it to work on either my "SUPER SYSMOD2" modified, or "straight" HDOS 2.0 systems. The picture created in the picture mode by ED-A-SKETCH is dumped first to the screen and then to the printer. SCREEN DUMPER has provisions for SET options which allow you to set it for your own needs. Be sure to SET all the drivers to your hardware environment before you use them. This will save you much frustration later.

The DH: and QD: drivers allow you to print more than one screen on a page. DH: allows up to three screens to print without losing the top of form, while QD: allows two screens. Both drivers allow either formfeeds or no formfeeds after print operations.

Here are some hints on how to use ED-A-SKETCH and SCREEN DUMPER if you've modified HDOS through SUPER SYSMOD2 or another similar mod that lists files copied before the files are copied. ED-A-SKETCH prepares screens correctly under HDOS or modified HDOS. However, when you dump the image to screen by typing:

```
COPY TT:=DRn:FILENAME.EXT" <CR>
```

the image rolls up and the top line or two is lost off the screen and will not print correctly. If you are using unmodified HDOS, SCREEN DUMPER will print the screen correctly. However, if you are using a SUPER SYSMOD2 or similar modification, a line will print: "DRn:FILENAME.EXT<cr> N FILES COPIED" on the printer. I avoid this distortion by using an unmodified HDOS 2.0 system disk when I print. I prepare the files normally, using the convenience of my SUPER SYSMOD2 modified system disk. Then, at the end of the session, I reboot with the unmodified system disk, and print out all my files. If you try to use SUPER SYSMOD2's "period" command to keep the picture from scrolling, you'll still get the "DRn:FILENAME.EXT" on your printout.

No product review would be complete without pointing out limitations. Both ED-A-SKETCH and SCREEN DUMPER have them. ED-A-SKETCH's most serious limitation is it's inability to deal with more than one screen of information at a time. This inhibits printing full page graphic drawings. However, ED-A-SKETCH isn't marketed as a printing program. It is sold as a screen graphics display program. A second limitation is ED-A-SKETCH's inability to read in several files at a time the way their MYCALC spreadsheet program does. This limitation prohibits you from creating alphabet or graphic templates and reading them in as needed. ED-A-SKETCH, unlike programs such as WordStar, doesn't allow you to read a directory while in the program. So, you'll need to have the drive and filename of all the files you plan to work with. I use a printed copy of the disk directory. I find keeping a copy of all my work disk directories an easy and cheap way to keep up with my work files. I simply print a new directory at the end of every work session. SCREEN DUMPER also has a limitation: it lacks side and top margin settings. As SCREEN DUMPER comes, the horizontal driver gives a 1-1/8 inch left margin and a 2-3/8 inch right margin when printing an eighty column screen. The vertical driver starts it's image at the top and 5/8 inches from the left margin. I would like to see SCREEN DUMPER have margin settings so the image being printed could be centered.

Lest you think ED-A-SKETCH and SCREEN DUMPER are the only products available for creating graphics and then printing them, here are some other products that I'm aware of. This, however, is not intended to be a comprehensive list of available software.

VIDEO ARTIST PLUS, marketed by Newline Software (Box 289, Triverton, RI 02878) is a graphics editor much like ED-A-SKETCH. It costs about $40 and is available only in the HDOS format. Newline also markets a couple of HDOS device drivers that allow you to process graphics to an MX-80 printer. I use the Softshop's MX.DVD (also marketed by Newline), but haven't used that feature because the ED-A-SKETCH and SCREEN DUMPER are so convenient. Newline also markets THE ILLUSTRATOR, a program which works with IMAGINATOR graphics board by Cleveland Codonics, Inc. It produces high quality graphics on the H/Z-89 and will print them on several different printers, including the Zenith/Epson models mentioned. A version is also available for the Z-100. Price: about $90.

SKYCASTLE COMPUTER PRODUCTS (Box 1412, San Carlos CA 94070) markets a CP/M program for creating printed graphics on the MX-80 printer with GRAFTRAX. Price: about $50.

In summary, both ED-A-SKETCH and SCREEN DUMPER are excellent products with some limitations. ED-A-SKETCH allows both HDOS and CP/M users to create screen drawings using the H/Z-89's graphic character set. SCREEN DUMPER is an HDOS device driver that allows owners of H/Z-89's and GRAFTRAX equipped Zenith or Epson printers to print the drawing created by ED-A-SKETCH or other screen graphic editors. Both ED-A-SKETCH and SCREEN DUMPER are priced right. There are other products, which I haven't used, that also might be very useful in creating and printing H/Z-89 graphics.

I hope you have as much fun as I do by creating and printing your own graphics. Happy Heath/Zenith computering!

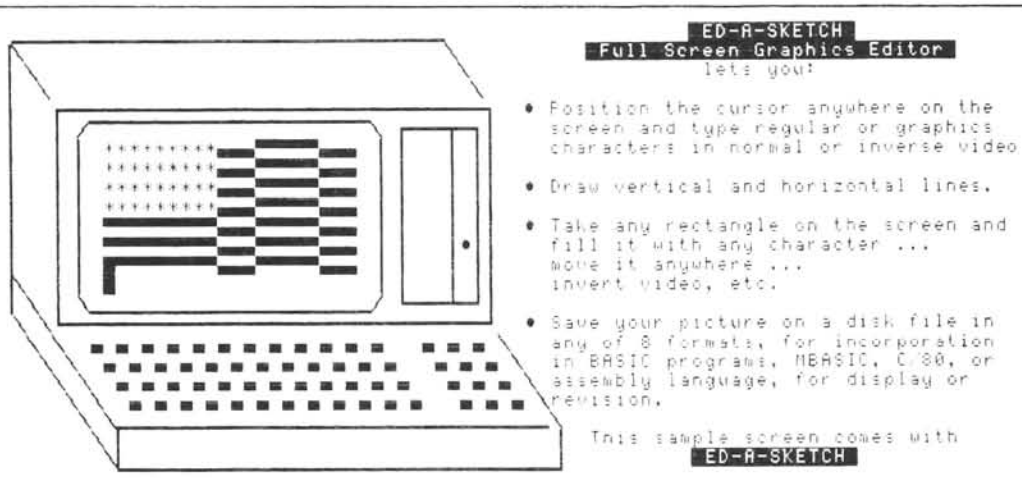


Figure 1

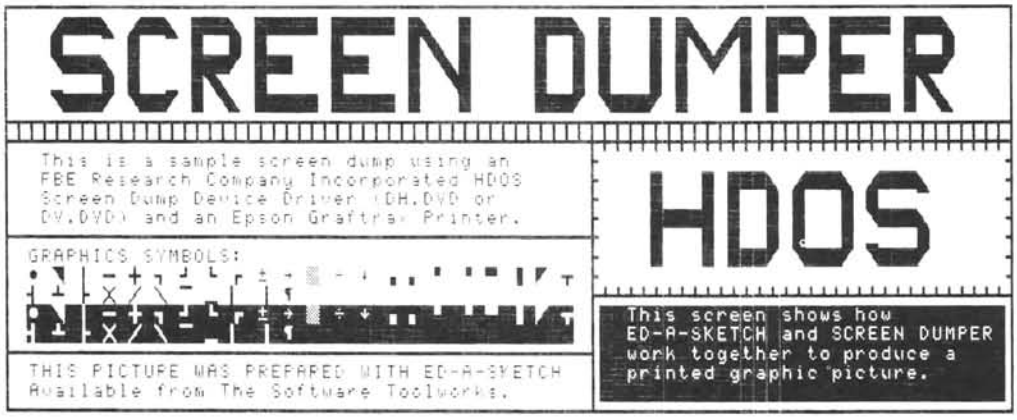
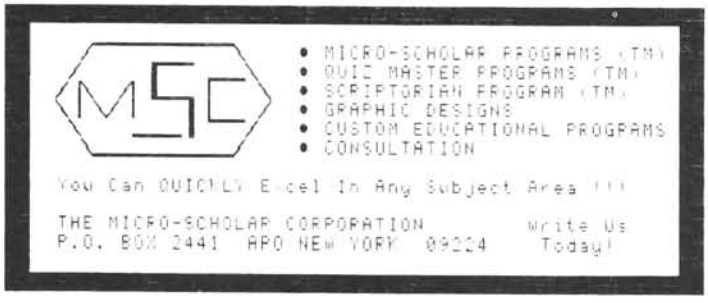


Figure 2

The Software Toolworks provides a sample alphabet with ED-A-SKETCH.

Figure 3



Tim Stanford, owner of THE MICRO-SCHOLAR CORPORATION used ED-A-SKETCH and SCREEN DUMPER to produce this business card on his H-89 and Epson MX-80 printer.

Figure 4

```
Set Option Status  #  MX80.DVD - Printer Device Driver  #  Version 1.2

Form        Feature          LP0:  LP1:  LP2:  LP3:  LP4:  LP5:  LP6:  LP7:
PORT n      Port Address     340Q  340Q  340Q  340Q  340Q  340Q  340Q  340Q
BAUD n      Baud Rate        4800  4800  4800  4800  4800  4800  4800  4800
FORM n      Lines/Form       066   066   066   066   110   066   066   066
PAGE n      Lines/Page       060   060   066   000   000   000   000   000
CPI n       Characters/Inch  017   010   010   010   010   017   010   010
LPI n       Lines/Inch       006   006   006   006   010   006   006   006
RMARG n     Right Margin     255   255   255   255   255   255   255   255
LMARG n     Left Margin      017   000   000   000   000   000   000   000
PAG N or Y  Pagination       Yes   No    Yes   No    No    No    No    No
MUC N or Y  Multiple Copies  No    Yes   Yes   Yes   No    No    No    No
TAB N or Y  TABs to Printer  No    No    No    No    No    No    No    No
DBL N or Y  Double Space     No    No    No    No    No    No    No    Yes
FFF N or Y  Final Form-Feed  Yes   Yes   Yes   No    Yes   No    No    No
EMP N or Y  Emphasize Print  No    Yes   Yes   Yes   Yes   No    No    No
ASC N or n  ASCII String     No    No    No    No    No    No    No    No
H>CO QD:=
ERROR - ?02 Illegal Format for File Name
H>CO QD:=TT:
```

Figure 5

Z-FORTH: A BASIC Approach To Forth

Ralph Nelson
205 Mercury Road
Newark, DE 19711

If you know nothing about FORTH, read a description of it before proceeding. The book "Starting Forth" (by Leo Brodie, $7, FORTH, Inc.) might be a good place to start. If you know something about FORTH, but have been frustrated in implementing it, this article should allow you to make rapid progress in writing error-free FORTH programs. You must be able to write programs in BASIC (such as Benton Harbor BASIC) and to have access to a FORTH system (such as HFORTH79, by Pat Daugherty, $39, SoftShop).

As a programmer who is quite familiar with BASIC, I have been annoyed to find that many descriptions of FORTH are nearly impossible to follow. In 1958, I learned to program in FORTRAN II, and in 1965, I had no trouble learning BASIC. But even well written books and articles did not make programming in FORTH easy for me. I was quite eager to use FORTH for some programs which take too much time in BASIC, so I was forced to develop a solution to my problem. Success came through writing a translation table which gives the FORTH equivalents for the most common program elements in BASIC. I can now write a program in BASIC, debug it with the many convenient features of BASIC, then translate it to a very fast FORTH program by writing out the elements from my translation table. I call my technique "Z-FORTH."

Some might argue that novices ought to learn FORTH by itself, rather than through another computer language. But I think that the use of Polish notation and stacks is so foreign to people with no computing background that they will be overwhelmed by the new concepts. They may quit before they get a simple program to work properly. Consequently, I suggest that a new programmer become familiar with BASIC to see what computing is all about, then make the transition to FORTH through Z-FORTH. This may not give the most efficient code, but at least novices can get started on interesting programs with some confidence that the finished code will work. As the novices study books on FORTH and come to understand the concepts and the more advanced operations, they can upgrade the Z-FORTH programs.

My table gives the BASIC and FORTH code for twelve program elements. For clarity, I have included spaces in the BASIC elements, even though spaces are required in only a few places. Spaces are required between all words in FORTH, and I have added a few extra spaces for clarity. Note that the amount of typing required to write the code is about the same for FORTH as for BASIC.

To make comparisons with the FORTH version easier, I have generally listed only one BASIC statement per line. Most versions of BASIC permit several statements (separated by colons) on one line. FORTH requires new lines only for clarity, so the examples given here could be compressed to fewer lines with no special characters added.

Many versions of BASIC limit variable names to a single letter, plus an optional single number with $ added if the variable is a string. In HFORTH79, a name may contain up to 32 characters. The name must start with a letter, but doesn't need a $ or other character to indicate that the reserved memory will contain a string. To maintain parallelism, I have used the same names in FORTH as in BASIC, but I could have used PLAYER-NAME instead of P$ in the HIROLLER program example.

Since FORTH allows the use of multi-letter names for subroutines (as well as for variables), a FORTH program is easy to understand without many remarks. If you wish to include comments, use REM xxxx in BASIC and {(xxxx)} in FORTH. Since FORTH used many punctuation marks as commands, I have enclosed FORTH statements in curly braces to set them off from the text. The unconditional jump to a specified line number (GOTO in BASIC) does not exist in FORTH. There are no line numbers, and all non-sequential execution paths are part of {IF...THEN...ELSE}, {BEGIN...UNTIL}, or {DO...LOOP} structures or subroutine calls and returns.

Some Comments On The Elements Of BASIC VS FORTH:

#1. We want to assign values to variables within the program. In FORTH, we write the value, then the name of the variable to which it will be assigned, then {!} to store the value at the address allocated to the variable. In BASIC, we need to allocate memory only if we plan to use an array. To do this, we write DIM A(50). In FORTH, we must allocate memory for every variable used. It is most convenient to do this for all program variables at the start of the program, so that we won't have to search through the whole program to see which names have been used. I shall illustrate the reservation of space in only a few examples below.

#2. We want to prompt the user to type in values for a variable while the program is running. In BASIC, we put the prompt inside quotation marks just after the INPUT statement. In FORTH, {.'xxxx'} fulfills the same function as a BASIC PRINT 'xxxx' statement, while the numeric input statement in FORTH is {#IN}. A carriage return-line feed is implicit in the BASIC statement. It must be explicitly stated using {CR} in FORTH.

#3. We want to display values, usually with some text to label what the value represents. The command {?} in FORTH displays the value for the variable name preceding it.

#4. We want to do algebraic operations. In FORTH, we must contend with reverse Polish notation. This is really a very convenient and efficient system, but you have to get used to putting two values on the stack before doing a binary algebraic operation. The stack is like a pile of cards with numbers written on them. The numbers may represent values, addresses, or ASCII characters. FORTH uses several stacks. The term "stack" when used by itself in this article refers to the program stack.

The command {@} fetchs the value for the variable whose name just preceeds it and puts that value on top of the stack. We do this twice. The {-} operation is then performed, leaving the result on the stack in place of the two values previously on the stack. {C !} then stores this result in memory. Because FORTH stacks the result of a computation, it doesn't use the = symbol. And since FORTH executes code strictly from left to right, the name of the variable to which the result is assigned (C in our example) comes after (to the right of) the algebraic expression, rather than before as in BASIC.

#5. We want to assign a string of alphanumerics to a variable. A string is usually several characters long, so we must allocate adequate memory for assigned or typed-in strings. BASIC automatically allocates memory for strings. In FORTH, we must reserve one more space than the number of characters we expect to enter. The {1 A$} puts on the stack the address of the first byte allocated to the A$ array. The statement {!' xxxx'} stores xxxx in memory starting at that address.

#6. We want to prompt the user to type in strings. Note that we may initialize an array by filling it with blanks (32 is the ASCII number for a blank). {max# EXPECT} takes in as many as max# characters until <CR> is pressed or max# of characters have been typed in, at which point it stores them starting at the address on the stack beneath max#.

#7. We want to display stored strings. In FORTH, we must specify the starting point within the array and the maximum number of bytes to be typed. To eliminate the blanks at the end of a string (which may not fill the whole array), use the command R{-TRAILING}.

#8. We want to repeatedly execute a section of program, while incrementing an index. The usual structure for this is the loop, which in its simplest form has an integer index starting at 1 (one) and incrementing by one to a maximum (N9 in the example). When {DO} is encountered in FORTH, the two top values on the program stack are transferred to the loop stack. The loop index is the top value on the loop stack. It increments to a maximum that is one-less-than the next-to-top value on the loop stack. In the example, we set up this maximum using {N9 @ 1 +}.

The loop index may be copied onto the program stack by simply writing {I}. Since {I} is the name of a procedure (not a variable) in FORTH, it should not be used as a variable name. (Neither should J, K, or I1.) We need not set up a separate variable for the loop index comparable to the N in the BASIC example. You may store values at a particular index location in an array by putting the value, then the index, then the array starting address on the stack, then giving the store command, as in {6 I E !}.

#9. We want execution to continue (branch to) at different places in the program, depending on whether the result of a specified comparison is true or false. In BASIC, we may branch to a line which comes either later or earlier in the program. FORTH does not use line numbers, and the statements required to branch to an earlier part of the program (a backward branch) are different from those used to branch to a later part (a forward branch). I shall discuss backward branching in #10.

The branch statement in BASIC consists of IF, the condition to be tested, THEN (on the same line), the true-action (on the same line), and the false-action (on succeeding lines). The true-action may include a GOTO, with execution proceeding at a distant point in the program, either before or after the branch point. BASIC does not require that the two execution paths merge again after the IF. FORTH does require that they merge. The condition is true if after a comparison in FORTH, the value at the top of the stack is not zero. For a true condition, the operations between {IF} and {ELSE} are executed. The condition is false if the top-of-stack value is zero. For a false condition the operations between {ELSE} and {THEN} are executed. In either case, execution then continues with the operations following {THEN}.

#10. To set up a backward branch in FORTH, we use {BEGIN} to indicate where to branch back to. The condition test occurs at the word {UNTIL}. If the condition is false when the program encounters {UNTIL}, then execution continues at {BEGIN}. If the condition is true, execution continues with the word after {UNTIL}. {BEGIN...UNTIL}, {IF...ELSE...THEN}, and {DO...LOOP} pairs may be nested, just as FOR...NEXT loops are in BASIC.

#11. We want to set up a subroutine (a common set of operations which will be executed at several points in a calculation). In BASIC, we write out a set of statements ending with RETURN. The Rsubroutine may be placed either before or after the call statement (GOSUB 1000) in a BASIC program.

In FORTH, we define a new word by starting the subroutine with {:}, followed by the name of the subroutine, then the statements, and ending with {;}. The definition of a word must precede any use of that word in FORTH. This contributes to the odd appearance of many FORTH programs, since in our top-down thinking, we expect the major structure of a program to be at the start, with the details handled below. It is possible to write FORTH programs so that blocks at the end are compiled before those appearing first in the listing. I have done this for the HIROLLER example to make it more parallel to the BASIC listing and because I believe that this is the most sensible way to present a FORTH listing. The statement {1005 LOAD} compiles block 1005 before proceeding with compilation of the current block.

#12. To use a subroutine in BASIC, we use GOSUB and specify the line number at which it starts. In FORTH, we simply use the names of the routines, which in our example might refer to writing a split octal address, moving a block of data, and scoring points.

A Worked Example Of A Conversion

The listings of HIROLLER show what a simple BASIC program looks like when translated into FORTH using my table of elements. I start with a top-down program outline, which should be the first step in developing any program. The operations enclosed in square brackets refer to subroutines. These are written out in detail later on in the outline. When you have read and understood this example, you should be ready to convert one of your own BASIC programs to FORTH. Good luck, and may Z-FORTH be with you!

The Twelve Program Elements In BASIC Versus FORTH

#1. To assign a value:

```
100 A=3
```

vs

```
VARIABLE A  3 A !
```

#2. To prompt for a value:

```
110 INPUT "Enter a number -> ";B
```

```
vs
  VARIABLE B   ." Enter a number -> " #IN  B !  CR
```

#3. To display a value:

```
  120 PRINT "Second Addend =";B
vs
  ." Second Addend ="  B ?  CR
```

#4. To subtract:

```
  130 C=A-B
vs
  A @ B @ - C !
```

#5. To assign a string:

```
  140 A$="Earnings"
vs
  81 CARRAY A$  1 A$ !" Earnings"
```

#6. To prompt for a string:

```
  150 LINE INPUT "Enter a string -> ";B$
vs
  81 CARRAY B$  1 B$ 80 32 FILL
  ." Enter a string -> "  1 B$ 80 EXPECT
```

#7. To display a string:

```
  160 PRINT B$
vs
  1 B$ 80 -TRAILING TYPE  CR
```

#8. To do an integer loop:

```
  170 DIM E(32)
  175 N9=5
  180 FOR N=1 TO N9
  190 E(N)=2*N
  200 NEXT N
vs
  32 ARRAY E    VARIABLE N9
  5 N9 !
  N9 @ 1 +  1 DO
  I 2 *  I E !
  LOOP
```

#9. To do a forward branch:

```
  210 IF X>1 THEN X=1: Y=0: GOTO 230
  220 X=0: Y=1
  230 PRINT X,Y
vs
  X @ 1 > IF  1 X !  0 Y !
  ELSE 0 X !  1 Y !
  THEN X ?  Y ?  CR
```

#10. To do a backward branch:

```
  320 X=0
  330 X=X+1
  340 INPUT "Enter 0 to sum, 1 to stop -> ";Q
  350 IF Q<>0 GOTO 330
  360 PRINT X
vs
  0 X !
  BEGIN  X @ 1 + X !
  ." Enter 0 to sum, 1 to stop -> "  #IN
  1 =  UNTIL
  X ?  CR
```

#11. To setup a subroutine:

```
  500 REM ---SPLIT OCTAL ADDRESS---
  510 H=M/256: L=M-256*H
  520 PRINT "split octal:";H;".";L: RETURN
vs
  \ ---SPLIT OCTAL ADDRESS---
  : SPLIT  M @ 256 / H !  M @ H @ 256 * - L !
  ." split octal:" H ?  ." ." L ?  CR  ;
```

#12. To call subroutines:

```
  250 GOSUB 500: GOSUB 1530: GOSUB 2350
vs
  SPLIT  MOVE  SCORE
```

Top-down Programming Outline

Description: This is a dice game for 2-5 players, who start with 10-1000 dollars each and use 2-5 (standard) dice in 1-10 rounds of play. At the start of the game and the end of each round, the order of play is reset so that the players with the most remaining cash go first. Players must bet something each round if they have anything left. The players who go last have an advantage, in that they can see what dice earlier players have thrown, so they may bet only a little if there are high rolls on the table. (This is not a very sporting game.) At the end of a round the pot is given to the high roller and the players positions are summarized on the screen.

Hiroller:

Print title
Print rules
[INITIALIZE PLAY]
For each round
    [PLAY A ROUND]
Print ending message

Initialize Play:

Get number of players (2-5) [CHECK]
For each player
    assign order = loop index
    get name
    get starting dollars (10-1000) [CHECK]
[RESET ORDER OF PLAY]
Get number of rounds (1-10) [CHECK]
Get number of dice (2-5) [CHECK]

Play a Round:

For each player
    print name and dollars left
    get bet (>0,<=dollars left) [CHECK]
    adjust dollars left for player
    add bet to pot
    [THROW DICE]
    keep index of highest roller to date
Print winner's name
[RESET ORDER OF PLAY]
[SUMMARIZE SCORES]

Reset Order of Play:

For each player except last
    for players not yet compared
        compare dollars
        if not in hi-lo order, swap order

Summarize Scores:

For each player
    print name, dollars left

Throw Dice:

Set sum = 0
For each die
    generate random number (1-6)
    print it
    add to sum
Print sum

Check:

* prompt for input value with limits
check input against hi-lo limits
if not OK, print error message, go back to *

Variable Names:

I kept these the same in FORTH as in BASIC, but could have used longer and more descriptive names in FORTH.

Arrays

P$ - player name
D - player dollars
O - order of player indices

Variables

P9 - number of players
R9 - number of rounds
D9 - number of dice
N - order of current player
N2 - order of comparison player
Q - index of high roller
S - dollars in pot
T - roll of current player
T1 - roll of highest roller to date
V - value of current die
Z - input value
Z1 - lower bound
Z2 - upper bound

Variables Used In BASIC Only
(stack used to hold these in FORTH)

B - player's bet
D - index of die throw
P - index of current player
P2 - index of comparison player
R - index of round

BASIC Program Listing

```
00100 REM --- HIROLLER ---
00130 DIM P$(5),D(5),O(5)
00140 PRINT :PRINT :PRINT
00150 PRINT "84.06.06  *** HIROLLER *** by Ralph Nelson"
00152 PRINT :PRINT
      "This is a dice game with several rounds."
00154 PRINT
  "The high roller or first to roll a high tie wins."
00156 PRINT
  "The order is from highest dollars to lowest."
00158 PRINT
  "If dollars are the same, use previous order of play."
00210 GOSUB 300:REM initialize play
00230 FOR R=1TO R9:GOSUB 400:REM play a round
00240 NEXT R
00250 PRINT :PRINT "----- The game is now over -----"
      :PRINT
00260 STOP
00300 REM --- SETPLAY ---
00310 PRINT "Number of players (2-5)";:Z1=2:Z2=5
      :GOSUB 700:P9=Z
00320 FOR P=1TO P9:O(P)=P:REM initialize order index
00325 PRINT :LINE INPUT "Name of player -> ";P$(P)
00340 PRINT "Starting dollars (10-1000)";:Z1=10:Z2=1000
      :GOSUB 700:D(P)=Z
00345 NEXT P:PRINT
00350 GOSUB 500:REM reset order of play
00360 PRINT "Number of rounds to play (1-10)";:Z1=1
      :Z2=10:GOSUB 700:R9=Z
00370 PRINT "Number of dice (2-5)";:Z1=2:Z2=5
      :GOSUB 700:D9=Z
00390 RETURN
00400 REM --- ROUND ---
00405 S=0:Q=0:T1=0
00410 FOR P=1TO P9:N=O(P)
00420 PRINT :PRINT P$(N);" has $";D(N);"."
00422 IF D(N)=0THEN PRINT "Sorry, you are out!"
      :GOTO 470
00425 PRINT "Place your bet (>0)";:Z1=1:Z2=D(N)
      :GOSUB 700:B=Z
00440 D(N)=D(N)-B:S=S+B
00450 GOSUB 600:REM throw dice
00460 IF T>T1THEN Q=N:T1=T
00470 NEXT P:D(Q)=D(Q)+S:PRINT :PRINT
      "The hi-roller this round is ";P$(Q)
00480 GOSUB 500:REM get new order of play
00485 GOSUB 650:REM summarize scores
00490 RETURN
00500 REM --- ORDER ---
00510 FOR P=1TO P9-1:FOR P2=P+1TO P9
00515 N=O(P):N2=O(P2):REM get indices for two players
00517 REM if dollars are not in hi-lo order,
      swap indices in the order
00520 IF D(N)<D(N2)THEN Z=O(P):O(P)=O(P2):O(P2)=Z
00530 NEXT P2:NEXT P
00540 RETURN
00600 REM --- DICE ---
00610 T=0:PRINT "Your dies are";
00620 FOR D=1TO D9:V=INT(RND(1)*6+1):PRINT V;
00630 T=T+V:NEXT D
00640 PRINT "  Your roll = ";T:RETURN
00650 REM --- SCORES ---
00660 PRINT :PRINT "Dollars left:"
00670 FOR P=1TO P9:N=O(P):PRINT P$(N),D(N):NEXT P
00680 PRINT :RETURN
00700 REM --- CHECK ---
00710 INPUT " -> ";Z:IF Z1<=Z AND Z<=Z2 THEN RETURN
00720 PRINT "*** bad, try again";:GOTO 710
```

FORTH Program Listing

The numbers at the top are the program block numbers, used for editing and for loading the program through the compiler.

```
------ 1001 ----------

4 0 DO 1005 I - LOAD LOOP
( This loads blocks in good FORTH order.)
```

```
: HIROLLER
   CR CR CR ." 84.06.01 *** HIROLLER *** by Ralph Nelson" CR
   CR ." This is a dice game with several rounds." CR
   ." The high roller or first to roll a high tie wins." CR
   ." The order of play is from highest dollars to lowest." CR
   ." If dollars are the same, use previous order of play." CR
   SETPLAY  R9 @ 0 DO  ROUND  LOOP
   CR ." --- The game is now over. ---" CR ;

------ 1002 ----------
: SETPLAY
   CR ." Number of players (2-5)"  2 Z1 ! 5 Z2 !  CHECK
   Z @ P9 ! P9 @ 1 + 1 DO  I I 0 !
   CR ." Name of player -> " I 1- 80 * P$ 80 EXPECT  CR
   ." Starting Dollars (10-1000)"  10 Z1 ! 1000 Z2 !  CHECK
   Z @ I D ! LOOP  ORDER
   CR ." Number of rounds (1-10)"  1 Z1 ! 10 Z2 !  CHECK
   Z @ R9 !
   ." Number of dice (2-5)"  2 Z1 ! 5 Z2 !  CHECK
   Z @ D9 ! ;

------ 1003 ----------
: ROUND  ( play one round)
   0 S ! 0 Q ! 0 T1 !
   P9 @ 1 + 1 DO  I 0 @ N !
   CR  N @ 1 - 80 * P$ 80 -TRAILING  TYPE
   ." has $" N @ D ? CR
   N @ D @ 0 = IF ."  Sorry, you are out!" CR
   ELSE ."  Place your bet"  1 Z1 ! N @ D @ Z2 !  CHECK
   N @ D @ Z @ - N @ D !  Z @ S @ + S !
   DICE  T @ T1 @ > IF  T @ T1 !  N @ Q !  ELSE  THEN
   THEN  LOOP  S @ Q @ D @ + Q @ D ! CR ." The hi-roller is "
   Q @ 1 - 80 * P$ 80 -TRAILING  TYPE  CR
   ORDER  SCORES ;

: ORDER ( reset order of play)
   P9 @ 1 DO  P9 @ 1+ I 1+ DO  J 0 @ D @  I 0 @ D @  >
   IF  ELSE J 0 @ I 0 @  J 0 ! I 0 !  THEN LOOP LOOP ;

------ 1004 ----------
: DICE  ."  Your dice are "  0 T !  D9 @ 0 DO
   5 RANDOM 1 + V !  V ? V @ T @ + T !  LOOP
   ."  Your throw = " T ? CR ;

: SCORES  ( list players and their dollars)
   CR ." Dollars left:" CR
   P9 @ 1 + 1 DO
   I 0 @ 1 - 80 * P$ 80 -TRAILING  TYPE
   ."  $" I 0 @ D ? CR LOOP CR ;

: CHECK ( get & check value within limits)
   BEGIN  ."  -> " #IN CR Z !
   Z2 @ Z @ >=  Z1 @ Z @ <= AND
   IF 1 ELSE ." *** bad, try again" 0 THEN UNTIL ;

------ 1005 ----------
400 CARRAY P$  1 P$ 400 32 FILL
5 ARRAY D  5 ARRAY 0
VARIABLE P9  VARIABLE R9  VARIABLE D9  VARIABLE T
VARIABLE S  VARIABLE Q  VARIABLE N  VARIABLE T1
VARIABLE Z  VARIABLE Z1  VARIABLE Z2  VARIABLE V
```

Giant Heath/Zenith Computer Hardware and Software Sale

Here's the sale you've all been waiting for. Until December 15, 1984, you can save hundreds of dollars on hardware and software for your Heath and Zenith computers. Now's the time to buy!

HARDWARE

Model No.	Product	List	Save	Now Only
H-125	Dot Matrix Printer	$ 899.00	$100.00	$799.00
Z-37	Dual 96 TPI 5¼" Disk Drive	1199.00	400.00	799.00
Z-87-89	Dual 48 TPI 5¼" Drives for H/Z-89	659.00	160.00	499.00
Z-87-90	Dual 48 TPI 5¼" Drives for Z-90	659.00	160.00	499.00
H-19-3	H-19 to H-88 Mod Kit	$395.00	$175.00	$220.00
H-88-5	Cassette I/O	49.00	24.00	25.00
H-88-10	Wirewrap Card	30.00	15.00	15.00
Z-89-11	Syn/Asc/Cen Interface	99.95	39.95	60.00

SOFTWARE

Z-100 Software

Model No.	Product	List	Save	Now Only
CB-463-9	Condor*/Peachtree* Interface	$ 99.00	$ 50.00	$ 49.00
CB-463-11	Z-Chart* Graphic Package	150.00	100.00	50.00
CB-5063-16	MSDOS 2 Programmers Pack	149.00	75.00	74.00
CD-463-2	Condor FMS	299.00	150.00	149.00
LS-463-1	Lotus 1,2,3	495.00	200.00	295.00
MS-463-7	Microsoft Multiplan	250.00	100.00	150.00
OS-63-2	CPM 86	250.00	100.00	150.00
PO-463-1	IBM 3270 Emulator	650.00	200.00	450.00
RS-463-1	Peachtree G.L.	399.00	200.00	199.00
RS-463-2	Peachtree A.R.	399.00	200.00	199.00
RS-463-3	Peachtree A.P.	399.00	200.00	199.00
RS-463-5	Peachtree Inv.	499.00	250.00	249.00
RS-463-6	Peachtree S.I.	299.00	150.00	149.00
RS-463-75	Peachtree 5000*	395.00	150.00	245.00
TA-463-1	ZDS Teacher	299.00	100.00	199.00
TA-463-2	ZDS Student	100.00	40.00	60.00

*Multiplan is a trademark of Microsoft.
Lotus 1,2,3 is a trademark of Lotus Development Corp.
CPM is a trademark of Digital Research Inc.
Peachtree and Peachtext 5000 are trademarks of MSA, Inc.
Condor and Condor FMS are trademarks of Condor Computer Corp.
Z-Chart and Pecon are trademarks of Zenith Data Systems Corp.

**Not available through Mail Order.

Z-151/161 Software

Model No.	Product	List	Save	Now Only
BP-5063-1	BPI General Accounting	$595.00	$300.00	$295.00
BP-5063-2	BPI Accounts Receivable	595.00	300.00	295.00
BP-5063-3	BPI Accounts Payable	595.00	300.00	295.00
BP-5063-4	BPI Payroll	595.00	300.00	295.00
BP-5063-5	BPI Inventory	795.00	400.00	395.00
BP-5063-6	BPI Job Costing	795.00	400.00	395.00
BP-5063-8	BPI Personal Accounting	195.00	50.00	145.00
BP-5063-10	BPI Association Management	795.00	400.00	395.00
CB-5063-9	Pecon*	99.00	50.00	49.00
CB-5063-16	MSDOS 2 Programmers Pack	149.00	75.00	74.00
CD-5063-1	Condor DBMS	650.00	350.00	300.00
CD-5063-2	Condor FMS	299.00	150.00	149.00
LS-5063-1	Lotus 1,2,3**	495.00	200.00	295.00
MS-5063-12	Microsoft Sort	199 00	99.00	100.00
RS-5065-1	Peachtree G.L.	499.00	250.00	249.00
RS-5065-2	Peachtree A.R.	499.00	250.00	249.00
RS-5065-3	Peachtree A.P.	499.00	250.00	249.00
RS-5065-5	Peachtree Inventory	499.00	250.00	249.00
RS-5065-6	Peachtree Sales Invoicing	399.00	200.00	199.00
RS-463-75	Peachtext 5000	395.00	150.00	245.00

SPECIAL SALE PRICES GOOD THROUGH DECEMBER 15, 1984.

QUANTITIES ON SOME PRODUCTS ARE LIMITED. ORDER NOW. Special prices not valid in combination with other special offers.

CALL TOLL-FREE
1-800-253-7057

Order from 8AM to 4:30PM Eastern Time Mon.-Fri.
In Alaska, Hawaii and Michigan call
616-982-3411.

USE THE HANDY RETURN ORDER FORM IN YOUR HEATHKIT CATALOG
(Found in Mail Order Catalogs)

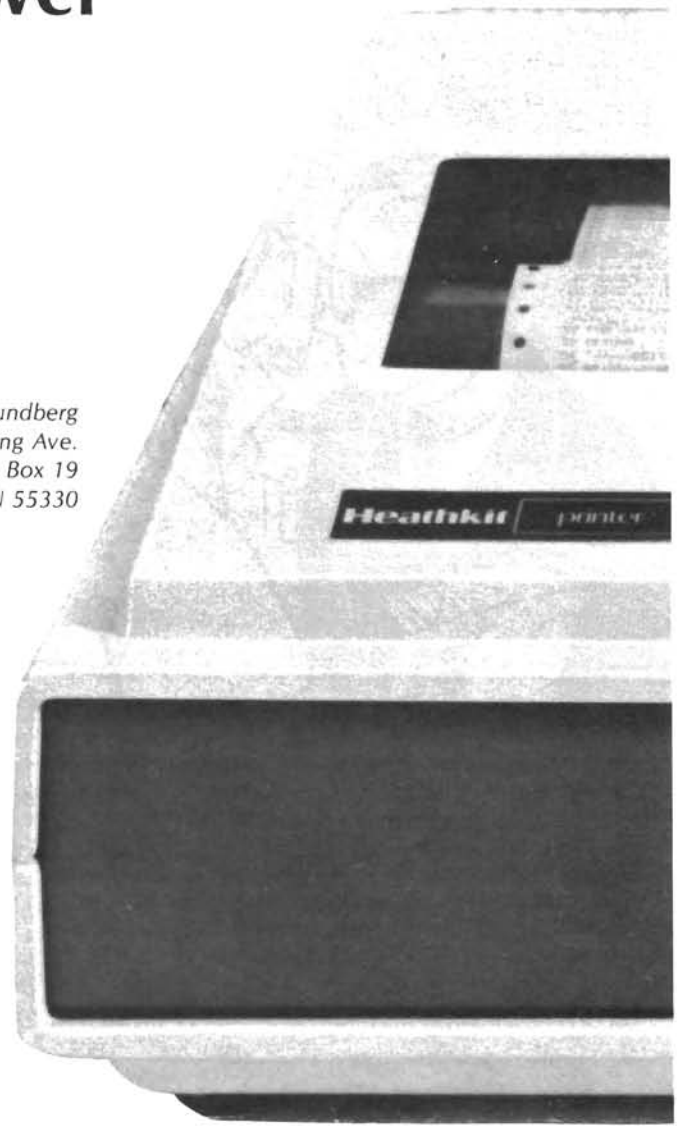
VISIT YOUR LOCAL HEATHKIT ELECTRONIC CENTER
(See your Retail Catalog for locations)

CP-233

Heath®/Zenith®
Computers and Electronics

Beware of Faulty Power Supply Components or Trouble With the H-25 Printer

Clifford C. Lundberg
310 King Ave.
P. O. Box 19
Elk River, MN 55330



One day last spring, I was happily romping playfully with my friends H-8, H-17, H-19 and H-25. H-25 is my newest friend (except for H.S. Modem). He moved into the neighborhood the previous fall. All these friends have different personalities. H-25 always had a quick word on any subject. This particular day, he was rattling off a string of BASIC, when suddenly he started mispronouncing letters of the alphabet. This became very disconcerting. I have difficulty enough understanding standard BASIC, let alone a foreign dialect of it. I tried to coax him to behave. I put him through his test routines. Nothing helped. "Perhaps he's just tired," I thought. I put him to bed for a nap.

That evening when he awoke, he did his warm-up exercises and performed wonderfully. I assumed his daytime antics were just a practical joke. We finished our chores in record time and all seemed well with the world.

Alas, my friend seemed to have developed a split personality. At night (after 10:00), he was the epitome of a gentleman. During the day, however, he was acting very strange. Every 17th vertical line was omitted. He just left it a blank space.

What to do? I am not too handy at neuro-circuitry. I took H-25 to the local specialist for a check-up. He acted the perfect gentleman. Back home, he started acting up again. I went to my power company to see if they were sending any strange frequencies to my house. No they were not, but they would put a monitor on my power lines to check the voltage. Behold, low was the voltage. H-25 was getting 118 volts at night, but only 112 volts during the day. A case of malnutrition. The power company would give me a new transformer so H-25 could have 120 volts.

But why did only H-25 suffer symptoms of low voltage and not my other friends? Was H-25 suffering from some immunizational deficiency? Out with my multi-tester to check his internal voltage and resistance. All the book tests were OK, but I did not have a frequency tester, and the regularity of the symptom seemed to indicate a cause related to a frequency deficiency. "It must be in the 9 volt power supply circuit on the motor drive circuit board," said I to myself. The circuit was only providing 7.6 volts. Fine tuning my resistance checker, unplugging connector P501 from the board, I found diode 504 solved the voltage deficiency and cured H-25.

Had my line voltage not been low, I may never have discovered the faulty diode. A line voltage of 118 or better apparently provides enough energy to allow the capacitors in the 9 volt supply circuit to store enough charge to keep the circuit energized during the second half of the alternating current cycle, when diode 504 would not permit sufficent energy to pass. Or perhaps the higher voltage was all that D504 needed to do its part. Anyway, all my friends are now communicating effectively, and I am getting a boost in house voltage as a result of these tribulations. One of these days, I will have to take a Heath course in electronic circuitry, so it won't take me six months to figure a diagnosis, let along a solution to such perplexing problems.

By the way, did you see the Heath H-88K memory board on the work bench of Andre Rubbia in the photo on page 42 of the January 1984 issue of Discover magazine? The picture shows Andre and his father, Carlo. Carlo Rubbia was named Scientist of the Year by Discover for his work in discovering the W and Z particles, which helped to pave the way for that elusive 'Unified Field' theory, which eluded Einstein. Discover is a monthly science news magazine published by Time, Inc. and a very good source for general science news.

YOU'RE IN GOOD COMPANY

WITH AN H-1000

Amdahl, Nestles, Bell Labs, Jet Propulsion Laboratories, the U.S. Army and Navy, Allied, Bendix, CDR – these are just a few of the many satisfied H-1000 users. They know that the H-1000 gives them performance and compatibility unmatched by the giants. And many distinguished hobbyists have discovered that the H-1000 is available at a very attractive price – typically half the cost of a new IBM-compatible. Just look at the benefits:

- emulates IBM PC with monochrome board; runs most PC software at 4 times the speed
- 100% H89 compatible; runs your programs twice as fast with NO changes
- on-board memory from 128K to a full megabyte; holds twice the data and programs of any comparable PC!
- built-in Ghost disk for all operating systems; adds an additional RAM disk for instantaneous disk accesses.
- from $995 to upgrade your H89, or $1995 for a complete computer.

A STATE-OF-THE-ART UPGRADE

TMSI has taken the best features of Heath's 8-bit H89 and combined them with the best of the 16-bit PC's to give you the best of both worlds . . . and the ONLY package that builds on your current investment.

The 8-bit side gives you access to the world of low-cost 8-bit software for the CP/M and HDOS operating systems. The H-1000 runs all Heath/Zenith software and hardware for the H89 family without change. And with its 4MHz Z80, expanded memory, and ghost disk, your existing programs perform like never before.

The 16-bit side of the H-1000 features an 8086 running at over 8MHz, offering you the fastest screen response and program execution of any PC compatible on the market today. The standard keyboard and screen emulate an IBM PC with monochrome display; accessory color and graphics boards can extend the emulation even further. The "Ghost disk" is an added bonus; entire disks can be loaded into memory, freeing your "real" drives for another disk. You get instantaneous access to your data that cuts out all that waiting while your computer talks to your disk drive.

FULL SERVICE BACKUP DIRECT FROM THE FACTORY

The H-1000 comes fully assembled and tested, with Heath's "we won't let you fail" quality manuals. A full 90-day warranty covers each unit, with an optional service contract available. And we work closely with our customers for special applications assistance. Get the benefits of high-performance computing today! Call or write:

TMSI, Dept. RN
366 Cloverdale
Ann Arbor, MI 48105
(313) 994-0784

TMSI

Versatile Expansion Interface For The H-89 From Microflash

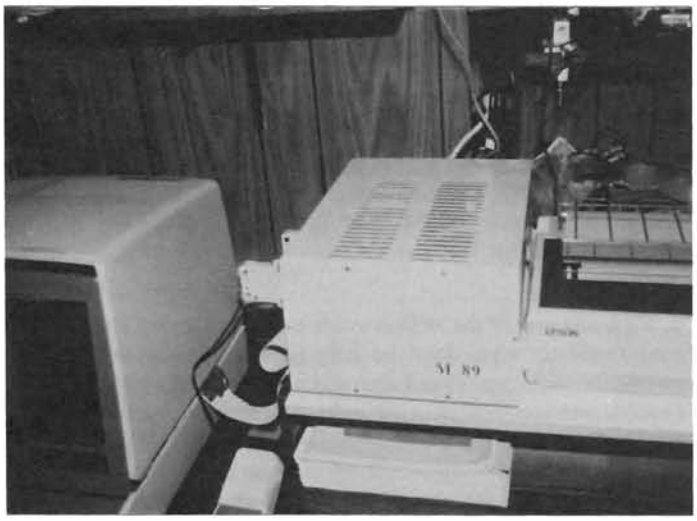
Peter Ruber
P.O. Box 502
Oakdale, N. Y. 11769

Last fall, when I installed Heath's Z-37 double-density Controller as a companion to my H-88-3 3-Port Serial and H-88-1 hard-sector Controller cards, I realized I had exhausted the expansion capabilities of my H-89. This problem became critical when, in a moment of greedy acquisition, I ordered a Z-89-11 Serial/Parallel card from Heath and an H-89-3 Color board from New Orleans General Data Systems.

After several weeks of switching cards, losing wire harnesses under the CRT, bending pins, and cracking the top cover hinges, I feared that sooner or later I would damage an expensive board and lose weeks waiting for it to be repaired.

Thus, the time had arrived for me to investigate the item Heath had forgotten -- an expansion device that would allow me to access my present and future boards without playing musical chairs.

According to the ads, there were three expansion models available offering varying degrees of flexibility. Mako Data Products offered a reasonably priced board that plugged into the three right-hand sockets of the CPU Logic board and provided the user with a total of six expansion slots. On the higher cost spectrum, Kres Engineering offered a novel item -- a full-size board that masks the CPU Logic board, and has seven expansion slots (three vertical and four horizontal) and includes 16k of additional memory to replace the 16k memory board whose space it occupies.



The final alternative was a gray box from Microflash Co. which resembled a large Swiss Cheese, but provided nine additional expansion slots, plus a lot of other goodies.

When I contacted each manufacturer for specific product information, only Microflash responded promptly. Mako ultimately took three months to send me a flyer, and Kres totally ignored my request. Based on past experiences with other suppliers, I decided that the company who replied the fastest would probably be equally responsive to service problems. Although the Mako and Kres units are fine products, I preferred having an expansion device removed from the computer. I live in an area where line voltage drops during hot weather occur frequently, and I didn't wish to have my terminal screen blank out during an inspired typing session due to a strain on my power supply.

A separate unit would also eliminate the jungle of wire inside my H-89, a particularly bothersome aspect due to the design of the system.

The Microflash unit was available in both kit and assembled versions. They offered a 30-day warranty on all kit parts and a 90-day warranty for the A&T unit. They further offered a 30-day money back deal if you were dissatisfied with either piece. Being daring and foolhardy by nature, I chose the kit.

As soon as the M-89 Expansion Box arrived, I got the feeling that the folks at Microflash took pride in their product. All parts were carefully packaged in separately marked bags. IC's were in protective foam and sockets were provided for all chips. All parts were prime grade (no unmarked or dubious parts one occasionally receives from electronics houses). There were heat sinks for the power transistors and voltage regulators, and heavy duty switches and sockets. The interface and motherboards were industrial-grade epoxy glass, completely solder-masked, with plated-through holes and clear silkscreen legends to identify all parts and their proper orientation. The case was ruggedly constructed with heavy gauge metal, and I suspected it would sustain little visual damage if I dropped it from my roof.

The construction proved to be quite straight-forward and I encountered no difficulties or problems. I all but ignored the assembly manual once I was on a soldering roll.

The first order of business was to mount the power supply, AC line filter, ETRI fan and guard, the fuse holder, switches, sockets and

receptacles, and wire them according to a clear drawing in the assembly manual. The power transformer is a large 8V/14V Sunny unit that sounds a comforting "thud" when you first turn on the power. Four additional cutouts are provided on the back of the case for two 12-pin Molex receptacles and two AC receptacles, so that external devices can be connected and accessed through computer control. More on this later.

Assembling the motherboard is somewhat laborious, requiring well over 600 solder connections. Two 25-pin Molex connector strips are provided for each of the 9 expansion slots.

The lower 25-pin strip and the bottom 10 pins of the upper 25-pin connector duplicate P505 on the CPU Logic board. All lines are tied together from slot #1 to slot #9, with the exception of the S0, S1, LP and CASS lines (the latter is not used, however), which must be enabled through an Address Jumping Matrix on the right side of the motherboard in order to avoid I/O conflicts.

The Address Matrix provides a method for decoding an additional 8 I/O lines in conjunction with Microflash's H89-1 Decoder Interface ($29.95* Kit, $69.95* A&T) -- a nifty board about two inches square that plugs into U550 on the CPU Logic board and allows you to access ports 0Q to 77Q. A 16-wire ribbon cable with DIP headers at each end connect the small Decoder Interface board with the M-89 Interface board (which takes the place of your Serial board (P505), so that you now have a total of 16 I/O lines to work with. The Decoder Interface board also contains a provision to further decode 8 more I/O lines (for a total of 24 I/O lines). Microflash does not at present provide support for the third set of I/O lines.

The 15 unused pins of the upper 25-pin Molex strip permit the user to bring the HALT, NMI and whatever clock, data, address lines are required to control external devices without infringing or conflicting with the microprocessor lines on any of the expansion slots.

Briefly, the concept works like this: The two AC sockets provide power to the external device. Voltage is fed to any of the four relay sockets where the user can install 125V double-pole double-throw relays that can turn "on" or turn "off" the connected device. Two schematics are provided to illustrate the necessary TTL buffering between the microprocessor lines and the relay switches. A 10-pin socket is provided for a wire-wrap board containing the buffer. With the proper user-written software, you can control such devices as automatic factory equipment, air conditioning and heating units, burglar alarms, laboratory equipment, light controls or whatever the creatively devious mind is capable of devising.

Since there are some potential hazards involved when you bring high AC voltages to any computer board, Microflash recommends that you avail yourself of their free consultation service to discuss your particular application before it is implemented. The average hobbyist, however, can make use of these sockets by connecting the power lines to the line-cord receptacle and then plug in his printer, external disk drives or modem.

The final portion of the assembly consists of crimping 50-pin headers to the interconnecting ribbon cable and soldering some sockets, chips, diodes, resistors, capacitors and connectors to the interface board. Total assembly time is approximately six to eight hours depending on the number of beers you wolf down.

Before you secure the motherboard to the bottom of the case, attach ten spacers to the board, lay it inside the case and connect the Molex power socket. Then turn on the juice and hold your breath. Voltage readings are made at several test points. If they fall within specified limits, you may turn off the power, configure the I/O ports via wire jumpers from the connector strips to the Address Matrix and plug in your boards.

Although my voltage readings were correct, I confess I did have a problem. My computer locked up during the boot procedure and I quickly relayed my tale of woe to Mr. Kan Cheng of Microflash. I received a reply about a week later suggesting additional test procedures. These didn't work either. I called Mr. Cheng one night and we discussed the problem at great length while I probed with my trusty Volt-Ohmmeter. As an escape from my frustration, I sent the interface card to Mr. Cheng.

He returned it advising me it worked fine in his H-89. I took drastic measures. I removed the motherboard and examined all connections with a bright light and a magnifying glass. Within minutes I had located a very thin bridge of solder between two pins on the 50-pin connector that had somehow crept under the solder mask and had defied previous detection. As soon as I cut the solder bridge the M-89 sprang to life and has been humming along for nearly four months without further difficulties.

The problem I had (due solely to my own carelessness) point up the only weak aspect of the M-89 package. The documentation does not live up to the quality of the hardware itself. For something as potentially complex, 29 pages of documentation (consisting of about 3 pages of theory and assembly instruction and 26 pages of diagrams, schematics, timing charts and parts lists) barely scratch the surface and many questions go unanswered. I wish that Mr. Kan Cheng had acquired some assistance in preparing the text to smooth out the grammatical faults and explain in greater detail some of the additional uses of his product. Clearly the documentation favors the knowledgeable computerist and some novices might shy away. But if you study the schematics and parts placement diagrams carefully and work in a neat and orderly fashion, you can easily avoid the problem I had. To aid other users, I have gleaned the following from my letters and telephone conversations with Mr. Cheng as a way of trouble-shooting the M-89 if you can't get it to work from the start.

Not to get too repetitious, remember to turn off all power and unplug line cords at any point where inserting or removing any board or cable is required.

1. Double check all wire placement and soldering connections in the power supply area.

2. Check for solder bridges and poor solder connections on both the interface and motherboards. Reheat any dubious connections.

3. Perform the following tests before mating the M-89 to the H-89.

4. Plug the interface card into P505. Make certain switches 1,2,3 (S0,S1,LP) are in the off position. Boot a test disk. If it boots, remove disk and turn off power. Remove interface board and slide switches 1,2,3 to the "on" position. Boot again and, if successful, go to the next step. If the boot procedure fails, check for a solder bridge on the 50-pin connector; check that the diodes are oriented correctly; check that the 2N3904 transistor is oriented as it should be. Substitute either or both the 74LS244 Octal Tri-State Driver and the 74LS245 Octal Bus Transceiver chips, or return them to Microflash for testing.

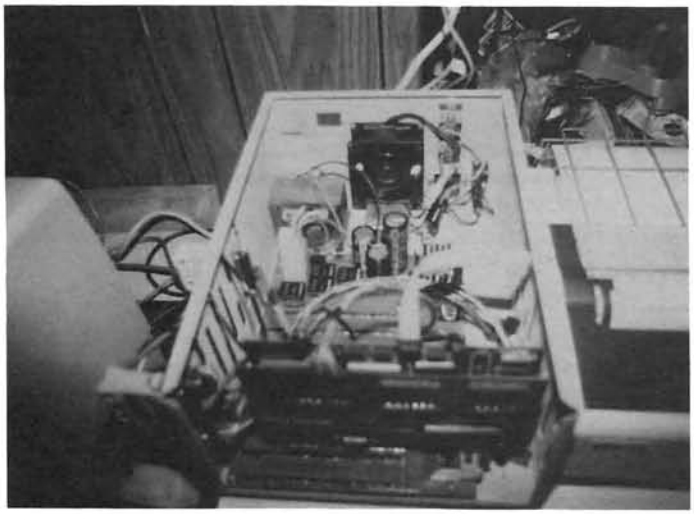
5. Plug one end of the ribbon cable to the Interface card. Turn on power and boot. If you do so, go to the next step. However, if the boot procedure locks up, check the connectors on the ribbon cable. Possibly one end is off its mark and two or more wires are creating a loop in the circuit. Use a Volt-Ohmmeter to verify or simply remove the connectors, slide them half an inch up the cable, crimp them again and trim off the excess.

6. With the computer turned off, plug the other end of the ribbon cable into the motherboard on the expansion buss. If you can't boot, you have a poor solder joint or bridge on either the 50-pin connector or on one or more of the 25-pin Molex strips creating a loop.

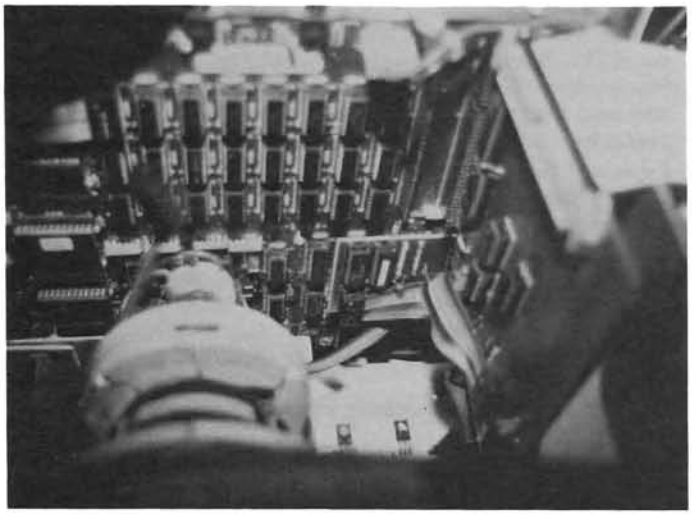
The M-89 Expansion Box measures approx. 10" wide, 6-1/2" high and 14-1/4" deep. Power output is 5V/5A, 12V/1.5A, -12V/1.5A. The box is well-vented on the top, with additional venting provided on each side by the twenty DB-25 connector slots and the eight 50-pin connector slots. The ETRI fan is so powerful and quiet that even after running the box all day every board was cool to the touch. I installed one like it inside the H-89 and have practically eliminated most of the previous heat problems.

Once you start installing board and cables you will really appreciate all the cutouts. The New Orleans General Data Systems Color Board alone requires openings for seven connectors (four joysticks, CTC cable, and video and audio cables). There is also the added benefit of being able to access any board for easy troubleshooting.

Under normal use, your Serial card installs in Slot #2. Since I use mine primarily for an MX-80 printer and a Hayes Smartmodem, I removed IC U603 (port 320Q) and jumpered the addressing matrix accordingly. My Color board is in Slot #3 and is jumpered for port 320Q. My Serial/Parallel card has the Serial chips removed and shares Slot #3 occasionally with the Color board. All peripheral equipment connects to the Expansion Box. I left the cables that originally connected my Serial card to the DB-25 connectors in back of the computer in the event I had to have periodic mobility, and also to provide me with an installed cable for connection to P613 on the CPU Logic board, so that I can attach my soon to be acquired H-29 terminal at a moment's notice.



1. The neat and spacious layout of the M89 Expansion Box provides for easy servicing. The entire unit can be dismantled and the mother board removed in less than five minutes.



2. Inside the H89, Microflash's M89-1 Decoder Interface Card plugs directly into the U550 socket. The U550 chip, in turn, plugs into the Decoder Interface Card. A 16-pin DIP cable bridges the M89-1 to the Expansion Box's Interface board now installed at P505. This gives you 8 additional I/O ports to work with (0Q- 77Q).

It is suggested by Microflash that Slot #1 be left unoccupied in the event someone wishes to design a 16-bit co-processor card at a reasonable cost. I hope someone out there takes the hint.

When installed, the Interface card takes total control of the Serial ports. If you wish to keep your Serial card in its present position, you may install the Interface card at P506 in place of your hard-sector controller. In this event, you must turn off the S0, S1 and LP switches so that the two boards don't fight for control of the Serial ports.

Despite the inadequate documentation, the average computer hobbyist can obtain easy and versatile expansion for his computer. An electrical engineer could have a field day with this device. Perhaps Mr. Cheng can be persuaded someday to issue a supplement that will document his product and fully explain some of the potential uses of the M-89. Perhaps some other users of the M-89 will be inclined to share their own experiences.

The price of the M-89 Expansion Box is $245.00* in the kit version, and $495.00* A&T. For additional information, contact Mr. Kan Cheng, Microflash Co., 8432 N. Laramie, Skokie, Illinois 60077. Tel. # (213) 677-4928. Microflash also produces a variety of boards that will enhance the usefulness of your H-89.

* Prices are at time of writing.

Support and More

Energize your tired old computer with...

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 the state-of-the-art technology of the Z80, the DG SUPER 89 offers the user increased speed and system reliability for years to come. Super 89 BIOS included. Full compatibility with all Health/Zenith software and hardware products is designed into the DG SUPER 89. 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. Super 89 BIOS 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).

SUPER 89 BIOS

The Super 89 BIOS is a greatly expanded operating system interface fully compatible with that provided with H/Z CP/M for use with the Super 89 and Heartbeat. Support is provided for all H/Z disk drive systems as well as Magnolia Microsystems 8"/5" disk controller, CDR systems FDC-880H, and hard disk systems using the popular XEBEC Winchester controller.

Expanded facilities of the BIOS include a built-in RAM Disk with capacity up to 170 Kbytes; a 48 Kbyte RAM print spooler; transparent operation at CPU clock rates of 2 or 4 MHz; increased TPA (Transient Program Area) for larger user programs; detailed error messages and user selection of recovery options; real-time clock support; simplified, self-prompting, configuration utility that is truly user-friendly; also, included are several useful utilities for expanded system operation.

The Super 89 BIOS is available separately, $60.

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.

Vectored from 7

```
100 FORMAT ( A1, 'The following variable was initialized:',
   1 /, A1, ' VALUE = ', F6.1 )
      :
      :
    END
```

In the example, a variable LF is declared to be one-byte long, and is initialized to be decimal 10 (the ASCII code for line- feed). Then, in every WRITE to a disk file (assumed to be unit 8 in the example), LF is output in an A1 field as the first character in the line.

This is awkward programming, but it works. However, while this will create a normal CP/M text file on output, it will also create a problem in F80 should you ever want to read the file back in. Since F80 assumes that CR is the end-of-line character, it will stop reading at each CR and will thus read the LF as the first character of the following line. You will need to read the file in by skipping the first character of each line with a 1X field in the input FORMAT.

To fix the problem permanently, and in a manner consistent with most other Fortran implementations, you are far better off if you patch the disk driver subroutine in the standard library (FORLIB.REL). A properly patched driver will always output CR/LF at the end of each line, and will ignore the LF on input. Thus, all CP/M text files would be directly readable by F80. One such patch was described in Microsystems (March, 1984) although my experience is that this patch has some problems of its own. However, that article is an excellent place to begin. Fortunately, the source for the disk driver is included in the standard distribution from Heath/Zenith.

If there is enough interest in making patches or fixes to the FORTRAN-80 library, or in interfacing assembly routines to F80, or in creating other general purpose libraries for F80, maybe we could get a short series started in REMark. I already have submitted an article on patching the library to fix a FORMAT error and an INTEGER*4 math error. (Appeared in September 1984, Volume 5, Issue 9.)

Bill Brandoni
37926 Wright Street
Willoughby, OH 44094

Setting The Record Straight

Dear Nancy,

The description of *Checkoff* (P/N 885-8010 and 885-8011) in the *HUG Software Catalog* is innaccurate in several areas now that HUG is offering version 3.0. To set the record straight, please allow me to briefly clarify a few key points and highlight the improvements (*) over version 2.1.

The following changes to *Checkoff* are for both HDOS and CP/M:

1. Only one drive is required to run *Checkoff*.
2. Any 12 or 24 line terminal may be used.
3. The file date is maintained (not Date of Entry).
*4. The arithmetic precision is improved.
*5. The running balance is displayed.
*6. The file error handling is improved.
*7. The data input routines provide greater protection.
*8. A utility is included to convert 2.1 files to the new 3.0 format.
*9. The program will not "creep" into the operating system.

Changes to CP/M *Checkoff* are the following:

*1. *Checkoff* automatically sizes itself to the host system.

*2. Printer support is included in File Scan and Summary modes.

Thank you for helping me to get this information out to HUG members.

Yours very truly,

Jim Meyers

Editor's Note: *This update was announced in the November 1983 issue of REMark. All products 885-8010, 885-8011-[37] are version 3.0 as of that date.*

Saran Wrap Keyboard

Dear HUG:

Here is tip for all of you who live in a dusty environment and would like to keep your keyboard neat and clean always, without having to use a bulky dust cover.

I use a sheet of Saran Wrap layed over the keyboard, (loosely) tacked in place with a bit of scotch tape here and there.

You can use the keyboard for about 3 or 4 months before the wrap gets brittle and cracks, then can be replaced. The keyboard stays neat and clean, always looking new.

After a year and a half of use on my H89, the magnetic properties of the Saran Wrap have had NO adverse effects on either my H89 nor the disks.

Carl Saluzzi
13 McCammon Rd.
Tonasket, WA 98855

RTTY - Or Amtor With The Z-100?

Dear HUG,

Does anyone have a program to take RTTY or AMTOR on the Z-100? I have a German Amateur Radio Station and a fine program working on my TRS80. But, now I have a Z-100 and no more RTTY?

Thank you for your help.

Sincerely,

Jurgen Meyer
Roggenkaump 10
D0284r Holdorf
West Germany

No Muss Nor Fuss

Dear H.U.G.,

I have owned an H-89 for three years, and have been pleased with the overall performance of the product. A built-in glitch has, however, been the cause of some irritation. The disk drive protocol, used by Heath, causes problems when configuring the hardware from hard/soft sector operation to soft/soft operation, and back again. Switching drive dip jumpers, and cables was a constant bother.

After a few of these switching exercises, I decided that something had to be done. As I looked at the schematics, I realized how simple the

task would be. All that was needed was a simple two wire swap. Then the Heath hard sector controller would handle drive select information in the same standard format as the soft sector controller. The following information explains the problem, and suggests two possible solutions. I personally selected the resistor modification, as it is the easiest to accomplish, and may be demodified easily, if necessary.

I freely give this information to H.U.G. for use in H.U.G. releases, as deemed appropriate by the Heath/Zenith Users' Group.

H-88-1 Hard Sector Controller Modification

Those of us who have converted our H-89s to Soft sector, and two or more drives, but still must occasionally use the H-88 Hard Sectored board have long had to deal with an inconvenience caused by the drive select output on the H-88-1 hard sector controller board.

The current industry standard disk drive to disk controller select coding is as follows:

Drive A => DS1 (drive select 1)
Drive B => DS2 (drive select 2)
Drive C => DS3 (drive select 3)
..... etc.

The Heath hard sector board isn't configured that way, however, but is configured in reverse:

Drive A => DS3
Drive B => DS2
Drive C => DS1

This configuration difference causes a lot of extra work when switching to dual soft/hard sector board operation. I fought it for a while, but finally decided to correct the problem at its source, the H-88 hard sector controller board! The fix is very easy, and eliminates the disk drive DIP jumper changes that are normally required to reconfigure the drive select pins.

*** Modification ***

REF: Schematic Diagram (Part 2 of 3)
Floppy Disk Interface

The schematic shows the three outputs of concern as:

SY 0 L => P803 Pin 12 - from L803 (DS2 to Drive A)
SY 1 L => P803 Pin 12 - from L803 (DS2 to Drive B)
Drive 1 L => P803 Pin 10 - from L802 (DS1 to Drive C)

As can be seen on the schematic, the drive select pins are isolated, and do not interact with any other circuits on the board. The fix is, therefore, very straight forward.

There are two simple methods which will fix the problem.

1. Physically interchange connector pins P803-14 and P803-10 at the connector plug or wiring.

or

2. Perform the following board modification.

Unsolder the L804 connection closest to the connector and pry the lead up to clear the circuit board. Do the same to L802. Locate the unsoldered coil ends to prevent any shorts, and solder a small insulated wire to each lead. Solder the L804 wire to the hole going to pin 10, and the L802 wire to the hole going to pin 14. Interchanging the connections in this way converts from Heath HS mode to the

mode used by the rest of the world. Now the system can be permanently configured to the following:

Internal drive = DS1
External drive = DS2

On my system, the internal drive is connected to the bottom connector of the H-37 controller. The external drive is connected to the H-88 controller or to the top connector of the H-37 controller. I have extended both connectors to the rear of my system and marked them accordingly, so that I may switch the external drive from hard to soft sector operation by simply moving the external plug.

No more muss nor fuss. Just plug the external drive into the appropriate connector, use the right BIOS, and away you go! On dual controller operation, the internal drive is drive A, B, C and the external drive is drive D, E, F.

Rocco Silvestri
3629 Rusty Rim
Ellicott City, MD 21043

That's All For The Patch!

The following patch script will correct a bug in Zenith MS-DOS 2.13, which causes characters to be dropped on serial I/O routines. The author makes no claims other than that the documented patch below works fine for him. Without the patch, any serial I/O on ZDOS fails because of the bug which exists in the interrupt handler.

The patch, as shown, assumes that the working disk to which the patch is being applied is a 5-1/4 double-density, double-sided floppy disk formatted with 8 sectors per track. This is the default in Z/MSDOS 2.13.

Enough about the disclaimer! Here is the script:

1. Format a bootable floppy disk. Command: format a:/s

2. Copy DEBUG.COM, ASSIGN.COM and SYS.COM to the disk just formatted.

3. Place the disk just created in drive A and boot it.

4. Bring up DEBUG. Command: debug

5. Load in IO.SYS. It should be located at block 10 (0AH) on the disk. Command: L 1000:0 0 A 20

6. Type in the following commands. Be sure to check for the indicated response. The stuff following a semicolon (;) are comments and are not typed in. An underline symbol (—) indicates where the cursor will be when appropriate. Arrows surrounding "CR" mean to just hit return. (<CR>)

```
Command:  U 1000:1E32 1E33     ;this dis-assembles a code fragment
Response: 1000:1E32 MOV AH,AL  ;this is the instruction we change
Command:  A 1000:1E32          ;tell DEBUG we want to assemble
Response: 1000:1E32 =          ;DEBUG waits for an instruction
Command:  MOV BH,AL            ;this is the new instruction
Response: 1000:1E34 =          ;just hit return here
Command:  <CR>
Command:  U 1000:1E47 1E49     ;dis-assemble another fragment
Response: 1000:1E47 POP BX
          1000:1E48 MOV AL,AH  ;these are the two we want to change
Command:  A 1000:1E47          ;tell DEBUG we want to assemble
Response: 1000:1E47 =          ;DEBUG waits for an assembly inst.
Command:  MOV AL,BH            ;this is the first new instruction.
Response: 1000:1E49
Command:  POP BX               ;this is the second new instruction.
Response: 1000:1E4A =          ;just hit return here
Command:  <CR>
```

```
Command:  U 1000:1E91 1E92     ;this dis-assembles a code fragment
Response: 1000:1E91 MOV AH,AL  ;this is the instruction we change
Command:  A 1000:1E91          ;tell DEBUG we want to assemble
Response: 1000:1E91 =          ;DEBUG waits for an instruction
Command:  MOV BH,AL            ;this is the new instruction
Response: 1000:1E93 =          ;just hit return here
Command:  <CR>
Command:  U 1000:1EA6 1EA8     ;dis-assemble another fragment
Response: 1000:1EA6 POP BX
          1000:1EA7 MOV AL,AH  ;these are the two we want to change
Command:  A 1000:1EA6          ;tell DEBUG we want to assemble
Response: 1000:1EA6 =          ;DEBUG waits for an assembly inst.
Command:  MOV AL,BH            ;this is the first new instruction
Response: 1000:1EA8 =
Command:  POP BX               ;this is the second new instruction
Response: 1000:1EA9 =          ;just hit return here
Command:  <CR>
                               ;This completes the patch.  Now
                               ;write out the BIOS.
Command: W 1000:0 0 A 20
```

That's all for the patch!! If you've done everything right, you can quit the debugger and re-boot from the floppy you just created. Try a program that was failing before the patch. If it works ok, then you can use this disk to format other system disks. You can also perform a "SYS E:" to fix up your winchester, if you have one, without having to re-format it. Enjoy.

Rick Schaeffer
E. 13611 26th Ave.
Spokane, WA 99216

Surely Someone Has Solved This Problem

Dear HUG:

I use an H-120 while I am in the office, where I practice civil engineering. When I am on out-of-town trips, I use a Radio Shack Model 100 lap computer, which I find to be very useful for writing letters and reports during time that would otherwise be wasted (for instance, while sitting on airplanes or waiting in terminals).

Usually, I transfer these letters and reports over to my office computer for final editing and printing out (I use Magic Wand or Peachtext 5000).

I have no trouble in transferring ASCII files from the Model 100 to the H-120 under CPM-86. My problem is that I usually use MS-DOS (Version 2) for word processing, and I can not get the Model 100 files transferred directly into MS-DOS. The way I do it now is to transfer under CP/M, then re-boot and use the RDCPM utility to convert the file to MS-DOS. It works under CP/M by attaching the serial cable from the Model 100 to the printer serial port (J1) on the H-120, and then simply using PIP as follows:

```
A:PIP filename=AXI
```

This seems to work, so long as baud rates are the same on the two computers, no matter what I see as stop bits, parity, etc.

But under MS-DOS, nothing seems to work, even though I have faithfully followed suggestions from several people (including the technical staff at Zenith Software Consultation). I get nothing at all when I hook up to the modem serial port (J2). When I hook up to the printer port, as under CP/M, the best I can get transferred is a file of about 80 characters, and everything past that is lost.

Surely someone out there has solved this problem. I would really appreciate hearing from anyone who has, and who will drop me a line.

Sincerely,

Frank T. Wheby
1604 Chicago Avenue
Evanston, IL 60201

Driving A Cadillac, But Running Gas-a-hol

Dear HUG,

Is there anyone out there making a hardware input to the Z-100 so it can run IBM-PC software? If so, please work fast, I feel like I'm driving a cadillac that will only run on "Gas-a-hol."

Bill Wirstrom
9429 Citrus Lane
River Ridge, LA 70123

Ron Hackney appointed Product Line Manager for ZDS ....

Ron Hackney has assumed the responsibilities of Product Line Manager for Zenith Data Systems replacing Barry Watzman. Ron will be in charge of computers and operating systems. In a recent visit to the HUG offices, Mr. Hackney requested that users with product suggestions or requests should drop him a line at the following address:

Attention: Ron Hackney, Product Line Manager

Zenith Data Systems
Hilltop Road
Saint Joseph, MI 49085

Ron comes to his new position with a great deal of experience in consultation at both the OEM and user level, and we wish him the best for the future.

Vectored from 9

```
        MOV   H,A
FREE8   DCR   B
        JMP   FREE7
FREE9   SHLD  FREEK   ; Store free space in "K"s
        POP   H       ; Clear stack
        LDA   DEFDRV  ; Log original disk back in
        MOV   E,A
        MVI   C,SELDSK
        CALL  BDOS
        RET
```

Dear Editor,

The following one liner to provide automatic centering of a string on a page is probably too simple to make much of, but I find I automatically add it to the preliminaries of every program.

```
100 PRINT TAB(40-LEN(L$)/2)L$:RETURN
```

Or, if the Line Printer is File #1, and set for 96 characters per line

```
110 PRINT #1,TAB(48-LEN(L$)/2L$:RETURN
```

Then, anywhere in the program, to print a string centered on the page of display.

```
XXX L$="ASCII string to be centered on page"
      :GOSUB 100 (or 110)
```

For a standard heading of two or three lines, such as a name and address,

```
100 L$="CALOOSAHATCHEE MARCHING AND":GOSUB 140
110 L$="CHOWDER SOCIETY":GOSUB 140
130 PRINT #1,:L$="Cape Coral, Fl  33904
140 PRINT #1,TAB(48-LEN(L$)/2):RETURN
```

To print the standard heading, then call GOSUB 100. The first two lines are centered by calling a sub-sub routine. The third line "falls into" the center-printer routine, so that the terminating RETURN then returns to the main program.

To center-print any other string, set L$ to that string and GOSUB 140, thereby skipping the standard heading.

Winslow Palmer
114 Montrose Drive
Fort Myers, FL 33907

Dear HUG,

Thank you for printing my subroutine in your "My Favorite Subroutines" column. However, there is one error in the magazine listing that is very important to the working of the program.

Line 140 reads:

```
140 IF ASC(TP$) = 97:REM if it is a capital no need to
    convert it.
```

It should read:

```
140 IF asc(tp$)>96 THEN 160:REM if it is lower case, no
    need to convert it.!
```

I apologize for this error, since part of it was my fault.

Sincerely,

Matt Jones
117 Freeman Drive
Warner Robins, GA 31093

Index of Advertisers

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

HUG ID Number: _____

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.

Heath / Zenith Users' Group

Hilltop Road
Saint Joseph, Michigan 49085

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

Volume 5, Issue 11

POSTMASTER: If undeliverable, please do not return.

P/N 885-2058