$2.50

REMark®

Issue 46 • November 1983

Official magazine for users of HEATH ZENITH computer equipment.

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

PRICE BREAKTHROUGH

Super Sale on New Disk Drives

Starting at **$169.00**

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

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

The Newest Technology Major Brands Capable Of Operating *On:*
RADIO SHACK[1] — HEATH/ZENITH[2] — APPLE[3]/FRANKLIN[4]
IBM/PC[5]-TEXAS INSTRUMENTS[6] & MOST OTHER COMPUTERS

Drive a Hard Bargain!!™ — 5 M.B.-12 M.B. Hard Drive, Everything You Need
Complete Systems . . . . . from $999.95

Diskette Breakthrough — 10 Pack in Library Case — ~~$18.95~~ NEW LOW PRICE

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

TOLL FREE ORDERING	GENERAL AND TECHNICAL
1-800-343-8841	**1-617-872-9090**

Model I/III/IV Drives (0 1 2 3)	starting at **$169.00**
Color Computer Drive (0 1 2 3)	**$ Call Toll Free**
Apple/Franklin Compatible Add-On Drives with Case & Cables	$219.95
Apple/Franklin Compatible Drives Complete with Controller — Now 40 Track	**$259.95**
Model I/III/IV Memory Upgrade	**$ Call Toll Free**
Printers — Daisywheel/Dot Matrix	**$ Call Toll Free**
Diskettes in Library Cases	~~$18.95~~ New Low Price
Cases and Power Supplies — (Single-Dual-1/2 Height)	starting at $44.95
Printer **Buffers** 8K to 512K	starting at $143.95
Percom Double Density Controller (Model I)	**$ Call Toll Free**
Holmes Model I/III Speed-up Mod-VID/80	starting at $90.00
Gold Fingered **Edge Card Extenders**	starting at $13.00
Cables — Printer/Disk Drive	starting at $23.00
DOSPLUS	**$ Special Prices**
Repair Services Now Offered — Fast Turn-a-Round	**$ Call Toll Free**

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

SOFTWARE SUPPORT, INC.

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

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

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

Dealer inquiries invited.

1 ™ TANDY CORPORATION
2 ™ ZENITH DATA SYSTEMS
3 ™ APPLE COMPUTER CORP.
4 ™ FRANKLIN COMPUTER, INC.
5 ™ IBM CORPORATION
6 ™ TEXAS INSTRUMENTS

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

Heath/Zenith Users' Group

HUG Manager ..... Bob Ellerton
Software Engineer ..... Pat Swayne
HUG Bulletin Board
and Software Developer .... Terry Jensen
Software Coordinator ..... Nancy Strunk
HUG Secretary ..... Margaret Bacon

REMark Editor ..... Walt Gillespie
Assistant Editor ..... Donna Melland

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

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

	U.S. Domestic	Canada & Mexico	International
Initial	$18	$20*	$28*
Renewal	$15	$17*	$22*

*U.S. Funds.

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

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

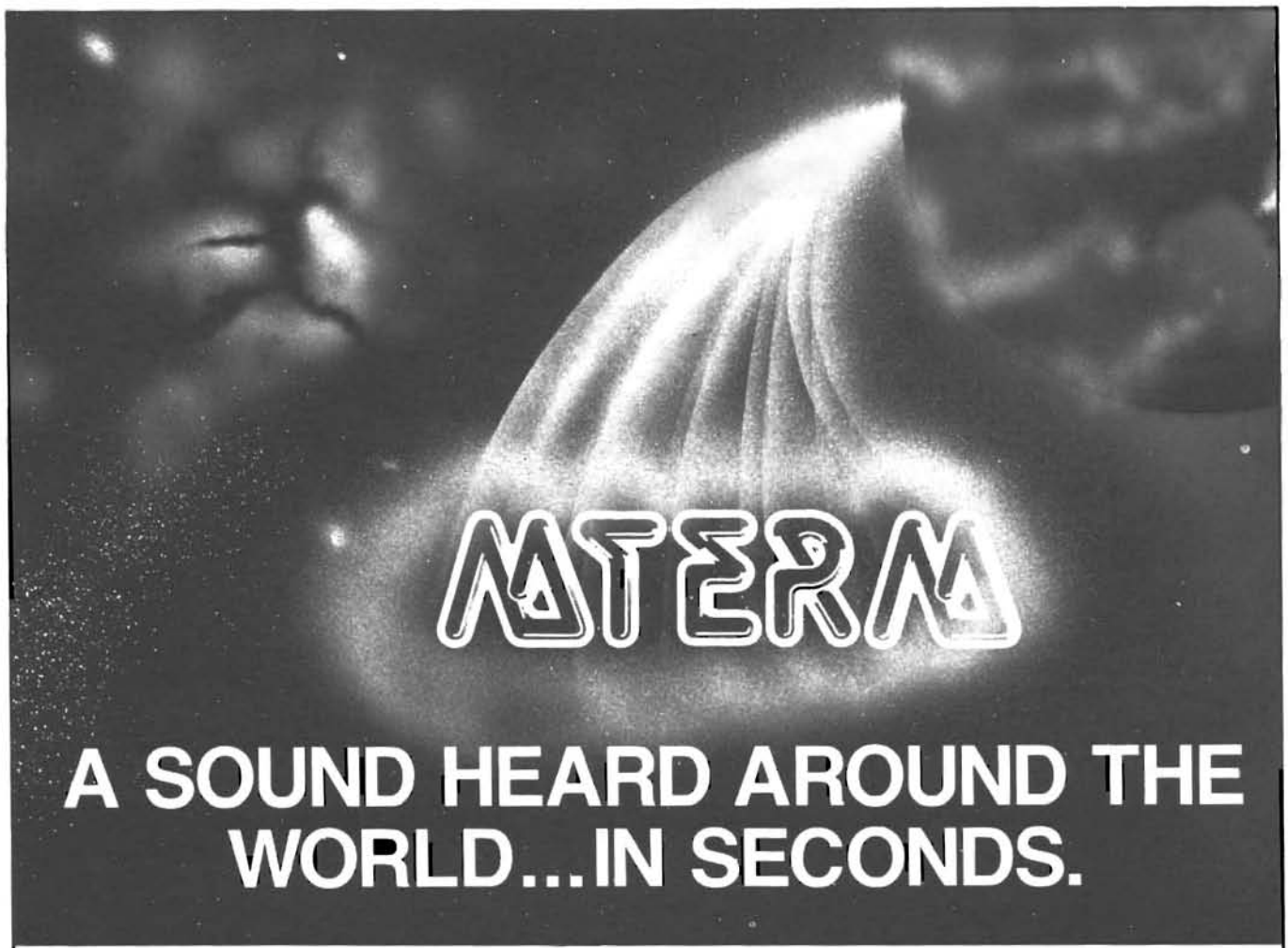
Send Payment to: Heath Users' Group
Hilltop Road
St. Joseph, MI 49085

616-982-3463

REMark®
Issue 46 • November 1983

on the stack

ON THE COVER: Pictured is part of the VOTECH ED Class at Benton Harbor High School, Benton Harbor, MI. See page 5.



A SOUND HEARD AROUND THE WORLD...IN SECONDS.

Micro-System Software presents MTERM – for Z-100, H-100 – the high speed smart terminal – for today and tomorrow's telecommunications. MTERM is the first truly high speed terminal program. Most versions can run at speeds up to 4800 baud without the insertion of null characters. And with the advent of 1200 baud modems and information services, this becomes an important consideration.

But MTERM is more than just a terminal program. It's a complete communications package. Supporting file transfer in both the traditional ASCII mode and the new "error-free" direct file mode.

Not only does MTERM surpass the competition in features offered, but it also offers ease of operation second to none. MTERM's powerful translation tables let you interface MTERM to any variety of communications service. Its unique MacroKey function allows you to have ten "user-defined" keys that transmit up to 64 characters at the touch of a key. MTERM even goes so far as to allow you to dial the phone and transmit the buffer at a specified time completely un-attended by the operator. MTERM is supplied with a user's manual that clearly explains program operation and answers many questions before they are asked.

MTERM will open up the world of telecommunications for you as never before. By using this program, you can do more in less time. The serious computer user cannot afford to be without a MTERM in their library.

And although MTERM geometrically outperforms the others, it's priced at only a fraction of their cost. MTERM is supported through the Micro-Systems technical support network, giving it support previously unavailable for a terminal program. Whether it is for the features, the performance, the documentation, the support, or the price, you just can't lose with MTERM.

CHECK THE FEATURES:

1. Spooled printer output and screen print function.
2. Easy to use translation tables with ready made default files for most applications.
3. Maximum auto dial support with user definable tables to allow dialing from a list of numbers at the touch of a key.
4. MacroKey function allows you to have up to ten "user-defined" keys with a maximum of 64 characters per key. Can be used for multiple "auto-logons" or any of hundreds of other applications.
5. Return to menu and execute MTERM command while still receiving data. Screen will be updated upon return. Never any lost data with MTERM!
6. Supports file transfer with either the traditional ASCII mode or new "error-free" direct file mode.
7. DOS commands from menu without exiting program.
8. Large capture buffer for uploading and downloading.
9. MTERM supports most major brands of modems, with full auto-dial capabilities where applicable.
10. MTERM fully supports some of these modem's advanced features that the competitors won't touch.
11. Simple and easy to use program with clear users' manual.

Now Available at Heath Kit Stores.

MICRO-SYSTEMS SOFTWARE, INC.

4301-18 Oak Circle, Boca Raton, Florida 33431, Telephone: (305)983-3390
Toll Free 1-800-327-8724

VOTECH ED:

New Look and New Meaning for Vocational Education

Myron Kukla
Heath Co.

Vocational Education, which for generations of American High School students has meant shop classes and woodworking, is taking on a new look and meaning for students at Benton Harbor High School in southwestern Michigan. The new look is Vocational Technical Education, or "VOTECH" for short.

Paul Bergen, Director of Vocational Education, explains, "In its simplest form, VOTECH Education means introducing high technology into all areas of traditional vocational education training."

And, in the 1980's, high tech translates into just one word — computers.

The Benton Harbor School System embarked upon the transition to VOTECH and high tech more than a year ago with the establishment of Industrial Advisory Boards and Internal Curriculum Committees to set the framework for the VOTECH Program. Starting in September, the Benton Harbor School System, through its Vocational Education Department, began offering the first of a planned series of computer applications programs designed to prepare its students for today's job requirements and, perhaps more importantly, prepare students for the job opportunities of local industry.

"The School System's philosophy on VOTECH courses," said Bergen, "is to first target major local industries that would eventually be employing the graduates of VOTECH courses and then involve those industries in an advisory capacity in setting up the courses. The courses and curriculum we've developed are designed to give the student training in specific applications of computers in areas such as accounting, word processing, programming, and secretarial procedures. The most important aspect of the program is that we have structured the curriculum of each course to be compatible with the need of local industry from the type of computer the students are using, to the software they are learning."

Two of the major industries targeted by the Benton Harbor VOTECH Program are the Heath Company and Whirlpool Corporation from the Twin Cities area of Benton Harbor and St. Joseph, Michigan. The computers used by the companies and selected by the course are the Zenith Data Systems Z-100 model Desktop Computer. The 16-bit, Z-100 model Business Computer is used extensively in the administrative departments of both companies.

Ron Griffin, Supervisor of Information Systems at Whirlpool, assisted the Benton Harbor School System in its selection on the Z-100 system for the courses as well as the business applications software to be taught.

"The computer courses being offered through Benton Harbor Vocational Education Center are aligned with the computer applications of the Whirlpool Administration Center," said Griffin. "Students are working with mainstream software packages that will allow them to springboard into jobs in industry."

The choice of the computer selected was doubly appropriate for Heath Company because, not only does Heath use the Z-100 in

its offices, it also manufactures the computer for Zenith Data Systems, both being subsidiaries of Zenith Radio Corporation.

"One of the real benefits of the way we've approached the set-up of the courses is that the Z-100 computer selected uses the same type of computer software as the IBM Desktop Computer," said Bergen. "Since both computers are widely used in business offices across the country, our students are not only being prepared for local jobs, but also for jobs in a wide range of industries across the country."

(The Z-100 series computer was also recently selected in a major government contract to be the desktop computer throughout all four branches of the Armed Forces, another major plus for the students graduating from the Benton Harbor VOTECH Program.)

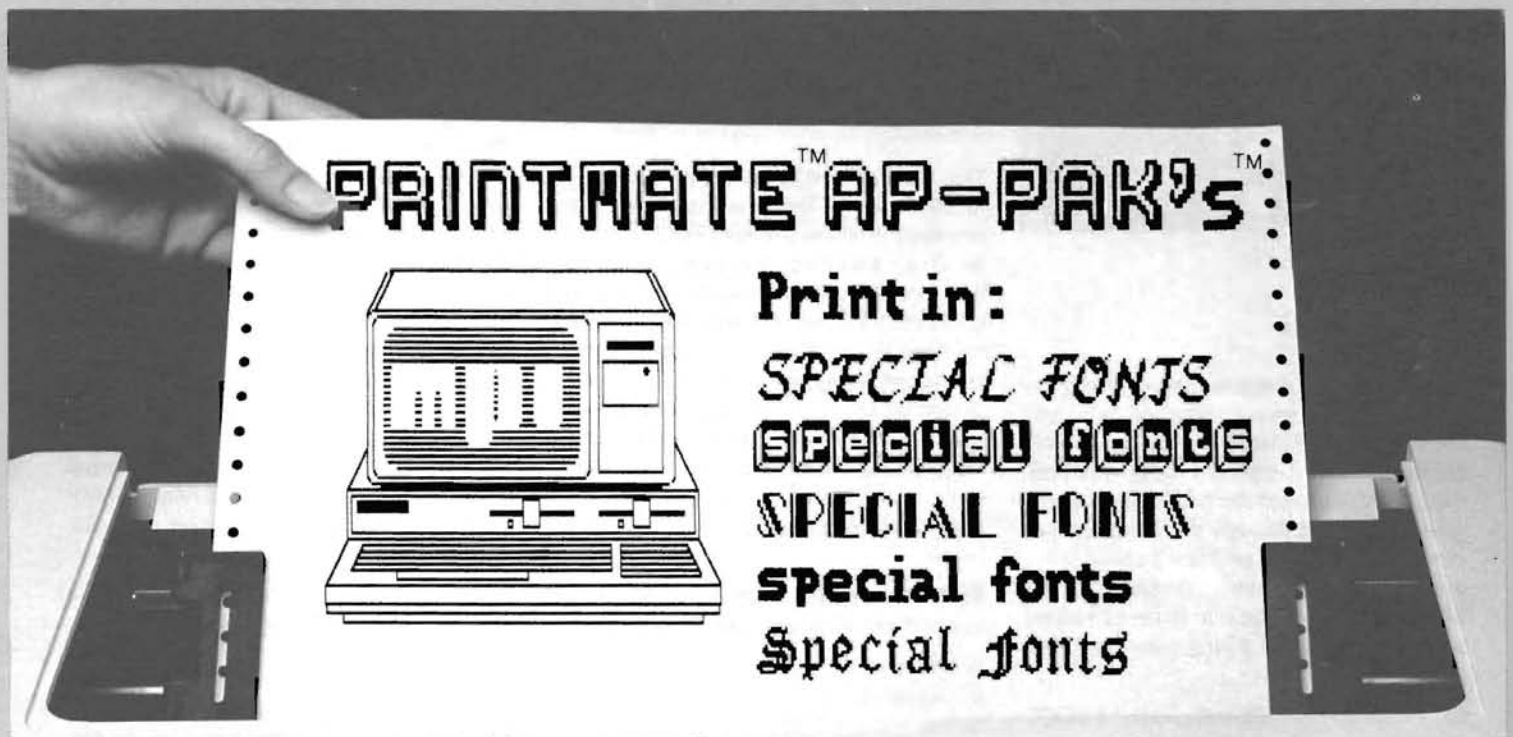
Dr. James Hawkins, Superintendent of Schools for Benton Harbor, said the cooperation received from local industry in setting up the courses has been excellent.

"Business and industry are really untapped resources for acquiring the type of expertise that is required to make high tech vocational education courses current," said Dr. Hawkins. "Our plan now is to expand this cooperation with the private sector to a greater degree including the establishment of additional courses and cooperative work-study programs."

Two courses the Benton Harbor School System is looking at adding to the current VOTECH curriculum are Computer Programming in COBOL, a high level programming language used in business, and Computer Aided Drafting (CAD).

A measure of the success of the Benton Harbor program can be found in the statistics kept by the State of Michigan for students graduating from vocational education programs. Measured on a standard of whether a vocational education graduate either enters the work force, goes on for additional training in college or enters the military, the Benton Harbor School District has a placement rate of 86 percent for its vocational education students in comparison to the State average of 70 percent.

"It's encouraging to see such figures," concluded Paul Bergen. "It goes to show that we're on the right track."



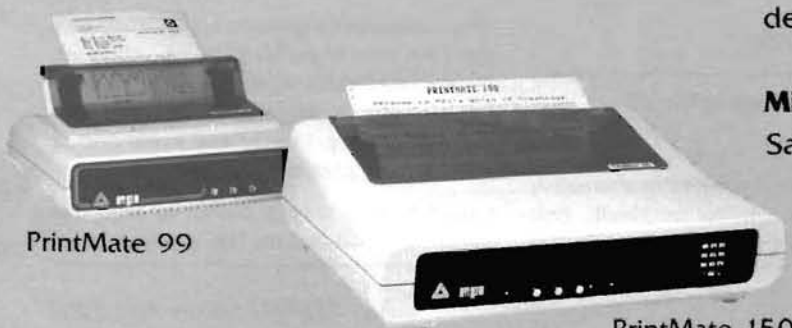
PrintMate GOES GRAPHIC WITH H/Z100!

Picture this—your logos, graphs and fancy print fonts on paper with a PrintMate. Our exclusive AP-PAK lets you print what you picture. In just one keystroke, you can send what's on the screen on to your printer. With an AP-PAK, you can intermix text and graphics on the same line and print in characters up to ⅝-inch high. AP-PAK runs with the Z-DOS™ (MS-DOS) BIOS release 1.00 (Winchester) version 1.10 and above.

With friction and tractor feed standard on the 100 cps PrintMate 99, virtually any form goes through without a hitch...from labels to multi-part. The PrintMate 99's optional single-sheet feeder lets you run single sheets through the front of the machine, thus simplifying all your personal printing. And its 1K (optional 2K) buffer provides additional printer performance.

Our premier performer for home or office is the PrintMate 150, featuring 150 cps print speed, three paper paths and 15-inch carriage. Expandable buffers (up to 68K!) dramatically increase flexibility. And with SoftSwitch,™ you get an impressive range of control at your fingertips right on the front panel of the printer.

No matter how you compute, you get more print versatility for the price with PrintMate: correspondence-quality printing, superb graphics capabilities and versatile paper handling. At MPI, we want to see you in print. See your dealer today for a demonstration.



PrintMate 99

PrintMate 150

Micro Peripherals, Inc., 4426 South Century Drive, Salt Lake City, Utah 84107 **1-800-821-8848**

The Printer People™ mpi™

H/Z 100 is a registered trademark of the Heath Company.
Z-DOS is a trademark of the Zenith Data Systems Corp.

BUGGIN' HUG

It Really Works!!

Dear HUG,

Today when I sat down to read my newly arrived "REMark", my eyes stopped on an article called "The Time Saver" by George W. Stephenson, Jr. It involved using an empty file (.COM) to speed the reloading of a program. As I read through the article, it occurred to to me that perhaps it could solve my biggest computer problem — the BDOS error. The article in Issue 42 helped some but it entailed a long drawn-out process.

So, I fired up my H89 and created a BDOS error by attempting to save a program after changing disks without typing reset. When the error occurred, I hit return and at the prompt, I typed (the empty program's name). Sure enough, the program reappeared without any damage.

I also found that if you are using a disk that does not have .COM on it, you may change disks either before or after you hit return and the result will not be changed.

Verne E. Leininger
Rt. 4, Box 296
Deer Park, MD 21550

Helpful Tips On Assembly of the H-29 Terminal Kit

Dear HUG,

I would like to relate my experience to other HUG members who enjoy the kit approach to putting their computers together. Readers, who in the future may assemble the H-29 terminal kit, should be made aware of possible confusion when transistor Q103 is installed on the video circuit board. If Q103 is installed backwards, like I did, strange things happen such as the cursor appears in the upper right corner of the CRT, the left side of the CRT fills with normal E's during the test phase but the right side fills with larger reversed E's, and the clock is in the center of the 25th line.

Checking the voltage at pin 6 of the CRT showed it to be 195 volts when it should have been closer to 500 volts. Being what I believe to be a good kit assembler (I haven't been at it quite as long as Larry Sites of CHUG), I rechecked my work for a fourth time. Everything looked fine. More voltage checks (I don't have a scope) and everything seemed to point to Q103 as the problem. Could I have installed Q103 backwards? I sure did.

The drawings on page 30 could be confusing to others also. The drawing showing the preparation of and the installation of transistor Q103 with heat sink is reversed in relation to the drawing showing its placement on the circuit board. Closer inspection of the enlarged drawing shows the edge of the circuit board facing you but this is easily overlooked (at 1:00 a.m.). A statement by Heath such as "Note: The curved portion of the heat sink should be facing the outside edge of the circuit board" should alert one to the proper orientation of Q103 on the circuit board.

To finish my story, turning Q103 around makes the H-29 one beautiful terminal.

Ted Elzerman
#3 Settler's Lane
Springfield, IL 62707

A New CRT For The H89

Dear HUG,

Here's a tip for H89 owners who have tired eyes from looking at a harsh, white screen but, like me, just can't "see" spending $75 or more to replace a perfectly good CRT.

BNF Enterprises, 119 Foster St., Peabody, MA 01960, offers a 12" green CRT for $28. The BNF part number is DOA60495. The CRT is apparently similar in all respects to the H89 CRT except that the heater voltage is 12.6V rather than 6.3V.

It is easy to modify the H89 to accommodate a 12.6V CRT. The CRT heater voltage is supplied by a filament winding on flyback transformer T2. This is the brown wire which runs from the video board, wraps around T2's core six times, and runs back to the video board. Simply remove this wire and replace it with a new wire which wraps around T2's core twelve times. You'll have to use a new length of wire because the existing wire won't reach once you've added the extra six turns. Of course, you should use wire of a similar type. Be sure you fix the new wire in place using a cable tie so the turns are tight about the flyback transformer's core.

Rodger L. Ellis
740 W. 71st Avenue
Anchorage, AK 99502

And Still More On Cooling The '89

Dear HUG,

I hope that I'm always open to an alternative point of view. As a writer for Heath, Rob Thorne should have an excellent technical point of view. His suggestion that blocking the top vents in the H-89 would not be a good idea, lead me to further investigation.

First, I warmed up my unit and dug out my trusty old Weston Photo Thermometer. Since I could not measure the actual components, a key should be in measuring the H-89's internal temperature. I slipped the thermometer through the top vents, above and to the left of the CRT. After a few minutes, the internal measurement was 93 degrees and the exhaust read 88 degrees. I quickly cleared two-thirds of the taped vents and proceeded with the test. Initially there was no temperature change, then the internal reading slowly began to drop. After 20 minutes it stopped, 5 degrees cooler, with the exhaust reading 11 degrees hotter.

These results seemed to agree with Rob's theory. If the Heath engineers feel that the fan is now carrying more heat out of the unit, then that is what we want. In this new light, I cannot recommend that the top vents be blocked. I do recommend removing the ribs above the fan and the addition of short heat sinks on the power rectifiers.

Larry Fina
Fina Software
16144 Sunset Blvd. #3
Pacific Palisades, CA 90272

Feedback On HUGCON II

Dear Walt,

Here I sit with my trusty Z-Machine with Magic Wand in hand, and I have so much to say that it's hard to know where to start. It was a real pleasure to attend your HUGCON II and to meet you and most of the HUG Staff. Although I didn't attend HUGCON I, I did read the report on it in REMark and there were some improvements, even to my untrained eye. I really liked the class concept that you used. If I could cast a vote for any changes, it would be to have a class that wasn't so high tech, such as maybe something on PRACTICAL APPLICATIONS. A lot of the classes were over the head of a relative newcomer like myself.

What I got the most out of was the personal contacts that you make accidently. A great bunch of people were in attendance including the vendors and it seems that everyone is so willing to be helpful.

Deserving special mention are your two gals that took care of the HUG booth. Although Nancy and Donna were deluged with hundreds of people, and worked like the devil all day Saturday, I never saw either one of them without a smile and a friendly greeting.

I want to give special thanks to Margaret Bacon. She handled my late application for

the conference in a very efficient and courteous manner. Even after a very hectic Saturday, she still was a very pleasant person to talk to. Margaret, I promise to get my application in early next year.

The facilities at the hotel were very nice. I would like to see the conference held there in the future.

In conclusion, I'm confused about one thing. Am I a HUGgie or a HUGger?? Thanks again to you and all the people at HUG for a great conference. I'm looking forward to next year.

Roger Fields, Captain
Rock Island Fire Dept.
1313 Fifth Avenue
Rock Island, IL 61201

Some Info On Peachtext 5000

Dear HUG,

I have encountered a problem with the Peachtext 5000 word processing package for my H-110 in that it would not print from the Edit menu, generating a "Printer not ready" message instead. My system was using a Centronics 730-3 serial printer with ZDOS configured for it under option "I", user defined, at 9600 baud, DTR negative (Pin 20), even parity.

In a conversation with Peachtree Technical Support, I was informed that Peachtext does a printer status check before printing, and that other users had reported a similar problem. Terry, Heath applications software consultant, informed me that ZDOS does not set the printer status as ready under the user defined option, and suggested that the printer might run under option B, MX-80 Serial. By configuring my printer to 4800 baud, DTR positive (Pin 20), and, by trial and error, to no parity, the Centronics 730-3 emulates the MX-80 interfacing.

Thus, while not as desirable as a patch to the ZDOS user defined option to permit use of 9600 baud rate and parity checking, it does allow Peachtext to function properly. Other users with similar difficulty may be able to use this solution, at least until someone submits a patch to ZDOS.

Kenneth D. Brown
P. O. Box 570
Newark, NJ 07101

Information Needed On Z-100 Resolution

Dear HUG,

In reading the John Stetson article on converting IBM-PC BASIC to ZBASIC (REMark Issue 44, Sept. '83, p. 9), he lightly breezed over the fact that the resolution of the Z-100 can be doubled by software control and interlaced video. This increased resolution could be very valuable to us and we are interested in how that resolution could be achieved.

Our machine is a Heath low profile version and we do have the 64K RAMs. We have only one 64K plane installed and we are presently using the ZVM-121 monochrome monitor. We realize that the 450v x 640h resolution is probably not possible with the composite video but the possibility of the increased resolution would merit the purchase of a ZVM-135 or equivalent. Any information on how such a resolution could be achieved (hopefully through assembly language) would be greatly appreciated. Color is not important.

I have been a member of HUG for only some four months but I do expect very soon to submit several articles concerning the Z-100 series of computers. I also expect to offer several programs which have been developed for the Z-100 by an unincorporated group called Cats Paw. Several of these will be freebees with source code included.

Enclosed is a small ZBASIC program which allows the user to specify a ZBASIC work file and sets the soft function keys to perform several convenient functions for that particular file. I understand and appreciate HUG's commitment to the H-89 user, but I would like to see more 100 series related articles.

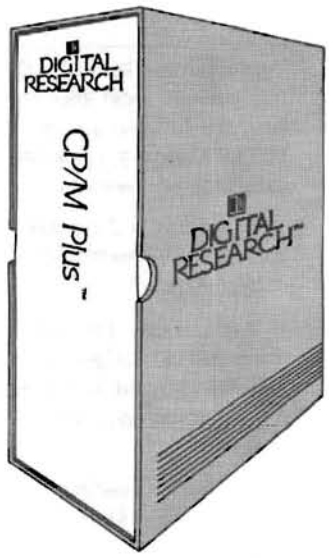
Wallace G. Hynds III
D & M Drafting Co., Inc.
114 Anderson Ave., N.W.
Knoxville, TN 37917

```
10 '
20 '                    RUN.BAS
30 '                    CP ZBASIC WORK FILE INITIALIZATION PROGRAM
40 '                    VERSION 07-02-83 (b.1)
50 '
60 DEFSTR Z
70 ZQ = CHR$(34)        ' DEFINE QUOTE
80 ZR = CHR$(13)        ' DEFINE CARRIAGE RETURN
90 '
100 CLS:LINE INPUT"ZBASIC Work File: ",ZFILE
191 '
192 '                   KEY DEFINITIONS
193 '
200 KEY OFF
210 KEY 1,"SCREEN 0,0:CLS" + ZR
220 KEY 2,"RUN" + ZR
230 KEY 3,"RUN" + ZQ + "RUN" + ZR
240 KEY 4,"SAVE" + ZQ + ZFILE + ZR
250 KEY 5,"SAVE" + ZQ + "B:" + ZFILE + ZR
260 KEY 6,"CLS:LIST" + ZR
270 KEY 7,"KEY LIST" + ZR
280 KEY 8,"KEY 9," + ZQ
290 KEY 9,"- def. by F8 -"      ' KEY 9 IS USER DEFINED BY KEY 8
300 KEY10,"SYSTEM" + ZR
310 KEY 11,"KEY OFF" + ZR
320 KEY 12,"FILES" + ZQ + "*.BAS" + ZR
391 '
392 '                   DISPLAY KEY DEFINITIONS AND LOAD WORK FILE
393 '
400 CLS:PRINT"Key Definitions:":PRINT
410 KEY LIST
420 PRINT:PRINT"Loading Work File: ";ZFILE
430 LOAD ZFILE
440 END
450 '                     *** WARNINGS ***
460 '
470 '   <F3>  Will load RUN.BAS over your work file.
480 '   <F4>  If you have changed work files by means other than RUN
490 '         then this key could save the new work file over the old
500 '         work file if the key has not been updated to the new work
510 '         file name.
520 '   <F5>  Carries the same hazzards as <F4>.
530 '   <F10> Can exit ZBASIC and lose a program if not saved.
540 '
550 '   To avoid catastrophes try to go from program to program only
560 '   by using RUN (F3).  Save a program as often as possible (F4,F5)
570 '   or remove the returns from the end of the sensitive functions.
```

Vectored to 65 ☞

FREE CP/M-Plus™ from MAGNOLIA MICROSYSTEMS

With purchase of 128K RAM Board at $595

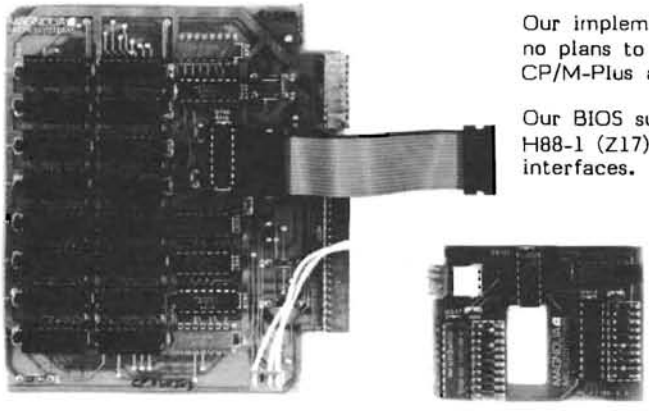


Digital Research's list price for CP/M-Plus is $350, but it will be included AT NO EXTRA CHARGE with purchase of our 77318 128K RAM board at its regular price of $595 by mentioning this ad when ordering.

CP/M-Plus™ support is available for Z89 and Z90 computers with our 128K RAM board and either the Z89-37 (Z90) or our own 77316 Double Density disk controller. Over 700K of files are included, so it will not be distributed on Hard Sectored media.

Using banked RAM, disk performance is enhanced through Hashed Directory tables, Directory Buffers, and LRU Data Buffers.

New utilities and features include a 'Help' command; optional Password protection and Time and Date Stamping of files; Console Redirection to or from disk files; and many others.



Our implementation of CP/M-Plus REQUIRES the use of our 128K RAM board. We have no plans to implement an un-banked memory version, most of the advantages of CP/M-Plus are neither available (nor practical) in an un-banked version.

Our BIOS supports both our 77316 and Zenith's Z89-37 Double Density controllers, Heath's H88-1 (Z17) Single Density controller, and our 77314 CORVUS and 77320 SASI-bus interfaces.

We are including the SOURCE code for our BIOS, together with Digital Research's MAC, RMAC, LINK, and SID software development tools, you may customize it to your specific application.

Orders must specify the Double Density controller and boot drive: the Z89-37 (as used in the Z90) or our 77316 (5-inch or 8-inch).

Earlier purchasers of the 128K RAM board, who are registered owners of Magnolia Microsystems CP/M 2 (S/N 2-175-xxxx) can update to CP/M-Plus for $100 (plus shipping and handling).

DISK INPUT/OUTPUT (I/O) BOARDS

These boards are made available apart from our subsystems for use by 'Systems Integrators'. End-users should carefully consider their skills and desired level of involvement before undertaking this responsibility; purchasing a complete subsystem from a qualified integrator may be a wise choice.

Although similar in name and apparent function, these boards fall into two distinct classes:

- INTERFACE boards connect a disk subsystem (containing a controller) to the computer. These boards include the Z89-47 and Z89-67 and our 77314, 77317, and 77320 interfaces.
- CONTROLLER boards actually control a plain disk drive. These boards include the H88-1 (Z17) and Z89-37, and our 77316 Double Density Controller.

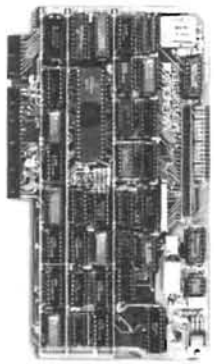
All Magnolia I/O boards use proprietary techniques to expand the 89's I/O addresses, allowing use of more than Zenith's 5 (3 serial ports, 2 disk controllers). Since our INTERFACE boards also contain 3 RS232 ports (replacing the machine's H88-3) allowing expansion of the I/O capability without using an additional card slot.

Each Magnolia I/O board package includes:

- The disk I/O board itself
- A new I/O decoder PROM
- A new Monitor EPROM
- A copy of MMS CP/M 2.2

Magnolia's CP/M software supports over 20 soft-sectored media formats, including all existing Zenith formats.

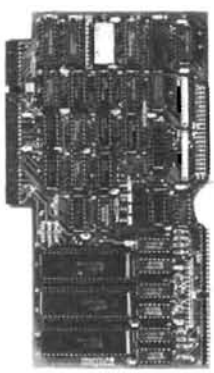
Double Density Disk Controller reduced to **$495**
order number 77316



This board adds complete hardware and software support for four 8" single- or double-sided AND four 5" SS or DS, 48- or 96-track-per-inch (40- or 80- track) drives, at both single and double recording densities.

This controller is compatible with 8" drives by Shugart, Qume, and Tandon; 5" drives by Siemens, Tandon, and MPI; and most other 'industry standard' drives (possibly depending on the skill level of the experienced 'Systems Integrator').

SASI-bus Winchester Interface **$350**
order number 77320

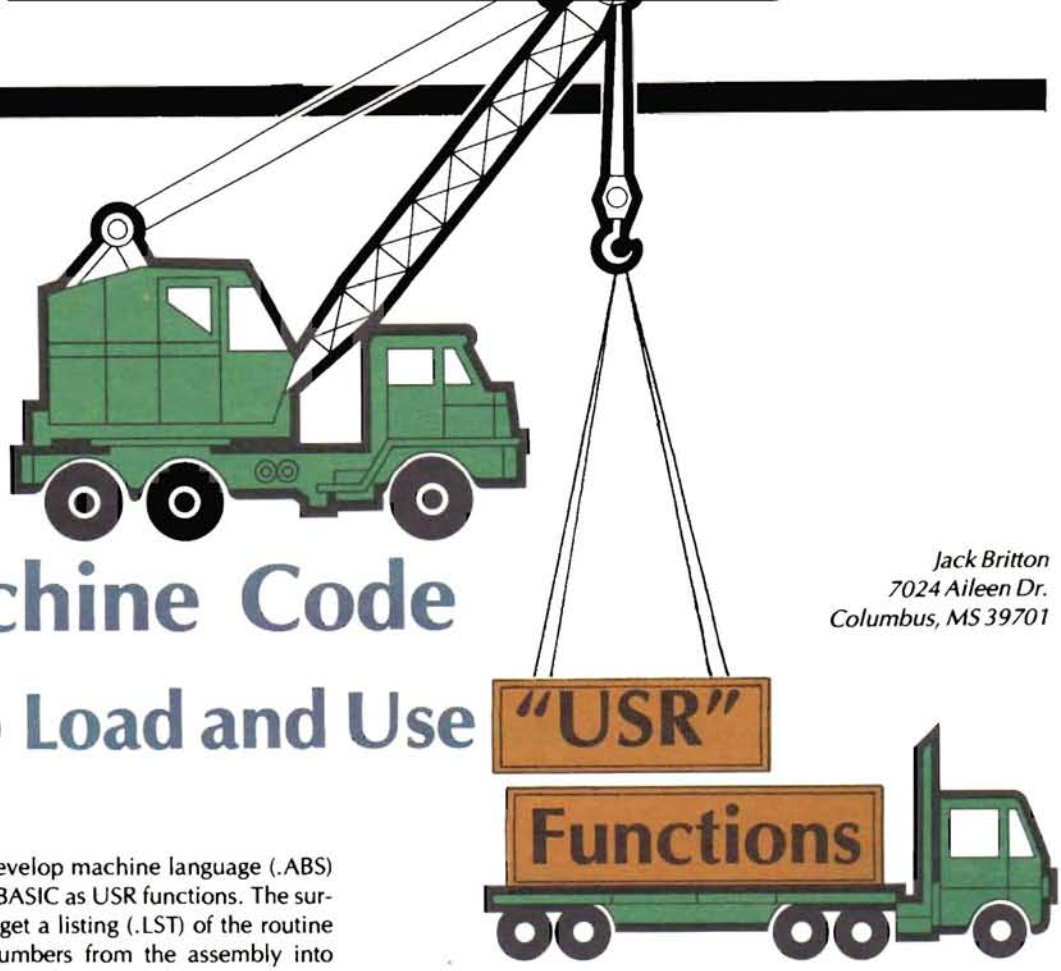


This board (functionally replacing the Z89-67 interface) contains 3 RS232 ports and can support up to 8 SASI-bus disk controllers, with up to 4 drives on each!

It is compatible with Winchester controlers by Xebec, DTC, Shugart, and others, as well as Zenith's Z67. CP/M software support for popular controller/drive combinations is included.

MAGNOLIA MICROSYSTEMS, INC. • 2264 - 15th Ave West
(206) 285-7266 (800) 426-2841 • Seattle, Washington 98119

Use MBASIC and Machine Code To Load and Use "USR" Functions



Jack Britton
7024 Aileen Dr.
Columbus, MS 39701

I have seen many articles that develop machine language (.ABS) programs that are to be used by MBASIC as USR functions. The surprise comes when I am forced to get a listing (.LST) of the routine and then, by hand, place the numbers from the assembly into MBASIC DATA statements, which are then POKEd into memory at run time. I feel the computer should load the .ABS file for me, thus eliminating the probability for error and the time of putting the numbers in by hand.

This article describes an alternative to POKEing the data into memory by using MBASIC to load binary programs into string variables for use as a USR function. The particular .ABS program I have developed for an example of this technique will then aid the MBASIC program in loading another .ABS program into reserved memory for execution during the MBASIC program run.

Before I continue I should tell you that I work on a 48K H89 with a single, hard sector, single density drive. Therefore, much of what I say regarding disk access assumes this configuration. However, the basic concepts should apply to all systems, and I believe the final program will work with any configuration using HDOS and MBASIC.

I will discuss in detail two programs: COPYBLK.ASM (Figure 1) and TESTPRG.BAS (Figure 2). COPYBLK will be read from disk and then reside in an MBASIC string. It will be used to COPY BLocKs of data to designated areas of memory. In this application the data it moves is another .ABS program which is temporarily stored in an MBASIC string. TESTPRG (TEST PRoGram) demonstrates how to put a binary file (COPYBLK.ABS) into a string and use it as a USR function. It then loads another .ABS program (TRIM.ABS) into a string, which COPYBLK then copies into the appropriate area of reserved memory. After the copy is complete, TESTPRG tests routine TRIM.ABS to verify it works properly. The third program listing, TRIM.ASM (Figure 3) is an adaptation of the program on page E-5 of the MBASIC manual. It merely "trims" off the trailing blanks on a given string. Please reference the MBASIC manual if you need more details. All three of the programs are self-documenting, so the text will focus primarily on concepts and particular programming techniques. Before discussing COPYBLK, let's explore how to load an .ABS file from disk into a string, and some limitations associated with trying to execute that string as a USR function.

"Random" disk access is the way MBASIC allows access to every byte on a sector of the disk. For example, the following program will access all 256 bytes of the first sector of program MBASIC.ABS:

```
10 OPEN "R",#1,"SY0:MBASIC.ABS"
20 FIELD #1, 255 AS A$, 1 AS B$
30 GET #1
```

A$ contains the first 255 bytes, and B$ contains the last byte. This program will successfully retrieve any file regardless of the file type (i.e. BASIC file, binary file, text file, etc.). We can print the value of each byte in the sector with this program:

```
40 FOR I = 1 TO 255
50 PRINT ASC(MID$(A$,I,1)),: REM PRINT THE VALUE OF THE Ith BYTE
60 NEXT
70 PRINT ASC(B$)
```

The above program prints the decimal value of each byte of the sector. Note that A$ can't hold the entire 256 byte sector because MBASIC strings are restricted to a maximum of 255 bytes. Suppose we replace "SY0:MBASIC.ABS" at line 10 with "SY0:COPYBLK.ABS", the assembly of the program in figure 1. Since the binary program COPYBLK is less than 255 bytes long, A$ contains the entire program, plus some garbage at the end of A$. A$ can be used as temporary storage of the program, or, if the programmer is clever, can be executed directly as a USR function.

There are some limitations in using a string as the place to hold an executable binary program. First, to fit into a string the binary program must be less than 247 bytes long. There are 8 other bytes associated with a binary file which are explained later in "ABSDEF reviewed". The maximum number of bytes in an MBASIC string is 255. Subtracting the 8 byte header leaves 247 bytes. This is not much of a restriction, however, since almost all of the routines I write for USR

functions are less than 100 bytes long. After all, I am writing the program in MBASIC for a reason, and the USR routines are there just to speed up some repetitive tasks or offer some unique function that MBASIC has difficulty handling.

Second, the binary program must be able to run regardless of its location. MBASIC may store string variables anywhere in the string storage area of memory. Also, MBASIC will often change the location of a string during a program run. Every time it moves the string, our binary program is also moved. This means that a program stored in a string cannot jump, call, or make reference to any location which is within the string. For example, in COPYBLK (Figure 1), an instruction JMP START or CALL START will not work, but CALL $MOVE, a call into ROM (outside of the string) will work. This restriction is greatly relieved if you use the Z80's relative jumps, but Heath does not recommend their use for upward compatibility purposes. Note that it is intended that COPYBLK be executed WHILE RESIDING in an MBASIC string. COPYBLK will be used to load other binary programs from disk into reserved memory (with MBASIC's help). Let's look at what COPYBLK does.

COPYBLK.ASM

The purpose of COPYBLK is to COPY a BLocK of data from one place in memory to another. The data block to be copied is presumed to be an executable binary file which is temporarily stored in a string.

Figure 1

```
00001   ***  COPYBLK.ASM
00002   *    USES $MOVE (in ROM) TO PUT .ABS FILE IN PROPER LOCATION IN MEMORY.
00003   *
00004   *    Since this file will be put into a string variable in MBASIC,
00005   *    it may exist anywhere in memory.  Therefore, it will not use
00006   *    any JMPs or CALLs to any routine not in ROM.
00007   *
00008   *    Assumes file (string sent to this routine by MBASIC) is headed by 8
00009   *    bytes in ABSDEF format.  However, it will move any file beginning
00010   *    with byte 9 as long as "Length" (bytes 5 & 6) is less than 256.
00011   *
00012   *
00013   *    NOTE: To use the Z80 Block Data Transfer command LDIR
00014   *       (Load-Increment-Repeat), change last 3 lines:
00015   *
00016   *            FROM              TO
00017   *
00018   *       XCHG              LDIR    DB 355Q,260Q     (ED,B0 hex)
00019   *       JMP     $MOVE             RET
00020   *       END     START             END     START
00021   *
00022   *    ENTRY:       A = 3   (if string sent)
00023   *                 (DE) = address of string descriptor
00024   *                 (HL) = address of FAC (not used)
00025   *
00026   *
00027           ITEIT   MOVE.ACM
00029           ORG     42200A          Can ORG this routine anywhere!
00030   START   CPI     3               check if string sent
00031           RNE                     no, return
00032           XCHG                    (HL) = address of string descriptor
00033           INX     H               skip "length of string" byte
00034           MOV     E,M
00035           INX     H
00036           MOV     D,M             (DE) = address of string
00037           XCHG                    (HL) = address of string
00038           INX     H               skip ABS.ID bytes (2): see ABSDEF.ACM
00039           INX     H
00040           MOV     E,M
00041           INX     H
00042           MOV     D,M             (DE) = load address ('to' destination)
00043           INX     H
00044           MOV     C,M             (BC) = length of entire record, but
00045           INX     H                       must be less than 255:
00046           MOV     B,M                     ie B must = 0.
00047           MOV     A,B
00048           CPI     0
00049           RNE                     RETURN if length > 255
00050           INX     H               skip ABS.ENT bytes (2)
00051           INX     H
00052           INX     H       (HL)=address of start of .ABS code ('from' address)
00053           XCHG            now (DE) ='from' address, (HL) ='to' address
00054           JMP     $MOVE           $MOVE has 'RET' in it.
00055           END     START
```

This string is sent to COPYBLK when MBASIC calls it via a USR function.

COPYBLK has one major restriction; since it will be stored in an MBASIC string, which may be located anywhere in memory, the code of COPYBLK cannot be dependent upon its location.

To understand what COPYBLK is doing, you must first understand how an MBASIC string is stored and how a USR routine finds that string when sent by MBASIC. MBASIC stores most of its variables after the program text alphabetically and according to variable type. However, in the case of string variables, only a string descriptor of three bytes is stored there. When using VARPTR with strings, the address of the first byte of the descriptor is returned. The string descriptor looks like this:

```
Byte 1 = length of string (255 or less)

Bytes 2 & 3 = address of string
```

However, all is not as straight-forward as we would hope. The address bytes (bytes 2 & 3) are put in reverse order. If the actual address is 263200A, then byte 2 is 200A and byte 3 is 263A. Much has been written in REMark and other publications about why this occurs, so I won't explain it here. Keep in mind that when MBASIC sends a string to a USR function, it sends the address of this descriptor in register pair (DE). For more information, read pages E-4 and E-5 in appendix "E" of the Heath MBASIC Manual.

Now let's look at the listing for COPYBLK (Figure 1). When MBASIC sends a string to a USR function, the 'A' register contains a 3. Lines 30 and 31 check that a string is sent. Lines 33 through 37 step through the string descriptor until (HL) contains the address of the string. Remember, the string is actually sector 1 of an .ABS file. HDOS puts an additional 8 bytes on the front end of every binary file that it creates which gives a lot of information about the file. File ABSDEF.ACM describes these 8 bytes.

ABSDEF Reviewed

On the HDOS Software Tools disk, the file ABSDEF.ACM defines an 8 byte header that is on every binary file created by the assembler ASM.ABS. ABSDEF looks like this:

```
**      ABS FORMAT EQUIVALENCES.
        ORG     0

ABS.ID  DS      1       377Q = BINARY FILE FLAG
        DS      1       FILE TYPE (FT.ABS) (See FILDEF.ACM)
ABS.LDA DS      2       LOAD ADDRESS
ABS.LEN DS      2       LENGTH OF ENTIRE RECORD
ABS.ENT DS      2       ENTRY POINT

ABS.COD DS      0       CODE STARTS HERE
```

Every time the operating system loads a binary file for execution, it checks the header to:

1) make sure it is a binary file [377Q, followed by a 0];

2) find where to load the program (Load Address);

3) find the length of the program in bytes (not including the eight byte header);

4) find where to start execution (Entry Point).

Based on the COPYBLK listing, we would expect the first eight bytes of file COPYBLK.ABS to be:

```
     Byte 1 = 377Q
     Byte 2 = 000
Bytes 3 & 4 = 042200A = load address (reference 'ORG' statement)
Bytes 5 & 6 = 000036A = length of program
Bytes 7 & 8 = 042200A = entry point (identified by 'END'
                        statement:  START = entry point = 042200A)
```

Now use this simple MBASIC program to check the header:

```
10 OPEN"R",#1,"SY0:COPYBLK.ABS"
20 FIELD #1, 8 AS A$
30 GET #1,1
40 FOR I=1 TO 8
50 REM
60 REM ** PRINT BYTE 'I' OF A$ IN OCTAL FORMAT
70 REM
80 PRINT OCT$(ASC(MID$(A$,I,1)));" ";
90 NEXT
99 CLOSE
RUN
377 0 200 42 36 0 200 42
Ok
```

Bytes 1 & 2 are as anticipated, but look at byte pairs 3-4, 5-6, and 7-8. Once again the Z80 put the bytes in "reverse" order. Byte 3 is 200A, byte 4 is 042A. Together they show the load address as 042200A. Likewise, bytes 5 & 6 are 036A and 000A respectively, indicating a length of 000036A bytes. Bytes 7 & 8, 200A and 042A, respectively, indicate an entry point of 042200A. Now that we know the header format, it will be easy to follow how COPYBLK and MBASIC use these 8 bytes to get the load address, length, and entry point address. The MBASIC program will also use them to check that the program is being put into reserved memory space.

Lines 28 through 53 of the COPYBLK listing simply step through the 8 byte header and put the information required by $MOVE into the appropriate registers (BC = count, DE = 'from' address, HL = 'to' address). Lines 47 and 48 check to make sure we're not trying to move a program which is longer than 255 bytes.

I have mentioned before the restrictions on putting an executable program into a string. Obviously, these are too restrictive for most applications. However, if we use COPYBLK to put other .ABS programs into reserved memory, then any .ABS program on disk can be put into its proper position in memory. Since this new program will be stationary in memory (not moving around like COPYBLK), none of the previous restrictions apply. The MBASIC program TESTPRG demonstrates the application of this technique.

TESTPRG.BAS

You should read the remarks in TESTPRG (TEST PRoGram, Figure 2) to get the general flow and logic of the program. However, I have four points which will clarify some programming techniques used.

First, the variables FWA (First Word Address) and LWA (Last Word Address) defined at the beginning of the program (line 110) represent the limits of reserved memory as defined in the CLEAR statement. I am running on a 48K system. If your system needs the re-

Figure 2

```
100 CLEAR 1000,46499!(46600!): REM ** May be different for your system
110 FWA=46499!: LWA=46650!:REM ** Chg FWA & LWA if chg CLEAR statement
120 REM
130 REM *** Function "FNSTRG" returns the address of the beginning of
140 REM *   the string "ZZ$".
150 REM
160 DEF FNSTRG(ZZ$)= PEEK(VARPTR(ZZ$)+2)*256+PEEK(VARPTR(ZZ$)+1)
170 REM
180 REM *** load COPYBLK.ABS into B$
190 REM
200 Z$="SY0:COPYBLK.ABS"
210 GOSUB 770: REM              *  Get "COPYBLK" from disk and put into A$
220 N=FNSTRG(A$):REM            *  N = address A$ (COPYBLK)
230 N=PEEK(N+5)*256+PEEK(N+4):REM*  N = length of COPYBLK (see ABSDEF.ACM)
240 B$=MID$(A$,9,N):REM         *  B$ = .ABS prog (without ABSDEF header):
250 REM                         *  DON'T REDEFINE B$!!!
260 REM
270 REM *** load TRIM.ABS to location 46500 (ORGed location)
280 REM
290 Z$="SY0:TRIM.ABS"
300 GOSUB 770: REM              *  Get "TRIM.ABS" from disk and put into A$
310 GOSUB 580: REM              *  move program in A$ to CLEARed space
320 IF A$="" THEN STOP: REM     *  USED ONLY FOR DEBUGGING
330 N=FNSTRG(A$):N=PEEK(N+7)*256+PEEK(N+6):REM *  N = entry point
340 IF N>32767 THEN N=N-65536!: DEFUSR1=N: REM *  See text
350 REM
360 REM     test TRIM.ABS
370 REM
380 LINEINPUT"TYPE LINE: ";A$:REM          *  get line from console
390 PRINT"LINE INPUT IS:  (";A$;")":REM    *  prt line with brackets
400 Z$=USR1(A$): REM                       *  trim off trailing blanks
410 PRINT:PRINT"LINE AFTER TRIMMED IS:  (";A$;")"
420 PRINT
430 GOTO 380:REM                           *  CTRL-C to STOP
440 REM
450 REM ****  DEFINE SUBROUTINES  ****
460 REM
470 REM
500 REM  *** Routine checks that A$ will be moved to
510 REM  *   'CLEAR'ed space (LWA to FWA).  If not, sends message
520 REM  *   sets A$ to null, and RETURNs.  If move is OK, then
530 REM  *   finds location of MOVE program (B$), and executes the
540 REM  *   move. NOTE:  IT IS NEVER CHECKED IF PROGRAM LOADED INTO
550 REM  *   A$ IS ACTUALLY AN .ABS FILE.
560 REM
570 REM
580 N=FNSTRG(A$)
590 ADD=PEEK(N+3)*256+PEEK(N+2):REM *   Get move address
600 N=PEEK(N+5)*256+PEEK(N+4):REM   *   Get length of file
610 IF N>247 THEN PRINT "Program too long":A$="":RETURN
620 REM
630 REM  ** see if program will load between LWA and FWA (CLEARed space)
640 REM
650 IF ADD<FWA+1 OR ADD+N>LWA THEN PRINT @
    "Abort move: ";ADD;" to ";ADD+N;" not in CLEARed space":A$="":RETURN
660 N=FNSTRG(B$):IF N>32767 THEN N=N-65536!: REM  *  See text
670 DEFUSR0=N
680 Z$=USR0(A$)
690 RETURN
700 REM
710 REM
720 REM  *** Routine gets first sector of file Z$ and puts
730 REM  *   it into A$.
740 REM  *   NOTE: USES FILE #1 FOR I/O OPERATION.
750 REM
760 REM
770 OPEN"R",#1,Z$
780 FIELD #1,255 AS A$
790 GET #1,1
800 CLOSE #1
810 RETURN
```

Figure 3

```
***  TRIM.ASM
*
*    This program is adapted from the example program
*    shown on page E-5 of the MBASIC manual.  It's
*    purpose is to 'trim' off trailing blanks of a
*    string.  For more information, see the MBASIC
*    manual.

$TYPTX  EQU     31136A          in H17 ROM

        ORG     46500

TRIM    CPI     3               string sent?
        JNE     ERR             no, type error msg

        XCHG                    HL = strg Desc addrs
        MVI     B,0
        MOV     C,M             BC = length of strg
        MOV     A,C
        CMP     B               test if zero length
        RZ                      nothing to do
        INX     H
        MOV     E,M             Byte 2 of descriptor (lo addrs)
        INX     H
        MOV     D,M             Byte 3 of descriptor (hi addrs)
        XCHG                    HL = strg addrs
        DAD     B               HL + BC = addrs of last+1

LOOP    DCX     H
        MOV     A,M             get 'last' char
        CPI     ' '             test if blank
        JNZ     FIN             no, we're done
        DCR     C
        JNZ     LOOP            all chars done

FIN     XCHG
        DCX     H               back up to
        DCX     H               byte 1 of desc. (length)
        MOV     M,C             reset length
        RET                     go home

ERR     CALL    $TYPTX
        DB      12Q,12Q,'String not sent to USR ...',12Q,212Q
        RET
        END     TRIM
```

served memory to be lower, change the CLEAR statement limits, LWA and FWA. TESTPRG uses these limits at line 650 to check if TRIM.ABS is being put into reserved memory. The second expression in the CLEAR statement (FWA) is really the last address that MBASIC CAN use. This means that, although I ORGed program TRIM to 46500, I must use 46499 as the second expression of the CLEAR statement. MBASIC will use up to AND INCLUDING address 46499.

Second, the function FNSTRG defined in line 160 looks at the two address bytes of the string descriptor of the passed string, and returns the actual address of that string. The string descriptor is found using the VARPTR function.

Third, a quirk of MBASIC integers rears its head when defining USR function addresses as in lines 340 and 660. Line 340 is:

340 IF N>32767 THEN N= N-65536

Before line 340, N = 46500 (Entry address of TRIM.ABS). However, MBASIC doesn't recognize any integer above +32767. Also, when using the DEFUSR statement, MBASIC automatically tries to convert the given address to an integer. When we DEFUSR0 to 46500, MBASIC flags an error telling us the number is too big! To solve the problem, we merely take the two's complement of the address by subtracting 65536 from the address. Obviously, this results in a negative number, but the negative two's complement integer gives the bit pattern for the desired address. To show how this occurs, lets take the case of 'DEFUSR = 65535'. 65535 is 1111 1111 1111 1111 Binary. It is also -1 in 16 bit two's complement arithmetic. Note that 65535 - 65536 = -1. Don't get upset about defining the entry address of a USR function as a negative number. MBASIC merely puts the two bytes, whatever they are, into its USR table and doesn't worry about it. Neither should you.

The last item of importance is in the subroutine which moves TRIM.ABS into reserved memory (lines 500 to 690). It is very important that every time COPYBLK (in B$) is used, the USR0 entry address is redefined (lines 670 & 680). This is because MBASIC may have moved B$ since it was last used. However, as long as strings are not changed, B$ will not move while it is running.

Conclusion

You can see that this scheme allows for a lot of flexibility during an MBASIC program run. If multiple .ABS programs need to be used, they could be read from disk as needed, overlaying the same area in memory if desired. Each time a new .ABS routine is needed merely ...

```
____ Z$="SY0:XXX.ABS"
____ GOSUB 770
____ GOSUB 500
```

... and the routine will be in memory.

I have discussed how to put a binary file into an MBASIC string, how to use that program while it resides in the string, and given an example program that aids MBASIC in loading other executable binary files to their proper memory locations for later use. Many aspects of MBASIC and HDOS have been discussed in order to give an understanding of this technique. Knowledge of random files, ABSDEF.ACM, MBASIC string storage format, and assembly language programming were used to develop these programs. With a little thought and programming overhead, the limitation of length of the .ABS programs that COPYBLK (or a similar program) can copy could be eliminated. I hope that these programs have helped you in using USR functions, or at least offered some ideas for your own programming.

About the Author:

Jack Britton, *a Captain in the Air Force, is currently a T-38 Instructor Pilot at Columbus AFB, MS, after having piloted RF-4s for two years at Bergstrom AFB, TX. He received his bachelor's degree in "Quantitative Methods" from St. Thomas College, St. Paul, MN, and has retained an interest in computers ever since. When he is not flying or 'playing' with his H89, Jack spends much of his time with his wife and his 4 month old daughter, Amy. His other hobby, music, provides an artistic outlet by playing the 'cello and singing in local groups in the city of Columbus, MS.*

NEW FROM FLOPPY DISK SERVICES, INC.

the H-89 TWOET SYSTEMS

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

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

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

Model TWOET 455

2 Shugart SA-455 half height
48 tpi double sided
All hardware
Metal, shielded mounting plates
Data cable with chassis connector
Power 'Y' connector
Complete documentation
Price .... $595.00 complete

Model TWOET 465

2 Shugart SA-465 half height
96 tpi double sided
All hardware
Metal, shielded mounting plates
Data cable with chassis connector
Power 'Y' connector
Complete documentation
Price .... $695.00 complete

Wondering what to do with your internal drive if you buy this system? Here's the solution. If you purchase a dual half height system for your Heath computer from Floppy Disk Services, just include an extra $60.00 plus shipping and receive a single 5¼ case with power supply *and* data cable ready to receive your SIEMENS internal drive! The case with data cable is normally a $75.00 item. And the cable that comes with your TWOET system includes the external chassis disk I/O connector.

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

Dealer inquiries invited.

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

Toll Free Order Line:
(800) 223-0306

Tech Help or Info:
(609) 799-4440

Floppy Disk Services, Inc. also sells other drives and peripherals for your system. Call for info and catalog!

FLOPPY DISK SERVICES™ INC.

741 Alexander Rd. Princeton, NJ 08540

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

A ZDOS Modem Package ... MTERM

Terry L. Jensen
Software Developer

When I first received MTERM, my first thought was "Oh no, not another modem program". This is (literally) the fifteenth modem program I have used. However, this program is different; MTERM is written for ZDOS. After scanning through the manual I began to realize that this package had some unique features which I was anxious to try.

The MTERM package includes a binder, documentation, registration card, and source disk. The entire package is contained in an attractive medium-sized three ring binder, with the manual assembled and ready to use. The disk is stored inside the binder. Also included is an announcement, which explains that MTERM requires the ZDOS BIOS release 1.00 version 1.10. (I tried it on version 1.00 and it was right, it did not work.)

I read through most of the manual before booting up ZDOS and running MTERM. The documentation was written by Mark Lautenschlager of Micro-Systems Software. I would like to comment that in my opinion, he did an excellent job of organizing and explaining the steps and operation of MTERM. The manual, I found, was interesting reading, as Mark does not just explain how to use the package but he points out the why's of some of the functions. Mark also gives comments along the way that help the reader feel comfortable with the manual and the program.

What good is a great manual if the program is sloppy, right? Well, the author, Steve Pagliarulo, did NOT do a sloppy job writing MTERM, as I soon discovered.

MTERM has the usual modem options of receiving and sending files, selectable system parameters (e.g. Baud rate, word length, etc.), and a receive buffer. It has features that are included on only some modem packages, e.g. predefined macro keys, auto-dial, printer toggle, display/kill files in the directory, and display/print the buffer contents. In addition to these familiar options, MTERM has other features of which I have not seen on any another modem program.

MTERM allows the user to define a string of characters to be stored in up to ten macro keys. Upon entering a macro key, the string of 1 to 63 characters would be sent to the remote system. This gives the program an auto-login facility. What is unique about the MTERM's macro keys, is the ability to link to one of the other macro keys. This makes it possible to send 612 characters of predefined characters with the touch of one key.

The next feature would best be explained by an example. Suppose you have three or four timeshare services that you access periodically. Each one of these systems will have different entry login requirements, and each may have different system parameters. The MTERM package will save separate "system" files for each service. These files will contain the stored parameters, macro keys, etc. which apply to the respective system. Each file can be invoked from the command line of ZDOS (e.g. MTERM MNET or MTERM SOURCE, for linking to CompuServe and the SOURCE, respectively), or from the command menu in MTERM. This, of course, will eliminate the need to reset the system parameters at each login session.

Another nice feature is a temporary capture buffer. This 2K temporary area allows the user to go into command (menu) mode and return to find that no characters have been lost. At 300 baud the user has approximately 68 seconds before characters will be lost. At 1200 baud there is 17 seconds to jump to the command level, enter an option, and return before losing any characters. I found after about an hour or so that I was familiar enough with the program to leave the conversation mode, enter a command, and be back in just a few seconds.

MTERM will suppress the sending and receiving of a carriage return (0DH) and also a line feed (0AH). Normal operations suppress the line feed (LF) character, which if changed will send a second line feed to the remote system. This option was included for any communication that might be done with teletype style terminals, which need the LF character.

The only problem I encountered with MTERM, I was able to find a "fix" for with the LF suppression toggle. MTERM, as with any modem program, will save all or part of the transmission in a storage buffer which can then be viewed, sent to the printer, or saved to a disk file. The contents of the storage buffer appeared properly on the screen and to the printer, however, I found when I saved the buffer as a disk file, the file was missing the line feed character at the end of each line of text. By toggling the LF suppression to off, MTERM inserted an additional line feed at the end of each line, which caused two line feeds in the display. However, the disk file was saved with a single line feed. I talked with Micro-Systems Software and was informed that this would be fixed on future releases of MTERM.

The most unique and most powerful feature of MTERM is the ability to alter any incoming or outgoing character of the communication session to another character value. This is handled by the use of Translation Tables. There are seven translation tables. Let me explain the effectiveness of each with an example.

The Keyboard Table will let the user assign an ASCII value, which is outside of the range of the standard H/Z-100 keyboard, to one of the keys. This could be done if the remote requires an ASCII character not included on the keyboard.

The Display Table can change incoming characters that affect the screen display. If the remote is sending clear screen codes for XYZ computer, the characters could be changed to reflect the proper sequence of codes for the H/Z-100 terminal.

The Printer Table can change received characters which will affect the printer. This is similiar to the display table, but directly relates to only the printer output. As in the Display Table, when the XYZ computer sends clear screen codes, the printer table can be changed to alter the clear screen characters to send form feeds to the printer. This is done independently of the display table, which is translating the same XYZ clear screen to the H/Z-100 clear screen code.

There are Buffer Input (BI) and Buffer Output (BO) Tables and RS232 Input and Output Tables. These tables are independent of each other. But with these tables, the user can effectively change each received and sent character to any value. The user can feasibly communicate with an IBM computer that uses EBCDIC code. The user would translate incoming EBCDIC characters to ASCII and outgoing ASCII characters to EBCDIC codes.

A modem feature that is becoming more popular is contained in a limited form in MTERM. The program has the ability to call a particular remote system at a preset time, upload the buffer contents, and then exit.

In addition to MTERM, the disk contains a program called XFER. This is a menu select program separate from MTERM, which provides the user with the ability to transfer files directly between two computer stations, error free. XFER will transfer binary files directly provided the other system is using XFER or a program compatible with XFER. When a file is tranferred, XFER checks the data for errors and will re-transmit the sector if an error occurs in transmission.

XFER will translate a binary file to ASCII and an ASCII file to binary. The ASCII file can then be sent using MTERM. MTERM uses XON/XOFF transmission handshaking, while XFER uses ACK/ENQ.

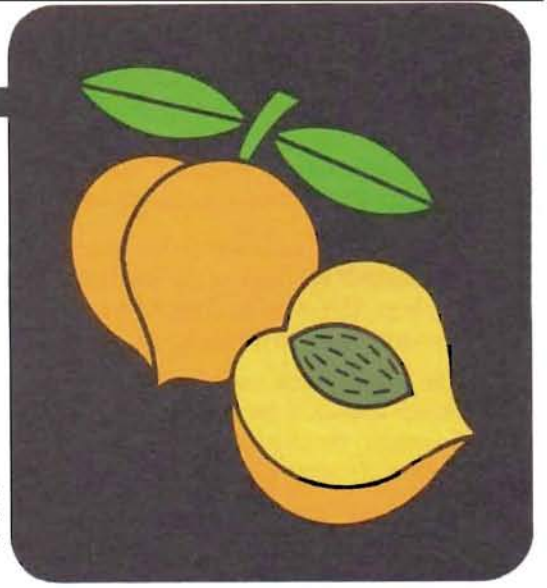
Micro-Systems informed me that they are currently incorporating the Ward Christianson Protocol into MTERM, which will be released soon.

I found MTERM quick to learn and easy to use. MTERM and XFER contain options that are similar to other modem packages. However, the unique features, i.e. the translation tables, will make this package attractive to many ZDOS users.

MTERM and XFER are products of:

Micro-Systems Software Inc.
4301-18 Oak Circle
Boca Raton, FL 33431
(800) 327-8724

The package sells for $79.95. MTERM is available at Heathkit Electronic Centers. MTERM also comes in other disk formats for other computer systems, contact Micro-Systems Software for details.



Part I

PeachText 5000

Your Complete Personal Productivity System

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

Introduction

Why would any HUG member buy an H/Z-100 computer? I will tell you why I did! I have been using an H-8 for five years, H-8/H- 9/Cassette upgraded over the years to H-8/H-19/H-17/H-37,etc. My biggest need for a computer was for a Word Processor, an Accounting System, and an Electronic Spreadsheet.

I am sure that you know that just one "good" software electronic spreadsheet, VisiCalc, sold APPLE computers by the thousands! How many of you HUG members can get along without an electronic spreadsheet on your computer? If you are not using a spreadsheet, it is because you have NEVER learned how to use one in your personal and business life! The most important "tool" for my business is the electronic spreadsheet. However, I found that many of my applications required more then my 64K of RAM memory. I found that my answer was to buy an H/Z-100 with the 16-bit MULTIPLAN spreadsheet. This single "need" was my motivation for buying my H/Z-100 a few months ago. With 192K of RAM memory (Multiplan can use up to 256K of RAM) and the two (2) double sided, double density built-in 5.25 inch floppy disk drives, I can now prepare large spreadsheets and combined spreadsheets to better serve my needs.

Software Review

I feel that I MUST tell you about a software package that I found! A few months after I purchased my H/Z-100, I went to my local Heathkit Center to attend a local HUG meeting. I could not help noticing a flashy Green and White Peachtree display. It was PeachText 5000—Your Complete Personal Productivity System! I am always looking for ways to make my computer more useful and efficient, so I investigated the display. I will never be disappointed! I found that this software package made my investment in the H/Z-100 really worth while! I would like to share my findings with you in detail.

PeachText 5000

PeachText 5000 (by Peachtree Software Inc.) combines Word Processing, Financial Modeling, Mailing List Maintenance, and Basic Database Management functions into one INTERACTIVE SYSTEM. (Yes, Interactive! Meaning each function will communicate and work with all the other functions.)

At the "core" of the system are the following improved Peachtree packages completely menu-driven, with "HELP" prompts:

1. PeachText word processor has long been recognized as MagicWand. Peachtree's latest Version 2.0 is written in Assembler Language producing a fast, efficient program that provides a very fast memory-mapped video display and an interrupt-driven keyboard with a type ahead buffer (all H/Z-100 features) which is an ideal combination for a word processing system. It also makes use of the H/Z-100 RAM and the two (2) high capacity drives.

It is fully menu-driven to access the Random House Electronic Thesaurus with its 4,400 indexed words and 26,000 synonyms. It also gives you instant access from PeachText menu to the Spelling Proofreader with its 21,000 word dictionary. You can add to this dictionary any number of words that your own requirements may need, limited only by your disk storage capacity.

2. PeachCalc turns your computer memory (up to 256K RAM) into a worksheet for financial and other mathematical analysis. Its merge function allows you to insert information from PeachCalc reports in PeachText documents. You can also use PeachText "execute" command to feed data into PeachCalc; again, all menu- driven. PeachCalc is in Assembler Language.

3. The database management function centers around List Manager. This MODULE uses Peachtree's new "Visionary" screen manager and their new index file manager, called "GORF". GORF is being used by Peachtree for the first time in List Manager. Users will find that GORF has no equal, in my opinion, in finding, storing, deleting, and updating the files. GORF also enables List Manager to remember records in your file by several different references (keys) and locate your record even if you don't quite remember how you filed it. They call this feature "browsing", you can start anywhere in the file and go forward or backwards until you find the desired record in your file. All GORF needs is at least one starting character (or more if you would like to narrow down the search) of the record to find the nearest match.

List Manager gives you complete control over the design and use of mailing lists, labels, files, and reference aids such as bibliographies. In conjunction with PeachText (menu-driven), List Manager can individualize your form letters or "template" documents. Any file can be sorted several different ways and can contain as many as 32,765 records. Each record can have up to 14 line items of information (Record total up to 509 characters).

List Manager is written in compiled Microsoft BASIC. The BASIC Run Time Library is supplied with the package. You do not need to have Microsoft Compiled BASIC software.

If any of you HUG members are using or attempting to use software, such as Ashton Tate's DB-II, you could find that easy- to-use List Manager could accomplish your requirements better and save you a lot of time and money.

This PeachText 5000 package is available from Zenith Data Systems, Heathkit Centers, and Heath's Catalog for $395.00. Verifiable H-100 owners can buy the package from Heath for $275.00. If you are buying an H-100, you can purchase Z-DOS and PeachText 5000 for $325.00. This software package is really a BEST BUY! However, you also get more. You will receive a box of 10 Peachtree/Wabash 5.25 inch diskettes with the package and free 90 days of support directly from Peachtree Software Inc. on return of the registration card included in the package.

The "clincher" that really sold me PeachText 5000 was the COUPON which allows you to purchase Peachtree's "Access Pak" for only $10.00 by direct mail. This Access Pak has a retail value of $525.00. It includes Peachtree's "Information Access" program. This program allows you to extract information from Peachtree's Accounting Software for use with PeachText and PeachCalc. Access Pak also includes programs that make any existing WordStar, EasyWriter, and VisiCalc files compatible with the PeachText 5000 package. This really completes an easy to use H/Z-100 "Interactive" software package that any HUGger could really use.

The concept of TOTAL INTEGRATION has provided us the first package, in my mind, where all of the software programs function together as one using a single menu-driven MODULE that walks the new user through its operations, and yet is powerful enough to suit experienced programmers doing Custom Work. Best of all, it works with the H/Z-100 like it was custom designed just for it! I found myself using the package productively in just a few hours. Also, you do not have to buy any other application software to improve it. You do need the Z-DOS operating system. The package requires no previous experience with automated systems and the well written documentation quickly introduces the user to this flexible and comprehensive package.

The documentation consists of two (2) manuals:

1. Training Manual with pictorial screens and understandable English step-by-step training instructions with plenty of examples (except for PeachCalc).

2. Reference Manual with expanded information on the commands and technical aspects of the system. The user can answer any question by simply consulting the manual. (An index is the only missing item and badly needed.)

One of the important innovations that Peachtree supplied with PeachText 5000 is an interface program (called a PSI emulator) which allows the package to be compatible with most 16-bit computer systems if their keyboards use arrow keys for cursor movement, insert and delete keys for characters, and have at least 10 special function keys. Doesn't this describe an H/Z- 100? The interface can also accommodate most printers, including dot matrix printers with the same "basic" functions as incremental space printers.

The user configures their H/Z-100 with the package's "Configuration" programs. These programs actually prompt the user for the information they need to get the system running. The system is really what I call "user friendly". If the user makes a mistake, the program asks for specific information that will remedy the problem. That is a big improvement over giving the user a choice of arbitrary numbers. You do not need to keep returning to the main program menu to correct mistakes. For example, if you make a selection that is not on the disk you are using, the program will simply tell you it is not on the valid disk. Users do not need to become involved with the Z-DOS operating system to configure their system either. The screens are memory-mapped so that the programs can go from one screen to the next rapidly and without problems. The Configuration process takes only a few minutes.

Two of the most difficult tasks for new users are learning the keyboard (what functions are performed by which keys) and learning to operate the software by wading through instruction manuals. Peachtree has improved the application of the software by using the H/Z-100 function keys heavily. This makes things easier for the user. For example, the function key F1 in PeachText means "Backward Page Scroll" and function key F1 in List Manager means "Prior Field". Also, function key F2 in PeachText means "Forward Page Scroll" and function key F2 in List Manager means "Next Field". This means that the same function key performs similar tasks reducing the new things to learn.

I believe we are fortunate to have PeachText 5000 to support the H/Z-100. I know I sound like a vendor for Peachtree, but I am not! I am only anxious to share my enthusiasm with fellow HUG members so that they may find more useful applications for their computering. In my mind, this PeachText 5000 is so valuable that it made the H/Z-100 investment worth while. I believe any serious computer user would find this to be true.

Closing

I believe that this PeachText 5000 package requires additional review in detail. The two new (List Manager and Information Access) modules are of sufficient importance to require a separate article! I will be reviewing these in the next article.

A New Column

William M. Adney
4821 Sunnybrook
Buena Park, CA 90621

How is the best way to introduce a new column? It has turned out to be a little more difficult than I thought, but I think that the most important part of the introduction is to tell you the purpose of the column. I plan to look at various pieces of hardware and software plus occasionally discuss current trends in the microcomputer world. Some of the currently planned items include a look at memory disk drives, implementing terminal features with assembly language (with programs), a slick way to manage disk files, review of various software (word processors, spreadsheets, data bases, and utilities), and a review of letter quality printers (Transtar 130). Telecommunications hardware and software (modems,etc.) won't be neglected, and of course we'll look at new Heath/Zenith products such as the ZS-217 Winchester drive for the H/Z-100. As with all articles in this magazine, each article will obviously be slanted toward all Heath/Zenith users, and hopefully provide some help with your home and business computing questions.

Since no column is really complete without some thoughts and ideas from you, I am looking forward to your comments on each one.

My Background

Since no introduction would be complete without including some of my background, I will take a little time out for that.

My professional background includes 15 years in data processing working with large IBM systems. I am currently the Manager of Security/Contingency Programs for a large company. My primary function includes physical and computer security of payroll/personnel data and disaster recovery planning for our organization. Job functions have included systems design, analysis, programming (primarily in COBOL), data base technology, security, disaster recovery, and documentation for all of the above. In general, most of my career has included working for large corporations like McDonnell Douglas and Transamerica. For obvious reasons, I have also worked with microcomputers in the business world.

So much for the business side...what about hobbies? Well, my first real hobby was ham radio where I started out as WA9AND. That was back in the days of the Heath DX-20 which was a terrific novice transmitter, so I've been building Heathkits for quite a while. I still have a ham license, but I haven't been too active since I began working with the Heath microcomputers.

I began with the H-89 with 64K of memory, (2) Z-37 drives, and an H-25 printer. Although I still have the H-25 printer, I now have the H-100 with 192K of memory and 2 disk drives. My primary word processor is Magic Wand (also known as PeachText 2.01 now) which I think is one of the best word processors on the market. In fact, I like it so much that I also have the PeachText 5000 package which is really great!

Why Choose Heath/Zenith?

In my opinion, one of the best kept secrets in the microcomputer world today is that the Heath/Zenith family of hardware and software is among the finest available. And I have found that the support in 3 of the local stores is also great...even for non-H/Z products. But why choose Heath/Zenith in the first place?

One of the most important things in purchasing any computer is the after sales support. Since I have spent a few million dollars of company money on computer hardware and software, I spent some time trying to justify my personal microcomputer since I was going to pay for that. As part of that justification, I spent considerable time in various computer stores just trying to understand all of the available hardware and software. In most cases, I came away very frustrated because, although I knew what I wanted, I couldn't find a salesman

in most stores who would take the time to discuss the advantages and disadvantages of their systems. Many of the salesmen simply ignored me when they found out that I wasn't going to buy a computer system "today". I also found that I asked too many questions that they couldn't answer. I couldn't help but wonder if I would really be out of luck when I had a problem after the sale!

Since I was already prejudiced toward the purchase of Heathkit, I went there last. I found that John Secor of the Anaheim store was willing and able to intelligently discuss the pros and cons of the various equipment on the market. In fact, he spent about 3 hours with me that first day, and didn't seem offended when I didn't buy anything at that point. Naturally I purchased the Heathkit equipment (and software) since I was convinced that I would be able to get support after the sale.

And I haven't been disappointed either! It seems to be an unfortunate fact of life that computers and software don't always work the way you expect. I've had some problems, mostly because of something that I did wrong, but I've always received courteous and prompt assistance in solving the problem, even if it was my fault. That is probably the most important reason for buying your equipment from a reputable supplier.

And now it's time for a little criticism. I have not been able to understand why Heath/Zenith does not promote their microcomputers more aggressively. As a manufacturer of top quality equipment and supplier of excellent software, there sure aren't many ads promoting that. The "I built it myself and saved money" ad really isn't too impressive. How about something like "Our equipment is better than ...and it comes with a dual processor...and it runs both CP/M and most IBM software at no additional cost for hardware...and it has a better keyboard...etc".

Current Trends

One of the more interesting subjects today is the answer to the question of: "Where are we going in data processing?". I specifically used the term "data processing" instead of microcomputers or computers because they are getting more difficult to separate every day. Ads discussing the interconnection of personal computers to large systems and local area networks are becoming more common.

Microcomputers will continue to be faster, smaller, and more powerful at less cost. Large computer system architecture will provide better interfaces for personal computing networks. Microcomputer hardware and software for telecommunicatons will be faster...at least on the order of 4800 baud which is the upper limit for a standard phone line. Some popular add-on hardware will help you obtain these capabilities, and I'll talk about that in a future article.

Software is still the weakest link. And that means both programs and documentation. Because of the rapid growth of micros, a lot of the software, particularly the documentation, is terrible! We'll be looking at some of the better software in future issues, and what "user friendly" really means.

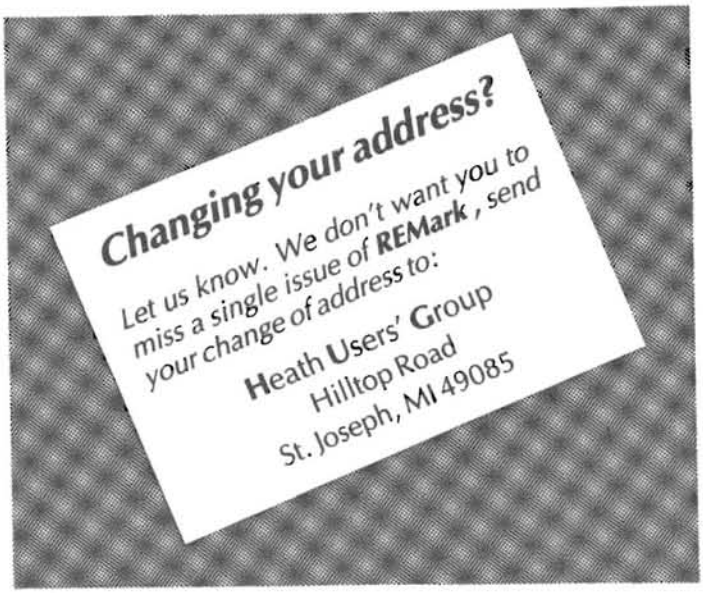
Microcomputers which are "versatile" will be the biggest sellers in the future. Dedicated word processors will be non-existent simply because they're too limited.

Those are just a few of the current trends that I have noticed. We'll continue to look at those from time to time in the future.

Next Month

With the introduction of the new H/Z-100, I wondered what the cost differences were between buying the new system and upgrading the H-89 with the boards from D-G Electronics, Magnolia Microsystems, and Technical Micro Systems Inc. (TMSI). I'll talk about the surprising results of that next month.

As a final note for those of you who read Jerry Pournelle's comments in the June 1983 issue of Byte, we'll provide the answer to his problem of turning off the H/Z-100 "keyclick" in a future column.



H19/H89

THE ULTRA ROM

ULTRA is an enhanced ROM for your H19 terminal display which retains compatibility with present software (except ANSI sequences), yet adds new capabilities that you probably thought were impossible: self-test, second page display (with option), operation to 38.4K baud, user function keys, and much, much more. Comes with a pair of ROM's, binder, and complete documentation.

ULTRA $49.95
Page Two Option $14.95

H89/H100

ZLYNK/II

ZLYNK/II is a CP/M or CP/M-85 modem system like nothing else in the entire industry!ZLYNK/II actually contains its own text editor and language (ZLINGO). It can be used directly by the operator, or else programmed to operate completely independently. Eight protocols are supplied with ZLYNK/II, but more can be added via overlays at any time.

ZLYNK/II (hard sector) $59.95
ZLYNK/II (soft sector) $59.95

Something for Everyone

H100

PALETTE

An interactive graphics system for artists and home users alike. With PALETTE you can draw superb color or monochrome graphics displays and even dump to compatible printers. PALETTE currently supports both a lightpen and interlace mode, and it comes with a complete tutorial! NOT to be confused with others that SEEM to be similar. PALETTE is truly one-of-a-kind!

PALETTE (ZDOS) $69.95
Z100 Lightpen $150.00

H8/H89

THE HDOS TOOLKIT

The HDOS Toolkit is a group of utilities for both the comfort of the average user, and for the education of developing assembly-language programmers. We supply full source code with the TOOLKIT with comments to unlock truly 'Wizard-Class' tricks. Any single program in the package is worth the price of the entire group. If you like HDOS, you'll LOVE this!

Toolkit (hard sector) $29.95
Toolkit (soft sector) $29.95

All of the above items (and a LOT more!) are available direct from Software Wizardry, Inc. as well as from most Heathkit Electronic Centers and many Zenith Data Systems dealers and distributors as well. Please add $2 minimum (or 2%, whichever is greater) shipping/handling plus sales tax if shipped to a Missouri address. Software Wizardry, Inc. also has over 450 other items of interest to Heath/Zenith users as well. Call or write for our FREE price list. Dealer inquiries and HUG group purchases welcome!

VISA MasterCard

ZENITH data systems
AUTHORIZED SALES & SERVICE

Software Wizardry
THE MAGIC TOUCH

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

Getting Started With Assembly Language

(or A Real Program, But What Good Is It?)

Pat Swayne
Software Engineer

(Let's see, Walt's deadline was yesterday, so I guess I better get this thing done.) Last month in my continuing discussion of assembly language, I presented an introduction to reading disk files in HDOS. The program I used was one from a HUG disk, which was an alternate to the TYPE command for looking at files. I wrote it because in HDOS, when you TYPE a file, the whole thing (or a large chunk) is loaded into memory and then blasted onto the screen so fast (especially at the 19200 baud rate that I use) that I sometimes miss things even with my finger poised over Control-S. The VIEW program printed last month, however, reads only one sector at a time and displays it on the screen, resulting is a slower display rate. For this month's segment on reading files in CP/M, I will use a CP/M version of VIEW that, although it is a good learning program, serves no useful purpose since the TYPE command in CP/M already works with one record (CP/M logical sector) at a time. Have patience, CP/M users, and we will eventually get to a useful program.

Now, if you think that reading disk files is a bit advanced for a tutorial article on assembly language, maybe you missed the first 6 articles in this series. If so, go way back to the April issue (#39) and get caught up!

Part VII — Reading Disk Files in CP/M

As I stated last month, there are two basic ways to read information from a disk file. The first, and easiest, is to read the file in segments that are the smallest chunks of data that the operating system can handle at a time. In CP/M, those segments are called "records", and are 128 bytes big. The other method is to read the whole file, or as much as will fit in memory, in one gulp. It's not too hard to do in HDOS, but in CP/M, the system can only read one record at a time, so "gulp" reading has to be handled by the programmer. In this article, we will cover single record reading for now, and discuss reading larger blocks of data later.

The example program, VIEW, that is listed following this article can be approximated by this BASIC program.

```
10 LINE INPUT "ENTER FILE NAME: ";N$
20 OPEN "I",1,N$
30 FOR I=1 TO 2 STEP 0: REM  INFINITE LOOP
40 IF EOF(1) THEN 80: REM  END OF FILE
50 LINE INPUT #1,L$: REM  INPUT A LINE
60 PRINT L$: REM  PRINT THE LINE
70 NEXT I
80 CLOSE #1
```

This program is not exactly like the assembly language program because it reads and displays the file one line at a time, while the assembly program reads and displays it one record at a time. Each record may contain several lines or only part of one line. Also, our CP/M VIEW program, like the HDOS one, does not prompt the user for a file name, because it gets it from the command line.

Most operating systems, including HDOS and CP/M, have ways of passing information from the command line to the program that is being run. When you enter a program name at the system prompt, at least one space, and some additional characters, the additional characters are passed to the program. In CP/M, they are stored in two reserved areas in memory that are below the Transient Program Area (TPA), or program starting point. In one area, starting at 81H, they are stored unchanged, with a character count at 80H. In the other area, starting at 5CH, the characters are assumed to be a file name, and are arranged into a format called the File Control Block format. If there is a space and a second group of characters in the command line, they are assumed to be a second file name, and are stored in FCB format at 6CH. Those areas of memory are called the Default File Control Blocks. If there is no command line information, the character count at 80H will be zero, so it can be used as a test for the presence of command line information. You can also use the second character of the Default FCB (5DH), which will be a space (20H) if there is no information.

Before a file can be opened in CP/M, the file name must be in FCB format. Since we will be using the command line, I will not discuss the FCB further now, but will cover it later. It's fairly complicated compared to the way HDOS handles file names.

In the VIEW program, we test for command line information by examining location 5DH (DFCB+1). If no command line information was entered, the program prints some instructions and exits. If a filename was provided, the program attempts to open the file so that it can be read. The BDOS OPEN function (function 15) requires that the DE register pair point to the File Control Block. Since we started with DE pointing to DFCB+1, we just have to back it up one before the file is opened. Notice that DE is stored on the stack. The FCB address will be used for all file operations in the program, and the stack is a handy place to save things. One nasty thing about CP/M is that we are not told which registers are destroyed by the system functions in any of the documentation, so to play safe, save any register that you might need later. HDOS usage of registers is spelled out plainly in the System Programmers Guide.

If the attempt to open the file is unsuccessful, CP/M will return 0FFH in the A register. We test for it by incrementing A, which will set the zero flag if the value was 0FFH. View assumes that the file does not exist if the attempt to open it is not successful, and prints an error message saying that. Since there are no HDOS-style built-in error messages in CP/M, you will have to provide your own for such situations as file opening attempts.

After the file is opened, the program enters a loop where it reads a sector and then prints the information on the screen. As with the open function, the READ function (BDOS function 20) requires that DE point to the File Control Block, so we get DE from the stack and save it again. When you read from a file in CP/M, the data goes into a place in memory called the DMA area. DMA originally stood for Direct Memory Access, but that name is meaningless here because the disk controllers in Heath/Zenith computers do not put informa-

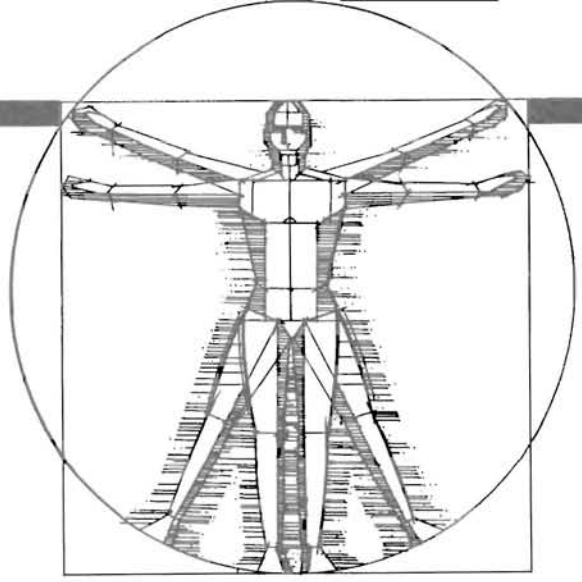
tion directly into memory. They are accessed via ports, and the BIOS puts the information into memory. Anyhow, the DMA is by default at address 80H (Where did we see that before?), and unless you change it with another BDOS function, the 128 bytes of data read from the file by each READ operation will go there. So, after each use of the READ function, 128 bytes of information starting at 80H are printed on the screen using the BDOS CONOUT function. If a Control-Z character (1AH) is found, it signals the end of the file, and so the program exits.

The READ function will return zero in the A register if the read operation is successful. About the only thing that an unsuccessful read can mean in CP/M is that the end of the file has been reached, so the program exits if the read is not successful. Any other error would probably result in a BDOS error message, and an abrupt cancellation of the program by the operating system. Not too nice, compared to HDOS.

While the program prints the information in the DMA area, it checks the console status with BDOS function 11 (CONST) to see if the user has typed any key, and if he has, the program exits. Actually, not every key is checked by CONST, because you can still use Control-S to pause the listing. CP/M allows any key to be used to resume a listing paused this way (HDOS only allows Control-Q), but you should use Control-S again, because with any other key, if the key "bounces" or REPEAT is accidentally pressed, the program will quit early. Many CP/M programs use a console status check to allow an early exit. Notice that the program reads the abort key itself with the CONIN function before exiting to prevent the character from being passed on to CP/M.

When the end of the file is found either by encountering a Control-Z or an end indication from the BDOS, the program exits back to CP/M using a method we have discussed before. In the next two installments of this series, I will cover HDOS and CP/M versions of a file copying program. With it, we will explore reading and writing large blocks of data and constructing an FCB in CP/M. Don't forget to "tune in"!

```
;       VIEW -- VIEW FILES ON SCREEN
;
;       TO USE THIS PROGRAM, ENTER
;
;       A>VIEW d:FILENAME.TYP
;
;       THE FILE WILL BE LISTED ON THE SCREEN.
;
;       BY P. SWAYNE, HUG  6-OCT-83

;       CP/M SYSTEM CALLS, ETC.

CONIN   EQU     1               ;CONSOLE INPUT
CONOUT  EQU     2               ;CONSOLE OUTPUT
PRINT   EQU     9               ;PRINT STRING
CONST   EQU     11              ;CHECK CONSOLE STATUS
OPEN    EQU     15              ;OPEN FILE
READ    EQU     20              ;READ FILE (SEQUENTIAL)
BDOS    EQU     5               ;BDOS VECTOR
DFCB    EQU     5CH             ;DEFAULT FCB
DMA     EQU     80H             ;DMA AREA

        ORG     100H            ;ALWAYS START HERE

START   LXI     H,0
        DAD     SP              ;LOCATE STACK
        LXI     SP,STACK        ;SET OUR OWN
        PUSH    H               ;SAVE CP/M'S STACK
        LXI     D,DFCB+1        ;POINT TO FCB SECOND CHAR
        LDAX    D               ;GET IT
        CPI     ' '             ;SPACE?
        JNZ     GOTFILE         ;NO, WE HAVE A FILE
        LXI     D,NOFILE
        MVI     C,PRINT
        CALL    BDOS            ;PRINT NO FILE MESSAGE
        JMP     EXIT1           ;RETURN TO CP/M

GOTFILE DCX     D               ;BACK UP TO FCB START
        PUSH    D               ;SAVE THIS ADDRESS
        MVI     C,OPEN
        CALL    BDOS            ;TRY TO OPEN FILE
        INR     A               ;TEST RESULT
        JNZ     RLOOP           ;GOOD OPEN
        LXI     D,NOOPEN
        MVI     C,PRINT
        CALL    BDOS            ;SAY CAN'T OPEN FILE
        JMP     EXIT            ;RETURN TO CP/M
RLOOP   POP     D               ;GET FCB ADDRESS
        PUSH    D               ;SAVE IT AGAIN
        MVI     C,READ
        CALL    BDOS            ;TRY TO READ A RECORD
        ORA     A               ;READ OK?
        JNZ     EXIT            ;NO, END OF FILE
        LXI     H,DMA           ;ELSE, POINT TO DMA AREA
        MVI     B,128           ;PRINT 128 CHARACTERS
PLOOP   MOV     A,M             ;GET A CHARACTER
        CPI     'Z'-40H         ;CONTROL-Z?
        JZ      EXIT            ;YES, END
        MOV     E,A             ;ELSE PUT CHARACTER IN E
        PUSH    H               ;SAVE POINTER
        PUSH    B               ;SAVE COUNTER
        MVI     C,CONOUT
        CALL    BDOS            ;PUT CHARACTER ON SCREEN
        MVI     C,CONST
        CALL    BDOS            ;CHECK CONSOLE STATUS
        POP     B               ;GET COUNTER
        POP     H               ;GET POINTER
        ORA     A               ;ANY KEY HIT?
        JNZ     CEXIT           ;YES, EXIT
        INX     H               ;ELSE, INCREMENT POINTER
        DCR     B               ;DECREMENT COUNTER
        JNZ     PLOOP           ;LOOP UNTIL 128 CHARS PRINTED
        JMP     RLOOP
CEXIT   MVI     C,CONIN
        CALL    BDOS            ;ABSORB ABORT CHARACTER
EXIT    POP     D               ;FIX STACK
EXIT1   POP     H               ;GET CP/M STACK
        SPHL                    ;SET IT
        RET                     ;RETURN TO CP/M

;       MESSAGES AND STACK AREA

NOFILE  DB      13,10,'The correct use of this program is'
        DB      13,10,10,'A>d:VIEW d:FILENAME.TYP',13,10,10
        DB      'where d: is a drive designation '
        DB      '(A:, B:, etc.),',13,10
        DB      'and FILENAME.TYP is the file to be VIEWed.'
        DB      13,10,'$'
NOOPEN  DB      13,10,'ERROR -- File not found.',13,10,'$'

        DS      32              ;SPACE FOR STACK
STACK   EQU     $               ;PUT OUR STACK HERE
        END     START
```



Standards For Terminal Control

Plotting Graphs On Your Screen and Printer

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

Well, here we are, six months into the BASIC COMPUTING column and the correspondence to date, both about this column and my prior articles, indicates a need and a desire on the part of HUG members to standardize the string names given to the terminal control sequences we use. Why should one programmer erase the screen with E2$, another with ES$ (Erase Screen), and yet another with CS$ (Clear Screen)? Many members suggested that the string name be an abbreviation of the function performed. That's an excellent approach, and the one I'll use in our first program module. While different versions of BASIC allow different sizes of variable names, I've kept these to two letters to assure compatibility with MBASIC (HDOS) and MBASIC (CP/M). Those who are using Benton Harbor BASIC should continue to use the variables as presented in the program module CONTROL.BAS. Two-letter variables are not allowed in Benton Harbor BASIC.

The following program should be called CONTROL2.BAS and saved in the ASCII format so it can be merged later. If your version of BASIC doesn't allow MERGE or CHAIN commands, you can save the program, then just load it and add program lines later.

```
2 REM ***CONTROL2.BAS
4 REM ***DAVID E. WARNICK
6 REM ***COPYRIGHT 1983 BY APPLIED COMPUTING
10 E$=CHR$(27)          'ESCAPE
20 CH$=E$+"H"           'CURSOR HOME
30 CF$=E$+"C"           'CURSOR FORWARD
40 CB$=E$+"D"           'CURSOR BACKWARD
50 CU$=E$+"B"           'CURSOR UP
60 CD$=E$+"A"           'CURSOR DOWN
70 CP$=E$+"n"           'CURSOR POSITION REPORT
80 CS$=E$+"j"           'CURSOR POSITION SAVED
90 CR$=E$+"k"           'CURSOR RETURN TO POSITION SAVED
100 CA$=E$+"Y"          'CURSOR ADDRESSED DIRECTLY
110 ED$=E$+"E"          'ERASE DISPLAY
120 EB$=E$+"b"          'ERASE BEGINNING OF DISPLAY
130 EE$=E$+"J"          'ERASE END OF PAGE
140 EL$=E$+"l"          'ERASE LINE
150 LB$=E$+"o"          'LINE BEGINNING ERASE
160 LE$=E$+"K"          'LINE END ERASE
170 LI$=E$+"L"          'LINE INSERT
180 LD$=E$+"M"          'LINE DELETE
190 CG$=E$+"N"          'CHARACTER GONE (DELETED)
200 CI$=E$+"@"          'CHARACTER INSERT
210 CE$=E$+"O"          'CHARACTER EXIT INSERT MODE
220 LX$=E$+"x1"         'LINE XTRA (25TH) ON
230 LO$=E$+"y1"         'LINE OFF (25TH)
240 KO$=E$+"x2"         'KEY CLICK OFF
250 KC$=E$+"y2"         'KEY CLICK ON
260 SH$=E$+"["          'SCREEN HOLD MODE ENTER
270 SS$=E$+"\"          'SCREEN SCROLL (EXIT HOLD MODE)
280 CW$=E$+"x4"         'CURSOR WHOLE (BLOCK)
290 CL$=E$+"y4"         'CURSOR LINE (UNDERSCORE)
300 CX$=E$+"x5"         'CURSOR OFF
310 CO$=E$+"y5"         'CURSOR ON
320 KS$=E$+"t"          'KEYPAD SHIFTED
330 KU$=E$+"u"          'KEYPAD UNSHIFTED
340 KA$=E$+"="          'KEYPAD ALTERNATE MODE
350 KX$=E$+">"          'KEYPAD EXIT ALTERNATE MODE
360 LF$=E$+"x8"         'LINE FEED ON CARRIAGE RETURN
370 LN$=E$+"y8"         'LINE NO-FEED ON CARRIAGE RETURN
380 RA$=E$+"x9"         'RETURN AUTO ON LINE FEED
390 RN$=E$+"y9"         'RETURN NOT-AUTO ON LINE FEED
400 VR$=E$+"p"          'VIDEO REVERSED
410 VN$=E$+"q"          'VIDEO NORMAL
420 GM$=E$+"F"          'GRAPHICS MODE ON
430 GO$=E$+"G"          'GRAPHICS OFF
```

I've attempted to give meaning to the first letter of the variable such as C for cursor, L for line, S for screen, etc. The second letter gives the action on the first (i.e. U for up). I also propose that we standardize certain line numbers as follows:

```
1-9         PROGRAM NAME, AUTHOR, DATE, ETC.
10-499      DEFINING STANDARD VARIABLES, USER FUNCTIONS, ETC.
500-999     CONDITION TERMINAL, SET UP DISPLAY SCREEN
1000-       MAIN PROGRAM
```

Why all the fuss about standard codes, and even further, standardizing the line numbers within a program? Heath and Zenith owners have been at the forefront of the computer revolution ever since the introduction of the first H8. While other companies may have done a better marketing job, we set many of the standards to be used. The introduction of CP/M and its acceptance by other brands gets us closer to sharing programs with their owners. If a variable name means the same thing in two programs (i.e. CU$=Cursor Up), it merely needs correct definition for the system it's being used on. Programs in MBASIC CP/M could be shared among brands by sharing lines 1000-END and supplying your own lines 10-999, provided each system uses similar CPU's and compatibly smart terminals. Your modem could do even more for you because instead of 17,000 HUG members to share your programs and ideas, your horizons would be greatly expanded.

There are bound to be a lot of arguments for differences in the standards I've proposed, but bear in mind, that's what has prevented establishing a standard in the past. So there it is, I've stuck my neck out and proposed a standard for programming. Let's hear what you think on the subject.

Now for the subject I planned to discuss this month, graph plotting. If the output showing the results of the calculations or data processing your program performed doesn't give you a picture, you may need to plot a graph. Remember the saying "A picture is worth a thousand words"? Well, it's especially true here. A table of output values may present all the facts, but a graph draws a picture.

The first thing we'll do is plot a graph (I'll use a sine wave) on the CRT. Then we'll do the same thing with our printer. Finally, I'll present a program which calculates three curves and plots them all on one graph on the printer.

Only three steps are required to plot a graph. They are:

1) Lay out the graph
2) Calculate the values to be shown
3) Plot the values on the graph

We'll use a sub-program at line 1000 to lay out the graph, then add titles and scales beginning at line 1500. Notice we're still dividing our programs into modules. To change titles and scales on our graph, just replace lines 1500-1580.

In laying out graphics in the past we've used the format:

```
PRINT CA$;CHR$(Y);CHR$(X);
```

We can save program space here or with any function we repeat several times in a program by defining a user function to do the job. Refer to your MBASIC manual for the DEF FN (DEFine FuNction) command. We'll go to line 440 and define a direct cursor addressing function like this.

```
440 DEF FNCA$(Y,X)=CA$+CHR$(31+Y)+CHR$(31+X)
```

Now all we have to do is PRINT FNCA$ and specify the values for Y and X. We even got the user function to add 31 to each for us, so we can specify them as the line number and position on that line the cursor should go to. First we'll condition our terminal.

```
450 PRINT  FNCA$(1,1)      'ALWAYS MOVE CURSOR OFF BOTTOM
                             BEFORE HOLD SCREEN IS ENTERED
500 PRINT GM$              'GRAPHICS MODE
510 PRINT SH$              'SCREEN HOLD MODE
520 PRINT CX$              'CURSOR OFF
530 PRINT ED$              'ERASE DISPLAY
```

Now print the blank graph.

```
1000 X=6                   'COLUMN 6
1010 FOR Y=2 TO 22         'FOR LINE 2 TO 22
1020 PRINT FNCA$(Y,X);"|"'PRINT GRAPHIC CHARACTER
1030 NEXT Y                'CONTINUE
1040 Y=12                  'LINE 12
1050 X=7                   'COLUMN 7
1060 PRINT FNCA$(Y,X);STRING$(61,97)     'PRINT THE LINE
```

Line 1060 slipped in a new function. Check it out in your MBASIC manual. Any time you want to repeat the same character several times, you can do it with STRING$. The first number in the parenthesis is the number of times to repeat the character (we wanted 61), and the second is the ASCII value (decimal) of the character to be printed. We could also use variables here as shown in your manual. Now let's label our graph for the sine wave.

```
1500 PRINT GO$                'TURN GRAPHICS OFF
1510 PRINT FNCA$(2,3);"+1"    'LINE 2, COLUMN 3 "+1"
1520 PRINT FNCA$(12,4);"0"    'LINE 12, COLUMN 4 "0"
1530 PRINT FNCA$(22,3);"-1"   'LINE 22, COLUMN 3 "-1"
1540 PRINT FNCA$(23,7);"0"    'LINE 23, COLUMN 7 "0"
1550 PRINT FNCA$(23,21);"90"  'LINE 23, COLUMN 21 "90"
1560 PRINT FNCA$(23,35);"180" 'LINE 23, COLUMN 35 "180"
1570 PRINT FNCA$(23,50);"270" 'LINE 23, COLUMN 50 "270"
1580 PRINT FNCA$(23,65);"360" 'LINE 23, COLUMN 65 "360"
```

Now for the body of our graph program, we'll begin at line 2000.

```
2000 FOR X=7 TO 68            'HORIZONTAL POSITION ON GRAPH
2010 A=6*(X-7)                'CALCULATE ANGLE SHOWN AT POSITION
2020 Y1=SIN(A/57.3)           'CALCULATE SINE OF ANGLE IN RADIANS
2030 Y2=INT(10*Y1)            'EXPAND VALUE TO FILL OUR SCALE
2040 Y=12-Y2                  'CENTER CURVE VERTICALLY ON GRAPH
2050 PRINT FNCA$(Y,X);"*"     'PLOT THE POINT ON GRAPH
2060 NEXT X                   'CONTINUE TO END
2070 FOR X=1 TO 1000:NEXT X   'DELAY TO READ GRAPH
2080 PRINT ED$                'ERASE THE DISPLAY
2090 PRINT SS$                'RETURN SCREEN TO SCROLL MODE
2100 PRINT CO$                'TURN THE CURSOR ON
2110 END
```

Now turn on your computer, call up MBASIC, and type in lines 440, 450, 500-520, 1000-1060, 1500-1580, and 2000-2110. MERGE CONTROL2.BAS above and RUN the program. You're now ready to apply what you've learned to your own programs. Remember you've got to keep the values of X and Y within the limits of the screen. We multiplied by 10 to expand our curve to fill out the scale. Then we offset the vertical value by 12 because that's where the center of our scale was. Depending on what you've calculated, you may have to expand or shrink your curve. Experiment and have fun.

Now let's do the same thing with our printer. There are a few differences here. First, not all printers will run forward and backward allowing you to print just anywhere at any time. Second, while we normally move across the screen of the CRT, we move down a sheet of paper with graphs. Third, if we're to print two items on the same line, the one to the left must be printed first. The programs presented here are written for the H-14 printer and should run on most printers.

```
2000 LPRINT TAB(10)"-1"TAB(30)"0"TAB(50)"+1" 'PRINT  GRAPH  VALUES
2010 LPRINT TAB(10)STRING$(42,95)       'PRINT GRAPH BASELINE
2020 FOR X=0 TO 360 STEP 6              'FOR ANGLES USED
2030 Y=SIN(X/57.3)                      'SINE OF ANGLE
2040 Y1=INT(20*Y)                        'EXPAND CURVE  TO  FILL
GRAPH
2050 Y=30+Y1                            'CENTER CURVE ON GRAPH
2060 LPRINT X;                          'SHOW ANGLE UNDER GRAPH
2070  IF  Y2<30 THEN LPRINT TAB(Y2)"*"TAB(30)"|"     'PLOT  CURVE
THEN BASELINE
2080  IF Y2>30 THEN LPRINT TAB(30)"|"TAB(Y2)"*"    'PLOT BASELINE
THEN CURVE
2090 IF Y2=30 THEN LPRINT TAB(30)"*"    'PLOT CURVE ON BASELINE
2100 NEXT X                             'GO TO NEXT ANGLE
2110 END
```

This is a short program and it can stand alone, so type lines 2000-2110 and save it as any file name you like. Make sure your printer is turned on and run the program.

The last program we'll write this month will calculate three different curves and plot them on the printer. We'll use the old standby Biorhythm chart as it's something we can have fun with as we're learning to program. I won't explain the biorhythm calculations other than to say that each curve cycles once every 23, 28, or 33 days from birth. Each is a sine wave.

With three curves, our computers must determine the relative value of each and print them in order. We'll also send a form- feed instruction to the printer so it advances the page when we want.

```
440   DEF  FNCA$(Y,X)=CA$+CHR$(31+Y)+CHR$(31+X)  'DEFINE   DIRECT
CURSOR ADDRESS FUNCTION
450  PRINT  FNCA$(1,1)     'FORCE CURSOR OFF BOTTOM  LINE  BEFORE
ENTERING HOLD-SCREEN MODE
500 PRINT SH$        'ENTER HOLD-SCREEN MODE
510 PRINT CX$        'TURN CURSOR OFF
520 PRINT ED$        'ERASE THE DISPLAY
1000 PRINT FNCA$(6,10)"GIVE ME THE NAME OF THE PERSON FOR WHOM";
1010 PRINT " I'M TO MAKE THIS CHART"
1020 PRINT FNCA$(8,10);:LINE INPUT;"",N$
1030 PRINT FNCA$(10,10)"WHAT IS THE PERSONS BIRTH DATE?"
```

```
1040  PRINT  FNCA$(12,10)"PLEASE USE ONLY NUMBERS AND  COMMAS  IN  THE";
1050 PRINT " FORMAT-MONTH,DAY,YEAR."
1060 PRINT FNCA$(14,10)"(EXAMPLE 4,26,44)"
1070 PRINT FNCA$(16,10);:INPUT;"",M,D,Y
1080 PRINT FNCA$(18,10)"ON WHAT DATE SHALL I START THIS CHART?"
1090  PRINT  FNCA$(20,10)"USE THE SAME FORMAT AS  FOR  THE  BIRTH  DATE."
1100 PRINT FNCA$(22,10);:INPUT;"",M1,D1,Y1
```

That completes our input. Now we'll compute the number of days from birth to the date the chart begins.

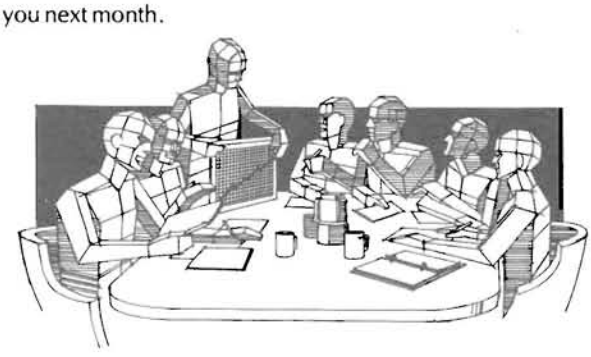
```
2000 DT=0:D3=0:D4=0:D5=0       'SET VARIABLES TO 0
2010 REM CALCULATE TOTAL DAYS IN WHOLE YEARS
2020 FOR X=Y+1 TO Y1-1         'FOR EACH WHOLE YEAR SINCE BIRTH
2030  IF (INT(X/4))*4=X THEN D2=366 ELSE D2=365     'TEST FOR LEAP YEAR
2040 DT=DT+D2        'SUM TOTAL DAYS
2050 NEXT X          'CONTINUE FOR EACH WHOLE YEAR
2500 REM ADD DAYS IN THE FIRST YEARS'WHOLE MONTHS
2510 ON M GOTO 2520,2530,2540,2550,2560,2570,2580,2590,2600,2610,
2620,2630
2520 IF (INT(Y/4))*4=Y THEN D3=29 ELSE D3=28 'TEST FOR LEAP YEAR
2530 D3=D3+31
2540 D3=D3+30
2550 D3=D3+31
2560 D3=D3+30
2570 D3=D3+31
2580 D3=D3+31
2590 D3=D3+30
2600 D3=D3+31
2610 D3=D3+30
2620 D3=D3+31
2630 D3=D3+0
3000 REM ADD THE DAYS IN THE WHOLE MONTHS OF THE REPORT YEAR
3010 ON M1 GOTO 3130,3120,3110,3100,3090,3080,3070,3060,3050,
3040,3030,3020
3020 D4=30
3030 D4=D4+31
3040 D4=D4+30
3050 D4=D4+31
3060 D4=D4+31
3070 D4=D4+30
3080 D4=D4+31
3090 D4=D4+30
3100 D4=D4+31
3110  IF  (INT(Y1/4))*4=Y1 THEN D4=D4+29  ELSE  D4=D4+28    'LEAP YEAR?
3120 D4=D4+31
3130 D4=D4+30
3500 REM ADD TOTAL DAYS IN BIRTH MONTH
3510 ON M GOTO 3520,3540,3520,3530,3520,3530,3520,3520,3530,
3520,3530,3520
3520 D5=32-D:GOTO 3600   '31-DAY MONTH
3530 D5=31-D:GOTO 3600   '30-DAY MONTH
3540 IN (INT(Y/4))*4=Y THEN D5=30-D ELSE D5=29-D  'FEBRUARY
3600 REM TOTAL DAYS SINCE BIRTH INCLUDING THE BIRTH MONTH
3610 DT=DT+D1+D3+D4+D5
4000 P1=DT/23            'NUMBER OF 23-DAY CYCLES SINCE BIRTH
4010 P=(P1-INT(P1))*23   'NUMBER OF WHOLE DAY IN PRESENT CYCLE
4020 E1=DT/28            'NUMBER OF 28-DAY CYCLES SINCE BIRTH
4030 E=(E1-INT(E1))*28   'SAME AS 4010
4040 I1=DT/33            'NUMBER OF 33-DAY CYCLES SINCE BIRTH
4050 I=(I1-INT(I1))*33   'SAME AS 4010
4100 4100 REM PRINT HEADER FOR BIORHYTHM CHART
4110 LPRINT TAB(20)"BIORHYTHM CHART FOR"
4120 LPRINT TAB(20) N$
4130 LPRINT TAB(20)"BORN";M;"-";D;"-";Y:LPRINT:LPRINT
4140 LPRINT TAB(10)"YOUR PHYSICAL CYCLE IS A 23-DAY CYCLE AND "
4150 LPRINT TAB(10)"IS SHOWN AS A 'P'."
4160 LPRINT TAB(10)"YOUR EMOTIONAL CYCLE IS A 28-DAY CYCLE "
4170 LPRINT TAB(10)"AND IS SHOWN AS AN 'E'."
4180 LPRINT TAB(10)"YOUR INTELLECTUAL CYCLE IS A 33-DAY CYCLE "
4190 LPRINT TAB(10)"AND IS SHOWN AS AN 'I'."
4200 LPRINT TAB(10)"WHEN TWO OR THREE LEVELS ARE EQUAL, THEY "
4210 LPRINT TAB(10)"ARE SHOWN AS AN '*'."
4220   LPRINT:LPRINT:LPRINT   "MONTH-DAY-YEAR"TAB(20)"LOW"TAB(40)
"ZERO-LINE"TAB(66)"HIGH"
4230 LPRINT
5000 FOR X=1 TO 90       'FOR A 90-DAY CHART
5010 IF X=40 THEN LPRINT CHR$(12)       'FORM ADVANCE FOR PRINTER
5020  P2=INT(25*SIN((360/23)*(P/57.3))+45     'CALCULATE POSITION
OF SINE WAVE IN CYCLE AND OFFSET BY 45 FOR GRAPH CENTER
5030 P=P+1:IF P=24 THEN P=1   'INCREMENT DAY FOR NEXT CALCULATION
5040 E2=INT(25*SIN((360/28)*(E/57.3))+45     'SAME AS 5020
5050 E=E+1:IF E=29 THEN E=1   'SAME AS 5030
5060 I2=INT(25*SIN((360/33)*(I/57.3))+45     'SAME AS 5020
5070 I=I+1:IF I=34 THEN I=1   'SAME AS 5030
5080 LPRINT M1;"-";D1;"-";Y1;       'PUT DATE ON CHART
5090 D1=D1+1         'INCREMENT DAY FOR NEXT ENTRY
5100 ON MI GOTO 5110,5130,5110,5160,5110,5160,5110,5110,5160,
              5110,5160,5110
5110 IF D1=32 THEN D1=1:M1=M1+1:GOTO 5170    'CHECK FOR LAST  DAY
OF A 31-DAY MONTH
5120 GOTO 5170       'NOT LAST DAY
5130 IF (INT(Y1/4)*4=Y1 AND D1=30 THEN D1=1:M1=M1+1:GOTO 5170
   'LEAP YEAR?
5140 IF D1=29 THEN D1=1:M1=M1+1:GOTO 5170    'LAST DAY OF FEB?
5150 GOTO 5170       'NOT LAST DAY OF FEB
5160 IF D1=31 THEN D1=1:M1=M1+1     'LAST DAY OF 30-DAY MONTH?
5170 IF M1=13 THEN M1=1  'CORRECT MONTH TO JAN
5180 IF D1=1 AND M1=1 THEN Y1=Y1+1       'UPDATE YEAR FOR JAN. 1
5500 REM THE FOLLOWING STEPS DETERMINT THE RELATIVE VALUES OF THE
THREE CURVES AND PLOT THEM IN ORDER
5510 IF P2<E2 AND P2<I2 GOTO 5580        'P2 IS SMALLEST
5520 IF E2<P2 AND E2<I2 GOTO 5620        'E2 IS SMALLEST
5530 IF I2<E2 AND I2<P2 GOTO 5660        'I2 IS SMALLEST
5540 IF P2=E2 AND P2=I2 GOTO 5690        'ALL ARE EQUAL
5550 IF P2=E2 AND P2<I2 GOTO 5700        'P2 AND E2 ARE SMALLEST
5560 IF P2=I2 AND P2<E2 GOTO 5710        'P2 AND I2 ARE SMALLEST
5570 IF E2=I2 AND E2<P2 GOTO 5720        'E2 AND I2 ARE SMALLEST
5580 LPRINT TAB(P2)"P";                  'P WAS SMALLEST, PRINT IT
5590 IF E2<I2 THEN LPRINT TAB(E2)"E"TAB(I2)"I":GOTO 6000
5600 IF I2<E2 THEN LPRINT TAB(I2)"I"TAB(E2)"E":GOTO 6000
5610 LPRINT TAB(I2)"*":GOTO 6000
5620 LPRINT TAB(E2)"E";                  'E WAS SMALLEST, PRINT IT
5630 IF P2<I2 THEN LPRINT TAB(P2)"P"TAB(I2)"I":GOTO 6000
5640 IF I2<P2 THEN LPRINT TAB(I2)"I"TAB(P2)"P":GOTO 6000
5650 LPRINT TAB(P2)"*":GOTO 6000
5660 LPRINT TAB(I2)"I";                  'I WAS SMALLEST, PRINT IT
5670 IF E2<P2 THEN LPRINT TAB(E2)"E"TAB(P2)"P":GOTO 6000
5680 IF P2<E2 THEN LPRINT TAB(P2)"P"TAB(E2)"E":GOTO 6000
5690 LPRINT TAB(P2)"*":GOTO 6000
5700 LPRINT TAB(P2)"*"TAB(I2)"I":GOTO 6000
5710 LPRINT TAB(P2)"*"TAB(E2)"E":GOTO 6000
5720 LPRINT TAB(E2)"*"TAB(P2)"P":GOTO 6000
6000 NEXT X    'BACK TO LINE 5000 TILL 90-DAY CHART IS COMPLETE
6010  LPRINT CHR$(12)     'CHART IS COMPLETE.   SEND PRINTER  THE
FORM-FEED INSTRUCTION TO ADVANCE PAPER.
6020 PRINT ED$           'ERASE THE DISPLAY
6030 PRINT SS$           'EXIT HOLD-SCREEN MODE
6040 PRINT CO$           'TURN THE CURSOR ON
6050 END
```

Well, there it is. It's a rather long program, and I didn't explain its operation other than in the REMarks within the program. Begin at line 440 and continue through 6050. Then MERGE CONTROL2.BAS and run the program. It is self prompting and should be easy to run. If you want, you can move END beyond 6050 and insert a routine which will permit you to produce more than one chart by answering a yes/no question as we did in our WORDS game last month.

Later in this series, we'll add these plotting functions to file- handling routines in order that we can follow trends of the stock market.

One final note: In many cases the programming I've used could be more compact. User functions would save space and time, etc. However, as this column attempts to be a tutorial, I don't want to introduce too many things too quickly. Please keep the letters coming so I can gauge my work to your needs. Because of my writing schedule, there's about a 5-month delay from my desk to a printed magazine, so the changes do take time.

See you next month.



What's an Algorithm?

"A defined process or set of rules that leads and assures development of a desired output from a given input."

- Heath Computer Dictionary

Jennifer T. McGraw
12741 SW 68th TR
Miami, FL 33183

If that is immediately clear to you , then you have a better command of the jargon than I do. Perhaps the best way to understand it is to look at an example, and then see if the definition can be put into everyday language.

Why bother? Well, many reasons, but the best one is because they're fun. I just finished reading "The Soul of a New Machine" by Tracy Kidder. Highly recommend it, but I bring it up here, because one of the engineers talks about the feeling of absolute satisfaction he got when he suddenly realized the simple (elegant) solution to a problem in the new computer they were designing. I've had that feeling a couple of times since starting programming. A great light dawns in your head and you sit there and grin and grin. This first happened when I 'invented' the second slowest sort ever. But that's the point; it wasn't the slowest.

Here's an algorithm for something called the Short Sort from the Algorithms Micro Chart from Micro Logic Corporation.

Function: Sorts into ascending order the N elements of array A. Only useful if N is small. Time: N ↑ 3.

Algorithm: Starting with elements 1 and 2, look for a pair out of order. If none, then finished, otherwise swap the pair and go back to step 1.

That's the algorithm. Possibly the slowest sort, for obvious reasons. Now here it is in BASIC:

```
100 I=1
110 IF A(I) <= A(I+1) THEN 140
120 SWAP A(I),A(I+1) : I=0
```

If your BASIC doesn't have SWAP:

```
120 H=A(I) : A(I)=A(I+1) : A(I+1)=H : I=0
130 I=I+1 : IF I<>N THEN 110
```

Ok, what's an algorithm? It's a recipe to be used by a computer! (I don't think Mom's peanut butter cookies are the result of one.) A program or subroutine is the interpretation of the algorithm into the particular language you are using.

The above is a simple example. Writing the function let's you know what you are trying to do. And if you can explain the steps, you understand exactly what you are doing. Furthermore, by stating them verbally, a shorter, simpler solution may occur.

Ok, having defined it, to my own satisfaction at least, here are some interesting algorithms in BASIC that I've picked up. Even wrote one or two.

Here's one used for a calendar program, that takes each number of a year as a single digit, to be used to print the year in BIG characters.

```
100 REM Separate each digit of Y$ (Year), put in array U (Unit)
110 FOR K=1 TO LEN(Y$)
120 U(K) = VAL ( MID$ (Y$,K,1) )
130 NEXT
```

This is a simple one to determine the number of days in February, bearing in mind that in years ending in '00', i.e. 1900, only those evenly divisible by 400 are Leap Years. Since this fact won't affect us until 2100, it's just included for 'class'.

```
100 REM Is it Leap year?  Y=Year  D2=Days in February in Leap Yr.
110 REM D2 is the number of days in February, natch.
120 IF (Y MOD 4=0 AND (Y MOD 100<>0 OR Y MOD 400=0)) THEN D2=29
```

If your BASIC doesn't have MOD:

```
120 Y1=4:IF Y/100=INT(Y/100) THEN Y1=400
130 IF Y/Y1=INT(Y/Y1) THEN D2=29
```

I found this one in Mr. Wilkenson's CALENDAR program in HUG Volume II, 885-1013. It's compressed a little.

```
100 REM    Compute day of week for date of years after 1752
110 REM    Y=Year : M=Month (1-12) : D=Date : DEFINT Z
120 REM Z7 is a number from 0 to 6, 0 = Sunday and 6 = Saturday
130 Z0=Y : Z1=M+1
140 IF M<3 THEN Z0=Y-1 : Z1=M+13
150 Z2=INT( Z0/100 )
160 Z7=( INT(Z0*1.25) + INT(Z1*2.6) + INT(Z2/4) - Z2+D-1) MOD 7
```

If you don't have the MOD function, or just don't like it, delete it from line 160 and add:

```
170 Z7=Z7-( 7* INT( Z7/7 )
```

No, I don't understand it completely, but it's neat. You could make a chart showing all the Friday the 13's for the next 200 years. The next is courtesy of Mr. Parker, HUG Volume I.

```
110 REM Determine monthly payment on declining balance loan
120 REM TT=Length of loan in months : B=Amount of loan
130 REM R=Interest rate in percent I.E. 10 (10%) NOT .1
140 REM PA=Monthly Payment
150 R=R/1200 : YR=(R+1)^TT
160 PA=B* ( (R*YR) / (YR-1) )
170 PA=INT( PA*100+1 )/100
```

I'm not sure that payment is "desired output" if I'm the one making the payment.

```
110 REM Change hours & minutes to decimal, N$ to N
120 REM  N$=hours and minutes without the colon.
130 REM  N$="100"=1 hour, "120"=1 hour, 20 minutes
140 N=VAL(N$/100) : N=INT(N) + (N- ( INT(N) ) *100/60)

210 REM Change decimal to hours and minutes, N to N$
220 N$=RIGHT$( STR$( INT(N) ),LEN( STR$( INT(N) )-1 )+":"
230 N$=N$+ MID$( STR$( INT( ( N-INT(N) )*60+100.5) ),3,2)
```

Adding .5 is obviously rounding off. The 100 is added to get the zeroes, if needed. At this point I'd better note that I've added a lot of spaces in the above listings for clarity.

To learn more algorithms, your best reference is all the programs you can find. See what the solution to a given problem is, and try to figure out a better way to do it. Or a worse way. At least you'll be programming, which is what this is all about.

I saw a book in the Dalton Bookstore that probably has all the algorithms you'd ever want, but all I can remember is the price, $34.95. Micro Logic's Algorithms Microchart contains a bit count, integer square root, shuffle, sampling, line plot, and a bunch of sort algorithms. It can be obtained at some Heathkit Electronic Centers. The algorithm for the Shell-Metzner sort is in Doc Campbell's article on disk files in issue #10 of REMark. A final note: if you write, and want a reply, please include a self- addressed stamped envelope, (SASE).

Echoes of the Crash

Mary Hess
St. Paul - Minneapolis
Users' Group - Heath

(Note: The following material was taken from The St. Paul - Minneapolis Users' Group - Heath Newsletter, SMUGH, 8895 72nd St., Cottage Grove, MN 55016.)

Rattle, clatter, thunder, boom, boom, boom, Mr. Karxxxman was not there to help us when our system went CRASH with a resounding thud. It is now a bit funny and a little embarrassing to admit. BUT THEN it meant no sleep and long faces. No subject was safe to talk about. Dinner was soft food only as long nights without sleep took their toll. Numerous phone calls taught us that no two computers are alike, and knowing one does not mean you know another. Hardware neighbors tried to help - to the embarrassment of both of us. They did not know the first thing about Heath or software. A family with three trained working computer persons offered help. We were thrilled, until they discovered they could not even talk to each other, let alone help us.

Thus we learned the generalities of this business. All computers are not created equal. All persons are not trained, motivated, or supplied alike. A fairly knowledgeable person told us the best way to buy a computer - 'Pick out the software that does what you want to do, check out the users' group in your city, and buy that machine'. All users' groups are not created equal either. In many ways, the broadness of this field allows self-selection so that the fellow who wants to rebuild the whole system and program his own software has room to do it, while the kids who need to word process college term papers can do that too. The only problem which persists is that we are used to thinking a cyclist can ride any bike, foreign or domestic; a driver can drive anything on wheels; a cook in any kitchen can produce Eggs Benedict. However, we are first shocked and then dismayed that an Apple user is just applesauce in our den. Rarely can we even discuss what we want done, let alone how to do it.

First I called every listing in the phone book for computer classes. No one who answered had even heard of Heath and few knew what 'crashing' was all about. I asked for classes in CP/M, word processing and/or MBASIC. None available. One girl asked if I were going to use this for business. When I replied in the affirmative, she said, "Then your boss will teach you". Honest! We soon learned that personnel in stores are order- takers, not trained salesman users. If we knew what we wanted, they could get it written up. If we did not, they had no idea, or worse, they had ideas that even with my limited knowledge, I knew would not work. The best discussion I had in several weeks was at a Bookseller's rack with another seeker (customer).

Next I called every university, college, tech school in the area. They either had no classes in software subjects or they are booked into the future. The Community College has an introductory course booked into late 1984, they are thinking of adding sections since this is a non-hardware (all lecture) course. The limit of machines puts the actual 'hands-on' course schedule into infinity and those are IBM, TRS-80, or Apple units. Little or nothing at these schools will fit the needs of a home user of Heath. If you find one, tell the editor.

For $5,000, several private schools can give me a course in data-processing guaranteed not to help with a Heath machine! The biggest name in the city suggested I take their test (which I passed) and then take their course for 2 years and $5,000 to get a general overview of the computer field. Wow! Are there some really falling for that —? As new as I am, I know the field is changing so fast that the time and money would be wasted.

In our area, there are five CPU's that I know of, none 'OUR' brand. The few discussions we have had left us all disappointed, no one can help nor enjoy the others. Some great software must go by the board.

To get technical; our first main problem was in the drive. Nowhere did we read that it needed to be turned on at all times, even when not in use. This is so illogical, it should be in large letters in a survival manual. Not having it turned on, we erased some materials. Then I did not know about the mount- dismount commands so I erased some more material including the system volumes. I could not make backups without erasing some, so software goofs were piling up. With every trip to the store, they could only advise "read the manual". That SOUNDS great, but you all know the manuals are not written in the best prose nor does reading cure everything. Only after I put the actions together with the instructions did I start to find success. Then we found some pages missing. Gene is more persistent than I am, so he got through the barriers before I did. I tend to stomp my foot a lot when this dumb thing says 'ok' and does not mean a thing by it.

After working with Ron's 'new members class' and some great advice from other Smuggies, we are at last talking to this dumb machine without having it stick out its tongue EVERY time. I am even getting brave enough to send this to our great editor via diskette with the positive expectancy that it will arrive in processed perfection!

Speaking of survival manuals, wouldn't that be nice? How about some of you word processors out there putting together a few pages for the new user? Heath assumes that only the already tuned-in person will buy a Heath unit - they told me that. Even a person familiar with computers in general needs some help for this unit specifically. The club could hand it out to new members, or even mail it to new purchasers as a goodwill gesture and advertisement for the SMUGH group. Each of us must have one favorite 'must' for starting up. It would not take long to get a few protection messages together to carry one through until the manual reading is completed and the dawn is beginning to break.

Having used cassette tapes, I knew about magnetic disturbances and heat and dust, etc. We did NOT make ALL the possible mistakes! We have lovely cases and safe storage. Gene has built a great table and shelves. This thing is beginning to fit into our lives with fewer problems every day. Calluses are easing the pains, I guess.

We found out about the SMUGH folks from a new friend and went to the next meeting which was fortunately the next day. There we found some helpful people who knew their limitations and pointed us to those who could help with such a big problem. We found very helpful 'experts' who would take the time for new persons and have patience with the simple questions and silly errors. This is not true in every users' group in this city. It took us two months to get all the damages repaired so we could start operations ... a time of distress only one who has been there can understand. Muchas gracias, amigos.

It was hard at first to admit failure and mistakes, especially since my former jobs put me in the leadership roles. I was supposed to understand and help others. This is a time when it was definitely more blessed to give than to receive help. This group is genuinely service-oriented with no putdowns. I have been treated to several hilarious stories about other crashings. We are very grateful to you all for being there when it seemed all the doors were closed.

The CP/M Directory Structure

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

In a previous article we examined how the Heath Disk Operating System, HDOS, keeps track of all the sectors and files on a disk. Obviously, each disk operating system keeps track differently than any other system. So this month we'll investigate CP/M, specifically the 10 sector, hard sectored, Heath implementation. I will limit myself to version 2.2.03 as earlier versions do things a little differently. If you understood HDOS, then you'll find CP/M a breeze. It is much simpler and has almost none of the protective features of HDOS. To understand the HDOS directory structure you have a lot of table look up and math to do. This is not the case in CP/M, so take heart and enjoy. Although I have included figures to help you follow the discussion, you will find it easier to use your H8 or H89 and either SZAP.COM, DDEU.COM, or any other of the direct disk access utilities.

To those of us familiar with HDOS, one of the first noticeable features of CP/M is the lack of the three system files. Initialized HDOS disks start with three files, RGT.SYS, GRT.SYS, and DIRECT.SYS, but CP/M apparently has none of these. As you may remember, the RGT.SYS is nothing more than a list of flags, one for each cluster on a disk. These flags indicate if a cluster is available for use or not. CP/M has no such animal! It does not keep track of good or bad sectors on the diskette. Well, so much for one of the protective devices of HDOS. For those of us who have some bad or questionable sectors on a CP/M disk or two, there are three ways of solving the problem. First, convert the disk(s) to HDOS. Once there, the bad sectors may be mapped out. Alternately, you can create a CP/M file, hide its name in the directory, and force it to contain the bad sector. I'll show you how to do this later on. The third and probably the best solution for a known bad sector is rip the jacket off the diskette, show the kids what a disk looks like inside, then throw it (the disk, not the kids) in the trash can. Before you try something this drastic, try reinitializing the diskette.

The functions of the remaining two HDOS files are included in the CP/M directory and are combined and saved at a fixed location on disk. Before we get into the contents of the directory itself, lets look at the disk structure. The HEATH hard sectored disk, whether HDOS or CP/M, contains 40 tracks of 10 sectors each for a total of 400 physical sectors. Each of these sectors contains 256 bytes of user accessible information. There is also a handful of bytes outside of the user accessible 256 bytes. These bytes contain the track, sector, volume number, and synchronization patterns (10 null bytes and an FD(hex)) and are used by the operating system to insure it has found the correct sector. They are identical in HDOS and CP/M.

This is where CP/M and HDOS begin to part company. HDOS considers each physical sector, 256 bytes, as a logical sector. But CP/M, to maintain compatibility with earlier versions and other CP/M systems, uses a 128 byte sector. Consequently, each physical Heath sector contains two logical CP/M sectors. Therefore, there are 800 CP/M logical sectors on a hard sectored HDOS CP/M disk. Throughout this discussion I will refer to the 128 byte CP/M sectors as CP/M logical sectors or simply as CP/M sectors.

We discovered that HDOS formed clusters of two sectors each to address all 400 of its sectors with a one byte address pointer. CP/M, not to be out done, has a nearly identical clustering scheme. In CP/M parlance a cluster of sectors is called a group. Each CP/M group contains 1024 bytes which is eight CP/M logical sectors or four physical sectors. Using this scheme a byte wide group pointer can address 256 groups. This is far in excess of the number required to address a 100K diskette. CP/M drives with larger storage capacity increase the number of sectors in a group.

HDOS begins numbering its clusters at the beginning of the diskette, but not so in CP/M. It starts with the first sector of track three! I know that brings the question "Why?" to mind. This is an easy one to answer. Track 0, 1, and 2 are boot tracks. On an HDOS disk, only track 0 is the boot track. HDOS needs only one track because only a bootstrap loader is stored there. That loader's job is to read the disk and bring in HDOS.SYS which is the heart of the HDOS operating system. CP/M uses three tracks because nearly the entire operating system can be saved there. Just as HDOS files cannot use track 0, CP/M files cannot use tracks 0, 1, and 2. This creates some good news and some bad news. Good news first, a CP/M diskette may be made bootable (SYSGENed) at any time after it has been initialized. You may have put numerous files on the disk and it still may be SYSGENed. HDOS doesn't have that luxury, it must be SYSGENed after initialization and before the disk is full. Do you want the bad news now? Those three reserved tracks consume 7680 bytes from all CP/M disks, whether bootable or not. Further, that's only half the bad news. The 7680 bytes does not include the directory itself or BIOS.SYS. The directory uses the first two CP/M groups, 2048 bytes. We now have 90.5K bytes even if the disk is a data disk and not bootable.

Well, it sounds like we are about ready to examine the directory itself. This is an area where the simplicity of CP/M is apparent. As I said earlier, the directory uses the first two CP/M groups. It consequently starts at track three, sector zero and continues through track three, logical sector fifteen. A word of warning is in order here. CP/M doesn't number its physical sectors consecutively like HDOS does. HDOS numbers them in the same order they are physically on the diskette. CP/M jumbles them up. This is done to decrease the amount of time required to wait between reading consecutive sectors. I won't go into how this works in this article, maybe I will sometime in the future. Be careful when using any of the CP/M disk dump programs. Some of them dump by logical sectors and some by physical sectors. SZAP.COM does physical, whereas DDEU.COM dumps by logical sector. To check which technique your favorite dump program uses, start dumping from track three and see if the

Physical To Logical Sector Translation

Heath Physical Sector Number (based on 0)	CP/M Logical Sector Numbers (based on 0)	CP/M Logical Sector Numbers (based on 1)
0	0 and 1	1 and 2
1	8 and 9	9 and 10
2	16 and 17	17 and 18
3	4 and 7	5 and 8
4	12 and 13	13 and 14
5	2 and 3	3 and 4
6	10 and 11	11 and 12
7	18 and 19	19 and 20
8	6 and 7	7 and 8
9	14 and 15	15 and 16

Figure 1.

Logical Sector To CP/M Group

CP/M Group Number	First Logical Sector Number (track/sector)	Last Logical Sector Number (track/sector)
0	3/0	3/7
1	3/8	3/15
2	3/16	4/3
3	4/4	4/11
4	4/12	4/19
5	5/0	5/7
6	5/8	5/15
7	5/16	6/3
8	6/4	6/11
9	6/12	6/19
10	7/0	7/7
11	7/8	7/15
12	7/16	8/3
.	.	.
.	.	.
.	.	.

(GROUP * 8) + 60 = (20 * TRACK) + SECTOR

Figure 2.

The CP/M File Control Block

```
                    - FIRST 16 BYTES -

          E
          N        F              F                R
          T        I              I            N   E
          R        L              L            O   C
          Y        E              E       E    T
                                          X        C
          T        N              T       T    U   O
          Y        A              Y       E    S   U
          P        M              P       N    E   N
          E        E              E       T    D   T

 relative pos. 00  01 02 03 04 05 06 07 08  09 0A 0B  0C  0D 0E  0F
typical  (hex) 00  42 49 4F 53 20 20 20 20  D3 D9 53  00  00 00  24
entry  (ASCII)  .  B  I  O  S               S  Y  S   .   .  .   $


                    - LAST 16 BYTES -

                        DISK ALLOCATION MAP BYTES

     relative pos. 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
    typical  (hex) 02 03 04 05 06 00 00 00 00 00 00 00 00 00 00 00
    entry  (ASCII) .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
```

Figure 3.

directory is interspersed with sectors from other files. Figure 1 depicts the physical to logical sector number cross-reference.

Examine the first sector of the directory (track 3/sector 0 or sector 60). If you have chosen a disk that has no files on it, you will notice the sector is filled with E5(hex). If this is the case, pick another disk to dump. You'll find an empty directory very boring. This first sector may have up to four entries in it. Each entry uses 32 bytes and is referred to as a 'file control block'. In 2048 bytes there is sufficient space for 64 file control blocks. As we discuss these 32 bytes, follow along on figure 3. The first byte is the directory entry type and has two uses. First, this byte is used to store the USER value for a file. USER 0 files will have a null byte, USER 1 files will have a 01(hex), and so on. Secondly, if the file has been deleted, this byte will contain an E5(hex), flagging the file control block as being available for use. The following eight bytes are the file name. The space character (20 hex) is used to pad this field to eight characters. Following the name, the three character file extention is next. It too is padded with spaces. The high order bit of each of its three characters are flags. The first character's high order bit flags the file as a system file ($SYS). If it is not set, the file is a normally displayed file ($DIR). The second character's high order bit flags the file as read only ($R/O). If it is not set, the file is capable of read and write ($R/W). Some documentation shows the last character high order bit as an update flag, but I haven't found any support of this on any of my disks or in the Heath documentation.

The 12th byte is the extent and is usually zero. In a few minutes, I'll explain how this byte is used. Be patient for now. The next two bytes are not used and are usually set to 00. The following byte is very important. It is called the 'record count' and represents the number of CP/M sectors included in this file control block. Remember a CP/M group contains eight CP/M sectors. The record count gives the information necessary to find which sector is the last sector in the last group. The maximum value you will ever see here is 80(hex).

The remaining fifteen bytes in each file control block are the meat of the directory. These numbers are the group numbers of those groups which a file uses. Wow, that was a tongue twister. The directory uses groups zero and one; therefore, the lowest value any group number entry in any file control block is 02. To convert this value to a CP/M sector number, multiply by eight and add 60. The first file in figure 4 starts in logical sector 76. If you didn't find it there, did you remember to convert from logical to physical sector? Logical sector 76 converts to track three and logical sector 16. If your disk dump routine dumps by physical instead of by logical sector, then use figures 1 and 2 to convert logical sector 16 to physical sector. You may find your disk dump routine counts the first track as track 0 or it may count it as track 1, the same may be true for sectors within a track. In this case you will have to take this into account. Both SZAP and DDEU start counting at 0, like all good programmers should. Be careful and remember there are 20 sectors per track.

My next point has probably already occurred to you, so now is a good time to discuss it. There is only room in each file control block for 16 CP/M groups. That creates a limit of 16K per file. You and I both know CP/M can have a file longer than that. Here is how it's done. Remember the extent byte I mentioned earlier? It is normally set to zero. If a file is longer than 16K, another file control block is used and the extent byte is used as a counter. The second block will have a 01(hex) in its extent byte, the third will have a 02(hex), and so on. All the information in each extented file control block will be identical to the initial block with the exception of the extent byte, record count, and of course the group numbers. Simple, isn't it?

Typical CP/M Directory

```
   TRACK: 3                GROUP: 0              LOGICAL SECTOR: 0

     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
    -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  ----------------
000: 00 42 49 4F 53 20 20 20 20 D3 D9 53 00 00 00 24  .BIOS....SYS...$
010: 02 03 04 05 06 00 00 00 00 00 00 00 00 00 00 00  ................
020: 00 53 57 45 45 50 20 20 20 43 4F 4D 00 00 00 80  .SWEEP...COM....
030: 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16  ................
040: 00 53 57 45 45 50 20 20 20 43 4F 4D 01 00 00 5C  .SWEEP...COM...\
050: 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 00 00 00 00  ................
060: E5 50 49 45 20 20 20 20 20 43 4F 4D 00 00 00 30  ePIE.....COM...0
070: 23 24 25 26 27 28 00 00 00 00 00 00 00 00 00 00  #$%&'(..........

   TRACK: 3                GROUP: 0              LOGICAL SECTOR: 1

     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
    -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  ----------------
080: 01 4F 4E 45 43 4F 50 59 20 43 4F 4D 00 00 00 07  .ONECOPY.COM....
090: 29 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  )...............
0A0: 10 50 41 54 43 48 43 20 20 43 4F 4D 00 00 00 01  .PATCHC..COM....
0B0: 2A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  *...............
0C0: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
0D0: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
0E0: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
0F0: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee

   TRACK: 3                GROUP: 0              LOGICAL SECTOR: 3

     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
    -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  ----------------
100: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
110: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
120: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
130: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
140: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
150: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
160: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
170: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  eeeeeeeeeeeeeeee
```

Figure 4.

A word of warning is in order here. In some circumstances PIP will not find all the file control blocks of a file. Some programs use the extent byte to indicate a part of the file which it will load later, like an overlay. SuperCalc is an example of this. There are two files with the same name on the SuperCalc disk. The first uses extent 00 and 01, whereas the second uses extent 06 and 07. When PIP tries to move the file, it finds extent 00 and 01 but cannot find extent 02. It consequently stops with the first two extents. To move the following two extents, simply edit the names to something other than SC and change the extent bytes to 00 and 01. PIP can then be used to move the remaining parts of the file. Don't forget to change the names back to what they were originally and to set the extent bytes back to 06 and 07.

Another practical example of this information is to un-erase a file. Have you ever deleted a file by mistake? Who hasn't? If you can't find your copy of UNERA.COM, use your disk editor and change that first byte from E5(hex) back to 00. If you haven't written anything else to the disk, changing the first byte will safely bring the file back to life. If you have written to the disk, your chances of having the deleted file's contents still intact is almost nil.

The final example we will investigate is how to reserve a bad sector. Remember, I promised you that one a long time ago. The hardest part of this procedure is to identify which CP/M group the bad sector belongs to. Use the file dump capability of your disk editor to find which sector from the start of the file is the errant sector. Divide this sector number by eight, discard the remainder, and then add one. If the result is one, the first group in the file control block for this file is the culprit. If it is two, suspect the second, and so on. Create a file containing less than or equal to one group, 1K bytes. Edit its file control block so that the one group it references is the bad group. While you are here, you should also change the first byte to 10(hex) to put the file name into the 16th USER area, hiding it from all the valid USER areas. The original file may be patched around the bad group by simply eliminating the errant group number. Do this by shifting each of the higher group numbers down one position. Don't forget to look for possible extents and don't forget to subtract the eight records from the record count byte. Notice that what you have patched out is up to an entire group, 1K, not just a sector. If it is the last group of a file, then correcting the record count byte is a little more difficult. In this case we need to compute the number of sectors contained in the last group and subtract it instead of eight. To find this value, simply divide the old record count by eight. Subtract the remainder from the old record count. The result is the new record count.

All of the information I have presented can be acquired from either examination of the contents of a diskette or from Heath's documentation. Figure 4 contains the first three logical sectors of group 0 from a CP/M diskette. You can spot both the $SYS and the $R/O flags on the file control block for BIOS.SYS. SWEEP.COM was on this disk and, as you can see, it requires two file control blocks. The next entry, PIE.COM, has been deleted. Did you find the E5(hex) in its first byte? I put ONECOPY.COM in USER 1 on this disk to show the 01 in the first byte. Logical sector 400 is a bad sector on this disk. To patch it out, I created the file PATCHC.COM consisting of one logical sector. I then edited the file control block to point to group 2A(hex) and also to put the file in USER area 16. Because 16 is an invalid USER area, the only way the file can be erased is with a disk editor. Group 2A(hex) contains logical sectors 396(dec) through 403(dec).

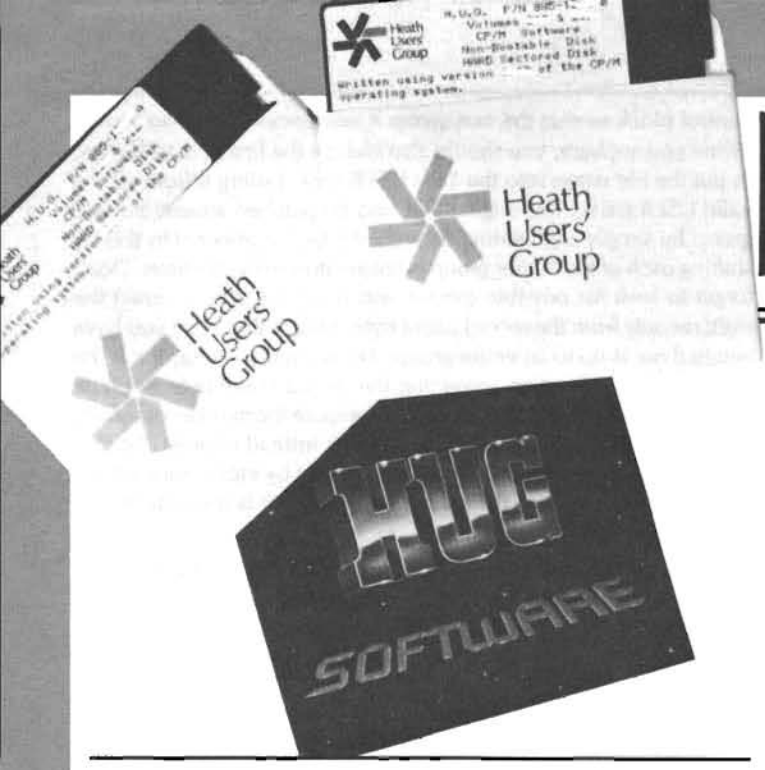
If you find some of this information confusing, use your disk dump program and play with a disk. Oh, and of course, if all else fails, read the documentation. The source listing for the Heath BIOS.SYS is very helpful too. Read it carefully!

In conclusion, any time you fiddle with any of the operating system, be very careful. It doesn't matter if the system is HDOS, CP/M, or any other operating system; you can loose everything on the disk if you aren't sure of what you're doing. I recommend you duplicate one of your CP/M disks and then muck with the copy for practice. Good luck and enjoy.

DON'T FORGET!!!

You will be voting next month for your favorite article in 1983.

HUG NEW PRODUCTS



885-1128 HDOS
DISKVIEW $16.00

Introduction: DISKVIEW is an advanced disk catalog program that lets the user see and manipulate all of the files from two disks at the same time. It lets the user scroll back and forth through the catalogs of one or two disks, sort the catalog in order of name, extension, size, date, or flags. By placing the moveable cursor on a catalog entry, the user can list to the screen, print to a printer, or copy to a disk the file that is marked. Files can be deleted, flags changed, or disks reset.

Requirements: This disk requires the HDOS operating system on an H8/H17/H19 or H/Z89 with 48K of memory. Only one disk drive is required, however, it is written for any number and type of disk drive formats. A printer is not necessary.

The programs are written in "C" and the source files are included. The programs have been compiled and are ready to run. To recompile the programs, the Software Toolworks C80 "C" compiler version 2.0 will be required. Also needed would be the Microsoft Assembler M80 and Linker L80 to assemble and link the various files into an .ABS file.

Note: The H/Z19 terminal is required.

The following files are included on the HUG P/N 885-1128 HDOS DISKVIEW disk:

README	.DOC	DEVTBL	.CDF
DISKVIEW	.ABS	USDT	.CDF
DSKVW	.C	DVHELP	.MAC
COPY	.C	SUBSTR	.C
FLGFIL	.C	STRCAT	.C
DELFIL	.C	PUTBUF	.C
SORT	.C	PRINTF	.C
NEWDRV	.C	UCASE	.C
READIR	.C	CPROF	.C
DATESTR	.C	SEEK	.C
DIRENT	.C	EXEC	.C
FMTSCR	.C	FGETS	.C
LINE25	.C	STRLEN	.C
FSPACE	.C	STRCPY	.C
TITLE	.C	STRCMP	.C
TERMCTL	.C	ATOI	.C
CURSOR	.C	ITOA	.C
GETNAM	.C	REVERSE	.C
SEXIT	.C	INDEX	.C
DSKVIEW	.CDF	GETS	.C
DIRLIST	.CDF	PUTS	.C
DIREC	.CDF	FPUTS	.C

Author: Dave Hildebrand

Program Content: This advanced disk catalog program displays two disk catalogs on the screen at the same time. If there are more than 22 files on either disk, the program will scroll back and forth through the catalog listing. There are a number of options available to the user once the program has begun.

The arrow keys of the keypad allow the user to select a desired disk and file. Once selected the user can delete, change flags, view it on the screen, transfer to another disk, copy with a new name, or send it to the printer. The catalog listing can be sorted. Each of these options are selected by depressing one of the special function keys.

In the catalog mode there are a two things different than when CAT (or DIR) is entered from HDOS. First, the number of sectors shown is always the number of sectors actually allocated to the file. Second, the DATE shown in the catalog mode is the last date the file was updated.

The SORT option will sort the catalog in ascending order by name and extension. It will sort in descending order by date, size, and flags.

The COPY mode will allow the user to view ASCII (text) files or binary (absolute) files in HEX format. Files longer than 23 lines can be controlled with the SCROLL key, either advancing one line at a time or an entire screen. In addition, files may be copied to another disk, sent to the printer, or sent through any HDOS device driver.

Compiling the source code files is not recommended for anyone not familiar with the "C" compiler. The executable file should be complete for most applications.

Comments: This is a unique and helpful program for maintaining and manipulating disk files. (E.G. With this program, you can sort by date and delete only those files with a particular date.)

885-1129 HDOS
885-1232 [-37] CP/M
CVT Color Video Terminal $20.00

Introduction: CVT (Color Video Terminal) is a program that allows you to operate HA-8-3, HA-89-3, or similar color boards for the H8 and H89 computers as if they were a color version of an H19 terminal. It allows you to create a color display using nearly all printable H19 characters including the graphic characters. It makes creating displays with your color board as easy as creating displays on your

H19 or H89. CVT is available in HDOS and CP/M versions, which are virtually identical.

Requirements: You will need the HDOS or CP/M operating system, at least 32K of RAM, and a color graphics board such as the HA-8-3 for the H8 or the HA-89-3 for the H89 (Z89/Z90). An H19 or H29 terminal is required for proper operation with an H8.

The following files are included on disks 885-1129 and 885-1232 [-37]:

HDOS 885-1129		**CP/M 885-1232 [-37]**	
README	.DOC	README	.DOC
CVT8	.ABS	CVT8	.COM
CVT89	.ABS	CVT89	.COM
CVT	.DOC	CVT	.DOC
CVT	.ASM	CVT	.ASM
CVT	.PIC	CVT	.PIC
GRAPH	.PIC	GRAPH	.PIC
MCVT	.ABS	MCVT	.COM
MCVT	.ASM	MCVT	.ASM

Author: Pat Swayne, HUG

CVT8 and CVT89 — These are ready-to-use versions of CVT for use with the HA-8-3 and HA-8-89 and compatible color boards. CVT will work with either the TMS-9918 or TMS-9918A video processor so that it will work properly on older HA-8-3 boards. The following specifications may help you in determining the usefulness of CVT in your application.

Color Processor required: TMS-9918 or 9918A.
CVT Modes used: Text or Pattern (G1).
Characters/line: 40 (text) or 32 (pattern).
Lines/screen: 24 (H19 25th line not supported).
Colors/screen: 2 (text - foreground, background), 15 (pattern).
Character set: All normal letters, numbers, and symbols. All graphic characters except the plus/minus and divide signs. The opposite corner squares missing from the H19 set are provided. Reverse video supported.
Character matrix: 6 by 8 (text), 8 by 8 (pattern). Letters and numbers are formed within a 5 by 7 matrix.

CVT supports most of the H19 escape codes and adds extra ones for setting modes, colors, etc. In addition, the function keys on your H19/H89 can be used to set up modes and colors while you are drawing a picture. In the pattern mode, the foreground and background colors of groups of patterns can be set to change the colors of textual or graphic characters. In the text mode, only the foreground and background colors of the entire screen can be set.

CVT can load a picture you have created directly from the HDOS or CP/M prompt, and can link to another program after loading the picture. This allows you to draw scenes with CVT and then use them in other programs. A memory map of CVT's usage of the video processor on the color board is provided in the .ASM file so that advanced users can manipulate the picture directly. CVT itself can also be utilized by other programs to take strings of ASCII text and process them into the picture.

In addition to being able to load its own files, CVT can load and display ordinary text files on the color screen. These files can contain escape codes to control not only normal H19 parameters, but also CVT modes and colors. (Files created by CVT are not text files, but are direct dumps of the video processor's memory to disk.)

CVT.PIC and GRAPH.PIC — These are sample pictures created with CVT. You can view these pictures by typing

CVT CVT

or

CVT GRAPH

MVCT — This is a program which demonstrates the ability to manipulate a picture after it is created. When you enter

CVT CVT MCVT

CVT loads in CVT.PIC and then links to MCVT.ABS (or MCVT.COM). This program causes parts of the picture to change color and creates a flashing sign effect.

Comments: CVT adds an excellent dimension to anyone having an H/Z89 or H8 with a color monitor.

885-8024 HDOS
BHBASIC Utilities Disk $16.00

Introduction: This Benton Harbor BASIC Utilities disk is designed to facilitate the BASIC programmer in developing modulized BASIC programs, creating and utilizing a simple library of subroutines, construction of CHAIN overlays, and in eliminating memory storage requirements for REM statements in executing BASIC programs.

Requirements: This package requires the HDOS operating system on an H8/H17/H19 or H/Z89 computer with 32K of memory. Only one disk drive is required. A printer is not required.

The program is written in Benton Harbor BASIC.

Note: The H/Z19 terminal is required.

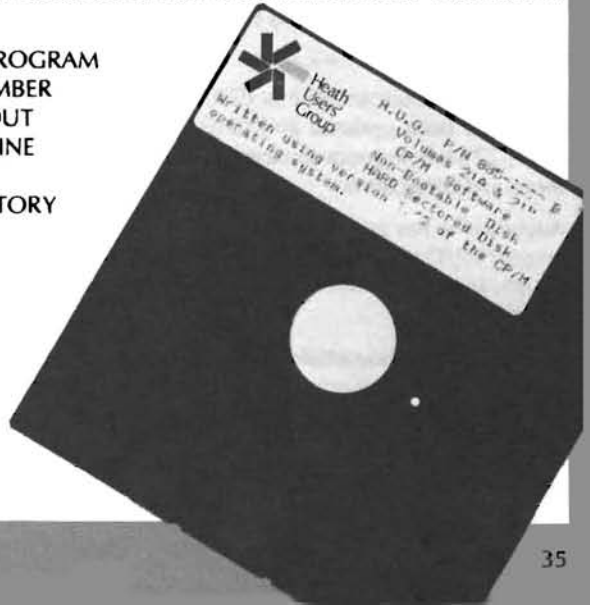
The following files are included on the HUG P/N 885-8024 HDOS BHBASIC Utilities disk:

BASICUTI	.BAS
TESTDATA	.1
TESTDATA	.2
TESTDATA	.3
TESTDATA	.4
TESTDATA	.5
TESTDATA	.6

Author: Terence Bordelon

Program Content: This Benton Harbor BASIC utility includes commands which are selected from a command menu. The menu is as follows:

(0) END PROGRAM
(1) RENUMBER
(2) REM-OUT
(3) COMBINE
(4) SPLIT
(5) DIRECTORY



Options one through four require input from one or two files and output to one or two files depending on the command selected. The file names may consist of any disk device, file name, and file extension. At each input or output file prompt, the user can ask for verification that the name does or does not exist on the disk.

Each one of these commands provide the user with the option of viewing all changes. If selected, each source line is displayed with the resultant line immediately following. At the end of the command, the program issues a count of the lines processed for user notification and verification.

RENUMBER — This command renumbers an entire BASIC program. The user provides the starting line number for the new file and the line number increment. Any warning messages may be stored to a separate file. (Note: The original file is still intact after the RENUMBER operation. A new renumbered file will have been created.)

REM-OUT — This command strips out all REM statements within a BASIC program and stores the resultant program in one file and the REM statements in another file. The COMBINE option will restore the REM statements to the BASIC file.

COMBINE — This command takes two BASIC files and combines them into one file. The lines from the two source files are written to the destination file in ascending line number order.

The COMBINE command has two operating modes. One mode combines two BASIC program files. The other combines a REM file with its corresponding BASIC program file. (See REM-OUT.)

SPLIT — This command splits one source file into two destination files. The split is based on block starting and ending line numbers. One or more blocks consisting of one or more lines may be split out. At completion, the source file will be intact.

DIRECTORY — This option allows the user to do a directory of any disk drive while in the program. The user can select to see files not flagged or all files including files flagged by the "S" flag.

Comments: The program has been written entirely in BASIC and is commented throughout.

Product Updates

885-8010 HDOS
885-8011[-37] CP/M
CHECKOFF $25.00

This is an announcement that CHECKOFF under HDOS (P/N 885-8010) has been updated and CHECKOFF under CP/M (P/N 885-8011[-37]) is being reannounced. Both programs have had improvements and corrections.

Anyone who has purchased either of these products may return the original HUG disk to HUG Software, Hilltop Road, St. Joseph, MI 49085, in care of Nancy Strunk, for an update. **There is no charge for this update.**

CHECKOFF is a household utility to maintain a balanced personal checking account. It permits editing, sorting, merging, and searching of data records in a file.

Refer to the HUG Software Catalog pages 54 and 55 for a detailed description of the abstract.

885-8018[-37] CP/M
FAST EDDY and BIGED $20.00

This is an announcement that FAST EDDY has been modified to run under the 8080 microprocessor. It will now run on the H8/H17/H19 or H/Z89 with 32K of memory. A printer would be useful. (It was previously announced as Z80 based only.)

FAST EDDY is a text file screen editor designed for anyone not familiar with an editor. It is a useful tool with many helpful commands. Refer to the August, 1983 issue #43 of REMark for the program options and abstract.

Remember, if ordering soft-sectored, add the "-37" to the part number, i.e. 885-8018-37.

The documentation to BIGED (BIGED.DOC) was not included with the last release of the product. If you will write or call Nancy Strunk, Heath Users' Group, Hilltop Road, St. Joseph, MI, (616) 982-3463, she will send you a listing of BIGED.DOC.

BIGED is a program that works with text files which are too large to be edited by FAST EDDY directly because of memory limitations. See issue #43 of REMark for details.

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

885-1115[-37] Navigational Program
885-1118[-37] MBASIC Payroll
885-1119[-37] BH BASIC Support
885-1120[-37] HDOS "WHEW" Utilities
885-1122[-37] MicroNET Connection

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

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

HUG Price List

The following HUG Price List contains a list of all products not included in the HUG Software Catalog. For a detailed abstract of these products refer to the issue of REMark specified.

Part Number	Description of Product	Selling Price	REMark Issue
HDOS			
885-1022 [-37]	HUG Editor (ED) Disk H8/H89	$ 20.00	44
885-1029 [-37]	Disk II Games 1 H8/H89	$ 18.00	40
885-1030 [-37]	Disk III Games 2 H8/H89	$ 18.00	45
885-1038 [-37]	Wise on Disk H8/H89	$ 18.00	42
885-1042 [-37]	PILOT on Disk H8/H89	$ 19.00	42
885-1044 [-37]	Utilities Disk VI	$ 18.00	43
885-1055 [-37]	MBASIC Inventory Disk H8/H89	$ 30.00	44
885-1060 [-37]	Disk VII H8/H89	$ 18.00	40
885-1062 [-37]	Disk VIII H8/H89 (2 Disks)	$ 25.00	40
885-1064 [-37]	Disk IX H8/H89	$ 18.00	42
885-1066 [-37]	Disk X H8/H89	$ 18.00	45
885-1067 [-37]	Disk XI H8/H89 Games	$ 18.00	40
885-1071 [-37]	MBASIC SmBusPk H8/H19/H89	$ 75.00	41
885-1078 [-37]	HDOS Z80 Assembler	$ 25.00	42
885-1079 [-37]	Page Editor PAGED	$ 25.00	43
885-1083 [-37]	Disk XVI Misc. Utilities	$ 20.00	43
885-1086 [-37]	Tiny HDOS Pascal H8/H89	$ 20.00	40
885-1088 [-37]	Disk XVII MBASIC Graphic Games	$ 20.00	45
885-1089 [-37]	Disk XVIII Misc H8/H89	$ 20.00	41
885-1090 [-37]	Disk XIX Utilities H8/H89	$ 20.00	41

Part Number	Description of Product	Selling Price	REMark Issue
885-1092 [-37[	Relocating Debug Tool H8/H89	$ 30.00	44
885-1093 [-37]	Dungeons and Dragons H8/H89	$ 20.00	43
885-1096 [-37]	MBASIC Action Games H8/H89	$ 20.00	45
885-1097 [-37]	MBASIC Quiz Disk H8/H89	$ 20.00	41
885-1107 [-37]	HDOS Data Base System H8/H89	$ 30.00	42
885-1108 [-37]	HDOS MBASIC Data Base System	$ 30.00	41
885-1111 [-37]	HDOS MBASIC Games H8/H89	$ 20.00	44
885-1112 [-37]	Graphics Games Disk	$ 20.00	43
885-1113 [-37]	HDOS Fast Action Games H8/H89	$ 20.00	44
885-1121	Hard Sectored Support Package	$ 30.00	37
885-1122	MicroNET Connection	$ 16.00	37
885-1123	XMET Robot Cross Assembler	$ 20.00	40
885-1124	HUGMAN & Movie Animation Pkg	$ 20.00	41
885-1125	MAZEMADNESS	$ 20.00	41
885-1126	HDOS UTILITIES by PS:	$ 20.00	42
885-1127	HDOS Soft Sector Support Pkg	$ 30.00	45
885-1127 [-37]	HDOS Soft Sector Support Pkg	$ 30.00	45
885-1130	HDOS Star Battle	$ 20.00	45
885-8015	TEXTSET Formatter	$ 30.00	42
885-8016	Morse Code Transceiver Ver 2.0	$ 20.00	41
885-8017	HDOS Programmers Helper	$ 16.00	42
885-8021	HDOS Student's Statistics Pkg	$ 20.00	44
885-8022	HDOS SHAPES	$ 16.00	45
CP/M			
885-1211 [-37]	Sea Battle	$ 20.00	36
885-1222 [-37]	Adventure	$ 10.00	36
885-1223 [-37]	HRUN HDOS Emulator	$ 40.00	37
885-1224 [-37]	MicroNET Connection	$ 16.00	37
885-1225 [-37]	Disk Dump and Edit Utility (DDEU)	$ 30.00	38
885-1226 [-37]	CP/M Utilities by PS:	$ 20.00	38
885-1227 [-37]	CP/M Cassino Graphic Games	$ 20.00	38
885-1228 [-37]	CP/M Fast Action Games	$ 20.00	39
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 Reference Utilities for MBASIC	$ 20.00	43
885-3003 [-37]	ZTERM Modem Package	$ 20.00	36
885-8012 [-37]	Modem Appl. Effector (MAPLE)	$ 35.00	36
885-8018 [-37]	FAST EDDY Text Editor and BIG EDDY	$ 20.00	43
885-8019 [-37]	DOCUMAT Formatter and DOCULIST	$ 20.00	43
885-8020 [-37]	CP/M RF Computer Aided Design	$ 30.00	44
885-8023 [-37]	CP/M 85 MAPLE	$ 35.00	45
ZDOS			
885-3004-37	ZBASIC Graphic Games Disk	$ 20.00	37
885-3005-37	ZDOS ETCHDUMP	$ 20.00	39
MISCELLANEOUS			
885-0004	HUG 3-Ring Binder	$ 5.75	
885-4001	REMark VOLUME 1, Issues 1-13	$ 20.00	
885-4002	REMark VOLUME 2, Issues 14-23	$ 20.00	
885-4003	REMark VOLUME 3, Issues 24-35	$ 20.00	
885-4600	Watzman/HUG ROM	$ 45.00	41

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

Heath/Zenith Users' Group

Call For Hero I Articles

REMark Magazine and the Heath Users' Group are very interested in your experiences with your Hero I Robot.

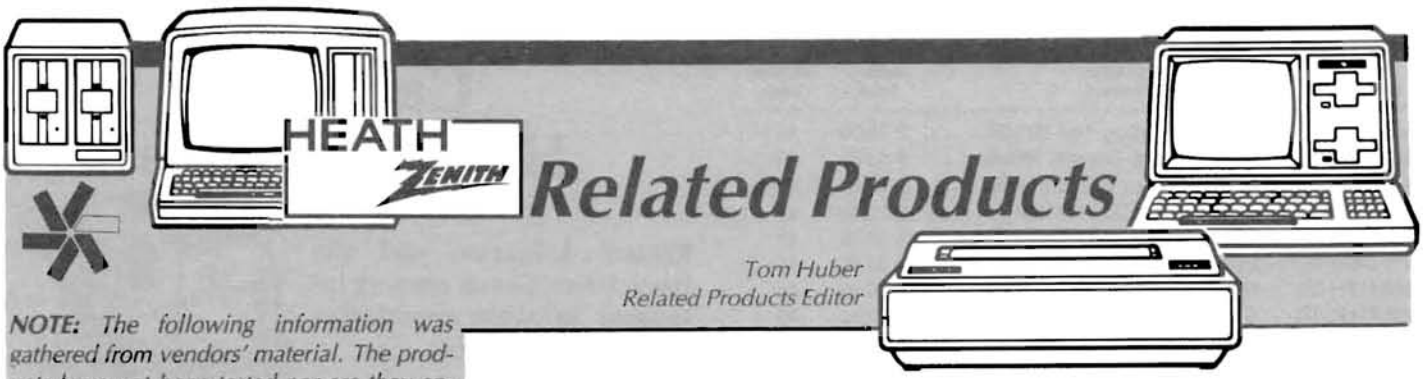
Share these experiences with other HUGgies, submit your ideas today!

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

maybe now is the time to join the National Heath Users' Group.

- *You will receive a copy of **REMark** filled with new and exciting articles and programs each month.*
- *You will also receive access to the HUG library filled with a large variety of programs.*

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



Related Products

Tom Huber
Related Products Editor

NOTE: The following information was gathered from vendors' material. The products have not been tested nor are they endorsed by HUG. We are not responsible for errors in descriptions or prices.

Home Budget Manager

This program, focusing of the family checkbook, allows entry of deposits and withdrawals with a self-prompting list of data items. The program keeps track of checks, deposits, dates, payees, account names, interest payments, and tax deductions. Fully menu- driven, Home Budget Manager can print all or part of the checkbook register, checks of preprinted forms, reconcile the account with the bank's monthly statement, and provides listing and search features for access to information in the data base. Available for either 48K or 64K under BASIC and CP/M. State RAM size and disk configuration (hard/soft, single/double sided/density) when ordering.

Vendor: ByteSize Concepts, Inc.
18101 Euclid Ave.
Cleveland, OH 44112
(216) 486-1114
Price: $49.95 + $2.75 S&H
(Ohio residents add 6.5% sales tax)

PolyLibrarian for Z-DOS

PolyLibrarian™, the Object Module Librarian, is a productivity enhancement software package for programmers using MS-DOS (Z-DOS or PC-DOS). It organizes related object code modules (.OBJ files) into a single library (.LIB file) which can then be used by LINK (an MS-DOS utility) to select only those modules needed to construct an executable program (.EXE file). The program works with any compiler or assembler that uses LINK and provides mainframe librarian functions to decrease software development time. Requires Z-DOS or MS-DOS and 128K RAM.

Vendor: Polytron Corporation
P.O. Box 787, DS 2-219
Hillsboro, OR 97123
(1-800) 547-4000
Price: $99.00 + $1.25 S&H

Newline offers a Z-100 HDOS and DENSE-PAC

Newline's Z-100 HDOS configures the Z-100 to appear as a Z-89 and supports the standard Heath device drivers (with this upgrade), ROM's, 2 ms. TICCNT timer, and 48- and 96-tpi single- and double- sided soft-sectored disk drives. Most application programs will run without difficulty (except those that directly interface with '89 I/O ports). This is not an emulator, it is a version of HDOS; the special Newline boot track boots directly on the Z-100—no other operating system is required. As an added incentive and for a limited time (contact the vendor), Newline is offering its Z- 100 HDOS version of TEXT PROCESSOR free with this package.

DENSE-PAC is a program that compresses (and/or encrypts) data files by applying one of five algorithms. ASCII or binary files can be encypted for security as well. Average compression is 30 to 40 percent. Single drive operation is fully supported including disk swapping.

Contact the vendor for a free 40-page catalog of '89 and '100 software.

Vendor: Newline Software
P.O. Box 402
Littleton, MA 01460
(617) 486-8535
Prices: Z-100 HDOS $79.95
DENSE-PAC $29.95

Audio Training Courses for Software Packages

Sound Training is an audio cassette based course in the use of major software packages. Available now are courses for WordStar, MailMerge, CP/M, and WordCraft. Planned are Multiplan, Lotus 1-2- 3, PC-DOS, dBASE II, SuperCalc, and Personal Pearl. For more information, contact Jeanne Nash at Information System Resources.

Vendor: Information System Resources
1444 Balsam St.
St. Paul, MN 55122
(612) 452-7913
Price: $69.95 per course kit

Mail Master

Mail Master is a mailing list manager program that will run on '89 and '100 Computers under CP/M(-85). It supports special codes for selecting special audiences, full names, addresses, and phone numbers. Sorting is by name, zip code, or special code. The program is offered in English, French, and Spanish. Requires CP/M or CP/M-85, and 48K RAM. Specify hard or soft-sectored drives.

Vendor: M.S.T.C.
75-23 Main Street
P.O. Box 111
Flushing, NY 11367
(212) 263-9100
Price: $59.95
(NY residents add sales tax)

Int Jet Printer and Z-100 Driver Features up to 125 Colors

Advanced Color Technology is offering a line of color ink jet printers, and Z-100 and IBM drivers for reproduction of color displays produced by either machine. The driver is run by pressing the F12 function key on the Z-100 or the (shift) PrtScn key on the PC. Z-DOS 1.01, version 1.25 (or above) or PC-DOS version 2.0 or above is required for support. The utility supports either the ACT I or ACT II/Chromajet™ printer and allows the user to select the number of copies, black/white reversal, and bi-directional printing. The ACT II printer has an operating noise level of 50dBA, 125 usable color shades, solid area fill, and high speed (90 seconds for an 8-1/2 x 11" image, regardless of the complexity) on plain paper or for overhead transparencies using the IJT/100 Ink Jet Transparency Film from Polaroid. Resolution is selectable from 40 to 140 dots per inch horizontal and 85 dots per inch vertical. Contact the vendor for full details.

Vendor: Advanced Color Technology, Inc.
21 Alpha Road
Chelmsford, MA 01824
(617) 256-1222
Prices: ACT II printer $6400.00
Z-100 or PC driver . . . $69.95

Programming For Fun

George Hill, Jr.
1810 Ruth Dr.
Indianapolis, IN 46240

I have been an Amateur Radio operator since 1970 and I have thoroughly enjoyed building radio equipment, test equipment and operating my radio station. When the Heathkit H-8 and the H/Z-89 computers first appeared in the Heathkit Catalog I was curious, but wondered what I would ever do with one of them.

Finally, early in 1981, my curiosity got the best of me and I ordered and built an H-88. Like my ham radio station, I wasn't satisfied with my setup and I had to up-grade it to an H/Z-89, add another drive, CP/M, MBASIC, and an MX-80 printer with Graftrax. I could never justify the cost of my ham radio station anymore than I could justify my computer system other than they are both super hobbies.

Now my enjoyment is in writing programs. A lot of them are like some of my building projects for Amateur Radio, the joy is in the building and if someone else can use and enjoy the results, so much the better.

The H/Z-89 is a great computer and its features leave areas of discovery that will take me years to explore. I have already enjoyed almost two years of exploration in learning first HDOS and then CP/M, Benton Harbor BASIC and then MBASIC and now I have just begun to study the Assembly Language course.

I know that the many things I have discovered are old hat to many, but for the novice programmer or someone not familiar with all the features of the H/Z-89, I would like to share a program that uses these features as a demonstration. The program listing can be used as a study to incorporate some of these features into MBASIC programs.

Some of these features have been described in articles in REMark and other publications. Other ideas have been gleaned from the various manuals and programs of others, but much has been learned from trial and error, and from experimenting.

The program, which I call "DEMOMBAS", is an extension of a program that consists of the lines from 10 to 130 that I use to start every MBASIC program that I write. Anyone who wishes can save these lines as a "STARTMB" program of their own. Of course, they will need to change the lines of identification with their own information. The "STARTMB" program is not a new idea, but it is a good one. This way, you have established a set of variables that remain the same in all your programs and it makes it easier to incorporate the graphic features of the H/Z-89 into your programs. While the use of graphics in your programs uses some of your memory, they dress up your displays on the screen and make your program more attractive and easier to follow.

The program is written in BASIC 80 of CP/M, but can be used with HDOS MBASIC with some changes. The variable assignments will work in either operating system, but the decoding of the function keys is different. In HDOS you can decode the function by the statement, "IF (RIGHT$(X$,1)="S" THEN GOSUB 1000". In CP/M BASIC 80 they decode directly "IF X$="S" THEN GOSUB 1000". I have assigned the variables names that can be easily identified as to their function, their names can be changed to ones that the programmer can best remember.

I have been a printer for 40 years, so I like a nice appearance on the screen when the program runs. I like rules, borders and structured input screens in my programs. The H/Z-89 graphics are great for this and you can create them using the variable assignments of this program. If you have "ED-A-SKETCH", a very good program from the "Software Toolworks", making the graphics is really easy and you can save them as an MBASIC subroutine. You can use a form to structure your input of data in a program by using cursor addressing made easy by the Define function feature of MBASIC.

```
DEF FN Y$(X,Y)=ESC$+"Y"+CHR$(31+"Y")+CHR$(31+"X")
```

Where ESC$=CHR$(27), the Escape Key. This function is defined in line 130 of the program. To use it, first decide where you want the cursor to go on the screen. "X" equals the number of the column from the left and "Y" the number of the row counting from the top of the screen. The screen has 80 columns and 24 rows, not counting the 25th line that the H/Z-89 uses for labeling or an instruction that you want to leave on the screen as a reminder. To place the cursor in the approximate center of the screen and print the letter "X", you would say:

```
100 X=40:Y=12:PRINT FN Y$(X,Y)"X"
```

You can address and print on the 25th line by using this function and the variables we have setup.

```
260 PRINT CLS$        'Clear the screen
270 PRINT CSAV$;EN$   'Save Cursor position, enable 25th line
280 X=1:Y=25          'Set values to coordinates of 25th line
290 PRINT FN Y$(X,Y) 'Call function addressing the cursor
300 PRINT RV$;        'Enter reverse video
310 PRINT "    f1    f2    f3    f4    f5    etc."
320 PRINT CRET$;RX$  'Return cursor to saved position and
                      'exit from reverse video.
```

The cursor will return to the home position after printing the 25th line. Before you exit the program be sure to disable the 25th line to erase it.

```
3160 PRINT DIS$:END 'Disable the 25th line and end program
```

This method of cursor addressing is used extensively throughout this demonstration program. Using the Define Function makes it real easy. Try it.

Another little jewel I like is the INPUT$(n) statement. If (n) is "1" the computer will stop and look for a key to be pressed. This allows you to stop the program while the user reads a screen of instruction, then allows them to touch any key to resume program execution. Of course you can decode their choice and branch the program according to the key they touched. You can also make (n) any number and the computer will look for "n" number of keys to be pressed before it will resume. The input from the keyboard is not echoed on the screen, so you could set up a "password" routine to run a program with this statement. Experiment with it. I am sure you will find this statement useful.

I have used a lot of remarks throughout the program so that you

```
5 REM *** DEMOMBAS.BAS--CP/M--MBASIC--DEMONSTRATION AND START PROGRAM ***
10 REM *** PROGRAM BY GEORGE HILL, JR., INDIANAPOLIS, IN ***
20 REM *** HEATHKIT H89 COMPUTER--MBASIC--CPM ***
30 CLEAR
40 ESC$=CHR$(27):CLS$=ESC$+"E" ' ESC$=ESCAPE  CLS$=CLEAR SCREEN
50 GR$=ESC$+"F":GX$=ESC$+"G" ' GR$=ENTER GRAPHICS  GX$=EXIT GRAPHICS
60 RV$=ESC$+"p":RX$=ESC$+"q" ' RV$=ENTER REVERSE VIDEO  RX$=CANCELS REVERSE
70 BELL$=CHR$(7):COFF$=ESC$+"x5":CON$=ESC$+"y5":CSAV$=ESC$+"j"
80 CBLK$=ESC$+"x4":CLIN$=ESC$+"y4"
90 REM *** BELL$=RINGS BELL  COFF$=TURN OFF CURSOR  CON$=TURN CURSOR ON
100 REM *** CSAV$=SAVE CURSOR POSITION  CRET$=RETURN CURSOR TO SAVED POSITION
110 REM *** EN$=ENABLE 25TH LINE  DIS$=DISABLES THE 25TH LINE
120 CRET$=ESC$+"k":EN$=ESC$+"x1":DIS$=ESC$+"y1"
130 DEF FN Y$(X,Y)=ESC$+"Y"+CHR$(31+Y)+CHR$(31+X)
140 REM *** The DEFINE FN statement here sets up a coordinate method of
150 REM *** cursor addressing. X becomes the column address and Y the
160 REM *** row address.  Give X and Y the column and row value then call
170 REM *** the define statement by a PRINT DEF FN Y$(X,Y) to move the cursor
180 REM *** to the desired position. To address the 25th line, first save the
190 REM *** cursor position PRINT CSAV$, then PRINT EN$ to enable the 25th line.
200 REM *** Assign the value of X=1:Y=25 and use the define statement above.
210 REM *** Print your 25th line, then PRINT CRET$ to return the cursor to the
220 REM *** previously saved position.
230 REM *** The function keys f1,f2,f3,f4,f5,erase,blue,red and white can be
240 REM *** decoded as follows--f1="S", f2="T", f3="U", f4="V", f5="W"
250 REM *** ERASE="J", Blue="P", Red="Q", and White="R"
260 PRINT COFF$;CLS$            ' Clear Screen and turn off cursor
270 PRINT CSAV$;EN$      ' Save Cursor position and enable 25th line
280 X=1:Y=25             ' Set values to coordinates of 25th line
290 PRINT FN Y$(X,Y)     ' Call function to address cursor
300 REM                  ' Next line enters reverse video and prints 25th line
310 PRINT RV$;"       f1      f2      f3      f4      f5     QUIT   Blue   Red
  White           "
320 PRINT CRET$;RX$      ' Return cursor to saved position and exit reverse
330 GOSUB 2220
340 GOSUB 2310
350 X=20:Y=12:PRINT FN Y$(X,Y);
360 INPUT "CHOOSE A  FUNCTION KEY TO PUSH--",A$
370 IF A$="S" THEN GOSUB 470    ' Decodes f1 function key
380 IF A$="T" THEN GOSUB 730    ' Decodes f2 function key
390 IF A$="U" THEN GOSUB 890    ' Decodes f3 function key
400 IF A$="V" THEN GOSUB 1330   ' Decodes f4 function key
410 IF A$="W" THEN GOSUB 1710   ' Decodes f5 function key
420 IF A$="P" THEN GOSUB 1880 ' Decodes blue function key
430 IF A$="Q" THEN GOSUB 2050' Decodes white function key
440 IF A$="R" THEN GOSUB 2570   ' Decodes red function key
450 IF A$="J" THEN GOTO 3000    ' Decodes erase key
460 GOTO 350
470 WIDTH 255
480 PRINT CHR$(27);"w";CHR$(27);"H";CHR$(27);"J";CHR$(27);"G";
490 PRINT CHR$(27);"q";CHR$(27);"F";CHR$(27);"Y%'faaaaaaaaaaaaaaaaaa";
500 PRINT "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac";CHR$(27);"Y&'";
510 PRINT "'";CHR$(27);"Y&g'";CHR$(27);"Y''`";CHR$(27);"G";CHR$(27);"Y";
520 PRINT "'+You have decoded function key f1 to call a sub-routine";
530 PRINT CHR$(27);"F";CHR$(27);"Y'g`";CHR$(27);"Y(''   ";CHR$(27);"G";
540 PRINT CHR$(27);"Y(2that printed this box and this message.";
550 PRINT CHR$(27);"F";CHR$(27);"Y(g`";CHR$(27);"Y(n ";CHR$(27);"Y)''`";
```

can find where the different features of the H/Z-89 are addressed by the MBASIC variables. So have fun dressing up your MBASIC programs. Of course, use the FRE(0) statement to check your memory left. I'm headed down to the Heathkit store to get myself another 16K of memory. Have fun programming.

```
1050 PRINT
1060 PRINT RV$"TO QUIT INPUT TYPE 'QUIT' FOR NAME"RX$:A=1
1070 PRINT CON$:Y=9
1080 FOR J=1 TO 10
1090 X=3:PRINT FN Y$(X,Y);
1100 INPUT "",N$(A)
1110 IF LEFT$(N$(A),1)="Q" THEN GOTO 1240
1120 X=19:PRINT FN Y$(X,Y);
1130 INPUT "",AD$(A)
1140 X=39:PRINT FN Y$(X,Y);
1150 INPUT "",CTY$(A)
1160 X=54:PRINT FN Y$(X,Y);
1170 INPUT "",ST$(A)
1180 X=62:PRINT FN Y$(X,Y);
1190 INPUT "",ZIP$
1200 X=70:PRINT FN Y$(X,Y);
1210 INPUT "",PH$
1220 A=A+1:Y=Y+1
1230 NEXT J
1240 X=1:Y=22:PRINT FN Y$(X,Y);RV$" TOUCH ANY KEY TO RESUME THE PROGRAM "
1250 PRINT RX$
1260 Z$=INPUT$(1)
1270 PRINT COFF$;CLS$:RETURN
1280 PRINT CSAV$;EN$
1290 X=1:Y=25
1300 PRINT FN Y$(X,Y);RV$
1310 PRINT "    BOX  GRAPHICS  INPUT  PRINT  CURSOR  QUIT  ADDRESS
VARIABLES  MENU     "
1320 PRINT CRET$;RX$:RETURN
1330 REM *** DECODED f4 KEY TO PRINT IN A FORM ON THE SCREEN ***
1340 X=1:Y=10:PRINT FN Y$(X,Y);
1350 PRINT GR$CLS$STRING$(79,97)
1360 PRINT GX$
1370 PRINT:PRINT TAB(10)"Your choice of the f4 function key has brought you to a"
1375 PRINT
1380 PRINT TAB(10)"demonstration of using a graphic form on the screen for"
1385 PRINT
1390 PRINT TAB(10)"printing data on the screen in a format that is easily"
1395 PRINT
1400 PRINT TAB(10)"understood.  The information could be read into memory"
1405 PRINT
1410 PRINT TAB(10)"from a disk file and then displaged in the form."
1415 PRINT
1420 PRINT TAB(10)"To keep this program self-contained we will use what you typed"
1425 PRINT
1430 PRINT TAB(10)"during the input demonstration called by the f3 key."
1435 PRINT
1440 PRINT GR$;FOR I=1 TO 79:PRINT "a";:NEXT I
1450 PRINT GX$
1460 PRINT:PRINT TAB(10)RV$" TOUCH ANY KEY TO RESUME PROGRAM "RX$
1470 Z$=INPUT$(1)
1480 GOSUB 2730
1490 A=1
1500 Y=9
1510 FOR J=1 TO 10
1520 X=3:PRINT FN Y$(X,Y);
1530 PRINT N$(A)
```

```
560 PRINT CHR$(27);"Y)g`";CHR$(27);"Y*'`";CHR$(27);"G";CHR$(27);"Y*+Th";
570 PRINT "e box does not use the variables I have created at the";
580 PRINT CHR$(27);"F";CHR$(27);"Y*g`";CHR$(27);"Y+'`";CHR$(27);"G";
590 PRINT CHR$(27);"Y+.beginning of this program because the box and t";
600 PRINT "his";CHR$(27);"F";CHR$(27);"Y+g`";CHR$(27);"Y,'`";
610 PRINT CHR$(27);"G";CHR$(27);"Y,,printed message were made using a ";
620 PRINT "very fine program by";CHR$(27);"F";CHR$(27);"Y,g`";
630 PRINT CHR$(27);"Y-'`";CHR$(27);"G";CHR$(27);"Y-+Software Toolworks";
640 PRINT " called Ed-A-Sketch";CHR$(27);"Cand then saved as a";
650 PRINT CHR$(27);"F";CHR$(27);"Y-g`";CHR$(27);"Y.'`";CHR$(27);"G";
660 PRINT CHR$(27);"Y.=sub-routine in MBasic.";CHR$(27);"F";CHR$(27);
670 PRINT "Y.g`";CHR$(27);"Y/'`";CHR$(27);"Y/g`";CHR$(27);"Y0'eaaaaaaaa";
680 PRINT "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaad";
690 PRINT CHR$(27);"G";CHR$(27);"q"
700 X=1:Y=23:PRINT FN Y$(X,Y)RV$"HIT ANY KEY TO RESUME PROGRAM"RX$
710 Z$=INPUT$(1)
720 PRINT CLS$:RETURN
730 REM *** GOSUB DEMONSTRATING CURSOR ADDRESSING AND GRAPHICS
740 X=35:Y=11
750 PRINT RV$;GR$;CLS$;FN Y$(X,Y)"r        ";CHR$(95)
760 Y=12:PRINT FN Y$(X,Y)"          "
770 Y=13:PRINT GX$;FN Y$(X,Y)"    f2    "
780 Y=14:PRINT FN Y$(X,Y)"          "
790 Y=15:PRINT RX$GR$;FN Y$(X,Y)CHR$(95)RV$"        "RX$"r"
800 PRINT GX$
810 PRINT:PRINT TAB(20)"You have decoded the f2 function key"
820 PRINT
830 PRINT TAB(8)"IN THE PROCESS YOU HAVE SEEN A GRAPHIC KEY PRINTED ON THE SCREEN"
840 PRINT:PRINT TAB(10)"USING CURSOR ADDRESSING AND VARIABLES TO GO INTO AND OUT OF"
850 PRINT:PRINT TAB(20)"THE HEATH H-89 GRAPHICS AND REVERSE VIDEO."
860 PRINT:PRINT TAB(25)RV$"HIT ANY KEY TO RESUME PROGRAM"RX$
870 Z$=INPUT$(1)        ' This statement will cause the program to halt
                        ' and will resume when any key is pressed
880 PRINT CLS$:RETURN
890 REM *** DECODED f3 KEY TO PRINT A FORM TO USE FOR INPUT ***
900 PRINT FN Y$(X,Y);
910 PRINT GR$CLS$STRING$(79,97)
920 PRINT GX$
930 PRINT:PRINT TAB(10)"Your choice of the f3 function key has brought you to a"
935 PRINT
940 PRINT TAB(10)"demonstration of using a graphic form on the screen for"
945 PRINT
950 PRINT TAB(10)"inputing data to the computer.  This data could be output"
955 PRINT
960 PRINT TAB(10)"to a disk file, recalled as needed and printed in the same"
965 PRINT
970 PRINT TAB(10)"graphic form.  As a demonstration, however, you will input"
975 PRINT
980 PRINT TAB(10)"to an array of variables in memory.  To demonstrate recalling"
985 PRINT
990 PRINT TAB(10)"and printing into the form on the screen you will use f4 key."
995 PRINT
1000 PRINT GR$:FOR I=1 TO 79:PRINT "a";:NEXT I
1010 PRINT GX$
1020 PRINT:PRINT TAB(10)RV$" TOUCH ANY KEY TO RESUME PROGRAM "RX$
1030 Z$=INPUT$(1)
1040 GOSUB 2730
1540 IF LEFT$(N$(A),1)="Q" THEN GOTO 1670
1550 X=19:PRINT FN Y$(X,Y);
1560 PRINT AD$(A)
1570 X=39:PRINT FN Y$(X,Y);
1580 PRINT CTY$(A)
1590 X=54:PRINT FN Y$(X,Y);
1600 PRINT ST$(A)
1610 X=62:PRINT FN Y$(X,Y);
1620 PRINT ZIP$
1630 X=70:PRINT FN Y$(X,Y);
1640 PRINT PH$
1650 A=A+1:Y=Y+1
1660 NEXT J
1670 X=1:Y=22:PRINT FN Y$(X,Y);RV$" TOUCH ANY KEY TO RESUME THE PROGRAM "
1680 PRINT RX$
1690 Z$=INPUT$(1)
1700 PRINT CLS$:RETURN
1710 REM *** DECODES THE f5 FUNCTION KEY ***
1720 PRINT CLIN$;CON$
1730 PRINT CLS$:X=1:Y=10
1740 PRINT FN Y$(X,Y);
1750 PRINT TAB(10)"You have decoded the f5 function key and you will notice that"
1760 PRINT TAB(10)"the cursor has disappeared.  When you examine the variables"
1770 PRINT TAB(10)"you will see that we have assigned COFF$ as a variable to"
1780 PRINT TAB(10)"to turn off the cursor and CON$ as a variable to turn it on."
1790 PRINT TAB(10)"CLIN$ will change your cursor to a line, CBLK$ to a block."
1800 PRINT:PRINT TAB(10)COFF$"The cursor is off, touch any key"
1810 Z$=INPUT$(1)
1820 PRINT:PRINT TAB(10)CON$;CBLK$"The cursor is on and it is a block"
1830 PRINT:PRINT TAB(10)"Touch any key";Z$=INPUT$(1)
1840 PRINT:PRINT TAB(10)CLIN$"The cursor is now a line, touch any key to resume"
1850 Z$=INPUT$(1)
1860 PRINT CBLK$;COFF$
1870 PRINT CLS$:RETURN
1880 REM *** CURSOR ADDRESSING ***
1890 GOSUB 2220
1900 X=32:Y=4:PRINT FN Y$(X,Y);RV$" CURSOR ADDRESSING "RX$
1910 X=8:Y=6:PRINT FN Y$(X,Y)"The method used in this program to move the cursor about the screen"
1920 Y=7:PRINT FN Y$(X,Y)"is a user defined function."
1930 X=18:Y=9:PRINT FN Y$(X,Y)"DEF FN Y$(X,Y)=ESC$+'Y'+CHR$(31+Y)+CHR$(31+X)"
1940 X=8:Y=11:PRINT FN Y$(X,Y)"The DEF FN statement here sets up a coordinate method of"
1950 Y=12:PRINT FN Y$(X,Y)"cursor addressing.  X becomes the column address and Y the row"
1960 Y=13:PRINT FN Y$(X,Y)"address.  Give X and Y the column and row value then call the"
1970 Y=14:PRINT FN Y$(X,Y)"DEF STATEMENT by a PRINT FN Y$(X,Y) to move the cursor"
1980 Y=15:PRINT FN Y$(X,Y)"to the desired position.  To address the 25th line, first save the"
1990 Y=16:PRINT FN Y$(X,Y)"cursor position with PRINT CSAV$, then PRINT FN$ to enable the"
2000 Y=17:PRINT FN Y$(X,Y)"25th line. Print your 25th line, then PRINT CRET$ to return the cursor"
2010 Y=18:PRINT FN Y$(X,Y)"to the previously saved position."
2020 Y=19:PRINT FN Y$(X,Y)"TO ERASE THE 25TH LINE PRINT DIS$ BEFORE YOU END THE PROGRAM."
2030 GOSUB 700
2040 RETURN
2050 REM *** VARIABLES ***
2060 GOSUB 2220
2070 X=21:Y=6:PRINT FN Y$(X,Y);RV$" A LIST OF VARIABLES AND THEIR FUNCTIONS "
RX$
2080 PRINT GR$
2090 PRINT TAB(3)"`"TAB(8)"ESC$ =ESCAPE KEY"TAB(42)"COFF$=CURSOR TURNED OFF"
```

```
2100 PRINT TAB(3)"'"TAB(8)"CLS$ =CLEAR SCREEN"TAB(42)"CON$ =CURSOR TURNED ON"
2110 PRINT TAB(3)"'"TAB(8)"GR$  =ENTER GRAPHICS MODE"TAB(42)"CBLK$=BLOCK CURSOR"
2120 PRINT TAB(3)"'"TAB(8)"GX$  =EXIT GRAPHICS MODE"TAB(42)"CLIN$=LINE CURSOR"
2130 PRINT TAB(3)"'"TAB(8)"RV$  =ENTER REVERSE VIDEO"TAB(42)"CSAV$=SAVE CURSOR POSITION"
2140 PRINT TAB(3)"'"TAB(8)"RX$  =EXIT REVERSE VIDEO"TAB(42)"CRET$=RETURN CURSOR TO SAVE"
2150 PRINT TAB(3)"'"TAB(8)"BELL$=RING BELL"TAB(48)"POSITION"
2160 PRINT TAB(3)"'"TAB(8)"DEF FN Y$(X,Y)=DEFINE FUNCTION"TAB(42)"EN$=ENABLE 25TH LINE"
2170 PRINT TAB(3)"'"TAB(11)"TO USE X,Y COORDINATES TO"TAB(42)"DIS$=DISABLE 25TH LINE"
2180 PRINT TAB(3)"'"TAB(11)"ADDRESS THE CURSOR"TAB(42)"H$=CURSOR TO HOME POSITION"
2190 PRINT GX$
2200 GOSUB 700
2210 RETURN
2220 REM *** PRINT BOX GOSUB ***
2230 PRINT :SP$=SPACE$(75)
2240 PRINT CLS$GR$
2250 PRINT TAB(3)"f";STRING$(75,97);"c"
2260 FOR J=1 TO 18
2270 PRINT TAB(3)"'";SP$;"'"
2280 NEXT J
2290 PRINT TAB(3)"e";STRING$(75,97);"d"
2300 PRINT GX$:RETURN
2310 REM *** PROGRAM DOCUMENTATION ***
2320 X=8:Y=4:PRINT FN Y$(X,Y)"This is a program to acquaint you with";
2330 PRINT " the versatility of the"
2340 Y=5:PRINT FN Y$(X,Y)"Heathkit H-89 Computer and its graphics";
2350 PRINT " using a MBasic program."
2360 Y=7:PRINT FN Y$(X,Y)"You will notice the label line at the bottom";
2370 PRINT " of the screen. It"
2380 Y=8:PRINT FN Y$(X,Y)"identifies the function keys across the top";
2390 PRINT " row of the keyboard."
2400 Y=10:PRINT FN Y$(X,Y)"You will call different parts of this program";
2410 PRINT " by choosing one of"
2420 Y=11:PRINT FN Y$(X,Y)"these keys and touching return.  I suggest that";
2430 PRINT " you start with"
2440 Y=12:PRINT FN Y$(X,Y)"the f1 key and work your way across the keyboard";
2450 PRINT ", but skip the"
2460 Y=13:PRINT FN Y$(X,Y)"Erase Key until you are ready to exit the program."
2470 Y=15:PRINT FN Y$(X,Y)"A print-out of this program will guide you";
2480 PRINT " in writing other MBasic"
2490 Y=16:PRINT FN Y$(X,Y)"Programs using some of these features.  Menu";
2500 PRINT " driven programs such"
2510 Y=17:PRINT FN Y$(X,Y)"As this one, makes it easy for the computer";
2520 PRINT " user to run the program."
2530 Y=19:PRINT FN Y$(X,Y)"WATCH THE LABEL ON THE 25TH LINE CHANGE";
2540 PRINT " WHEN YOU TOUCH A KEY—"
2550 GOSUB 700
2560 GOSUB 1200
2570 GOSUB 2220
2580 X=7:Y=4:PRINT FN Y$(X,Y)"To guide you through the available features of the Heath-Zenith H-89"
2590 Y=5:PRINT FN Y$(X,Y)"Computer using MBasic we call your attention to the row of function"
2600 Y=6:PRINT FN Y$(X,Y)"keys across the top of the keyboard. The running of this program is"
2610 Y=7:PRINT FN Y$(X,Y)"a demonstration of the use of these keys."
2620 X=38:Y=9:PRINT FN Y$(X,Y);RV$" MENU ";RX$
2630 X=7:Y=11:PRINT FN Y$(X,Y)"f1 key—Box with message created with the Ed-A-Sketch program"
2640 Y=12:PRINT FN Y$(X,Y)"f2 key—Using variables to print graphics and reverse video"
2650 Y=13:PRINT FN Y$(X,Y)"f3 key—Inputing data by filling in a form printed on the screen"
2660 Y=14:PRINT FN Y$(X,Y)"f4 key—Calling data from memory and printing it into a form on screen"
2670 Y=15:PRINT FN Y$(X,Y)"f5 key—Cursor manipulating"
2680 Y=16:PRINT FN Y$(X,Y)"Blue  —Cursor addressing"
2690 Y=17:PRINT FN Y$(X,Y)"WHITE —List Variable assignments used in program"
2700 Y=18:PRINT FN Y$(X,Y)"RED   —Prints this Menu"
2710 GOSUB 700
2720 RETURN
2730 WIDTH 255
2740 PRINT CHR$(27);"w";CHR$(27);"H";CHR$(27);"J";CHR$(27);"G";
2750 PRINT CHR$(27);"q";CHR$(27);"F";CHR$(27);"Y% fa
2760 PRINT "
2770 PRINT CHR$(27);"Y& '     NAME      '      ADDRESS      '      CITY ";
2780 PRINT "    ' STATE ' ZIP  ' ";CHR$(27);"C PHONE   '";CHR$(27);"Y'";
2790 PRINT " v
2800 PRINT "ab                 ";CHR$(27);"Y( '                '     ";
2810 PRINT "                '           '       '       '          '";
2820 PRINT CHR$(27);"Y) '                '                  '          ";
2830 PRINT "    '       '       '          '";CHR$(27);"Y* '          ";
2840 PRINT "     '                '                '       '       ' ";
2850 PRINT "         '";CHR$(27);"Y+ '                '                  ";
2860 PRINT "  '              '       '       '          '";CHR$(27);"Y";
2870 PRINT ", '              '                '                '     ";
2880 PRINT "  '       '          '";CHR$(27);"Y- '              '     ";
2890 PRINT "                '           '       '       '          '";
2900 PRINT CHR$(27);"Y. '                '                  '          ";
2910 PRINT "    '       '       '          '";CHR$(27);"Y/ '          ";
2920 PRINT "     '                '                '       '       ' ";
2930 PRINT "         '";CHR$(27);"Y0 '                '                  ";
2940 PRINT "  '              '       '       '          '";CHR$(27);"Y";
2950 PRINT "1 '              '                '                '     ";
2960 PRINT "  '       '          '";CHR$(27);"Y2 e
2970 PRINT "a
2980 PRINT CHR$(27);"G";CHR$(27);"q"
2990 RETURN
3000 REM *** EXIT PROGRAM SCREEN ***
3010 GOSUB 2220
3020 X=33:Y=4:PRINT FN Y$(X,Y)RV$" SUMMING IT UP "RX$
3030 X=8:Y=6:PRINT FN Y$(X,Y)"I hope this program will help you dress up your MBasic programs and"
3040 Y=7:PRINT FN Y$(X,Y)"structure them to make them easy to use.  A listing of the program"
3050 Y=8:PRINT FN Y$(X,Y)"will allow you to study the various aspects of the program."
3060 Y=10:PRINT FN Y$(X,Y)"Delete all lines after line 130 and start your program at line 140"
3070 Y=11:PRINT FN Y$(X,Y)"This gives you a start program for all your MBasic programs.  Delete,"
3080 Y=12:PRINT FN Y$(X,Y)"change or add to the variables as needed."
3090 Y=14:PRINT FN Y$(X,Y)"If you don't have the Ed-A-Sketch program, the box around this screen"
3100 Y=15:PRINT FN Y$(X,Y)"GOSUB 2220, is an example of a way to print graphics using the"
3110 Y=16:PRINT FN Y$(X,Y)"variables in this program."
3120 Y=18:PRINT FN Y$(X,Y)"GOOD LUCK WITH YOUR H-89 -- ";
3130 PRINT "GEORGE HILL, JR., INDIANAPOLIS, IN 46240"
3140 X=3:Y=23:PRINT FN Y$(X,Y);RV$" TOUCH ANY KEY TO END PROGRAM "RX$
3150 Z$=INPUT$(1)
3160 PRINT CON$;DIS$:END
```

DUMP YOUR Z100!

Dump your Z100 screen image, that is, onto an MPI Printmate 150g printer! This American made high-quality printer can output any ZDOS graphics image displayed on your Z100 color or monochrome monitor, when used with the MPI Artisan software. Both high-resolution graphics and textual material can be printed out simultaneously. Different colors are represented by various shades of gray on your paper. And, you also may design custom type fonts to use during normal printouts. They can range from mathematical symbols to Hebrew, or script. Several pre-defined fonts are already included with the Artisan software.

For Z100, Z89 and Z90 CP/M users, it's simple to interface for standard text output or user-defined graphic routines. You have the ability to design and print with user-defined type fonts, however screen dumps are not currently supported under CP/M.

The 150g prints at a speed of 150 characters per second in a bidirectional logic-seeking mode using high-speed tabs for maximum throughput. Character densities, baud rates, form lengths and a variety of other programmable features may be selected from the optional keypad.

Standard print formats include an 11x9 correspondence quality density, as well as 10, 12, 15 and 17 cpi (231 columns wide). Double-wide characters, underlining, upper and lower case characters with true descenders, and dot-addressable graphics are all standard.

If a more compact printer is desirable, we also carry the Printmate 99g. It has many of the above features including Z100 screen dumps, and handles both tractor and friction feed with serial and parallel interfaces standard!

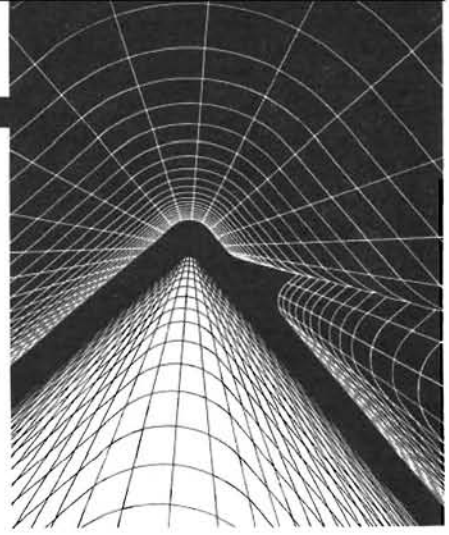
Before purchasing any printer, give us a call and request our fully illustrated catalog on Zenith products and peripherals, and a special PRINTMATE-PAC of color brochures with sample printouts. Printmates start from under $500.

Our prices and service are just plain tough to beat!

Studio Computers inc.

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

Building Block Subroutines



Michael J. Caddy
4700 Watson Drive
Doylestown, PA 18901

Introduction

The impact of low cost micro computer systems has primarily been in the business world. However, many engineering tasks can be performed using micro computer systems. This article describes an engineering/scientific application using an H-89 or Z series system.

Many engineering/scientific tasks require functional representations of one dependent variable in terms of one or more independent variables. Several techniques which can be implemented for this are polynomial curve fits, least squares fits, and linear interpolation. Polynomial curve fits are often difficult to determine and do not fit the required data satisfactorily because of their osculatory nature. Least squares fits are unsatisfactory because the interpolated curve does not go through each set of data coordinate points. Linear interpolation, although very fast if coded efficiently, requires excessive numbers of data coordinate points to closely represent the desired relationship. A non-linear technique such as a cubic spline technique can eliminate many of these problems and can smoothly represent a function using a minimum of data points. Spline interpolation reference (a), simply takes consecutive data point intervals and fits a third order polynomial to the interval such that the resulting curve goes through each data coordinate point and has smooth first and second derivatives between intervals. This curve is analogous to a thin flexible spline constrainted to pass through each data coordinate point such that a minimum strain energy exists. Cubic spline interpolation also has undesirable features. First, changing one data coordinate point at one end of the curve, changes the interpolation values along the entire curve. Second, the goal of any interpolation is to represent the desired function as closely as manually drawn with a French curve. In this context, the cubic spline tends to overshoot turns and oscillate unnaturally around sharp bends.

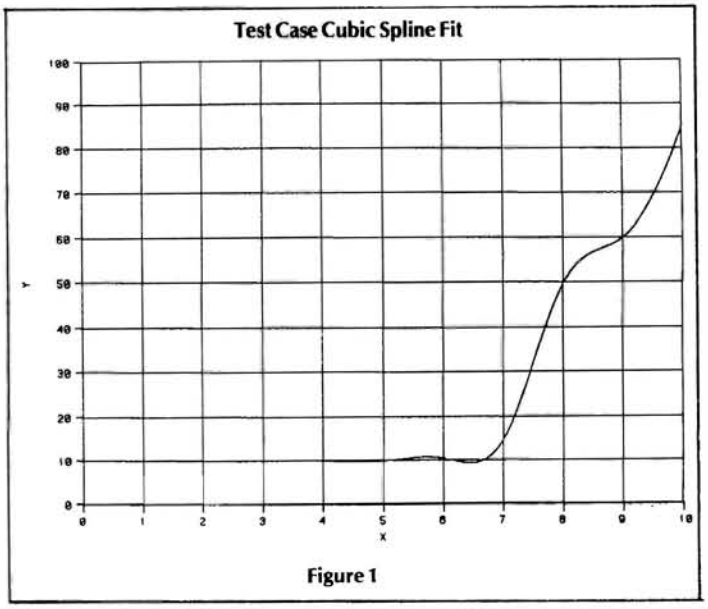


Figure 1

An interpolation which overcomes most of these problems was developed by Hirshima Akima in reference (b). This interpolation technique, designated Akima's technique, overcomes the undesirable features of the cubic spline fit and is computationally simpler. For comparison, the same set of data coordinates is plotted using both Akima's technique and the cubic spline technique shown in figures 1 through 4. The spline curve overshoots the data while the Akima curve smoothly transitions to horizontal without overshoot.

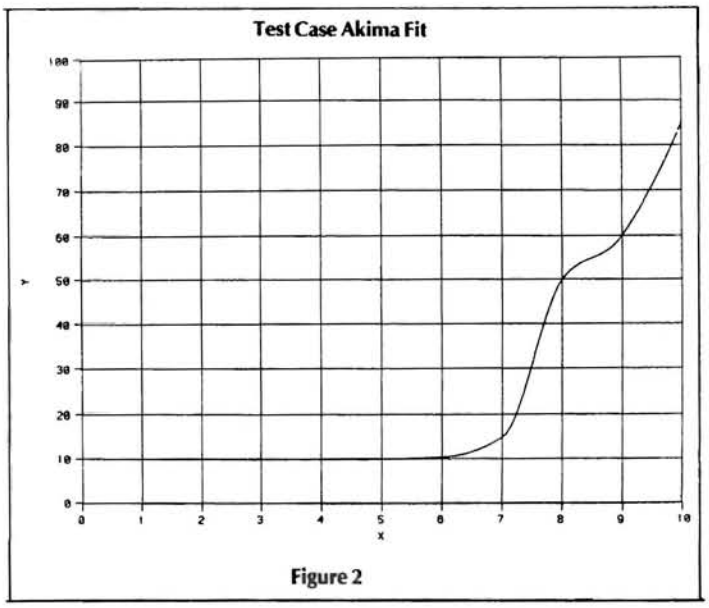


Figure 2

Code Development Using Akima's Technique

In the development of the BASIC code shown in Listing 1 for Akima's interpolation technique, the following features were implemented:

a) Data coordinate storage is in a single dimensional array to permit compact multiple curve storage.

b) Binary search is used to find interpolation interval to minimize search time.

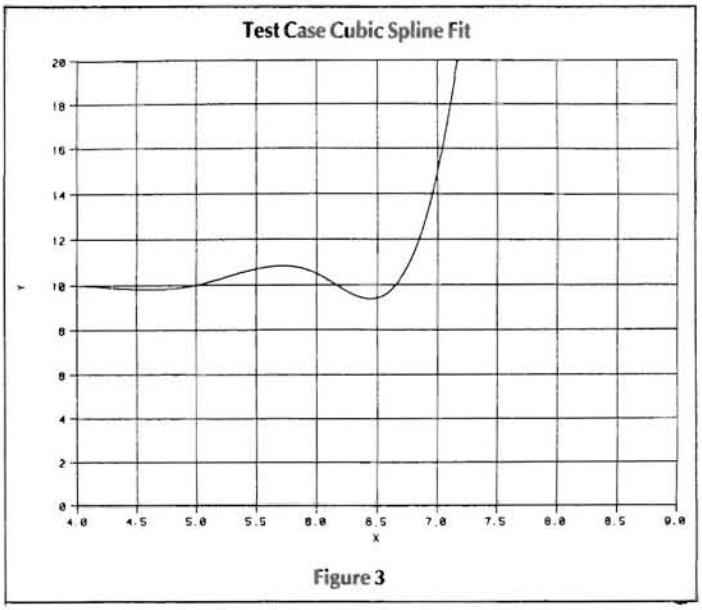


Figure 3

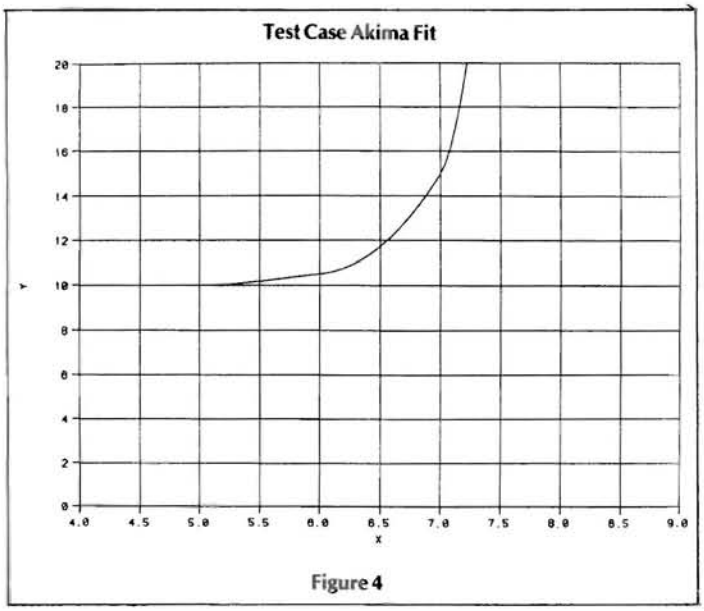


Figure 4

c) Search is initiated at last found interval, to minimize search time.

d) Storage and usage of previously computed polynomial interpolation coefficients to minimize execution time in repeated usage.

In the discussion of the code development, explanation of the Akima's derivation will be left in reference (a). One important feature of the code developed here is the flexibility of the data storage. Each N set of x,y data coordinate points represents a single degree of freedom table interpolation. The storage of these N data points is as follows:

1 No. of data points in this set = N

2 x1

3 x2

4 x3

. .

. .

N+1 xn

N+2 y1

N+3 y2

N+4 y3

. .

. .

. .

2N+1 yn

2N+2 Array location of last table access to this data set. Points to upper interval and initialized to zero.

2N+3 Switch used for fixed data sets. These are data which is never generated as a result of cross interpolation, ie. the y1 through yn data never changes. (1 = fast mode fixed data.)

2N+4 through 2N+6

These are coefficients used in last interpolation. These are only used in the fast mode, when 2N+3 switch is set.

As previously discussed, the user is permitted to store sets of data, as many as possible into a single dimensional array. The user only has to specify the starting location (the array position of N) and GOSUB to the function. A listing of the BASIC code is shown in Listing 1.

A discussion of the inner workings of the code is as follows:

The function is begun by the user storing the independent variable value in X0 shown on line 70 and then a GOSUB 220 to use the function. On line 220 the secondary array positions are set and other data initialized. K0 is the array position starting the data set. Line 230 checks for a single point data set. Line 240 checks for a linear 2 point curve and line 250 sets the coefficients for the linear calculation when only 2 points are input. Lines 260 through 390 comprise the binary search for the interval. The binary search is used to shorten the calculations to find the interval. Further reduction in computation is achieved by storing the array position of the last used interval. The last used array position is stored in the variable L. Line 370 stores the new value for L. At line 400, the lower value of x and y is obtained for the interval. Line 440 checks to see if the fast mode is applicable. The J0 check is used for the single point and linear interpolations. I8 and L are fast mode switch and last array point flags, respectively. Simply, if I8=0 (fast mode inactive) or L=0 (first time through this interpolation), then the fast mode is not permitted. Line 450 transfers the noted polynomial coefficients to the Q1 array and jumps to the final calculations. Lines 460 through 580 contain the calculation procedure as outlined in Akima's reference (b). Lines 590 and 600 check for special conditions of extrapolation and linear interpolations. Line 630 is the linear interpolation. Lines 640 and 650 compute coefficients used in the calculation and line 660 checks for the fast mode to determine if coefficients are to be stored in line 670. Finally, line 680 and 690 compute the Y0 value from the polynomial coefficients. If an extrapolation is required, the coef-

```
10 REM PROGRAM TEST(INPUT,OUTPUT)
20 DIM D(40),Q1(3)
30 DEFINT I-N
40 DATA 11,0,1,2,3,4,5,6,7,8,9,10,10,10,10,10,10,10
45 DATA 10.5,15,50,60,85,0,1
50 FOR I=1 TO 25:READ D(I):NEXT I:K0=1
60 FOR M9=1 TO 50
70 X0=.2*M9
80 GOSUB 220
90 PRINT X0,Y0:NEXT M9
100 END
210 REM LOCAL CUBIC FIT AKIMA..
220 X1=X0:M0=K0:N0=D(M0):I3=M0+N0:N1=M0+1:N2=M0+2
230 IF N0<=1 THEN Y0=D(N2):GOTO 700
240 IF N0>2  THEN 260
250 N=I3+N0:Q1(1)=(D(N)-D(N-1))/(D(I3)-D(I3-1)):M=I3:J0=1:GOTO 610
260 N4=N0*2+N1:L=D(N4):L0=N4+1:I8=D(L0):K=L+M0:NL=N1:J9=I3:J0=-1
270 REM  BINARY SEARCH FOR INTERVAL
280 IF X1=D(I3) THEN 360
290 IF X1>D(I3) THEN J0=0:GOTO 380
300 IF X1<=D(N1) THEN J0=1:K=N2:GOTO 390
310 IF L<=0 THEN 340
320 IF X1>=D(K) THEN NL=K:GOTO 340 ELSE J9=K:K=K-1
330 IF X1<D(K) THEN J9=K ELSE NL=K
340 K=INT((J9-NL)/2)+NL
350 IF K<>NL THEN 330
360 L6=L-J9+M0
370 D(N4)=J9-M0
380 K=J9
390 M=K
400 N=M+N0:Y3=D(N-1):X3=D(M-1)
430 REM CHECK FOR FAST MODE AND  EXTRAPOLATION
440 IF J0>=0 OR I8*L=0 OR L6<>0 THEN 460
450 FOR I=1 TO 3:Q1(I)=D(L0+I):NEXT I:GOTO 680
460 Y4=D(N):X4=D(M):A3=X4-X3:S3=(Y4-Y3)/A3
470 IF M=N2 THEN 500
480 X2=D(M-2):Y2=D(N-2):S2=(Y3-Y2)/(X3-X2)
490 IF M=I3 THEN S4=S3+S3-S2:GOTO 520
500 X5=D(M+1):Y5=D(N+1):S4=(Y5-Y4)/(X5-X4)
510 IF M=N2 THEN S2=S3+S3-S4
520 IF M<=(N2+1) THEN S1=S2+S2-S3 ELSE S1=(Y2-D(N-3))/(X2-D(M-3))
530 IF M>=(I3-1) THEN S5=S4+S4-S3 ELSE S5=(D(N+2)-Y5)/(D(M+2)-X5)
540 W2=ABS(S4-S3):W3=ABS(S2-S1):S0=W2+W3
550 IF S0=0 THEN W2=.5:W3=.5:S0=1
560 Q1(1)=(W2*S2+W3*S3)/S0:W3=ABS(S5-S4):W4=ABS(S3-S2):S0=W3+W4
570 IF S0=0 THEN W3=.5:W4=.5:S0=1
580 T4=(W3*S3+W4*S4)/S0
590 IF J0<0 THEN 640
600 IF J0=0  THEN Q1(1)=T4
610 I1=M-J0
620 REM FAST EXIT FOR 2 POINTS AND LINEAR EXTRAPOLATION
630 Y0=D(I1+N0)+(X1-D(I1))*Q1(1):GOTO 700
640 Q1(2)=(2*(S3-Q1(1))+S3-T4)/A3
650 Q1(3)=(-S3-S3+Q1(1)+T4)/(A3*A3)
660 IF I8*L6=0 THEN 680
670 FOR I=1 TO 3:D(L0+I)=Q1(I):NEXT I
680 D1=X1-X3
690 Y0=Y3+D1*(Q1(1)+D1*(Q1(2)+D1*Q1(3)))
700 RETURN
```

Listing 1

```
.2              10
.4              10
.6              10
.8              10
1               10
1.2             10
1.4             10
1.6             10
1.8             10
2               10
2.2             10
2.4             10
2.6             10
2.8             10
3               10
3.2             10
3.4             10
3.6             10
3.8             10
4               10
4.2             10
4.4             10
4.6             10
4.8             10
5               10
5.2             10.0339
5.4             10.1218
5.6             10.2427
5.8             10.3757
6               10.5
6.2             10.7616
6.4             11.3294
6.6             12.2164
6.8             13.4356
7               15
7.2             19.1708
7.4             26.8226
7.6             35.889
7.8             44.3037
8               50
8.2             52.7549
8.4             54.2868
8.6             55.4412
8.8             57.0637
9               60
9.2             64.04
9.4             68.47
9.6             73.38
9.8             78.86
10              85
```

Listing 2

ficients at the respective end intervals are computed and a linear extrapolation at each end point is computed.

In Listing 2 is a sample output showing interpolation/extrapolation for a 50 point data set. Computational time using an H-89 is 20 seconds.

References

(a) "Introductory Computer Methods and Numerical Analysis", Ralph Pennington.

(b) "A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures", Hiroshi Akima. Journal of the Association for Computing Machinery, Vol. 17, No. 4, Oct 1970.

About the Author:

Michael J. Caddy *is a senior aerospace engineer employed in the Navy civil service. He is a Navy wide technical expert in gas turbine performance, computer simulation, and computer interactive graphics. He is an expert in Fortran, BASIC, and C. In addition, he has a part-time custom software business specializing in professional software for the H89/Z120. At the HUG 82 convention, he was the grand prize winner.*

IF YOUR ROBOT CAN MAKE IT TO THE COUPON, YOU COULD WIN $5,000.

Announcing the Virtual Devices 1st Annual Robot Roll-Off.

The race is on to find the smartest, fastest robot in the U.S.

It's the 1st Annual Robot Roll-Off, sponsored by Virtual Devices, Inc., a manufacturer of the most advanced remote control hardware and software for robot hobbyists.

To win, you have to build a robot that can maneuver a simple course on its own, open doors, detect sound, light, and motion. And do it all against the clock—faster and better than any other robot around.

Robot Roll-Offs are scheduled in cities across the country during the summer of '84. So robot enthusiasts can meet and compete. The Roll-Off grand prize winner walks away with $5,000. And there's cash and merchandise for the runners-up.

Because you'll want every advantage, you'll want to check into Virtual Devices radio remote control products. They let you use your personal computer for greater control and ease in programming.

So if you have or plan to build a HERO, Androbot, RB Robot, other comparable kit, or home-brew of your own design, get ready to roll. You don't have to use Virtual Devices equipment to enter the Roll-Off, but it just might give you the control you need to cross the finish line first.

For more information and Roll-Off rules, mail the attached coupon today to: Virtual Devices, Inc., P.O. Box 30440, Bethesda, MD 20814. Or call toll-free 1-800-762-ROBO.

VIRTUAL DEVICES INC.

HERE I AM. SEND ME THE RULES.

Yes! I'd like to enter the Virtual Devices 1st Annual Robot Roll-Off. Please send me the rules and details. And tell me all about Virtual Devices products for robot hobbyists.

Name _____

Address _____

City _____ State _____ Zip _____

Telephone ( ) _____

Mail to: Virtual Devices, Inc., P.O. Box 30440, Bethesda, MD 20814.

HERO, Androbot and RB Robot are Trademarked names.

Add the Underline Character to ASM

Charles R. Winchester
970 S.W. 191st Court
Aloha, Oregon 97006

I received issue 41 of REMark yesterday and the article 'Using "C" For Fast Action Games' by Clement S. Pepper prompted me to write the following.

First, let me add two books to the reference material Mr. Pepper lists. Both are elementary texts on the C language and are very good material.

Title: The C Primer
Authors: Les Hancock & Morris Krieger
Publisher: McGraw Hill Book Co.
1221 Avenue of the Americas
New York, NY 10020

Title: C Programming Guide
Author: Jack Purdum
Publisher: Que Corporation
7960 Castleway Drive
Indianapolis, IN 46250

When I first started my study of the C language, I soon discovered that all of the texts on C use one character that the ASM assembler will not accept in a Label. That character is the underline which is used as a connector between two words. When typing in text from an example given in a book, it is quit easy to let the underline slip in. The C compiler will accept it, of course. It is only after attempting to assemble the compiled code that the Label is kicked out with a "U"-type error.

The only characters that ASM will accept in a Label are "alpha characters". These are the letters from A to Z, and the characters "$" and "." (period). Numbers are also accepted in a Label so long as the first character is an alpha character. ASM will not accept the character "_" (underline) in a Label.

I liked the way the underline character was being used, so I decided to see if I could make ASM accept it. I tried to find a listing for the HDOS version 2.0 ASM assembler but didn't find it to be available. Heath did offer it for a while, but our local Heath store said it was no longer listed in the catalog.

I do have the listing for HASL-8. HASL-8 is the cassette version of Heath's ASM assembler and I reasoned that it would be similar to HDOS ASM.

Looking through the HASL-8 listing, I found a "look-up" table entitled "LCT LOOKUP - CHARACTER TYPE". This table was to be my starting point.

The table is used to determine if a character is a valid alpha character, a valid number, a valid operator, or a valid postradix symbol. It is quite different from the ASCII table I had expected to find and looked like it would make it simple to do what I wanted to do.

The table consists of all the characters between the space (blank) and the underline character. It consisted of "DB" statements and is formatted as follows:

```
LCTA    EQU     *

        DB      00000000B       BLANK
        DB      00000000B       !
        DB      00000000B       "
        DB      00000000B       #
        DB      10000000B       $
        DB      00000000B       PERCENT
        DB      00000000B       &
        .       .               .
        .       .               .
        DB      01000000B       0
        DB      01000000B       1
        .       .               .
        .       .               .
        .       .               .
        DB      00000000B       @
        DB      11010111B       A
        DB      11010001B       B
        .       .               .
        .       .               .
        DB      10000000B       Y
        DB      10000000B       Z
        DB      00000000B       [
        .       .               .
        .       .               .
        DB      00000000B       _       (underline)
```

If the most significant bit (bit 7) is a "1", then the character is a valid alpha character. If bit 6 is a "1", the character is a valid number; and if bit 5 is a "1", the character is a valid operator. Bit 4 will flag a valid postradix character if it is a "1".

Looking at the table, I noted that the bit pattern for the "$" character and the bit pattern for the "." (period) are both 10000000B. The equivalent octal code is 200Q. The underline character has the pattern 00000000B or octal 000Q. I reasoned that if I could locate the table in HDOS ASM, I could patch the byte representing the underline character and make ASM accept the underline as a valid character. Changing the code from 00000000B to 10000000B (200Q) should do the job. This would make the underline the same as a letter and it could then be used in any position in a label.

I loaded RDT from HUG into my H8 and then loaded ASM. Using RDT's "EXAMINE" utility, I searched through ASM to see if I could locate the table in question. The first time through, I didn't locate a table of DB statements or anything that looked like the table I was looking for. I did notice an area that looked strange at first. It was a series of NOP's followed by a number of MOV B,B statements all on a column. There were 10 MOV B,B statements followed by 7 NOP's.

Looking up the octal value for MOV B,B gave me the clue I needed. 100Q is the opcode for MOV B,B and the opcode for NOP is 000Q.

Looking at the HASL-8 table showed that the numbers were indeed represented by 100Q (MOV B,B) while the seven characters between 9 and A were 000Q (NOP). Stepping back a sufficient amount showed me a series of statements as shown below:

```
053.215  000          NOP                    *
053.216  000          NOP                    *
053.217  000          NOP                    *
053.220  000          NOP                    *
053,221  200          ADD      B             *
053.222  000          NOP                    *
053.223  000          NOP                    *
053.224  000          NOP                    *
053.225  000          NOP                    *
053.226  000          NOP                    *
```

This matched the first ten lines of the HASL-8 Lookup Character type table. Following these first ten lines was what looked like valid OP-CODE however. These lines were as follows:

```
053.227  042 040 000    SHLD     000040A       *"
053.232  041 200 043    LXI      H,043200A     *! @
053.235  100            MOV      B,B           *@
053.236  100            MOV      B,B           *@
```

Stringing these out in a column as follows matched them directly with the HASL-8 table.

```
053.227  042
053.230  040
053.231  000
053.232  041
053.233  200
053.234  043
053.235  100
053.236  100
```

I found the table I was looking for. Counting down through a series of ADD B's, SUB A's etc., I found the byte I wanted. It was located at address 053.314 (2BCCH). I proceeded to change the byte at address 053.314 in ASM from 000Q to 200Q (80H).

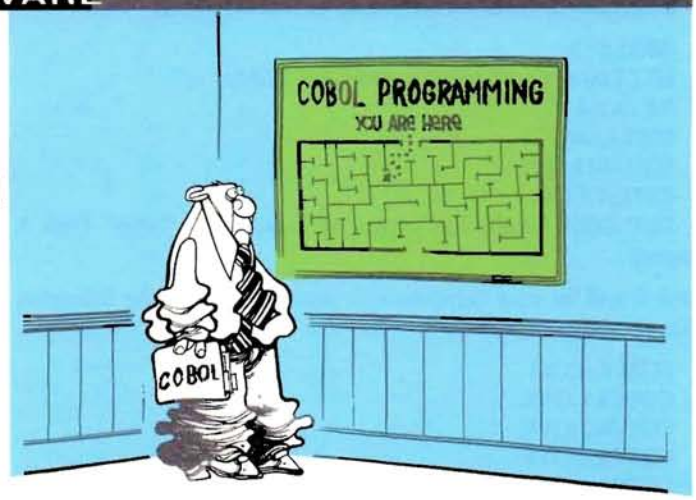
I then wrote a short program in C which uses the function call "prov_it()". This function call will translate into the Label "prov_it" when it is compiled.

I compiled the program using C/80 version 2.0 and assembled it. This was the acid test. If the message "No Errors Detected" came up, I would be successful. After what seemed like hours, the message did come up. My assembler had accepted the underline as a valid character.

Just to make sure that everything was OK, I decided to run it through the assembler again. This time I would ask for a Symbol Table and a Cross Reference Table. Again, everything worked out OK. Both the Symbol Table and the Cross Reference Table had the Label "prov_it" listed. I had successfully made my assembler recognize the underline character in a Label.

```
#include "printf.c"

main()
{
    int i, x;

    for(i = 1, x = 0; i < 10; i++, x++){
        printf("\n The value of x in main: %d", x);
        prov_it();
    }
}
prov_it()
{
    int x;

    printf(" The value of x in prov_it: %d", x);
}
```

COBOL Corner II



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

Introduction

Welcome back to "COBOL Corner". If you missed "COBOL Corner I" (REMark Issue #45), it is not too late to join us. As I stated in the first article, READING will not teach you "Good" programming or COBOL. You MUST have the incentive to work with your COBOL-80 Manual and these articles on your computer as we go along. I assume that by now you have your COBOL system and know your operating system (HDOS or CP/M). I will be using CP/M (I use CP/M because it uses less RAM). If you are using HDOS, the programming format and COBOL coding will be nearly the same. Refer to your COBOL-80 Manual for the HDOS differences. I will not cover these.

COBOL System Set-Up

Referring to the COBOL-80 User's Guide, you will find that the Distribution Disks (files will vary slightly with HDOS) contain the following:

1 - COBOL Compiler

```
A--COBOL.COM
B--COBOL1.OVR
C--COBOL2.OVR
D--COBOL3.OVR
E--COBOL4.OVR
```

2 - RUNTIME SYSTEM

```
A--COBLIB.REL-----RUNTIME LIBRARY
B--CRT DRIVERS----FILE WHOSE NAME BEGIN WITH--CD
      SOURCE--CD___.MAC
      OBJECT--CD___.REL,CRTDRV.REL
```

3 - UTILITY SOFTWARE

```
A--L80.COM--------MICROSOFT LINKING LOADER
B--LIB.COM--------MICROSOFT LIBRARY MANAGER
C--M80.COM--------MICR0SOFT MACRO ASSEMBLER
D--CREF.COM-------MICROSOFT ASSEMBLY
                  CROSS-REFERENCE PROGRAM
```

4 - MISCELLANEOUS FILES

```
A--SQUARO.COB
B--CRTEST.COB
C--SEQCVT.COM
D--COPCOB.SUB
```

COBOL Compiler

The reason COBOL-80 is so practical for the H/Z Microcomputers is because Microsoft furnished us with a compiler consisting of a Main Program and four (4) overlays. This gives us a COBOL Compiler that saves working memory in the computer's RAM. The five (5) parts corresponds to five (5) "steps" of compilation. The main program (COBOL.COM) is always in memory and controls the compilation from one step to the next. The Overlay portion of the main program compiles the IDENTIFICATION and ENVIRONMENT DIVISIONS (we will get into the COBOL component structure shortly). Overlay 1 (COBOL1.OVR) is brought to memory to compile the DATA DIVISION. The PROCEDURE DIVISION is compiled by Overlay 2 (COBOL2.OVR). These three (3) steps complete the first pass of the compilation. This pass creates an INTermediate version of the program, which is stored on the current disk in a temporary file named STEXT.INT. Overlay 3 (COBOL3.OVR) reads STEXT.INT from the disk and creates the Object Code. Overlay 4 (COBOL4.OVR) allocates the file control blocks and checks for certain error conditions. Then STEXT.INT is deleted from disk. Thus, you never see STEXT.INT unless, for some reason, the compiler terminates its functions part way through its steps.

COBOL-80 Runtime System

The RUNTIME LIBRARY contains a group of subroutines that interpret the Object Code of the program produced by the compiler. Not all programs will require all of the library routines. The Linking Loader will search the library and include the routines your program needs and exclude the ones not needed.

The CRT drivers are provided to enable you to configure the COBOL-80 system for your CRT terminal. (See Appendix A of the COBOL-80 Manual.) If you are using an H/Z-19, simply copy CDWH19.REL to CRTDRV.REL. Once you have your CRTDRV.REL configured to your CRT, the driver will be available for each program you load with the Linking Loader.

The L80.COM (Microsoft's Linking Loader) is used to link COBOL Object Code (from compiler) with the Runtime Library. We will not go into the other utility files (M80.COM and CREF.COM) at this time.

COBOL-80 System

The COPCOB.SUB is a miscellaneous file that you can use to make back-up copies of the Original Distribution Disks supplied with COBOL-80. Now would be a good time to make your copies (if you have not done so) and SAVE the Original in a safe place! NEVER work with Original Distribution Disks with "COBOL Corner"! Having made your copies, use these to make yourself three (3) COBOL SYSTEM working disks.

Disk A should contain your operating system (CP/M in my case), SQUARO.COB, CRTEST.COB, and your Editor. This MUST be a bootable disk for use in Drive A (SY0: for HDOS). This disk does not

require many of the CP/M files. I only use the following:

(BIOS.SYS)
ED.COM or your Editor (I use PAGED.COM)
PIP.COM
STAT.COM
SQUARO.COB
CRTEST.COB
DUP.COM (Might find useful for making the "NEW" Disk A copies)

Disk B will be your compiler disk and should have the following files:

COBOL.COM
COBOL1.OVR
COBOL2.OVR
COBOL3.OVR
COBOL4.OVR

This disk will be used in Drive B (SY1: for HDOS) when you are compiling your programs. It does not have to be bootable.

Disk C will be your runtime and linking loader disk and it should have the following files:

L80.COM
COBLIB.REL
CRTDRV.REL

Disk C will be used in Drive B (or Drive C if you are using 3 drives) and it does not have to be bootable.

Note: If your Drive B has 160K or more storage, Disk B and C can be combined with both set of files. I would always keep Disk A as we have specified above. This allows us to use a separate Disk A for each COBOL program. This gives us back-up protection and a nice filing method for storage of your working programs for review from time to time.

Verify Your COBOL System

To verify our Disk A, Disk B, and Disk C, put Disk A in Drive A and Disk B in Drive B. Boot the system and type B: and Return. With CP/M you MUST be logged on the disk that contains the COBOL compiler, since the compiler overlays are always read from the current disk! Now, since Disk A has the test program— SQUARO.COB— type either of the following two (2) command lines:

```
B> COBOL A:SQUARO.REL,TTY:=A:SQUARO.COB
```

OR

```
B> COBOL A:SQUARO,TTY:=A:SQUARO
```

and type Return. Either of these two (2) commands will compile SQUARO.COB and place the relocatable object code file named SQUARO.REL on Disk A, and display the program listing on your CRT. The second command line takes advantage of COBOL-80 default filename extensions assumed by the compiler. If you want a program listing from your printer, type:

```
B> COBOL A:SQUARO,LST:=A:SQUARO and Return
```

Hopefully, when the compilation has been completed, you will find the following message:

```
"NO ERRORS OR WARNINGS"
```

You should find the file SQUARO.REL on Disk A. If you get (1) or more Error Messages or you do not find the SQUARO.REL on Disk A, recheck your work as described in the above procedures. Also, check out the meanings of the Error Messages in your COBOL- 80 Manual. If all your disks have the right files, with the correct extensions, and you have typed the correct command line (watch for spelling errors), you should obtain the correct results (If you cannot, write to me with a SASE, business size, giving complete information and I will offer suggestions.)

Now that the Source Program has been compiled, we must remove Disk B from Drive B and replace it with Disk C. If you have three (3) drives, you do not have to do this step; also, if you have combined these two (2) disks on one 160K disk. The last step before Execution of your test program requires loading SQUARO.REL file using the Linking Loader L80.COM. This step converts the relocatable (.REL) Object Code File into the absolute version. This absolute version is built in RAM memory, from where you may or may not save it on disk, execute it directly, or save & execute it, depending on which Command Line you use.

Now, type B: and Return. Next, type the following command line to load SQUARO.REL and Execute the program without saving the absolute version on disk.

```
B> L80 A:SQUARO/G/E and Return
```

L80.COM assumes the .REL extension for the file SQUARO that is being loaded. The /G is a "switch" that directs the loader to complete the loading process and begin execution after the loader searches the COBLIB.REL and CRTDRV.REL to resolve global symbols. The /E "switch" directs the loader to Exit to the operating system when execution is complete. With this command you cannot Execute the program again without performing the complete command line over, since we did not save the absolute version on disk. So, if we type the following command line:

```
B> L80 A:SQUARO/N,A:SQUARO/E and Return
```

we will now load, link, search, and SAVE the SQUARO.COM File on Disk A. The program will not be executed. The "switch" /N directs the loader to save the executable file on disk. Now, to Execute the program type the following command line:

```
B> A:SQUARO and Return
```

Since the absolute version is saved on disk, we can execute the program at any time with the above command. Also, we can combine the above command lines so that the absolute version is saved on Disk A and the program is Executed directly by typing the following command line:

```
B> L80 A:SQUARO/N,A:SQUARO/G and Return
```

The executed SQUARO program computes the square root of the number you provide and verifies that you have a working COBOL-80 SYSTEM! Lastly, you can use CRTEST.COB File that you have on Disk A in Drive A to test the functions of the interactive CRT driver (see Appendix A in your COBOL-80 Manual). To do this, just substitute A:CRTEST for A:SQUARO in the above compiler and runtime commands.

Closing

Now, you should have your COBOL SYSTEM up and running and understand how to compile and execute a COBOL program. Do this several times so that you can perform these procedures without referring to this article. We will be using these procedures over and over in COBOL Corner. I have covered some of what is going on with the COBOL-80 software but you might want to read your COBOL User's Guide for a better understanding.

In the next COBOL Corner we will Key-In, Compile and Execute a simple COBOL Program where we will review the four (4) DIVISIONS every COBOL program must have along with the necessary

SECTIONS. For your "HOMEWORK" please refer to your COBOL-80 Reference Manual and read the following:

Chapter 1
1.11—Structure of a program
1.12—Coding rules

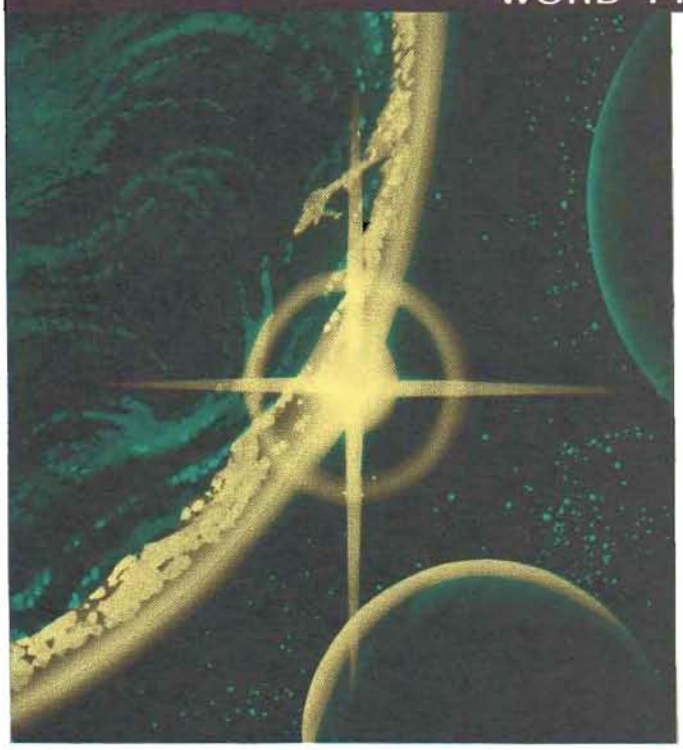
Chapter 2
2.1—IDENTIFICATION DIVISION
2.2—ENVIRONMENT DIVISION
2.2.1—CONFIGURATION SECTION
2.2.2—INPUT-OUTPUT SECTION
2.2.2.1—FILE-CONTROL (SELECT Entry)

Chapter 3
3.15—WORKING-STORAGE SECTION

Chapter 4
4.1—PROCEDURE DIVISION
4.2—Organization of PROCEDURE DIVISION

This reading will help you understand the COBOL PROGRAM STRUCTURE that we will create next time.

Remember, "COBOL Corner" is YOUR article in REMark! If you have constructive criticism, suggestions and/or questions, please send them to me for action. We want to have all readers be part of this series and learn not only my idea of what COBOL is all about, but the broadest possible understanding of COBOL!



Z-100 WordStar: Some Added Flexibilities

Robert A. Metz
Whirlpool Corp.
2000 US 33 N
Benton Harbor, MI 49022

An article in PC WORLD magazine (Volume 1, Number 2) - "WordStar Made to Order" by Emil Flock about the IBM PC version of WordStar piqued my curiosity. It indicated some patches which could be made to WordStar to change defaults, add print commands, and change the function key codes. I looked at the Zenith Z-100 version of WordStar (release 3.21) and, to my surprise, all patch locations indicated in the article for the IBM PC WordStar directly corresponded with the Z-100 WordStar image, including the function key expansion table. Further examination of Z-100 WordStar with DEBUG revealed code for accessing the IBM PC keyboard and display (indicated by telltale INT 10H and INT 16H instructions). It is apparent that Micropro used the IBM PC version of Wordstar as boilerplate in developing the Zenith version and did not even bother to remove the PC dependent code. In the remainder of this article I will describe my findings regarding the implementation of the Z-100 version of WordStar and will also list specific locations within WS.COM which may be changed via DEBUG to customize your copy of WordStar.

The Zenith Install program for WordStar conveniently allows modification of most of the defaults indicated in the above- mentioned article and also some that were not, but the article gives good insight as to exactly how that is done. Based on user selection of options, Install builds an image of WordStar's default parameters, reads WS.COM, overlays the defaults, and writes WS.COM back to disk. Using this knowledge, programs may be easily written to customize WordStar to given applications which require changes to many defaults at a time. Also, Install has a basic flaw in that it itself reverts to a set of defaults on each invocation. It should have read the current values from the WordStar image and displayed them as the current set of defaults allowing simple updating of WordStar defaults rather than having to change everything again from the fixed base. This drawback can be circumvented through special programs written to change only the desired default(s).

The Zenith WordStar manual is of little help if one desires to implement the "User Patches" shown in the ↑P menu. It refers to the "WordStar Customization Notes" from Micropro, a document not many people have access to (myself included). The above-mentioned article explains in a round-about way how to make these "User Patches". These patches allow placement in a WordStar document of control characters which when encountered during the printing of that document, cause the user-defined character strings associated with that control character to be sent directly to the printer. Each "User Patch" (↑PQ, ↑PW, ↑PE, and ↑PR) consists of a 5-byte table entry (address shown in Table 1 below). The first byte is a character count (maximum of 4) and the next four bytes will contain the actual character string to be associated with the "User Patch". If the character count byte is zero (default), the particular "User Patch" is disabled. A typical example of a "User Patch" is to associate ↑PE with an ESC character (01BH). This would allow the user to embed escape sequences in a document to take advantage of some special features of his printer not supported in standard WordStar (e.g. shifting to alternate character sets such as italics). The DEBUG session to perform this example WordStar modification would be as follows (user entries are underlined, x=don't care, s=required space, <cr>=carriage return):

```
A:DEBUGsWS.COM<cr>
>E789<cr>              (address of ^PE patch)
xxxx.0789  00.01s xx.1B<cr>
>W<cr>
Writing 5180 bytes
>Q<cr>

A:
```

The IBM PC function key expansion table, while present in the Zenith version of WordStar, is just so much dead code as it is not used at all to expand the Z-100 function keys. I did, however, locate and decode the key expansion table that is used. It immediately precedes the IBM PC function key table and therefore can easily be expanded. Expansion will wipe out the PC function key table, but who cares

at this point? By adding entries to the active table, codes can be included to process all the Z-100 function keys.

The Z-100 key expansion table consists of 4-byte entries as follows:

byte 1 - Z-100 key code (key expansion disabled).

byte 2 - If non-zero, the key code for the 2nd character of a particular key sequence (may be masked to 6 bits). If zero, the operation involves a single key-stroke only.

bytes 3,4 - Internal WordStar codes defining the operation to be performed.

The table is terminated with a zero as the 1st byte of an entry.

By creating entries in this table corresponding to the Z-100 function keys, these function keys may be associated with any standard WordStar control key sequence within certain limitations. All ↑P commands are processed in such a manner that it is not possible to associate a Z-100 function with them using this method. Function keys may be associated with a 1 or 2 key sequence only. Sequences requiring a third key (e.g. ↑OL, ↑OR) cannot be as fully implemented as they are with the IBM PC. The PC function keys are handled differently in that they are expanded to an up to 6 character string before being processed as if typed in by the user. Code which does this expansion is no doubt buried somewhere in the Z-100 version of WordStar, but finding and accessing it is another matter. Table 2 shows the function keys as implemented in the IBM PC version of WordStar. It would be nice for compatibility's sake to assign the Z-100 keys to equivalent functions but, given the above- mentioned limitations, it can be seen that IBM function keys F3, F4, F5, and F6 cannot have direct counterpart in the Z-100 version. Table 3 lists a set of patches to WordStar to associate the Z-100 function keys with some of the more heavily used double keystroke WordStar functions. The procedure for assigning function keys is as follows:

1) Determine what you want a function key to do. For example, let's say we want key F0 to be equivalent to ↑OC (center text).

2) Find the entry for ↑OC in the Z-100 Main Menu key expansion table (use DEBUG to look at the WS.COM program image). This entry has 0FH (↑O) as the first byte and 03H (C masked to 6 bits) as the second byte. When found, make note of the third and fourth bytes of the entry.

3) Build a new table entry:

byte 1 = 96H (Z-100 key code for F0)
byte 2 = 00H (single key function)
bytes 3,4 = 3402H (from ↑OC entry)

4) Patch this new entry into WS.COM. An example DEBUG session to do this follows (user entries are underlined, x=don't care, s=required space, <cr>=carriage return):

```
A:DEBUGsWS.COM<cr>
>E66D<cr>          (current end of table)
xxxx.066D  00.96s xx.00s xx.34s
xxxx.0670  xx.02s xx.00<cr>
>W<cr>
Writing 5180 bytes
>Q<cr>
A:
```

Be sure there is a new end of table indicator (00 as the first byte of an entry).

With knowledge of the locations within Z-100 WordStar and their meanings as revealed in this article, the user can better customize WordStar for his particular application and operation preferences. These locations are those which I found with a cursory examination of WS.COM with DEBUG. I am sure that there are many features yet to be uncovered which, when known about, can provide the WordStar user with added flexibility in its operation. I would encourage those of you with the inclination to make use of DEBUG (a super tool) to look into otherwise mysterious software such as WordStar to unlock some of those mysteries. I would be particularly interested to hear if anyone can find a way to fully duplicate the IBM PC WordStar function key implementation in the Z-100 version of WordStar.

Table 1

WordStar Internals

Address	Description	WS Access	Default	Install (1)
287	Msg Fgnd Color	-	30	*
288	Msg Bkgnd Color	-	37	*
28E	Txt Fgnd Color	-	37	*
28F	Txt Bkgnd Color	-	30	
292	Z-100 Init Msg(2)	-	05	
29B	Z-100 Exit Msg(2)	-	02	
360	Help Level	^JH	03	*
362	Insert Flag	^V	FF	*
363	Directory Flag	F	FF	*
367	Paper Length(3)	.PL	42	*
368	Paper Length(3)	.PL	10	*
369	Paper Length(3)	.PL	02	*
36B	Top Margin(3)	.MT	03	*
36C	Top Margin(3)	.MT	18	*
36D	Top Margin(3)	.MT	00	*
373	Bottom Margin(3)	.MB	08	*
374	Bottom Margin(3)	.MB	40	*
375	Bottom Margin(3)	.MB	00	*
37C	Standard Pitch	-	0C	*
37D	Alt Pitch	-	0A	
37E	Page Offset	.PO	08	*
37F	Left Margin	^OL	00	*
380	Right Margin	^OR	40	*
385	Word Wrap Flag	^OW	FF	*
386	Justify Flag	^OJ	FF	*
387	Variable Tab Flag	^OV	FF	
388	Soft Hyphen Flag	^OE	00	*
389	Hyphen Help Flag	^OH	FF	*
38A	Print Control Flag	^OD	FF	
38B	Ruler Line Flag	^OT	FF	*
38C	Pg Brk Dsply Flag	^OP	FF	
38D	Pg Brk Suprs Flag	^OP	FF	
38E	Line Spacing	^OS	01	*
38F	Col Move Mode Flg	^KN	00	
392	Non-docn Mode Flg	N	00	*
39A	Hyphen Criterion	-	04	*
3CA	Prt File Output(4)	P	00	
3CB	Prt w/ Frm Feeds(4)	P	00	
3CC	Prt Pg Suprs(4)	P	00	
3CD	Prt Pg Pause(4)	P	00	
3D3	Omit Pg Nums	.OP	00	
3D4	Microjustify	.UJ	FF	
3D5	Prt Bidirectional	.BP	FF	
430	No File Menu Z-100 Key Expansion Table (5)			
471	Main Menu Z-100 Key Expansion Table (5)			
66E	IBM PC Function Key Table (6)			
747	# Bold Overstrikes	-	02	
77F	User Patch 1 (7)	^PQ	00	
784	User Patch 2 (7)	^PW	00	
789	User Patch 3 (7)	^PE	00	
78E	User Patch 4 (7)	^PR	00	

Notes:

1) Quantities changeable through the Zenith WordStar Install pro-

gram are indicated with an *.

2) These messages take the form of a 1-byte character count followed by the character string (up to 8 bytes). Defaults are:

Entry msg: 05,1B,7A,1B,79,3F - reset, disable expansion
Exit msg: 02,1B,7A - reset

The following suggested modification allows for a stable cursor instead of a blinking one:

Entry msg: 08,1B,7A,1B,79,3F,1B,78,3B
Exit msg: 05,1B,7A,1B,78,3B

3) These quantities are in groups of 3, all of which must be updated at once to cause a desired change.

4) These are the default (Y/N) answers to questions asked in response to the 'P' command.

5) Addresses of the 4-byte per entry key expansion tables as indicated above.

6) IBM PC function key expansion table is dead code which may be patched over to expand the active key table.

7) Addresses of the 5-byte "User Patch" commands as indicated above.

General: Flags take on values of FF (on) and 00 (off). To obtain desired values for other defaults, if not immediately obvious as to the encoding of that default (as is the case with the margin and length parameters), run Install to change the desired parameter. Use DEBUG to look at WS.COM and observe the new value for the parameter in question.

Table 2

IBM PC Function Key Equivalences

Function Key	Control Key Equivalent
F1	^JH
F2	^OG
F3	^OL<esc>
F4	^OR<esc>
F5	^PS
F6	^PB
F7	^KB
F8	^KK
F9	^QC
F10	^QR

Table 3

Z-100 Function Key Implementation Patches

Function Key	Patch Address	Patch Value	Equivalence
F0	66D	96003402	^OC
F1	671	97001C02	^OX
F2	675	9800AA82	^G
F3	679	99004602	^QY
F4	67D	9A00FE86	^B
F5	681	9B006085	^KB
F6	685	9C000202	^KD
F7	689	9D001802	^KH
F8	68D	9E006185	^KK
F9	691	9F000602	^KQ
F10	695	A0000C02	^KV
F11	699	A1001002	^KY
(end of table)	69D	00	

About the Author:

Bob Metz is a Sr. Technical Analyst with Whirlpool Corporation in Benton Harbor, Michigan. He graduated from Purdue University in 1969 with a B.S. degree in Aeronautical Engineering. He has had 14 years of computer related experience, primarily with mini and micro-computers, spanning a wide range of applications from engineering analysis to process control and monitoring to the design of languages, operating systems, communications processors, etc. His experience in the so-called "personal computer" area began last October when he gained access to a Z-100 to which he has become quite attached. His main interests lie with system internals about which he has become conversant in the case of the Z-100 as well as the comparable IBM PC.

More Assembly Language Programming - HDOS

P. John Hagan, M.D.
Rd 1, Box 294
Harveys Lake, PA 18618

The number of articles published for the beginning assembly programmer in 'REMark' are limited. In addition, the manual supplied by Heath is often most difficult to decipher. Dr. William Campbell's course, "Getting Started With HDOS and Assembly Language Programming" has been exceptionally helpful, and allows one to begin the study of the language in conjunction with the Heath Assembly Language course. Hopefully, this article will prove useful to those who are continuing their study of the language.

The following program is submitted with extensive documentation in hopes that it will prove useful as a 'next step' in assembly language study. It contains examples of many of the techniques that are necessary to adequately manipulate files, and demonstrates one method of formatting output. It is not presented as a polished and elegant finished product, but if studied, will allow the beginner to progress more rapidly.

First, prepare a diskette containing:

1. ASM.ABS
2. An Editor (i.e. 'Edit', 'Paged')
3. TYPTX.ACM
4. HOSDEF.ACM
5. HOSEQU.ACM
6. ASCII.ACM

Using the editing program, make a sample file exactly as shown in Figure 1, and when finished, save it as "OPERA.DAT". Then, copy Figure 2, and save it as "OPERA.ASM". Assemble this program by typing "ASM OPERA=OPERA" and correct any mistakes until it assembles. Run the assembled program by typing "OPERA".

Before reading further, print copies of the "OPERA.DAT" and "OPERA.ASM" files so they can be examined as you read the rest of this article.

The following comments will point out some of the areas that may be of special interest. This particular program is designed to work with a large file of opera tapes and records. The file data is stored in records as shown in Figure 1. Note that the file ends with a '$' as a last record. It also has delimiters (\) between each data field, as used by Dr. Campbell.

At the start of the program, the 'XTEXT' references are listed, and those files must be on your disk. The first routine simply places an 'X' in an unused address designated by 'SWITCH' (see last few lines of program). We will make use of that space later by placing a 'P' in that address if we want the printer to be used to print data.

The '$TYPTX' call rings a bell, then clears the screen to illustrate the use of escape sequences, and then prints a message. A time delay is used here to hold a screen message.

Whether or not the printer is used, it was decided to load its name, and then open it for writing with the appropriate SCALLs. Notice the error routines, and how easily they are handled with the '$TYPTX'.

Assembly language programming saves memory space and this often allows the entire file to be read and stored in memory as a table. Obviously, the total file size has limits depending on the amount of memory available in your computer. Illustrated is the 'SCALL' for opening a file and reading it. The starting address is set up by equating 'MEMADD' with '*'. This starting address will be assigned efficiently by the system, and will be the reference point for all file searching. The data will be read in groups of 256 characters, and a counter will be set up in the B-C registers to count the characters as they are stored. The routine 'CTR' is of special interest. It uses 16 bits, since 8 bits will count only to a maximum of 255. In addition, note that the 'NL' characters are eliminated in 'ELIMCR' so as to prevent their being stored in memory. As soon as the '$' sign is found, the end of the file has been found and the table is complete.

Next, one decides whether to use a printer to print output. The use of 'CHARACTER MODE' is illustrated. In this mode, the input will be accepted immediately, and not require that a 'Return' be entered as in 'Line Mode'. If the printer is to be used, a 'Y' is typed, and the routine 'OKPRINT' is called. That routine is at the end of the program and perhaps can be studied now. Notice that the data presently in the accumulator must be pushed onto the stack, since we will be using the accumulator to move a 'P' into the address represented by 'SWITCH'. The 'STA' command is useful here. When this is done, the data from the stack is returned to the accumulator and we return.

The routine 'DELAY' is needed for the printing of output since the printer reproduces data slower than the console. By making the value 11Q small enough at 'LOOP15', one can force the printer to fall gradually behind, and the formatted printer output will eventually disintegrate, while the console output remains normal. Try this later as an experiment.

The main menu is now displayed after blanking the screen and skipping appropriate lines. Once again, the 'CHARACTER MODE' is used with input, comparisons using 'CPI', and then various jump routines. In this sequence, an error routine is added in case none of the numbers listed are used.

A few words may be in order here as to what output we want and how we had best get it. The first sequence, 'ENTIRE', is more complicated since it prints the entire file, and formats the output. These sections are well worth careful study, and it is hoped that the documentation is extensive enough that it will be clear. The other sequences involving the choices of tape number, composer, opera, and artist are not formatted in the output, but the record is simply printed as it is, including the delimiters. As an exercise, the reader is invited to modify this output later to suit himself.

In the 'ENTIRE' routine, the '\' delimiters are counted, and assigned numbers from 1 to 7. In this way, the file data can be formatted in any way desired. Variables 26 and 60 can be changed later to suit your particular file needs. The number 26 represents space for that many characters between each delimiter. The number 60 defines

and limits the total length of each record. The routines can be made more efficient, but they are left as they are for purposes of clarity.

All of the other routines which can be chosen from the menu are explained in the documentation. The printing of the individual record is straight-forward since there is no formatting. Because each record may contain the names of four artists, the routines labeled 'ART1' through 'ART4' allow for sequentially scanning each of the 'ARTIST' fields, and printing the record if the artist's name is found.

Finally, the last portion of the program contains multiple-use and 'CALL' routines. All are labeled and hopefully they will be clear. Note in the closing routine, 'CLOSE', that both Channel 2 used for the printer and Channel 1 used for the 'OPERA.DAT' file are closed separately. Further error routines can also be added as necessary. The use of '$TYPTX' simplifies this. Choose the locations of 'INBUFF' and 'SWITCH' according to your memory space.

Summary

Using HDOS, an assembly language program has been presented which contains examples of 'SCALL' usage as well as sequential file manipulation of records, the opening and closing of files, and the formatting of console and printer output. While all of these techniques are documented in the HDOS manuals, they are presented here in simplified fashion in order to clarify their usage for the beginning assembly language programmer.

Figure 1

```
001A\DONIZETTI G.\LA FAVORITA\PAVAROTTI L.\MILNES S.\VERRETT S.\GIAIOTTI B.\
001B\MOZART W.\DON GIOVANNI\SUTHERLAND J.\MORRIS J.\TOURANGEAU H.\VARADY J.\
002A\GIORDANO U.\ANDREA CHENIER\CORELLI F.\TEBALDI R.\COLZANI A.\CASEI N.\
002B\VERDI G.\SIMON BOCCANEGRA\TIBBETT L.\RETHBERG E.\PINZA E.\MARTINELLI G.\
003A\VERDI G.\I VESPRI SICILIANI\GEDDA N.\MILNES S.\CABALLE M.\DIAZ J.\
004A\BIZET G.\THE PEARL FISHERS\GEDDA N.\MICHEAU J.\BLANC E.\MARS J.\
005A\GIORDANO U.\ANDREA CHENIER\DOMINGO P.\VELIS A.\MACNEIL C.\ARROYO M.\
006A\BELLINI V.\NORMA\DOMINGO P.\CABALLE M.\COSSOTTO F.\RAIMONDI R.\
007A\BELLINI V.\NORMA\CALLAS M.\STIGNANI E.\SUTHERLAND J.\PICCHI M.\
007B\GOUNOD C.\ROMEO AND JULIET\BJOERLING J.\SAYAO B.\MOSCONA N.\BROWNLEE J.\
$
```

Figure 2

```
*       TITLE OPERA.ASM
*
*       P. JOHN HAGAN, M.D.
*       RD1 BOX 294
*       HARVEYS LAKE, PA. 18618
*
        XTEXT   HOSDEF
        XTEXT   HOSEQU
        XTEXT   ASCII
        XTEXT   TYPTX
        ORG     USERFWA
*
*
*
*
**************ROUTINE TO SET PRINTER SWITCH ADDRESS TO 0***********
START   MVI     A,'X'           INITIALIZE ACCUM WITH 'X'
        STA     SWITCH          PLACE DATA IN SWITCH ADDRESS
*                               THIS WILL BE USED TO CONTROL PRINTER
*                               LATER IN PROGRAM (SEE OUTPUT RTNS)
*******************************************************************
        CALL    $TYPTX          SHOWING MESSAGE HANDLING, BELL,
        DB      BELL,ESC,45H        & CLEARING SCREEN
        DB      NL,NL
        DB      NL,'               HAGAN''S OPERA PROGRAM',NL
        DB      NL,NL,NL
        DB      NL,'                         THIS IS A TEACHING PROGRAM',NL
        DB      '                            WRITTEN FOR HDOS ASSEMBLY',NL
        DB      '                            LANGUAGE.  IT CAN BE ',NL
        DB      '                            IMPROVED UPON, BUT IS LEFT',NL
        DB      '                            LESS THAN ELEGANT PURPOSELY',NL
        DB      '                            AS A TEACHING EXERCISE',NL
        DB      '                            TO INCREASE ITS CLARITY.',NL
        DB      NL,NL,NL
        DB      '                            P. JOHN HAGAN, M.D.',NL
        DB      NL,NL
        DB      'WAIT',NL,ENL
*
        MVI     B,03Q                   THIS IS A TIME DELAY ROUTINE
*                                           TO HOLD A MESSAGE ON SCREEN.
HOLD    MVI     C,377Q                      SEE DOCUMENTATION AT 'DELAY'
HOLD1   CALL    DELAY                       SUBROUTINE.
        DCR     C
        JNZ     HOLD1
        DCR     B
        JNZ     HOLD
*
        CALL    $TYPTX          CLEAR SCREEN AGAIN
        DB      ESC,45H
        DB      NL,ENL
*
*****************ROUTINE FOR LOADING PRINTER********************
*
LOADPR  LXI     H,DEVICE        LOAD NAME FOR PRINTER
        SCALL   .LOADD          SYSTEM CALL
        JC      ERROR           IF UNSUCCESSFUL LOAD
        CALL    $TYPTX
```

```
        DB      'LP: PRINTER IS LOADED.',NL,ENL
*
*****************ROUTINE FOR OPENING PRINTER TO WRITE***************
*
OPENPR  MVI     A,2             OPEN PRINTER FOR WRITE
        LXI     D,DEFALT2
        LXI     H,DEVICE
        SCALL   .OPENW
        JC      ERROR2
        CALL    $TYPTX
        DB      'IF PRINTER IS ON, THEN IT IS OPEN FOR PRINTING',NL,ENL
*
*****************MESSAGE WHILE WAITING FOR FILE READING*************
*
FILELD  CALL    $TYPTX
        DB      'WAIT!  ENTIRE FILE IS BEING LOADED INTO MEMORY',NL,ENL
*
*******MAKE A TABLE--REQUIRES ENOUGH MEMORY FOR SIZE OF FILE*********
*
OPENR   MVI     A,1             RTN FOR OPENING FILE TO READ
        LXI     D,DEFALT            SEE MANUAL FOR OUTLINE
        LXI     H,FNAME
        SCALL   .OPENR
        JNC     TEMP            PROCEED IF ALL OK
        CALL    $TYPTX          IF 'READ' UNSUCCESSFUL, THEN END PROGRAM
        DB      'UNABLE TO OPEN FILE.',NL
        DB      'PLEASE CHECK ITS FORMAT AND LOCATION.',ENL
        XRA     A
        SCALL   .EXIT
*
TEMP    LXI     D,MEMADD        SET UP ADDRESS FOR TABLE
        PUSH    D               SAVE IT IN THE STACK
READ    MVI     A,1             OPEN CHANNEL 1
        LXI     B,256           READ ONE SECTOR
        LXI     D,BUFFER        SET D-E TO BUFFER ADDR TO READ RECORD
        SCALL   .READ           READ IT
        JC      BADRD           ERROR ROUTINE
        LXI     B,256           THIS WILL COUNT CHARACTERS IN BUFFER
*
READ1   LXI     H,BUFFER        SET HL TO START OF BUFFER DATA
LOOP20  MOV     A,M             MOVE A CHARACTER TO ACCUMULATOR
        CPI     NL              RTN TO ELIMINATE NL FROM MEMORY
        JZ      ELIMCR                JUMP BEFORE STORING
        POP     D               BRING BACK STARTING ADDRESS OF TABLE
        STAX    D               MOVE CHARACTER TO MEMORY
        INX     D               INCREMENT ADDRESS OF TABLE
        PUSH    D               SAVE THAT NEW ADDRESS
ELIMCR  INX     H               INCREMENT BUFFER ADDRESS TO NEXT BYTE
        CPI     '$'             IS IT END OF FILE YET?
        JZ      DECIDE          IF SO, TABLE DONE, GO TO MENU
        CALL    CTR             DECREMENT CHARACTER COUNTER FOR BUFFER
        JZ      READ            IF EMPTY, GET 256 MORE CHARACTERS
        JMP     LOOP20          IF NOT, READ ANOTHER CHAR INTO TABLE
****************DECISION TO USE PRINTER OUTPUT*********************
DECIDE  CALL    $TYPTX
        DB      NL,NL,NL,NL,NL,NL
        DB      'DO YOU WANT TO USE PRINTER FOR OUTPUT?',NL
        DB      NL
        DB      'TYPE Y OR N',ENL
        CALL    CHRMODE         USE CHARACTER MODE ROUTINE
        CALL    INPUT           ACCEPT THE INPUT
        CPI     'Y'             IS IT 'Y'?
        CZ      OKPRINT         IF SO, THEN CALL PRINT ROUTINE
*************************************************************
*
*****************NOW TO DISPLAY MENU**************************
*
MENU    CALL    $TYPTX
        DB      ESC,45H
        DB      NL,NL,NL,NL,NL,NL,NL,NL
        DB      'TYPE ONE OF THE FOLLOWING INSTRUCTION NUMBERS TO:',NL,NL
        DB      '       1--PRINT ENTIRE OPERA LIST--',NL
        DB      '       2--PRINT LISTING BY TAPE NUMBER--',NL
        DB      '       3--PRINT LISTINGS OF SPECIFIED COMPOSER--',NL
        DB      '       4--PRINT LISTINGS OF SPECIFIED OPERA--',NL
        DB      '       5--PRINT LISTINGS OF SPECIFIED ARTIST--',NL
        DB      '       6--EXIT FROM PROGRAM--',NL,NL,NL,ENL
*
        CALL    CHRMODE         TO ILLUSTRATE USE OF CHARACTER MODE
*
        CALL    INPUT           GET A CHARACTER
        CPI     '1'             IS IT FOR ENTIRE LIST
        JZ      ENTIRE          IF SO PRINT ENTIRE LIST
        CPI     '2'             IS IT FOR OPERA NUMBER?
        JZ      NUMBER          IF SO DO NUMBER
        CPI     '3'             IS IT FOR COMPOSER?
        JZ      COMPOSE         IF SO DO COMPOSE RTN.
        CPI     '4'             IS IT FOR OPERA TITLE?
        JZ      OPERA           IF SO DO OPERA RTN
        CPI     '5'             IS IT FOR A PARTICULAR ARTIST?
        JZ      ARTIST          IF SO DO ARTIST RTN
        CPI     '6'             IS IT FOR EXIT
        JZ      CLOSE           IF SO EXIT FROM PROGRAM
        JMP     MENERR          IF NONE OF ABOVE GO BACK AGAIN
*
*****************ROUTINE TO PRINT ENTIRE LIST*********************
*************THIS SECTION SHOWS EXAMPLE OF OUTPUT EDITING***********
*
ENTIRE  MVI     A,00            INITIALIZE 'A' REGISTER AS '\' CTR
        PUSH    PSW             SAVE '\' COUNT
READ2   LXI     H,MEMADD        SET MEMORY TO START OF TABLE
LOOP    MOV     A,M             MOVE CHAR TO ACCUMUL.
        INX     H               INCREMENT MEMORY
        CPI     '\'             IS CHAR THE DELIMITER?
        JZ      SPACES          IF SO, JUMP TO SPACES RTN
        CPI     '$'             IS IT END OF FILE?
        JZ      FINIS           IF SO, GO BACK TO MENU.
        CALL    OUTPUT          IF NOT, PRINT THE CHAR
        DCR     D               DECREMENT CHARACTER COUNTER D
        JMP     LOOP            GET ANOTHER CHAR IF NOT DONE
*
************THIS SPACES RTN SHOWS ONE WAY TO EDIT OUTPUT***********
*
SPACES  POP     PSW             BRING BACK '\' COUNTER
```

```
        INR     A               INCREMENT IT BY ONE
        PUSH    PSW             SAVE IT AGAIN
        CPI     01              IS '\' COUNTER 1?
        CZ      DATA1           IF SO GO TO DATA1 RTN
        CPI     02              ETC. ETC.
        CZ      DATA2
        CPI     03
        CZ      DATA2   THE NUMBER 26 REPRESENTS TOTAL OF LETTERS
        CPI     04              AND SPACES ALLOTTED TO EACH CATEGORY
        CZ      DATA4           THIS CAN BE CHANGED AS LONG AS WITHIN
        CPI     05              LIMITS OF CTR SCREEN SPACE
        CZ      DATA4
        CPI     06
        CZ      DATA4
        CPI     07
        JZ      DATA7
        JMP     LOOP
*
DATA1   MVI     D,04            FOUR SPACES AFTER NUMBER IN OUTPUT
        MVI     A,' '           LOAD A SPACE INTO ACCUMU
LOOP1   CALL    OUTPUT          PRINT THE SPACE
        DCR     D               DECREMENT SPACE COUNTER
        JNZ     LOOP1           LOOP IF NOT FOUR SPACES
        MVI     D,26            RELOAD CHAR COUNTER FOR NEXT
        RET
*
DATA2   MVI     A,' '           BEGIN RTN FOR SPACES AFTER COMPOSER
LOOP2   CALL    OUTPUT                  NAME IN OUTPUT.
        DCR     D               DECREMENT COUNTER ACCORDING TO HOW
        JNZ     LOOP2                   MANY CHARACTERS AND SPACES
        MVI     D,26                    ARE PRESENT, THEN SET UP
        RET                             NEXT COUNTER
*
DATA4   CALL    NL1             RTN FOR SECOND 'ARTIST' LISTING
        MVI     D,60            LOAD COUNTER FOR # SPACES FROM LEFT
LOOP4   MVI     A,' '                   MARGIN.
        CALL    OUTPUT
        DCR     D
        JNZ     LOOP4
        RET
*
DATA7   CALL    NL1             SINCE '7', IT MEANS THAT 4TH ARTIST
        POP     PSW                     HAS NOW BEEN PRINTED AND CAN
        MVI     A,00                    RESET '\' COUNTER, SAVE IT,
        PUSH    PSW                     AND BEGIN NEW RECORD.
        CALL    NL1
        JMP     LOOP
*
**********ROUTINE TO FIND PARTICULAR NUMBER TAPE**********
*
NUMBER  CALL    LNMODE          GO BACK TO LINE MODE FOR INPUT
*
        CALL    $TYPTX
        DB      NL,NL,NL,'WHAT NUMBER TAPE DO YOU WANT?',NL
        DB      NL,NL,NL,ENL
        LXI     D,INBUFF
        MVI     B,03            SEE COMMENT BELOW ABOUT NUMBER '3'
LOOP7   INX     H               MOVE MEMORY ONE ADDRESS
        DCR     B               DECREMENT COUNTER
        JNZ     LOOP7           IF NOT ZERO, GO BACK
CMP2    LXI     D,INBUFF        SET DE AT FIRST CHARACTER
        MVI     B,05            CTR FOR 5 CHARACTERS OF COMPOSERS NAME
        LDAX    D               MOVE FIRST CHAR INTO ACCUMULATOR
        CMP     M               COMPARE WITH RECORD FIRST CHAR
        JNZ     NOTSAME         IF NO MATCH GO TO NOTSAME
**********FALL INTO SAME ROUTINE**********
SAME    DCR     B               IF MATCH WILL INCREMENT
        JZ      REDPRIN         IF ALL 5 MATCH THEN READY TO PRINT
        INX     H               INCREMENT TO NEXT CHAR IF NOT 5 MATCH
        INX     D               READY FOR NEXT INPUT CHAR
        LDAX    D               LOAD ACCUMULATOR WITH NEXT CHAR
        CMP     M               COMPARE AGAIN WITH CHAR IN MEMORY
        JNZ     NOTSAME         JMP AGAIN IF NOT SAME
        JMP     SAME            IF SAME, GO TRY ANOTHER
**********ROUTINE FOR NOT SAME**********
NOTSAME MVI     C,07            SINCE NO MATCH, MOVE TO NEXT RECORD
LOOP8   MOV     A,M             CTR SET FOR '\''S, MOVE CHAR TO ACCUMU
        INX     H               INCREMENT MEMORY ADDRESS
        CPI     '$'             ARE WE AT END OF FILE?
        JZ      FINIS           IF SO, GO BACK TO MENU
        CPI     '\'             IS IT A DELIMITER
        JZ      COTDEL          IF SO, MAKE A COUNT OF IT
        JMP     LOOP8           IF NOT A DELIMITER MOVE TO NEXT CHAR
**********COUNTER FOR DELIMITERS**********
COTDEL  DCR     C               DECREMENT THE DELIMITER COUNTER
        JNZ     LOOP8           IF NOT YET ZERO, THEN NOT FAR ENOUGH
        MVI     B,05            SHOULD BE AT BEGINNING OF RECORD NOW
        JMP     CMP2            GO BACK AND COMPARE THIS RECORD WITH INPUT
**********ROUTINE TO PRINT A RECORD**********
REDPRIN MVI     B,04            GOING TO PRINT, SO WILL GO TO BEGINNING
GOBACK  DCX     H                    OF RECORD.  DECREMENT MEMORY
        MOV     A,M             MOVE CHAR TO ACCUMULATOR
        CPI     '\'             IS IT A '\'
        JNZ     GOBACK          IF NOT GET ANOTHER CHARACTER
GOBACK2 DCX     H               IF SO, THEN WILL GO BACK 4 MORE CHAR
        DCR     B               DECREMENT COUNTER
        JNZ     GOBACK2         NOT YET BACK TO BEGINNING, SO DO AGAIN
        MVI     C,07            NOW AT BEGINNING OF RECORD
        CALL    PRINT
        JMP     COMPMEM         BACK TO SCAN NEXT RECORD FOR NAME
*
**********ROUTINE TO PRINT OPERA TITLE**********
*
OPERA   CALL    LNMODE
        CALL    $TYPTX
        DB      NL,NL,'WHAT IS THE OPERA TITLE? (AT LEAST 5 LETTERS)',NL
        DB      '(IF ONLY 4 LETTERS, ADD A ''\''',NL
        DB      NL,ENL
*
        LXI     D,INBUFF        SET UP SPACE FOR INPUT IN MEMORY
        MVI     B,05            WILL USE ONLY 5 LETTERS FOR NOW
        CALL    INRTN
*
```

```
        CALL    INRTN
*
COMPARE LXI     H,MEMADD        SET MEMORY AT START FOR FILE DATA
LOOPCMP LXI     D,INBUFF        SET DE AT ADDRESS FOR STORED INPUT
        MVI     B,03            HERE WE USE THREE SINCE FIRST THREE
*                                   CHARACTERS ARE IMPORTANT NUMBERS
*                                   YOU MAY CHANGE THIS ACCORDING TO
*                                   YOUR OWN FILE
        LDAX    D               MOVE CONTENT OF DE INTO ACCUMULATOR
        CMP     M               COMPARE IT WITH FIRST CHAR IN MEMORY
        JNZ     NOMATCH         IF NOT THE SAME GO TO NOMATCH
**********IF THEY MATCH, FALL INTO THAT ROUTINE**********
MATCH   DCR     B               IF THE SAME DECREMENT B
        JZ      RESET           IF ALL FOUR CHAR THE SAME, PRINT
*                                   BUT FIRST BACK UP MEMORY AT
*                                   'RESET' TO FIRST CHARACTER OF NUMBER
        INX     H               NEXT ADDRESS OF MEMORY
        INX     D               NEXT ADDRESS OF INPUT
        LDAX    D               MOVE CONTENT OF DE TO ACCUMULATOR
        CMP     M               COMPARE CHAR WITH THAT IN MEMORY
        JNZ     NOMATCH         IF NOT SAME GO TO NOMATCH
        JMP     MATCH           IF SAME GO BACK AND COMPARE ANOTHER
**********ROUTINE FOR NOT MATCHING**********
NOMATCH MVI     C,07            SET COUNTER FOR '\'
LOOPSET MOV     A,M             MOVE CHAR TO ACCUMULATOR
        INX     H               INCREMENT MEMORY ADDRESS
        CPI     '$'             IS IT END OF FILE
        JZ      FINIS           IF SO GO BACK TO MENU
        CPI     '\'             IS IT A DELIMITER?
        JZ      CTDELIM         JMP TO DELIMITER COUNT RTN
SETREC  JMP     LOOPSET         CONTINUE TO INCREMENT
**********THIS ROUTINE WILL COUNT THE DELIMITERS**********
CTDELIM DCR     C               DECREMENT '\' CTR
        JNZ     SETREC          IF NOT 7TH '\', CONTINUE TO INCREMENT H
        JMP     LOOPCMP         NOW AT START NEXT RECORD, CONTINUE
********ROUTINE TO USE WHEN WANT RECORD TO BE PRINTED**********
RESET   MVI     B,02            RTN TO BACK UP RECORD, READY TO PRINT
BACKSET DCX     H               DECREMENT H
        DCR     B               DECREMENT CTR
        JNZ     BACKSET         DO IT AGAIN
        MVI     C,07            NOW AT BEGINNING OF THAT RECORD
        CALL    PRINT               SO CAN NOW PRINT IT
        JMP     LOOPCMP         DO IT AGAIN
*
**********ROUTINE TO FIND COMPOSERS**********
*
COMPOSE CALL    LNMODE          STAY IN LINE MODE
        CALL    $TYPTX
        DB      NL,NL,'WHAT IS COMPOSER''S NAME? (5 LETTERS)',NL
        DB      NL,NL,ENL
        LXI     D,INBUFF        SET UP SPACE FOR INPUT IN MEMORY
        MVI     B,05            WILL USE ONLY 5 LETTERS FOR NOW
        CALL    INRTN
*
COMP2   LXI     H,MEMADD        SET MEMORY TO FIRST RECORD
COMPMEM MVI     B,05            CTR FOR MOVE TO COMPOSER FIELD
COMP3   LXI     H,MEMADD        SET MEMORY TO FIRST RECORD
COMMEM3 MVI     B,02            CTR FOR'\' FOR OPERA TITLE FIELD
LOOP61  MOV     A,M             EXAMINE FIRST CHARACTER
        INX     H               INCREMENT MEMORY ADDRESS
        CPI     '\'             IS IT '\'
        JNZ     LOOP61          IF NOT GO BACK
        DCR     B
        JNZ     LOOP61
*       INX     H
CMP3    LXI     D,INBUFF        SET D-E AT FIRST CHARACTER
        MVI     B,05            CTR FOR 5 CHARACTERS OF OPERA TITLE
        LDAX    D               MOVE FIRST CHAR INTO ACCUMULATOR
        CMP     M               COMPARE WITH RECORD FIRST CHAR
        JNZ     NOTSAM3         IF NO MATCH GO TO NOTSAM3
SAME3   DCR     B               IF IT IS A MATCH WITH CHECK NEXT CHAR
        JZ      ALLOK3
        INX     H               INCREMENT TO NEXT CHAR IF NOT 5 MATCH
        INX     D               READY FOR NEXT INPUT CHAR
        LDAX    D               LOAD ACCUMULATOR WITH NEXT CHAR
        CMP     M               COMPARE AGAIN WITH CHAR IN MEMORY
        JNZ     NOTSAM3         JMP AGAIN IF NOT SAME
        JMP     SAME3           IF SAME, TRY ANOTHER. SEE IF ALL 5 OK
*
NOTSAM3 MVI     C,07            SINCE NO MATCH, MOVE TO NEXT RECORD
LOOP62  MOV     A,M             CTR SET FOR '\', MOVE CHAR TO ACCUMU
        INX     H               INCREMENT MEMORY ADDRESS
        CPI     '$'             ARE WE AT END OF FILE?
        JZ      FINIS           IF SO, GO BACK TO MENU
        CPI     '\'             IF IT A DELIMITER?
        JZ      COTDEL3         IF SO, MAKE A COUNT OF IT
        JMP     LOOP62          IF NOT A DELIMITER, MOVE TO NEXT CHAR
COTDEL3 DCR     C               DECREMENT THE DELIMITER COUNTER
        JNZ     LOOP62          IF NOT YET ZERO, THEN NOT FAR ENOUGH
        MVI     B,05            SHOULD BE AT NEXT RECORD ADDRESS NOW
        JMP     CMP3            GO BACK, COMPARE THIS RECORD WITH INPUT
*
ALLOK3  MVI     B,02            GOING TO PRINT, SO WILL GO TO BEGINNING
GOBAC3  DCX     H                   OF RECORD AND DECREMENT '\' CTR
        MOV     A,M             MOVE CHAR TO ACCUMULATOR
        CPI     '\'             IS IT A '\'?
        JNZ     GOBAC3          IF NOT GET ANOTHER CHAR
GOBAC4  DCR     B               IF SO, THEN WILL GO BACK MORE
        JNZ     GOBAC3
        MVI     B,04
DOWNM   DCX     H               DECREMENT MEMORY ADDRESS
        DCR     B               DECREMENT '4' CTR
        JNZ     DOWNM
        MVI     C,07            NOW AT BEGINNING OF RECORD
        CALL    PRINT
        JMP     COMMEM3         BACK TO SCAN NEXT RECORD FOR TITLE
*
**********ROUTINE TO SEARCH ALL FOUR ARTIST FIELDS**********
*
ARTIST  CALL    LNMODE          STAY IN LINE MODE
        CALL    $TYPTX
        DB      NL,NL,'WHAT IS THE NAME OF THE ARTIST? (5 LETTERS)',NL
        DB      NL,NL,ENL
```

```
*
        LXI     D,INBUFF
        CALL    INRTN
*
ARTBEG  LXI     H,MEMADD        SET MEMORY TO BEGINNING ADDRESS
ART1    MVI     B,03            SET CTR FOR '\' FOR FIRST ARTIST
        CALL    SETMEM          FIND PROPER DELIMITER FOR FIRST ARTIST
*
CMP4    LXI     D,INBUFF        SET D-E AT FIRST INPUT CHARACTER
        MVI     B,05            CTR FOR 5 CHARACTERS OF ARTIST NAME
        LDAX    D               MOVE FIRST CHAR INTO ACCUMULATOR
        CMP     M               COMPARE WITH RECORD FIRST CHAR
        JNZ     NOTSAM4         IF NO MATCH GO TO NOTSAM4
SAME4   DCR     B               IF IT IS A MATCH, CHECK WITH NEXT CHAR
        JZ      ALLOK4          IF ALL FIVE MATCH, THEN WILL PRINT
        INX     H               INCREMENT TO NEXT CHAR IF NOT 5 MATCH
        INX     D               READY FOR NEXT INPUT CHAR
        LDAX    D               LOAD ACCUMULATOR WITH NEXT CHAR
        CMP     M               COMPARE AGAIN WITH CHAR IN MEMORY
        JNZ     NOTSAM4         JMP AGAIN IF NOT SAME
        JMP     SAME4           IF SAME, GO TRY ANOTHER TO SEE IF 5 OK
*
NOTSAM4 MVI     C,07            SINCE NO MATCH, MOVE TO NEXT RECORD
LOOP72  MOV     A,M             CTR SET FOR '\', MOVE CHAR TO ACCUMU
        INX     H               INCREMENT MEMORY ADDRESS
        CPI     '$'             ARE WE AT END OF FILE
        JZ      ART2            GO ON FOR NEXT ARTIST POSITION
        CPI     '\'             IS IT A DELIMITER?
        JZ      COTDEL4         IF SO, MAKE A COUNT OF IT
        JMP     LOOP72          IF NOT A DELIMITER, MOVE TO NEXT CHAR
COTDEL4 DCR     C               DECREMENT THE DELIMITER COUNTER
        JNZ     LOOP72          IF NOT YET ZERO, THEN NOT FAR ENOUGH
        JMP     CMP4            GO BACK AND COMPARE  WITH INPUT
**************ROUTINE WHEN ALL FIVE CHARACTERS MATCH INPUT*********
ALLOK4  MVI     B,03            GOING TO PRINT, SO WILL GO TO BEGINNING
GOBAC5  DCX     H                 OF RECORD AND DECREMENT '\' CTR
        MOV     A,M             MOVE CHAR TO ACCUMULATOR
        CPI     '\'             IS IT A DELIMITER?
        JNZ     GOBAC5          IF NOT GET ANOTHER CHAR
        DCR     B               IF SO THEN WILL GO BACK MORE
        JNZ     GOBAC5          NOT YET BACK FAR ENOUGH
        MVI     B,04            WILL HAVE TO GO BACK PAST NUMBER
DOWNM4  DCX     H               DECREMENT MEMORY ADDRESS
        DCR     B               DECREMENT '4' CTR
        JNZ     DOWNM4          NOT YET ZERO?, THEN REPEAT
        MVI     C,07            NOW AT BEGINNING OF RECORD WANTER
        CALL    PRINT           PRINT IT
        JMP     ART1            BACK TO SCAN NEXT RECORD FOR ARTIST 1
*********NOW LOOKING FOR INPUT ARTIST IN SECOND POSITION**********
ART2    LXI     H,MEMADD        SET MEMORY ADDRESS TO START OF FILE
DECTR2  MVI     B,04            CTR FOR DELIMITERS FOR SECOND ARTIST
        CALL    SETMEM               ADDRESS
CMP5    LXI     D,INBUFF
        MVI     B,05                ETC.
        LDAX    D
        CMP     M               ALL OF THE ROUTINES FOR ARTIST CAN
        JNZ     NOTSAM5            BE MADE MORE ELEGANT BY USING CALL
        CPI     '\'
        JZ      COTDEL6
        JMP     LOOP76
COTDEL6 DCR     C
        JNZ     LOOP76
        JMP     CMP6
*
ALLOK6  MVI     B,05
GOBAC8  DCX     H
        MOV     A,M
        CPI     '\'
        JNZ     GOBAC8
        DCR     B
        JNZ     GOBAC8
        MVI     B,04
DOWNM6  DCX     H
        DCR     B
        JNZ     DOWNM6
        MVI     C,07

        CALL    PRINT
        JMP     DECTR3
**************LOOKING FOR INPUT ARTIST IN FOURTH POSITION*************
ART4    LXI     H,MEMADD
DECTR4  MVI     B,06            SIXTH DELIMITER NOW IMPORTANT
        CALL    SETMEM
CMP7    LXI     D,INBUFF
        MVI     B,05
        LDAX    D
        CMP     M
        JNZ     NOTSAM7
SAME7   DCR     B
        JZ      ALLOK7
        INX     H
        INX     D
        LDAX    D
        CMP     M
        JNZ     NOTSAM7
        JMP     SAME7
NOTSAM7 MVI     C,07
LOOP78  MOV     A,M
        INX     H
        CPI     '$'
        JZ      FINIS
        CPI     '\'
        JZ      COTDEL7
        JMP     LOOP78
COTDEL7 DCR     C
        JNZ     LOOP78
        JMP     CMP7
*
ALLOK7  MVI     B,06
GOBAC9  DCX     H
        MOV     A,M
        CPI     '\'
        JNZ     GOBAC9
        DCR     B
```

```
SAME5   DCR     B               ROUTINES, BUT HAVE BEEN LEFT AS IS
        JZ      ALLOK5          FOR TEACHING PURPOSES.
        INX     H
        INX     D
        LDAX    D
        CMP     M
        JNZ     NOTSAM5
        JMP     SAME5
NOTSAM5 MVI     C,07
LOOP74  MOV     A,M
        INX     H
        CPI     '$'
        JZ      ART3
        CPI     '\'
        JZ      COTDEL5
        JMP     LOOP74
COTDEL5 DCR     C
        JNZ     LOOP74
        JMP     CMP5
*
ALLOK5  MVI     B,04
GOBAC6  DCX     H
        MOV     A,M
        CPI     '\'
        JNZ     GOBAC6
        DCR     B
        JNZ     GOBAC6
        MVI     B,04
DOWNM5  DCX     H
        DCR     B
        JNZ     DOWNM5
*
        MVI     C,07
        CALL    PRINT
        JMP     DECTR2
**********LOOKING FOR INPUT ARTIST IN THIRD POSITION**********
ART3    LXI     H,MEMADD
DECTR3  MVI     B,05            FIFTH DELIMITER NOW IMPORTANT
        CALL    SETMEM
CMP6    LXI     D,INBUFF
        MVI     B,05
        LDAX    D
        CMP     M
        JNZ     NOTSAM6
SAME6   DCR     B
        JZ      ALLOK6
        INX     H
        INX     D
        LDAX    D
        CMP     M
        JNZ     NOTSAM6
        JMP     SAME6
NOTSAM6 MVI     C,07
LOOP76  MOV     A,M
        INX     H
        CPI     '$'
        JZ      ART4
        JNZ     GOBAC9
        MVI     B,04
DOWNM7  DCX     H
        DCR     B
        JNZ     DOWNM7
*
        MVI     C,07
        CALL    PRINT
        JMP     DECTR4
*
**********STORAGE AREA FOR MULTIPLE USE ROUTINES***********
*
FINIS   CALL    NL1             SKIP A LINE
        CALL    $TYPTX
        DB      'HIT ''CR'' TO GO BACK TO MENU.',NL,ENL
LOOP50  CALL    INPUT           WAIT FOR 'CR'
        CPI     NL              IS IT A CARRIAGE RETURN?
        JZ      MENU            IF SO GO TO MENU
        JMP     LOOP50          IF NOT WAIT AGAIN
**********IMPORTANT ROUTINE TO SHOW COUNTER IN 16 BIT REGISTERS*******
CTR     MOV     A,C             THE MAIN BUFFER CTR FOR 256 CHARACTERS
        SUI     01              SUBTRACT 1 FROM C REGISTER
        MOV     C,A             MOVE IT BACK
        MOV     A,B             MOVE B INTO ACCUMULATOR
        SBI     01              SUBTRACT 1 IF A CARRY
        MOV     B,A             MOVE IT BACK
        RET
**********THE DELIMITER COUNTER ROUTINE*****************************
SETMEM  MOV     A,M             MOVE FIRST CHARACTER TO ACCUMU
        INX     H               INCREMENT MEMORY ADDRESS
        CPI     '\'             IS CHAR A DELIMITER
        JNZ     SETMEM          DO IT AGAIN IF IT IS NOT
        DCR     B               IF IT IS A DELIMITER, DECREMENT CTR
        JNZ     SETMEM          IF NOT THE PROPER DELIMITER, DO AGAIN
        RET
**************ROUTINE TO ACCEPT INPUT**************************
INRTN   CALL    INPUT
        STAX    D               STORE IT IN D-E
        INX     D               INCREMENT INPUT MEMORY
        CPI     NL
        JNZ     INRTN           GO BACK FOR ANOTHER CHARACTER
        RET
***************PRINTING ROUTINE*************************************
PRINT   MOV     A,M             PRINT RTN, KEEPING TRACK OF '\'
        CALL    OUTPUT          OUTPUT FIRST CHARACTER
        INX     H               INCREMENT ADDRESS OF MEMORY
        CPI     '\'             IS IT A '\'
        JNZ     PRINT           IF NOT DO NEXT CHARACTER
        DCR     C               IF IT IS '\', THEN COUNT IT
        JNZ     PRINT           DO NEXT PART OF RECORD
        CALL    NL1             NEXT LINE IF RECORD ALL PRINTED
        RET
**************ROUTINE ILLUSTRATING USE OF CHARACTER MODE***********
CHRMODE XRA     A               GO TO CHAR MODE, NOT LINE MODE
        MVI     B,2010
        MVI     C,2010
```

```
        SCALL   .CONSL
        RET
************ROUTINE ILLUSTRATING USE OF LINE MODE*****************
LNMODE  XRA     A               GO BACK TO LINE MODE
        MVI     B,000Q
        MVI     C,201Q
        SCALL   .CONSL
        RET
*****************CLOSING ROUTINE*********************************
CLOSE   CALL    $TYPTX
        DB      'PROGRAM COMPLETED-BACK TO HDOS.',NL
        DB      'TO RETURN TO MAIN MENU, TYPE "MBASIC BEGIN"',NL
        DB      'TO CLOSE DOWN COMPUTER, TYPE "BYE"',ENL
END2    MVI     A,2             CLOSE CHANNEL 2
        SCALL   .CLOSE
END1    MVI     A,1             CLOSE CHANNEL 1
        SCALL   .CLOSE
        XRA     A               EXIT TO HDOS
        SCALL   .EXIT
*******************INPUT SCALL*************************
INPUT   SCALL   .SCIN            INPUT A CHARACTER
        JC      INPUT           LOOP WHILE WAITING
        RET
*********************TO SET PRINTER FOR USE*************************
OKPRINT PUSH    PSW             SAVE WHAT IS IN ACCUMULATOR ON STACK
        MVI     A,'P'           MOVE 'P' INTO ACCUM.
        STA     SWITCH          STORE 'P' INTO SWITCH ADDRESS
        POP     PSW             BRING BACK DATA
        RET                     RETURN
******OUTPUT ROUTINES FOR PRINTER AND TERMINAL*****************
OUTPUT  PUSH    PSW             SAVE DATA THAT IS IN ACCUM
        LDA     SWITCH          MOVE DATA IN SWITCH ADDRESS TO ACCUM
        CPI     'P'             IS THAT DATA A 'P' FOR PRINTING?
        JZ      OUTPUT2         IF IT IS THEN USE THE PRINTER ROUTINE
*                                  IF NOT, FALL INTO TERMINAL ROUTINE
***********ROUTINE FOR TERMINAL PRINTING**********************
OUTPUT1 POP     PSW             BRING BACK THE DATA FROM THE STACK
        SCALL   .SCOUT          OUTPUT A CHARACTER TO TERMINAL
        RET
************ROUTINE FOR PRINTER AND TERMINAL BOTH************
OUTPUT2 POP     PSW             BRING BACK DATA FROM THE STACK
        SCALL   .SCOUT          OUTPUT A CHARACTER TO TERMINAL
        OUT     340Q            OUT TO PRINTER AT PORT 340Q
        CALL    DELAY           DELAY USED BECAUSE PRINTER SLOW
*                                  SEE COMMENT IN TEXT OF ARTICLE
        RET
***********ROUTINE FOR TIME DELAY--SEE BOB ELLERTON,REMARK,MARCH 1981***
DELAY   PUSH    D               SAVE OUR COUNTER
        MVI     D,377Q          PUT 256D INTO FIRST LOOP
LOOP15  MVI     E,11Q           PUT 11D INTO SECOND LOOP
LOOP9   DCR     E               NOW GO THROUGH LOOPS
        JNZ     LOOP9
        DCR     D
        JNZ     LOOP15
        POP     D               GET BACK '\' COUNTER
        RET
```

```
***************ROUTINE TO SKIP A LINE**********************
NL1     MVI     A,NL            NEW LINE RTN
        CALL    OUTPUT
        RET
*
**********SOME ERROR ROUTINES FOLLOW AS EXAMPLES***************
*
BADRD   CALL    $TYPTX          ERROR RTN--CAN'T READ FILE
        DB      'UNABLE TO READ FILE.',NL
        DB      'PLEASE CHECK FILE FORMAT AND LOCATION.',ENL
        JMP     END1
*
ERROR   CALL    $TYPTX          ERROR RTN--CAN'T LOAD LP:
        DB      'ERROR IN LOADING LP:',ENL
        JMP     END2
*
ERROR2  CALL    $TYPTX          ERROR RTN--CAN'T OPEN LP:
        DB      'ERROR IN OPENING LP: FOR WRITE',ENL
        JMP     END2
*
MENERR  CALL    $TYPTX          IF IMPROPER NUMBER CHOSEN IN MENU
        DB      'IMPROPER INPUT, PLEASE TRY AGAIN.',NL,ENL
        JMP     MENU            GO BACK TO START OF MENU
*
*
BUFFER  DS      256             256 CHARACTER INPUT LINE BUFFER
MEMADD  EQU     *               BEGINNING OF MEMORY
INBUFF  EQU     200000A         CHOOSE MEM LOCATION ABOVE SIZE OF FILE
*                                  (FOR TEMPORARY INPUT DATA)
DEFALT  DB      'SY0TMP'        A DUMMY NAME-REQUIRED
DEFALT2 DB      'LP:',0,0,0
DEVICE  DB      'LP:',0
FNAME   DB      'OPERA.DAT',0   LOCATION & NAME OF DATA FILE
SWITCH  EQU     177376A         MEMORY LOCATION FOR STORAGE OF EITHER
*                                  'X' OR 'P' FOR PRINTER SWITCH
        END     START
```

Vectored from 9

ZBASIC Subroutines

Dear HUG,

Enclosed is a listing of two subroutines which I have written to perform two functions not easily handled in Z-BASIC. The first routine (CMPSTR) compares two string variables and returns a result dependent on the alphabetic order of the two strings. The second routine (SRHSTR) searches one string variable to determine if a second one is embedded within it. I use these routines in a mailing list program. The equivalent functions in Z-BASIC are more complicated and much slower.

These subroutines are invoked via the CALL statement as described in Appendix E of the Z-BASIC manual. Generating the *.BIN file and loading it into memory are also done as described in Appendix E. However, when calling Z-BASIC, I use a value for maximum memory of D000 hexadecimal (/M:&HD000) rather than the 32768 bytes shown on page E.8. The former value provides more memory for the BASIC program.

Both of these subroutines use the 'repeat' prefixes of the 8088 microprocessor. The repeat prefix causes the microprocessor to execute the instruction following the prefix over and over so long as certain conditions are met (CX not 0 and zero flag either set or reset according to the particular prefix). Since the instruction is already in memory, the microprocessor can execute this instruction without an instruction—fetch cycle and so execute it with significantly increased speed.

The first subroutine (CMPSTR) uses the 'compare string' command (CMPS) which compares two bytes, sets flags accordingly, and then increments registers to point to the next bytes in the two strings. Since this command assumes that several registers are set with the correct pointers and values, most of the code in the beginning is devoted to presetting registers. Thus CX holds the length of the string to be compared (the shorter of the two in this case); SI and DI contain pointers to the two strings and BX points to the location to store the result. With these setups completed, the command REPE CMPS scans the two strings until a mismatch is found (zero flag not set) or the end of the strings is reached (CX=0). The arguments BYTE PTR [BX] AND [BX] have no function except to tell the assembler to compare bytes instead of words.

SRHSTR is similar in its setup to CMPSTR except that it first scans the searched string to find a match with the first character of the target string. When one is found, it then

```
sub     segment         ;subroutine to compare strings
        assume  cs:sub
cmpstr  proc    far     ;arguments are string1, string2, result
        jmp     cmpa    ;compare string
        jmp     srhstr  ;search for string
cmpa:   mov     bp,sp   ;result=-1 if string1<string2
                        ;result=1 if string1>string2
        cld             ;result=0 if string1=string2
        mov     dh,0
        mov     ch,0
        mov     ax,-1   ;assume length of 1 less than 2
        mov     bx,8[bp]        ;point to string1
        mov     cl,[bx] ;length of string1 in cl
        mov     si,1[bx]        ;character pointer in si
        mov     bx,6[bp]        ;point to string2
        mov     dl,[bx] ;string2 length in dl
        cmp     cl,dl   ;use shorter string length
        jl      cmp1    ;all set up if cl<dl
        mov     ax,0    ;set up equal case
        jz      cmp1
        mov     cl,dl   ;otherwise use string2 length
        mov     ax,1
cmp1:   mov     di,1[bx]
        mov     bx,4[bp]        ;point to result
        repe    cmps    byte ptr [bx],[bx]      ;compare strings
        jz      zcase
        jg      gcase
        mov     word ptr [bx],-1        ;less than case
        jmp     cmpz
gcase:  mov     word ptr [bx],1 ;greater than case
        jmp     cmpz
zcase:  mov     [bx],ax
cmpz:   ret     6
srhstr: mov     ah,0
        mov     ch,0
        mov     dh,0
        cld
        mov     bp,sp
        mov     bx,8[bp]        ;point to string1
        mov     al,[bx]
        mov     di,1[bx]
        mov     bx,6[bp]        ;point to string2
        mov     dl,[bx]
        mov     si,1[bx]
        mov     bx,4[bp]        ;point to result
        dec     dx      ;adjust count to exclude first character
        sub     ax,dx   ;search length=len(1)-len(2)+1
        mov     cx,ax   ;put in cx
        mov     al,[si] ;first character in al for scanning
        inc     si      ;point to second character
srh1:   cmp     cx,0    ;don't scan if cx<=0
        jle     srh2
        repnz   scas    byte ptr [bx]   ;scan for match of first byte
        cmp     al,-1[di]       ;if last character matched
        jnz     srh2    ;check the rest
        push    di      ;save if no match
        push    cx
        mov     cx,dx   ;use count of string to be matched
        repe    cmps    byte ptr [bx],[bx]
        pop     cx
        pop     di      ;get ready in case no match
        jnz     srh1    ;try again if no match
        mov     word ptr [bx],1  ;1=no match
        ret     6
srh2:   mov     word ptr [bx],0  ;0=no match
        ret     6
cmpstr  endp
sub     ends
        end
```

compares the two strings to determine if the rest of the target string is matched. To do this, it saves the setup needed for the scan command and sets up the registers for the compare command. After the compare command, it immediately pops the scan parameters off the stack to ensure that the pop is done as well as to set up for a continuation of the scan if that should be necessary. Before each scan, SRHSTR also makes sure that the length of the string is not negative. A negative string length is interpreted as one greater than 32678! This will cause the microprocessor to scan beyond maximum memory and cause a bus error.

Another feature of the two subroutines is the use of vectors for calling the subroutines from BASIC. A jump instruction is included for each of the subroutines at the beginning of the program. With these jump instructions, CMPSTR can be called with a displacement of 0 and SRHSTR always has a displacement of 3 in the CALL command regardless of the actual length of the subroutines. Thus changes can be made to the subroutines, which might change their length, without the need to adjust the BASIC program to take the new lengths into account, which would be the case if the displacement used pointed to the actual start of the subroutine.

Victor Skowronski
511 Union Street, Apt. #5
Schenectady, NY 12305

A Problem With Heath's LPMX80 Device Driver

Dear HUG,

I recently found a problem with Heath's LPMX80 device driver supplied with the HDOS 2.0 update disk. Apparently the driver counts not only the printable characters sent to the printer, but also the non-printable codes that control the printer's attributes. In short, if a user selects the maximum line width of 132 characters and then attempts to send a line of 132 printable characters plus embedded printer control codes, the driver will automatically issue a carriage return and line feed after passing the 132nd printable or non-printable character. This makes quite a messy printed page. A simple solution I found is to patch the device drivers maximum line width setting (adjusted by using SET.ABS) from 132 to 255. For the MX-80 or FX-80 users', this will give ample room for control codes within the print line. This patch is located at address 1276 and can be changed to 255 by using PATCH.ABS. The patch codes can be found in REMark #29 or with the documentation provided with the MX-80 printer.

Also, if there are any Heath users' in the Rapid City area interested in forming a users' group, please contact me at the address below.

Joe McIntosh
9857A Piedmont St.
Ellsworth AFB, SD

Some Tips For Benton Harbor BASIC

Dear HUG,

It was not an unusual situation: I was writing a Benton Harbor BASIC program, and I wanted to print some rather complex column headings to the printer and to disk as well as to the screen. The normal, tedious, and memory intensive way to do this is as follows:

```
PRINT "Whatever it is you want to print."
PRINT #1,"Whatever . . . (etc.)"
PRINT #2,"Whatever . . . (etc.)"
```

The thought of copying out all that material three times had me in a mood.

But then, inspiration struck, and the discovery of yet another handy but undocumented feature of Benton Harbor BASIC. I tried this:

```
FOR N=0 TO 2
PRINT #N, "Whatever . . . (etc.)"
NEXT N
```

Voila! Insert a "CLOSE #N" before the "NEXT N" statement and you can close all open files just as easily!

Another (totally unrelated) point, in reference to Joseph Bobbit's letter concerning the DIRECT.SYS file (Buggin' HUG, REMark #42, 7/83). Mr. Bobbit is correct in his description of DIRECT.SYS, but there is one important point that should be mentioned. DIRECT.SYS takes up 18 sectors on the disk, or 9 two-sector clusters. Each cluster holds information for 22 file names, which at 23 bytes each comes out to 506 bytes. That leaves 6 free bytes, which HDOS apparently chooses to "pad out" so that the next sector can start with the beginning of a file name, rather than "spill over" into the next sector. The routine in Mr. Bobbit's letter should therefore be modified to keep track of how many bytes have been read; otherwise, on a full disk, things will get out of sync. Add the following lines:

```
13055 X=X+1:IF X=22 THEN 13070
13070 FOR X=1 TO 6:A=CIN(1):NEXT:X=0:GOTO 13010
```

Randall Stokes
401 S. Silver
Centralia, WA 98531

In Reference to CONC.BAS File Concentrator

Dear HUG,

This letter refers to the article by Roger Evans in Issue #43, titled "CONC.BAS File Concentrator". Since I have a single drive system myself, I thought that this program would be worth the effort to translate to Benton Harbor BASIC. I would like to pass along the changes and additions which were necessary to allow the program to run under Benton Harbor BASIC. Since BASIC checks the syntax of a program line when a program is loaded, I found it necessary to add a few lines to take care of the more common errors which would be encountered. From time to time you may run across a syntax error which was generated by the concentrator. These errors are easily corrected manually.

Change the following lines:

```
00170 OPEN A$ FOR READ AS FILE #1
00190 LINE INPUT #1,;X$
00440 NEXT K: PRINT: PRINT#2,
00450 I=0: E=CIN(1): IF E<>0 THEN CLOSE #2:
                                        GOTO 660
00540 IF V$+Z$ = "REM" THEN F=0: K=L: C=C+1:
                                        GOTO 450
```

LINES 480,490,500,510,515
Replace the word "and" with a "+", and combine strings on the right side of the "=" sign. The remainder of the line remains the same.

Add the following lines:

```
00045 REM  E=END of file test variable
00175 OPEN B$ FOR WRITE AS FILE #2
00195 X$=CHR$(E)+X$
00315 IF V$+Z$="T " THEN GOTO 330
00325 IF Z$="T" THEN V$=Z$: I=I+1: J=3: GOTO 310
00552 IF V$+Z$="THEN" THEN PRINT" ";
                              PRINT #2," ";: C=C-1
00555 IF V$="T" THEN Z$=V$: I=I-1: GOT 420
```

In regard to your article in Issue #44, "Turning on the Super 89". If the new CPU board were installed, do you know of any market for the original CPU board? The new board from D-G has many of the features that are lacking in my H89A. If there were a market for the old board, I may be able to afford to purchase the new one. Thank you for a very informative article. I have been anxious to know more about this board ever since I first saw it advertised in REMark.

Jim Vaccaro
725 Lake Ave. NE
Massillon, OH 44646

Cooling The H-89

Dear HUG,

I have been reading with interest the various suggestions for improving the cooling in H-89 computers and decided to put all the suggested methods to the test. I instrumented my H-89 with internal drives, using thermocouples located as described below. I then connected the thermocouples to a Leeds & Northrup 8692 precision temperature bridge and conducted the tests. I initially let the unmodified computer warm up for 1 1/2 hours. After each modification, I al-

lowed the computer to re-stabilize for 1/2 hour. Temperatures were recorded immediately after a disk DUPlication was performed in order to bring the disk drive shield and power supply to a "worst case" temperature condition.

Seven thermocouples were used, located as follows:

#1 — Ambient air temperature (air in)
#2 — Power supply heat sink, near U101
#3 — Terminal Logic Board heat sink, near U401
#4 — CPU Board heat sink, near U567
#5 — Video Circuit Board, near Q211
#6 — Disk Drive shield, on the top surface
#7 — Air outlet temperature

The six test configurations made use of three suggestions which were published in REMark recently. These configurations are summarized below:

A — Unmodified H-89 (since my computer has modification "C", I unmodified it using duct tape).

B — Unmodified H-89, but with the top cover ventilation slots (except those above the fan) blocked, similar to the KRES Engineering clear cover demonstrated at a LAHUG meeting, described by Larry Fina in REMark 41.

C — Ventilation slots opened up as described in REMark 37 article by Bob Small.

D — Modification "C", except ventilation slots blocked as in "B".

E — Turn cooling fan upside down to blow air into the H-89 as suggested recently (in REMark 42?).

F — Modification "E", except ventilation slots blocked as in "B".

The results of the tests are shown below. It is interesting to note that in no case did blocking the top cover vent holes (configurations "B", "D", and "F") improve the cooling of the H-89, as in the KRES Engineering demonstration unit.

Modifying the vent slots above the fan ("C") slightly improves the cooling overall.

Reversing the fan ("E") greatly improves the Power Supply and Video Board cooling, but at the expense of higher temperatures at the Terminal Logic board and CPU board. In my H-89 I noticed the airflow coming out through the internal disk drives, and felt it could be detrimental to the diskettes since dust may be deposited on the disk surface. For this reason, I decided to leave my computer modified in "C" configuration with the fan exhaust flowing out through the top.

Chuck Hansen
Winding Brook
3 Beverly Ct.
Tinton Falls, NJ 07724

DATA - Temperatures shown in deg F

Configuration	air in	DC supply	Terminal board	CPU board	Video board	Disk drive	air out
A	75	118	118	97	122	82	91
B	75	118	124	108	122	86	90
C	75	116	118	93	120	82	88
D	75	118	122	108	122	86	86
E	75	99	126	108	111	86	91
F	75	99	129	115	116	90	91

Changing your address? Be sure and let us know since the software catalog and REMark are mailed bulk rate and it is not forwarded or returned.

CUT ALONG THIS LINE

HUG MEMBERSHIP RENEWAL FORM

When was the last time you renewed?

Check your ID card for your expiration date.

IS THE INFORMATION ON THE REVERSE SIDE CORRECT? IF NOT, FILL IN BELOW.

Name _____

Address _____

City-State _____

Zip _____

REMEMBER - ENCLOSE CHECK OR MONEY ORDER

CHECK THE APPROPRIATE BOX AND RETURN TO HUG

	NEW MEMBERSHIP RATES	RENEWAL RATES
US DOMESTIC	$18 ☐	$15 ☐
CANADA	$20 ☐	$17 ☐ US FUNDS
INTERNAT'L*	$28 ☐	$22 ☐ 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 4, Issue 11

POSTMASTER: If undeliverable, please do not return.

885-2046