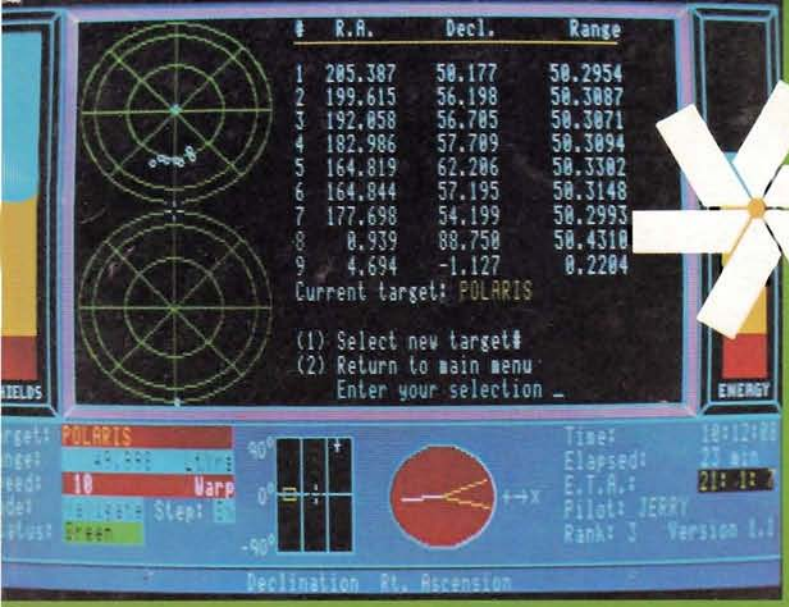
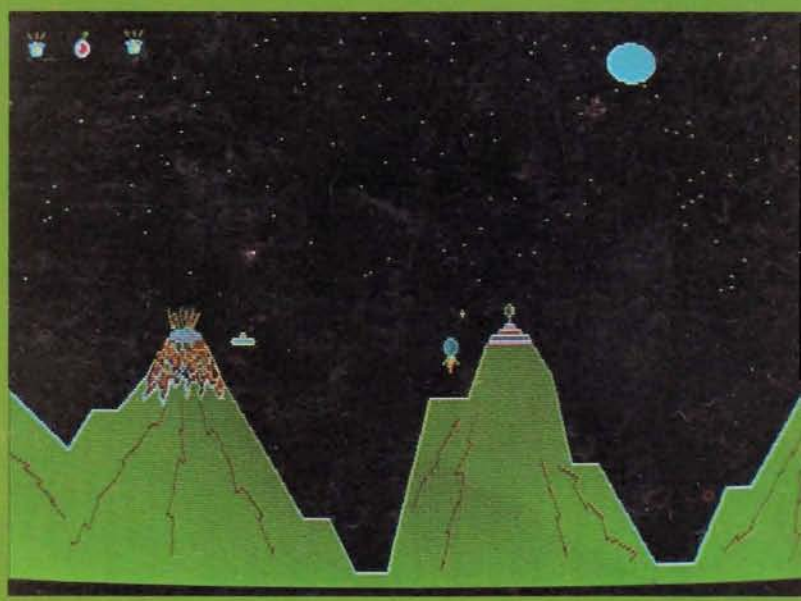


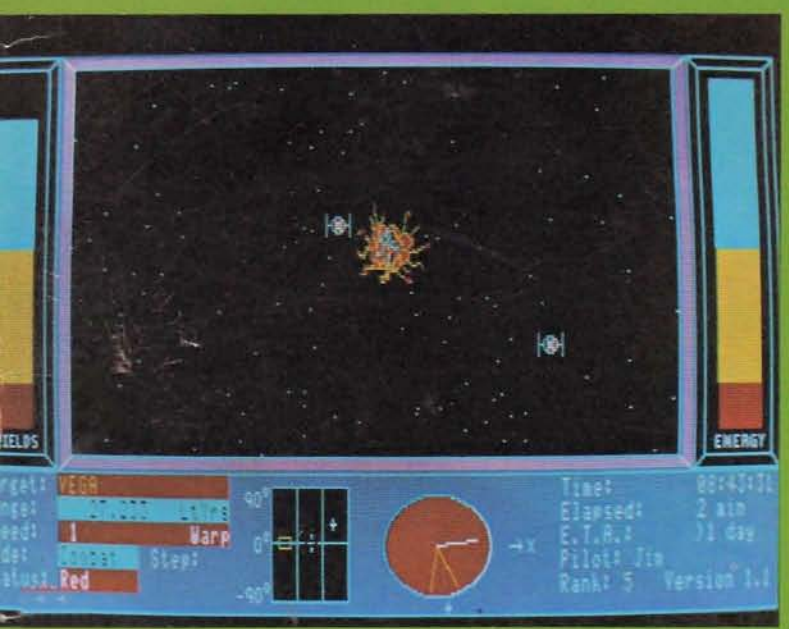
$2.50

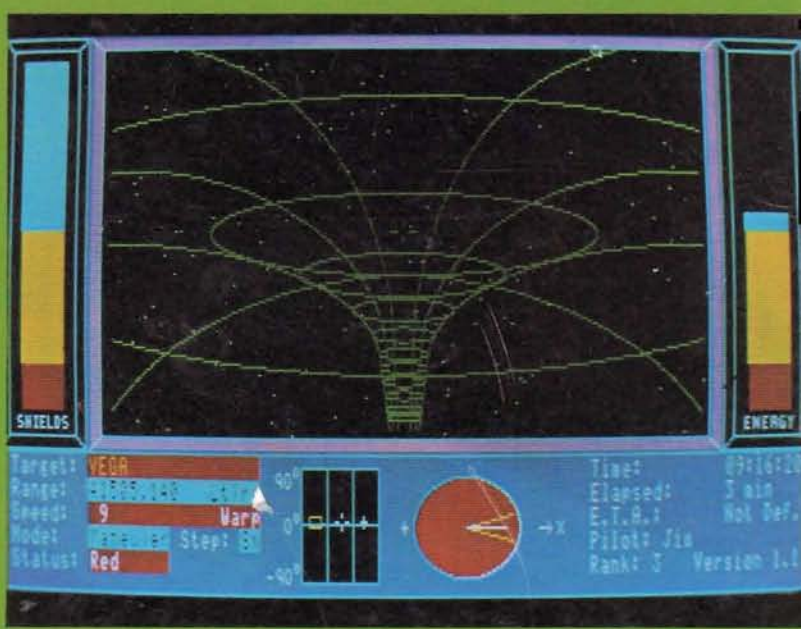
REMark®

Volume 5, Issue 4 • April 1984

P/N 885-2051







Official magazine for users of HEATH ZENITH computer equipment.

Support and More

From the Heath/Zenith Compatibility Leaders

H8 PRODUCTS

The Most Extensive Line of Hardware Support for the H8®

- **DG-80/FP8**
  Z80® based CPU including the powerful FP8 monitor — both only $199.00. The acclaimed FP8 monitor package is included with the DG-80 CPU.
- **DG-64D/64K RAM Board**
  Reliable, Low Power, High Capacity Bank-selectable RAM
  Priced from $233.00 (0K) to $299.00 (64K)
- **DG Static 64**
  Fully Static, High Capacity, Bank-selectable RAM. Also can be used as EPROM/PROM board (2716 type EPROMS). Priced from $199.00 (0K) to $499.00 (64K).
- **DG-32D/32K RAM Board**
  Low cost, Dependable RAM for the H8 32K Version Only $179.00.
- **DG-ADP4**
  H17-4 MHz disk adaptor — $19.95

THE SUPER 89

The DG SUPER 89 is a replacement central processor board for the Heath/Zenith 88-89 series of computers. The DG SUPER 89 offers advanced features not available on the standard Heath/Zenith 88-89 such as 4 MHz operation, real-time clock, optional AM9511A arithmetic processor, up to 256K of bank selectable RAM with parity check, and HDOS, CP/M and MP/M compatibility. By incorporating current state-of-the-art technology available for the Z80, the DG SUPER 89 offers the user increased speed and system reliability for years to come. Full compatibility with all Health/Zenith software and hardware products is designed into the DG SUPER 89. Electronic Disk Software included. Priced from $829.00 (128K) to $989.00 (256K).

HEARTBEAT

The DG Heartbeat is a compact computer system designed to be hardware and software compatible with the popular Heath/Zenith Z89/90 computer product line. The Heartbeat offers advanced features not found on the standard Heath/Zenith computer such as 4 MHz operation, real-time clock/calendar, two RS-232 serial ports, five peripheral expansion slots, 128 Kbytes (expandable to 256 Kbytes) parity checked RAM and provisions for an optional AM9511 Arithmetic Processor. Compatible with HDOS, CP/M and MP/M II (Multi-user) operating systems. Electronic Disk Software included. The Heartbeat may be used with most popular video terminals on the market although the Heath/Zenith H/Z19, H/Z29 and ZT-10/11 video terminals are recommended for full Heath/Zenith software compatibility. The Heartbeat cabinet design provides for inclusion of hard and/or floppy disk drives as well as other desired peripheral interfaces and is color-coordinated for use with the Zenith Z29 and ZT-10/11 video terminals. Priced from $1350.00 (Basic Unit).

D·G ELECTRONIC DEVELOPMENTS CO.

Ordering Information: Products listed available from DG Electronic Developments Co., 700 South Armstrong, Denison, Tx 75020. Check, Money Order, VISA or MasterCard accepted. Phone orders call (214) 465-7805. Freight prepaid. Allow 3 weeks for personal checks to clear. Texas residents add 5%. Foreign orders add 30%. Prices subject to change without notice.

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

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

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

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

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

*U.S. Funds.

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

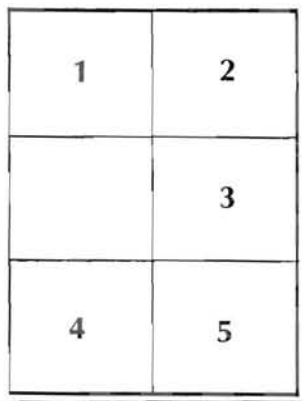
Limited back issues are availble at $2.50 plus 10% handling and shipping. Check HUG Product List for availability of bound volumes of past issues. Requests for magazines mailed to foreign countries should specify mailing method and appropriate added cost.

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

616-982-3463

REMark®
Volume 5, Issue 4 • April 1984

on the stack



ON THE COVER: A series of photos depicting Vega-Bound 1. See page 7 for an in-depth review.

1. One of the navigation screens used for selecting a new target star for the computer to "lock" onto.

2. In a cavern, I battle for my life against attacking allens. If I survive, I'll obtain their book of knowledge and be eligible for advance in rank.

3. I have already picked up some valuable findings on this planet and am taking off from a landing point. The defense tower is to my right and the pesky flying saucer is to my left.

4. Bingo! Got one of the enemy space ships on my way to Vega. Two more to go before I can resume normal speed.

5. Oh, oh. Black hole ahead and no way to avoid it.

$5 $5

WHY NOT SAVE SOME $ AND HAVE FUN AT THE SAME TIME?

CASTLE-GOR (CP/M only)
The unique and challenging adventure game. List $24.95
with coupon only **$19.95**

FOOTBALL (HDOS, CP/M, ZDOS)
America's most popular game with real-life action on your computer. List $24.95
with coupon only **$19.95**

SPACE WARRIOR (HDOS, CP/M)
Fast-action game of strategy as you try to capture the Galaxy for the Federation. List $19.95
with coupon only **$14.95**

Make your selection (as many as you like)

☐ **CASTLE-GOR**
☐ CP/M(—17) ☐ CP/M(—37) or CP/M—85

☐ **FOOTBALL**
☐ HDOS ☐ CP/M(—17)
☐ CP/M(—37) or CP/M—85
☐ ZDOS

☐ **SPACE-WARRIOR**
☐ HDOS ☐ CP/M(—17)
☐ CP/M(—37) or CP/M—85

$5 $5

$5 $5

NOW YOU CAN SAVE DISK SPACE & SOME $

with **ARCHIVE-80** you can back-up your disk files and save disk space while doing it. Typical file compression can be as high as 50%.

☐ **ARCHIVE-80**
List $24.95
with coupon only **$19.95**

☐ CP/M (—17) ☐ CP/M (—37) or CP/M—85

$5 $5

CLIP A COUPON TODAY & SAVE!

VISA **For Faster Delivery Call 906-249-9801** master charge

Add $2.00 for shipping. Michigan residents please add 4% for sales tax. COD orders are welcome.
PAYMENT IN U.S. FUNDS PLEASE
Include signature, card number, and expiration date for all credit-card orders.
Please specify operating system (HDOS, CP/M, or ZDOS) and disk format (hard or soft-sector) when ordering.

GENERIC SOFTWARE
P. O. Box 790 - Dept. 484R
Marquette, Michigan 49855
906-249-9801
10AM - 5PM EST

$30 $30

THE BEST LOW-COST GENERAL DATABASE SYSTEM*

Now you can save $30 on **AUDIT**, the flexible, easy to use database system.

☐ **AUDIT** List $129
with coupon only **$99**
☐ CP/M(—17) ☐ CP/M(—37)
☐ CP/M—85

*as rated by a recent CHUG evaluation of database programs available for Heath/Zenith computers.

$30 $30

FREE FREE

1984 SOFTWARE CATALOG

Over 20 software products for your H8, H89, Z90 and Z100 computer.
Call or write for more information!

FREE FREE

FREE FREE

SOFTWARE AUTHOR'S KIT

ATTENTION SOFTWARE AUTHORS

GENERIC SOFTWARE is interested in high quality and well documented software for HEATH/ZENITH computers. GENERIC offers professional packaging, and high royalties. If you are interested in making money from the software you develop, then request our **FREE** booklet, **"SOFTWARE AUTHOR'S KIT"!**

FREE FREE

$5 $5

CAN WE TALK?

You sure can with **GCOMM!** You can talk with other H8/H89 computers running **GCOMM**, as well as bulletin board systems like MicroNet and the SOURCE.

☐ **GCOMM** List $24.95
with coupon only **$19.95**
☐ CP/M(—17) ☐ CP/M(—37)

$5 $5

$5 $5

High Quality TDK Disks

5¼" Soft-Sector Disks
Prices Per Box of 10

☐ Single-Side/Double Density Boxes — Catalog Price $31.00
with coupon only **$26.00**

☐ Double-Side/Double Density Boxes — Catalog Price $37.00
with coupon only **$32.00**

Limit 4 Boxes/Coupon
While Quantities Last

$5 $5

BUGGIN' HUG

Color Graphics For the H8 and H/Z-89

Dear HUG,

During the past HUGCON and CHUGCON, a number of people asked if there was a way for people to share information and programs relating to Heath H8 and H/Z-89 color graphics. At the time, I had to answer 'not really' as there wasn't any organized disk library specifically set up for graphics programs although some have been submitted to HUG for distribution. The only information sharing has been short notes and occasional items in the REMark and Sextant magazines. Due to the number of requests I received, I prepared a questionnaire which I sent to all the people I knew had color graphics boards.

Based on these questionnaires and quite a number of follow-up letters and phone calls, I found the graphics users wanted a place where they could get next to anything in the public domain, even incomplete programs and subroutines. I already have two volunteers for collecting and distributing such program code, one in VA and another in Canada. It looks like this area is well covered.

Another item which which was requested was a list of graphics users and their interests. At least for now, I am handling that myself.

The last main item was a way of sharing information. Almost all respondents to the questionnaire indicated that they would like to see such information passed along in an existing publication. REMark was clearly the preferred publication. Based on these inputs, I ask this question. "Would REMark be willing to have a 'graphics corner' to handle the sharing of such information?" Clearly, I cannot predict with accuracy what would be submitted. The following items seem to be a reasonable guess based on the letters I have received and numerous discussions on the phone.

1. Contents and status of the public domain program library.

2. Reviews of 'for sale' graphics software.

3. Questions and answers to questions relating to graphics.

4. 'How to' tutorials.

5. Helpful hints and ideas.

Vectored to 39 ☞

INTERNATIONAL HUG CONFERENCE

Official Conference Registration Form

Pheasant Run Convention-Resort Hotel

July 27, 28 and 29

Name(s) _____

Address _____

Company _____

City _____ State _____ Zip _____

Enclosed is $22.00 per individual to attend the International HUG Conference to be held the weekend of July 27, 28 and 29, 1984. Please send tickets along with information regarding hotel reservations and transportation.

Amount Enclosed: _____ Number attending: _____

For our information:

Which Heath/Zenith computer do you now operate? _____

Are you a Non-User-Attendee? ☐Yes ☐No

Are you a Heath/Zenith related vendor? ☐Yes ☐No

If yes, do you want exhibit space during the Conference? ☐Yes ☐No

Special Notice to Vendors:

Vendor Information Packages will be made available to Heath/Zenith Related Vendors who are planning to exhibit their products while at the Conference. You must contact us prior to May 1, 1984.

For your information:

The $22.00 you are paying for your reservation to the International HUG Conference entitles you to all functions of the Conference. This includes Saturday breakfast, buffet lunch and hors d'oeuvres in the evening. The Prize Drawing will be held during the Saturday evening Cocktail Party. You must be present to win. Vendors and $22.00 ticket holders will be eligible for prizes. ALL prizes will be awarded at that time.

Visitor tickets, for those of you simply attending the seminars and looking at the exhibits, are available for $10.00. Visitor Tickets do not include meals or eligibility to the Prize Drawings.

Send your registration form or a suitable copy to:

Heath/Zenith Users' Group
Attention: International HUG Conference Registration
Hilltop Road
St. Joseph, Michigan 49085

Registration(s) must be postmarked no later than July 15, 1984. Cancellations will not be accepted after this date.

Professional Graphics at Practical Prices

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

Check Imaginator's special features:

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

Now check Imaginator's low cost:

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

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

Call or write us today for additional information.

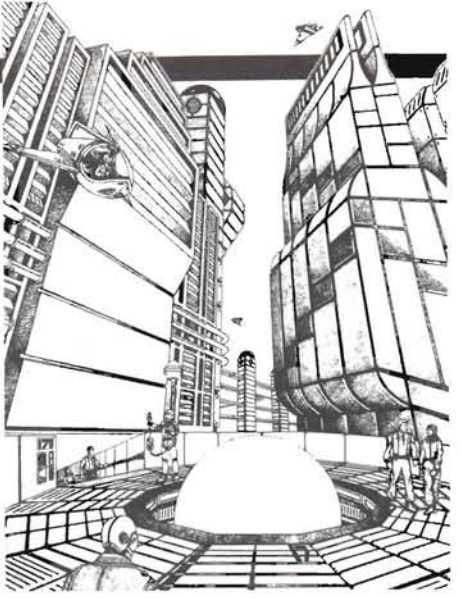
CLEVELAND CODONICS, INC.
18001 Englewood • Cleveland, OH 44130 (216) 243-1198

Your H-19 Can Do This...

IMAGINATOR

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

Photos courtesy of Interdiscipline, Inc.



Vega-Bound 1

Tom Huber
Related Products Editor

So you want to fly your own spaceship, but you're not up to meeting the requirements of NASA to qualify for the shuttle program or you can't find that pesky formula for time travel so you can journey into the world of Star Trek?

With the Z-100's graphics, somebody, you say, just somebody has got to come along with a decent space game that will give you realistic graphics and some fun at the same time, all with moving from your favorite computer, right? Right!

Well, a game has been created, and it is for the Z-100, and it uses the Z-100's superior color graphics to produce realistic images on the screen, and it employs decent adventure, arcade action, and speed to be a welcome game. The name is Vega-Bound 1, and it is a new game from Interdiscipline, Inc. It comes complete with a twenty-eight page Operator's Manual and a non-bootable disk for Z-DOS. It requires a Z-100, 128K system RAM, ROM version 1.2 or later, Z-DOS version 1.0 or later, one disk drive, and 32K or 64K of video RAM. Color helps and adds immensely to the enjoyment of the game but is not absolutely necessary.

Many of you who have been around computer space games for any length of time are familiar with the typical "Space War" games written along the lines of the popular "Star Trek" television series. The first such game that I ran appeared in DEC's "101 BASIC Computer Games" (DEC, 1973). The objective of such games was simply to track down and destroy enemy spaceships within a given universe. Stars were randomly placed in the quadrants as they were entered, along with enemy ships and an occasional space station where you could refuel and repair a damaged spaceship. The problem with these games is that they offered a limited challenge and dealt almost exclusively with a fantasy universe. Time and space were not realistically portrayed. The game waited patiently for user entries or until the computer was shut down; space was flat, two dimensional. Later versions of the games appeared with arcade action, but movement was still limited and stars and enemy ships appeared randomly.

Since then, a number of interesting, if not challenging programs and articles have appeared, among them, "Space Maze", Creative Computing, January, 1979, and "A Simulated View of the Galaxy", Byte, April, 1979.

I adapted the latter's FORTRAN algorithm to Radio Shack's Level II BASIC (a Microsoft derivative) and most recently to Z-BASIC on my Z-100. The program allows you to plug in the real space coordinates of actual stars and then move to various points in space and observe the results on the computer's screen. What you end up seeing is rather remarkable, literally a view from any imaginable point in space, including the Earth; the neighboring galaxy, Andromeda; or some point in deep space with both galaxies on the screen. Unfortunately, even in the compiled languages, real-time movement is not possible due to the large number of trigonometric calculations and conversions necessary to place individual stars in their correct location on the screen.

The game "Space Maze" offers a similar challenge by orbiting ten "killer" satellites around a space station. If you come within 1,000 kilometers to any satellite, it self destructs along with your spaceship, floating mines in space (with proximity fuses), so to speak.

The next best thing, then, is to create a pseudo-universe with real coordinates and a very limited number of real stars. Then the trick is to fly to each star, locate a planet, land on it and recover any valuables that you might locate. And that is what Vega-Bound 1 does; you are working with a field of nine stars ("real" in some cases) against a backdrop of (apparently) random stars -- I didn't check the star patterns against any star maps, but, because I didn't recognize any of the constellations (for instance, Orion, when traveling toward Sirius), I suspect the star patterns are not representative of the real night sky.

Vega-Bound 1 is actually three separate game "modules" that make up a whole. This is the first of its kind, and while it is a very good "first", it could stand even more variety.

The object is simple, go out into space, find an inhabited planet, obtain valuables and the book of knowledge from the local natives, and return to your home port (Earth). You may select one of six alternate star groups, several of which are "real" and several are those which make up interesting star patterns. There is only one inhabited planet (besides the Earth) in each star group, apparently selected at random from eight other stars in that group. The ninth star is always our own sun, the home base.

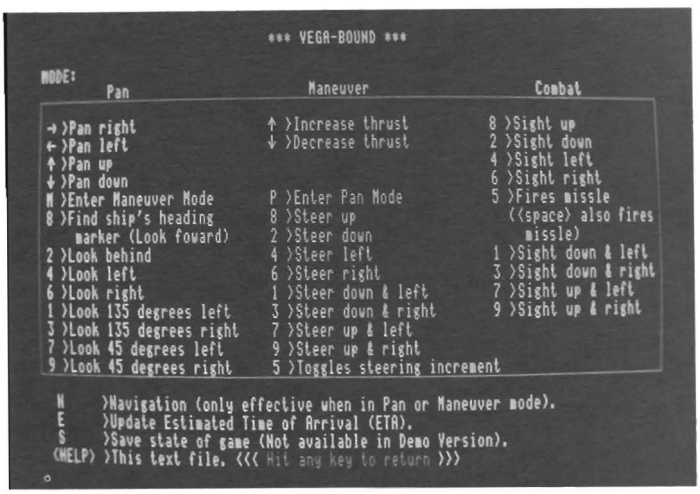
Flight is accomplished by accelerating to warp speeds and watching one's fuel supply (it goes down rather rapidly over large distances at high rates of speed), updating the ETA, and, when possible, keeping the screen's cross hairs on the target star.

On the way to locating the inhabited planet, you might run into a band of "pirates", whose ships look similar to certain spaceships from a science fiction movie. Shooting down these guys is a little like shooting ducks in an arcade; as long as you are fast enough, you'll easily get them before they get you. (They don't use very heavy artillery against your ship's guns and shields.)

The only other objects I ran into was a black hole or two, an interesting phenomenon that "throws" you light years from your target. According to the documentation, one might also come in contact with Ion Storms, have to run through fields of asteroids, and

sight a pulsar. In playing this game over several weeks, David and I failed to come upon any of these latter three phenomenon. I suspect that some of them are located in only certain parts of the universe and will appear only under certain circumstances.

As you approach each star, you end up having to slow down and (in many cases) manually maneuvering your ship close to the star (less than a billion miles). Then, under automatic guidance from previously (and automatically) fired probes, your ship will orbit the most likely planet of that star that might contain life. You will be asked if you want to send out a lander or leave orbit. If you send a lander and if the planet has life, the action is transferred to the second phase of the game, landing.



The "Help Screen". Nice touch, but not needed more than once or twice, particularly, since the 28-page instruction book was more than adequate.

Landing is accomplished while avoiding a pesky flying saucer, reminiscent of a leech (it hunts you down and then latches on), and slowing moving missiles fired from an attack tower. The tower is raised to the surface once you enter the atmosphere and immediately starts shooting at your landing craft. The missiles are relatively slow moving, so it is fairly easy to dodge them, land at various points on the surface and pick up goodies. I once recovered two containers of oxygen before destroying the tower and revealing the entrance to the caverns, the third phase of the game.

In the cavern, landing the craft is simple, but rather difficult as the action is a lot closer and more limited than on the surface. It is easy to destroy your landing craft by coming down at an angle (moving to the left or right). Once landed, a flock of "birds" (aliens) attack with bombs. Destruction of the incoming birds and their deadly cargo is taken care of by a handy gun on board. During one reprieve in the attack, however, a load of deadly stalactites fall from the ceiling of the cavern, evidently loosened by the destruction of the birds. A final attack is launched before a last, lone alien flies out with a white flag, surrenders and presents your lander with the Book of Knowledge. An automatic, quick flight back to the surface and up to your spaceship ends that round, except for a calculated flight back to the home planet.

Depending upon the amount of booty you return with, you may or may not be able to advance your rank. Rank is established by entering a number between 1 (lowest rank) and 9 (highest) after you enter your name at the beginning of the game. Unfortunately, rank in the name is not automatically handled, nor are there any provisions for a continuation of your character from one mission to another; each mission is an entirely new game.

If you have only a few minutes and want to start playing a mission, or somebody calls you to dinner in the middle of exploring a star field, or you want to shut down the Z-100, you can save an individual mission on disk. However, only one mission can be saved on a disk at a time. Furthermore, you can't control the name of the mission.

Evaluation

First impressions of the game are great, the graphics are fine for the Z-100, though not necessarily spectacular. This isn't bad either, because it leaves a lot of room for improvement. Apparently, paging is used for some of the graphics (requiring 64K elements in the video memory), but the way the Z-100 is set up does seem to matter. The program works well, even with 32K video memory elements.

The action of the game is also quite good, outstanding in some respects, but not as great in others. Real time is handled in one-second increments and when it comes to increasing or decreasing your speed, sometimes you need split-second action to help out. Onscreen action is very smooth, showing that the authors spent a lot of time perfecting the animation. Animation? Yes, I said animation, which is very well done.

Three-dimensional navigation is not as easy as it may seem, especially when you are traveling toward a very small object that is far away. A fraction of an angle off one way or another can place you a long way from your target, so course corrections must be expected. Speed is a bit tricky, too. I already mentioned the need for split-second action, and sometimes, the once-a-second updating will shoot you well past your target star. But these are all tricks that must be learned. Also, triggering the navigation screens doesn't stop your movement through space, just the updating of the screen and distance indicators -- a nice "real life" touch to the game.

Black holes end up being a real challenge (and a pain in the neck at higher ranks, they appear far too often to be enjoyable or funny). One of the other fellows that helped me test this game reported that he was able to avoid falling into black holes through a rapid 90 degree course correction. Frankly, I was never able to avoid any of them (traveling too fast, I guess), so I just put up with the trip through the hole and navigated my way back to the star.

I mentioned randomness that was present in many earlier games. The initial setup of the nine stars (other than location) appears to be random (in terms of which has inhabited planets and where the black holes have been placed in space), but otherwise, everything stays put (except, of course, for the spaceship that you pilot around the universe).

The viewport (not a typical screen, by the way) allows you an unlimited view around the ship, but not above and below you. Unfortunately, the ship cannot be tilted from the plane of the eliptic, so it is always parallel with the celestial plane (the only unrealistic part of navigating the ship). This can present you with situations where the star you are traveling toward cannot be viewed out the ship's viewport unless you are a very, very long distance from it. But getting out long distances and then coming back is very costly in terms of fuel use. And, you do have a limited amount of fuel; enough for most missions, but still limited.

There is actually so much to this game, that one must start analyzing it (as in trying to write a review) in order to really appreciate it. Yet, the game is amazingly simple to master. It is the type of game that once mastered, it can be put to one side where it may sit for quite a while. Eventually, it draws you back to it again.

I could easily write a small book about all the in's and out's of this game and their ramifications, but neither time nor space in this

magazine will allow for it. In a capsule, the game is an excellent launching point for which I hope we see many more games of a similar nature. The documentation, while not perfect, is more than adequate. The inclusion of a "help" screen is a nice, but unneeded touch. All-in-all, a good "starter game" for a new company to the Z-100. Let's see more!

Just a few side thoughts you might be interested in. Two packages aren't acknowledged although they appear to have been used in creating the game (I don't have proof one way or another to the following other than outside appearances): I think the game was written in Z-BASIC and then compiled. It runs very quickly and smoothly as a result; the Z-BASIC compiler is really very good and this game proves it. The animation is very good and I believe the authors used SYMED from New Horizon Software to create the action, again with very, very good results.

Aside from the minor frustrations already mentioned, I have just one major complaint: There is only one planet surface and just one cavern. With the capacity of the Z-100's disk and some innovative programming, I would have liked to have seen at least two or more planet surfaces, and perhaps three or four caverns with different attack routes. The sameness wears thin after the first couple of days, but like the first Star Trek and Star Wars movies, one is drawn back for replays time after time after time.

Vendor: Interdiscipline, Inc.
403 S. Brandon
Seattle, WA 98108
(206) 763-2099
Price: $49.95

DISKSORT
A Speedy File Sorter

Ralph Rumpf
Zenith Software Consultant

Once in a while I really need a fast sorting program that's easy to use, gives reliable results, and doesn't take forever to sort a large file of records. I've tried a few in the past with varying results, but none of them ever seemed to quite hit the mark. Enter DISKSORT.

DISKSORT is a sort/merge program that has a wide range of capabilities and applications from alphabetical listings to mail lists and a number in between. DISKSORT is available from Sunflower Software, Inc., 13915 Midland Drive, Shawnee, Kansas, 66216. DISKSORT was written in the C programming language and that is what interested me enough to give it a try. The C programming language is a high-level language with the responsiveness of an assembly language program. It can be compact and very fast.

DISKSORT Ver 2.0 is provided in a number of formats. It is available for HDOS, CP/M, Z-DOS, or PC-DOS and performs essentially the same way on each operating system. That can eliminate a lot of confusion when trying to work with a number of different computers. The following files are provided on the distribution disk:

DISKSORT.ABS/COM/EXE
DISKSORT.BAK
DISKSORT.HLP

Example files (10 are included)

The executing program is provided in a format for your operating system with a backup in the event of an accident, also a help file is provided in the event you get lost.

The Manual

The manual provided with DISKSORT is brief and to the point, the same manual is good for whatever version you are using. The manual may be brief but it is well written and organized. All the information you will need to master DISKSORT is provided, along with a little additional technical information on the program itself. The manual provides an introduction, a command summary, and a tutorial of sorts using the provided example files. I was pleased with the manual for the most part but I could not seem to find a description of the computer system requirements (if any) necessary to run the program. I had no problems with my version (CP/M running on an H89 with 17/37 drives) but I would hate to purchase a program and find I required some other piece of equipment to make it work.

Program Commands

DISKSORT may be operated in either Command Mode, Manual Mode, or a combination of the two. In the Manual Mode, DISKSORT is essentially menu-drive and prompts the user for all but a very few inputs. The few commands that cannot be entered in the Manual Mode are entered from the command line and affect default conditions that the user would not normally need changed. Here is a list of the full range of commands available in DISKSORT.

input file	fixed record length
output file	keep tabs
input & output file	expand tabs
merge file	suppress messages
drive for temp files	retain case for sort
single line group	lower=upper case for sort
multi-line group	continue after error
inter-line delimiter	quit after error
sort field begin	retain input file
sort field width	delete input file
sort field line #	max line length
ascending order	max line number
descending order	help
variable length	

As you can see, that is a fairly large number of commands. From the Manual Mode this is no problem, since you are prompted for each command the program requires, based on your inputs. In the Command-Mode this is a different matter. Each of the above commands is represented with a one or two character input line. For example, a typical Command-Mode entry might look like this:

```
DISKSORT *TEST1 ~B S I/ B1 W80 A V E R C Z
```

How's that for a cryptic line? I'm sure with a bit of time you can get used to it, but I would still want to have the manual handy just in case. I would have preferred to see this type of entry accomplished somehow through function keys, but I get all upset when I have to repeat a procedure because I misunderstood it or made an improper entry. I'm also a believer in the easier way, so that's just me. The above command line directs DISKSORT to do the following.

Use Test1 as the input and output file name, use drive B for temporary files, use single-line groups with an inter-line delimiter of /, begin the sort on column 1. The field is 80 characters wide, and the sort is in ascending order. The lines are of variable length, and the tabs are expanded, do not change the letter case for the sort but continue after a non-fatal error, and it erases the input file after the sort is done.

Whew... all that and in one breath too! Like I said before, I'm sure that you could get used to this format after a while (I never did well in foreign languages either). The Command Mode does offer an advantage however. In this mode you can use the DISKSORT program in Submit files in CP/M or Batch files in Z-DOS or even Docom files in HDOS. That would provide an advantage in processing some types of files to be sure. Since I don't personally have that many to process, I avoid the whole process and use DISKSORT in the Manual-Mode. I like it just as well and it isn't any slower as far as I can tell. DISKSORT can also merge two files into one and sort them as it does so. That's real handy when you want to add more items to a list you already have sorted.

The Program

DISKSORT is easy to use. Again, I use it in the Manual Mode and it is fast! In order to get an idea of how fast it is, I prepared a random order file of 263 single line entries of variable lengths up to 80 characters long. Then I let DISKSORT do its thing. I told DISKSORT to sort the file in ascending order based on the first character position with expanded tabs and no case conversion for sorting. Well, I was a bit taken aback. In approximately five seconds DISKSORT said that it was done. WHAT?!... So, I read the sorted file. Sure enough, there were all 263 records in ascending order, A - Z. Well, one of the toughest sorts is a reverse order sort to completely reverse the order of the file, and I already had it in ascending order...AH..HA! So, I told DISKSORT to do it over the same way except in descending order. Five seconds later, guess what? Yup! Only Z - A this time. I thought that my watch had broken, (or I had) so I tried it again, then I tried multiple lines and different groupings...it didn't bother DISKSORT. It continued to sort the file, in the way I requested in about the same time. I was impressed, the program really works! The algorithm that DISKSORT uses is one of the fastest known and actually gains speed as the size of the file grows. Needless to say, the program can sort a file before you can get a cup of coffee (maybe even before you can whistle Dixie).

Conclusion

DISKSORT is an impressive program that is speedy and easy to use. If you should have occasion to need a sort/merge program to handle a number of files in short order, then you might give DISKSORT a second look. Although somewhat cumbersome in the Command Mode, in the Manual Mode it is a breeze to use and I don't think that you will be disappointed when you watch it sort a file that really needed sorting. In fact, you just might find a few more.

Vendor: Sunflower Software Inc.
13915 Midland Dr.
Shawnee, KS 66216

Price: $49.95

Machines: Zenith (ZDOS) (CP/M) (HDOS)
H/Z-100, H/Z-89, H-8

Sign up now for the
INTERNATIONAL HUG CONFERENCE

see registration on page 5 of this issue

Special Notice

HUG will no longer accept program submittals for the Software Library for the H11 or cassette systems.

A Review of WatchWord, a Word Processor

Watching Your Words

Pat Swayne
HUG Software Engineer

WatchWord is a text editor and word processor for the Z-DOS operating system from S & K Technology, Inc. WatchWord provides all of the features that you expect from a word processor (super and subscripts, underlining, boldface, centering, justifying, etc.) with the added ability to show those features (or at least, an approximation of them) on the screen. Even though WatchWord is priced lower than many other word processors, it offers advanced features including powerful macro capability and split screen editing of two files at once or two places in the same file.

Before I get on with this review of WatchWord, I think I should tell you that my favorite editors are WordStar, as modified by KEYMAP, and the HUG editor. I normally use WordStar/KEYMAP for writing documents and the good old HUG editor for writing assembly source code. However, this article was written using WatchWord, so that I could evaluate it in a normal work situation. (I must admit that I am biased towards WordStar/KEYMAP because I wrote KEYMAP, but the function/keypad layout is not entirely my own idea. A modified test version of WordStar was produced by ZDS that had the keypad layout I used for KEYMAP. In my opinion, it is the most logical use of the keypad I have seen for editing. By the way, the new version of WordStar for Z-DOS does not use that layout, which was a bit of a disappointment to me, and hastened the production of the Z-DOS version of KEYMAP.)

My first impression of WatchWord is that it is very complex. By that, I do not necessarily mean that it is difficult to use, but that there is a lot to it. It is fairly easy to get started with, and to do a job such as writing this article, but it would take a while to learn every feature and be able to use them without referring to the manual or pressing the HELP key (more on that later).

Editing a file with WatchWord can be started either by specifying the file name on the command line when you invoke WatchWord, or by entering the file name after you start. When WatchWord starts up, it divides the screen into three sections: the text section, the command section, and the status line. The status line, at the bottom, shows the line and column of the cursor in the file, the name of the file, the size of the file, the free space in the buffer, the current time, and an indication of the condition of two modes (input and insert). The size and free space are shown as nn+mm where nn is a count of 256-byte "pages" and mm is the number of bytes in the last 256-byte page. When I first started up WatchWord, I multiplied the nn number in the free space part by 256 and found that I had over 80k of free space. It is obvious that WatchWord is one of the few programs around that can really make use of the 8088 processor's large addressing space. It is also obvious that the reason for the unusual nn+mm scheme for indicating space is that the routine for printing numbers in decimal is limited to 16 bits, or 65535 max. I guess I'll have to publish a 32 bit decimal number routine in REMark one of these days.

Immediately above the status line, and separated from it by a solid line, is the command section which is not only for entering commands, but displays error messages. Above that, and separated from it by a "squiggly" line, is the text section. The squiggly line gives you an indication of how WatchWord handles screen video, because it is not made up of H19 graphic characters or any other character built into the machine, but by directly writing to the video RAM. By doing screen video that way, WatchWord is able to change the display quickly, and to put characters on the screen that are not part of the Z-100 character set. An enhanced video mode is available in which superscripts, subscripts, and underlining are shown as such, and boldface is shown as reverse video. In the enhanced mode, text appears to be double spaced on the screen to make room for underlining, superscripts, etc. Watchword uses its own font generator, and individual characters can be modified from within the program. Double height characters can be defined, which can only be viewed as they are while you are in the enhanced mode.

WatchWord has two split screen modes of operation. In one, the screen is divided into two complete sections, each with its own command section and status line. You can edit two completely different files in this mode, and transfer text from one to another. (WatchWord has full "cut" and "paste" capabilities in any screen mode.) In the other split screen mode, you can edit two parts of the same file. In this mode, there are two text screens, but only one command section and one status line.

When you start up WatchWord, the cursor is in the text section, and if you press the ENTER key on the keypad, it moves to the command section and the words Enter Command appear. Another press of the ENTER key places the cursor back into the text section, and the command section is cleared. WatchWord is what many refer to as a modal editor, because it has separate and distinct edit and command modes. Some commands can be entered through function keys while you are in the edit mode, but most must be entered by typing out the command after entering the command mode. The decision as to which commands would be assigned to function keys, and which would be invoked by commands does not seem to have been logically thought out, in my opinion. For example, if you want to center some text, you must press ENTER, type the word CENTER (upper or lower case), and press enter again. That kind of procedure is also required to re-format a paragraph. With WordStar/KEYMAP, all you do is press a function key that is labeled (on the 25th line) "Center" or "Format".

Here, for what it is worth, is my idea of how function and keypad keys should be utilized in an editor. Centering, indenting, formatting, underlining, boldfacing, setting margins, and other frequently used commands should be available at the press of a function key, and key functions should somehow be labeled on the screen. All cursor and text movement on the screen should be done with the keypad, including paging forward and backward and moving to the top or bottom of the file. Because of the positively awful arrangement of the arrow keys on the Z-100 keyboard (the only bad thing about it), the 2, 4, 6, and 8 keys should serve as duplicate arrow keys, providing an

easy to use diamond pattern.

With WatchWord, indenting is on the keypad, paging forward and backward are on the function keys, and centering is a command mode function. The whole arrangement seems illogical to me (no, I don't have pointed ears, but I have been accused of thinking Spockishly anyhow). Other aspects of WatchWord seem illogical as well. For example, if you move the cursor to the beginning of a line within your text while the insert mode is on and press return, you would expect (at least, I would) a carriage return- line feed to be inserted at the cursor, with the result that a blank line is inserted at the cursor. With WatchWord, however, a blank line is inserted below where the cursor is placed. Another problem is that WatchWord cannot delete around the beginning of a line. Sometimes while typing fast I will make an error and not discover it until the word processor I am using has wrapped me around to a new line. With WordStar, I can just lay on the delete key until the error is gone and move on, but with WatchWord, I must remove my hand from the keyboard, move to the keypad, move the cursor to the error, fix it, move the cursor back, go back to the keyboard, and continue typing.

Another problem with WatchWord occurs when you are in the word wrap (they call it the "Input") mode. If you move the cursor back somewhere within a line and type some text, word wrapping does not work, and your inserted line will go on and on with WatchWord moving things over as you go (lines in WatchWord can be up to 65535 characters long!).

Although function key usage may seem illogical with WatchWord, it is such a powerful system that it provides ways to fix the keys. For simple fixes, like adding a one-key FORMAT command, there are two user programmable keys, F0 and F1. For more complete key changes, commands are provided that enable you to change all of the function and keypad keys so that the layout is the way you want them. Once you have made your changes, a STORE CONFIGURE <filename> command saves them and a RECONFIGURE <filename> can read them back in. WatchWord has a very good macro system that enables you to write command to reconfigure, or whatever, into a file and have the commands in that file executed at startup. Other macro files can be executed during the editing process. Special printers are handled through macros that set up what WatchWord sends out to initiate boldface, underlining, etc. WatchWord supports Diablo and NEC printers in its default state, as well as "plain" printers, and a sample macro for EPSON printers is provided.

One "bug" in the version of WatchWord that I tried is in the way it works with mapped drives. My office Z-100 system has the Z- 217 Winchester installed, and I use a partition of it as my system disk. Since I do not like the idea of my system disk being called drive E:, I have an AUTOEXEC file that executes at cold boot and runs the MAP program to map my drives so that my two Z-DOS Winchester partitions are drives A: and B:, and the 5- inch floppy is drive C:. When I run WatchWord under these conditions, the message "Put B disk in drive A" is displayed, and I must hit RETURN. Then, when I exit WatchWord and try to access the floppy drive (C:), the message "Put A disk in drive A" is displayed, and I must hit RETURN again.

Documentation and Help

Documentation provided with WatchWord is somewhat "thin" for a program of such complexity. Users may shy away from some of the more advanced features because of brief coverage. For example, only about two thirds of a page is devoted to the subject of changing fonts. There are a lot of people buying computers these days who would have to be "stepped through" that sort of thing. Some of the operating information is not in the manual at all, but on the disk. There is no index, so a bit of "thumbing" may be required to find what you want. A Help feature is provided which is activated by the HELP key. When you press it, a menu of help topics is displayed, and items from it can be selected with function keys. All of the help topics are separate text files that must be present on the currently logged disk along with the WatchWord program itself in order for the help feature to work. Since the help files are ordinary text files, you can edit them to say what you want.

Conclusion

WatchWord is a complex feature packed editing program. I have not come close to mentioning all of the features it supports in this article. Because it is so feature packed, some users may find it overwhelming. However, since you can start using it without knowing everything, its complexity should not be the only factor you consider in evaluating it. The documentation is minimal, but the help feature can guide you through most problems you might encounter. The function and keypad keys, in my opinion, are not used logically, but they can be re-programmed to alleviate that situation somewhat.

WatchWord is priced much lower than WordStar or many other word processor programs, yet, if you have a letter quality printer, it can produce output that is indistinguishable from WordStar output. The pricing policy, however, deserves one final comment. It is sold for $100 in single copies, or for $50 each if 5 or more copies are purchased. I feel that this pricing policy is discriminatory towards users who live in rural areas, and that it also encourages pirating. If two people who are not really all that dishonest want to use WatchWord, and cannot find 3 others who want it, they may reason to themselves that it is OK for them to buy one copy and share it, since they have, in effect, paid $50 each to use it, the same as the 5 users in Big City.

WatchWord is available from S & K Technology, Inc., 4610 Spotted Oak Woods, San Antonio, TX 78249, phone (512) 492-3384.

ZD - A Sorted Directory Utility For the H/Z-100

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

```
A:ZD /VIP

Enter Title: ZDOS Master of Operations
ZD - Version 1.1 Jan. 1984                                    By Jeff Kalis

>>> ZDOS Master of Operations          Drive: A  Free:   1024   01/15/84 <<<
ALTCHAR.SYS    431   DEBUG.COM     6003   FORMAT.COM   10542   PRINT.COM     1740
BACKUP.EXE   49920   DSKCOMP.COM   1528   LIB.EXE      32128   PSC.ASM       5760
CHKDSK.COM    1754   DSKCOPY.COM  12350   LINK.EXE     41856   SYS.COM        637
COMMAND.COM   5114   EDLIN.COM     2313   MAKE.COM     13310   SYSCOPY.DAT    813
CONFIGUR.COM  7921   EXE2BIN.EXE   1280   MAP.COM       2291   ZD.COM        1216
CREF.EXE     13824   FILCOM.COM    8320   MASM.EXE     70784

A:
```

I have never regretted the decision I made to purchase an H/Z-100 computer. Since the day the kit arrived, I have been impressed with the ease of construction, the extremely well written manuals, and the power of the operating systems, both CP/M and ZDOS. I must admit however, that hearing about all the available programs for the IBM you know what had me a little down. That is until I successfully rewrote a PC utility for the Z100 and found that it actually worked. Three cheers for MS-DOS.

The January 1984 issue of Doctor Dobb's Journal contains a sorted directory utility for the PC. I present here a greatly enhanced directory utility for the H/Z-100 based on that utility. I say greatly enhanced because I have added a set of options so you may tailor the output to your own personal needs. ZD (Z100 Directory) takes care of all those little things that you wished DIR could do.

The basic operation of ZD is very simple. Just type 'ZD' and "wa la", there you have the directory listing you had always hoped for. Notice the new format, four columns across instead of one long one that always seemed to disappear off the top of your screen before you can stop it. Even if the screen should fill up, ZD will stop and wait until you ask it to continue listing the rest of the directory. Take notice of the best part of all, the files are sorted alphabetically. That's nice.

Now focus your attention on the first line of the listing. Not only does ZD tell you what drive you're looking at, but it also lets you know how much free disk space you have left. Kiss CHKDSK goodby, say hello to ZD. The current date is also given to you on that first line. There seems to be a lot of open space on the left side of that first line up there. Well, we can take care of that. Enter this command, 'ZD /T'. That's one of our available options. The '/' tells ZD that we want an option. The 'T' is an option to allow a user title on the directory listing. A 35 character title can be entered when ZD asks 'Enter title'. Just type in what ever blows your hair back and in that open space on the left side of the first line, your title will appear. Not much good when the listing goes to the screen, except maybe to practice your typing. But when you use the 'P' option, you get a titled directory listing on your printer. Now that's handy. Title the printout the same as the name on your disk and you can keep track of what programs are on which disk.

Here is a list of the options ZD will recognize:

C - Clear the screen before listing the directory
V - Displays the ZD version number
P - Sends the listing to the printer
F - Sends a form feed to the printer after listing
T - Allows the user to enter a title

To list to the printer with a user title, use 'ZD /PT'. As a matter of fact, you can use as many options in one command as you would like. There are only two stipulations. First, the '/' must be preceded by a space. If there is not a space in front of the '/', ZD will not recognize any of the options specified. Second, if an illegal option is put in the option field, all the other options that follow the bad one are ignored.

Alternate drive specs can also be used. 'ZD B: /VC' would cause the screen to be cleared and the directory of drive B to be listed on the screen along with the ZD version number.

Examples of bad commands would be 'ZD B:/CVTP' and 'ZD /CQVTP'. The first example does not have a space in front of the '/' so all the options would be ignored. The second example has a 'Q' in the option field which is an illegal option. In this case, ZD would accept the 'C' option but ignore the other options. Options and drive specs can be upper case or lower case letters. Spaces in the option field are ignored. A command that looks like this, 'ZD b: /vC tp' would be an acceptable command.

When using ZD, all files on the disk will be listed. ZD will not recognize specific or wild card selection of file names. Nor will ZD show any hidden files.

The source code looks like a lot of typing although it is not really that much. You can skip the comments to save some time and your fingers. After entering the source code, it must be assembled, linked, and then converted from an EXE file to a COM file. Don't worry about it. Here is a step by step procedure to follow.

1 - Enter the source code under the file name ZD.ASM using ED or some other text editor.

2 - Be sure the following programs are on the same disk. ZD.ASM

(you just created this file in step one). MASM.EXE, LINK.EXE and EXE2BIN.EXE. Also, make sure there are at least 50,000 bytes free on this disk because during the COM creating process, there are some files that are going to be created.

3 - Enter the command 'MASM ZD,,,;'. Notice the three ',' and the ';' after the 'ZD'. This will assemble the ZD source code. When MASM is finished, your screen should display the following message:

```
Warning Severe
Errors  Errors
0       0
```

Anything other than this indicates an error in the source file. Go back and find the typo, fix it and redo step three. If you should get an ERROR 108, you have a full disk.

4 - Enter the command 'LINK ZD,,,;'. This will start up the LINK program and generate an EXE file. LINK will report that there is no stack segment. That is right but you don't need one when you're developing a COM file. LINK will also report one error. This is the lack of a stack segment which we don't need anyway so ignore this error. If LINK says anything other than that, you have a mistake in your source file. Go back and fix it. You will also have to rerun MASM again.

5 - Now enter the command 'EXE2BIN ZD'. This will cause the file ZD.EXE to be converted to ZD.BIN.

6 - Change the name of ZD.BIN to a COM file using the REN command 'REN ZD.BIN ZD.COM'. And there you have it. Now that wasn't so bad, was it?

The listing contains a generous amount of comments so it should not be too difficult to follow if you want to dig into the innards. The first part sets aside some memory for flags and defines the messages used by ZD. There are also a couple of buffers set up. The actual program code begins at the START label. The DOS version is retrieved and saved. This is used later to decide which method to use for figuring out the disk's free space. Version 2.x has a service call that will give us the free space where version 1.x doesn't. Next we save the default drive so when ZD is finished, you are returned to the same drive.

Next ZD looks for the option delimiter '/'. If it is found, then the options are checked and the corresponding flags are set. If a different drive is specified, ZD makes sure it is a legal drive and makes it the default drive.

A few subroutine calls complete the data gathering and directory listing. The default drive is reset and we are sent back to DOS. GETTTL is the first subroutine to be called. Here ZD clears the title line and checks for the 'T' option flag to be set. If it is, the user is prompted to enter a title. If the 'T' flag is clear, ZD branches around the user title input code and goes directly to the area that puts the date into the title line.

GETFRE is called next and here is where the DOS version flag is checked. If we're running version 1.x, each FAT (File Allocation Table) must be looked at to find the free space on the disk. Version 2.x won't let you get to the FATs but has a system call that will give you the free space remaining. In either case, the routine CONVRT takes the free space, changes it to ASCII with any leading 0's (zeroes) blanked out, and stores it in memory pointed to by the DI register. CONVRT is also used by the print routine to display the file size.

All the files on the disk are gathered up and stored in a table by the SCAN routine. The SORT routine is a bubble sort which arranges an array of pointers. These pointers are used to point to the file names in the table. PRINT is the routine that formats the listing and determines if it should be sent to the screen or the printer. The 'C' option flag is checked. If it is set, the screen is cleared. The 'P' option flag is checked next. If it is set, the output device flag is changed from the default screen to the printer. A check of the version option flag is made next and the version is output if this flag is set. The total number of entries per column is figured and saved. The title line is sent out next. After that, the first line of entries is sent out. If ZD is sending its listing to the screen, one count is added to the line counter and a check is made to see if the screen is full. If the line counter has reached the set value, the '[More]' message will appear and ZD will wait until any key is pressed before the listing will continue. When dumping the directory to the printer, this check is not made. After all the file names and file sizes have been listed, a final check of the print flag is made. If this flag is set, the 'F' option flag is checked. A printer form feed is sent if the 'F' flag was set, otherwise we are finished.

If you want ZD and you really don't want to type the code in yourself, just send me a 5 1/4 soft sector floppy and a buck for return postage and I'll put the source and COM files on it for you. You can also send me a check for $6.00 and I will send you a brand new floppy with all of the ZD files on it, including this explanation. Feel free to contact me about any problems you may have with ZD.

```
        TITLE   ZD - Z100 Directory Utility - Version 1.1
        SUBTTL          By Jeff Kalis - January 1984
;
        PAGE    60,132
;
;               Based on 'SD' by Bruce R. Ratoff
;                 and 'COVER' by Dan Daetwyler
;
CODE    SEGMENT PARA PUBLIC 'CODE'
        ASSUME  CS:CODE,DS:CODE
;
        ORG     100H
BEGIN:  JMP     START
;
PNTRFG  DB      0               ;Flag to send listing
                                ;to printer
CLRFG   DB      0               ;Flag to clear the screen
TFLG    DB      0               ;Flag to prompt for title
FFFLG   DB      0               ;Flag to send form feed
                                ;to printer
VERS    DB      0               ;DOS version flag
CDRV    DB      0               ;ASCII of current drive
DDRV    DB      0               ;Default drive at entry
NDRV    DB      0               ;Number of drives in
                                ;system
LINCT   DB      0               ;Line counter for pause
OUTDV   DB      2               ;Default to CON
VERSFG  DB      0               ;Display version flag
;
VERSN   DB      'ZD - Version 1.1 Jan. 1984'
        DB      39 DUP (' ')
        DB      'By Jeff Kalis',13,10,0
NOFLSM  DB      '>>> NO FILES <<<',13,10,'$'
ERMSG1  DB      13,10,'Invalid drive$'
TPRMT   DB      13,10,'Enter Title: $'
CRLF    DB      13,10,0         ;
CLRST   DB      27,'E$'         ;Clear the screen
                                ;sequence
MMSG    DB      '[More]',0      ;Pause prompt
DBLK    DB      '  ',0          ;Space btwn columns
RESTR   DB      12,0            ;Printer forms
;
TBUF    DB      35,0            ;Input buffer for title
        DB      35 DUP (?)      ;
DFCB    DB      0,'???????????' ;Dummy FCB Buffer
        DB      24 DUP (?)
DBUF    DB      7 DUP (0)       ;File size buffer
;
TITLX   DB      '>>> '
TITL1   DB      37 DUP (' ')    ;
        DB      'Drive: '
DRV     DB      '  '            ;Spot for drive
        DB      'Free: '
```

```
FREE    DB      9 DUP (' ')     ;
MONTH   DB      ' /'            ;fields for date
DAY     DB      ' /'            ;
YEAR    DB      '   <<<',0
;
X10000  DW      10000           ;Conversion constants
X1000   DW      1000,100,10     ;
STKCNT  DW      0               ;Stack entry counter
;
;
START   PROC    NEAR
        MOV     AH,30H          ;
        INT     21H             ;Check the DOS version
        OR      AL,AL           ;
        JZ      NOTTWO          ;Jmp if ver=1.x
        DEC     AL              ;
NOTTWO: MOV     VERS,AL         ;Save DOS flag
        MOV     AH,19H          ;
        INT     21H             ;Get default drive and
        MOV     DDRV,AL         ;  save it
        MOV     DL,AL           ;
        ADD     AL,'A'          ;Convert to ASCII
        MOV     CDRV,AL         ; and save it
        MOV     AH,0EH          ;
        INT     21H             ;Get number of drives
        MOV     NDRV,AL         ;  and save it

        MOV     BX,0080H        ;Point to command buffer
        MOV     CL,BYTE PTR [BX] ;Get length of command
                                ;buffer
        XOR     CH,CH           ;
CKNXT:  INC     BX              ;
        DEC     CL              ;Search for the option
                                ;delimiter
        JS      CKDRV           ;
        CMP     BYTE PTR [BX],'/'
        JNZ     CKNXT           ;If '/' is found, a ' '
                                ; must
        DEC     BX              ; precede it else ignore
        MOV     AL,BYTE PTR [BX]
        INC     BX              ;
        CMP     AL,' '          ;
        JNZ     CKNXT           ;

CKOPT:  INC     BX              ;Here only on valid '/'
        DEC     CL              ;Check option field for
        JS      CKDRV           ;  legal options
        CMP     BYTE PTR [BX],' ';If char is space
        JZ      CKOPT           ; ignore it
        MOV     AL,BYTE PTR [BX] ;
        OR      AL,' '          ;force lower case
        CMP     AL,'c'
        JNZ     CKNT1           ;If char is 'C'
        MOV     CLRFG,0FFH      ; set the clear flag
        JMP     CKOPT           ;  then get next option
CKNT1:  CMP     AL,'p'
       JNZ     CKNT2           ;If char is 'P'
        MOV     PNTRFG,0FFH     ; set the printer flag
        JMP     CKOPT           ;  then get next option
CKNT2:  CMP     AL,'t'
        JNZ     CKNT3           ;If char is 'T'
        MOV     TFLG,0FFH       ; set the title flag
        JMP     CKOPT           ;  then check next
                                ;   option
CKNT3:  CMP     AL,'f'
        JNZ     CKNT4           ;If char is 'F' then
        MOV     FFFLG,0FFH      ; set the form feed flag
        JMP     CKOPT           ; then check the next
                                ; option
CKNT4:  CMP     AL,'v'
        JNZ     CKDRV           ;If char is 'V' then
        MOV     VERSFG,0FFH     ; set the version
                                ; display flag
        JMP     CKOPT           ;An illegal opt.
                                ; terminates scan
CKDRV:  MOV     BX,0080H        ;Point to command
                                ; buffer
        MOV     CL,BYTE PTR [BX] ;Get length of command
                                ; buffer
        XOR     CH,CH           ;

DVNXT:  INC     BX
        DEC     CL              ;Look for a drive spec
        JS      DFDRV           ;If ':' not found, use
                                ; default
        CMP     BYTE PTR [BX],':'
        JNZ     DVNXT
        DEC     BX
        MOV     AL,BYTE PTR [BX]
        AND     AL,0DFH         ;Force upper case for
                                ; display
        MOV     CDRV,AL
        OR      AL,' '          ;
        SUB     AL,'a'-1        ;Convert drive spec
        JNC     DRV0K           ; branch if it looks ok
ERR1:   MOV     DX,OFFSET ERMSG1
        MOV     AH,9            ;Send invalid drive msg
        INT     21H             ;
        INT     20H             ;
DRV0K:  CMP     AL,NDRV         ;Check for installed
                                ; drive
        JA      ERR1            ;
        DEC     AL              ;
        MOV     DL,AL           ;Make the selected
                                ; drive
        MOV     AH,0EH          ;  the default drive
        INT     21H             ;

DFDRV:  CALL    GETTTL          ;Generate title line
        CALL    GETFRE          ;Figure free space
        CALL    SCAN            ;Get the directory
                                ; entries
       OR      CX,CX            ;
        JNZ     FLSR            ; If the directory is
        MOV     DX,OFFSET NOFLSM ;  empty, print no
        MOV     AH,9            ;  file message
        INT     21H             ;
        JMP     WEREOT          ;
FLSR:   CALL    SORT            ;Sort 'em
        CALL    PRINT           ;Spit them out
WEREOT: MOV     DL,DDRV         ;Reset the default drive
        MOV     AH,0EH          ;
        INT     21H             ;
        INT     20H             ;Back to DOS
START   ENDP
;
GETTTL  PROC    NEAR            ;
        MOV     DI,OFFSET TITL1 ;
        MOV     CX,37           ;Clear the title line
        MOV     AL,' '          ;
        REP     STOSB           ;
        CMP     TFLG,0          ;Check for title opt flag
        JZ      TI1             ; Branch if not enabled
        MOV     DX,OFFSET TPRMT
        MOV     AH,9
        INT     21H
        MOV     DX,OFFSET TBUF
        MOV     AH,0AH          ;Get the users input
        INT     21H             ;  for the title
        MOV     CL,TBUF+1       ;
        XOR     CH,CH           ;
        MOV     SI,OFFSET TBUF+2
        MOV     DI,OFFSET TITL1 ;Move it to title line
        REP     MOVSB
        MOV     DX,OFFSET CRLF  ;Output a CR,LF
        CALL    DOPRT
TI1:    MOV     AL,CDRV         ;Get the current drive
        MOV     DRV,AL          ;Put it in title
        MOV     AH,2AH
        INT     21H             ;Get the date
        SUB     CX,1900         ; kill century
        MOV     DI,OFFSET YEAR
        MOV     AL,CL
        CALL    DECMAL
        MOV     DI,OFFSET MONTH
        MOV     AL,DH           ;Stick the month
        CALL    DECMAL          ; day and year
        MOV     DI,OFFSET DAY   ; into the title
        MOV     AL,DL
        CALL    DECMAL
        RET
```

```
GETTTL  ENDP
;
DECMAL  PROC    NEAR
        AAM
        OR      AX,'00'         ;Converts AL to two
        XCHG    AL,AH           ;  decimal digits and
       STOSW                    ;  stores 'em at SI
        RET
DECMAL  ENDP
;
GETFRE  PROC    NEAR
        TEST    VERS,1          ;Check for dos version
        JZ      VERS1           ; if its 2.x then
        XOR     DL,DL           ; we do it the easy way
        MOV     AH,36H          ;
        INT     21H             ;System call to get
        MUL     BX              ;  free space
        MUL     CX              ;Free contained in AX,DX
        JMP     VCOM            ;
VERS1:  PUSH    DS              ;Version 1.x
        MOV     AH,1BH          ;  get FAT
        INT     21H             ;
        XOR     AH,AH           ;
        XCHG    CX,DX           ;CX contains # of units
        MUL     DX              ;
        PUSH    AX              ;Save 'em
        XOR     AX,AX           ;
        MOV     SI,2            ;FAT entry
FAT2:   MOV     DI,SI           ;
        SHR     DI,1            ;
        ADD     DI,SI           ;Figure 1 1/2 bytes
        MOV     DI,WORD PTR [BX+DI]
        TEST    SI,1            ;Is FAT entry odd or
                                ; even
        JZ      FAT3            ;
        SHR     DI,1            ;
        SHR     DI,1            ;
        SHR     DI,1            ;12 nibble adjust
        SHR     DI,1            ;
FAT3:   AND     DI,0FFFH        ;
        JNZ     FAT4            ;
        INC     AX              ;Move to the next entry
FAT4:   INC     SI              ;
        LOOP    FAT2            ;Get the whole FAT
        POP     CX              ;
        MUL     CX              ;Figure total free bytes
        POP     DS              ;
VCOM:   MOV     DI,OFFSET FREE  ;Point to title line
        CALL    CONVRT          ;Make ASCII and store
        RET
GETFRE  ENDP
;
CONVRT  PROC    NEAR            ;Convert 6 digits to
                                ; ASCII
        PUSH    DI              ; and suppress leading 0's
        DIV     X10000          ;
        AAM                     ;
        OR      AX,'00'         ;
        XCHG    AH,AL           ;
        STOSW                   ;
        MOV     CX,3            ;
       MOV     SI,OFFSET X1000 ;
DIVLP:  MOV     AX,DX           ;
        XOR     DX,DX           ;
        DIV     WORD PTR [SI]   ;
        OR      AL,'0'          ;
        STOSB                   ;
        ADD     SI,2            ;
        LOOP    DIVLP           ;
        OR      DL,'0'          ;
        MOV     AL,DL           ;
        STOSB                   ;
        MOV     CX,5            ;
        POP     DI              ;
        MOV     AL,' '          ;
SUPLP:  CMP     BYTE PTR [DI],'0'
        JNZ     DNECVT          ;
        STOSB                   ;
        LOOP    SUPLP           ;
DNECVT: RET
CONVRT  ENDP
;
SCAN    PROC    NEAR            ;
        MOV     DI,OFFSET PNTR  ;
        XOR     AX,AX           ;Clear out the
        MOV     CX,120          ; pointer table
        REP     STOSW           ;
        MOV     BX,OFFSET PNTR  ;BX is base of pointer
                                ; list
        MOV     DI,OFFSET SRCE  ;DI is base of file stack
        XOR     CX,CX           ;
        MOV     DX,OFFSET DFCB  ;
        MOV     AH,11H          ;Get the first file
        INT     21H             ;
        JMP     SHORT INNER1    ;
LOOP:   MOV     DX,OFFSET DFCB  ;
        MOV     AH,12H          ;Get the next file
        INT     21H             ;
INNER1: OR      AL,AL           ;AL is FF when we're done
        JNZ     DONE            ;
        CALL    SAVE            ;Put file on stack
        INC     CX              ; Bump file counter
        JMP     LOOP            ;
DONE:   RET
SCAN    ENDP
;
SAVE    PROC    NEAR
        PUSH    CX              ;Save file entry pointer
        MOV     WORD PTR [BX],DI ;
        ADD     BX,2            ;
        MOV     SI,81H          ;SI is pointing to
                                ; beginning
        MOV     CX,8            ; of the file name
SVLP:   MOV     AL,BYTE PTR [SI] ;Get a character
        CMP     AL,' '          ; if its a space, we're
        JZ      NMDNE           ; at the end of the name
       MOV     BYTE PTR [DI],AL ;Put the character in
        INC     SI              ;  the stack and bump
        INC     DI              ;  the pointers
        LOOP    SVLP            ;
NMDNE:  MOV     SI,89H          ;Point to file type
        CMP     BYTE PTR [SI],' '
        JZ      ALLDNE          ;If no file type branch
        MOV     BYTE PTR [DI],'.'
        INC     DI              ;If there is a file type
        MOV     CX,3            ; put a . in the stack
                                ; entry
        REP     MOVSB           ; and move the file type
ALLDNE: MOV     BYTE PTR [DI],0 ;A 00 marks the end of
                                ; the string
        INC     DI              ;
        MOV     SI,9DH          ;Point to the file size
        MOV     CX,4            ; move all 30 bits used
                                ; for size
        REP     MOVSB           ;
        POP     CX              ;
        RET
SAVE    ENDP
;
SORT    PROC    NEAR
        MOV     STKCNT,CX       ;Store number of entries
        DEC     CX              ;
        MOV     SI,OFFSET PNTR  ;SI is first file entry
                                ; ptr
OUTER:  MOV     DI,SI           ;
        ADD     DI,2            ;DI is 'next' pointer
        MOV     DX,CX           ;
INNER:  CALL    COMPAR          ;compare the entries
        JBE     LEAVE           ;
        MOV     AX,WORD PTR [SI] ;
        XCHG    AX,WORD PTR [DI] ;Swap the pointers if
        MOV     WORD PTR [SI],AX ; they need to be
LEAVE:  ADD     DI,2            ;
        DEC     DL              ;Check this entry
        JNZ     INNER           ; against the rest
        ADD     SI,2            ;Move to the next entry
        LOOP    OUTER           ;  and check that one
        RET
SORT    ENDP
;
```

```
COMPAR  PROC    NEAR
        PUSH    SI              ;
        PUSH    DI              ;Save the pointers
        PUSH    CX              ;
        MOV     CX,12           ;Strings are 12 bytes
                                ; long
        MOV     SI,WORD PTR [SI] ;Start of first entry
        MOV     DI,WORD PTR [DI] ;Start of another entry
        REP     CMPSB           ;Compare them
        POP     CX              ;
        POP     DI              ;Restore the pointers
       POP     SI              ;
        RET
COMPAR  ENDP
;
PRINT   PROC    NEAR            ;
        CMP     CLRFG,0         ;Check to see if were
        JZ      NOCLR           ; suppose to clear the
        MOV     DX,OFFSET CLRST ; screen.  If so, do it
        MOV     AH,9            ;
        INT     21H             ;
NOCLR:  CMP     PNTRFG,0        ;Check to see if were
        JZ      PNT1            ; to dump to the printer
        MOV     OUTDV,5         ; If so, set the output
                                ; device
PNT1:   CMP     VERSFG,0        ;Check to see if the
        JZ      NOVERS          ; version was requested
        MOV     DX,OFFSET VERSN ;Point to the version
        CALL    DOPRT           ;  Send it out
NOVERS: MOV     AX,STKCNT       ;
        MOV     DH,4            ;Divide the number of
        DIV     DH              ; entries by 4
                                ; (the number of columns)
        OR      AH,AH           ;
        JZ      SETCNT          ;
        INC     AL              ;
        CBW                     ;
SETCNT: PUSH    AX              ;Save the number of
        CALL    DOCRLF          ; entries per col.
        MOV     DX,OFFSET TITLX ;
        CALL    DOPRT           ;Send the title line out
        CALL    DOCRLF          ;
        POP     CX              ;
        MOV     BP,CX           ;
        SHL     BP,1            ;

        MOV     SI,OFFSET PNTR  ;
OTLP:   MOV     DL,4            ;DI is number of columns
        XOR     BX,BX           ;
INLP:   CALL    PRTENT          ;Send entry out
        ADD     BX,BP           ;move to next column
        DEC     DL              ;if all 4 aren't done
        JNZ     INLP            ;do the next one
        CALL    DOCRLF          ;New line
        CMP     OUTDV,2         ;
        JNZ     NEXTLP          ;
        INC     LINCT           ;Add 1 to the line
                                ; counter
        CMP     LINCT,20        ; Is the screen full?
        JNZ     NEXTLP          ;  if not then continue
        MOV     LINCT,0         ;Reset the line count
        PUSH    DX              ;
        PUSH    AX              ;
        MOV     DX,OFFSET MMSG  ;Ask for more
        CALL    DOPRT           ;
        MOV     AH,1            ;Continue on any key
                                ; input
        INT     21H             ;
        CALL    DOCRLF          ;
        POP     AX              ;
        POP     DX              ;
NEXTLP: ADD     SI,2            ;Point to next pointer
        LOOP    OTLP            ;
        CMP     OUTDV,2         ; if out to CON, we're
                                ; done
        JZ      PRTOUT          ;
        CMP     FFFLG,0         ;Check for form feed
                                ; flag
        JZ      PRTOUT
        MOV     DX,OFFSET RESTR ;Form feed for the
                                ; printer
        CALL    DOPRT           ;
PRTOUT: RET                     ;
PRINT   ENDP
;
DOPRT   PROC    NEAR
        PUSH    DX              ;Print a string pointed
        PUSH    SI              ;to by DX.  Null byte
                                ; terminates
        MOV     SI,DX           ;
        MOV     AH,OUTDV
DPLP:   MOV     DL,BYTE PTR [SI]
        OR      DL,DL
        JZ      PRTEND
        INT     21H
        INC     SI
        JMP     DPLP
PRTEND: POP     SI
        POP     DX
        RET
DOPRT   ENDP
;
PRTENT  PROC    NEAR
        PUSH    CX              ;Prints one file entry
        PUSH    DX              ;from the stack
        MOV     CX,12           ;
        MOV     DI,WORD PTR [SI+BX]
        OR      DI,DI           ;DI points to the entry
        JZ      BLNK1           ; if it is 0 then output
                                ;  blanks
        MOV     AH,OUTDV        ;
PELP:   MOV     DL,BYTE PTR [DI]
        OR      DL,DL           ;Output the name and
        JZ      BLNK2           ; pad with blanks to
        INT     21H             ; fill all 18 spaces
        INC     DI              ;
        LOOP    PELP            ;
BACK:   INC     DI              ;Get the file size
        MOV     AX,WORD PTR [DI]
        MOV     DX,WORD PTR [DI+2]
        PUSH    SI              ;
       MOV     DI,OFFSET DBUF  ;
        CALL    CONVRT          ;Change it to ASCII
        POP     SI              ;
        MOV     DX,OFFSET DBUF  ;Output the size
        CALL    DOPRT           ;
GONE:   POP     DX              ;
        PUSH    DX              ;Restore the entry value
        DEC     DL              ;
        JZ      PUNT            ;
        MOV     DX,OFFSET DBLK  ;Put 2 blanks out if its
        CALL    DOPRT           ; not the last column
PUNT:   POP     DX              ;
        POP     CX              ;
        RET                     ;
BLNK1:  MOV     CX,18           ;Output all blanks for
        CALL    CLER            ; no entry
        JMP     GONE            ;
BLNK2:  CALL    CLER
        JMP     BACK
PRTENT  ENDP
;
DOCRLF  PROC    NEAR
        PUSH    DX              ;Output a CF,LF combo
        MOV     DX,OFFSET CRLF  ;
        CALL    DOPRT
        POP     DX
        RET
DOCRLF  ENDP
;
CLER    PROC    NEAR
        MOV     DL,' '
DLFIL:  MOV     AH,OUTDV
CLRLP:  INT     21H
        LOOP    CLRLP
        RET
CLER    ENDP
;
PNTR    DW      0

SRCE    EQU     PNTR+240
;
CODE    ENDS
;
        END     BEGIN
```

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~~!** NEW LOW PRICE

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

40 or 80 Tracks — Single or Dual Head — New 3½'' Drivette™

Our Disk Drives are Capable of Single **and** Dual Density Operation

The **NEWEST** Technology Capable of Operating on Most Popular Computers

Drive a Hard Bargain!!™ For your TRS/80, Color Computer, IBM, Apple, Franklin

5 M.B.-20 M.B. Complete Systems.....from $999.95

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

SAVE!! PLEASE CALL FOR OUR MOST CURRENT PRICE REDUCTIONS.

TOLL FREE ORDERING
1-800-343-8841

GENERAL AND TECHNICAL
1-617-872-9090

CALL TOLL FREE FOR NEW PRICES

- Disk Drives (0123) TRS/80-IBM-Apple - TI Franklin-Max/80-LNW .....◄
- Model I/III/IV Upgrade (Disk Drives - Memory).....◄
- **Printers** — Daisywheel/Dot Matrix .....◄
- Percom Double Density Controller (Model I) .....◄
- Color Computer Printer Interfaces.....◄
- Disk Drive Operating Systems .....◄
- **Repair Services** Now Offered — FAST Turn-a-Round.....◄
- **Apple/Franklin Compatible Add-On Drives with Case & Cable**.....◄
- Diskettes in Library Cases .....◄
- **DISK DRIVE CASES AND POWER SUPPLIES** ..... starting at $59.95
- Printer **Buffers** 8K to 512K ..... starting at $143.95
- Holmes Model I/III Speed-up Mod ..... starting at $90.00
- Cables — Printer/Disk Drive ..... starting at $23.00

Warranty on Disk Drives — 6 Months to 1 Year

SOFTWARE SUPPORT, INC.

One Edgell Road, Framingham, MA 01701 (617) 872-9090

Hours: Mon. thru Fri. 9:30 am to 5:30 (E.S.T.) Sat. 10 am to 4:30 pm

DEALER INQUIRIES INVITED.

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

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

Service! Service!

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

TRS/80 Registered Trademark Tandy Corp. IBM-PC Registered IBM Corp. Apple Registered Trademark Apple Computer Corp.
Franklin Registered Trademark Franklin Corp. Max/80 Registered Trademark Lobo Int.

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

Patch Page

Pat Swayne
HUG Software Engineer

Note: When you patch a program, be sure that you have at least one backup copy in case something goes wrong.

UDUMP Patch

The Print Screen command in the UDUMP program on HUG disk 885- 8004 will not work properly on H8 computers. The problem can be fixed with UDUMP itself by entering UDUMP as the file to edit and using the F3 key to go to sector 32 of the file. Then enter the HEX modification mode (F4) and move the cursor to the byte at address 42B0. Enter F3 to enter the assembly mode, and enter the following code.

```
DI              DCR     C
MVI     C,50    JNZ     42B3
IN      ED      MVI     A,0A
ANI     01      MOV     M,A
JZ      42B3    INX     H
IN      E8      JMP     42B1
MOV     M,A     NOP
INX     H       NOP
CPI     0D      EI
JZ      42D0
```

After you enter this code, type Control-D to exit the assembly mode, press the RED key (or F7) to write the patch to the disk, and press the White key (or F8) twice to exit UDUMP. The next time you use UDUMP, the Print Screen command will work correctly.

3 or 4 MHz DUP Patch

If you made the 4 MHz modification to your H/Z-89 described in the November 1982 issue of REMark, or the 3 MHz modification in the March 1983 issue, you may want to perform the following patch to make the DUP program work at 3 or 4 MHz. Use the DDT program to make the patch as follows:

```
DDT DUP.COM
NEXT  PC
1200 0100
-SE70
0E70  30 60  (48)
0E71  xx .   (type a period)
-SE7C
0E7C  50 F0  (A0)
0E7D  xx .   (xx indicates an unimportant value)
-^C          (Control-C)
A>SAVE 17 DUP.COM
```

The numbers 60 and F0 are what you enter for 4 MHz, and 48 and A0 are for 3 MHz. This patch is for the CP/M 2.2.03 DUP program. If you have 2.2.04, the patch addresses are F02 and F0E, and you should SAVE 18 pages.

CAMERA Patch

The CAMERA program (graphics editor) on HUG disk 885-1124 does not work on H8 computers that use the H8-5 card for the console. If you use that configuration, you can fix the program by altering the source and re-compiling as follows. Edit the file CONSOL.TPI and change the line that reads

```
MEM[%63207] := %333   MEM[%63210] := %350;    ! IN   350Q   !
```

so that it reads

```
MEM[%63207] := %333   MEM[%63210] := %372;    ! IN   372Q   !
```

and then re-compile the program. You will need the Tiny Pascal compiler, available from HUG as P/N 885-1086[-37]. If you would like to fix the program without re-compiling it, use the Patch program supplied with HDOS to patch address 105341 from 350 to 372, in the program CAMERA.ABS. You will not need the compiler for this patch.

HSY.DVD Thin Drive Patch

The HSY.DVD device driver included on the HUG Hard Sector Support Package (885-1121) will not work properly with some half height drives such as the Shugart SA-455. Those drives require a small time delay when the controller switches from one side to another while the disk is being formatted. To patch the driver, load the file MFINIT.ACM into your editor and locate the label MFILP3. The code should look like this:

```
MFILP3  CALL    D.SDT           ;SEEK TRACK, SKIP SECTOR,
                                ;DIS...
        LDA     D.DVCTL         ;GET DEVICE CONTROL BITS
        ORI     DF.WG           ;TURN ON WRITE
        OUT     DP.DC
```

Edit the code so that it looks like this:

```
MFILP3  MVI     A,255
        CALL    D.UDLY          ;WAIT A WHILE
        CALL    D.SDT           ;SEEK TRACK, SKIP SECTOR,
                                ; DIS...
        MVI     A,255
        CALL    D.UDLY          ;WAIT AGAIN
       LDA     D.DVCTL         ;GET DEVICE CONTROL BITS
        ORI     DF.WG           ;TURN ON WRITE
        OUT     DP.DC
```

Re-assemble the driver following the instructions included on the Hard Sector Support disks.

The SDUP program on the Package must also be patched. To make the patch, load the file SDUP.ASM into your editor and find the label INIT171:

```
INIT171 XRA     A
        OUT     UPST            SET FILL CHARACTER
```

Modify the code as shown below:

```
INIT171 MVI     A,255
        CALL    UDLY
        XRA     A
        OUT     UPST            SET FILL CHARACTER
```

Now, find the label DLY, and change the MVI A,1 a few lines above it to MVI A,2 as shown below:

```
        MVI     A,2             WAIT 4 MS
*       JMP     DLY

*       DLY - DELAY (A * 2MSEC)

DLY     LXI     H,TICCNT
        ADD     M
DLY0    CMP     M
        JNZ     DLY0
        RET
```

Re-assemble SDUP.ASM to get a new SDUP.ABS.

A Review of

Query!2

and Report Writer

Paul W. Franchina
4102 Greenfern Drive
Orlando, FL 32810

Overview

Occasionally a product comes along that offers the consumer a true value for his hard earned dollars. I consider Query!2, which is the subject of this review, to be just such a product. Query!2 by Hoyle & Hoyle Software is a group of 8 programs that perform the functions necessary for database management. The programs run under Z-DOS (MS-DOS) operating system on H/Z-100 computers with one disk and 128K of RAM. The documentation is clear, concise, and above all easy to understand. The programs are command driven. The commands are accessed with a single character, then the program fills in the rest of the word(s). Commands are the same for common functions across all the individual programs, and if you just remember that H stands for help, you'll have the command summary list at your finger tips whenever necessary. In this day of megabuck, super-software that requires an engineering degree to understand and use, it's refreshing to come across an offering that is easy to use and understand, and can allow its user to be producing valuable results within minutes from taking it out of the package. The people at Hoyle & Hoyle have a lot to be proud of. They have a good product that performs a useful service to its owner. They support it 100% (that being from my experience), and they're responsive to the market and their customers. And no, I'm not being paid to say this, it's just the way I feel. Let's find out more about what has me so excited.

Program Features

Query!2 provides all the features necessary for true database management. The program names leave little guesswork as to their function and each one returns you to DOS upon exiting. The first one to use is CREATE which does just that, creates a database file with the name YOU specify. You can send output to any drive on your system, using the appropriate identifier, and use an extension of your choosing or the default '.DTB'. With CREATE, YOU can define the number of fields in this new file, the amount of characters in each field, and the title or description assigned to each. Query!2 allows up to 255 fields in a record with up to 255 characters in a field. Now before all you fast mathematicians go running off wild over having each record consuming 64K of your precious memory, there is an automatic upper limit of 4096 characters imposed by the program. This allows us to have a possible 16 fields with the maximum 255 characters in each or the maximum 255 fields with an average of 16 characters. There are almost limitless combinations available that won't violate the upper bound. CREATE is the only program in the group where some thought must be exercised beforehand. This is not a fault of the program, but rather a necessity of proper file definition. Once your file format has been defined, you must either live within that definition or start over from scratch. Query!2 will not allow a file definition to be modified once it has been stored on disk. It is recommended that you start out with small files and test your information with real life inquiries until you have your needs well defined. If you should load up a file with hundreds of records, then find you're missing some critical field, you'll have lots of typing to do over for Query!2 doesn't support loading a new file with information from an existing one. You are, however, given an opportunity, before making the description permanent, to correct for spelling errors or even change field lengths, but you cannot add or delete fields without again starting over. You are also given the opportunity at this point only to get a hardcopy of the file description for future reference. This is a useful feature and I strongly recommend you obtain this printed copy. It can be a good reference aid and will be a necessary tool for working with the report generator that you can purchase as a utility for Query!2.

From CREATE we move on to ADD. This is where individual records are filled with the information you want to store in the database file. ADD allows editing any and all fields before filing the record in the database, and the ESCape key can be used at any point in the process of writing a field or record without any harmful effects. After loading a database information, we're ready to let Query!2 flex its muscles.

So far we've seen how files are created and filled with information. Actually the files are loaded with data. Data can be converted to information when it is put to some use. Just by looking at the records in a database, we can begin to extract information from it. The next program to use then is VIEW and, as you may have already guessed, it lets you page through your data file looking at the information contained there. The program always displays the first record in a file and from there you can go forward or backward as you wish or go directly to any particular record by (L)oading that number. Once you have a record displayed on the CRT you can edit the fields, delete the entire record, or send the information to the printer in any one of three available formats. The deletion feature bears special mention and is of great value. Query!2 considers each record as either active or inactive in normal operation. Deleting a record merely moves it to inactive status. The record remains in the file and can be viewed, edited, even reactivated with no loss of material. This is positive prevention against accidental loss of information. At any time thereafter, these inactive records can be removed from the file using the PURGE program. If you check your disk directory after a purge, you'll notice that the file size hasn't changed due to the deletion of records. My guess is that Query!2 marks the records to a non-display/locate status and moves them to the end of the file for over-writing as new records are added. This would indicate that lost records might still be retrieved if new records have not been added since the purge. I'll leave that one up to those assembly language

guys that need a challenge. While using VIEW to find information is certainly possible, it's far from efficient and definitely not putting the computer or this fine program to good use. How about getting this informational house in order?

A trip to the directory and, sure enough, there's one there called SORT! This little beauty can put things in order on any field we choose and even warns us that it would be wise to make a back-up of our file at this point, just for safety's sake. To test the program, I created a file called "LIBRARY.DTB" and loaded it with 120 records of various book information from my collection. The file was then sorted for alphabetical order on the first field which contained the book title. The internal sort took approximately 40 seconds to complete with another 40 seconds devoted to rewriting the disk file to its new order. This makes finding information using VIEW much easier, but again there is a much better way to accomplish the task of locating the records we desire.

The ingredient we're looking for is called SEARCH and this is where the power of Query!2 really shines. Would you like to search on a single field? Maybe search on multiple fields with ANDs or ORs for conditions? What about ANDs connected by ORs or possibly ORs connected by ANDs? Would you like just the inactive records? Query!2 gives you all these abilities and more. Comparisons can be made for equalities (=), less than greater than situations (<,>), any substring, or even if the field begins with a particular reference string. Query!2 supports up to 40 search conditions! This wasn't verified in my use of the program. I couldn't think of enough conditions to try on my data base so I'll take the Hoyle's word on it. Once the searching is complete, the program will tell you whether or not it found any records matching your requirements and if so, how many were found. If it has indeed found at least one record, you will be prompted for the type of output you desire. A quick tap of the H key and all your alternatives are there in front of you. You have the option of sending the records to the CRT or out to the printer in one of the three available formats. Also you may create a sub-file to disk if you desire. This is one of the features that really endeared me to this program. I have a lot of science fiction books in my collection so I took advantage of this option to create a sub-file "SCIFIBKS.DTB". All the records are duplicated there in the original file format defined by CREATE. The records remain intact in the original file, but searching through this smaller file is much faster though limited to just the sci-fi titles. Another output option available is to create what Query!2 calls a standard file. This creates a sequential disk file that can be accessed by BASIC programs. The full range of VIEW options are also accessible as an output option except that you exit back to the Output prompt from SEARCH rather than the operating system, as is the case when using the VIEW program itself.

I've talked a number of times about the printing options available to you with Query!2. The PRINTER program allows us to get hardcopy information from our database. The options are the same as those usable from VIEW or the view option under SEARCH. The difference here is that the whole file may be printed at one time whereas the others only allow individual records or selected groupings to be sent to the printer. You can print one record to a page (provided the record is 60 lines or less), or select continuous feed where multiple records will be fitted to each page when possible. The format choices send the information out with or without the field names included or in the special mail-label mode. Query!2 uses the first seven fields in a record to produce mailing labels. An option is included for multiple labels per line with a self-checking feature so that proper label spacing will be maintained without one label overwriting another. You may also save the particular print definition to disk to be used over and over with a single call to this routine. It is up to you to make sure the information is properly entered in the fields of each record, but the directions to do so are very clear. These records may have any additional data in any other fields you choose so the applications are almost limitless.

The last program option in the Query!2 repertoire is called RECOVER. This is a small utility program that will go thru the chosen file and reactivate all the deleted records in one pass. This function could be accomplished on an individual basis as described in the section on VIEW. Doing it that way would be a very time consuming operation so this program is provided for those times when you might want everything back in a hurry!

Impressions

There are some limitations you should be aware of. The proper term for this type of program is file handler or manager. It is not of the class known as relational database managers. It could be used to set-up an inventory system for a small business, but you would have to use individual database files for records such as customers, suppliers, parts, and so on. This would be functional, but not necessarily as efficient as the relational managers that let you move between information that have defined relationships, e.g. parts, their suppliers, and the customers that have purchased these parts. The fact remains that given your investment in Query!2 in comparison to a relational DBM you could be dollars ahead for a long time. This time could be that necessary for a small business to get on its financial feet and have the purchasing power necessary to acquire the relational software. Along this same line, Query!2 does not allow for transferring information between files and only one file may be open at any time. Because each program returns you to DOS when you exit, moving back and forth between the programs (functions) requires that you specify which file it is that you're working with each time. I talked to Janet Hoyle over the phone a number of times during the course of this evaluation and not only was the service excellent, but according to her, this problem is being addressed in an upcoming revision to the program. This update should implement a master menu whereby a file will remain active until changed by the operator.

Report Writer

Producing reports from Query!2 database files leaves me with the complete opposite impression. Report-Writer is the name for Query's report generator. This program, which is purchased separately, allows you to extract, manipulate, and format data from your files. The hows and whys to do this are rather involved and are solely your responsibility to figure out ahead of time. To do this you must create an ASCII text file using an editor or word processor. EDLIN, which comes with your Z-DOS operating system disk is suitable for this purpose, though most word processors will do the job with less effort. The file will contain up to five sections. Each section consists of single letter commands either with or without an argument. Only one command is allowed per line and only certain commands can be used in specific sections while others are permissible in multiple sections. Report-Writer will process this text file as a batch file taking each command and performing the necessary operations on the file you define. This will get the job done producing the reports you need, but requires much patience and debugging. No matter how many times I plan something out on paper before I get started, there is always something that requires further attention before all is running smooth. Going back and forth between text editor and Report-Writer to iron out these bugs chews up paper and my nerves in large quantities. Again the staff at Hoyle & Hoyle informs me that they're on the way to our rescue with a replacement for the Report-Writer. This program will be called Query!-Calc and is to be an interactive report generator. I was told it is due for release this Spring so it may be available by the time you read this. Ms. Hoyle said that verified owners of Report-Writer will be offered this program for a nominal fee with the return of their original R-W disk. My Report-Writer disk

even came with a certificate for just such an exchange. I intend to upgrade as soon as possible. I believe the folks at Hoyle & Hoyle have the talents to produce a first-rate report generator. I just don't believe it happened this time around.

Conclusion

If you are like me, you will love Query!2. I have a need to store and access information in many categories and would like to do so without going broke. I admit that computers aren't worth much without the software to make them perform, but I still can't understand why some software seems to cost as much as the machines they run on. Query!2 does the job it was designed to do, and does that job well. Books, records, tapes, photos, recipes, parts, names, addresses, dates, the uses for this program are limitless. Anything that can be cataloged is a candidate for Query!2. I have other programs that belong to the same category as Query!2. Some are much fancier, with eloquent screen formats and various file formatting and handling methods. But when it comes to a price/performance leader, it has to be Query!2. I compare this program to my old British sports car. It's not as plush or flashy as my friend's 280 Turbo ZX, but it sure is a blast to drive, easy to maintain, and I've got a lot more spending money in my wallet than he'll have in his for quite a time yet to come.

Ordering Information

Hoyle & Hoyle Software
716 South Elam Avenue
Greensboro, NC 27403

Query!2 - $29.95
Report Writer - $19.95
128K RAM, one or more disks.
Test editor or Word Processor required for Report Writer.

ZLYNK/II Reviewed

Haywood N. Nichols, Jr.
10900 S.W. 104th St., Apt. #414
Miami, FL 33176

Introduction

ZLYNK/II is a modem communications package written by Dale Lamm and sold by Software Wizardry. This program is available for both the H/Z-100 and the H8/H19/H89 computers running the CP/M operating system. The version of the program reviewed here is 83.09.18(a), set up to run on my H8/H19, 64K, 5 1/4 inch four drive (2 hard & 2 soft) system using a Hayes 300 baud smartmodem.

I know what you're probably saying to yourself right now, ARGH! Just what I don't need, 'ANOTHER MODEM PROGRAM'. Having about five or six modem programs myself that was my first reaction, but with Dale Lamm's reputation I decided to take the plunge just one more time in hopes of finding the one modem package that might serve all my needs.

Getting Started

The package I received contained one professionally styled loose leaf binder, 126 pages of documentation, 6 pages of errata sheets, a smaller sealed envelope containing one 5 1/4 inch (48 TPI soft sectored double sided floppy) and a user registration card (which should be filled out and returned so you'll be informed of any updates to this package). Also included is a sheet of pink paper titled 'Your First Session With ZLYNK/II'. This is a nice touch for people like me. I'm one of those who insists on trying something out first before reading the instructions completely. It describes how to get the program running in the 'DT' Dumb Terminal Mode in almost no time. Anything you type will be sent directly to the modem and anything received will be displayed on the console. Operation past this point will require some study of the manual.

Operation

When you execute ZLYNK/II for the first time you'll be asked to enter your name, this will be displayed on an entry banner when you run the program from now on. Then you're asked to enter the time and date or a <CR> carriage return to bypass both. The screen will then fill up with a menu of commands (more then you thought possible in a modem program). A command is executed by either typing the complete word 'DIRECTORY' or by typing the first two letters of the command 'DI'. Multiple commands may also be entered from the menu page. They will be executed in the order that you typed them in.

To the first time user this large menu is a little overwhelming, but don't worry, there's an extensive 'HELP' feature that allows you to call up information on any given command. After several days I became quite comfortable with the system and deleted the 'HELP' file from my disk (its a very large file and takes up considerable space).

You have the option of bypassing the menu section of ZLYNK/II by entering one or as many commands as you like on the CP/M command line (limit of 80 characters of course) when initially invoking ZLYNK/II.

For example: A> ZLYNK command-1 command-2 etc., etc.....

After the time and date prompt has been satisfied, the commands will be executed one after the other without requesting input from the user.

You will normally be operating ZLYNK/II in the 'IT' Intelligent Terminal Mode. This is where the program starts to really show its stuff. When the 'IT' command is given from the menu section of the program, all the special function keys are activated and the 25th line will display function key labels.

Saving Text

You have three choices for capturing data from a remote system. This is done by opening a file in the following manner:

[**C**]**ontinuous** - all data that you see on your screen in the 'IT' Intelligent Terminal mode is written to disk as it comes from your modem a sector (128 characters or bytes) at a time. The amount of free disk space remaining is shown on the 25th line.

[**B**]**uffered** - data is stored in a buffer area (about 16K). The percentage of free buffer space is displayed on the 25th line in a box on the right side of the screen. Your terminal bell starts beeping at 2%. There is a safety feature that sends a '↑S' or stall to the remote when there is about .5% buffer space remaining. This will give you a chance to hit the '[f1]' key. If nothing is done, the buffer will fill up and the box will indicate 100% again as the new data overwrites the old. To save the data to disk before it is overwritten you have two options. You can hit an '[f1]' special function key and the data will be written to disk clearing the buffer back to 100%, or you can issue a 'CF' Close File command from the menu page.

Incoming data is always being stored in the buffer even if a file hasn't been opened. If you see something worth saving all you have to do is return to the main menu, hit a '↑S', open a file, close the file and you're set.

[**S**]**creen** - when you see something on the screen you want to save you hit the '[f8]' special function key.

Special Function Keys

[**f1**] Flushes the contents of the capture buffer by writing to disk.
[**f2**] Hitting this key once will cause a '↑S' to be sent causing incoming data to be halted, striking the key a second time will send a '↑Q' resuming data flow from the remote.
[**f3**] Causes the special function key labels on the 25th line to disappear. Striking the key a second time will bring the labels back.
[**f4**] This will immediately close any file you have open that is capturing incoming data.
[**f5**] Any data written to the capture file will be discarded.
[**f6**] Gives a continuous hard copy output of your modem session.
[**f7**] Is a dual purpose key, when hit once it returns you to the main menu, when hit twice, it will disconnect your modem.
[**f8**] This special function key labeled 'DUMP' will save screen information to either disk or your line printer.

Special Features

Now we get to my favorite part, those special features I haven't seen anywhere else.

ZLINGO Command Language

This language allows ZLYNK/II to operate with you at or away from the computer using BASIC/ENGLISH type commands. ZLINGO will

allow your computer to perform predetermined tasks at user request or from data received while on line. It supports if/then/else type conditional branching, eight flags for storage of numerical data, string manipulation, goto and gosub commands that allow you to jump around within your current file or chain to another file, a trace feature like 'MBASIC' for debugging, and most of the commands listed on the menu page are executable from a ZLINGO file.

Using ZLINGO I wrote a HUG Bulletin Board retrieval system that does the the following:

1. Activates itself at a preset time.
2. Call and sign on Compuserve from either the 'OK' prompt or the opening list of options page (depending on how you have your default parameters set).
3. Sign on the HUG Bulletin Board.
4. Retrieve any messages sent to you.
5. Send a canned response to those messages.
6. Check to see if anyone is on the <CO>nference section of the BB. If so, go to 'CO' and chat with them.
7. Do a <R>ead <F>orward of all the new messages since you were last on.
8. Scan the entire HUG database to see what new programs have been uploaded in the last 3 days.
9. Log off the HUG Bulletin Board and Compuserve.
10. Hang up the phone.
11. Save the entire session on disk for you to read at a later date.

All of the above is done while I'm away from home with the computer doing all the work by itself. I've placed this set of command files in the public domain. They are available for downloading from the HUG Bulletin Board on Compuserve. The files are located in section 'XA2' of the database. If you don't access the HUG BB then you can send me a SASE and a 5 1/4 inch (hard or soft) disk and I'll get a copy right back to you.

Text Editor

A built in screen oriented text editor that allows you to edit data after a download or before an upload. There is also a subeditor that allows you to create ZLINGO command files with automatic line numbering.

File Transfers

There seems to be as many different protocols for downloading and uploading files as there are versions of BASIC. For the past several years I've had to own several different modem programs all supporting different protocols to access my favorite bulletin boards. With ZLYNK/II, I finally have a program that will handle all of them. ZLYNK/II's protocols are in the form of 'overlays' that are called in as needed. Five protocols are currently supported.

1. XMODEM

Ward Christensen's standard for the hobbiest community, used on just about all the RBBS and CBBS CP/M boards around the country. Two xmodem protocols are supported: 'XM' the older method using checksum error detection, and 'XC' using the newer and more reliable 'crc' (cyclic redundancy check) method. Both methods support ASCII and binary uploading and downloading of files.

2. COMPUSERVE

This protocol supports the Compuserve Information Service and the various sigs such as the HUG and CP/M Bulletin Boards. Both ASCII and binary file transfers with error checking are supported.

3. VM-370 and TSO

These protocols are one-way transfers to send data to an IBM system.

4. PROMPTED and THROTTLED SEND

Kind of a catch-all protocol that will allow you to operate in almost any environment (sort of on the fly). THE 'PS' Prompted Send is one of my favorites. With it you have the ability to send a batch of replies or messages all at once on your favorite Bulletin Board. On the HUG BB of Compuserve, I usually sign on and retrieve any messages sent to me. I log off to read them at my leisure and compose replies with my favorite editor (the one on the BB is crude to say the least). Then sign back on the bulletin board to leave several replies and other messages all at once. You'd be surprised how much connect time (remember, time = money) you can save using this method. This will work in almost any bulletin board environment. Neat Huh!!?

5. WESTERN UNION EASYLINK

A new service recently installed by Western Union that allows just about anyone with a terminal to access an 'EasyLink' network to send Telex, TWX, and mail to other subscribers.

Hayes Smartmodem

ZLYNK/II supports those neat features you've grown to love using your Hayes Smartmodem: touch-rotary dial, auto-dial, repeat dial, auto-answer, etc., etc.

CP/M Housekeeping Chores

Alot of the functions normally associated with the CP/M operating system can be handled within your ZLYNK/II modem program, such as:

1. Disk directories.

The names of files on any given disk can be viewed. The standard CP/M wildcards are also supported ('*' and '?').

2. Selecting a new default drive.

This command redefines your default drive to handle any operation where an explicit drive name is not given.

3. Run a 'COM' file.

When you invoke this command you will exit from ZLYNK/II and any valid CP/M command or machine code program will be executed.

4. Change your current user area.

5. Resetting the disk system.

6. Amount of disk free space.

7. Erase a file.

8. Rename a file.

9. Copy a file.

10. Merge two files.

11. Type a file.

Log Files

ZLYNK/II supports two different types of logfiles:

1. Connect Time Logfile

This file is a record of the remote systems you access. This logfile includes:

a. date of access
b. time logged on
c. time logged off

d. total time
e. system accessed

This can be quite useful when you access remote systems that charge you a connect rate, such as The Source and Compuserve. Believe me, both these services can make mistakes and it pays to keep track of your connect time.

2. Comment Logfile

This Logfile can be used to store various strings and numerical data to be used in 'ZLINGO' files.

Remote Access Switch

This allows a caller to access your system and command ZLYNK/II the same as you would (with certain exceptions). You can make certain commands (of your choosing) unavailable to the other system you are communicating with.

Save Configuration

ZLYNK/II can be configured from the main menu and you can also perform a complete configuration using the 'SC' command. This command will save an exact memory image of ZLYNK/II along with all the changes to disk. Some of those changes might include baud rate, stop bits, word length, parity, full or half duplex, remote echo, and xon/xoff's.

ZLYNK/II For The H/Z-100

The H/Z-100 version of this program is virtually identical to the H8/H89 reviewed with the following exceptions:

1. Minor modification to the RS232 cable if you're using a Hayes Smartmodem or compatible intelligent modem.

2. Optional use of color in menu and other displays.

Documentation

The manual is broken into two sections, 'Overview' and 'Reference Manual'. There is a table of contents for both sections. The 'Overview' section of the manual is complete and fairly easy to read. The Reference Manual caused me some confusion at times. Only the different types of commands are broken down and given page locations within the manual, the actual commands themselves are not listed in the table of contents or in an index at the end of the section. You'll have to do a little hunting to find what you need. Both an expanded Table of Contents and an Index at the end of both sections of the manual might be a good idea for a future update.

Bugs

Since receiving my copy of ZLYNK/II two overlays have been updated:

1. A revised version of the Compuserve's 'A' protocol that corrected one bug and improved the performance of downloading/uploading files.

2. The 'SC' (save configuration) overlay had a bug that could cause a newly 'saved' ZLYNK/II to crash sooner or later.

Both of these revised overlays are available for downloading from the HUG Bulletin Board on Compuserve. They are located in the 'XA2' section of the database. For those of you who don't access Compuserve, you can contact Software Wizardry.

The only other known bug is currently being worked on by the author. It can cause the system to hang up when the buffer space gets down to 2% free space remaining (saving incoming data in the [B]uffered mode). Actually you should never let it get down to 2% anyway. The temporary fix is simple, close or flush (hit an '[f1]' special function key) before the buffer gets to 2%.

Product Support

The support the end user gets with this software package is excellent. Both the author and Software Wizardry are extremely helpful in getting inexperienced users up and running. They have indicated their continued support for this package in the form of protocol updates and enhancements. An update to ZLYNK/II is planned in the near future that will include improved documentation, new commands for the ZLINGO language, support for another intelligent modem (possibly the U.S. Robotics S100 modem), and a fix for the above mentioned bug. Registered owners will be able to get this update at a substantial discount ($20 or less).

Anyone who buys this product should gain access to the HUG Bulletin Board on Compuserve. In addition to the author and distributor being on the board to answer any questions, Dale and other users have uploaded loads of nifty utilities and ZLINGO command files that are there just waiting to be downloaded. These freebies are located on 'XA2' of the HUG database.

Ordering Information

Software Wizardry, Inc.
122 Yankee Drive
St. Charles, Mo. 63301

Phone: 1-314-946-1968

Price: $59.95 + shipping

Conclusion

This is a fine piece of software and well worth the $59.95 price tag. The only modem package I've seen on the market that even comes close to the capabilities of ZLYNK/II had about half the features and a cost of $250 (TERMII running on an *ech!* Apple). A beginner might be a little overwhelmed by both this review and ZLYNK/II itself, but the time you take learning how to operate this program is well worth the effort. The capabilities of this package should satisfy just about any modem communications application.

NEW FROM FLOPPY DISK SERVICES, INC.

the H-89 TWOET SYSTEMS

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

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

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

Model TWOET 455

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

Price .... $605.00 complete

Model TWOET 465

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

Price .... $755.00 complete

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

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

Dealer inquiries invited.

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

Toll Free Order Line:
(800) 223-0306

Tech Help or Info:
(609) 799-4440

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

FLOPPY DISK SERVICES™ INC.

39 Everett Dr., Bldg D Lawrenceville, NJ 08648

A Hard A.C.T. To Follow

A Review of a 5 Meg Removable Hard Disk

Terry Jensen
Software Developer

The contents of the SOLO Subsystem package.

Not too long ago, I can remember that the question most commonly asked between two computer users was; "How much memory does your system have?". At one time 8K was considered alot of memory. Programs had to be designed and written with the memory requirements clearly spelled out. Oh, but how times have changed! The H/Z-100 computer comes standard with 128K of memory, twice as much as possible on the previous computers. 256K memory expansion boards make the memory question a matter of interest not necessity. Programs are designed to use whatever memory is available. Today, however, one of the major concerns is not how much memory a system has but what is the disk storage capacity.

10, 20, 30, 40 megabyte (with up to 128 megabyte possible) hard disk drives are readily available in the microcomputer market. Large capacity disk drives generate a new problem; what affordable options are there for backing up the large data files possible with a hard disk drive unit? Until recently, floppy disk drives and high-speed tapes provided the only practical backup facility.

For most of you this will be your first introduction to removable hard disk cartridges. The latest technology designed into a removable cartridge combines the features of a mass storage device and a direct access device into one unit, with additional features such as 1) the entire unit operates as a separate hard disk drive, both in speed and accessibility, and 2) the disk media is transportable for storage and/or mobility.

This article is a review on one such removable drive unit but in addition I will take a close look at a company which was a pioneer in developing the technology used in hard disk systems today. H/Z-89, Z90, and H/Z-100 owners hold on to your hats, as you are about to be introduced to one supplier of the single most important unit of your computer system, second only to the computer itself.

An Overview

This product review was made on a side-by-side 10 megabyte hard disk and a 5 megabyte removable hard disk drive SOLO Subsystem available through A.C.T. Marketing (ACT) of Hunt Valley, Maryland. ACT has many different SOLO Subsystem configurations. Any combination of two hard disk drives from 5 meg to 41 meg (including the 5 meg removable hard disk drives) may be used side by side in a subsystem.

The entire subsystem package consisted of the 10 meg hard disk drive and the 5 meg removable hard disk drive mounted in a chassis similar to an H-37/77/87, an interface card, connector cable, power cord, removable disk cartridge, CP/M-85 and ZDOS software distribution disks and documentation. The disk controller card (unseen by the user) is mounted inside of the drive cabinet.

I was supplied little information on the 10 megabyte hard disk drive, so I cannot report the name of the drive or technical statistics about the drive. The disk drive was a four head drive with 306 data track surfaces. The 10 meg hard disk drive performed flawlessly throughout the review.

The removable hard disk unit (hereafter referred to as the "removable") consisted of two parts; 1) the disk drive, and 2) the storage media. The drive unit is the SyQuest SQ306R, a two head random access storage device. The storage media utilizes two 100 mm disc (306 data track) surfaces in one removable cartridge unit. The total formatted capacity of the "removable" is five (5) megabytes.

The interface card is just that; an interface between the computer (in this case, an H-100) and the disk controller unit of the subsystem. The Z100HD interface card uses the IEEE-696 standard S-100 buss at speeds up to 6 MHz for use with the H/Z-100 line of computers. Interface cards are available for H/Z-89 (Z-90) users, as well as a number of other computers. (The H/Z-89 and Z-90 units are supported by HDOS and CP/M.) This feature makes the SOLO subsystem very attractive as a product; the subsystem can be run on any computer, provided there is an interface card.

The Z100HD interface card was designed with a provision for an optional clock/calendar. According to the documentation, the clock/calendar is fully controllable by system software and has battery backup to support the clock when the system power is off. The clock/calendar option was not included on the Z100HD and will not be covered in this review. According to ACT, the parts and schematic are available upon request.

The CP/M-85 and ZDOS BIOS software modification and utilities provide support to the hard disk units. The subsystem formatting and defining utilities are written under CP/M-85.

The disk controller card, which is mounted inside the subsystem, was designed in 1980 and has been field tested and has proven itself to be a very reliable unit. It is used in each of the subsystems to be interfaced and operates with several different computer manufacturers.

The software and hardware of the SOLO Subsystem support the

operation of two hard disk drive units on-line at one time. The two drive units can be any combination of fixed hard disk drives (of any size) and the removable hard disk drives.

The documentation consisted of three manuals; the "SOLO Winchester Disk Subsystem Operation and Installation Manual", the "A.C.T. Marketing Z100HD S-100 Winchester Interface and Clock/Calendar Card Manual", and the "SQ306 OEM Manual". The manuals covered distinct areas.

A Closer Look

As with any review, I always read through the documentation before attempting to setup and run the product. It is nice to have a "feel" for the product before I actually start in.

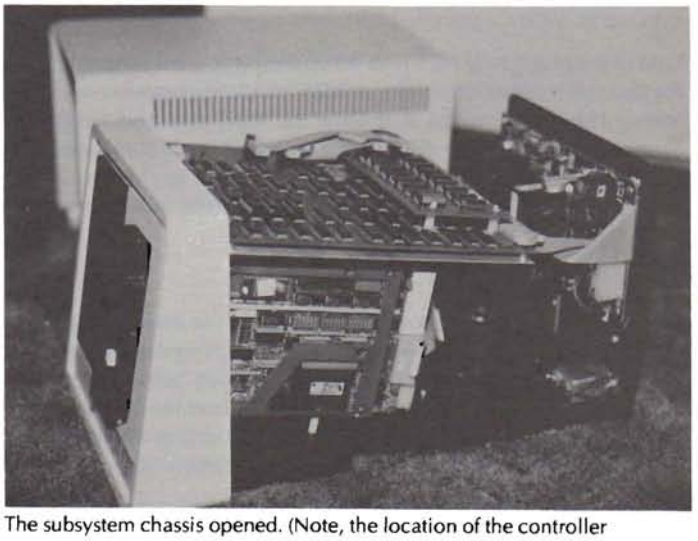
The technical information, covered in the Operation and Installation Manual, was very detailed and complete. Interfacing information was spelled out very explicitly; including component description, controller commands, and port assignments. A detailed description of the SOLO Digital Data Separator, the first hard disk digital data separator ever invented, was also included. Virtually all technical information has been documented and for that I give ACT an "A+".

The technical information, however, overpowers the setup and first-time through documentation, which is what most purchasers will be looking for. The Operation and Installation Manual was laid out with three chapters dedicated to technical information, followed by one chapter on the hardware interface to ten (10) different computers, and concluded with one chapter on the software modifications and utilities for the same ten computers. Most of the reading in the manual was not pertinent to the setup and running instructions for the SOLO unit on one particular computer, in my case the H-100. Some of the Z-100 documentation had become outdated since the Operation and Installation Manual was published.

The "Z100HD Interface" manual, however, contained the updated "first-time through" documentation. The Interface Manual contained all the pertinent information I needed to install the interface card and software. I did not feel it was presented in an easy "step by step" format but found that by reading the material carefully (and a couple of times), I had little difficulty working through the setup procedure. I rated the setup documentation a "B".

Hardware and Software Installation

The hardware installation of the interface card took very little effort. The most important part of the instructions was for the user (myself) to verify that the dip switch was set properly, (which it was). The entire hardware installation consisted of inserting the interface card into the S-100 slot of the H-100 and plugging in the connector between the interface card and the subsystem. The hardware installation gets an easy "A".



The subsystem chassis opened. (Note, the location of the controller

The software installation took careful reading and concentration to complete. By reading completely through the documentation before attempting any use of the software, I felt more comfortable taking the first step toward installing the software.

The software support programs consist of new versions of the CP/M-85 BIOS and the ZDOS IO.SYS. In order to use ZDOS with the winchester subsystem, the installation steps under CP/M-85 had to be done first. The formatting utility (DISKTEST) and the disk-defining utility (DISKDEF) were only available under CP/M-85. ZDOS did not require these utilities due to the fact that the ACT support BIOS for ZDOS uses sector skewing compatible with the CP/M-85 skewing.

The program DISKTEST (under CP/M-85) contained the formatting utility and must be run first if the hard disks are not formatted. DISKTEST also contains the diagnostic routines to test the hard disk units. The format routine, as expected, destroys all information stored on the disk units.

I found that DISKTEST had a minor bug when running a seek test. According to ACT, the problem was a result of the removable drive having a different timing characteristic than originally specified. This resulted in the head of the removable hard disk getting lost when seeking.

ACT preformatted the SOLO winchester subsystem before shipping, therefore, my first step was to define the disk partitions with the program DISKDEF. This turned out to be the most important step of setting up the software.

The disk partitions of the two hard disk drives must be divided between CP/M-85 and ZDOS. The ACT software will support two (2) partitions for CP/M-85 and two (2) partitions for ZDOS per hard disk drive. Two hard disk drives are supported for a subtotal of four (4) CP/M-85 partitions and four (4) ZDOS partitions, for a total of eight partitions. The ACT software supports all four (4) CP/M-85 or ZDOS partitions on-line at the same time. Whereas, the Zenith software for the Z-217 only supports two partitions on-line.

DISKDEF was very easy to use and was self-documenting. First, I was to define the physical characters, i.e. the number of cylinders and heads of the two hard disk units. This information was found in the manual and on the bottom of the subsystem chassis (not documented however). Second, DISKDEF prompted me to define the logical drive partitions to any size, from a minimum 150 tracks to a maximum of 1000 tracks (1.2 to 8 megabytes). Both CP/M-85 and ZDOS partitions are defined with DISKDEF. This information is written to the first addressed hard disk unit (Drive 0) and is used by CP/M-85 and ZDOS. As expected, these definitions would be erased by the format utility of DISKTEST.

The next step of installing the software under CP/M-85 was to copy the programs from the ACT distribution disk to a CP/M-85 system disk. The INSTALL program which came on the ACT CP/M-85 distribution disk was a CP/M SUBMIT file. INSTALL did the entire installation procedure of replacing the BIOS85 and BIOS88 system files on the CP/M-85 system disk, while I sat back and watched.

The CP/M-85 distribution disk contained a BUFFERED and NO-BUFFERED BIOS. The BUFFERED BIOS would provide for a slightly faster disk access on the hard disk drives, however, due to the bad timing specs (mentioned above), the BUFFERED BIOS could not be used with the SyQuest SQ306R.

The ZDOS distribution disk also contained a BUFFERED and NO-

BUFFERED IO.SYS. The ZDOS BUFFERED IO.SYS was supported and worked beautifully. The ACT ZDOS BIOS processes an entire read request in a single block transfer, which provides for optimum access on the hard disk drives. (See bench test results below.)

The ZDOS disk also contained a FORMACT program, which did not format the ZDOS partitions but rather built a new File Allocation Table and a blank directory structure for ZDOS.

Bench Mark Tests

The bench mark tests which I performed were based on executing a large number of disk accesses, so as to give an accurate picture of the difference between the floppy disk, hard disk, and removable hard disk. Each test was done with the command entry made from the system disk in drive A:. Each of the other drives had newly formatted surfaces.

The following are the drive specifications and results under CP/M-85:

```
Drive A:   ==>  system disk
Drive B:   ==>  second floppy
Drive C:   ==>  first hard disk partition
Drive D:   ==>  second hard disk partition
Drive E:   ==>  the only defined removable partition
```

Command line	Time
A>PIP B:=A:*.*[V]	3 min. 49 sec.
A>PIP C:=A:*.*[V]	2 min. 24 sec.
A>PIP D:=A:*.*[V]	2 min. 24 sec.
A>PIP E:=A:*.*[V]	4 min. 50 sec.
B>MBASIC	3.4 sec.
C>MBASIC	2.0 sec.
D>MBASIC	1.9 sec.
E>MBASIC	3.9 sec.

The following are the drive specifications and results under ZDOS:

```
Drive A:   ==>  system disk
Drive B:   ==>  second floppy disk
Drive E:   ==>  first hard disk partition
Drive F:   ==>  second hard disk partition
Drive G:   ==>  the only defined removable partition
```

Command Line	Time
A:COPY A:*.* B:	1 min. 32 sec.
A:COPY A:*.* E:	48 sec.
A:COPY A:*.* F:	48 sec.
A:COPY A:*.* G:	51 sec.
B:ZBASIC	5.3 sec.
E:ZBASIC	1.5 sec.
F:ZBASIC	1.5 sec.
G:ZBASIC	1.6 sec.

Note the big difference in performance of the removable hard disk drive between CP/M-85 and ZDOS, drive E: and G:, respectively. The CP/M-85 version actually took longer to do the PIP than the floppy drive. Once again the seek timing was the problem, causing the drive to do several retries with each file transfer. I actually thought something was wrong with the drive, until I saw the superb performance under ZDOS.

The poorer performance of CP/M-85 verses ZDOS for both the hard disk drive and removable is the result of the way CP/M reads only 128 bytes of data at a time. CP/M is optimized for floppy disk access, and cannot be changed for hard disk drive units. ZDOS, on the other hand, has been optimized for hard disk accessing by the ability to execute an entire block read.

Time for a few notes:

1) ACT has updated the existing installation software under CP/M-85. The seek "bugs" in the CP/M-85 software which I received have been fixed. This will greatly increase the performance of the removable unit by correcting the retries while seeking. The update will not affect the way CP/M reads 128 bytes of data.

2) A complete installation software program has been written and released for ZDOS. No longer does the H/Z-100 user have to have CP/M-85 to setup and use the SOLO subsystem.

This information was furnished by Bruce Johnson, Software Engineer of ACT and Mike Cogswell of MicroMagic, the software writer for many of the programs supporting the SOLO Subsystems, including the ZDOS installation and support utilities which have just been released.

Unfortunately, I was not able to cover the performance of the CP/M-85 updates and the ZDOS installation programs in this review. This is due to the fact I had not received the software programs prior to completing the review.

On a related note, ACT will support the H/Z-100 on future releases of ZDOS. ACT may release support software for CP/M-86, if there is enough demand for it.

More Facts and Opinions

Once I had completed the software installation, I wanted to see if any of the ZDS winchester utilities and programs would work with the ACT subsystem. The documentation did not give any explanation or comparison of its software verses the ZDS winchester software. My curiosity forced me to find out for myself. This is what I found:

1) Only a couple of the CP/M-85 supported winchester programs have practical use with the ACT winchester software, such as BACKUP and RESTORE. However, these programs will only recognize the first two hard disk partitions. In addition, due to the fact that the ACT software supports all four possible partitions, the ASSIGN program isn't needed.

2) The same is true for the ZDOS programs used with the ACT software. The MAP program worked and even recognized the G: and H: drive partitions. ACT recommended not using it and warned of unpredictable results if it is used.

3) The ACT software does not provide for booting on any of the winchester partitions. This is due to the lack of required code in the Z-100 ROM and the proprietary rights of the Z-100 ROM. The boot feature may be incorporated in future versions of the software according to Mike Cogswell.

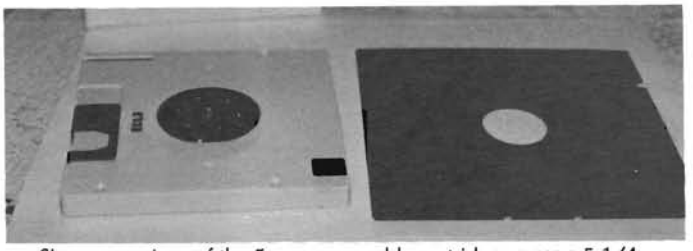
One of the projects of ACT is to set up the basic partitioning scheme to provide for complete transportability between computer manufacturers. This would mean that once the hard disk and removable drives are formatted, the drives could be read by any computer using the ACT interface. This would provide the ability to change computer systems, but maintain the existing formatted SOLO hard disk subsystem without losing a single file.

Two nit-picky comments:

1) When placing the removable cartridge in the drive unit, there is no indication when the drive is ready for accepting data. If an access on one of the removable partitions is attempted before the drive is ready, the whole system locks up. This was true for CP/M-85 and ZDOS. By waiting a full 15 seconds after inserting the removable cartridge, I eliminated any chance of that happening.

2) The fan inside the subsystem is noisy. Most reviews on the Z-100 state the fan is too loud, well, the SOLO Subsystem is at about the

same level as the Z-100. (I personally don't have any problem with the "loud" noise of the Z-100 or the SOLO Subsystem. Actually, I prefer it to a "so-quiet-you-can-hear-a-pin-drop" noise level.)



Size comparison of the 5 meg removable cartridge verses a 5 1/4 inch floppy.

Pricing

ACT offers a full two year warranty on the entire SOLO subsystem. This includes both the hard disk and the removable hard disk.

All SOLO Subsystems use the same disk controller card. The interface card provides the link between the computer and the subsystem. ACT has adapter sets for the IBM PC, COMPAQ, Osborne I and II, Otrona Attache', KAYPRO I and II, Seequa Chameleon, Telcom Zorba, Intertec Superbrain I and II, Tandy TRS Models II, III and 16, H/Z-89, Z-90, H/Z-100, Televideo TS-800, Columbia Data MPC, XEROX 820, S-100 Z-80 CPU's, among others.

The price of the drives used in this review are as follows:

10 megabyte drive (alone)	$2299.00
5 megabyte removable (alone)	$2900.00
10/5 megabyte subsystem	$4199.00

One cartridge is included with each 5 meg removable drive. Each additional cartridge is available through ACT for $100 per cartridge.

Contact ACT directly for pricing on the other drives and subsystems which they have available.

Final Thoughts

The removable hard disk offers a variety of unique uses; from back-up to an additional hard disk for that extra on-line storage. A tape drive unit will backup an entire winchester unit but that creates a backup each time of system files, programs, and utilities. The removable cartridge with its 5 megabytes of storage can be used to backup just those important data files which are updated continuously. That can be done very easily by creating a submit or batch file to do the repetitious copying of the files that are to be stored.

With the announcement that the ACT software under CP/M-85 has been updated and that the ZDOS installation software has been released, the ACT SOLO Subsystem with the removable hard disk unit becomes a feasible on-line hard disk AND storage media for the H/Z-100 computer under CP/M-85 and ZDOS.

When you think about it, a hard disk drive system may cost more than the computer it is to be used with. When deciding which system to choose, you need to pick a reliable drive and a company to support it. ACT subsystems have been field tested and supported since 1980. The hardware performance is superb. The software support has the limitation of not booting from the winchester, but then both CP/M-85 and ZDOS will address four partitions on-line. The A.C.T. Marketing SOLO Subsystems, with their full two year warranty, is one company that offers a proven and reliable hard disk subsystem.

For detailed information on the A.C.T. SOLO Subsystems contact:

A.C.T. Marketing
104 Lakefront Drive
Hunt Valley, MD 21031
(301) 628-0260

The HUG SIG .... What Am I Missing???

Bill Parrott
7010 Caenen
Shawnee, KS 66215

I've been a member of the HUG Special Interest Group (or bulletin board) on CompuServe since it first began operation and every time someone asks me what the SIG might offer to them, all I seem to be able to come up with for an answer is "Well, gee... lots of neat stuff!". The problem I have is that there is so much neat stuff available and so much happening on the SIG that I am unable to think of a place to start listing. I am excited about the SIG because of the things it can offer and if I may have your attention for a few minutes, I'd like to let you in on what must certainly be the single most valuable resource available to users of Heath/Zenith computers today.

I'll start with the first thing you see when you sign in to the SIG... the message base. As of this writing the membership of the SIG stands at over 2500 members and is increasing at a steady rate. The message base serves primarily as a forum for exchange of information regarding our hardware and software including products from vendors other than Heath or Zenith. This immediately puts at our disposal the knowledge of over 2500 other users. For example, a user recently had a problem getting his WH-8-37 controller card to run in his H8 computer at 4 MHz. A message or two on the SIG and the solution was made available to him (and to the rest of us). Any guesses what Heath Technical Correspondence would say if you called THEM with a problem like that? Another example of more general interest is the user who found a bug (gasp!) in CP/M-86 for the H/Z-100 and not only reported it to the users of the SIG but even supplied a solution! Keep in mind that this was BEFORE Heath Technical folks even knew the bug existed. Then there is the user who wrote and made available patches to dBase II to use the H/Z-19 keypad keys. Now I could go on, and on, and on, and... but I think you get the idea. The knowledge base which is present on the SIG is greater and more complete than you'll find reading any reference guides, articles, or books and it covers the entire line of Heath/Zenith hardware and software. There is even a section for robotics for those who have HERO I to occupy their time while their computer is busy doing something else. I should point out at this point that the message base (or bulletin board) is divided into ten sections to make it easier for users to leave and find messages pertaining to particular subjects. The sections include General Information, Requests for Assistance, Telecommunications, HDOS, CP/M ™, CP/M-86 ™, Z-DOS ™, Hardware, Robotics, and For Sale.

Have something to sell? Quite a few members have either bought or sold pre-owned (used?) computers, terminals, printers, plotters, and other assorted surplus hardware. This is the place to find a real deal if you are in the market for that elusive disk controller card or extra terminal or spare disk drive or whatever. Even entire systems are made available from time to time if you are thinking of upgrading, etc.

For those who love a good rumor and seem to make a hobby of speculating what Zenith and Heath might have planned, there are always juicy tidbits of information to be found on the bulletin board. Of course, they are just rumors but it can often be fun and exciting trying to separate the fact from the fiction, not unlike solving a jigsaw puzzle.

Great! Now we've established that the message base or bulletin board has much to offer, but what else is there? What probably the majority of SIG members initially join for is access to the SIG data base. This data base includes at this writing over 6 megabytes of files, programs, documentation, etc. All in all, well over 600 files. Offerings include programs and enhancements written for all available Heath operating systems and include everything from games to astronomy to modem software to payroll. There are even programs written for HERO I. All submissions have been either written or adapted for Heath/Zenith hardware by fellow HUGgies and are available for downloading by other members at no charge (other than standard CompuServe connect charges). I should mention here that if what you want is not among the files found in the HUG SIG data base, there are other SIGs on CompuServe which may be accessed. Of most immediate interest is the CP/M SIG (known as CP-MIG) which boasts a membership (and corresponding data base) more than three times the size of the HUG SIG. Predictably, CP-MIG is devoted to support of CP/M and CP/M-86 and related operating systems.

So what else? You mean that's not enough? Well how about weekly live conferences with fellow HUGgies? Each Saturday evening SIG members gather for a live on-line "rap session" or conference. Topics of discussion might be the latest new computer or just the weather but you can be sure a good time is had by all. In fact, it can occasionally get a little crazy, what with the bad jokes, harassing of certain editor of REMark, and the like. But seriously, what transpires during the weekly conference can at times be enlightening to say the least. And if you get nothing else from the SIG, you will make some of the best friends you will ever have occasion to know. When I first used the SIG those years ago, my primary concern was for what I might learn and get from the SIG (The conference facility did not exist in those days!), but I find that I now derive much more from the interaction with the many members whom, although I have never "met", I consider very good friends. It was exactly because of this that the first discussions of a National HUG Conference began on the SIG!

What I've tried to do here is to provide some idea of what the HUG SIG has to offer to all HUG members. Certainly, if you have any interest in getting the most from your Heath/Zenith computer, the HUG Special Interest Group on CompuServe can be a most valuable resource. Let me just add at this point, if you are not completely sold on the SIG yet, all users of the SIG receive a $.50/hour discount from their standard CompuServe connect charges as long as they are in the SIG, its associated data bases, or conference section. The reason for this is that HUG has arranged with CompuServe to return the royal-

ties earned by the SIG (normally paid to the SYSOPs) to the users in the form of a discount. The HUG SIG is the only SIG present on CompuServe which does this.

The only thing left is for me to tell you how you can become a part of the HUG SIG. The best way is to order the MicroNET Connection from HUG (p/n 885-1122[-37] for HDOS, 885-1224[-37] for CP/M) which contains everything you need to get into CompuServe and the HUG SIG, including your personal user number, password, and modem software. You will need a modem too, of course. Once you have these you are ready to become a part of the exciting HUG SIG and begin reaping its benefits, not to mention contributing your own knowledge and experience to other users. Everyone wins and no one loses!

I'll be looking forward to seeing you on the SIG soon!

OOPS!!!

A listing was left out of the article "ZBASIC Mapping Program: BASMAPER" which appeared on page 23 of the February 1984 issue. On page 26, where it says ***INSERT LISTING***, the following listing should appear:

```
A=011, B=012, C=013 and D=014
```

We apologize for any inconvience this may have caused.

ZDOS RAMDRIVE
Added Flexibility For The H/Z-100

Walt Gillespie
REMark Editor

Ever wondered just what to do with the extra memory over 64K on your H/Z-100? Yes, some programs do utilize that extra memory, but ZBASIC, for one, does not. It uses only about 5K above the normal 64K that MBASIC uses under CP/M. And now there are add-on 256K memory boards like the Z-205. Just how can you put all that additional memory to work for you?

One answer is RAMDRIVE, a ZDOS disk memory utility written by Shing Wong of Wong's Advance Technologies. RAMDRIVE will provide, through your additional memory, an extra (memory) disk drive. This RAM drive, called drive G:, can contain up to 700K of user area in increments of 2K. On a two drive H/Z-100 this may provide the answer to problems such as where to put the programs so two data disks may be used.

Along with the RAMDRIVE.COM program the distribution disk contains a ZBASIC program called DTEST. The purpose of DTEST.BAS is to prove just how much faster reading and writing to a 'memory-disk' can be. On my Z-100, reading and writing to the floppy disk drive through DTEST took 138 seconds while using RAMDRIVE the same test took 4 seconds.

Installation is easy. Just type RAMDRIVE 20 for a 20K RAM drive. The only real drawback I found to this program is once it has been installed you must reboot in order to re-initialize the RAM disk. Instructions for RAMDRIVE are minimal, just a single sheet of 8.5 x 11 paper. But, there really isn't that much you need to know about this program. Most all ZDOS programs will work with RAMDRIVE including COMMAND.COM, CHKDSK.COM, MASM.COM, and ZBASIC. DSKCOPY and FORMAT will not work as they require a physical disk to operate.

Conclusion

If you run programs that require a lot of reading and writing to disk, especially temporary files, or two disks just can't hold everything you need, then RAMDRIVE is for you. The price of $49.95 plus $2.00 shipping is a little high but again if this program does what you need it may well be worth it.

Author:	Shing Wong Wong's Advanced Technologies Kenner, LA
Distributor:	Omega Systems, Inc. 11814 Coursey Blvd. Suite 510 Baton Rouge, LA 70816 (504) 923-0388
Price:	$49.95 RAMDRIVE $2.00 Shipping & Handling Discounts for Multiple orders, etc.
Machines:	Heath/Zenith 100 (ZDOS)

HUG NEW PRODUCTS

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

885-3010-37 Z-DOS KEYMAP Function Key Mapper $20.00

Introduction: Z-DOS KEYMAP is a program that lets you designate the responses produced by your computer's function keys. It works like the KEY command in ZBASIC except that more keys can be defined, and the defined keys are part of the "system" and can be used with any program, not just with ZBASIC.

Requirements: Z-DOS KEYMAP requires the Z-DOS operating system on an H/Z-100 series computer or an ET-100/ETA-100 computer and at least 128K of user memory.

This disk contains the following files:

README	.DOC	KEYSYS	.COM
KEYMAP	.DOC	KEYSYS	.DOC
KEYMAP	.COM	DATETIME	.COM
KEYCON	.COM	UNMAP	.COM
KEYWS	.COM	KEYMAP	.ASM
KEYWS	.DOC	KEYCON	.ASM
KEYBAS	.COM	UNMAP	.ASM
KEYBAS	.DOC	ASM	.BAT

Author: Patrick Swayne, HUG

KEYMAP -- This is the executable KEYMAP program, provided in unconfigured form so that you can set up the keys the way you want to. It allows you to define a response of up to 20 characters for each of the following keys: F0 through F12, SHIFT-F0 through SHIFT-F11, I CHR, D CHR, INS LINE, DEL LINE, HOME, the Arrow keys, and the HELP key. If the keypad is shifted, the 1 through 9 keys can also be defined. You can designate one of the keys as an alternate response key, which gives all of the other keys two responses of up to 20 characters each. A total of 35 different responses can be produced without an alternate response key, or 69 responses with one. In addition to the ability to define keys, Z-DOS KEYMAP offers these other features:

** Off/on toggle. A control code (normally CTRL-SHIFT-6) is provided to toggle KEYMAP on or off, so that it can be temporarily disabled to allow other programs to control the function keys.

** Off line toggle. A control code (normally CTRL-\) is provided to allow the terminal section of your computer to be taken "off line" so that you can enter escape sequences to set terminal characteristics, etc. This function duplicates the badly missed OFF LINE key found on earlier Heath/Zenith computers.

** Coexistence with other programs. Not all of the keys must be configured with KEYMAP. Some can be left "unconfigured" so that KEYMAP can coexist with programs such as Z-BASIC. For example, if you configure only the shifted function keys, the ZBASIC KEY command will still work properly with the unshifted function keys.

KEYCON -- This program is used to configure the KEYMAP program, and allows you to designate the response of each mappable key.

KEYWS -- This is a pre-configured KEYMAP for use with WordStar. The requirement to use hard-to-remember control codes is practically eliminated. Cursor and text movement, indenting, centering, underlining, and many other functions are available at the touch of function or keypad keys. Even if you use the new version of WordStar with programmable keys, you will find that KEYWS gives you more programmed keys in an easier-to-use layout.

KEYBAS -- This is a pre-configured KEYMAP for use with ZBASIC. 33 BASIC keywords are "programmed" into your keys without interfering with the ZBASIC KEY command or the editing keys.

KEYSYS -- This is a pre-configured KEYMAP for use with the operating system. Commands such as DIR, DATE, TIME, etc. are available at the press of a key. One of the keys runs the DATETIME program that was published in REMark. DATETIME is included on the disk.

UNMAP.COM -- A program that disables KEYMAP. It lets you change from one KEYMAP to another without re-booting.

Comments: Ever since Pat wrote the CP/M KEYMAP program, we have had many requests for a version under ZDOS. Well, here it is! ZDOS KEYMAP is a program that every H/Z-100 ZDOS user should have.

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

885-5001-37 CP/M-86 KEYMAP Function Key Mapper $20.00

Introduction: CP/M-86 KEYMAP is a program that lets you designate the responses produced by your computer's function keys. Up to 20 characters, including control characters and RETURNs can be programmed into a single keystroke. When loaded, KEYMAP becomes part of the "system", so that the expanded key responses are available to any program.

Requirements: CP/M-86 KEYMAP requires the Heath/Zenith version of the CP/M-86 operating system on an H/Z-100 series computer or an ET-100/ETA-100 computer and at least 128k of user memory.

This disk contains the following files:

README	.DOC
KEYMAP	.DOC
KEYMAP	.CMD
KEYCON	.CMD
KEYWS	.CMD
KEYWS	.DOC
KEYBAS	.CMD
KEYBAS	.DOC
KEYSYS	.CMD
KEYSYS	.DOC
UNMAP	.CMD
KEYMAP	.A86
KEYCON	.A86
UNMAP	.A86

Author: Patrick Swayne, HUG

KEYMAP -- This is the executable KEYMAP program, provided in unconfigured form so that you can set up the keys the way you want to. It allows you to define a response of up to 20 characters for each of the following keys: F0 through F12, SHIFT-F0 through SHIFT-F11, I CHR, D CHR, INS LINE, DEL LINE, HOME, the Arrow keys, and the HELP key. If the keypad is shifted, the 1 through 9 keys can also be defined. You can designate one of the keys as an alternate response key, which gives all of the other keys two responses of up to 20 characters each. A total of 35 different responses can be produced without an alternate response key, or 69 responses with one. In addition to the ability to define keys, CP/M-86 KEYMAP offers these other features:

** Off/on toggle. A control code (normally CTRL-SHIFT-6) is provided to toggle KEYMAP on or off, so that it can be temporarily disabled to allow other programs to control the function keys.

** Off line toggle. A control code (normally CTRL-\) is provided to allow the terminal section of your computer to be taken "off line" so that you can enter escape sequences to set terminal characteristics, etc. This function duplicates the badly missed OFF LINE key found on earlier Heath/Zenith computers.

** Coexistence with other programs. Not all of the keys must be configured with KEYMAP. Some can be left "unconfigured" so that KEYMAP can coexist with programs that must use the original key responses. For example, if you configure only the shifted function keys, unshifted function keys will continue to function normally.

KEYCON -- This program is used to configure the KEYMAP program, and allows you to designate the response of each mappable key.

KEYWS -- This is a pre-configured KEYMAP for use with WordStar. The requirement to use hard-to-remember control codes is practically eliminated. Cursor and text movement, indenting, centering, underlining, and many other functions are available at the touch of function or keypad keys.

KEYBAS -- This is a pre-configured KEYMAP for use with BASIC. 34 BASIC keywords are "programmed" into your keys to make developing BASIC programs easier.

KEYSYS -- This is a pre-configured KEYMAP for use with the operating system. Commands such as DIR, STAT, FORMAT, etc. are available at the press of a key.

UNMAP -- A program that disables KEYMAP. It lets you change from one KEYMAP to another without re-booting.

Note: If you would like to program your function keys under 8-bit CP/M-80 or CP/M-85, order the original KEYMAP program (HUG part no. 885-1230[-37]).

Comments: This version of KEYMAP will give you complete control of your H/Z-100 computer's function keys under CP/M-86.

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

HUG Price List

The following HUG Price List contains a list of all products not included in the HUG Software Catalog or in the January 1984 issue of REMark. For a detailed abstract of these products, refer to the issue of REMark specified.

Part Number	Decription of Product	Selling Price	Volume - Issue
HDOS			
885-1030[-37]	Disk III, Games II	$ 18.00	5-2
885-1096[-37]	MBASIC Action Games	$ 20.00	5-2
885-8026	Space Drop	$ 16.00	5-2
885-8027	HDOS SCICALC	$ 20.00	5-3
CP/M			
885-1234[-37]	CP/M Ham Help	$ 16.00	5-2
885-8025-37	CP/M 85/86 FAST EDDY	$ 20.00	5-2
ZDOS			
885-3009-37	ZBASIC Dungeons & Dragons	$ 20.00	5-3
885-8028-37	ZDOS SCICALC	$ 20.00	5-3
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	
885-4004	REMark Volume 4, Issues 36-47	$ 20.00	
885-4700	HUG Bulletin Board Handbook	$ 5.00	5-3

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

Ordering Information

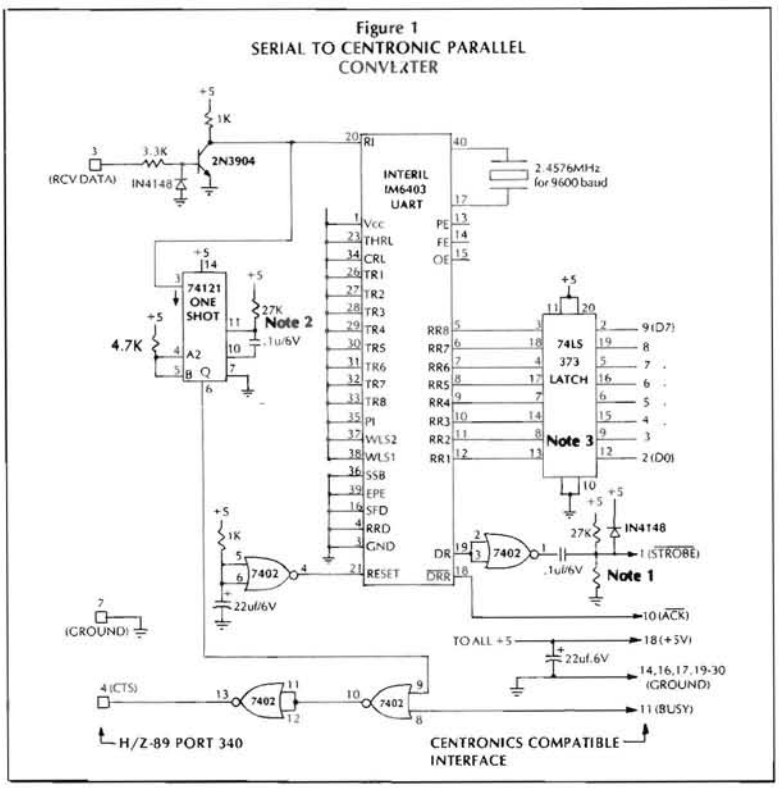
For Visa and MasterCard phone orders; telephone Heath Company Parts Department at (616) 982-3571. Have the part number(s), description, and quantity ready for quick processing. By mail; send order, plus 10% postage and handling, up to a maximum of $3.50 to Heath Company Parts Department, Hilltop Road, St. Joseph, MI 49085. Visa and MasterCard require minimum $10.00 order.

Any questions or problems regarding HUG software or REMark magazine should be directed to HUG at (616) 982-3463. REMEMBER - Heath Company Parts Department is NOT capable of answering questions regarding software or REMark.

A Simple Serial To Centronics Parallel Converter

J. D. Ross
Custom Electronics
1307 Darlene Way Suite A12
Boulder City NV 89005

Figure 1
SERIAL TO CENTRONIC PARALLEL
CONVERTER



Notes:

1. Adjust resistor for 3.5 volts at pin 1 ($\overline{\text{STROBE}}$).
   OKI84 → 680 OHM
   JUK16100 → 3.9K ohm
2. 15K OHM for 19200 baud.
3. 74LS373 not required if printer pull-up > 2.5K OHM.

The new OKIDATA line of printers are out - and WOW what great features! Near letter quality, down-loadable graphics, and print speeds of 160 characters per second (ML92 and ML93) or 200 characters per second (ML84). And prices beat most any of the other printers without these features. But there's one problem for Heath/Zenith computer owners - no serial interface. By the time all the neat features were designed into the printer's micro, there was no room left for the serial interface (which was standard on the ML82A and ML83A). An optional high speed serial interface board is available, but adds another $100 to $150 to the price of the printer.

The circuit described in Figure 1 is a bare bones serial to parallel converter which will allow the new OKIDATA printers, or any other Centronics compatible printer for that matter, to connect to the H/Z-89 or the H8. It utilizes only 4 integrated circuits, and draws it's power supply from the printer. Baud rate is fixed at 9600 baud, which allows the printer to accept characters much faster than they can be printed. The only drawback of this scheme is that, since the characters from the computer are not buffered by the serial to parallel interface, the handshaking must ensure that the next character is not sent out on the serial link before the previous character has been accepted by the printer. The 74121 One Shot takes care of this by forcing a "Printer Busy" signal back to the computer as soon as a character is seen on the link. This forced busy signal remains on for slightly over one character transmission time (1 millisecond at 9600 baud). After the One Shot times out, the printer may continue to enforce the busy signal if it is unable to take any more characters. The net result of all this is that characters will be output on the serial link at about half the rate that they could be if the printer interface was buffered. On my 200 CPS ML84, this effect is negligible.

A few notes on the parts. The Intersil IM6403 UART is available from JDR Microdevices, San Jose, Cal, at $8.95. The crystal is also available from JDR at $3.95. Jade Computer Products, Los Angeles, CA, also has these parts at slightly higher prices, along with the Centronics Type connector (male) at $10.95.

Standard HDOS and CP/M software works with the OKI printers. Use the H24 device driver supplied with HDOS 2.0 and set the baud rate to 9600. The initialize routine which runs each time the printer is accessed causes a couple of front end garbage characters. The source is available on the HDOS 2.0 device driver diskette, and the initialization characters can be easily changed. They are just past the label INITLP EQU * in LPH24.ASM.

A better way to go is the UD.DVD device driver or the UDSP.DVD spooler available from Jim Teixeira's SoftShop, Sudbury, MA. With these device drivers, set CTS to yes, and set IVC to yes for any of the devices LP0: through LP7: that will be communicating with the OKI. The default baud rate is already set at 9600.

No changes should be necessary in CP/M besides setting the baud rate to 9600.

Checklist for Z-100 Computer Users:

1. Are you getting the best prices?

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

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

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

3. Do you receive merchandise on time?

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

4. Is your supplier a proven performer?

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

5. What are your dealer's references?

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

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

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

Software Wizardry
THE MAGIC TOUCH

ZENITH | data systems
AUTHORIZED SALES & SERVICE

MasterCard VISA

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

"My Favorite Subroutines"

Dear HUG,

I like Bob Moskus' suggestion about his (and our) favorite subroutines. Most of the computer magazines have gotten away from the fun part of owning a home computer and are concentrating on selling us things they say we must have. REMark is one of the few that still includes some things to do with our computers.

Here is one of my favorite BASIC subroutines. It is a fairly simple timer routine that I use quite a lot to compare different methods for best efficiency. It is for HDOS only, I guess. I don't know how to use it with CP/M and I don't know anything about ZDOS. If you have a 4MHz modification, it will certainly show what you've gained.

The routine starts at line 1000. Lines 1016 and 1065 are necessary on my '89 but I don't think they should be. When the counter is to be updated and the program is peeking, the update is missed. The rest of this program is just a sample of its use. T5 contains the seconds that have elapsed since the last call to the routine.

```
00010 GOSUB 1000
00012 REM          Put the routine to be timed here
00014 REM          - for example;
00016 REM          To find how long it takes to sum
00018 REM
00020 FOR A = 1 TO 100
00022 B=B+A
00024 NEXT A:REM    End of routine
00026 REM
00028 REM          Get the time and print the results
00100 GOSUB 1000
00110 PRINT "THE TOTAL OF THE FIRST";@
      A-1;"NUMBERS IS";B,T5;"SECONDS:
00120 STOP
00130 REM
00140 REM
00150 REM
01000 REM          Timing subroutine
01005 REM
01010 T2=PEEK(8220)
01015 T1=PEEK(8219)
01016 IF T1<T8 AND T2=T9 THEN T2=T2+1
01020 T3=T2*256+T1
01030 IF T4>T3 THEN T4=T4-65535
01040 T5=(T3-T4)/500
01060 T4=T3
01065 T8=T1:T9=T2
01070 RETURN
```

Ralph Seiler
3977 So. 775 West
Bountiful, UT 84010

Send in your "Favorite Subroutine" to REMark today! Share you ideas with others.

Dear HUG,

Here is a nice subroutine that tells one the number of days between two dates. It takes the second date from DATE$ in ZBASIC. Perhaps some people will get the hint?

I hope the "My Favorite Subroutines" column (reference Buggin' HUG letter from Bob Moskus, REMark February 1984) gets off the ground.

```
10 MM(1)=2:DD(1)=13:YYYY(1)=1984:' The day I read Bob
   Moskus' suggestion.
20 GOSUB 30:PRINT
   "It has been "DP" days since I read Bob's letter":END
30 MM(2)=VAL(LEFT$(DATE$,2)):DD(2)=VAL(MID$(DATE$,4,2)):
   YYYY(2)=VAL(RIGHT$(DATE$,4))
40 FOR W=1 TO 2
50 IF MM(W)=1 OR MM(W)=2 THEN Z(W)=365*YYYY(W)+DD(W)+31*
   (MM(W)-1)+INT((YYYY(W)-1)/4)-INT(.75*INT(((YYYY(W)-1)
   /100)+1))ELSE Z(W)=365*YYYY(W)+DD+31*(MM(W)-1)-INT
   (.4*MM(W)+2.3)+INT(YYYY(W)/4)-INT
   (.75*INT(YYYY(W)/100)+1)
60 NEXT W
70 DP=Z(2)-Z(1):RETURN: 'DP is the number of days past
   since date one.
```

Matt Payne
6934 Charles St.
Omaha, NE 68132

Dear HUG,

The following subroutine is for MBASIC and the CP/M operating system. It acts as a "software switch" and makes PRINT statements act like LPRINT statements at my discretion. By changing the IOBYTE in memory location 0003, I can output the PRINT statements to either the CON: or the LST: logical devices.

```
10 IOBYTE=PEEK(3):TEMP=IOBYTE AND 252 OR 2
```

In a BASIC program, I ask on the screen if the program outputs should be displayed on the screen, or printed out as hardcopy. I then either POKE3, IOBYTE for screen display; or POKE 3, TEMP for hardcopy. It is a good idea to perform a POKE 3, IOBYTE prior to exiting the program to insure that all logical devices are returned to the default conditions. Here is a short sample program:

```
10 IOBYTE=PEEK(3):TEMP=IOBYTE AND 252 OR 2
20 PRINT
   "Do you wish output on the consol or on the printer?"
30 LINE INPUT "TYPE 'C' OR 'P'";A$
40 IF A$="C" THEN POKE 3, IOBYTE:GOTO 70
50 IF A$="P" THEN POKE 3, TEMP:GOTO 80
60 GOTO 20
70 PRINT "This will appear on the consol":GOTO 90
80 PRINT "This will appear on the printer"
90 POKE 3, IOBYTE:END
```

Ralph Stiewe
6320 N. Joyce Ave.
Milwaukee, WI 53225

Vectored from 5

Since my main interest is in graphics, soon to include H/Z-100 graphics as well, I am not interested in getting into the publishing business. I have been most pleased with the steady improvement in the format and content of REMark over the years. I, and the others interested in graphics, hope you will find the idea of a 'graphics corner' to be an idea which you can consider a useful addition to the contents of REMark.

Fred Pospeschil
3108 Jackson St.
Bellevue, NE 68005

More Response To 'PeachText 5000'

Dear HUG,

I write in some wonder at the different reactions people have to systems. I refer to the almost fawning article on PeachText 5000 by H. W. Bauman in the November and December issues of REMark. I wonder what prompts his enthusiasm?

Parts of PeachText are very good. As an editor PeachText (Magic Wand) is fine. It is flexible and easy to use, indeed I'm using it right now. PeachCalc is also fine as any user of SuperCalc would agree. The spelling checker and list manager leave a great deal to be desired! Still at the going price of about $200 - $250, it is a good deal.

Mr. Bauman makes much of PeachText's concept of "Total Integration". It is exactly here that we differ! As an integrated system PeachText is one of the worst and wrong headed botches I can imagine.

A few specifics:
1) You exit the editor with "end" <CR>. You exit the main menu with "en" <CR>. You exit the spelling checker with "e", no <CR>.

2) The editor, spelling checker, and print programs are constantly asking you to hit <CR> just to move on. It serves no purpose.

3) The entire main menu package must be present to use any part. The space is not too much of a problem, but PeachText seems to take forever to get through the menu. This is bad enough if there is reason for all this main menu nonsense. But there is no room on a disk for all the items on the main menu. There is not even enough room for just the word processing parts, editor, printer, speller, and thesaurus.

4) The system parts of PeachText seem to slow the application parts severely. A few minutes with Cherry Engineering's ESE editor or Steve Robbin's WatchWord spoils one for PeachText's antics. PeachText takes forever just to quit and get out of the way.

Vectored to 66 ☞

HERO THE ROBUTLER

Dr. Kenneth R. Hill
404 Ben Oaks Drive East
Severna Park, MD 21146

The robot wanders around the room or from room to room, avoiding obstacles, until commanded to "STOP". Another vocal command (any loud sound) will restart or jump to a different utility program (supplied by user) as desired. Any motion in the room after "There is something in my way" will cause the "Dinner" announcement, otherwise robot continues wandering looking for someone to talk to.

```
                                0200-03D6
0200 83          M.L.          )
0201 86 00      LDA A w $0     )
0203 97 10      STA A @ 0010   )
0205 97 40       "  " " 0040   )
0207 97 41       "  " " 0041   )--Initialize
0209 86 EE      LDA A w $EE    )
020B 97 11      STA A @ 0011   )
020D 3F          R.L.          )
020E 45         enable sonar   )
020F 83          M.L.          )
0210 BD 03 2B  JSR 3.                          START of main program.
0213 7C 00 40   INC (M+1->M; M=0040)
0216 96 40      LDA A w val. in 0040
0218 BD F6 4E   JSR (REDIS)
021B BD F7 AD   JSR (OUTBYT)
021E BD 03 70   JSR 4. (delay for display)
0221 81 50      CMP A w $50 =.80
0223 24 02      BCC
0225 20 07      BRA
0227 3F          R.L.
0228 21         zero all motors; center some
0229 83          M.L.
022A 86 00      LDA A w $10
022C 97 40      STA A @ 0040
022E 3F          R.L.
022F C3 D0 61   center head,med.,abs.,wait
0232 C3 F0 4A   center steer,med.,abs.,wait
0235 CC 0B FF   move forward,slow,abs.,cont. (or CC 13 FF for medium motion)
0238 83          M.L.
0239 CE 00 03   LDX w $03
023C 96 11      LDA A w val. in 0011
023E BD F6 4E   JSR (REDIS)    )          Display range value.
0241 BD F7 AD   JSR (OUTBYT)   )
0244 0C         CLC
0245 82 48      SBC A (A-M-C->A)           Range limit = 48H.
0247 25 02      BCS
0249 20 EE      BRA
024B 96 11      LDA A w val. in 0011
024D 0C         CLC
024E 82 20      SEC A (A-M-C->A)
0250 24 0A      BCC
0252 CE 00 0A   LDX w $0A
0255 BD 03 0B   JSR 1.
0258 25 07      BCS
025A 20 DD      BRA
025C BD 03 20   JSR 2.
025F 24 DB      BCC
0261 3F          R.L.
0262 02         abort drive motor
0263 72 FB 7B   speak.wait "There is something in my way"
0266 83          M.L.
0267 BD 03 79   JSB "DINNER"
026A 3F          R.L.
026B C3 C8 5E   turn head,CCW(L),slow,abs.,wait   -3   )
026E C3 D0 5B     "    "    "   ,med., "    "     -3   ) -16 "Look left"
0271 C3 D8 51     "    "    "   ,fast, "    "     -10  )
0274 8F 00 02   pause,1/8 sec
```

```
0277 83          M.L.
0278 CE 00 03    LDX w $03
027B BD 03 0B    JSR 1.
027E 24 0E       BCC
0280 3F          R.L.
0281 C3 EC 93    steer,CW,slow,abs.,wait              )
0284 D3 08 0D    move forward,slow,rel.,wait          ) - Turn R 90
0287 C3 E8 4A    steer,CCW,slow,abs.,wait             )
028A 83          M.L.
028B 7E 02 13    JMP (back to start+1)
028E 96 11       LDA A w val. in 0011
0290 BD F6 4E    JSR
0293 BD F7 AD    JSR
0296 97 41       STA A @ 0041
0298 3F          R.L.
0299 8F 00 03    pause,3/16 sec.
029C 83          M.L.
029D 96 11       LDA A w val. in 0011
029F B1 00 41    CMP A w val. in 0041
02A2 26 EA       BNE (br. if not = 0)
02A4 96 41       LDA A w val. in 0041
02A6 16          TAB (A->B;store L range in B)
02A7 3F          R.L.
02A8 8F 00 02    pause,1/8 sec.
02AB D3 C8 03    turn head,CW(R),slow,rel.,wait   +3   )
02AE D3 D0 03      "    "    "  ,med., "    "     +3   ) +32 "Look right"
02B1 C3 D8 71      "    "    "  ,fast,abs., "     +26  )
02B4 8F 00 02    pause,1/8 sec.
02B7 83          M.L.
02B8 CE 00 03    LDX w $03
02BB BD 03 0B    JSR 1.
02BE 24 0E       BCC
02C0 3F          R.L.
02C1 C3 E8 00    steer,CCW,slow,abs.,wait             )
02C4 D3 08 0D    move forward,slow,rel.,wait          ) - Turn L 90
02C7 C3 EC 4A    steer,CW,slow,abs.,wait              )
02CA 83          M.L.
02CB 7E 02 13    JMP (back to start+1)
02CE 96 11       LDA A w val. in 0011
02D0 BD F6 4E    JSR
02D3 BD F7 AD    JSR
02D6 97 41       STA A @ 0041
02D8 3F          R.L.
02D9 8F 00 03    pause,3/16 sec.
02DC 83          M.L.
02DD 96 11       LDA A w val. in 0011
02DF B1 00 41    CMP A w val. in 0041
02E2 26 EA       BNE
02E4 96 41       LDA A w val. in 0041
02E6 10          SBA (A-B->A)                     if R > L, turn right, etc.
02E7 24 11       BCC
02E9 3F          R.L.
02EA C3 F0 35    steer,CCW(L),med.,abs.,wait      )
02ED D3 08 10    move forward,slow,rel.,wait      ) - Turn L   15
02F0 C3 F0 4A    center steer,med.,abs.,wait      )
02F3 C3 D0 61    center head.med..abs..wait       )
02F6 83          M.L.
02F7 7E 02 10    JMP (back to START line)
02FA 3F          R.L.
02FB C3 F0 5F    steer,CW(R),med.,abs.,wait       )
02FE D3 08 10    move forward,slow,rel.,wait      ) - Turn R   15
0301 C3 F0 4A    center steer,med.,abs.,wait      )
0304 C3 D0 61    center head,med.,abs.,wait       )
0307 83          M.L.
0308 7E 02 10    JMP (back to START line)
030B BD F6 4E    JSR REDIS                        Subr. 1.
030E 96 11       LDA A w val. in 0011
0310 BD F7 AD    JSR OUTBYT
0313 0C          CLC
0314 82 20       SBC A (A-M-C->A)
0316 25 01       BCS
0318 39          RTS
0319 BD 03 20    JSR 2.
031C 25 FA       BCS
031E 20 EB       BRA
```

```
0320 0C          CLC                                    Subr. 2.
0321 09          DEX (X-1->X)
0322 26 01       BNE
0324 0D          SEC
0325 3F            R.L.
0326 8F 00 03    pause,3/16 sec.
0329 83            M.L.
032A 39          RTS
032B 3F            R.L.                                 Subr. 3.
032C 42          enable sound
032D 8F 00 02    pause,1/8 sec.
0330 83            M.L.
0331 CE 08 00    LDX w $0800
0334 BD F6 4E    JSR
0337 B6 C2 40    LDA A w val. in C240
033A BD F7 AD    JSR
033D 0C          CLC
033E 82 A0       SBC A (A-M-C->A)
0340 24 08       BCC (br. if A>M;M=A0)
0342 09          DEX
0343 26 EF       BNE
0345 CE 00 03    LDX w $03
0348 0D          SEC
0349 39          RTS
034A 3F            R.L.
034B 8F 00 04    pause,1/4 sec.
034E 72 FD 46    speak,wait "Your wish is my command"
0351 8F 00 04    pause,1/4 sec.
0354 83            M.L.
0355 BD F6 4E    JSR
0358 B6 C2 40    LDA A w val. in C240
035B BD F7 AD    JSR
035E 0C          CLC
035F 82 A0       SBC A (A-M-C->A)
0361 24 08       BCC
0363 BD 03 20    JSR 2.
0366 20 ED       BRA
0368 01 01 01    NO-OPS
036B 39          RTS       (replace with a jump to a user utility
routine if desired - change line 0361 accordingly)
036C 01 01       NO-OPS
036E 01 01       NO-OPS
0370 3F            R.L.                                 Subr. 4.
0371 8F 00 10    pause. 1 sec.
0374 83            M.L.
0375 39          RTS
0376 01 01 01    NO-OPS
0379 3F            R.L.                                 "DINNER" Subr.
037A 55          disable sonar
037B 83            M.L.
037C 0E          CLI
037D B6 7E       LDA w $7E
037F 97 27       STA @ 0027
0381 CE 03 CF    LDX w $03CF
0384 DF 28       STX @ 0028,0029
0386 CE 00 03    LDX w $0003
0389 3F            R.L.
038A 4B          enable motion detector
038B 83            M.L.
038C B6 00       LDA w 0
038E B7 03 D5    STA @ 03D5
0391 B6 03 D5    LDA w contents of 03D5
0394 26 0E       BNE (go to speak if motion)
0396 3F            R.L.
0397 8F 00 10    pause, 1 sec
039A 83            M.L.
039B 09          DEX
039C 26 F3       BNE (back to check for motion)
039E 3F            R.L.
039F 5B          disable motion dectector
03A0 5B            "       "         "
03A1 45          enable sonar
03A2 83            M.L.
03A3 39          RTS
03A4 3F            R.L.
```

```
03A5 5B          disable motion detector
03A6 72 03 AF    speak, wait
03A9 BF 00 50    pause, 5 sec
03AC 83            M.L.
03AD 20 F5       BRA (loop back to repeat speech)
03AF 1E          D
03B0 0B          I
03B1 0D          N        DINNER
03B2 0D          N
03B3 3A          ER
03B4 0B          I        IS
03B5 12          S(Z)
03B6 2B          R
03B7 02          EH1      READY
03B8 00          EH3
03B9 1E          D
03BA 29          Y
03BB 3E          PA1,  185 ms
03BC 3E          PA1,   "
03BD 25          P
03BE 18          L
03BF 3C          E1       PLEASE
03C0 21          AY
03C1 12          Z
03C2 3E          PA1,  185 ms
03C3 19          K
03C4 32          UH1      COME
03C5 23          UH3
03C6 0C          M
03C7 3E          PA1
03C8 8D          N    (+2=8D)
03C9 95          AH1 (+2=95)    NOW!
03CA 63          UH1 (+1=63)
03CB 37          U1
03CC 2D          W
03CD 03          STOP
03CE FF          end of speech
-------------------------------
03CF 86 01       LDA w $01
03D1 B7 03 D5    STA @ 03D5
03D4 39          RTS
03D5 00          data
03D6 FF          END
```

An Introduction To 'C'

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

This is the sixth in a series of articles intended to introduce the 'C' programming language.

Before getting started with new stuff, let's take one last look at last month's program. That program was called 'wc' and counted the number of lines, words, and characters in a file. Two things about that program need further clarification.

You may have noticed that we checked to see if a flag was turned on simply by specifying it in an 'if' statement (e.g. 'if (character_flag)'). We can do this because any non-zero expression will evaluate as 'true' in a logical syntax. We can also simplify an expression such as 'if (character_flag==0)' to 'if (!character_flag)', where '!' means 'not'.

Along the same lines of logical expressions, I used the '||' operator without mentioning what it did. It should have been obvious from the context that this is the logical 'or' connector. The logical 'and' connector is '&&'.

The program we will analyze today is called 'phone' and is a simplified telephone database. As it will be presented, it is probably quite useless, but with a little imagination and newfound 'C' ingenuity, it can be developed into a very useful tool.

The program uses the 'scanf' function for input. This is the input equivalent of 'printf', and its syntax is similar. A list of conversions is supplied, along with a list of variable addresses. For example,

```
scanf("%d", &i);
```

will read one decimal integer from the standard input, and put it into the variable 'i'. Notice that the variable expression represents the ADDRESS of 'i'. This is an important point to note when using 'scanf'. Make sure all variables mentioned are pointers. Usually, this means that the variable will be preceeded by the '&' symbol, except for character arrays. The source code for this function resides in "scanf.c", which must be included in our program. Watch out for this file, it has a bug in it. The file contains a declaration of a 'long' and a 'float', which are invalid unless you have the 'Mathpack' option. If you don't, you must edit this file and comment out the declaration. The line to be changed is:

```
static union { char c; int i; long l; float f; } **STKtop;
```

Change it to:

```
static union { char c; int i;
/* long l; float f; */ } **STKtop;
```

This fix seems to work fine. If you do have the 'Mathpack' option installed, then you can leave the above line unchanged and instead remove the comments from '#define FLONG' on the first line of the file.

One of the handy features of 'C' is the ability to intermix assembler code with 'C' code. There have been many articles written recently about how to get assembler subroutines into MBASIC. It's nice to have a language where the capability is easily utilized. By the way, the reason it is so easy to accomplish is that the 'C' compiler generates assembler code, so all it does is include our code 'as is' in the appropriate place. One word of caution: be careful when including assembler code which contains the 'ORG' assembler statement. This changes the memory location where code is loaded and can cause unpredictable results.

Many times we would like to change the terminal from line mode to character mode so that response is instantaneous to input. In line mode, the operating system waits for a carriage return before processing data from the terminal. In character mode, each character is processed as it is typed. This is equivalent to using the INPUT$ function in MBASIC. I couldn't find a way of switching modes from within 'C', but I did know of an easy way to do it in assembler. Therefore, I included this code in the appropriate spots. All we have to do is put '#asm' before the code, and '#endasm' afterward. If you don't know assembler, don't worry about what the code actually does. Just realize that it changes the terminal's mode. This assembler feature is handy for including routines such as this one into a program.

Here's this month's program:

```
#include "tprintf.c"
#include "scanf.c"
#define EOF -1
#define NULL 0
#define SCOUT 6
extern int fin;
main()
{
char name[10][30], s_name[30];
int area_code[10], s_area_code;
int exchange[10], s_exchange;
int extention[10], s_extention;
char description[10][30], s_description[30];
int c,i,entries;

/* let's read the input file into an array */

if ((fin=fopen("sy1:phone.dat","r")) == NULL)
   {
   printf("File Not Found.\n");
   exit(8);
   }
for (i=0; (scanf("%s %*s %d %d %d %s",
                                name[i],
                                &area_code[i],
                                &exchange[i],
                                &extention[i],
                                description[i])) != EOF; i++);
entries = --i;
fin=0;
/* let's ask what field to search on */
putchar(27);
putchar('E');   /* erase screen */
printf("Phone Selection Menu\n\n");
printf("1 = Search On Name\n");
printf("2 = Search On Area Code\n");
printf("3 = Search On Exchange\n");
printf("4 = Search On Extention\n");
printf("5 = Search On Description\n\n");
printf("Enter Selection: ");
/* change terminal to character mode */
```

```
#asm
        XRA     A
        MVI     B,201Q
        MVI     C,201Q
        SCALL   SCOUT
#endasm

c=getchar();

/* change terminal to line mode */
#asm
        XRA     A
        MVI     B,000Q
        MVI     C,201Q
        SCALL   SCOUT
#endasm
printf("\n");
switch(c)
{
case '1': printf("Enter Search 'name': ");
          scanf("%s", s_name);
          for(i=0; i<=entries; i++)
             if(!strcmp(s_name, name[i]))
                printf("%s %d %d %d %s\n", name[i],
                                           area_code[i],
                                           exchange[i],
                                           extention[i],
                                           description[i]);
          break;
case '2': printf("Enter Search 'area code': ");
          scanf("%d", &s_area_code);
          for(i=0; i<=entries; i++)
             if(s_area_code==area_code[i])
                printf("%s %d %d %d %s\n", name[i],
                                           area_code[i],
                                           exchange[i],
                                           extention[i],
                                           description[i]);
          break;
case '3': printf("Enter Search 'exchange': ");
          scanf("%d", &s_exchange);
          for(i=0; i<=entries; i++)
             if(s_exchange==exchange[i])
                printf("%s %d %d %d %s\n", name[i],
                                           area_code[i],
                                           exchange[i],
                                           extention[i],
                                           description[i]);
           break;
case '4': printf("Enter Search 'extention': ");
          scanf("%d", &s_extention);
          for(i=0; i<=entries; i++)
             if(s_extention==extention[i])
                printf("%s %d %d %d %s\n", name[i],
                                           area_code[i],
                                           exchange[i],
                                           extention[i],
                                           description[i]);
          break;
case '5': printf("Enter Search 'description': ");
          scanf("%s", s_description);
          for(i=0; i<=entries; i++)
             if(!strcmp(s_description, description[i]))
                printf("%s %d %d %d %s\n", name[i],
                                           area_code[i],
                                           exchange[i],
                                           extention[i],
                                           description[i]);
          break;
default: printf("Invalid Option.\n");
}
}
#include "stdlib.c"
```

Here's a sample data file for the program:

Brian Polk 212 555 1234 Analyst
Linda Moretti 913 555 4567 Programmer
Liza Johnson 212 555 5678 Sexy
Phil Peters 516 555 6543 HUG
Warren Smith 516 555 2345 HUG
Joe Jones 516 555 5454 Vet

1) Notice the 'switch' statement. Remember last month when we used nested 'if' statements to test multiple conditions? This was the easier way to accomplish the same thing. The variable in the 'switch' statement is used for testing. Each 'case' causes a test to be performed against this variable. If it matches, the following code is executed. In either case, execution continues at the next 'case' statement. That is why we use the 'break' statement to break out of further testing once we have found a match. You can also use the 'break' statement to break out of other structured statements, such as 'do' and 'while'. The 'default' case will be executed if no other cases are matched.

2) I set up a #define for the SCOUT variable as opposed to including the HDOS XTEXT file HOSDEF. The reason for this was mentioned above regarding the 'ORG' statement. The number 6 is the location within HDOS of the system call for the terminal set- up routine.

3) Notice the use of '%*s' in the 'scanf' statement. The asterisk means that the field exists in the input, but should be skipped during the conversion. In our example, this means that the person's last name physically exists in the data file, but it will be skipped when assigning values to the variables. Another thing to note about the 'scanf' function is that it treats blanks as field delimiters. Since a blank space will delimit a field, they don't have to start or end in any specific columns. All they have to do is have a space between them. This means that our file cannot contain imbedded blanks within the 'name' or 'description' fields. 'Scanf' will search for fields across lines, and will pick up searching from where the previous 'scanf' left off, even if it was in the middle of a line.

4) The 'strcmp' function compares two strings, returning '0' if they are the same, '-1' if string1 is less than string2, and '+1' if string1 is greater than string2.

5) Notice the use of two-dimensional arrays to hold the name and description fields. We have allowed for ten entries. This number can be expanded as you wish. We have also allowed 30 characters for each name and description field. This can also be changed very easily. But be careful. The 'scanf' function does not check to see if the string it converts can fit into the allocated character array. Whatever value you make the field length, make sure not to exceed that length in the data file. Also remember that 'C' terminates each string with the null character '\0'. This means that the maximum length for our name and description is 29 characters each.

Try these exercises with this program once you understand how it works.

1) Add a check to only read the first ten entries from the file. As the program stands now, if the input file contained more than ten entries, results are unpredictable because we will have overwritten memory outside of our array bounds. 'C' is not very particular on checking that we know what we are doing with arrays!

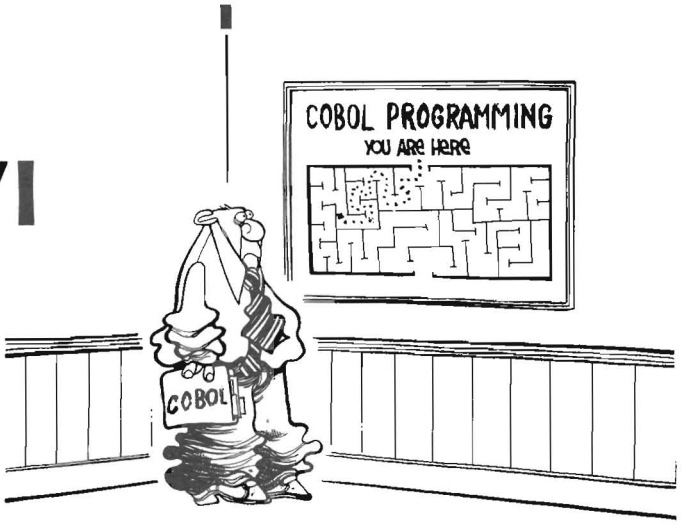
2) Allow for more than one option to be run for each execution of the program by returning to the main menu after completing one option. This means you will have to add an 'exit' option to the menu.

Next time we will see how to use structures, allocate storage, and set pointers.

'C' you later.

COBOL Corner VI

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



Introduction

Are you ready to complete the development of Sample Program #1? We were in Phase III--Step 1 and had three (3) "homework" assignments last time. Did you do them? Remember, you will have to work with "COBOL Corner", not just read it! If you need help, write to me in detail about your problems (with SASE, business size). Please, no phone calls! I want to decide whether to answer problems in writing individually or to answer the general problems in "COBOL Corner".

Procedure Division

When we write our COBOL programs the Procedure Division may have Sections, Paragraphs, and Sentences/Statements. Sections will not be required until we get into advanced, complex programs; therefore, we will discuss Sections later.

Paragraphs will be the "heart" of the Procedure Division. Each Paragraph is a procedure. Each of the Paragraphs must begin with a paragraph-name which will be User-Defined (be sure it is unique and self-documenting) and start in column 8. The paragraph will end with a period. (These will make up our Modules in our Structured COBOL.)

Sentences/Statements will make up the Paragraphs. Each of the Paragraphs will consist of a series of Sentences. Each Sentence may contain one (1) or more Statements. They will all begin in Area A or B (columns 8-72), depending on their indentation for readability and they will end with a period. (A Statement is defined as a syntactical, valid combination of words and symbols beginning with a COBOL Verb.)

COBOL Verbs

(We will start our use of Verbs and expand the list in future Sample Programs so that you will not have to learn too many at one time and will use them right away. The best way to learn these Verbs is to use them over and over many times in various ways.)

```
1-OPEN------Each file used in the program MUST be opened before any
            input or output operation involving the file is executed.

        POSSIBLE FORMATS:
            000XXX OPEN INPUT file-name-in.     (Note one period
            000XXX OPEN OUTPUT file-name-out.    for each code line)
         OR
            000XXX OPEN INPUT file-name-in      (No period on this line)
            000XXX      OUTPUT file-name-out.   (Period required)
```

I will be using the second Format, because I think it is easier to read. What do you think? Of course, each file opened must be defined in the FD clause in the Data Division and have a Select Entry in the Environment Division. Check these Divisions to see that we did this!

```
2-CLOSE-----This Verb terminates processing of the named files.

        POSSIBLE FORMATS:
            000XXX CLOSE file-name-in.        (Note one period
            000XXX CLOSE file-name-out.        for each code line)
        OR
            000XXX CLOSE file-name-in         (No period on this line)
            000XXX       file-name-out.       (Period required)
```

Again, I prefer the second Format. Also, note the file-name-in or -out for both the OPEN or CLOSE verbs must be the same as those used in the FD clause!

```
3-STOP RUN--This verb set terminates the Execution of the program.

        FORMAT:
            000XXX STOP RUN.     (Must have period)

4-READ------This verb makes a logical record from the Input Record
            available for processing.

        FORMAT:
            000XXX READ file-name-in   (No period & same file-name-in)
            000XXX     AT END          (No period & one verb per line)
            000XXX         imperative statement.   (Period required!)
```

The AT END phase is required when reading a line sequential file from disk as we are doing in our Sample Program #1. The "imperative statement" is read by the computer when END-OF-FILE condition is detected (empty file or all records have been read). In our sample program we move "YES" TO WS-END-OF-FILE-SWITCH to indicate we have completed reading the Input file to the computer.

```
5-WRITE-----This verb is used to transfer a record to an Output device
            (Printer in our program).

        POSSIBLE FORMATS:
            000XXX WRITE record-name.  (Period & Name of Output File)
        OR
            000XXX WRITE record-name    (No period)
                       AFTER ADVANCING "N" LINES.  (Period!)
                           NOTE:"N" equals whole integer--2=double space
        OR
            000XXX WRITE record-name     (No period)
                       AFTER ADVANCING PAGE.  (Period required!)
                           NOTE: Performs a FORM FEED!

6-DISPLAY---COBOL-80 Verb used to Output the program data at runtime on
            your CRT terminal.

        FORMAT:
            000XXX DISPLAY record-name.     (Note period!)

7-MOVE------This verb is used to move data from one area of computer
            storage to one or more areas within the computer memory.

        FORMAT:
                          identifier-1
            000XXX MOVE       or        TO identifier-2.  (Period!)
                          literal
```

```
        identifier-1——is the sending field.
        identifier-2——is the receiving field.
        literal————————actual value specified in the program.
```

Note: The sending field remains in the sending field. For "Good", readable, structured COBOL programming we will seldom MOVE one sending field to more than one receiving field. We will use one or more MOVE Statements to do this.

```
8-PERFORM——This verb is a branching instruction.  It temporarily
           transfers control of the program from the sequence of
           procedure execution in one MODULE to the specific User-Named
           PARAGRAPH (MODULE) in our structured COBOL.

        POSSIBLE FORMAT:
            000XXX PERFORM paragraph-name.      (Period)
```

Once the paragraph named has been executed, control returns to the Statement immediately following the PERFORM Statement.

```
        OR
            000XXX PERFORM paragraph-name       (No period)
                       UNTIL condition.         (Now the period)
```

This Statement causes an evaluation of a specified "condition" to occur! The paragraph named will be performed over and over until the "condition" is satisfied. Control will then return as above.

COBOL Figurative Constants

Figurative constants are reserved words that have predefined values in COBOL. We start using figurative constants in this Sample Program and we use them in many ways in future programs. You MUST get to know them! They are a special type of literal and they ARE NOT bounded by quotation marks! We will divide them into two (2) groups:

Numeric Literal:

ZERO or ZEROS--Places 0 or 0's in the specified fields.

Non-Numeric Literal:

SPACE or SPACES--Puts BLANK characters ("octal" 40) in the specified fields.
LOW-VALUE/s--Character with "octal" 00 representation.
HIGH-VALUE/s--Character with "octal" 177 representation.
ALL "literal"--One or more instances of the "literal", which must be a non-numeric literal or a figurative constant, in which ALL is redundant, but we will always use it.

A figurative constant may be used anywhere a literal is called for in the program format; except, when the literal is restricted to being NUMERIC (PIC 9(0n)). The only NUMERIC figurative constant permitted by COBOL-80 is ZERO!

Rules For Literals (Literal is a constant not identified by a Data-Name.)

Numeric Literals--Value implicit in the characters themselves.
Example:- 1983 is a literal and a value.
Rules:
1) Composed of digits 0-9, + (plus sign), - (minus sign) and . (decimal point) only.
2) May contain 1-18 digits.
3) Can have only one (1) sign as left most character.
4) Only one (1) decimal point per literal and it can be anywhere except at right most character.
5) Cannot be enclosed in quotation marks.

Non-Numeric Literals--Cannot be used in calculations!
Rules:
1) Must be enclosed in quotation marks!
2) Cannot be used for computations!
3) May contain any character in the COBOL character set.
4) The beginning and ending quotation marks are not counted as part of the length of the literal.

Developing the Program

We have now reviewed your "HOMEWORK". Please have these COBOL Verbs, Figurative Constants, and Literals well in mind as we will now start using them.

Put your Hierarchy Chart and Flowchart for this Sample Program #1 from previous articles in front of your work area so that you can easily refer to them. I want you to match the following coding lines for the Procedure Division with these program tools so that you see how they help you write this division!

Developing Sample Program #1 (continued)

(**Note:** The first line of numbers is not part of the program!)

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
000760 PROCEDURE DIVISION.
000770*
000780*
000790 MAIN-CUSTOMER-LIST.
000800*
000810 OPEN INPUT FILE1
000820      OUTPUT CUSTOMER-LIST-LINE.
000830 PERFORM INITIALIZE-VARIABLE-FIELDS.
000840 READ FILE1
000850     AT END
000860         MOVE "YES"                TO WS-END-OF-FILE-SWITCH.
000870 PERFORM PROCESS-CUSTOMER-LIST
000880     UNTIL WS-END-OF-FILE-SWITCH IS EQUAL TO "YES".
000890 CLOSE FILE1
000900       CUSTOMER-LIST-LINE.
000910 STOP RUN.
000920*
000930*
000940 INITIALIZE-VARIABLE-FIELDS.
000950*
000960 MOVE "NO "                        TO WS-END-OF-FILE-SWITCH.
000965*
000970*
000980 PROCESS-CUSTOMER-LIST.
000990*
001000 MOVE SPACES                       TO CL-CUSTOMER-LIST-LINE.
001010 MOVE CR-CUSTOMER-ACCT-NO          TO CL-CUSTOMER-ACCT-NO.
001020 MOVE CR-CUSTOMER-NAME             TO CL-CUSTOMER-NAME.
001030 MOVE CR-CUSTOMER-ADDRESS          TO CL-CUSTOMER-ADDRESS.
001040 MOVE CR-CUSTOMER-CITY             TO CL-CUSTOMER-CITY.
001050 MOVE CR-CUSTOMER-STATE            TO CL-CUSTOMER-STATE.
001060 MOVE CR-CUSTOMER-ZIP              TO CL-CUSTOMER-ZIP.
001070 WRITE CL-CUSTOMER-LIST-LINE
001080     AFTER ADVANCING 2 LINES.
001090 DISPLAY CL-CUSTOMER-LIST-LINE.
001100 READ FILE1
001110     AT END
001120         MOVE "YES"                TO WS-END-OF-FILE-SWITCH.
```

Procedure Division Review

Look at our new COBOL code lines and see if you can find the COBOL Verbs, Figurative Constants, and Literals. Can you find examples of each? Did they fit the Format for each one? Do they meet the rules we specified? Note how we used the self- documenting User-Defined Names with their prefix to help ourself follow the sending fields and receiving fields in the many Move Statements. Did our file-name-in and file-name-out names make it easy to know which was which? Can you see how the Working- Storage-Section Switch does its job? Did our Hierarchy Chart Modules check out with the Flowchart and the Procedure Division paragraphs? Working with these program tools is the only way to program! You do not sit down at the terminal and try to Key-In the program!

Now, walk-through the Logic of the program comparing it with the Program Specification. Does it do what the program should do? Will it READ our Input Record as described on the Record Chart from the disk? Will it WRITE the Output to our Printer per our Print Chart? Until you know that the program will do all of the above, you are not ready to even think about keying the Procedure Division! If you have

errors, we want to find them on our Coding Forms! When you are sure, you are ready to continue.

Phase III--Step 3 Keying the Procedure Division

Review your "homework" Project 1. Put Disk A in Drive A and using your Editor, Key-In the Procedure Division exactly as you have put the code lines on your coding forms and "APPEND" this entry to your PRGM01.COB disk file and save. Be careful to use the right Area and Columns, put in the hyphens & periods only in required places, and watch for typos & spelling errors!

Phase III--Step 4 Compile Your Source Code

Now, as we have done before with SQUARO.COB, we will compile our Sample Program #1. Do you remember what to do? We will go over it again as a refresher. With Disk A still in Drive A, put Disk B (compiler disk) in Drive B. With the DIR or STAT *.* check both Drive A and Drive B disks to see that the correct disks are present. Drive A must have FILEL1.DAT and PRGM01.COB. Drive B must have the COBOL compiler files we have described in earlier "COBOL Corner" articles. If this "checks" out, Type B: and Return. Next, Type the following command line:

```
B> COBOL A:PRGM01,LST:=A:PRGM01  and  Return
```

You will obtain a listing of the program and hopefully a "Clean Run"--Compiler Message "NO ERRORS OR WARNINGS"! If you do have errors, the compiler will provide you with "error messages" telling you which compiler line to start looking for your error and possibly what name, etc. to look at. A careful review of your listing to your Code Forms and this article's Coding Lines should enable you to find your errors. Correct your errors on the Listing with a red pencil! Go back to Drive A and your Editor and make the necessary corrections to your file A:PRGM01.COB Disk File, SAVE the corrected file with the same name, and compile again. If you are careful, you should never get compiler errors a second time! If you do, you did not perform your Code walk-through carefully!

When you complete your compilation you should find A:PRGM01.REL on Disk A!

Phase IV Link and Execute Sample Program #1

If you are using two (2) drives, replace Disk B in Drive B with Disk C. If you have three (3) drives, Disk C should be in Drive C. If you have doubts of how to Execute your program, refer back to the previous "COBOL Corner" articles. Type B: or C: and Return. Now Type the following command line:

```
B> or C> L80 A:PRGM01/G/E  and  Return
```

You should obtain a Print-Out that should match your Print Chart Format! If you do not, recheck your listing with your Print Chart and your Flowchart. Also, you can check your listing with the "COBOL Corner" code lines for your errors. You might find a Runtime Error, with error messages. If you get such errors, refer to your COBOL-80 Manual and look up Runtime Error Messages. Make the necessary correction to your files and compile your program again. Then, Execute the program again.

Hopefully, you obtained a Print-Out that did match your specifications! If your format is not correct with the Print Chart, the error will most likely be one of Program Logic or Field Size. If you have Keyed-In the Code Lines and the Transaction File (FILEL1.DAT) correctly, you will obtain correct results!

Closing

Your HUG COBOL Corner Disk-I has the file PRGM01.COB on it. If you are still having problems, put this file and the FILEL1.DAT from the HUG Disk on a "NEW" Disk A (not the Disk A you have been working with). Compile and Execute the program with this disk, as described above. You should obtain a Print-Out that will match your specifications. Compare the listing from this run with the one you obtained from your Keyed-In program Listing line by line. You should now be able to find your errors.

We have covered a lot of "ground" that might be new to you. If I have not made it clear, and your COBOL-80 Manual, plus working with the program, does not make it clear, write to me about your problems in detail (enclose a SASE, business size) and I will attempt to clear things up!

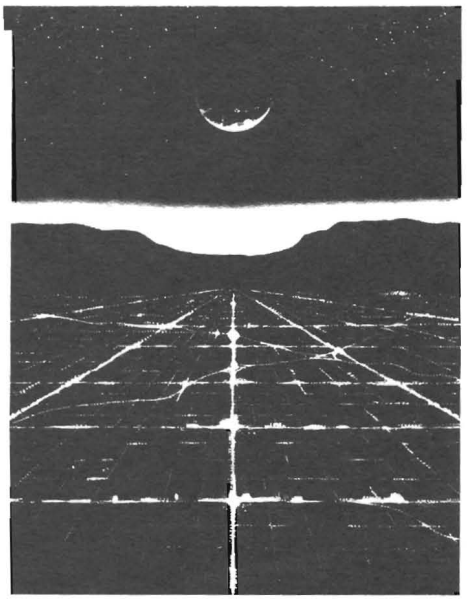
Next, "COBOL Corner" will vary this Sample Program #1 to add some refinements. Starting with this Sample Program #2, I will want you to do more of the program development yourself! I will not provide all the Phases of program development. I will supply the program specification and some explanation about the new COBOL programming required. Therefore, be sure you have understood everything up to this point!

Vectored from 53

```
500 IF NY>80 OR NY<1 THEN 570
510 IF FNLOOK(NX,NY)<>32 THEN 570
520 IF NY=1 THEN D=127
530 Z=FNMAP$(X,Y,32)
540 X=NX:Y=NY
550 Z=FNMAP$(X,Y,D)
560 IF D=127 AND Y=80 THEN 590
570 N=N+1:PRINT CHR$(27);"Y6a";N
580 GOTO 470
590 INPUT
    "WOULD YOU LIKE ANOTHER RUN ON THE COURSE (Y\N)";Z
600 IF Z="Y" THEN 40
```

Pseudo Memory Mapped Video

Another Assembly Language Subroutine For MBASIC Under HDOS



Fred Hochschild
733 Brunswick Avenue
Trenton, NJ 08638

In my last article, I gave an introduction to the principles needed to use assembly language subroutines with MBASIC under HDOS. If you haven't read that article, I suggest you do so before continuing with this one (see the Sept. '83 REMark, "Learning To Write Assembly Language Subroutines For MBASIC Under HDOS").

In this article I will present a more elaborate subroutine for simulating memory mapped video, give some ideas on convenient ways to interface the subroutine to MBASIC, and finally make combined use of this subroutine and the one from my previous article in a sample program.

First let me give an explanation of memory mapped video for those that are not familiar with it. Many of the personal computers on the market today have a section of RAM that is used to set up a one-to-one correspondence with the character locations in the video screen. The ASCII, or similar, code for each character on the screen is placed in the corresponding memory location. In fact, the screen is refreshed from these RAM locations. One can then place any character anywhere on the video screen just by POKEing the proper code into the appropriate memory locations. Of course, determining what is at a particular location on the screen is then just a matter of PEEKing the correct RAM address.

The routine as written will perform three separate, but related functions. The first is to clear the screen (CLS) and set the memory map RAM to ASCII spaces (code 32). Second, it will put (MAP) a specified character on the screen and its code in the corresponding RAM location. The third function is to LOOK at a specified RAM address to determine the character at the corresponding screen location. The routine determines which function is to be performed by examining the variable passed to it from MBASIC. If a number is passed, it will do the clear screen function (CLS). If a string is passed, the function is determined by the length of the string. A two byte string will have the routine LOOK up a particular location for a character code, and a three byte string will cause a character to be MAPped to the screen and its code to be placed in the appropriate RAM location. How this is done will be detailed later.

Before going into the details of the routine, a few words need to be said about the character codes used. The code for the standard, printable ASCII characters is used as is (e.g. 32-126). To accommodate the graphics characters and reverse video characters, the codes sent from MBASIC to the subroutine have been adjusted. The codes 127-159 will be used to represent the graphics characters in order. Codes 160-254 represent ASCII characters in reverse video, again in order. Finally, codes 255- 031 (MOD 256) represent the graphics characters in reverse video. You should refer to the reference card supplied with the H/Z-19 terminal to find the order of the characters.

The screen is broken into a coordinate grid of lines and columns. The upper left hand corner is line one, column one. There are twenty-four lines (the twenty-fifth line cannot be accessed by this subroutine), and eighty columns.

Listing 1 contains the source code for the routine. I would advise taking a few minutes to look over the headings for each section of the subroutine. This will give you some idea of where we will be going and how we will get there. I will discuss the routine by sections. I will assume that you have some familiarity with assembly language programming and will therefore only discuss the highlights and/or intricacies of each section.

The ORiGin for this routine is in low RAM between the H-8 and H-17 ROMs. If you do not have RAM at these addresses, then you will need to calculate an ORG in upper RAM just below HDOS. Also you will need to reserve enough memory from MBASIC to accommodate this routine (at least two and a quarter K). I suggest that you ORG this routine so it will not overlap with the INKEY routine from the previous article. This will allow you to use both routines in the demo program to come later. Refer to the previous article to see how to calculate the ORG address and provide memory space for it when MBASIC is loaded.

The PRELOAD section is not a direct working part of the routine, but is used to allow easy loading of the routine before loading MBASIC. This is the same technique used in the previous article.

The MAINLINE section handles supervision of the working sections of the subroutine. By examining register A, it determines which operation is to be performed and calls the appropriate subsection. A '3' in register A indicates a string has been passed from MBASIC. For MAPping characters, a three byte long string is passed consisting of the line number on the screen, the column number, and the character code, respectively. The LOOK up function passes a two byte string consisting of line and column numbers only. If a nonstring is passed to the routine, it will perform the clear screen function.

The operations of clearing the console screen and the memory map

area take place in the CLEAR section. MAPAREA is a 2000 byte area of memory reserved for the codes of the characters mapped onto the screen. The loops are self-explanatory. The INTCUR routine resets the HDOS cursor counter to one, avoiding unwanted carriage returns being inserted by HDOS during the transmission of the clear screen command sequence to the console. The SCALL.PRINT is an HDOS routine for transmitting sequences of character codes to the console.

For either the MAPping or the LOOKing function, the line number and the column number need to be placed in the DE register pair. This task is accomplished by the LOADDE routine. At the entry to this routine, the DE register pair points to the string descriptor that in turn points to the string that contains the line number and column number. The length of the string, which is used to distinguish between MAPping and LOOKing, is moved into register A.

The LOOK section retrieves the character code for the character on the screen at the location specified by the line and column numbers that are now in register pair DE. The MAPADDR call places the memory address corresponding to the line and column number in the HL register pair. The two byte string that was passed from MBASIC is used to pass back a two byte string that can be converted (CVI) to an integer (the character code). In order for the CVI function to operate properly, a zero must be placed in the higher address byte.

MAPRTN is a supervisor section that organizes the necessary steps to get a character in the proper location on the screen and its code into the memory map.

The MAPADDR calculates the displacement from the beginning of the memory map that corresponds to the line and column number. The displacement is given by the column number minus one, plus the line number, minus one, times eighty (e.g. {line-1}*80+col-1).

For a character to be placed on the screen in the proper location, the direct cursor addressing mode must be activated and the cursor located by transmitting the line+31 and column+31. Next there must be a series of codes that activate reverse video and/or graphics mode if appropriate. Finally the character code must be transmitted and reverse video and graphics disabled. These preparations and decisions are handled by the PRTLINE section. First the line and column numbers are adjusted by 31. The ESCape sequences that enable reverse video and graphics modes are disabled by changing the code to an ESC 87, which has no effect on the console. What follows this is a series of decisions based on the code numbers discussed earlier that will re-enable the reverse video and/or graphics mode if necessary. The entire sequence of codes that is sent to the console begins at PRTOUT and is twenty-three bytes long. In addition to what has already been discussed, this sequence also turns the cursor off during MAPping and returns it to its original position.

The PRTRTN section handles the actual transmission of the code sequence and protects the HDOS cursor counter. The SCALL.CONSL is an HDOS routine that can be used to examine and/or set the HDOS cursor counter. Care must be taken when transmitting sequences of nonprintable codes to the console that HDOS does not insert unwanted carriage returns in the middle of the transmission. Therefore, the cursor counter is saved, set to one, and then restored to its original value. Setting the counter to one is accomplished by the INTCUR section.

Listing 2 contains a set of MBASIC lines for interfacing the assembly language subroutines to an MBASIC program. Lines four and five set the location of the INKEY subroutine from my last article and define a function (FNKEY$) to be used in calling this subroutine. I feel it is very useful to use mnemonics for the functions that access the subroutines. Lines six through nine give the location of the subroutine in this article and three defined functions that are used when executing the three operations this subroutine was designed to accomplish.

To execute the INKEY function, use the MBASIC statement X$=FNKEY$. The clear screen function requires the command X=FNCLS. To LOOK at a memory map location, use X=FNLOOK (line, column). MAPping can be accomplished with X$=FNMAP$ (line, column, character code).

The USR addresses in Listing 2 are for low RAM. If you are going to place these subroutines in high RAM, you must change the USR definition statements to correspond with the ORG statements in the subroutines. Remember to ORG the two subroutines so they will not overlap in memory and to define the USR address as three bytes past the ORG address.

By entering Listing 2 and saving it in ASCII form (i.e. with a ,A after the file name in the SAVE command), you can merge it into any program which has a need for these functions. A note of caution: if your MBASIC program contains DEFinition statements for variable types, they should come before the function definitions. Otherwise the function definitions may be overridden. This is why Listing 2 starts at line four.

Listing 3 contains a sample use of the two subroutines. It is a simple game of running a maze against time. The motion of the character on the screen is controlled by the keys on the numeric keypad. The even numbered keys correspond to motion in the directions of the arrows on those keys. The odd numbered keys cause diagonal motion between the neighboring arrows. The moving character starts as an asterisk. You must move it from the right side of the maze to the left, where it will turn into a dot. It must then be moved back through the maze to the starting position.

This concludes my little excursion into assembly language subroutines for MBASIC. I hope it has been of some use to other beginners out there.

Listing 1

```
        TITLE   'PSEUDO MEMORY MAPPED VIDEO'
        XTEXT   HDOS.ACM
MMAP    ORG     0800H
*****   PRELOAD
*       Returns immediately to HDOS. This allows
*       loading of MMAP without execution. Also,
*       low RAM is safe from MBASIC loadin.
*       Use DEFUSR=&H0803 in MBASIC.
*
*
        XRA     A
        SCALL   .EXIT
        EJECT

*****   PSEUDO MEMORY MAPPED VIDEO-MAINLINE
*
*       ENTER   See articles, also MBASIC manual
*
*       EXIT    CLEAR-I.CUSOR=001Q
*               LOOK-Returns character code
*               MAP-I.CUSOR unchanged
*
*       USES    ALL
*
        CPI     3           String?
        JNE     CLR           no
        CALL    LOADDE        yes
        CPI     3           Mapping?
        JNE     LOOKUP        no
        INX     H             yes
        MOV     C,M         Char. code to reg. 'C'
        CALL    MAPRTN
```

```
        RET
CLR     CALL    CLEAR
        RET
LOOKUP  CALL    LOOK
        RET
        EJECT

*****   CLEAR   Clears memory map area(2000 bytes)
*               to all spaces and sends clear
*               screen code to console.
*
*       USES    A,B,C,H,L
*
CLEAR   LXI     H,MAPAREA
        MVI     B,8AH       Initialize
        MVI     C,08H        loops
AGAIN   MVI     M,' '
        INX     H           Loops to
        DCR     B             put spaces
        JNZ     AGAIN           into
       MVI     B,0FFH            memory
        DCR     C                    map area
        JNZ     AGAIN
        CALL    INTCUR      Clear cursor counter
        LXI     H,CLS
        SCALL   .PRINT      Send ESCape code to console
        CALL    INTCUR      Clear cursor counter
        RET
*
CLS     DB      27,69+200Q   Clear console screen
        EJECT

*****   LOADDE  Loads register D with line number
*               Loads register E with column number
*
*       ENTER   (DE) points to string descripter
*
*       EXIT    A=lenth of string(2 or 3)
*               D=line number
*               E=column number
*               (HL) points to second byte of string
*
*       USES    A,D,E,H,L
*
LOADDE  XCHG                (HL)=string discripter
        MOV     A,M         Load string lenth into reg A
        INX     H           (HL)=low order string address
        MOV     E,M         Load into reg E
        INX     H           (HL)=high order string address
        MOV     D,M         Load into reg D
        XCHG                (HL)=string
        MOV     D,M         Column # to reg D
        INX     H
        MOV     E,M         Line # to reg E
        RET
        EJECT

*****   LOOK    Looks up character code in memory
*               map at given location
*
*       ENTER   D=line number
*               E=column number
*               HL=(address for returned character
*                   code)+1
*
*       EXIT    (HL) points to two byte string with
*                    character code in first byte and
*                    zero in second byte
*
*       USES    A,B,D,E,H,L
*
LOOK    PUSH    H           Save (HL)
        CALL    MAPADDR
        MOV     A,M         Char code to reg A
        POP     H           Restore (HL)
       MVI     M,0         Prepare
        DCX     H             returned
        MOV     M,A             string
        RET
        EJECT

*****   MPTRTN  Maproutine-puts character on screen
*                           and character code to map
*
*       ENTER   C=character code
*               D=line number
*               E=column number
*
*       USES    ALL
*
MAPRTN  CALL    MAPADDR
        MOV     M,C         Char code to memory map
        CALL    PRTLINE     Prepare output
        CALL    PRTRTN        print it
        RET
        EJECT

*****   MAPADDR calculates memory map address from
*               line number and column number
*
*       ENTER   D=line number
*               E=colunm number
*
*       EXIT    (HL)=address within memory map
*               D,E unchanged
*
*       USES    A,B,D,E,H,L
*
MAPADDR LXI     H,MAPAREA
        MOV     B,D         Line # to reg B
        DCR     B           Zero base displacement
        MVI     A,80        Add eighty
EIGHTYS DCR     B             for each
        JZ      ONES            line
        CALL    $DADA       (H,L)=(H,L)+(0,A)
        JMP     EIGHTYS
ONES    MOV     A,E         Reg E to reg A
        DCR     A           Zero base displacement
        CALL    $DADA       (H,L)=(H,L)+(0,A)
        RET
*
MAPAREA DS      2000        Memory reserved for
*                           video map
        EJECT

*****   PRTLINE Prepares the line of output
*
*       ENTER   C=character code
*               D=line number
*               E=column number
*
*       USES    A,C,D,E,H,L
*
PRTLINE LXI     H,OUTPUT
        MOV     A,D
        ADI     31          Line#+31
        MOV     M,A           to output line
        INX     H
        MOV     A,E
        ADI     31            Column#+31
        MOV     M,A             to output line
        LXI     H,GRAPHIC+1   Disable graphics
        MVI     M,87            and reverse
        LXI     H,REVIDEO+1       video ESCape
        MVI     M,87                codes
        LXI     H,OUTPUT+2
        MVI     A,31
        CMP     C           Reg C > 31 ?
        CNC     SETG          no
        CNC     SETRV         no
        MOV     M,C
        RNC                   no
        MVI     A,126          yes
        CMP     C           Reg C > 126 ?
        MOV     M,C
        RNC                   no
        MVI     A,159          yes
        CMP     C           Reg C > 159 ?
```

```
        CNC     SETG            no
        MOV     M,C
        RNC                     no
        MVI     A,254           yes
        CMP     C              Reg C > 254 ?
        CNC     SETRV           no
        MOV     M,C
        RNC                     no
        CALL    SETG            yes
        CALL    SETRV           yes
        MOV     M,C
        RET
*
SETG    XCHG                    Save (HL)
        LXI     H,GRAPHIC+1     Graphics ESCape
        MVI     M,70              code to ouput line
        MOV     A,C             Adjust
        ADI     223               character
        MOV     C,A                 code
        XCHG                    Restore (HL)
        STC                     Clear
        CMC                       carry
        RET
*
SETRV   XCHG                    Save (HL)
        LXI     H,REVIDEO+1     Reverse video ESCape
        MVI     M,112             code to output line
       MOV     A,C             Adjust
        ADI     128               character
        MOV     C,A                 code
        XCHG                    Restore (HL)
        STC                     Clear
        CMC                       carry
        RET
*
PRTOUT  DB      27,106          ESC j  save cursor
*                                      position
        DB      27,120,53       ESCx5  turn cursor off
GRAPHIC DB      27,70           ESC F  enter graphic
*                                      mode
REVIDEO DB      27,112          ESC p  enter reverse
*                                      video mode
        DB      27,89           ESC Y  direct cursor
*                                      addressing
OUTPUT  DS      3               line,column,character
        DB      27,113          ESC q  exit reverse
*                                      video mode
        DB      27,71           ESC G  exit graphics
*                                      mode
        DB      27,107          ESC k  return cursor
*                                      to saved position
        DB      27,121,53+200Q  ESCy5  turn cursor on
        EJECT

*****   PRTRTN  Print routine- prints output line
*                              keeps I.CUSOR unchanged
*
*       USES    A,B,C,H,L
*
PRTRTN  MVI     A,I.CUSOR       Get HDOS
        MVI     C,00H             cursor
        SCALL   .CONSL              counter
        LXI     H,CUSAVE        Save
        MOV     M,A               counter
        CALL    INTCUR          Reset cursor counter
        LXI     H,PRTOUT        Print output
        SCALL   .PRINT            line
        MVI     A,I.CUSOR
        LXI     H,CUSAVE        Restore HDOS
        MOV     B,M               cursor
        MVI     C,0FFH              counter
        SCALL   .CONSL
        RET
*
CUSAVE  DS      1               Cursor counter save
*                               area
        EJECT

*****   INTCUR  Initialize I.CUSOR to 001Q
*
*       USES    A,B,C
*
INTCUR MVI      A,I.CUSOR       Set HDOS
        MVI     B,01H             cursor
        MVI     C,0FFH              counter
        SCALL   .CONSL                to 001Q
        RET
        EJECT

        END     MMAP
```

Listing 2

```
4 DEFUSR1=&H1103
5 DEF FNKEY$=USR1("A"+"")
6 DEFUSR0=&H0803
7 DEF FNCLS=USR0(0)
8 DEF FNLOOK(I,J)=CVI(USR0(CHR$(I)+CHR$(J)))
9 DEF FNMAP$(I,J,C)=USR0(CHR$(I)+CHR$(J)+CHR$(C))
```

Listing 3

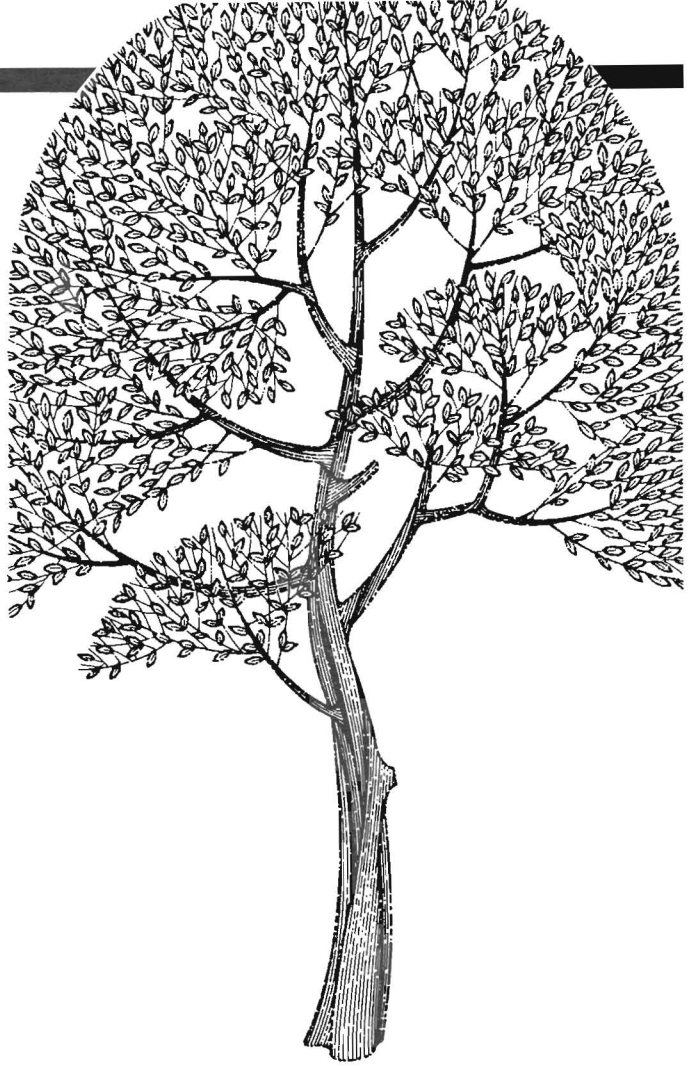
```
4 DEFUSR1=&H1103
5 DEF FNKEY$=USR1("D"+"")
6 DEFUSR0=&H0803
7 DEF FNCLS=USR0(0)
8 DEF FNLOOK(I,J)=CVI(USR0(CHR$(I)+CHR$(J)))
9 DEF FNMAP$(I,J,C)=USR0(CHR$(I)+CHR$(J)+CHR$(C))
10 DEFINT D-K,M-Y
20 DIM O(20,10)
30 DEFSTR Z
40 X=FNCLS
50 '
60 '     DRAW BORDER
70 FOR I=2 TO 79
80 Z=FNMAP$(1,I,17):Z=FNMAP$(20,I,145)
90 IF I>19 THEN 120
100 Z=FNMAP$(I,2,160):Z=FNMAP$(I,79,160)
110 NEXT I
120 Z=FNMAP$(10,2,32):Z=FNMAP$(10,79,32)
130 Z=FNMAP$(9,1,17):Z=FNMAP$(9,80,17):
    Z=FNMAP$(11,1,145):Z=FNMAP$(11,80,145)
140 PRINT CHR$(27);"Y5?";"OBSTICLE COURSE"
150 '
160 '     GENERATE OBSTICLES
170 FOR I=2 TO 19
180 FOR J=1 TO 10
190 O(I,J)=138
200 NEXT J,I
210 FOR I=1 TO 10
220 O(INT(RND(1)*18)+2,I)=32
230 NEXT I
240 '
250 '     DRAW OBSTICLES
260 FOR I=2 TO 19
270 FOR J=9 TO 78 STEP 7
280 Z=FNMAP$(I,J,O(I,(J-2)\7))
290 NEXT J,I
300 '
310 '     SET MOTION CODES
320 E(1)=1:E(2)=1:E(3)=1:E(4)=0:E(5)=0:
    E(6)=0:E(7)=-1:E(8)=-1:E(9)=-1
330 F(1)=-1:F(4)=-1:F(7)=-1:F(2)=0:F(5)=0:
    F(8)=0:F(3)=1:F(6)=1:F(9)=1
340 '
350 '     START
360 X=10:Y=80
370 Z=FNMAP$(X,Y,42)
380 PRINT"ON YOUR MARK";
390 FOR I=1 TO 450:NEXT I
400 PRINT"    GET SET";
410 FOR I=1 TO 450:NEXT I
420 PRINT"    GO!!";
430 PRINT TAB(60);"TIME:"
440 D=42:N=0
450 '
460 '     RUNNING LOOP
470 Z=FNKEY$
480 IF Z>"9" OR Z<"1" THEN 570
490 NX=X+E(VAL(Z)):NY=Y+F(VAL(Z))
```

Vectored to 49 ☞

Introduction To Data Structures

Trees

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



Trees are particularly useful and versatile data structures. You are probably already familiar with some tree structures used in everyday life. Your family tree can be viewed as a tree structure in which the names of your ancestors are stored. The diagrams used to show the progress of the teams in a sports tournament is another familiar sort of tree structure. Tree structures are natural extensions of some of the ideas presented in the first article in this series. That article introduced the notion of data structures as methods of storing data so that a needed item can be retrieved quickly. Stacks, queues, and linked lists, which are forms of "linear" data structures, were discussed in that article. Those structures are called linear structures because the data is stored in some sort of linear sequence, and no branching is possible. Like linked lists, trees consist of nodes containing several fields, including key and link fields (see figure 1). However, unlike linked lists, trees are branched, nonlinear, structures. In fact, they are called "trees" because, like botanical trees, they have branches. Certain parts of trees are also given arboreal names. One node is called the "root", and certain other nodes are called "leaves". However, unlike botanical trees, tree structures are usually depicted as growing downward, i.e. with the "root" at the top and the "leaves" at the bottom.

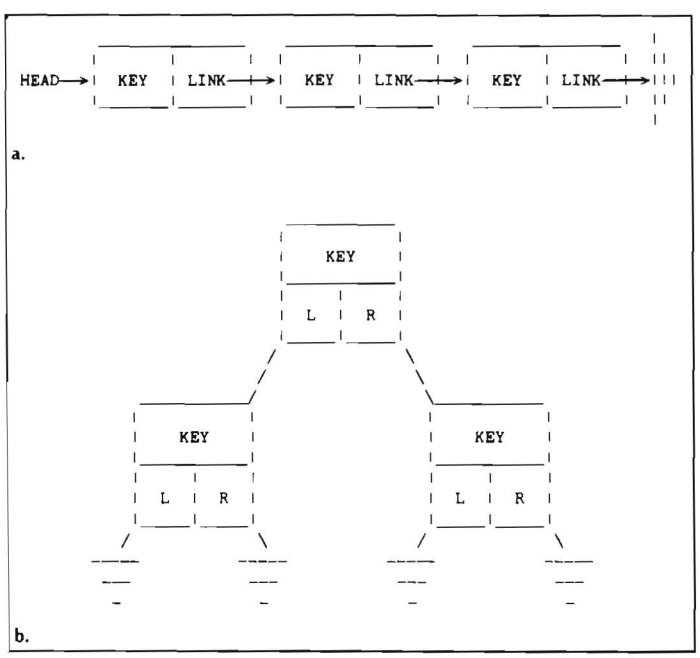


Figure 1. a. Diagram of fields in a linked list. **b.** Diagram of fields in a tree.

Trees are often defined as a finite set of nodes such that there is one node called the "root" and the remaining nodes can be divided into disjoint sets called "subtrees" and the subtrees are also trees. Since the sets are disjoint, if any two subtrees have the same root, they cannot have any nodes in common. This also implies that a path formed by the links in a tree structure may not form a cycle. This definition of trees may seem a little confusing since trees are defined in terms of themselves (i.e. they are defined "recursively"), but some examples should make things clearer. A tree is depicted in figure 2a. and an example of a structure that is NOT a tree is shown in figure 2b. Trees may or may not have an order assigned to their branches. If the branches are assigned a numerical order, (1,2,3...) from left to right, then the tree is an "ordered" tree.

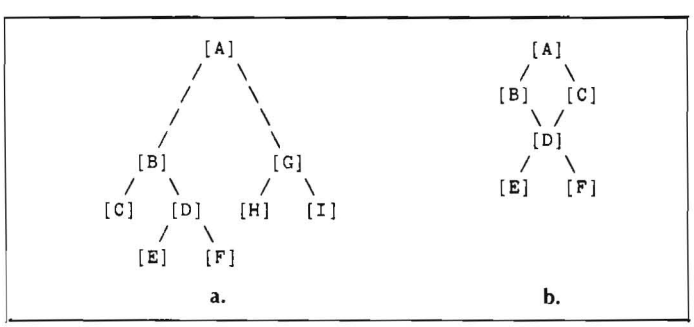


Figure 2. a. A tree. **b.** A structure that is NOT a tree.

There are many terms that are commonly used in the description of tree structures. Some of these terms are words borrowed from botany, genealogy, or graph theory. The "degree" of a node is the number of subtrees linked to that node. In figure 2a., all of the nodes are of degree two or zero. Nodes of degree zero are called "leaves" or "terminal nodes". Leaves are not linked to any other nodes; instead they are linked to "null". A null link is depicted in figure 1 by three lines (long, medium, short) like the symbol for an electrical ground.

If the degree of a node is greater than one, then it is a "nonterminal", "interior", or "branch" node. The roots of the subtrees of a node are the "children" or "descendents" of that node (current non-sexist terminology; some people refer to these nodes as "sons"). A node is the "parent" (formerly "father") of its children, and nodes that have the same parent are "siblings" or "brothers". The nodes that lie on a path from the root to the parent of a node are the "ancestors" of that node.

The "level" of a node is 1 + the level of that node's parent, and the level of the root is defined to be one (note: some authors define "level" differently, Knuth (3), defines the level of the root to be zero; Aho et al. (1), use yet another definition for "level"). The "depth" or "height" of the tree is usually defined as the level of the node that has the greatest level.

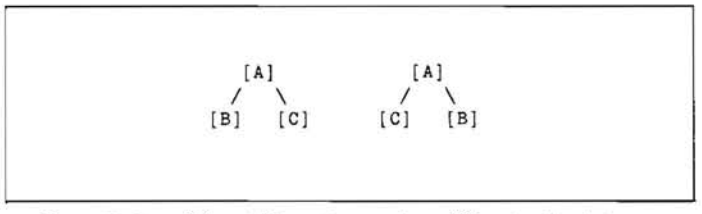


Figure 3. Two different binary trees or two different ordered trees.

A binary tree is a particularly important type of data structure. A binary tree is a set of nodes such that the set is either empty (i.e. contains zero nodes) or is made up of a root and two disjoint binary subtrees (left and right subtrees), which are also binary trees (another recursive definition!). In a binary tree, no node has more than two children, a left child and a right child. Binary trees may appear to be just a special form of tree, but binary trees differ from ordinary trees in two important respects. First, binary trees may be empty, i.e. contain no nodes, but a tree, by definition, contains one node, the root. Also, the branches of a binary tree are always ordered. If the structures shown in figure 3 are considered to be ordinary, unordered trees, then the two trees are actually the same tree, since the order of the branches makes no difference. However, if they are considered binary trees, they are different trees, since the left and right subtrees of the root are different. Furthermore, if a node of a binary tree is linked to only one descendent node, this node must be either a left child or a right child, with an empty subtree on the opposite side. In this way, binary trees differ from ordinary ordered trees. In figure 4, the two trees are different if they are considered binary trees, but they are the same if they are considered to be ordinary ordered trees. The distinction between ordinary trees and binary trees may be clearer if you think about two examples from genealogy. A family tree for storing information about you and your direct ancestors (parents, grandparents, great-grandparents,...) can be placed in the form of a binary tree if, say, the left links always point to males, the right links always point to females, and you are at the root. A tree depicting all the direct descendents of one individual could be placed in the form of an ordinary tree. A binary tree would not be appropriate in this case since some individuals are likely to have more than two children. Also, if an individual is an only child, there would be no reason to use a right or left link; just a link is sufficient. If all the siblings in a group are ordered according to age (e.g. eldest on the left), the tree would be an ordered tree. In summary, although binary trees resemble ordinary trees, by definition they are different structures with special properties.

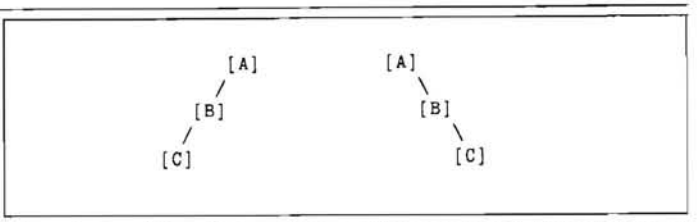


Figure 4. Two different binary trees or different representations of the same ordered tree.

Binary trees are usually stored in memory using a linked representation. Each node has a key field, a LEFT field, and a RIGHT field, where the LEFT and RIGHT fields store pointers to the left child and the right child respectively. In BASIC, one would dimension three arrays, KEY(N), LEFT(N), and RIGHT(N), where N is the maximum number of nodes one expects to need. If the left subtree of node I is empty then LEFT(I) = 0, i.e., its left link is null. If the right subtree of I has J as its root, then RIGHT(N) = J. We also need a pointer to the root, so if node I is the root, then we set ROOT equal to I.

Once a binary tree is set up, there must be some way of finding the information that is stored in the tree. We need a procedure that will let us start at the root, then examine, or "visit", each node of the tree exactly once before halting. Such a procedure is called a "traversal" algorithm. There are three standard traversal algorithms: preorder, inorder (symmetric order), and postorder. They differ in the order in which the root and subtrees are visited, see below.

Preorder	**Inorder**	**Postorder**
Vist root	Traverse left subtree	Traverse left subtree
Traverse left subtree	Visit root	Traverse right subtree
Traverse right subtree	Traverse right subtree	Visit root

We "traverse" an empty subtree by doing nothing. You may have noticed that these procedures are, like trees and binary trees, defined recursively. The amount and complexity of the code needed to implement these algorithms depends on whether or not the language being used allows recursion, i.e. allows a subroutine to call itself, directly or indirectly. PASCAL allows recursion, FORTRAN does not, and some forms of BASIC allow some recursion while others do not. However, in PASCAL, when a procedure (i.e. subroutine) calls itself, it saves the current value of certain (local) variables, and these values are restored when the subroutine ends. It is this aspect of PASCAL that makes recursion so useful in that language. This sort of feature does not exist in BASIC. Therefore, even if BASIC does allow a subroutine to call itself, most recursive algorithms will not work in BASIC without extensive modification. In BASIC, it's best to avoid recursion even if the interpreter or compiler does let a subroutine call itself.

So, if we can't use recursion in BASIC, how do we traverse trees? The answer is in the use of a stack, a data structure discussed in the first article in this series. In fact, by using a stack, it is possible to convert any recursive procedure to non- recursive procedure (see pp. 160-161 of Horowitz and Sahni(2) for details. Also, a non-recursive procedure may be more efficient than its recursive equivalent (5). A sample non-recursive BASIC program for an inorder traversal of a binary tree is shown below.

```
100 REM SUBROUTINE FOR INORDER TRAVERSAL OF A BINARY
    TREE
105 REM WHEN A NODE IS VISITED, THE VALUE OF ITS KEY
    IS PRINTED
110 REM ROOT POINTS TO THE ROOT OF THE TREE
120 REM N IS THE MAXIMUM HEIGHT OF THE STACK
130 TREE = ROOT
```

```
135 REM TREE IS A TEMPORARY VARIABLE, POINTS TO ROOT
    OF TREE OR SUBTREE
140 TOP = 0
145 REM INITIALIZE THE STACK; TOP IS TOP OF STACK
150 WHILE TREE <> 0
155 REM MOVE DOWN THE LEFT SUBTREE
160 TOP = TOP + 1
170 IF TOP > N THEN GOSUB 1000
180 STACK(TOP) = TREE
185 REM PUSH POINTER TO  ROOT OF SUBTREE ONTO STACK
190 TREE = LEFT(TREE)
195 REM MOVE TO LEFT SUBTREE
200 WEND
210 IF TOP = 0 THEN PRINT:RETURN
215 REM RETURN WHEN THE STACK IS EMPTY, NO MORE NODES
    TO VISIT
220 TREE = STACK(TOP)
225 REM POP A NODE OFF THE STACK
230 TOP = TOP - 1
240 PRINT KEY(TREE)
245 REM VISIT THE NODE AND MOVE TO THE RIGHT SUBTREE
250 TREE = RIGHT(TREE)
255 REM LOOP UNTIL DONE
260 GOTO 150
270 END
.
.
.
1000 REM SUBROUTINE CALLED WHEN STACK IS FULL
.
.
.
```

In PASCAL, procedures for tree traversal are much shorter, but they may seem obscure to persons who aren't used to recursive subroutines. We will assume that our nodes are defined by the following TYPE statement:

```
TYPE ptr = ↑node;
    node = RECORD
               left,right : ptr;
               key : integer
           END;
```

The procedure for an inorder traversal is shown below.

```
PROCEDURE inorder (tree:ptr);
BEGIN
  IF tree <> NIL THEN
    BEGIN
      inorder(tree↑.left);
      WRITE (tree↑.key);
      inorder(tree↑.right);
    END
END
```

If you are new to PASCAL, you may find this procedure enigmatic. However, it is not really much different from the BASIC program given above. The main difference is just that PASCAL handles all the dirty work of saving values on the stack, rather than leaving this job to the programmer. Also, in PASCAL, there is a special value called NIL that is the value of a pointer when the pointer does not point to anything.

A binary search tree is a special form of a binary tree that is particularly useful for storing information in sorted order when the number of items to be stored changes as insertions and deletions are made. In a binary search tree, if a node N contains a key with value K, all the keys in the nodes of the left subtree of N have values less than K and all the keys in the nodes of the right subtree of N have values greater than K. The program shown below is a sample program (in BASIC) for searching a binary search tree and inserting a key if the key is not already in the tree.

```
5  REM SUBROUTINE FOR A BINARY SEARCH TREE WITH
   INSERTIONS
10 REM KEY(J) IS THE VALUE OF THE KEY OF NODE J
20 REM LEFT(J) IS A POINTER TO THE LEFT CHILD OF NODE J
30 REM RIGHT(J) IS A POINTER TO THE RIGHT CHILD OF
   NODE J
40 REM ROOT IS A POINTER TO THE ROOT OF THE TREE.
45 REM ROOT = 0 WHEN THE TREE IS EMPTY
50 REM K IS THE KEY VALUE TO BE FOUND OR INSERTED
55 REM AVAIL IS A POINTER TO AN AVAILABLE NODE
56 REM IF K IS IN THE TREE, ITS LOCATION (NODE) IS
   GIVEN
57 REM IF K IS NOT IN THE TREE, IT IS INSERTED
60 LET J = ROOT
70 REM COMPARE AND BRANCH
80 IF K < KEY(J) GOTO 105
90 IF K = KEY(J) GOTO 190
100 IF K > KEY(J) GOTO 115
105 IF LEFT(J) <> 0 THEN LET P = J:LET J = LEFT(J):
    GOTO 80
110 IF LEFT(J) = 0 GOTO 120
115 IF RIGHT(J) <> 0 THEN LET J = RIGHT(J): GOTO 80
120 REM SEARCH UNSUCCESSFUL, INSERT K INTO TREE
125 IF AVAIL = 0 THEN GOSUB 1010
130 LET I = AVAIL
135 LET AVAIL = RIGHT(AVAIL)
140 LET KEY(I) = K
150 LET LEFT(I) = 0
160 LET RIGHT(I) = 0
165 IF ROOT = 0 THEN ROOT = I: GOTO 180
170 IF K < KEY(J) THEN LET LEFT(J) = I ELSE
    LET RIGHT(J) = I
180 RETURN
190 REM SEARCH SUCCESSFUL
200 PRINT K;" IS AT NODE",J
210 RETURN
.
.
.
1010 REM SUBROUTINE TO HANDLE SITUATION WHEN NO MORE
     NODES AVAILABLE
.
.
.
```

Again, the PASCAL version is less verbose. The pointer "ptr" is defined as shown above.

```
PROCEDURE searchtree(K : integer; VAR J : ptr);
BEGIN
  IF J = NIL THEN
  BEGIN
    NEW (J);
    WITH J↑ DO
      BEGIN
        key := K;
        left := NIL;
        right := NIL
      END
  END
  ELSE
  IF K < J↑.key
              THEN searchtree(K, J↑.left) ELSE
  IF K > J↑.key
              THEN searchtree(K, J↑.right) ELSE
              WRITE(K, "is in the tree");
END
```

This procedure would be called by a main program like the one shown below.

```
BEGIN
  tree := NIL;
  WHILE NOT EOF(INPUT) DO
    BEGIN
      READ(item);
      searchtree(item,tree)
    END;
END.
```

It is also possible to delete items from a binary search tree. A BASIC subroutine to delete a node (node D) is shown below.If the node to be deleted is a leaf, or if the node has a null left or right link, the node is simply deleted and the link of its parent node (PARENT) is given its new value. If the node to be deleted has non-null left and right links, then the algorithm finds the node that would be next in an inorder traversal. This node will have a null left link. (Try some examples, you'll see why this must be so.) This node (the "nex'" node) is moved to the position of the node to be deleted and the right subtree of the 'next' node is linked to the parent of the "next" node.

```
1100 REM KD IS THE KEY OF THE NODE TO BE DELETED
1105 REM FIRST WE MUST FIND THIS NODE AND ITS PARENT
     IF IT HAS ONE
1110 LET J = ROOT
1115 LET PARENT =0
1120 IF KD < KEY(J) GOTO 1135
1125 IF KD = KEY(J) GOTO 1160
1130 IF KD > KEY(J) GOTO 1145
1135 IF LEFT(J) <> 0 THEN LET PARENT = J:LET J = LEFT(J):
     GOTO 1120
1140 IF LEFT(J) = 0 GOTO 1150
1145 IF RIGHT(J) <> 0 THEN LET PARENT = J:
     LET J = RIGHT(J):GOTO 1120
1150 PRINT KD; " IS NOT IN THE TREE.  CANNOT DELETE IT."
1155 RETURN
1160 REM FOUND THE NODE
1165 LET TREE = J
1210 REM "TREE" IS THE NODE TO BE DELETED
1215 REM  D WILL BE THE NODE THAT MOVES INTO THE
     LOCATION OF "TREE"
1220 LET D = TREE
1230 IF RIGHT(TREE) = 0 THEN LET D = LEFT(TREE):
     GOTO 1360
1240 IF LEFT(TREE) = 0 THE LET D = RIGHT(TREE):
     GOTO 1360
1245 REM TREE DOES NOT HAVE A NULL RIGHT LINK NOR A
     NULL LEFT LINK
1250 LET R = RIGHT(TREE)
1260 IF LEFT(R) = 0 THEN LET LEFT(R) = LEFT(TREE):
     LET D = R : GOTO 1360
1270 LET NEXT = LEFT(R)
1290 WHILE LEFT(NEXT) <> 0
1295 REM FIND INORDER SUCCESSOR ("NEXT") OF THE NODE
     TO BE DELETED
1296 REM R IS THE PARENT OF THE NEXT NODE
1300 LET R = NEXT
1305 LET NEXT = LEFT(R)
1310 WEND
1315 REM PUT "NEXT" NODE IN POSITION OF "TREE" NODE
1320 LET LEFT(NEXT) = LEFT(TREE)
1330 LET LEFT(R) = RIGHT(NEXT)
1340 LET RIGHT(NEXT) = RIGHT(TREE)
1350 LET D = NEXT
1355 REM DELETE NODE AND LINK IT TO THE LIST OF
     AVAILABLE NODES
1360 LET RIGHT(TREE) = AVAIL
1365 LET AVAIL = TREE
1370 IF PARENT = 0 THEN ROOT = D:GOTO 1385
1375 IF LEFT(PARENT) = TREE THEN LEFT(PARENT) = D:
     GOTO 1385
1380 IF RIGHT(PARENT) = TREE THEN RIGHT(PARENT) = D
1385 RETURN
```

Now that you know a little about tree structures, you may want to try out your knowledge on a program of your own. A good exercise would be to program a preorder or postorder binary tree traversal. If you are planning to use a binary search tree to store a large amount of information, it would be a good idea to do some further reading before writing your program. The simple programs described in this article do not force a binary search tree to stay balanced (a binary tree is balanced if, for any node in the tree, the absolute difference between the heights of its subtrees is never more than one). For applications involving a large amount of data, it is more efficient to use an algorithm that keeps the tree balanced. The classic procedure for making insertions and deletions in a balanced binary tree is the AVL (Adel'son-Vel'skii Landis) tree algorithm. This algorithm is described in references 2-5 below.

Reference

1. Aho, A. V., Hopcroft, J. E., and Ullman, J. D., section 2.4 in The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass. 1974.

2. Horowitz, E., and Sahni, S., Chapter 5 in Fundamentals of Data Structures, Computer Science Press, Rockville, Md., 1982.

3. Knuth, D. E., Section 2.3 in The Art of Computer Programming, Vol 1, 2nd Edn., Addison-Wesley, Reading, Mass., 1973, and Section 6.2 in The Art of Computer Programming, Vol 3, Addison-Wesley, Reading, Mass., 1973.

4. Standish, T. A., Chapter 3 in Data Structure Techniques, Addison-Wesley, Reading, Massachusetts, 1980.

5. Wirth, N., Section 4.4 in Algorithms + Data Structures = Programs, Prentice-Hall, Englewood Cliffs, NJ, 1976.

N E W P R O D U C T S

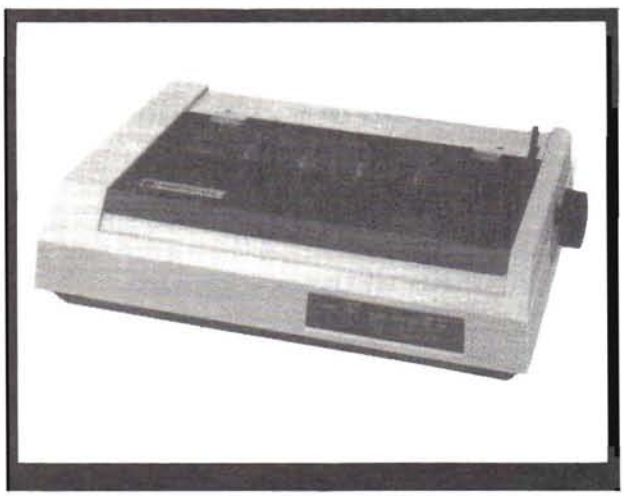
TRANSTAR 120 & 130 LETTER QUALITY PRINTER

Lightweight yet durable, the Transtar 120 features bidirectional printing and is fully compatible with the leading word processing packages. Supporting Diablo compatible codes, it can produce superscript, subscript, underlining and a true boldface. It prints quietly at 14 characters per second using a 96 character daisy wheel, up to 121 characters wide. Spacing may be set at 10, 12 and 15 cpi and is capable of 1/120-inch increments. An 'Autoload' button loads single sheet paper to one of four selectable positions. Available in either serial or parallel models, for only $540!

The Transtar 130 daisy wheel printer (pictured to the right), has all of the features of the model 120 and additionally being heavier-duty, printing at 20 characters per second. It also has a wider carriage for a maximum print width of 132 characters as well as true proportional printing. Front panel controls add extra convenience. Both serial and parallel models are priced at only $745.

IDS/DATAPRODUCTS PRINTERS

The Microprism 480 dot-matrix printer prints in a Near Letter Quality type font as well as higher speed draft quality. Character pitches are selectable including proportional pitch with a 24 x 9 matrix or 48 x 9 matrix using an enhanced mode. It can also auto-justify text and interfaces directly to Louts 1-2-3. Print speed is 110 characters per second using logic seeking and bidirectional printing as well as dot-addressable graphics of 84 x 84 dots per inch resolution. Also standard is pin-feed and friction feed using a rubber platen and serial/parallel interfaces. Only $469.

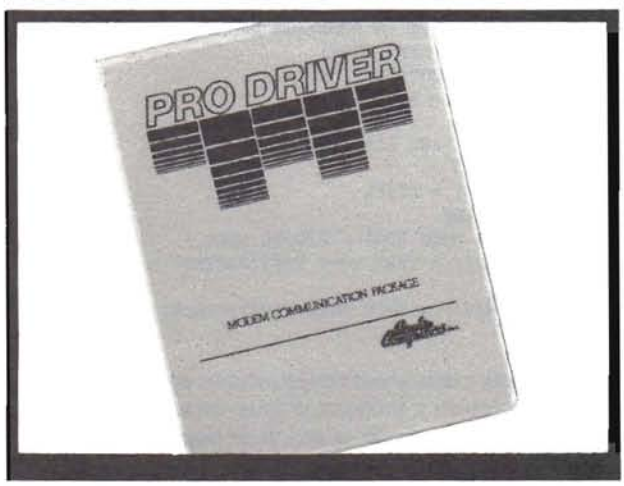


P-80 and P-132

The IDS P-80 and P-132 are high-speed dot-matrix COLOR printers. They interface directly to Lotus 1-2-3 for high resolution color graphic images. The P-80 is an 80 column wide model while the 132 is the wide carriage version featuring print speeds of up to 200 characters per second. Both print using a 24 x 9 matrix and have a 3.4K internal print buffer. Call for current pricing.

PRO DRIVER

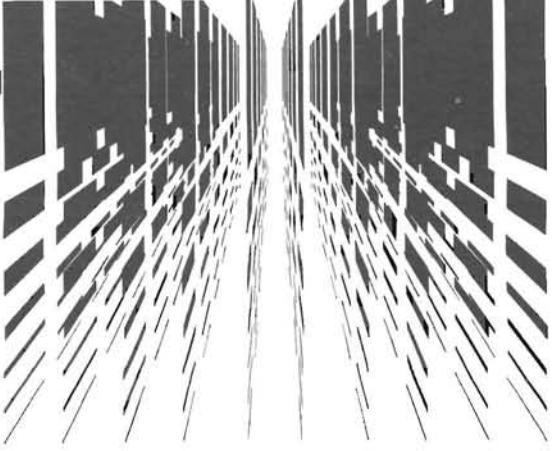
A new professional communications package for Z100 computers using ZDOS. Allows you to 'talk' to remote computers using any modem or directly to another computer through a null modem cable. It allows transmitting and receiving of both ASCII and binary files. Clear, uncluttered menu-driven displays makes using Pro Driver a pleasure. Here are just some of Pro Driver's features: Direct support for Hayes Smartmodems, Report Generator for tracking calls made, support of Xmodem, Compuserve B and X/On X/OFF protocols, 32 user-definable auto-logon sequences, printer trace for hardcopy printouts, selectable baud rates from 45 to 19,200 baud, configurable entry and exit colors, cursor modes and keyclick options, Help files and system command usage. With Pro Driver you get a professionally designed package that's simple to operate even for the novice. Packaged with over 200 pages of indexed documentation and pictorial screen displays with major sections organized with tab dividers. Only $49+$2 shipping.



Studio Computers inc.
999 South Adams
Birmingham, MI 48011
Phone (313)-645-5365

ZENITH data systems

For a complete guide to Zenith Computer systems and accessories, request our ALL NEW free 24 page fully illustrated catalog, containing many new products and lower prices. Give us a call or write and ask for the 1984 Spring/Summer edition.



Random Files

Sorting - Part 1

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

In the last two months we have seen how to set up a random file, put information into it, and retrieve that information from it. This worked fine as long as all the information put into the file was entered in the correct order. That is, the field we used to look up the record we wanted was in alphabetic order throughout the file. If it was not, we needed to know the record number and ask for it directly.

Obviously this kind of file handling is very limited. Any change to the file requires retyping to insure that everything is in the correct order. If we wish to select items by a different field (i.e. by Zip Code rather than by Last Name), a separate file would have to be entered and maintained. Disk after disk would begin to overflow with duplicate information, and there would be countless hours spent typing the same information in various orders.

There must be an easier way, and there is. It is called sorting. An entire file can be rewritten in any order we want based on the information contained in any field we choose within the individual records of the file. Several copies of the file can be maintained, each ordered on a different field to meet the needs of our data processing. Finally, to eliminate the requirement of maintaining large duplicate files arranged in different ways, we can develop "KEY FILES" which are much smaller than the original file, but which specify the order of the file based on any field we want. This is the essence of Data Base Management. But, alas, we are getting ahead of ourselves. A thorough understanding and development of file management must be taken a step at a time and this month's step is sorting.

For the different sort routines presented here, we'll use the file ALPHA.DAT which we developed last month. We'll alternate the sorted file names among the different sort programs to enable us to reorder the file as often as we want without changing the original file. We'll also use last month's lookup program SRCHFILE.BAS and modified versions of it to permit looking up information from our newly sorted files. Each sort routine we write will contain a program line which starts the actual sorting by pressing a key on the console. This isn't really necessary, but it will permit you to time the different sort methods and compare their relative performance in terms of speed.

The first method of sorting we'll look at is also the simplest. As you will soon see, it is also the least efficient but because of its simplicity it is quite easy to understand and implement. For that reason, it is very popular and is often used.

Consider this series of numbers:

4 8 2 6

They certainly aren't in order. To sort them, we could compare the 4 to the 8. As they appear to be in the correct order with respect to each other (the 4 is smaller than the 8), we could next compare the 8 to the 2. These are out of order so we would exchange them producing this series:

4 2 8 6

Next we'll compare the 8 to the 6 and again make a swap giving us

4 2 6 8

Things are better, but not exactly right. We'll have to go through the series again and again comparing adjacent numbers and exchanging them until no more changes are required. Our next pass would produce this series:

2 4 6 8

However, a change was required so we would make one last pass through the series. This time no swap of numbers would be necessary so we could exit the procedure with a correctly ordered (sorted) series of numbers. Notice how the big numbers moved to the top of the series like bubbles rising in a glass of soda (champagne for you H/Z-100 users). What we just went through is called a bubble sort.

In our example we compared numbers. However, our program could just as well have instructed the computer to compare strings. We'll write a program to do that now. We'll use the file ALPHA.DAT (make sure you keep a backup copy safely on a separate disk in case we mess this thing up) which we created last month. We'll sort it on the third field of each record and see what we get. Once sorted, we'll print our new file to see what's in it. Then we'll write a lookup program to find any item we want by searching the newly-sorted third field.

First we'll set up three arrays to handle the three fields in each record. Then we'll open our old file ALPHA.DAT and read each record into those arrays. When the file is ready to sort, an input will be asked for. This input doesn't do anything but make the computer wait to start the actual sorting process. This way, if you want to time the bubble sort, you can start timing when you press the key and stop timing when the message "SORT DONE" appears on the screen. You'll notice that even though we only compare items in the "C" array (our key field), we swapped the information in all three fields to keep the file's associated data together. When our arrays have been sorted, we write the information to a new file called ALPH2.DAT. This preserves our original in its initial order. We'll sort identical information from ALPHA.DAT for each program we write so that timing comparisons are meaningful.

In our program we'll encounter a new MBASIC command. If we wanted to exchange the values of A$(1) and A$(2) we could write

```
LET D$=A$(1)
LET A$(1)=A$(2)
LET A$(2)=D$
```

In three steps we exchanged the data of two strings. However the implementers of MBASIC did their homework and provided us with the SWAP command. Its form is:

```
SWAP variable,variable
```

or for the example above

```
SWAP A$(1),A$(2)
```

This performs the same function in one step that the three steps above performed.

Now call up your MBASIC and enter the program BUBBLE.BAS. Make sure the file ALPHA.DAT is on the disk and that there's room to repeat it as ALPH2.DAT on the same disk. Save the program as BUBBLE.BAS and run it.

```
2 '********** BUBBLE.BAS **********
4 '******** DAVID E. WARNICK ******
6 '********* COPYRIGHT 1983 *******
1000 DIM A$(26),B$(26),C$(26)       'DIMENSION ARRAYS
1010 OPEN "R",#1,"ALPHA.DAT",26     'OPEN THE FILE
1020 FIELD #1,1 AS A1$,10 AS B1$,15 AS C1$  'DESCRIBE RECORD
1030 FOR X=1 TO 26                  'FOR EACH RECORD
1040 GET #1,X                       'GET A RECORD
1050 A$(X)=A1$                      'PUT FIRST FIELD INTO ARRAY
1060 B$(X)=B1$                      'PUT SECOND FIELD INTO ARRAY
1070 C$(X)=C1$                      'PUT THIRD FIELD INTO ARRAY
1080 NEXT X                         'CONTINUE TO END OF FILE
1090 PRINT "PRESS ANY KEY TO START THE SORT."
1100 X$=INPUT$(1)                   'START THE SORT
1110 Y=0                            'SET THE FLAG TO ZERO
1120 FOR X=1 TO 25                  'GO THROUGH THE ARRAY
1130 IF C$(X)>C$(X+1) THEN SWAP C$(X),C$(X+1):
SWAP A$(X),A$(X+1):SWAP B$(X),B$(X+1):Y=1    'IF THE ITEMS
WERE OUT OF ORDER, SWAP ALL FIELDS AND SET FLAG TO 1
1140 NEXT X                         'CONTINUE THRU THE ARRAY
1150  IF  Y=1 GOTO 1110           'IF A SWAP WAS MADE,  DO  IT
AGAIN
1160 PRINT "SORT DONE"              'YOU CAN STOP YOUR TIMER
1170 OPEN "R",#2,"ALPH2.DAT",26     'OPEN A FILE
1180 FIELD #2,1 AS A2$,10 AS B2$,15 AS C2$   'DESCRIBE RECORD
1190 FOR X=1 TO 26          'FOR EACH RECORD
1200 LSET A2$=A$(X)         'LEFT JUSTIFY FIRST FIELD
1210 LSET B2$=B$(X)         'LEFT JUSTIFY SECOND FIELD
1220 LSET C2$=C$(X)         'LEFT JUSTIFY THIRD FIELD
1230 PUT #2,X               'PUT THE RECORD INTO THE FILE
1240 NEXT X                 'CONTINUE TO END OF THE ARRAY
1250 CLOSE #1               'CLOSE FILE #1
1260 CLOSE #2               'CLOSE FILE #2
1270 END
```

With the sorting done and the new file written, we can exit to our operating system and use the CP/M LIST utility or the HDOS LP:=ALPH2.DAT to see what's in our new file. Notice that the records are now in alphabetical order by the third field. With the file sorted by the third field we must use the information in that field as our key for lookup operations. For that purpose, we'll write the program LOOKUP.BAS. It is a modification of last months binary search routine designed to key on the third field of each record.

```
2 '********** LOOKUP.BAS **********
4 '******** DAVID E. WARNICK *******
6 '******** COPYRIGHT 1983 ********
1000 OPEN "R",#1,"ALPH2.DAT",26     'OPEN THE FILE
1010 FIELD #1,1 AS A$,10 AS B$,15 AS C$ 'DESCRIBE THE FILE
1020 INPUT "WHAT POSITION IN THE ALPHABET SHALL I LOOK UP";I$
'GET AN INPUT TO LOOK UP
1030 IF I$="DONE" GOTO 1250   'TYPE DONE TO END
1040 IF LEN(I$)<15 THEN I1$=I$+SPACE$(15-LEN(I$)) 'ADD SPACES
1050 A=1            'SET LOWER LIMIT OF SEARCH
1060 C=26           'SET UPPER LIMIT OF SEARCH
1070 B=INT((A+C)/2)       'FIND MIDDLE OF SEARCH RANGE
1080 PRINT "A=";A;"  B=";B;"  C=";C     'SHOW WHAT'S GOING ON
1090 IF C<A GOTO 1200     'OBJECT OF SEARCH NOT IN FILE
1100 GET #1,B       'GET THE RECORD
11f c$<I1$ THEN A=B+1:GOTO 1070      'TOO SMALL. MOVE UP
1120 IF C$>I1$ THEN C=B-1:GOTO 1070      'TOO BIG.  MOVE DOWN
1130 P$=LEFT$(B$,(INSTR(B$,CHR$(32))-1))      'REMOVE SPACES
1140 N$=LEFT$(C$,(INSTR(C$,CHR$(32))-1))      'REMOVE SPACES
1150 PRINT          'SKIP A LINE
1160 PRINT "THE ";N$;" LETTER OF THE ALPHABET IS ";A$
1170 PRINT "ITS PHONETIC IS ";P$;"."
1180 PRINT          'SKIP A LINE
1190 GOTO 1020      'GO BACK TO DO IT AGAIN
1200 PRINT          'SKIP A LINE
1210 PRINT "YOU ASKED ME TO LOOK UP ";I$      'ERROR MESSAGE
1220 PRINT "IT'S NOT IN THE FILE I'M USING." 'ERROR MESSAGE
1230 PRINT          'SKIP A LINE
1240 GOTO 1020      'GO BACK FOR NEXT INPUT
1250 CLOSE #1       'ALL DONE, CLOSE FILE
1260 END
```

Type and save LOOKUP.BAS. It will be used for the rest of our sorted file searches. The only change we'll need to make is the name of the file on line 1000. There are some important additions to this program which we haven't seen before and should understand if we're going to become proficient in file handling. The first is line 1040. If we were to type the word "SECOND" as our input, we'd never get a match. Why? The word "SECOND" is in our file and it's in the field we're searching. Remember our field statement on line 1010. The third field of each record is 15-characters long. The word "SECOND" has 6 characters. The other nine are spaces. Our computers are just tools with no reasoning ability. You and I see a match but the computer sees a 6-character variable and a 15-character variable and they don't match exactly. Line 1040 adds sufficient spaces to our input to make it 15 characters long just as the LSET command did when we added the information to our file. This way the computer sees two 15-character variables and an exact match can be found.

What happens if we add a string of characters which does not exist in our file? As you recall from last month's article, the binary search routine moves the upper and lower limits of its search range closer together each time a match is not found. It then looks at the item in the middle of that range. If no match for the input is found, both the upper and the lower limits will be equal. If we don't stop the computer when these limits are reached and a match is not found, it will do crazy things. Finally it'll have a nervous breakdown and hang up in a loop or tell us we've asked for an out of range variable. Line 1090 prevents all this. If the lower limit of the search range becomes greater than the upper limit of the range, the item we want isn't in the file. We'll branch to line 1200 for an error message and go back for another input. Line 1080 has been added so we can see what the lower limit, item looked at, and the upper limit are. It doesn't do a thing for the operation of the program, but is intended to make the operation of a binary search visible and therefore, easier to understand. The computer will tell us what it's doing and where it's looking for the information we want.

At last, we've corrected the problem of too many spaces in the data removed from the file. The MBASIC command INSTR looks up the first occurrence of a character within a string of characters and tells us where it is. If we ask it to find the first space in the 15-character string B$ we write

```
INSTR (B$,CHR$(32))
```

where a space is represented by ASCII character #32. If our string contained "ALPHA" and 10 spaces, the response would be "6" for the location of the first space. We can use this function with the LEFT$ function to keep the first 6 characters of the string B$ (ALPHA plus one space as we want the space in our printout). Lines 1130 and 1140 use this function.

Run the program and when asked for an input, type the positions of the letters (SECOND, TWENTY-FIRST, etc.) in all upper-case letters exactly as we did when we created the file ALPHA.DAT. Make sure both LOOKUP.BAS and ALPH2.DAT are on your disk. Line 1030 permits you to exit the program by entering DONE.

Thus far we have seen one type of sort routine, the BUBBLE sort. We've also improved our lookup program. The bubble sort worked fine with our little 26-record file and it would work well with any size

file but it is slow as computer processes go. Now we'll try to find faster methods of sorting our records. Until we get into key files, the result of every sort, no matter what routine is used, will be the same. A reordered file. The only difference among sort routines is how they go about deciding what to compare. Those making the fewest comparisons and the fewest swaps are the fastest. We'll treat the math and logic behind these faster routines rather lightly. They can become mind boggling. However, once you've seen the sample programs presented here, you should be able to apply them to your file-sorting needs rather easily.

The only way to make a sort run faster is to reduce either the number of comparisons made, the number of swaps of variables made, or both. It can be mathematically shown that the bubble sort above, when working on a list of 1000 items, will need an average of approximately 500,000 comparisons.

A short list is much easier to sort than a long one. If we were to break up that same list of 1000 items and put items 1, 256, 511, and 766 in order, then sort items 2, 257, 512, and 767, etc. until we ran out of short lists, the file would be roughly in order. No matter where we started in item 1 to 254, every 255th item after it would be in the correct order and we would have sorted 254 very short lists. If we then halved our interval from 255 to 127 and sorted items 1, 128, 255, 382, 509, 636, 763, and 890 followed by a sort of items 2, 129, 256, etc., we could sort 126 lists of 7 or 8 items each. If we continue halving the interval and sorting ever larger lists until the interval equals one, we will have a correctly ordered list.

The method outlined above was originally authored by Donald Shell and is therefore called the Shell Sort. On long lists it runs in one-tenth or less of the time required for a bubble sort. So why did we look at the bubble method? There are three or four methods of sorting which are very popular, and which you're likely to encounter in your reading and programming. If you get a taste of each one here, you'll be able to understand them better at that time. The following program is a SHELL-SORT for our program ALPHA.DAT. It will produce a file called ALPH3.DAT. Type it and run it. As with the bubble sort routine, you'll be given a chance to start the actual sorting procedure so you can time the operation.

```
2 '********** SHELL.BAS **********
4 '******* DAVID E. WARNICK ******
6 '******** COPYRIGHT 1983 *******
1000 DIM A$(26),B$(26)          'DIMENSION ARRAYS
1010 OPEN "R",#1,"ALPHA.DAT",26     'OPEN THE FILE
1020 FIELD #1, 11 AS A1$,15 AS B1$   'DESCRIBE THE RECORD
1030 FOR X=1 TO 26              'FOR EACH RECORD
1040 GET #1,X                   'GET THE RECORD
1050 A$(X)=A1$                  'PUT INFO INTO ARRAY
1060 B$(X)=B1$                  'PUT INFO INTO ARRAY
1070 NEXT X                     'CONTINUE THRU FILE
1080 PRINT "PRESS ANY KEY TO START THE SORT."      'MESSAGE
1090 X$=INPUT$(1)               'START THE SORT
1100 X=(2↑INT(LOG(26)/LOG(2)))-1    'CALCULATE INTERVAL
1110 X=INT(X/2)                 'HALVE THE INTERVAL
1120 IF X<1 GOTO 1280           'SORT IS DONE
1130 FOR Y=1 TO X               'FOR RANGE OF INTERVAL
1140 FOR Z=Y+X TO 26 STEP X     'FOR RANGE OF SORT
1150 W=Z                        'SET ITEM NUMBER
1160 A$=A$(W)                   'GET ITEM FROM ARRAY
1170 B$=B$(W)                   'GET ITEM FROM ARRAY
1180 IF B$(W-X)<B$ GOTO 1230    'IF ITEMS IN CORRECT ORDER
1190 A$(W)=A$(W-X)              'SWAP ITEM 1
1200 B$(W)=B$(W-X)              'SWAP ITEM 2
1210 W=W-X                      'REDUCE ITEM # BY INTERVAL
1220 IF W>X GOTO 1180           'IF SUBLIST NOT DONE
1230 A$(W)=A$                   'INFO BACK TO ARRAY
1240 B$(W)=B$                   'INFO BACK TO ARRAY
1250 NEXT Z                     'CONTINUE THRU SUBLIST
1260 NEXT Y                     'NEXT SUBLIST
1270 GOTO 1110                  'HALVE INTERVAL AND SORT AGAIN
1280 PRINT "SORT IS DONE"       'MESSAGE
1290 OPEN "R",#2,"ALPH3.DAT",26     'OPEN ANOTHER FILE
1300 FIELD #2,11 AS A$,15 AS B$     'DESCRIBE THE RECORD
1310 FOR X=1 TO 26              'FOR EACH ITEM IN ARRAY
1320 LSET A$=A$(X)              'DATA FROM ARRAY TO BUFFER
1330 LSET B$=B$(X)              'DATA FROM ARRAY TO BUFFER
1340 PUT #2,X                   'DATA FROM BUFFER TO FILE
1350 NEXT X                     'CONTINUE THRU ARRAY
1360 CLOSE #1                   'CLOSE FIRST FILE
1370 CLOSE #2                   'CLOSE SECOND FILE
1380 END
```

Another improvement has been made with the SHELL sort. In line 1020 we described our record as having only two fields. We know there were three when we put them in. What happened? In an effort to save time and computer operations, all fields to the left or to the right of the key field used for sorting have been handled as a single field. The computer couldn't care less whether 1 character is a letter and 10 characters are a phonetic, or whether a single 11-character variable exists. The big difference is that we handle one, not two strings and that two, not three swaps of data are made.

Now that the list is sorted and ALPHA.DAT is written to the disk we can print it as we have done with our previously sorted files. We can also change line 1000 of our program LOOKUP.BAS to open ALPH3.DAT and use it to look up info.

As a point of interest, the bubble sort took 12.73 seconds and the Shell sort took only 5.04 seconds on my H-89.

So what have we accomplished this month? We have had an introduction to two types of sorting routines and have made some improvements to the way we look things up.

Next month we'll improve the way we open files for sorting and we'll look at some other routines to get the job done. We'll also develop six standard data files to be sorted. Then we can use them to compare the performance of the sort programs we write. This will show us why different sorting methods are preferred under different conditions. After that article we'll be able to get back to our Random Files and begin using the real power of our magnificent machines.

See you next month.

Bells and Whistles For "MAILPRO"

Thomas F. Best
Butler University
4600 Sunset Ave.
Indianapolis, IN 46208

Introduction

HDOS MAILPRO under MBASIC is a good basic mailing list package. With a few changes to the code, it can become much more flexible and efficient. In this article I'll explain how to add the following features to the program. (Note that these modifications are based on the HDOS version -- the CP/M one is completely different. However the principles should be applicable to many input/output situations in MBASIC.)

1. Notice of number of items in the data base, in both update and printout mode.

2. Notice of number of items added this session, how many more are allowed, and an overrun warning.

3. Visual indication of field-length, with overrun warning.

4. Notice of number of items to be printed out (if you sort out a subset of the data base, for instance).

5. Choice between continuous and single-sheet printout, and pause at top-of-form to allow insertion of letterhead.

6. Running total of the number of labels or letters printed out.

I'll list the code changes first, followed by a brief explanation of their purpose and operation.

Modifications To UPDATE.BAS

1. Add LINES 102 and 104:

```
102 V$=LEFT$(PA$(6),INSTR(PA$(6),".")-1)
104 PRINT "The data base ";V$;" currently has";Q%+1;@
    "item(s)." : PRINT
```

This extracts (and then prints out) the name of the data base, and number of items in it, from LINES 40 and 50 of UPDATE.BAS.

2. Change line 110:

```
110 INPUT "Maximum # of additions";@
          " to be added this session";A%
```

This corrects a misprint in the original code (it has "if" instead of "of").

3. Add LINES 112 and 382:

```
112 X=A% : Y=0
382 Y=Y+1
```

Here we store the maximum number of additions this session (A%), and set up a loop counter (Y) to keep track of them.

4. Modify LINES 260-266:

```
260 PRINT CHR$(27);"E"; : REM CLEARS SCREEN
262 PRINT " The data base ";V$;" currently has";@
    Q%+1;"item(s)."
264 PRINT Y;"Addition(s) already made. "; :@
    IF X>Y THEN PRINT X-Y;"more allowed."@
    ELSE PRINT "Any further additions may be lost."
266 PRINT : PRINT "Command? " : A$=INPUT$(1)
```

The changes are self-explanatory.

5. Modify/add LINES 310-326:

```
310 PRINT "Add";
312 PRINT TAB(L1%+3);@
    "(Any letters beyond the 'I' will be lost)" : PRINT
320 FOR J=1 TO N%
322 FOR Z=1 TO B%(J) : PRINT TAB(L1%+5);"I"; :@
    NEXT Z : PRINT
324 PRINT N$(J); : PRINT TAB(L1%+3); :@
    LINE INPUT ": ";J$(J)
326 NEXT J
```

This gives an "overrun warning" when you are adding information to the data base. The length of each field (name, street address, city, etc.) is stored in the array B% (LINES 70 and 80). This routine prints a 'I' character, above the data entry line, for each space in the field, and warns you that anything beyond that will be lost. You could use most any character just by substituting it for the 'I' in LINE 312.

6. Add LINE 582:

```
582 FOR J=1 TO B%(F) : PRINT TAB(LEN(N$(F))+7);"I"; :@
    NEXT J : PRINT
```

This adds the same warning when changing information already in the data base.

Modifications To LABELS.BAS

7. Add LINES 162-164:

```
162 V$=LEFT$(PA$(6),INSTR(PA$(6),".")-1)
164 PRINT "The data base ";V$;" currently has";@
    Q%+1;"item(s)." : PRINT
```

These lines give the name and number of items in the data base (they have been read in in LINES 80, 130, and 140 of LABELS.BAS).

8. Add LINES 972-976:

```
972 PRINT : PRINT "Number of items to be printed: ";X%+1
974 PRINT : PRINT "Do you want ";E$+"p";"C";E$+"q";@
    "=Continuous Printing or ";E$+"p";"S";E$+"q";@
    "=Single Sheet"; : INPUT V1$
976 IF V1$ <> "C" THEN IF V1$ <> "S" THEN PRINT CHR$(7);@
    : GOTO 974
```

After indicating the number of items to be printed, this routine checks for continuous or single-sheet-at-a-time printing.

9. Modify/add LINES 1300-1306:

```
1300 PRINT "Labels/Letter(s) Printed: ";Y%+1 :@
     IF Y%=X% THEN GOTO 1310
1302 IF V1$="C" THEN GOTO 1020
1304 LINE INPUT "Insert new paper,";@
     " then hit <RETURN> to continue";V2$
1306 GOTO 1020
```

This implements the printing mode (continuous or single-sheet) which was chosen.

10. Add LINE 1310:

```
1310 PRINT #3, : PRINT#3,X%+1;@
     " LABELS/LETTERS PRINTED" : CLOSE 3
```

A cosmetic change to indicate either labels or letters printed out.

Conclusion

I think you'll find these modifications straightforward, enjoyable, and a great enhancement to an already very useful program. Even if you don't have MAILPRO you'll probably find the techniques to be helpful elsewhere. As a bonus, by doing the modifications and figuring out how they work you can gain a better understanding of the workings of MAILPRO in particular, and MBASIC in general.

Using Special Function Keys In 'C'

Dr. Robert Crawford
Western Kentucky University
Dept. of Mathematics & Computer Science
Bowling Green, KY 42101

The 'C' programming language is rapidly gaining in popularity among Heath users' due not only to its inherent flexibility and efficiency, but also to the excellent and inexpensive compiler (C/80) available from the Software Toolworks. The truly structured form of 'C' lends itself naturally to the development of many application programs. The H/Z-89's special function keys can lend a professional touch to such programs. Unfortunately the input functions provided by 'C' (such as getchar) automatically echo the input on the screen, leaving little pieces of garbage (like ↑ [Q for the red key) lying about on the display.

The accompanying program (sfk.c) illustrates how to overcome this problem. The basic idea is simple enough — a switch inside a switch — and there are only a few tricks. First is the realization that we need to access BIOS function 6 to do direct console input without the nasty echoing problem. This is accomplished in straightforward fashion by the assembly language subroutine inkey(). (Readers who need to access many BIOS or BDOS functions from C/80 are urged to consult the article "CP/M BDOS and BIOS Calls for C" by Terje Bolstad in the June 1983 Dr. Dobb's Journal.) The second trick is the use of the construct

```
while((c = inkey()) == 0);
```

to get the computer to whir silently while waiting for console input. Finally, there is the use of the "do-nothing loop"

```
for(i=0; i < 100; i++);
```

to provide a slight delay following the recognition of an escape. If another character has not arrived from the console by the end of this delay, we are not looking at the escape code for a special function, but simply at the escape key itself. You may need to tinker with the value 100 in this loop to lengthen or shorten the delay in order to get the best response on your system.

The program was written and tested on a 64K H-89 using C/80 under CP/M.

```
/*************************************/
/* sfk.c -- special function key demo */
/*                                   */
/*    Robert Crawford — June 1983    */
/*************************************/

#include printf.c
#define  SUB   26       /*   CTRL/Z (used to exit the program)   */
#define  ESC   27       /*                ESCAPE                 */
main()
{
   int c,d,i;
   while((c = inkey()) == 0);
   while ( c != SUB ) {
      if ( c == ESC) {
         for(i=0; i < 100; i++);
         if ((d = inkey()) != 0) {
            switch (d)  {
            case '?':
               c = inkey();
               switch (c) {
               case 'M': printf("\nENTER key in alternate keypad mode\n");
                         break;
               case 'n': printf("\n. key in alternate keypad mode\n");
                         break;
               case 'p': printf("\n0 key in alternate keypad mode\n");
                         break;
               case 'q': printf("\n1 key in alternate keypad mode\n");
                         break;
               case 'r': printf("\n2 key in alternate keypad mode\n");
                         break;
               case 's': printf("\n3 key in alternate keypad mode\n");
                         break;
               case 't': printf("\n4 key in alternate keypad mode\n");
                         break;
               case 'u': printf("\n5 key in alternate keypad mode\n");
                         break;
```

```
                case 'v': printf("\n6 key in alternate keypad mode\n");
                          break;
                case 'w': printf("\n7 key in alternate keypad mode\n");
                          break;
                case 'x': printf("\n8 key in alternate keypad mode\n");
                          break;
                case 'y': printf("\n9 key in alternate keypad mode\n");
                          break;
                default:  break;
                }
                break;
             case 'Q': printf("\nRED key\n");
                       break;
             case 'P': printf("\nBLUE key\n");
                       break;
             case 'R': printf("\nWHITE key\n");
                       break;
             case 'S': printf("\nf1 key\n");
                       break;
             case 'T': printf("\nf2 key\n");
                       break;
             case 'U': printf("\nf3 key\n");
                       break;
             case 'V': printf("\nf4 key\n");
                       break;
             case 'W': printf("\nf5 key\n");
                       break;
             case 'J': printf("\nERASE key\n");
                       break;
             case 'L': printf("\nIL key\n");
                       break;
             case 'M': printf("\nDL key\n");
                       break;
             case 'H': printf("\nHOME key\n");
                       break;
             case '@': printf("\nIC key\n");
                       break;
             case 'N': printf("\nDC key\n");
                       break;
             case 'A': printf("\nCursor up key\n");
                       break;
             case 'B': printf("\nCursor down key\n");
                       break;
             case 'C': printf("\nCursor right key\n");
                       break;
             case 'D': printf("\nCursor left key\n");
                       break;
             default:  break;
             }
          }
          else
             printf("\nESC key\n");
       }
       else
          putchar(c);
       while((c = inkey()) == 0);
    }
}

inkey()    /* Direct console input via BIOS */
{
#asm
   MVI   E,0FFH
   MVI   C,6
   CALL  5
#endasm
}
```

Vectored from 39

I hope that HUG/REMark will be able to avoid such extended advertisements in the future. I am delighted with my Z-100 and would like to see some very technical material on its inner workings in REMark. Especially on extending ZDOS to accept, for instance, a RAM disk and on interfacing to the BIOS.

Samual Green
Barrett Pond
Stoddard, NH 03464

Three Items For HUGgers

Dear HUG,

I have three items to offer fellow HUGgers.

1. Those of you who have had a frustrating time running Smith Corona TP-1 printers have done so because the TP-1 is factory set to check even parity. Heath software normally does not observe parity. This is how to configure the printer: (a) Raise the front casing top and remove the two screws that attach the top to the base. Snap the tree ball fasteners at the rear of the casing top out of their clips by giving a sharp vertical tug. Be careful though, the clearance between the casing top and platen knobs is quite tight. Spread the sides of the top and carefully move the top to the rear to expose the logic board at the left rear of the unit. (b) The logic switch is a little weird in that the red dots that appear on the "on" or "off" side indicate the side NOT selected. Hence if the dot is next to "off", the switch is in the "on" position. Use a pointed instrument to push in on the red dot to change the switch setting. (c) The following tables summarize switch configuration.

Table 1:

Char Length	S1
7 bits	on
8 bits	off

Table 2:

Parity	S2	S4
None	on	off
Odd	off	on
Even	off	off

Table 3:

Baud Rate	S4	S5	S6	S7
50	ON	ON	ON	ON
75	ON	OFF	ON	ON
110	OFF	ON	ON	ON
134.5	OFF	OFF	ON	ON
150	ON	ON	OFF	ON
300	ON	OFF	OFF	ON
600	OFF	ON	OFF	ON
1200	OFF	OFF	OFF	ON
1800	ON	ON	ON	OFF
2000	ON	OFF	ON	OFF
2400	OFF	ON	ON	OFF
3600	OFF	OFF	ON	OFF
4800	ON	ON	OFF	OFF
7200	ON	OFF	OFF	OFF
9600	OFF	ON	OFF	OFF
19200	OFF	OFF	OFF	OFF

Thanks to Smith-Corona Product Service Reps for providing this information.

2. I recently spent a frustrating week of troubleshooting, kicking the pets, and tearing my hair out over a problem with my H- 89. The solution surprised me somewhat, so I want to pass it on. My computer refused to come up. I'd get but one beep (the terminal worked fine when off line). I checked the power busses, the interconnect cables, the Console Serial Output port on the CPU board, etc., etc., etc. In an effort to keep a steady flow of data on the busses, I placed switch 501 on the CPU in the constant memory test mode and to my surprise the test messages appeared on the screen! What surprised me further was that the Octal LWA indicated 337377 -- 48K, when I had an expansion board installed to bring the system up to 64K! I pulled the ribbon connector from the IC socket on the CPU board and cleaned the pins thoroughly and have had no trouble since.

3. A bunch of us in the Rapid City, SD area have discovered we are not the only Heath/Zenith computer owners in the western part of the state, and are trying to form a Heath Users' Group for the entire West River area. Give Denny Nichols a call (506-923-4784) and we'll be sure you will be invited to the next organizational meeting.

Walter R. Washburn III
9994A St. Onge Street
Ellsworth AFB, SD 57706

The Dreaded HT Bug

Dear HUG,

I am writing concerning a small Screen Dump program that appeared on Page 50 of REMark #47 (supplied by Ken Terry).

My experience in trying to use the program may be of interest to other readers.

I used the program to dump a graphics pic, (actually a map of Australia) by inserting the appropriate code in the middle of the program. The result on the screen was all that I could ask for, but the screen dump to the C.Itoh M8510SCP left a little to be desired. At some places in the print out the drawing did not match the screen, there appeared to be some extra dots, and the rest of that line was displaced.

After making some use of the printer HEX DUMP capability, it was found that there was some spurious bytes in the stream to the printer. They were all Hex20. (This is a clue.)

With the help of a friend, and using some extra code to dump each line to the screen before sending to the printer, it was found that the trouble spots all involved a byte value of Hex09. (Another clue.)

Then we realized we were being bitten by the dreaded HT bug, and also that ZBASIC was showing its Microsoft ancestory. Any use of 09 in a message that is being 'LPRINT' wakes up a gnome inside the machine that has a thing about 09 and turns it into spaces (Hex20) regardless of what the 09 is!!!

There are two options, either make sure that your graphic.pic has no 09 bytes, or use BASCOM to compile the program. (The HT gnome does not live in BASCOM!)

This same bug will bite if you are sending 09 to a printer in an ESC string to select a print mode (e.g. Epson FX-80).

SO....beware the dreaded HT bug....!!

L. T. Scotney
5/146 Chester Hill Rd.
BASS HILL N.S.W. 2197
AUSTRALIA

New HUG In Seoul, Korea

Dear HUG,

I am attempting to start a users' group. I am located in Seoul, Korea. I feel there is a good chance that there are other Heath users here because of the US Army here and there are many users in the service. I contacted Ms. Bacon and also Sextant to see if there were other people receiving REMark or Sextant at a 963-- APO. Sextant said they have 23 people receiving their magazine at that APO. Ms. Bacon did

not tell if anybody else received REMark at this APO due to privacy laws, as did Sextant, but they did give me that number so I know the chances are good.

The address below can be used, and I may be contacted at the following phone numbers:

Military -- Younsan 293-8990
Commercial -- Seoul 792-3894

The name of the group could be "KHUG" or "KORHUG" and as of now the group size is one, and I have no set time for meetings.

Robert J. Collins
Kentron Int. Inc.
HHC 1st Sig. Bde.
APO SF 96301

Where Are the Books On Heath/Zenith Computers?

Dear HUG,

I am renewing my membership in HUG. I don't have any other choice if I want any support for my H-89. I think it is interesting that Stacey's Bookstore has "well over a thousand" titles concerning computers and not one book on:

Heath/Zenith Computers
HDOS Operating System
Benton Harbor BASIC

I have also found that the twenty or so computer stores in San Francisco that I have checked with do not carry:

software for Heath/Zenith Computers, or
hard sectored disks of any description.

I think it unfortunate that the people at Heath have not seen a computer other than their own.

E. Sheffield
627 46th Ave.
San Francisco, CA 94121

Question On Assembly Language

Dear HUG,

I have a question which I would like you to put in a questions & answers article. I have been working with the assembly language alot, and Pascal, too. My question is "How do you get a Catalog of the disk while using Pascal or the assembler language?".

I would be very grateful for you to answer my question for me. My mom works with horses greatly, typing in the pedigrees, too. She doesn't know most of the commands, and doesn't want to. I am trying to write a program to do all of this for her her, such as catalogs, deletions, renaming, and typing or printing. I need my question answered for the catalog part. Thank you very much.

John Ruark
RR #1 Box 4
O'Fallon, IL 62269

Ed:) See the article "Disk Catalogs From BASIC or Tiny Pascal" which appeared in Issue 18, page 25 of REMark.

Index of Advertisers

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

CUT ALONG THIS LINE

HUG MEMBERSHIP RENEWAL FORM

When was the last time you renewed?

Check your ID card for your expiration date.

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

Name _____

Address _____

City-State _____

Zip _____

REMEMBER - ENCLOSE CHECK OR MONEY ORDER

CHECK THE APPROPRIATE BOX AND RETURN TO HUG

	NEW MEMBERSHIP RATES	RENEWAL RATES
US DOMESTIC	$20 ☐	$17 ☐
CANADA	$22 ☐	$19 ☐ US FUNDS
INTERNAT'L*	$30 ☐	$24 ☐ US FUNDS

* Membership in France and Belgium is acquired through the local distributor at the prevailing rate.

Heath Users' Group

Hilltop Road

Saint Joseph, Michigan 49085

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

Volume 5, Issue 4

POSTMASTER: If undeliverable, please do not return.

885-2051