REMark

Issue 30 • July 1982

Official magazine for users of Heath/Zenith computer equipment.

on the cover . . . .

Z Series
Low-boy unit

on the stack

>CAT

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

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

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

Back issues are availble at $2.50 plus 10% handling and shipping. Requests for magazines mailed to foreign countries should specify mailing method and add the appropriate cost.

Send payment to:

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

HUG Manager . . . . . Bob Ellerton
Software Coordinator and Developer . . . . . Jim Blake
Software Engineer . . . . . Pat Swayne
Hug Bulletin Board and Software Developer . . . . . Terry Jensen
HUG Secretary . . . . . Margaret Bacon

REMark Editor . . . . . Walt Gillespie
Assistant Editor and Layout . . . . . Nancy Strunk

REMark

10...9...8...7... 6...5..

4 weeks is all that is left before about 1000 HUGGIES will descend on the Hyatt Regency O'Hare in Chicago for the first National HUG Conference August 6,7 and 8. There are about 400 tickets left, so if you have not registered, do so soon. We would like to see you there. The Agenda is as follows.

HUG CONFERENCE AGENDA

FRIDAY Afternoon

Most of the HUG staff will be found running around the main floor in an interrupt driven mode.

4:00-10:00PM - Exhibitors set-up time. Stop by and interrupt them too.

8:00-10:00PM - Late Registration ($20). Right by the front entrance. Even if you've already registered, you will want to stop and pick up a holder for your badge and a HUG note pad.

SATURDAY

8:00 - More late registration.

The Exhibits will be open all day.

9:00 - Opening remarks and introductions. There will be many of the Heath/Zenith hardware and software engineers and executives present during the two days and they will be glad to answer any of your questions.

9:15 - Bill Johnson, Heath Company President will open the conference with a dicussion of the growth and direction of Heath/Zenith with regard to all product lines as well as the computers and a visual tour of the twelve acre facility in St. Joseph.

10:15 - Jerry Pearlman, Senior Vice President of Finance, the man responsible for Zenith's acquisition of Heath Company in 1979 will address the overall corporate position as well as the computer industry as a whole.

11:15 - Tom Dornback, Vice President of Software Development, will discuss the overall philosophy of the software group, how they operate, what's coming and give us the first look at the software capability of the 'Z' series of computers. Tom has reserved a period of time for questions.

12:30 - Lunch for all HUGGIES courtesy of Heath Company.

1:45 - Barry Watzman, Computer Product Line Manager, will present the new 'Z' series and answer questions.

3:30 - Gregg Chandler, Operating Systems Engineer, will present a preview of the next release of HDOS.

The afternoon will conclude with a overall Q and A session on anything covered today.

5:00 - Free time to visit the exhibitors and socialize.

7:30 - Dinner courtesy of ZDS, followed by the Keynote Address by Don Moffet, President of Zenith Data Systems. After that, the HUG staff, assisted by Joe Schulte President of Veritechnology (your local Heathkit Electronic Centers) and the exhibitors, will conduct the drawing for over 20 door prizes, including a Z machine, H-89 and a H-25!... Yes... you must be present to win. Large items will be shipped to your home.

SUNDAY

8:00AM - The Exhibit area re-opens. You can sleep in and catch a few Z's or there will be a few Z's to play with and a chance to chat with your new and old friends.

10:00 - Back to work. HUG business meeting. Your opportunity to rain on us.

11:00 - Gordon Eubanks Jr. Vice President of Languages from Digital Research. Gordon is the author of CBASIC and the former president of Compiler Systems until its' acquisition by DR. He will preview 'What's next from Digital Research.'

12:30 - Leisurely snack lunch on your tab.

1:30 - Gordon Letwin, or JGL to some. Gordon is the original author of HDOS and is now the project leader at Microsoft on MS-DOS or Z-DOS the operating system chosen for the IBM PC and the Z-100 series. Gordon will give a closer view of this operating system and a peek at the future.

2:30 - Mike Brenner, Computer Product Line Manager for Terminals and Applications Software may unveil some 'newies'.

3:45 - Doug Bonham, the Director of Educational Products will report on how microcomputers are being used in the educational field and a review of present and future educational products.

4:45 - See ya!

YOUR FRIENDS AND OURS

Among those planning to attend (in no particular order) are... most of our advertisers, including Software Toolworks Ltd. Walt Bilofsky; CommSoft Inc. Howard Nurse; Ultimeth Inc. Dean (SY: Driver) Gibson; Trionyx; Keyboard Studio, Ray (HUG posters) Massa; Generic Software, Dave Powers; D G Electronics, Bruce Denton et al; Sextant/Buss, Charlie Floto; Software Wizardry Inc. Tom (ZLYNK by Dale Lamm) Jorgenson; Livingston Logic Labs, Ray (BIOS 80) Livingston; Evryware, (Invasion, Y-WING etc.) Dave Murray and Joe Gargiulo; Phillips Engineering, makers of A-D hardware for the 89, Mel Winters; Micro-Interface, Ted Bengal; Sunflower Software, Richard Kerbel; Kres Engineering, Bob Koepel President will introduce the new expansion unit for the 89; Atra (Housemaster home control unit) Reuel Launey, President.

Also, the Chicago VEC stores will have a booth featuring very special buys on many Heath products. (Bring your plastic money.)

SPECIAL AWARDS

Saturday evening at the dinner, some special awards will be made to the following individuals:

Doc Campbell... For authoring the longest SUMMARY in the history of journalism, which apppears on page 10 of this issue... and continues next month!

Bill Parrott... for suggesting that there be a National HUG conference and that it be held in his home.

Henry Fale (H-Scoop)... for telling it like it almost is.

Tom Jorgensen...for maintaining the longest running dialog on MicroNet without a carriage return... or modem.

Ray and Nancy Massa... A four color life-sized poster of the H-10 high speed paper punch mass storage device.

Charlie Floto... for introducing sex to the computer industry.

There will be other prizes too, including a yet-to-be announced 64k expansion for the 89. This expanded memory allows multi-tasking and can be arranged so that, for instance, you have a full 64k of work space left for your program, after loading Basic! Here is how it works. The 64k is not quite random access. 65,535 eight bit bytes are arranged side-by-side and spaced a minute 5 mm apart (an industry de-facto standard) on special weather-proof high speed mylar tape only one inch wide. Using a special adhesive, supplied with the kit, the user then glues the ends together which renders a tape measuring 682.35 feet

Vectored to page 37

Getting Started with CP/M Part 5

William N. Campbell, M.D.
855 Smithbridge Road
Glen Mills, PA 19342

Abstract -

An MBASIC program for creating a mailing list with optional field (allowing for 3 OR 4 lines of name and address), AND with optional field for phone number or OTHER data is presented and discussed. A printer label program for outputting to continuous form labels (one to four across) is given. Some basic string handling techniques are discussed. In addition, for the new MBASIC programmer, I will try to explain how each program works, and I will explain in detail how I actually went about writing these programs. These programs may also be suitable for use with data other than mailing lists, if appropriately modified.

Many mailing list programs I have examined are deficient in some important areas. Frequently one needs more than one line for the "street address" portion of the name and address. When printed out, the name and address will occupy 3 OR 4 lines for each label. Did you ever receive mail that had a label on it with your name, then a blank line, followed by 2 lines of address? I know I frequently get this type of labeled mail. The blank line usually means that there was an optional line of address allowed for in the computer program that produced the label, but that the programmer didn't take the time to remedy the blank line! Also, many times one desires to include data information in the records which are not part of the actual name and address. An example would be a phone number. And, some folks desire to print out to continuous form labels which are 1, 2, 3, or 4 across the carrier paper. I will discuss programs which "automatically" handle these features. I previously presented (REM issues 10 and 11) "bare bones" programs for a mailing list. Now, we will elaborate on some of the issues. The material in this article is presented using sequential file handling techniques. I will show how to change these sequential programs to random file programs, and discuss how to transfer data created by MAKESEQ.BAS to random access data files in a later article. These programs were written using CP/M version 2.2 and MBASIC version 5.2. Actually, they will work fine with HDOS MBASIC version 4.82 with very minor and minimal changes.

****
FOR HDOS and MBASIC version 4.82 MAKE THESE CHANGES: In MAKSEQ.BAS, line 60 should read

60 CLEAR 2500.

To alter LABEL.BAS for HDOS MBASIC, change line 100 as just mentioned above. Then, change all "LPRINT" statements throughout the program to "PRINT #2,". This is most easily done as you enter the program. Next, add line 255:

255 OPEN "O",2,"LP:"

Last, add line 445:

445 IF I-1=-1 THEN 680

To alter PRDATA.BAS for HDOS MBASIC, change the "LPRINT" to "PRINT #2," and add a line --> 25 OPEN "O",2,"LP:"

Don't forget to LOAD LP: before you load MBASIC into memory with HDOS.
****

From here on I assume only that you know how to load MBASIC into memory (from the monitor prompt type MBASIC<cr>); that you know how to load a preexisting MBASIC program Einto memory (LOAD "FNAME<cr>) (ALWAYS USE UPPER CASE LETTERS when loading or saving programs in CP/M MBASIC); that you know how to save an MBASIC program (SAVE "FNAME<cr>); that you exit from MBASIC to the system monitor by typing SYSTEM<cr>; and that whenever you see "<cr>" in an article that I write, it means you hit the RETURN key on your terminal keyboard. You should also know that with CP/M MBASIC you list the program by typing LIST, and you list the program to your printer by typing LLIST. And, you do NOT have to CLEAR any memory string space for string variables in CP/M MBASIC as it is done dynamically for you. Also with CP/M MBASIC you reference drives as A:, B:, etc., rather than SY0:, SY1:, etc. Last, both HDOS and CP/M MBASIC use an apostrophe (') OR REM to indicate that whatever follows is a comment and not to be acted upon as a program statement. The programs that I will present have numerous "REMarks" and you should NOT leave out the apostrophes that are present in the program listings when you enter the programs.

I urge the newcomer to take his or her time when reading this text and to actually enter the program lines as I will outline. Also it would be to the reader's advantage to reference his MBASIC manual to clear up any points that I have failed to clarify sufficiently.

When writing these programs, I began by creating a test data file using an editor. (I used PIE from Software Toolworks.) This test data file is shown in Figure A. Note that this data file consists of several records (each record is one line and each record, therefore, is separated from the next record by a <cr>. (<cr> means "carriage return".) Each line is composed of numerous items, and these items are logically associated with each other. The items are termed "fields". Each field is separated from the next field by a "delimiter", and I used a reverse slash (\) as a delimiter. Summarizing this, a sequential data file consists of one or more records, each record

being separated from the next by a delimiter (<cr> in our example); each record consists of one or more fields, and each field is separated from the next field by a delimiter (\ in our example.) Now, examine figure A. Note that the first line contains 7 fields (6 delimiters). Note that the second line has 6 fields (5 delimiters). Note also in the second line that this record has a "null" field (\\). This means, as you will see later, that there was NO optional line of address entered. Note that all the records (all the lines) except records 2 and 3 have a phone number in the last field. Last, note that the city/town and the 2 letter abbreviation for state, and the 5 digit zip codes are NOT separated from each other with delimiters. Here, no delimiters are necessary as there are ALWAYS 2 letters for state and ALWAYS 5 digits for zip code. Finally, I intentionally kept all the names and addresses very short, so that I could test the output on standard (9 1/2 inch wide) continuous form paper. In actual use the "LABEL" program should accept and print out any line lengths, maximum length being determined only by label width and the capability of the printer. I suggest that you now use your editor and create the data file shown in Figure A. Name the file DATA.

So, with the above in mind, I wrote the program LABEL.BAS. I wished to allow for correct printout of name and address, regardless of whether the output labels were 1, 2, 3, or 4 across the page. The correct printout should allow for 3 OR 4 lines for EACH name and address. And, of course, the phone numbers should NOT be printed on the labels if they were going to be used for mailing purposes. On the other hand, we might wish to have the phone numbers print out for our own use, so allowance would have to be made for this contingency. Also, I wished to be able to EASILY alter all the necessary parameters for appropriate label output. Now, for a preview of the output of LABEL.BAS examine Figure B. Note that the output assumed the user desired to print his labels three across (3 horizontal labels on the paper). Note that when there were only 3 lines total for a name and address, that there is NO blank line where the optional line had not been entered and was a "null". Note also that the phone numbers were NOT printed out. We also note that there are NO restrictions on whether or not the user desired upper or lower case output. Last, note that NO delimiters were printed out.

I decided to write LABEL.BAS using the following approach:

I would first write and test the program "module" that dissected any record into its component parts. Then, I would use a FOR-NEXT loop to input the desired number of records into an ARRAY. Next, I would use more FOR-NEXT loops to output to each horizontal row of labels. I would have to set certain parameters such as the beginning left margin and tabs for the output early in the program. Also, I would have to OPEN the data file, and CLOSE it when done, using an EOF(n) statement to trigger the CLOSEing. Last, it would be nice to have the program tell us how many labels had been printed out when we were finished. Let us see how these objectives were carried out.

LABEL.BAS

The heart of the program, and the most intimidating

```
CAMPBELL\JOE\MR\BOX 957\123 SMITHS ROAD\GLENSIDEPA98765\459-1234
JONES\SAM\MS\2 SECOND AVENUE\\CHESTERPA19013
CRIPES\JOHN\MR\BOX 999\4 FOURTH LANE\ASTONPA19024
ADAMS\CHARLES\MR\5 FIFTH BLVD\\CHESTERPA19013\(666)/247-9999
STRALL\JACK\MR\BOX 66\8 NEXT STREET\PHILAPA19209\444-4444
JULIAN\JOHN\MR\10 LAST STREET\\LEIPERSPA19012\666-6666
Fink\Glen\Mr\That Lane\Box 123\TownvilleDE12345\999-9999
Johns\Helen\Mrs\1st Street\\CitytownPA12345\214/123-4559
```

Figure A. This data file is named DATA. It was created by using an editor, although MBASIC program MAKESEQ.BAS can produce this and similar files in a more elegant and error-free manner.

```
MR JOE CAMPBELL          MS SAM JONES          MR JOHN CRIPES
BOX 957                  2 SECOND AVENUE       BOX 999
123 SMITHS ROAD          CHESTER PA 19013      4 FOURTH LANE
GLENSIDE PA 98765                              ASTON PA 19024

MR CHARLES ADAMS         MR JACK STRALL        MR JOHN JULIAN
5 FIFTH BLVD             BOX 66                10 LAST STREET
CHESTER PA 19013         8 NEXT STREET         LEIPERS PA 19012
                         PHILA PA 19209

Mr Glen Fink             Mrs Helen Johns
That Lane                1st Street
Box 123                  Citytown PA 12345
Townville DE 12345
```

Figure B. This is the printed output for labels (3 across), produced from the DATA file shown in Figure A by the program LABEL.BAS (see text).

for the newcomer, are lines 300 to 410. I suggest that the newcomer ENTER LINES 300 THROUGH 410 NOW. IN ADDITION, ADD THE FOLLOWING LINES for test purposes. We will delete them later.

```
200 X$="LAST\FIRST\MR\STREET\OPT ADDRESS\CITYTOWNPA12345\(111) 123-4567"
500 PRINT A;B;C;D;E;F
505 STOP
507 PRINT X$
508 PRINT LEN(X$)
510 PRINT A$(0),LEN(A$(0))
520 PRINT B$(0),LEN(B$(0))
530 PRINT C$(0),LEN(C$(0))
540 PRINT D$(0),LEN(D$(0))
550 PRINT E$(0),LEN(E$(0))
560 PRINT F$(0),LEN(F$(0))
570 PRINT G$(0),LEN(G$(0))
580 PRINT H$(0),LEN(H$(0))
590 PRINT F1$(0),LEN(F1$(0))
```

Now, type RUN and <cr>. The program should print 6 numbers on the screen and stop.

If you do NOT get the results I describe, then either I have made a typographical error in this article, or you have made one when entering the lines described above! Lets double check! A frequent error is to enter the letter O when you meant a zero (0), and vice versa. Another error is to forget to enter an apostrophe where a REMark was intended. Also, it is easy to overlook a following semicolon (to prevent a <cr> or return after a PRINT or LPRINT statement respectively). And don't forget that each statement, on a multistatement line, is separated from the next statement by a colon (:).

Next, look up the INSTR function in your MBASIC manual and read it carefully. You will find that this function is actually a "match" function. In fact, some BASICs call the function MATCH. Note that there are 3 items in each INSTR function, separated by commas. The first is the starting position, the second is the string in which we are looking for a match, and the third is the character(s) you desire to use for matching purposes. The second and third items can be either strings enclosed in quotes ("this string","is") or string variables such as X$,Y$. In our use in this program, we put our name and address in X$, and then we used "\" as the third item in the INSTR function. You could use the INSTR function by saying PRINT INSTR(1,X$,"\") etc., but for our purposes we will put them in variables A through F (look at lines 300 and 310.) For A, we will start with position 1 (the first character in our name and address string (eg. the first character in X$).

In the screen printout you got when you ran the program, the first number should have been 5. Now, beginning at the left, count 5 characters (line 200). You will see that the 5th character is a reverse slash. THIS IS HOW THE INSTR FUNCTION OPERATES. IF it can find a match, it returns the position in the string to be searched of the string we are searching for; in our use here, we are looking for a "\" in the name and address string. So, that is why the first number in the printout was 5!

Now examine line 300 again and look for our next use of the INSTR function. This time we are putting the results in variable B, and, we begin our search with A+1. We do this because we found one "\" and we don't want to find it again. We are looking for the location of the NEXT "\". So, we begin with A+1. In our example, A+1=6. Examine the next number on the screen and you see that it is 11! Sure enough, when we examine line 200 again, we note that the 11th character in the string X$ is another "\". And so on, with C, D, E, and F. Check them out. Make sure you understand what we are doing. We are simply locating each delimiter (\) in the string X$. What do you suppose the INSTR function would return if there was NO match in the string. Well, to find out, type PRINT INSTR(F+1,X$,"\")<cr> and see what is returned. You are correct. It returned a 0 (zero). We began THIS matching procedure just AFTER the last "\" in our string that we placed in X$ in line 200, and there are NO more "\" characters in the string. The INSTR function is a very powerful function, and you should make sure you understand the function!

Now, remove line 505 (just type 505<cr> and it is gone). Type RUN<cr>. You should be rewarded with a screen printout of the numerical values you just studied, a printout of the contents of X$, a number representing the length of the string contained in string variable X$, AND the contents of array string variables A$(0) through F1$(0), and the length of each individual string variable.

Let us see just how the program did this.

In line 508 we simply determined if X$ did indeed contain all the characters, including delimiters, that we placed into it in line 200. This is how to determine the length of a string. Look up the LEN function now. As you can see it is not difficult. It simply returns the total number of characters in the string used in the argument.

Next, examine line 340. Do not be concerned about the array subscript (I) at this time. I will explain that later. What we are concerned with here first is the LEFT$ function. Look this up in your manual and carefully read about it. This function lets you "grab" out any given number of characters from a string beginning with THE FIRST CHARACTER. It has 2 arguments. The first is the "string" or a string variable. The second is a numerical value which states just how many characters, including the first character you want to grab. This second argument may be an actual number or it may be a variable, or a combination such as we used in line 340. The LEFT$ function that we used, says to take that portion of the string contents of X$, beginning with the 1st character, for a TOTAL of A-1 characters

and place those characters in a string variable named A$(I). Since A is the position of the first "\" in X$, A-1 tells MBASIC to place the first 4 (5-1) characters of X$ into the named string variable. This is exactly what happened as you should see on your screen. The printout should have been LAST 4. LAST is what variable A$(I) contained and 4 equals the length of this variable. All is how it should be so far.

Continuing with line 340 we come to the MID$ function. We could have used this instead of the LEFT$ function just mentioned, but I wanted to use LEFT$ to show how it worked. Read up on the MID$ function in your manual. This is probably the most important of all string handling functions, for you can isolate ANY part of a string with it. The MID$ function requires 3 arguments usually. (As we will see later, sometimes we can use only 2.) The first is the "string" or string variable such as we have used. The second argument is the position of the starting character (that we wish to extract) in the string. And the third is how many characters total we desire to extract. So, on line 340 in our example, the program tells MBASIC to use X$ as the "string", begin with the A+1 character (6th character in our string in X$) and continue for a TOTAL of B-1-A characters. You may have to think about this for a little bit. (I know I always have to!) B-1-A simply states that B-1 is the last character we wish to "grab" and if we subtract all of the characters from the beginning of the string to the start of what we wish to extract, then the difference is the total number of characters we wish to extract. Think about it and you will agree. Then, MBASIC (using this MID$ function) puts the Edesired string in string variable B$(I). We see that it has done what we wanted when we examine the screen printout and see that the contents of B$(I) is indeed FIRST, and its length is 5. Again, all is going well.

Hopefully you now understand the MID$ function and if so, you will see that lines 350 and 360 use EXACTLY the same reasoning as for the way we just used the MID$ function on line 340. This brings us to lines 370 through 390. Here the program is going to go after 3 strings within one string, and there are NO delimiters WITHIN this particular string.

The string I refer to is in X$ and consists of "CITYTOWNPA12345". Now, we didn't put any delimiters within this string since we did not need to, and since not putting in 2 delimiters within this string saves 2 bytes of space per record, both in memory AND on disk. The program is able to separate these 3 items since the last 5 characters are ALWAYS the 5 digit zip, and the 2 previous characters are ALWAYS the 2 letter abbreviation for state. So, line 370 says, examine string X$, and using MID$, extract from X$ the string beginning at E+1 (E is the position of "\" just BEFORE the C of CITYTOWN) and include everything to the position of the next "\" LESS the delimiter itself and the previous 7 characters. The previous 7 characters, of course, are the 2 letter state abbreviation and the 5 digit zip code. 2 + 5 = 7 and 7 + 1 (the delimiter) = 8. That's where that 8 came from. And, we now subtract this from the position of the delimiter following this desired string (F) to obtain the total number of characters in the "city field". And, if you examine the screen printout you should see CITYTOWN 8 which is as it should be. The 8 of course represents the length of string variable F$(I). I believe you will be able to figure out how the next 2 MID$ functions operate, since we know in advance exactly how long each string will be (2 and 5 respectively).

This brings us to line 400. Here, we simply concatenate (add together) the city field, the state field, and the zip code field and separate each from the other by a space. (You can vary the number of spaces as you desire.) The result of the concatenation is placed in string variable F1$(I).

Line 410. This is the program line which fixes up our printout and which allows us to have either 3 or 4 lines total on our label output. This line says that if the length of the contents of string variable E$(I) is 0 (IF the contents of E$(I) are a null), THEN we will move the last line (city-state-zip) up from F1$ and put its contents into E$(I). THEN, we will make the contents of F1$(I) (which originally had the city-state-zip as its contents) a "null" (F1$=""). If we did not have this line, and if there was no optional line of address (E$(I) was a null), our printout would have a blank line between "STREET ADDRESS" and "CITY STATE ZIP"!

Let's go back and examine line 320. The purpose of this line is to add a "\" to X$ (by concatenation) IF (and only if) there were ONLY a total of 5 delimiters within X$. Now, the example we used in line 200 contains 6 delimiters (6 "\" characters) and therefore line 320 was not called into play. However, if variable F contained a zero, then the line WOULD be called into play. And, variable F would contain a zero ONLY if there were no phone number as part of any given individual record. See figure A again and examine the listings of the records without phone numbers. This line also puts the correct position of the added delimiter into F (F=LEN(X$)+1). I suggest that you now change the string contents of X$ in line 200 by deleting the LAST delimiter (\) and ALL characters from this delimiter up to (but not including) the quote (") mark. Then RUN the program and examine the screen printout. The first time you run it you might want to add line 505 just to make sure that F really does contain a zero under these circumstances. The screen printout should verify all that we have discussed.

We have finished with the hard part of this program. Lets continue with the easier parts. I assume you have used your editor and entered the test DATA file as shown in Figure A, and that it is on disk and has the name "DATA".

Whenever you access any data file there are almost always 3 things that you must do. First, you must OPEN the file. Second, you usually either read from, or write to, the file. Third, you CLOSE the file. So, refer to the program LABEL.BAS and enter the following lines 240, 250, 280, and 700. Delete line 200 (and also 505, if it is still present). Now, type RUN<cr>. When you are asked to enter a file name, enter DATA<cr>. When the program stops, you should be rewarded with a printout of the name and address record which is the first line in FIGURE A. Line 240 asks us to give the name of the data file (DATA in this instance) and places our response in string variable Y$. Then, line 250 opens Y$ for input as file number 1. Line 280 then uses the LINE INPUT statement to input the first record (the first line) of our data file, AND it places the input into string

variable X$. Then all the string manipulation that we previously discussed takes place, and lines 507 through 590 print out on our screen the information that we previously discussed in detail. Finally, line 700 closed the file using the CLOSE statement.

Assuming you have a printer that uses "standard" continuous form paper (9 and 1/2 inch wide including the punched strips on each side), refer again to program LABEL.BAS and add lines 120 through line 220 inclusive. These lines set certain parameters which we will discuss later. Next, add lines 260 and 270. Next, add line 420. Now, delete lines 500 through 590 inclusive.

Since we want to be able to output to a maximum of 4 horizontal labels, and since we must print all the names for each of the 4 labels BEFORE we print the correct corresponding 1st line of each address under each name, etc., it becomes obvious that we MUST have a maximum of 4 records in memory before we start any printout. A convenient way of doing this is to use a string ARRAY. All that this means is that we can use a numerical (or variable) SUBSCRIPT (enclosed within parentheses) immediately after our string variable. Look up and read about arrays in your MBASIC manual at this time. After gaining some concept about what arrays are, look at lines 260 and 420. These demonstrate a FOR-NEXT loop. And, all that this does is to execute the lines BETWEEN the FOR statement and the NEXT statement a given number of times. For example, a "FOR I=1 to 2" statement, followed by any number of lines, followed by a NEXT I statement, would execute all the statements between the "FOR I=1 to 2" statement and the "NEXT I" statement 2 times. The first time MBASIC executes the FOR statement (in this example), I becomes SET to 1 (from zero). The following statements are then executed until MBASIC gets to the "NEXT I" statement which increments I by one (I+1=2). I now equals 2. The NEXT statement also returns E to the statements immediately after the FOR statement and executes them for the second time. This time the "NEXT I" statement increments I by 1 and I NOW equals 3. This is one more than the number of times we specified in the "FOR I=1 to 2" statement. MBASIC then realizes that this "FOR-NEXT" loop is finished, and immediately goes to the statement immediately FOLLOWING the "NEXT I" statement. The "FOR-NEXT" loop just described, therefore, executes all the statements between the FOR and the NEXT statements two times. The first time through the loop, I=1 and the second time through I=2. I=3 when we left that loop. This last is IMPORTANT, as different BASICs differ in this regard. In both HDOS and CP/M MBASIC, you exit a "FOR-NEXT" loop with the variable (I in the above example), always one more than the maximum given in the FOR statement. (Note that we are using the default STEP of +1. See your manual. You can increment by more than one using the STEP part of the FOR statement each time the variable is incremented, and you can also use descending increments, "FOR T = 10 TO 0 STEP -2", for example.) You should study "FOR-NEXT" loops in your MBASIC manual now, unless you are already familiar with them.

We can utilize a "FOR-NEXT" loop in setting up our string array. And, that is exactly what we have done in our program. Line 260 says that the first time through the loop, I=0. The next time through the loop, I=1. The next time through the loop, I=2. (The next time through the loop I would = 3, assuming that the program went through the loop 4 times if N=4). Note our use of "N-1" in line 260. By varying the value of N we can control the number of times we go through the loop. If we want to go through the loop 3 times, then we simply set N to 3, and if you refer to line 120 of our program, that is precisely what we have done. Why do we say "N-1" in line 260? Well, we are beginning with 0 (zero), NOT 1. 0,1,2 will give us 3 times through the loop. (Incidentally, upon exit from this loop and with the values given, I will = 3.) (I did not set N to 4 in the program listing, as 4 across would print the 4th name and address partly OFF standard width paper.)

Here we come to the crux of the matter. The first time through the loop, I=0. And, in lines 340 through 410, each array variable subscript is assigned the value 0 (since we are using I as the subscript). This means that after the first time through the loop, if we entered PRINT A$(0)<cr> for example, the appropriate last name would be displayed on the screen. (If we entered PRINT A$(1)<cr> at this time NOTHING would be displayed as nothing is all that is in this subscripted string variable until AFTER the next time through the loop!) Now, each time through the loop, I is incremented by the "FOR-NEXT" routine, and each time we are finished the "FOR I NEXT I" loop, the contents of 3 records are contained in the subscripted string variables. I suggest you run the program now, and when it stops, enter some random PRINT subscripted string variable statements and <cr> and see what is displayed. For example PRINT C$(0)<cr>, PRINT C$(1)<cr>, PRINT H$(0)<cr>, ETC.. Do NOT, by the way enter PRINT C$(I)<cr>, for example. Remember, that I is always incremented by 1 on exit from a "FOR-NEXT" loop, and therefore NOTHING was placed in it! You can of course type I=0, and then PRINT C$(I) etc..

Now that we have our array filled with 3 names and addresses, we will print them on 3 labels (when N=3, as in our example). Obviously, we must print the first name on label 1, the second name on label 2, and the third name on label 3 BEFORE we print the first line of address for each of the 3 labels. We will do this using several "FOR NEXT" loops.

Add lines 430 through 490 inclusive to the program. This is the first loop which prints out 3 names on the first 3 labels. Here is an explanation of these lines.

We set LM to 5 back in line 130. This represents our left margin (on my printer). CP/M MBASIC has a "bug" (a "feature") so that it ALWAYS TABs one space LESS than I think it should. For example, if you "TAB(5)", you actually only tab 4 spaces! So, line 150 remedies that situation by adding 1. (This problem is NOT present in HDOS MBASIC version 4.82.) Next, we determined (by trial and error, actually) that there were 20 spaces from the first printing character of the first label to the first printing character of the second (and from second to third, etc.) and we assigned 20 to our TAB. T = our TAB on line 140. Note that we also store the value of T into memory variable R in line 160. The value of T will change as we TAB from one label to the next, but the value of R will never change. So, line 430 sets up the value of the second TAB (T=R+LM). Then, we LPRINT our left

margin distance (LM), by tabbing. Line 440 is the FOR part of our next loop.

Why do we use "I-1"? Well, recall that when we left the "FOR I NEXT I" loop that I was one more than the value represented by N. Hence we deduct 1 from I by saying "FOR J=0 to I-1". Think about that for a moment and you will realize that we will be printing (in this example) our 3 names. The FIRST time line 450 is utilized, we print the name (etc.) on label 1. Also, line 450 then TABs to the next label (TAB(T);). Note that semicolon (;) at the end of the line which prevents the printer from doing a return! Line 470 then increments the value of T by R! (We add on the TAB value to take us to the third label (after the second name, etc. are printed)). Line 480 takes us back to line 440, where J is incremented by one. Note that throughout this loop, we use the different values of J to reference the subscripts in our array. The first time through the loop J was 0, the second time through the loop, J will be 1, etc. This second time through this loop prints the second name on label 2. And, exactly the same thing applies for the third time through the loop.

Now, the loop is ended and we drop to line 490. We must do a return here. Remember that the line that printed the name, etc. ended with a ";". Line 490 remedies this situation by telling the printer to do a return (LPRINT). Lets now explain line 460. S in this program is used as a counter. Whenever you type RUN<cr> all variables are set to 0. Line 460 is the first time in this program that S is referenced, and the first time through the loop shown from lines 440-480, S is incremented by 1. Since S=0 when line 460 is first used, S=1 after line 460 is executed by MBASIC. The second time through the loop (we are printing the SECOND name, etc.) S is again incremented by 1 to 2. Do you see now why we call S a counter? Incidentally, there was nothing magical about selecting S as a counter. This was the last thing I added to this program. All the other variables (A,B,C, etc.) seemed to have been used, I didn't see any S in the program so I used it! And this line is the only time S is used in this program (until line 710 when the program if almost done, and the value of S is printed out for the users inspection on his terminal screen S of course tells the user how many labels he used!) Type RUN<cr> and be sure your printer is turned on. You should be rewarded by one row of names across one line.

We are almost done explaining this program, for LINES 500 660 are simply more "FOR J=0 TO I-1 NEXT J" loops which print out (1 line of 1 label for each execution of each loop) with identical logic as to that just discussed. And, this program, whether each name and address are 3 or 4 lines long, outputs a total of 4 lines (although the 4th line is a null if the unit is only 3 lines long watch your printer.) ENTER ALL THE REST OF THE PROGRAM LINES NOW. WATCH OUT FOR THE SEMICOLONS (;)!

Line 665 (if you removed the REM) would print out the phone number field if you so desired; remember also to remove the REM from line 415 to do this.

After the 4th line of each label has been printed out by lines 630 to 660 FOR-NEXT loop, we fall to line 670, which does three RETURNs on the printer spacing (on our labels) to the correct position for the next horizontal row of labels.

Then, line 690 RETURNS us to line 260 which is the start of the whole process of gathering data from the next 3 records and printing them out.

SUMMARY

The program initially sets up some parameters which the user can modify before running the program. Then, after opening the file, we used a "FOR I=0 to N-1 NEXT I" loop to place the first three names and addresses in an array in memory. Next, using a series of almost identical "FOR-NEXT" loops (one for each horizontal line for 3 horizontal labels), we printed out the name, etc. fields, then the 1st line of address, then the 2nd line of address, then the city/state, 2 letter abbreviation for state, and zip. (If there were only 3 lines of total name and address in any unit we had previously moved the 4th line of that UNIT up to the 3rd, so that the 4th line was a null.) Then line 670 was responsible for spacing down to the next horizontal row of labels. Then, line 690 returns us to line 260 to repeat the whole process all over again.

But, you may ask, what happens when we run out of names and addresses to input? (We have reached end of file (EOF)!) Line 270 is called into play! This first statement in the "FOR I = 0 TO N-1 NEXT I" loop takes care of this. Here is what happens.

Whenever you use LINE INPUT to input records from a sequential data file, you almost always include an "EOF(n)" statement just BEFORE your line input statement. You should now read about EOF in your MBASIC manual if you are not familiar with it. It will automatically determine the end of your data file for you. And, you always go back to the EOF(n) statement (instead of the line input statement) whenever you desire to LINE INPUT the next record. (The "n" just after the EOF(n) is which file number was opened in the OPEN statement). Whenever EOF(1) (in our program) occurs, the IF part of line 270 becomes true for the first time and then we immediately set a numerical variable (which I arbitrarily called FLAG) to -1. (Note that this is NOT a special variable. I could have used F or BE or JB or KIT or anything else NOT used elsewhere in the program as my variable. BUT, since I intended to use THIS variable as a "flag" I simply named it "FLAG".) Also, note that the value -1 was simply picked out of the air. I could have used 99 or 66 or 123 or whatever. But, -1 has a nice computer "feel" to it.)

Now, up until EOF(1) was reached, FLAG=0. Note that immediately after EOF(1) was reached we set the variable FLAG to a value of -1. Then we immediately branched to line 430. At this time, the value of I indicates the number of names and address units in our array that we were inputting, and which as yet have NOT been printed out. The value of I at this time could have been from 0 to 2 (with N set to 3). Then from line 430 on, using the J "FOR-NEXT" loops, we printed out the present last group of array names and addresses.

(Note that if I=0 when we reached EOF(1) in line 270, then in line 440 and the other J loops, I-1 would = -1. (0 minus 1=-1.) Now, under these circumstances CP/M MBASIC version 5.2 skips the loops. However, HDOS MBASIC version 4.82, makes one pass through

EACH loop and will print out whatever was in the array subscript (0) from the PREVIOUS array (0). Hence, we must add line 445 if we run this program under HDOS. I am in the debt of Mark Rausher for uncovering this problem!)

When we reach line 680, which the program had passed through many times before (with a sizable mailing list), it intercepts our FLAG for the first time. Always before, FLAG = 0. But THIS time FLAG = -1 (BECAUSE the program set it to -1 in line 270!.) So the IF statement in line 680 is TRUE and we branch to line 700 which closes the file, prints out how many labels we printed and ENDs!

If you have an H/Z19 or H/Z89 then use line 740 instead of the present contents of line 730 as shown in the program listing. This simply clears the screen in one fell swoop. In other words -> 730 PRINT CHR$(27)+"E":RETURN

Line 110 in the program makes ALL numerical variable values in our program integers. We are doing no "number crunching" in our program, so this is legitimate for our purposes. Doing this speeds up the execution of our program.

You may have noticed that we did not need to use a DIM statement in this program, to declare the maximum number of subscripts in our array. You do not need to DIMension an array if there are under 11 total subscripts. See your manual for further details on the DIM statement.

I promised earlier to explain how the MID$ statement can be used without 3 items in it. Examine line 415. Note that only the first TWO items are used and that the last item (which tells MBASIC how many characters are in the string) is not used. When used in this way (it IS legal see your manual), MBASIC returns ALL of the characters to the END of the string, beginning with the character position given in the 2nd item. That is what happens in line 415.

We are now finished with program LABEL.BAS. Let us examine program MAKESEQ.BAS which will allow us to create our data file in a more elegant fashion than just using an editor.

Here was my approach. I strongly believe that any interactive program such as this MUST be "user friendly". By this, I mean that if the user DOES make a mistake in entering data (creating the records), he should have an opportunity to CORRECT these mistakes BEFORE any given record is actually written to disk. And, any such correction must be easily made and not require the user to type ALL of any given data record ALL over again.

Next, we should remember that the ONLY way to append new records to a sequential data file, is to first input (record by record) and immediately output (record by record) to a TEMPORARY file. This must be done for ALL the previous records. After EOF(n), then we can append the desired records, CLOSE the file and REName the file to the original name. Whenever we deal with multiple files such as this, we SHOULD protect the user, and (as far as is possible) prevent the deletion of the original file or any new files. And, as you shall see, we will find MBASIC's error handling routines to be of much value in this particular program.

I suggest that you now enter program MAKESEQ.BAS in its entirety. Take your time. Watch out for the apostrophe (') meaning a REMark; also watch out for semicolons (;). RUN this program (RUN<cr>). If it doesn't run either you or I have made one or more typographical errors. Double check! When the program does run to your satisfaction, then proceed with the following explanations.

MAKESEQ.BAS is a relatively straightforward program and I will begin the explanation at line 70 which simply GOSUBs to line 950 which clears the terminal screen and then returns (in this case to line 80). The program GOSUBs to line 950 several times in this program to clear the screen. I will defer the discussion of the error handling until later. Line 90 defines all numeric variables as integers, just as in the label program. I stored 19 in memory variable T in line 100. This was an arbitrarily small value, and used for demonstration purposes. A more realistic value would be 25 to 30, depending on your label width and also on your printer's capability. Line 120 prompts the user as to how the file name should be entered. Line 130 further prompts with an input statement and the user enters his file name which is accepted and stored in string variable T$. ELine 140 Esimply spaces, then lines 150, 160, and 170 all return the user to start again IF any of the 3 conditions tested for exist. 150 and 160 force user to enter ONLY capital letters A through Z (into T$). And, line 170 ensures that user did not enter a "." as part of file name (which he might have done if he typed "filename.ext" we will be providing our own extension of ".OLD" for a preexisting file in line 210).

Line 180 opens the named file (T$) if it exists, for Input ("I"), and 190 opens file named "TEMP" for output ("O"). Line 200 informs user that he should wait (we have to input all records from old file to TEMP file). Line 220 line inputs each record into string variable X$, then line 230 immediately outputs the record to the TEMP file (and we use S as a counter just like we did in LABEL.BAS). Line 240 then returns for another line input of the next record, and note that 240 goes to the EOF(1) statement just BEFORE the line input statement of 220. This loop continues until end of file is reached (all the preexisting records are now in file TEMP, but please note that we do NOT close the TEMP file at this time for we desire to append one or more NEW records onto the end of the file BEFORE closing it.) When line 210 detects EOF(1), it then closes T$ (the preexisting file), deletes any preexisting file that might be present called T$+".OLD", and then names T$ (the preexisting file we have been working with) as T$+".OLD". Now, what this does is to delete an "OLD" file created by a PREVIOUS session with this program, then names the CURRENT preexisting file to have an extension of ".OLD".

For example, if our file is called DATA, we would have entered DATA at line 130. The program would have transferred all the records in DATA, into TEMP. Then, after closing file DATA, DATA is renamed to DATA.OLD. This provides the user with some measure of security, since if a "crash" occurred, the original data file is now closed, and hopefully would have survived the "crash". Then, the program goes to line 250 where the user begins to enter new data (new

records), appending them onto file TEMP (which contains all the original records).

Let us pause for a moment and consider what would happen at line 180 IF there was NO preexisting file named DATA (for example) on disk. MBASIC can NOT open for INPUT a NONEXISTENT file. What do you think would happen? EWell, of course, MBASIC would stop and print a message to user that the file didn't exist. And, we would have to start over again. This is where the error routines come into play. Here is what you do. Near the beginning of your program you place a statement such as we did on line 80. We told MBASIC, with this line, that any time MBASIC detected an error and would ordinarily stop the program and inform us of the error, that INSTEAD we wanted MBASIC to go to a particular line (in this case 860). Why did I say 860? Well, when I wrote the program I did not know where I wanted it to go so I said line 5000. When the program was almost finished I RENUMbered it with the RENUM command, and 860 is what showed up! At this point look at lines 860, 890, and 900. Look in your manual for a discussion of ERRORs and ERROR codes. The codes are in Appendix J in my CP/M MBASIC manual. I will discuss line 860. This statement says that IF the ERRor was number 53 (and appendix J says that error # 53 is "File not found") AND IF the error occurred on line 180 then MBASIC should prompt the user and ask him if he is opening a new file, OR (assuming that the error was not 53 or line number was not 180, THEN (ELSE) MBASIC is to proceed to line 890 to see if whatever error is present is to be found there or on following lines.

Now, in the case under discussion, assuming that the error DID occur on line 180 and that the error WAS "file not found", then OUR program will ask user for a Y/N answer to the "new file" prompt. If the user replies in the affirmative, line 870 opens file TEMP, and then (since there is no preexisting file) RESUMEs the program at line 250. That is an example of how the error handling routines work in MBASIC. If the error occurred later in the program on lines 800 or 210 AND the error was again 'file not found' (53) on either lines 800 or 210, then lines 890 or 900 respectively would have simply resumed operation on the next line (lines 810 or 220 respectively). (RESUME NEXT means "goto NEXT line after the error" occurred.) Finally IF there was an error occurring NOT covered by lines 860, 890 or 900 of this program, then I desired to have MBASIC print out the error and STOP operation. That is what line 910 provides.

In general, you should always use the "ON ERROR GOTO 0" statement as the last line of MBASIC's error handling routines. An example of the latter is if the error would be "ERROR IN SYNTAX" (Heaven forbid). This should only occur during program development, but you DO want to be notified of such an error by MBASIC since MBASIC tells you the line number where the problem exists! If there were such an error, MBASIC would stop and print out its usual syntax error message. You should study the error handling as given in your manual, and the error codes also, at this time. If you desire to see just what happens in line 180 when a nonexistent file if referenced, just RUN<cr> this program, and type in a nonexistent file name and see what happens! End of our digression. Back to rest of the program which is easier to understand than the error handling routines!

So, we now are at line 250, having been sent there by EOF(1) if there was a preexisting file, or, if no file existed, line 870 would have sent us there (unless user was negative and entered "N" or "n" to the prompt of line 860 in which case the program ENDs.) Line 250 clears the screen, then prints a message for the user. Then follow a series of line input statements which accept the user's entries into different string variables beginning with A$ (lines 260 through 360). These are easy to understand if you just RUN the program and follow the prompts and refer to the actual lines doing the prompting. Note that line 270 provides the way for the user to terminate all his entries for any particular session with the computer. Note also that lines 340 and 350 force the user to enter only 2 characters for the state, and 5 characters for the zip, respectively. (If you are entering data and do NOT know the zip, you could enter 5 spaces, and "fool" the IF statement.)

Lines 380 through 510 provide a display of all the data that the user has entered for any given one particular record. Note that line 420, 440, 460, and 500 each test any given line of the potential record as to that lines length, and if the length exceeds the value of T (line 100), then, using GOSUB calls, the user is also informed of this fact and prompted to enter again and abbreviate! ENote that each prompt is preceded by an identifying number (from 1 through 9).

Here is how the input routine for editing works. We give the user the option of entering digits 1 through 9 for editing purposes, or 0 (zero) if ALL is OK. Line 540 uses the INPUT$() statement. You put the number of characters to be entered, before the statement "takes off", between the parentheses. Usually you use 1 as we have done. Whenever ANY active key on the keyboard is depressed, then MBASIC immediately proceeds to the next statement AND the user did NOT have to depress the return key. All automatic! Note that the INPUT$() statement also puts the character into a string variable, here Z$. BUT, there are some precautions that we can take to force the user to only enter a numerical value from 0 to 9. On line 540, we use an IF statement to return to line 540 IF the user depressed the ESC key (see any ASCII chart). Lines 550 and and 560 also return the user to 540 if the character entered is not one of the desired digits. (Note that this also would take care of the ESC key, but I wanted to show you specifically how you might use it in another program.)

Line 570 prints whatever key the user depressed on the screen, as the INPUT$ statement does no printing. Then we change the string digit to a numerical value and place it in numerical variable Z in line 580. Look up the VAL function in your manual now. Line 590 checks to see if the digit entered was a zero (in which case we are finished checking and editing and proceed to line 730 if so.) If NOT a zero, then we use the "ON GOSUB" statement of line 630 to go to the selected statement. (Lines 600 and 610 put informative strings into W$ and CO$ respectively.) Since lines 640 through 720 are repetitious, the only differences being the string Evariable selected, lets just go through one selection.

Suppose that there was an error in the "Last Name".

The user depresses the 3 digit on the keyboard and 3 is put in Z$ in line 540. Then after changing Z$ to Z (line 580), line 630 says ON 3 GOSUB etc.) What this ON statement does is to select the THIRD (in this instance) line number following the GOSUB. (If the user had selected 1, then it would be the first line number, etc.) So, in this example, the third line number is 660. The "ON GOSUB" statement then branches to line 660 and executes it. Line 660 prints the contents of string variable W$ followed by the contents of C$. If we look back at line 410, we see that C$ contains the "Last Name"! Then, continuing with line 660, we have printed out for us the contents of CO$ which prompts us to enter the "Corrected entry", and then accepts our corrected entry into string variable C$ (through the line input statement), and so we have replaced the old contents of C$ (in error) with the corrected value. We then RETURN to the first statement after the ON-GOSUB statement which is at the end of line 630, namely GOTO 380. Line 380 now prints out the entire list of entries again for us, only now the "Last Name" has been corrected!

The above routine has looped, and would continue to loop as described, until the user is satisfied that ALL is correct and enters a zero, in which case line 590 branches to line 730. Line 750 concatenates most of the variables, in the desired order, into X$, inserting the desired delimiters (\) at the proper places. Line 760 checks to see if P$ was or was not a null. If NO phone number was entered, then the LEN of P$ is zero and we branch to line 770. IF there was data stored in string variable P$ (a phone number was entered), then the LEN of P$ is NOT zero and line 760 adds a delimiter and then P$ onto the present contents of X$. (X$=X$+"\"+P$). Hence, we now have the entire contents of the present record, properly formatted, as the contents of X$. Now, line 770 writes the contents of X$ to file TEMP on disk. In line 780 we have another counter, this one called S1, and you can see that this keeps a running total of the number of NEW records added at this particular time. Line 790 then loops back to line 250 where we are all ready for the next unit to be entered.

The above loop continues until the user types "DONE" in response to the prompt at line 250, in which case, "DONE" is stored in string variable A$ at line 260. Line 270, which checks each entry here, now finds its IF statement to be TRUE and we branch to line 800 which is the beginning of the END! Line 800 CLOSEs file TEMP, then renames it to T$. Lines 820 through 840 display for the user the results of our record counting, and then END.

In both of the above programs, there are many lines devoted to remarks. And, many times, for the sake of clarity, I made NO attempt to combine statements on fewer lines. Obviously, with some care, the above 2 programs can be shortened considerably. To show you it IS possible for me to write a short program that does do something, I give the 3rd and last program. It is called PRDATA.BAS, and like the other 2 programs, is written in CP/M MBASIC. HDOS users can alter it as explained previously, but then it will have to take up one more line! The program simply prints out from a disk file, data which has been produced by MAKESEQ.BAS or an editor, and prefaces each record with a number which IS the absolute position of that record in the data file.

Last, I would like to add a few brief remarks on troubleshooting MBASIC programs. What do you do when MBASIC comes up with an ERROR message, and you haven't the vaguest idea of why? The easiest and fastest way to find out why (in the vast majority of such problems) is the judicious use of the STOP statement. Simply put a temporary line number with the statement STOP just BEFORE the line (if the line in question is the target of a GOTO or a GOSUB, put STOP: at the beginning of the line in question) that seems to be causing the difficulty. Then, when the program stops (remember now, it will stop just before the problem), you can question MBASIC as to the contents of the various numerical and/or string variables that are present on the problem line. You simply type PRINT X<cr>, or print X$<cr>, etc. depending on whatever variables are there.

Many times you will find a value within the named variable that couldn't possibly work. This may well lead you elsewhere in your program to see just how such a value got in the variable. And, you may then have to take corrective action by using other statements, or by using an "IF THEN" statement to eliminate a possible offender. In any event, once you know WHY you are getting an error, it then becomes much easier to fix the problem. You should also check the manual and reread about the statement causing the problem. You may find that what is being attempted is just not feasible. So, from my own experience, I urge you to try this method of troubleshooting.

I will cover CP/M MBASIC random files in my last article of this series on "GETTING STARTED with CP/M".

CP/M is a registered trademark of Digital Research

MBASIC refers to "Microsoft Basic-80", a product of Microsoft, Inc.

```
10 'LABEL.BAS  CP/M MBASIC pgm for printing labels, 1 or 2 or 3 or 4 up
20 '           Also shows how to handle optional line of address
30 '           See text for changes for HDOS MBASIC
40 '   Copyright 1982 (c) by William N. Campbell, M.D.
50 '                    (0 = zero; O = 'OH')
60 'Assume data file consists of records each containing 6 OR 7 "fields".
70 'Each field is delineated by a reverse slash. A typical record might look
80 'like this -> LAST\FIRST\MR\STREET\OPTIONAL ADDRESS\CITYSTATEZIP\PHONE NUMBER
90 '
100 REM CLEAR 2500:' remove 'REM' from this line if HDOS MBASIC & see text!
110 DEFINT A-Z:'All numeric values defined as integers.
120 N=3:' Set N to number of labels across you are printing (1, 2, 3, or 4).
130 LM=5:' Set LM to number of spaces before 1st printing character of 1st label
```

```
140 T=20:' Set T to number of characters from LM first label to LM 2nd label.
150 LM=LM+1:' fix CP/M MBASIC 'bug'; delete this line for HDOS MBASIC-see text
160 R=T:' Here, we save original value of T in variable R
170 ' **IMP** -> Set printer's width to MINIMUM of LM + T*N + 1. (Ex. 5+60+1=66)
180 '            Note - width can be ANYTHING greater than minimum value.
190 ' **IMP** -> Program creating data file MUST disallow maximum line length
200 '            greater than T-1. Ex. If T=20 then NO label line's length
210 '            should be greater than 19 (else TABS thrown out of kilter).
220 '            See program MAKESEQ.BAS which does this for you.
230 GOSUB 730:' clear screen
240 PRINT:INPUT "Name of file to be printed (use UPPER CASE)... ";Y$:PRINT
250 OPEN "I",1,Y$:'  open this seq data file for input
260 FOR I=0 TO N-1:' N = number of labels across printout paper
270 IF EOF(1) THEN FLAG=-1:GOTO 430
280 LINE INPUT #1,X$
290 'find location of each delimiter, then place in separate variables A,B,etc
300 A=INSTR(1,X$,"\"):B=INSTR(A+1,X$,"\"):C=INSTR(B+1,X$,"\")
310 D=INSTR(C+1,X$,"\"):E=INSTR(D+1,X$,"\"):F=INSTR(E+1,X$,"\")
320 IF F=0 THEN F=LEN(X$)+1:X$=X$+"\":' need total of 6 '\'s - see text
330 'isolate each field and assign to subscripted variables (I= the subscript)
340 A$(I)=LEFT$(X$,A-1):B$(I)=MID$(X$,A+1,(B-1)-A):'last name, first name
350 C$(I)=MID$(X$,B+1,(C-1)-B):D$(I)=MID$(X$,C+1,(D-1)-C):'salut., street add
360 E$(I)=MID$(X$,D+1,(E-1)-D):'optional 2nd line of address
370 F$(I)=MID$(X$,E+1,F-E-8):'town or city
380 G$(I)=MID$(X$,F-7,2):'2 letter abbreviation of state
390 H$(I)=MID$(X$,F-5,5):'5 digit zip code
400 F1$(I)=F$(I)+" "+G$(I)+" "+H$(I):' F1$(I) now contains city, state, zip
410 IF LEN(E$(I))=0 THEN E$(I)=F1$(I):F1$(I)="":'if NO opt address then this
415 REM P$(I)=MID$(X$,F+1):' remove REMs from this line & 665 to print phone #
420 NEXT I
430 T=R+LM:LPRINT TAB(LM);
440 FOR J=0 TO I-1
445 REM IF I-1=-1 THEN 680:' Remove REM from this line if HDOS MBASIC-see text
450 LPRINT C$(J)+" "+B$(J)+" "+A$(J);TAB(T);:'print salut, 1st & last name, N up
460 S=S+1:' increment counter by 1
470 T=T+R
480 NEXT J
490 LPRINT
500 T=R+LM:LPRINT TAB(LM);
510 FOR J=0 TO I-1
520 LPRINT D$(J);TAB(T);:'print 1st line street address N up
530 T=T+R
540 NEXT J
550 LPRINT
560 T=R+LM:LPRINT TAB(LM);
570 FOR J=0 TO I-1
580 LPRINT E$(J);TAB(T);:' print opt line address, else city, state, zip N up
590 T=T+R
600 NEXT J
610 LPRINT
620 T=R+LM:LPRINT TAB(LM);
630 FOR J=0 TO I-1
640 LPRINT F1$(J);TAB(T);:'print city, state, zip; if NO opt then null, N up
650 T=T+R
660 NEXT J
665 REM T=R+LM:LPRINT TAB(LM);:FOR J=0 TO I-1:LPRINT P$(J);TAB(T);:T=T+R:NEXT J
670 LPRINT:LPRINT:LPRINT:' vary # of LPRINTs to get to next row of labels
680 IF FLAG=-1 THEN 700
690 GOTO 260
700 CLOSE
710 PRINT:PRINT "There were a total of";S"labels printed."
720 END
730 FOR K=1 TO 24:PRINT:NEXT K:RETURN:'clear screen
740 ' H19/89 users substitute PRINT CHR$(27)+"E":RETURN for above line.
```

```
10 'MAKESEQ.BAS  An MBASIC pgm to create or add to mailing list; allows for 3
20 '             OR 4 lines for name and address, and optional phone number.
30 '             Program also demonstrates 'error handling routines'.
40 '      Copyright 1982 (c) by William N. Campbell, M.D.
50 '                (0 = zero; O = 'OH')
```

```
60 REM CLEAR 2500:' remove 'REM' from this line if HDOS MBASIC
70 GOSUB 950:' clear screen
80 ON ERROR GOTO 860:' see text for discussion of the Error Handling Routines
90 DEFINT A-Z:PRINT
100 T=19:' ****** Change T to maximum length any label line can be. ******
110 '      ****** (Not including left or right margins of label printout) ******
120 PRINT "Do NOT use '.EXT' in naming file. Ex. Use DATA, NOT DATA.DAT!":PRINT
130 INPUT "Sequential file name (Ex. DATA)... ";T$
140 PRINT
150 IF LEFT$(T$,1)<"A" THEN PRINT "Illegal Entry - Enter again -":GOTO 90
160 IF LEFT$(T$,1)>"Z" THEN PRINT "USE UPPER CASE - Enter again - ":GOTO 90
170 IF INSTR(1,T$,".")<>0 THEN PRINT "No extensions - Enter again -":GOTO 90
180 OPEN "I",1,T$:'open a pre-existing data file
190 OPEN "O",2,"TEMP":'if pre-existing file we put it in TEMP file
200 PRINT:PRINT "Please wait for next message..... ":PRINT
210 IF EOF(1) THEN CLOSE #1:KILL T$+".OLD":NAME T$ AS T$+".OLD":GOTO 250
220 LINE INPUT #1,X$
230 PRINT #2,X$:S=S+1:' S=counter for number of preexisting records.
240 GOTO 210
250 GOSUB 950:PRINT:PRINT "If done, type DONE & hit RETURN key":PRINT
260 LINE INPUT "'Salutation (Mr Mrs Ms)'.... ";A$
270 IF A$="done" OR A$="DONE" THEN 800
280 LINE INPUT "First Name & Middle Initial.... ";B$
290 LINE INPUT "Last Name... ";C$
300 PRINT:PRINT "You may type 1 or 2 lines of address, PLUS City, State, Zip
310 PRINT:LINE INPUT "1st line of address... ";D$
320 LINE INPUT "Opt 2nd line address, OR just RETURN.. ";E$
330 LINE INPUT "Town or City.... ";F$
340 LINE INPUT "State (2 letter abbreviation)... ";G$:IF LEN(G$)<>2 THEN 340
350 LINE INPUT "Zip Code (5 digit)... ";H$:IF LEN(H$)<>5 THEN 350
360 LINE INPUT "Opt phone number, OR just RETURN.. ";P$
370 ' Let user correct any mistakes in entries, tell user if any lines too long.
380 GOSUB 950:PRINT "Here are your entries -":PRINT:' clear screen, print msg
390 PRINT "1 Salutation..... ";A$
400 PRINT "2 First Name & Middle Initial.... ";B$
410 PRINT "3 Last Name..... ";C$
420 M=LEN(A$)+LEN(B$)+LEN(C$)+2:IF M>T THEN GOSUB 930:PRINT CHR$(7);
430 PRINT "4 First line of address..... ";D$
440 M=LEN(D$):IF M>T THEN GOSUB 940:PRINT CHR$(7);
450 PRINT "5 Optional 2nd line of address... ";E$
460 M=LEN(E$):IF M>T THEN GOSUB 940:PRINT CHR$(7);
470 PRINT "6 Town or City..... ";F$
480 PRINT "7 State (2 letter abbrev.)..... ";G$
490 PRINT "8 Zip Code (5 digits)..... ";H$
500 M=LEN(F$)+LEN(G$)+LEN(H$)+2:IF M>T THEN GOSUB 930:PRINT CHR$(7);
510 PRINT "9 Optional phone number..... ";P$
520 PRINT:'   See text for explanation of following input routine
530 PRINT "Type number at left to correct individual item, 0 if all OK... ";
540 Z$=INPUT$(1):IF Z$=CHR$(27) THEN 540
550 IF Z$<"0" THEN 540
560 IF Z$>"9" THEN 540
570 PRINT Z$
580 Z=VAL(Z$)
590 IF Z=0 THEN 730:' If Z=0 then ALL current entries OK
600 W$="Current entry ------> "
610 CO$="Corrected entry-----> "
620 PRINT
630 ON Z GOSUB 640,650,660,670,680,690,700,710,720:GOTO 380
640 PRINT W$;A$:PRINT:PRINT CO$;:LINE INPUT A$:RETURN
650 PRINT W$;B$:PRINT:PRINT CO$;:LINE INPUT B$:RETURN
660 PRINT W$;C$:PRINT:PRINT CO$;:LINE INPUT C$:RETURN
670 PRINT W$;D$:PRINT:PRINT CO$;:LINE INPUT D$:RETURN
680 PRINT W$;E$:PRINT:PRINT CO$;:LINE INPUT E$:RETURN
690 PRINT W$;F$:PRINT:PRINT CO$;:LINE INPUT F$:RETURN
700 PRINT W$;G$:PRINT:PRINT CO$;:LINE INPUT G$:IF LEN(G$)=2 THEN RETURN ELSE 700
710 PRINT W$;H$:PRINT:PRINT CO$;:LINE INPUT H$:IF LEN(H$)=5 THEN RETURN ELSE 710
720 PRINT W$;P$:PRINT:PRINT CO$;:LINE INPUT P$:RETURN
730 '  Now we put ALL fields into X$ by concatenation and we separate
740 '  each field with a '\' as a delimiter.
```

Vectored to page 27

New HUG Software

SOFT-SECTORED DISKS NOW AVAILABLE FROM HUG

In a continuing effort to offer the lastest in software to the membership of the Heath Users' Group, HUG now has available several selected products in **40 track single-sided soft-sectored format.** These products will operate with CP/M by Digital Research when used with the Z-89-37 soft-sectored controller card and either the H77, Z87 or the Z37.

The HUG part numbers remain the same for these new disks with the addition of the "-37" after the regular HUG number (e.g. 885-1211-37). If you are using the H/Z89 or the Z90 with only the soft-sectored controller, order all of the disks you select being sure to place the **-37** after your selection to ensure that you receive the correct product.

The following disks are available in soft-sectored format:

885-1206-37	**885-1210-37**	**885-1214-37**
885-1207-37	**885-1211-37**	**885-1215-37**
885-1208-37	**885-1212-37**	**885-1217-37**
885-1209-37	**885-1213-37**	

****NOTE:** Check the HUG parts list for a description of these products. Watch for the expansion of this list in the near future.

885-1118 HUG Payroll (HDOS) $60.00

Introduction: The HUG Payroll Package consists of documentation and one disk. The programs may be subdivided onto other disks depending on the type, and how many drives you have available. The Payroll Package can handle 30 employees on a single low density drive while maintaining up to 100 employees on two low density drives such as those used in the H77, Z87 or H-17.

Hardware Requirements (HDOS): The HDOS version of the HUG Payroll Package requires the use of a 64K H/Z89 (or Z90) or a 64K H8 using the H/Z19 terminal. A printer that can be interfaced with your computer is a must. The H/Z25 is highly recommended since the Payroll Package was developed using this printer.

Software Requirements (HDOS): The HUG Payroll Package operating under the Heath Disk Operating System (version 2.0) requires only MicroSoft BASIC (version 4.82) and an appropriate LP.DVD for your printer to obtain proper results.

Disk Contents:

PAYNEW .BAS	**PAYCHECK.BAS**	**PAYCLOSE.BAS**
PAYMENU .BAS	**PAYDEDUC.BAS**	**PAYDELET.BAS**
PAYADD .BAS	**PAYDISPL.BAS**	**PAYTABLE.BAS**
PAYDAY .BAS	**PAYCHANG.BAS**	**PAYSORT .BAS**
PAYCAL .BAS	**PAYREPOR.BAS**	**TABLES .TAX**

** **NOTE:** The HDOS version of the HUG Payroll contains a **PROLOGUE.SYS** for automatic startup during normal use.

Program Features: The HUG Payroll Package contains several powerful features that allow for user friendly operation including a startup routine for entering new data. The program automatically checks for previously entered information so that you are ensured of proper results as you proceed. **A Master Menu** is used and linked to several sub-menus for ease of operation. Complete instructions are supplied with each disk. As suggested by some of the program names, the HUG Payroll allows the user to track an employee using several pay methods including: hourly, bi-monthly, monthly, etc. Also included is a program that will allow you to change information on a given employee. You have available the standard deductions along with several variable deduction areas for items such as credit unions, United Fund, employee contributions, etc.

Printing of reports is a very important feature of any software package. The HUG Payroll Package allows you to print a quarterly report, yearly report, checks, W-2 forms, etc. When using the H/Z25, you will be able to reproduce the H19/H89 screen on paper. This unique feature decreases the confusion that sometimes accompanies the transition from computer to hardcopy. The HUG Payroll allows sorting of your employee list to ensure continual accuracy as you build your records.

Comments: As with any business package, the HUG Payroll will have its limitations. However, the software in this case is very solid. Problem areas, such as changing tax deductions or using the tax table have been simplified using specific programs that can be updated as necessary. This particular piece of software has been used by several companies for tests to ensure the best results with minimal problems.

COMING SOON: The HUG Payroll Package will be available shortly under the **CP/M Operating System** by Digital Research. This package will have the same features as presented here for HDOS.

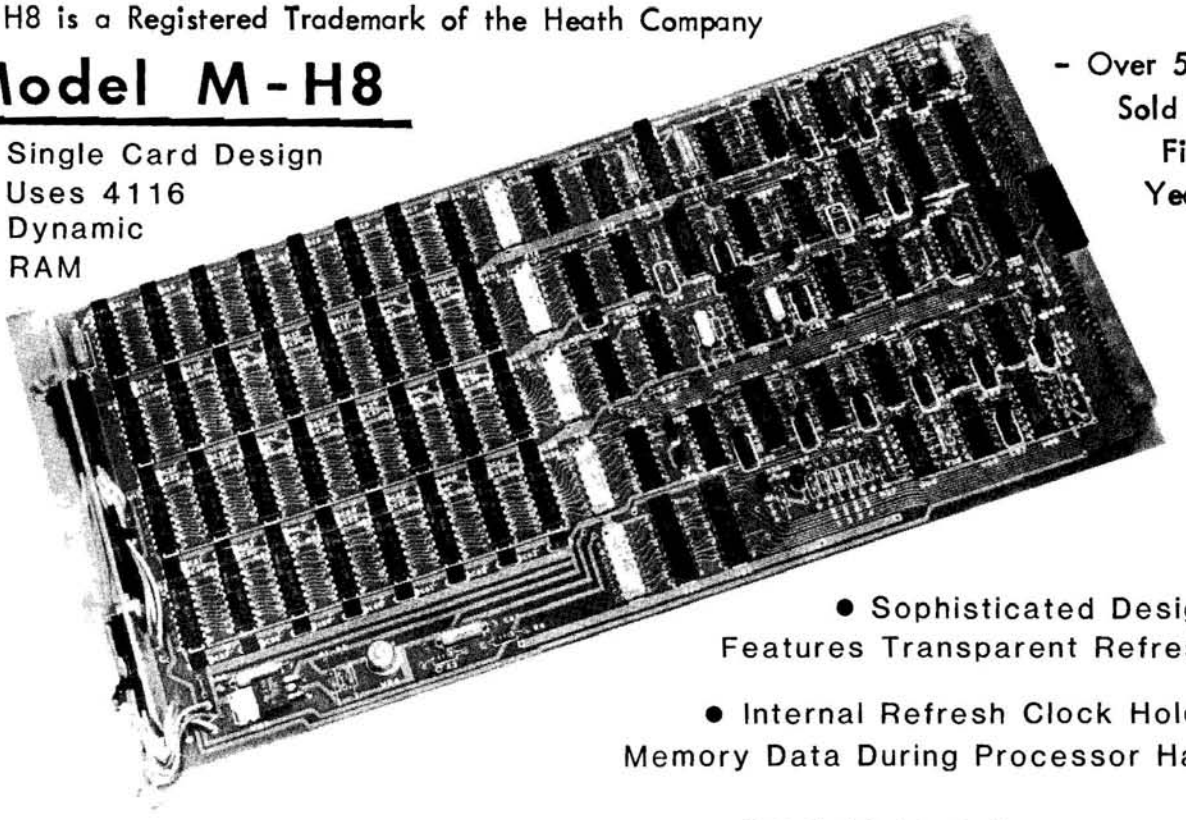
Vectored to page 25

64K MEMORY for the H8*

* H8 is a Registered Trademark of the Heath Company

Model M-H8

- Single Card Design Uses 4116 Dynamic RAM



- Over 500 Sold in First Year!

- Sophisticated Design Features Transparent Refresh
- Internal Refresh Clock Holds Memory Data During Processor Halt

NEW PRICES:

Assembled - Without IC Sockets	Kit - IC Sockets Included	
$ 500	$ 415	64K
440	360	48K
380	305	32K
320	250	16K

16K Memory Expansion Kit - $ 60
(Memory Chips, Sockets, Capacitors)

Set of 8 Tested Memory Chips - $ 50

SPECIALS!

64K Kit -
Without Memory Chips
$ 225

64K Assembled -
Without Memory Chips
(Sockets Installed
for Memory Chips)
$ 300

Printed Circuit Board -
With Documentation
$ 50

Guaranteed To Work With Any Existing or New Heath H8 Products
— Upgrade Modification Instructions Will Be Issued As Required

Call or Write For Free Brochure • Call 714-830-2092 Ask For Bill Perry
Get On Our Mailing List For New H8 Product Announcements

TRIONYX ELECTRONICS, INC.

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

DG IS Heath/Zenith

...with the F

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 $249.00. The acclaimed FP8 monitor package is now included with the DG-80 CPU!

***DG-64D/64K RAM Board**
Reliable, Low Power, High Capacity Bank-selectable RAM
Priced from $333.00 (0K) to $399.00 (64K)

***DG-64D5 64K/5 volt only RAM Board**
SUPER low power, Reliable, High Capacity, Bank Selectable RAM from $333.00 (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 DG STATIC 64

IF YOU STILL BELIEVE YOU WOULD RATHER USE STATIC MEMORY...OR IF YOU NEED A PROM/EPROM BOARD...
OR IF YOU NEED MORE ROOM ON YOUR H8 MOTHER BOARD...
OR IF YOU HAVE BEEN WAITING FOR STATIC MEMORY FOR THE H8 TO BECOME REASONABLY PRICED.
WE HAVE THE ANSWER—THE STATIC 64 RAM BOARD

FULLY STATIC.
SUPER LOW POWER—Power dissipation typically less than 4 watts (less than any memory board available for the Heath H8). Uses Single Supply 5-Volt memory devices.
CAPACITY—Up to 64K RAM OR Up to 64K EPROM (type 2716/2516) OR any combination of RAM and EPROM up to 64K.
SPEED—4MHz with NO wait states.
Compatible with the OLIVER ADVANCED ENGINEERING PROM Programmer Model #PP-2716.
Bank-select fully compatible with the DG-64D—Hardware selectable in 8K increments/software selectable in 16K increments.
Bank-select port addressable to any of the available 256 I/O ports using solderless jumpers.
On-board ROM-disable port for use with the DG-FP8 Monitor Package
Fully assembled and tested. Priced from $299.00 (0K) to $599.00 (64K). 16K Chip Sets: $125.00.

H8/H89 Software

Software Support for HDOS 2.0

*** Disk Management Utility Package**
Includes "Universal Dump", Intelligent Disk Dump Utility; "Universal Dup", Disk Copy Utility; and "BAD", Bad Disk Recovery Utility. $39.95 (Source: add $40.00)

*** Archive**
Space saving diskette back-up utility. For use with 5¼" or 8" disk systems. $39.95 (Source: add $40.00)

*** SYSCMD/plus**
A must for the serious HDOS user. Enhanced HDOS 2.0 system command processor provides extended commands and capabilities. $39.95 (Source: add $30.00)

*** Preload**
Support utility for systems using multiple, bank selectable memory boards. $29.95 (Source: add $20.00)

*** Advanced H17/H77 Driver**
Software driver for operation of double-sided and/or double-track double density drives with the H-17 and H-77 5¼" disk systems. Low cost alternative to high priced double density disk controllers. $19.95 (Source included).

Computer Support
uture Built-In

SUPER 89

TURN YOUR H/Z 88/89 INTO A PROFESSIONAL QUALITY COMPUTER
HIGHER RELIABILITY NOW — MORE EXPANDABILITY FOR THE FUTURE

FEATURES	DG-SUPER 89	H/Z 88/89
USER MEMORY The DG Super 89 comes standard with 64K of user RAM "on-board". This configuration can be increased up to 256K of bank selectable/write protectable RAM.	64K-256K	16K-64K
Ø ORIGIN MODIFICATION No further modification, such as required on the H/Z-89, is necessary to operate in either a standard Heath/Zenith CP/M or HDOS 2.0 Operating System environment.	NOT NECESSARY	ADD-ON
CP/M-HDOS COMPATIBLE	YES	REQUIRES MODIFICATION
PERIPHERAL EXPANSION SLOTS DG Electronics has made provision in the design of the unit not only for compatibility with the standard factory expansion slots, but also for future expansion by doubling the number of available expansion slots on the unit to 6 instead of the standard 3.	6	3
ON-BOARD AMD9511 For those users who perform large amounts of arithmetic computations the DG Super 89 has provision on-board for use of the AMD 9511 arithmetic processor.	YES (PURCHASE SEPARATELY)	NO
CPU CLOCK FREQ. The CPU in the DG Super 89 operates at twice the speed of the standard H/Z-89.	4MHz+	2.048MHz
MULTI-USER CAPABILITY With up to 256K of bank selectable RAM on board the DG Super 89 offers the option of MULTI-USER CONFIGURATIONS of up to 4 users.	YES	NO
ENHANCED MONITOR Enhanced monitor supports the advanced features of the Super 89.	YES	NO
REAL TIME CLOCK The DG Super 89 comes standard with an on-board real time clock.	YES	NO
PARITY CHECK ON RAM For those who are sticklers for accuracy, the DG Super 89 has parity check to make the user aware of errors occuring in the RAM during use.	YES	NO
SERIAL PORTS ON BOARD The DG Super 89 offers an additional serial I/O port for greater convenience and flexibility.	2	1

Now you can have all of the features in your H/Z-89 that you have always wanted. High speed and greater expandability are only the beginning of what our NEW DG SUPER 89 has to offer. DG Electronic Developments Co. has given the "89" capabilities of fast number crunching and data verification through parity. We have incorporated into the DG SUPER 89 such necessary items as 64K of user RAM, a powerful Keyboard monitor, and CP/M compatibility, items others require you to "add-on". Add to these features an extra serial port, a realtime clock, three more peripheral expansion slots, and multi-user capability and you have the computer that you really wanted to begin with; for a lot less than you would think.

Compatible with all currently available Heath/Zenith hardware devices.

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 (charge only) 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.

SMASH THE '89's 64K RAM LIMIT!

128K RAM BOARD — $595

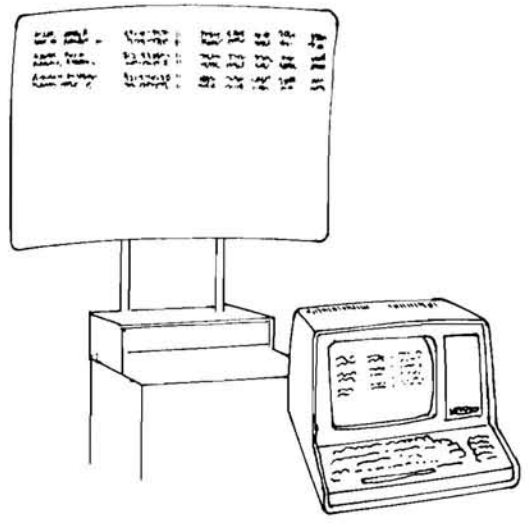
order no. 77318

Now have 176K of fast dynamic RAM at your disposal in your Heath/Zenith '88, '89, or '90.

Featuring:

- 128K of 200nSec Dynamic RAM, added to the 48K on your CPU board, for a total of 176K.
- Full compatability is retained for MMS and Zenith CP/M® (64K), as well as HDOS (56K).
- Versatile bank switching technique supports three full MP/M II® compatible banks.
- 112K "Electronic Disk" BIOS module for MMS CP/M included, complete with source code.
- Ultimeth Corp. supplies HDOS support.

Delivery beginning March 15, 1982.



VIDEO OUTPUT — $79

order no. 77319

Add an industry standard video output to allow reproduction of the CRT image on another monitor or projection TV system to enhance the usefulness of your terminal in classroom and other educational applications. Allows simultaneous viewing of the display in group situations.

The auxillary display unit should be capable of high resolution for satisfactory performance.

Delivery beginning March 15, 1982.

LOWER PRICE! 16K RAM — $125

order no. 77311

Magnolia's 16K Add-On RAM board has been so popular we lowered the price.

DOUBLE DENSITY DISK CONTROLLER — $595

order no. 77316

Complete hardware and software support for FOUR 8" single or double sided drives and FOUR 5" Single or Double sided, 48tpi (40 track) or 96tpi (80 track) drives, in addition to the three 5" drives supported by your Heath/Zenith controller.

Plus, the obvious advantage of being able to use Single Density 8" media for program and data interchange.

Full compatability is retained with MMS support of the '89s built-in 5" floppy, as well as several Winchester hard disk subsystems.

The package includes:

- Double Density Controller Card.
- Cables for both 5" and 8" disk drives
- CP/M 2.2 on both 5" and 8" media
- New I/O Decoder and Monitor PROMs
- Ultimeth Corp. supplies HDOS support.

If your '89 isn't ORG-0 CP/M compatible yet, our modification is available for $50 additional.

DOUBLE DENSITY SUBSYSTEMS

Dual 8"	DS	48tpi (2.4M)	order no. D8DS	$2695
	SS	48tpi (1.2M)	order no. D8SS	$1995
Single 5"	SS	48tpi (162K)	order no. S540SS	$ 945
	DS	48tpi (343K)	order no. S540DS	$1095
	DS	96tpi (700K)	order no. S580DS	$1295
Dual 5"	DS	96tpi (1.4M)	order no. D580DS	$1995

COMPLETE SYSTEMS

As well as manufacturing enhancements for the '89 (also '88 and '90), we are a Zenith Data Systems OEM, and have all of their hardware and software products available as well. We can provide a completely integrated system, combining the best Zenith products with our own to provide the exact system capabilities to best satisfy your requirements.

ORDERING INFORMATION

Our products are available from many Heathkit Electronic Centers and independent computer stores throughout the United States. If your local dealer doesn't stock our products, you may order direct or request further information by calling our Sales Department on our toll-free number, (800) 426-2841.

CP/M and MP/M II are registered trademarks of Digital Research, Pacific Grove, CA

MAGNOLIA MICROSYSTEMS

2264-15th Avenue W. • Seattle, WA 98119

(206) 285-7266 (800) 426-2841

LIVINGSTON LOGIC LABS

BIOS–80

Now you can get up to 392K storage on a single hard-sectored 5" disk while still maintaining complete compatibility, and WITHOUT buying an expensive double density disk controller. BIOS-80® is a specially modified Heath/Zenith CP/M 2.2X03 BIOS which supports any combination of 40 and 80 track single- and double-sided 5" drives using the Heath H-17 controller. BIOS-80 allows booting from any disk, as well as reading of 40 track disks in 80 track drives. BIOS-80 does for CP/M what the Ultimeth "SY.DVD" does for HDOS. Requires H-8 or H-89 and CP/M 2.2X03.

BIOS-80—$35

8" Disk Controllers

THE low cost way to add HDOS and CP/M compatible 8" drives to the H-8 and H-89 computers! These controllers support up to four single- or double-sided single-density 8" drives for up to 2 Megabytes of FAST on-line storage. They provide complete CP/M and H-47 compatibility. Requires either H-89 or H-8 with Z-80 CPU (any model). Heath/Zenith CP/M 2.2X03 BIOS supplied at no charge. HDOS 2.0 device driver is available for $35. Monitor ROMs to allow booting from 8" drives are available; write or call for further information.

FDC-Z8—$225 **FDC-Z89—$200**

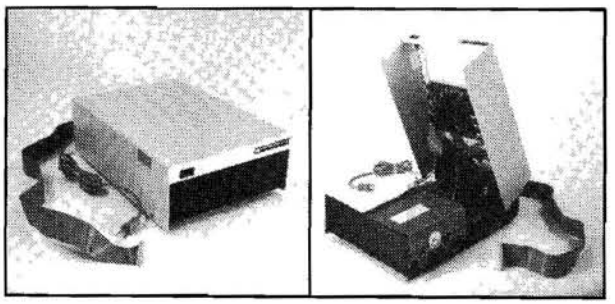
ALSO AVAILABLE

HDOS 2.0 device driver for CDR Systems FDC-880H double-density disk controller, allowing use of any 5" and 8" formats with complete H-47 and H-37 compatibility. Permits use of single-density disks in double-density drives, and includes automatic detection of track and recording densities. **CDRDVD—$40**

CP/M 2.2X03 support software for Magnolia 128K RAM expansion for the H/Z-89. Allows use of additional memory capacity as either a "solid-state disk" or cache memory, with dramatically reduced execution and loading times. **CACHE22—$40**

ORDERING INFORMATION—Send all orders to Livingston Logic Labs, P.O. Box 5334, Posodeno, CA 91107. Payment by personal check or money order. No cash, please. Add $2 per order for shipping and handling, $5 for overseas orders. California residents add 6% tax. Please allow 2-4 weeks for delivery. Write for free catalog.

LIVINGSTON LOGIC LABS
PO Box 5334, Pasadena, CA 91107
(213) 792-4763



NOW 12 MEGABYTE (CDR-10M) $3195
and 6 MEGABYTE (CDR-5M) $2495

WINCHESTER SYSTEM
For the Heath/Zenith Computer

Systems complete with software case, power supply & signal cable.
Runs with CP/M, on the H/Z89, Z90 & H8 (with Z80 card).

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

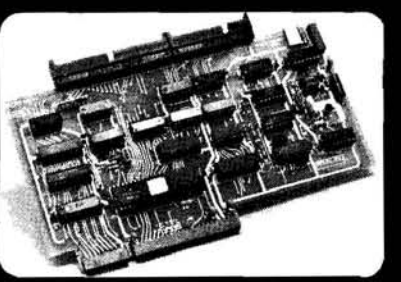
CP/M is a trademark of Digital Research. Heath, H8, H89 are trademarks of Heath Corporation. Zenith, Z89, Z90 are trademarks of Zenith Data Systems.

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

Contact:
C.D.R. Systems Inc.
7667 Vickers St. Suite C
San Diego, CA 92111
Tel. (714) 275-1272

1 CONTROLLER

FOR 8" & 5.25" DRIVES



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

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

NEW PRICE $495

includes controller board CP/M boot prom, I/O decoder prom, hardware/software manuals BIOS source listing.

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

Contact:
C.D.R. Systems Inc.
7667 Vickers St. Suite C
San Diego, CA 92111
Tel. (714) 275-1272

MONEY$WORTH

Provides a method of categorizing and summarizing expense and income transactions. Easy for the user since it's designed to minimize keystrokes. Additional flexibility is obtained by allowing the user to set-up category definitions in parameter files. Also provided is report generation capability for any range of months, and to any user-specified device. Included is a unique budget forecasting feature which provides the user with a 'spread-sheet' like display. The forecasting display can give the user an overview of the financial picture. Requires 48K RAM for HDOS, 56K RAM for CP/M and Microsoft Basic. PRICE $35.

qulk-ref

Rapidly creates a sorted cross-reference for Microsoft Basic application programs. It outputs statement number references, variables and all user-defined function references in a paginated report. Variables and function names will be displayed up to a maximum of 16 characters. Supplied in binary format and operates under HDOS version 1.6 or 2.0. The output may be directed to any HDOS supported device. Currently, 200 variables may be cross-referenced along with 2000 statement references. Requirements are 32K RAM and an H8 or H89 computer. PRICE $19.95

COMPACTA

A system allowing one to store an entire disk of files in a single file, thereby conserving disk space on high capacity drives. Perfect for backing-up data from a 5" disk to a higher capacity disk drive, such as dual-headed, 80 track units. COMPACTA allows this kind of back-up storage to be done easily and efficiently. An added advantage is that files from several different disks with identical names, can be stored on one disk without any chance of a name conflict. Requirements are 32K of RAM, HDOS 2.0 and an H8 or H89 computer. PRICE $24.95

UDUMP

The Universal Dump Utility Manipulation Program (UDUMP) modifies files using its own built-in screen editor. Totally menu driven for ease of operation. Editing may be done in ASCII, HEX or OCTAL formats, (track/sector or file basis) with all changes shown in reverse video. A 'Search Function' allows you to quickly find a specific string of characters. Any combination of flags, even locked ones may be removed. Also included is a program that will assist you in recovering files from a corrupted disk. Requires HDOS 2.0 with 48K RAM and an H8/H19 or H89. PRICE $34.

CAVERNS OF THE DOOMED

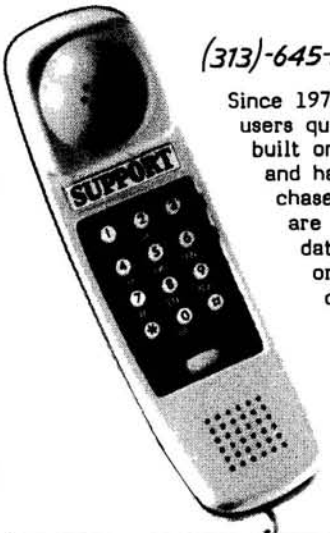
A fun-packed adventure game with moving graphics and voice output! It operates in a REAL-TIME mode and leads you into 30 different caverns each with a dangerous little creature lurking about. You'll also run into such things as pits, spear traps, ghosts, flying boomerangs and even poison arrows. And, if you own a Type 'N Talk synthesizer, the program will automatically 'talk' to you during your journey about impending dangers. A non-talking version is also included. Requirements are 48K for non-talking, 56K RAM for talking version, HDOS 1.6 or 2.0. PRICE $22.95-requires MBasic).

TERMINAL

A menu-driven, intelligent screen-oriented communication interface for linking to remote computers while retaining control to many of HDOS's commands! Some options available are: mounting, dismounting, and reseting disks, renaming and displaying files, directory listings and even deleting files. An AUTOLOG feature is built-in which allows 16 unique command lines to be stored, for sending remote systems a specific sequence of commands in order to logon. Perfect for user's with Hayes Smartmodems. Linking to a remote system can be as simple as selecting the proper Autolog! TERMINAL also provides a TRACE function which selectively directs output to your hard-copy device with just one keystroke. When communicating to a remote computer, you may control what is transmitted or received with one keystroke, while the 25th line of the H19/H89 displays the status of TERMINAL. Requires an H8/H19 or H89 with 48K RAM. PRICE $25.

ZENITH

As a Zenith dealer, we stock many of the original hardware and software products offered by Heath & Zenith, all at reduced prices. A sample of some of the items include CP/M, Despoolers, Microsoft Basic, 16K expansion boards, DataStar, Supersort, Z25 printers, Pascal, Supercalc, CBASIC, Fortran, Z19 terminals, Wordstar, Magic Wand, Z90's and more. We also carry some selected non-Zenith software and hardware, such as the Spellbinder word processing system, Spellguard, Supervyz menu generator for CP/M, Type 'N Talk synthesizers, Housemaster voice recognition boards and low cost 8" disk controllers for the H8 and H89. We stock Tandon 48 & 96 track per inch, single or dual-sided 5" and 8" disk drives and enclosures - at great prices with full support! Give us a call to check all of our current prices.



(313)-645-5365

Since 1978, The Keyboard Studio, Inc. has been dedicated to bringing Heath/Zenith computer users quality software and hardware products at reasonable prices. Our reputation has been built on fast delivery, low prices & good service. For a complete listing of all our software and hardware, please call or write for our free 16 page catalog. Our software may be purchased from your local Heathkit Electronic Center or direct from us. Registration forms are included with all software products, so that we may keep you informed of any updates made to the products. Payment may be in the form of check or money order (in order to keep our prices as low as possible, we do not accept credit cards). Please add $2.00 shipping and insurance on orders under $100 in the continenental United States Orders over $100, please add $5 shipping. C.O.D. orders are available for a $2 additional charge (phone orders welcome). Foreign orders please call or write for shipping charges. We ship most items within 24 hours upon receipt of your order.

Keyboard Studio inc.

125 Aspen, Birmingham, MI 48009

Vectored from page 16

885-8007 EZI-TRANSfer $30.00

Introduction: EZI-TRANSfer is a multi-function utility that makes the transfer of individual files from a source disk to a destination disk an easier task. With the use of the **Alternate Keypad Mode** and the **Special Function Keys**, copying of files, file deletion, mounting, and remounting functions are moved above the operating system level for fast user friendly characteristics.

Author: Dale Grundon

Hardware Requirements: EZI-TRANSfer requires the use of the H/Z89 or the H8/H19 with at least 32K of random access memory installed. Further, this program will support a system with any combination of two disk drives (H17, H77, Z87, etc.). The program has been tested on hardware modified to operate at 4MHz.

Software Requirements: EZI-TRANSfer requires the **Heath Disk Operating System** (HDOS version 2.0). An HDOS version 1.6 disk may be used **with** EZI-TRANSfer, as long as the program was entered using HDOS version 2.0.

Operating Overview: EZI-TRANSfer can operate in stand-alone mode when drive SY0: is used as the source or the destination drive. You may select the use of the "default drives", SY0: and SY1:, or any other valid set of drives you choose for the use of the program. Any disks currently mounted on the drives to be used will be dismounted by EZI-TRANSfer.

The two disks to be used are inserted and mounted, and the directory of the source drive containing the files to be transferred is displayed on the screen in brief format. You are then allowed to move the block cursor to the file to be operated on using the appropriate keys of the Alternate Keypad. While the cursor is located at the selected file, you may either copy or delete that file by pressing the appropriate **Special Function Key**. During a transfer, error checking is done on the destination disk to ensure that a file of the same name does not already exist, or that the disk is not full. Deletion of a file from the source disk will not occur without verification by the user. On-screen prompts inform you about actions occurring during both the copying and deletion process, and during disk changes.

Comments: Dale has used extensive graphics to indicate the operations of his EZI-TRANSfer program. His use of the Special Function Keys and the Alternate Keypad Mode make EZI-TRANSfer easy to use even for the beginner. EZI-TRANSfer is a fully documented product, with easy to follow instructions included with the disk. Dale has obviously put much thought into the program and the results of his efforts are excellent.

885-8008 Farm Accounting System $45.00

Introduction: The Farm Accounting System provides the farmer with timely and accurate management information. The system has most of the advantages of a full double entry accounting system without the confusing use of debits, credits and journal entries. Management reports are generated for cash receipts, cash expenses, capital purchases and liabilities, and an income statement. The reports show both a monthly and year-to-date figure along with the budget figures.

Author: Danny Jones

Hardware Requirements: The Farm Accounting System Requires the H/Z89 or the H8/H19 with at least 48k of random access memory installed. A line printer capable of 132 characters/line, or an 80 character/line printer capable of compressed print, is necessary to obtain proper results. Two disk drives are required if your system is comprised of the lower density drives such as the H77, H87 or H17.

Software Requirements: The Farm Accounting System makes use of the **Heath Disk Operating System** (HDOS version 2.0) and Microsoft BASIC (MBASIC version 4.82). The appropriate LP.DVD is necessary for your choice of line printers.

How the System Works: The heart of the system is a **code file** maintained on the diskette. There is an entry in this code file for each expense, income, capital purchase and liability item for which the farmer wishes to collect data. The code file is tailored to fit the individual farmers operation. Each entry in the code file is identified by a unique five digit code number. The code number is generic in that each of the digits in the code number represents something. Associated with the code number is a user assigned description and the amount budgeted for that item for the year, if you have entered an amount for the budget.

Transactions from the checkbook and the deposit slips are recorded on the transaction register forms. These transactions are entered into the system on a monthly basis through the cash entry program. There is an option in the cash entry program to **Print a Report** listing the cash receipts, cash expenses and non-cash transactions.

The **Report Print** program will print a cashflow report showing all the cash receipts with the month-to-date and the year-to-date figures along with year-to-date budgets. A similiar report is available for the cash expenses. Totals are printed by major and minor categories based on the type of code number. The major category is controlled by the first digit of the code number and the minor category is controlled by the second digit in the code number. Also available is a status report of all the capital purchases and liabilities as well as an income statement report.

Comments: The Farm Accounting System by Danny Jones is a fine example of a DBMS or **D**ata **B**ase **M**anagement **S**ystem. Danny has done an excellent job of documenting his work with a complete instruction manual included with the disk. His documentation steps the user through the programs available and even includes the typical error messages and their meanings. The Farm Accounting System has been tested in actual operation and appears to be a very solid well written software package.

HUG Product List

NOTE: The number in the REM # column refers to the issue of REMark containing a description of the software. Usually, it refers to the "New HUG Sofware" column, but it may refer to an article.

Part numbers shown in **bold print** are available in soft sector 5.25-inch format. Add -37 to the part number to order soft sector. For example, to order 885-1206 in soft sector, use 885-1206-37.

Part Number	Description	Selling Price	REM #
CASSETTE SOFTWARE (H8 and H88)			
885-1008	Volume I Documentation and Program Listings (some for H11)	$ 9.00	
885-1009	Tape I Cassette	$ 7.00	
885-1013	Volume II Documentation and Program Listings	$12.00	
885-1014	Tape II ASM Cassette H8 Only	$ 9.00	
885-1015	Volume III Documentation and Program Listings	$12.00	
885-1026	Tape III Cassette	$ 9.00	
885-1036	Tape IV Cassette	$ 9.00	8
885-1037	Volume IV Documentation and Program Listings	$12.00	8
885-1039	WISE on Cassette H8 Only	$ 9.00	
885-1057	Tape V Cassette	$ 9.00	
885-1058	Volume V Documentation and Program Listings	$12.00	
HDOS SOFTWARE (H8/H17 or H89 -- 5-inch only)			
MISCELLANEOUS COLLECTIONS			
885-1024	Disk I H8/H89	$18.00	6
885-1032	Disk V H8/H89	$18.00	8
885-1044	Disk VI H8/H89	$18.00	
885-1064	Disk IX H8/H89	$18.00	
885-1066	Disk X H8/H89	$18.00	10
885-1069	Disk XIII Misc H8/H89	$18.00	
GAMES			
885-1010	Adventure Disk H8/H89	$10.00	4
885-1029	Disk II Games 1 H8/H89	$18.00	8
885-1030	Disk III Games 2 H8/H89	$18.00	8
885-1031	Music 8 & 89 H8/H19 and H89	$20.00	25
885-1067	Disk XI Graphic Games .ABS and B H BASIC (H19/H89)	$18.00	12
885-1068	Graphic Games (H19/H89) *	$18.00	10
885-1088	Graphic Games (H19/H89) *	$20.00	14
885-1093	Dungeons and Dragons Game * Requires H89 or H8/H19	$20.00	16
885-1096	Action Games (H19/H89) *	$20.00	18
885-1103	Sea Battle Game (H19/H89)	$20.00	20
885-1111	HDOS MBASIC Graphic Games *	$20.00	23
885-1112	HDOS Graphic Games	$20.00	23
885-1113	HDOS Fast Action Games	$20.00	23
885-1114	Color Raiders and Goop (HA-8-3)	$20.00	23
UTILITIES			
885-1019	Device Drivers (HDOS 1.6)	$10.00	6
885-1022	HUG Editor (ED) Disk H8/H89	$15.00	20
885-1025	Runoff Disk H8/H89	$35.00	
885-1050	M.C.S. Modem for H8/H89	$18.00	
885-1060	Disk VII H8/H89 SUBMIT, CLIST, FDUMP, ABSDUMP, etc.	$18.00	
885-1061	TMI Cassette to Disk H8 only	$18.00	
885-1062	Disk VIII H8/H89 (2 disks) MEMTEST, DUP, DUMP, DSM	$25.00	
885-1063	Floating Point Disk H8/H89	$18.00	
885-1065	Fixed Point Package H8/H89	$18.00	10
885-1075	HDOS Support Package H8/H89	$60.00	
885-1077	TXTCON/BASCON H8/H89	$18.00	
885-1079	HDOS Page Editor	$25.00	15
885-1080	EDITX H8/H19/H89	$20.00	
885-1082	Programs for Printers H8/H89	$20.00	
885-1083	Disk XVI RECOVER, etc.	$20.00	11
885-1089	MACRO, CTOH, and misc Utilities	$20.00	20
885-1090	Misc. HDOS Utilities CCAT, HPLINK, AH, MBSORT, etc.	$20.00	22
885-1092	RDT Debugging Tool H8/H89	$30.00	14
885-1095	HUG SY: Device Driver HDOS 2.0	$30.00	18
885-1098	H8/HA-8-3 Color .ABS/.ASM	$20.00	19
885-1099	H8/HA-8-3 Color in Tiny Pascal	$20.00	19
885-1105	HDOS 2.0 Device Drivers MX-80, Paper Tiger, Clock, etc.	$20.00	24
885-1116	HDOS Z80 Debugging Tool	$20.00	27
885-1119	B H BASIC Support H8/H19 or H89	$20.00	29
885-8001	SE UCSD-Style Screen Editor	$25.00	28
885-8003	B H BASIC to MBASIC Converter	$25.00	28
885-8004	UDUMP and FAKEMNT Disk Manipulation Utilities	$35.00	28
885-8005	MAPLE Modem Program	$35.00	29
885-8007	EZI-TRANSfer	$30.00	30
PROGRAMMING LANGUAGES			
885-1038	WISE on Disk H8/H89	$18.00	
885-1042	PILOT H8/H89	$19.00	
885-1059	FOCAL-8 H8/H89	$25.00	13
885-1078	HDOS Z80 Assembler	$25.00	21
885-1085	PILOT Documentation	$ 9.00	
885-1086	Tiny Pascal H8/H89	$20.00	13
885-1094	HUG Fig-Forth H8/H89 2 Disks	$40.00	18

BUSINESS, FINANCE AND EDUCATION

885-1047	Stocks H8/H89		$18.00	
885-1048	Personal Account H8/H89		$18.00	
885-1049	Income Tax Records H8/H89		$18.00	
885-1055	Inventory H8/H89	*	$30.00	
885-1056	Mail List H8/H89	*	$30.00	
885-1070	Disk XIV Home Finance H8/H89		$18.00	
885-1071	SmBusPkg III 3 Disks H8/H19 or H89	*	$75.00	17
885-1091	Grade and Score Keeping	*	$30.00	14
885-1097	Educational Quiz Disk H89 or H8/H19	*	$20.00	18
885-1118	Payroll	*	$60.00	30

DATA BASE MANAGEMENT SYSTEMS (DBMS)

885-1107	Amateur Radio Logbook and TMS		$30.00	23
885-1108	Telephone/Mail Info. System	*	$30.00	23
885-1109	Retriever (2 disks)		$40.00	23
885-1110	Autofile		$30.00	23
885-1115	Aircraft Navigation H8/H89	*	$20.00	25
885-8008	Farm Accounting System	*	$45.00	30

AMATEUR RADIO

885-1023	RTTY Disk H8 Only		$22.00	6
885-1106	Morse-89 H8/H19 or H89		$20.00	22

* Means MBASIC is required

H11 SOFTWARE

885-1008	Volume I Documentation and Program Listings (some for H11)		$ 9.00	
885-1033	HT-11 Disk I		$19.00	
885-1053	H11/H19 Support Package EXEC Modem Software, etc.		$20.00	27
885-1117	Pirate's Adventure for H11/H19		$20.00	28

CP/M SOFTWARE (5-inch only)

885-1201	CP/M (TM) Volumes H1 and H2	%	$21.00	
885-1202	CP/M Volumes 4 and 21-C	%%	$21.00	
885-1203	CP/M Volumes 21-A and B	%%	$21.00	
885-1204	CP/M Volumes 26/27-A and B	%%	$21.00	
885-1205	CP/M Volumes 26/27-C and D	%%	$21.00	

The above CP/M products are 2 disks each.

885-1206	CP/M Games Disk	*	$20.00	11
885-1207	TERM and H8COPY		$20.00	26
885-1208	HUG Fig-Forth H8/H89 2 Disks		$40.00	18
885-1209	Dungeons and Dragons Game MBASIC and H89 or H8/H19	*	$20.00	19
885-1210	HUG Editor		$20.00	20
885-1211	Sea Battle Game for CP/M		$20.00	20
885-1212	CP/M Utilities I		$20.00	21
885-1213	CP/M Disk Utilities		$20.00	22
885-1214	Amateur Radio Logbook	*	$30.00	23
885-1215	BASIC-E		$20.00	26
885-1217	HUG Disk Duplication Utilities		$20.00	26

% Means CP/M 1.43 only (ORG-4200).
%% Means CP/M 1.43 or 2.2 (Heath). MBASIC programs on these disks are for version 4.8 or earlier.
Other CP/M disks are for 2.2. * means MBASIC is required.

MISCELLANEOUS

885-0017	H8 Poster	$ 2.95	
885-0018	H89 Poster	$ 2.95	
885-0019	Color Graphics Poster	$ 2.95	
885-4	HUG Binder	$ 5.75	
885-4001	REMark VOLUME I	$20.00	23
885-4002	REMark VOLUME II	$20.00	

CP/M is a registered trademark of Digital Research Corp.

Vectored from page 15

```
750 X$=C$+"\"+B$+"\"+A$+"\"+D$+"\"+E$+"\"+F$+G$+H$:'concatenate all but phone
760 IF LEN(P$)=0 THEN 770 ELSE X$=X$+"\"+P$:' add '\' + phone # ONLY if entry
770 PRINT #2,X$:'write this name and address, all formatted, to file "TEMP"
780 S1=S1+1:' S1= counter for new records added.
790 GOTO 250:' Go back for next name and address entries
800 CLOSE:NAME "TEMP" AS T$
810 ' Tell user record number status.
820 PRINT:PRINT "There were";S;"records in ";T$+".OLD data file."
830 PRINT "There were";S1;"records just added."
840 PRINT "There are now";S+S1;"records in file ";T$;". Bye-bye!":END
850 ' Error handling (see text) routines -
860 IF ERR=53 AND ERL=180 THEN INPUT "Opening NEW file? (Y/N)... ";N$ ELSE 890
870 IF N$="Y" OR N$="y" THEN OPEN "O",2,"TEMP":RESUME 250
880 IF N$="N" OR N$="n" THEN PRINT "NO file ";T$;"! Check and start over!":END
890 IF ERR=53 AND ERL=800 THEN RESUME NEXT
900 IF ERR=53 AND ERL=210 THEN RESUME NEXT
910 ON ERROR GOTO 0
920 'Line Length handling routines -
930 PRINT:PRINT"Above 3 items too long by";M-T;"chars. Abbreviate!":PRINT:RETURN
940 PRINT:PRINT "Above item too long by";M-T;"chars. Abbreviate!":PRINT:RETURN
950 FOR K=1 TO 24:PRINT:NEXT K:RETURN:' clear screen
960 ' H19/89 users substitute PRINT CHR$(27)+"E":RETURN for above line


10 'PRDATA.BAS  prints DATA & prefaces each record with its absolute position
20 LINE INPUT "Enter FILE NAME.... ";T$:OPEN "I",1,T$:N=1
30 IF EOF(1) THEN CLOSE:END:ELSE LINE INPUT #1,X$:LPRINT N;X$:N=N+1:GOTO 30
```

An Editor for BENTON HARBOR BASIC

In last month's issue of REMark, I presented a number of improvements for Benton Harbor Disk BASIC. In this article I will describe a co-resident editor for B H BASIC that uses features of the H19 terminal or H89 computer. Both the editor and the programs presented last month are available on HUG disk 885-1119. It was incorrectly stated in last month's REMark that the programs require 48k. They will actually run in a 32k system. Here is a review of the programs on disk 885-1119:

BAS16.ABS and BAS20.ABS -- This is standard B H BASIC (for HDOS 1.6 and 2.0) patched so that the FREEZE and UNFREEZE commands save and load programs in compressed format, to save time and disk space.

LBASIC.ABS -- This is a preloader for B H BASIC that allows you to load and run programs from a single HDOS command line, and to set a memory limit for BASIC.

EDBASIC.ABS -- This is the co-resident editor described in this article.

BC.DVD -- This is a device driver which, when loaded, re-defines the use of the function and keypad keys (while activated) to produce certain BASIC keywords with one stroke, such as PRINT or GOTO.

RENUM.ABS -- A fast renumber for B H BASIC programs (released previously on another HUG disk).

MAP.ABS -- Generates cross references of all variables and referenced lines in BASIC programs (also previously released).

BASE.BAS -- A program to convert from split octal to decimal and vice versa.

SUBS.BAS -- BASIC subroutines that allow calling machine language routines from B H BASIC programs, single character input without RETURN, and keyboard scanning.

FKEYS.BAS and REFLEX.BAS -- demonstrations of the above subroutines.

If you are uncomfortable working with assembly language and would like the editor presented here along with the other things mentioned, you should order disk no. 885-1119.

The Editor Programs

The assembly listings following this article make up the editor. BEDIT.ASM is the actual editor, and ELOADR.ASM is a loader program that places the editor in memory. ELOADR is a modified version of the program RELOC by Ted Eitel, from HUG disk 885-1090. After BEDIT and ELOADR are assembled, they should be combined into one program with PIP as shown in the ELOADR source listing, to make the file EDBASIC.ABS. Note that BEDIT.ASM contains an assembly parameter that allows you to assemble it for either HDOS 1.6 or 2.0. Be sure you set it for the version you are using.

When you run EDBASIC, the loader portion locates the highest memory area available (under the HDOS overlays), places the editor portion there, and passes control to it. The editor then loads BASIC.ABS from the same disk, patches it to allow interface to the editor, and passes control to it. BASIC then signs on just as it does without the editor.

Using the Editor

To use the editor, BASIC.ABS and EDBASIC.ABS must be on the same disk. To get things started, enter

```
>EDBASIC
```

at the HDOS prompt. If EDBASIC and BASIC are on a disk other than the system, you can include a drive designation, such as

```
>SY1:EDBASIC
```

Normally, EDBASIC will set the memory limit (the limit to the memory used by BASIC) just below the editor, but you can lower it more by specifying an address in decimal in the command line, such as

```
>EDBASIC 40000
```

This would set the limit at 40000 decimal, and you could use the area from there to the editor for user machine language subroutines. For more information, see REMark issue 29.

When you run EDBASIC, B H BASIC signs on normally. You can load a program and/or enter program lines in the usual way. When you want to use the editor, type control-D (hold down the CTRL key and press the D key). The prompt "Enter line to edit:" will appear on the 25th line of your screen. Type the number of the line that you wish to edit. If you want to get back to BASIC without editing, type control-D again. After you type a line number, hit RETURN, and the line with that number will appear on the 25 line of the screen with the cursor at the left end of the line. You can use the left and right arrow keys on the number keypad to the right of the main keyboard to move the cursor anywhere along the line. If you type a key on the main keyboard, its character will replace the one where the cursor is, and the cursor will move to the right. You can replace words just by typing the new one over the old one. If you would like to insert characters in the line, press the IC key on the keypad. The letters IC will appear in the upper right hand corner of the screen, indicating that the insert mode is on. Now any characters you type will be inserted where the cursor is, and the cursor and the rest of the line will move to the right. To cancel the insert mode, press the IC key again, and the letters IC will disappear from the screen.

To illustrate what we have covered so far, suppose you want to change the word GOTO to GOSUB. First you would position the cursor to the T in GOTO and

type SU, which would replace the TO. Now to get in the extra letter, press IC and type B. The word GOSUB now appears in the line instead of GOTO.

If you want to delete a character, you can position the cursor to the character and type the DC key. The character at the cursor will be deleted and the rest of the line will move to the left to take its place. You can also use the DELETE key to delete characters. In this case, the character to the left of the cursor is deleted, and the line and the cursor move to the left. The BACK SPACE key will not delete characters, but has the same effect as the left arrow key. You can hold down the repeat key while pressing the left arrow, right arrow, DC, DELETE, or BACK SPACE keys for multiple movements or deletes.

After you finish editing, hit RETURN to end the edit and insert the edited line into your program. If there are any syntax errors unacceptable to BASIC in the edited line, it will not be accepted, and the old line (before any editing) will still be in the program. If you want to abort the edit, type control-D.

You can edit any part of a line, including the line number, so the editor can be used to replicate lines in a program. You can also use it to enter new lines into the program by hitting RETURN at the "Enter line to edit:" prompt. In this case, the 25th line will be blank, with the cursor at the left end. Just enter the new line (don't forget the line number) and hit RETURN. You can use any of the editing features described above to make changes in the line while you are entering it.

Restrictions

The length of a line edited cannot exceed 80 characters, including the line number. If there are more than 80 characters, the extra ones will "fall off" the end and be lost. Any leading zeros in the line number can be deleted to get in a few more characters. You cannot put any control characters in the line, and any in it already will be lost. Control characters should be entered with CHR$(or a variable. For example, if you want to beep the terminal beeper, use CHR$(7) or a variable assigned to that value.

How it Works

EDBASIC makes heavy use of the editing features built in to the H19/H89 and routines in BASIC, and does little of the work itself. When it loads BASIC, it patches the routine that accepts user input so that a control-D causes a jump to the routine that asks you for a line number to edit. When you type in a number, EDBASIC converts it to binary and calls part of the LIST routine in BASIC, and the line is listed on the 25th screen line. Then it sets the keypad to the shift mode and enters the actual edit loop. While in this loop, most characters you type are simply echoed back to the terminal, which does the actual editing. The loop checks for illegal escape sequences and toggles the IC display as you type the IC key. It also looks for control-D to abort the edit and RETURN to end it.

When you type RETURN, EDBASIC sends the "transmit 25th line" sequence to the terminal and enters a short wait loop. During this wait, the characters being transmitted from the screen are stored by HDOS in its type-ahead buffer. Then things are cleaned up (shift mode off, insert mode off, IC display cleared, etc.) and control is passed to BASIC. BASIC sees the type-ahead buffer full of characters, and accepts them just as if you had typed them all in yourself. If the new line accepted has a line number the same as one in the program, the new line replaces the old one in the usual way, and the edit is accomplished.

PS:

```
*       BEDIT - An Editor for B H BASIC
*
*       THIS PROGRAM PROVIDES A LINE EDITOR FOR
*       DISK B H BASIC.  THE EDITOR IS ENVOKED BY
*       TYPING CONTROL-D.  THE USER WILL BE ASKED
*       FOR A LINE NUMBER TO EDIT.  AFTER HE ENTERS
*       THE LINE NUMBER, THE LINE WILL BE LISTED ON
*       THE 25TH LINE OF THE TERMINAL.  THE USER
*       CAN THEN EDIT THE LINE USING THE LEFT AND
*       RIGHT ARROW KEYS AND THE IC AND DC KEYS.
*
*       THIS PROGRAM MUST BE COMBINED WITH ELOADR.ABS.
*       SEE INSTRUCTIONS IN ELOADR.ASM.
*
*       BY P. SWAYNE, HUG  3-MAY-82

        XTEXT   HOSDEF
        XTEXT   HOSEQU
$TYPTX  EQU     31136A
S.SYSM  EQU     40320A

HDOS2.0 EQU     0               ASSEMBLE FOR HDOS 2.0

*       MAKE THIS LINE READ "HDOS2.0  EQU   1" IF YOU
*       ARE USING HDOS 1.6.  USE THIS PROGRAM WITH HDOS
*       2.0 OR 1.6 ONLY!

        CODE    PIC             POSITION INDEPENDENT CODE
        DW      START           ESTABLISH START ADDRSS

*       LOAD BASIC.ABS INTO MEMORY

START   LXI     H,0
        DAD     SP              FIND STACK
        MOV     A,L
        CPI     200Q            HAS IT MOVED?
        JZ      NONUM           NO, NO ARGUMENT ENTERED
GETARG  MOV     A,M             GET A CHARACTER
        INX     H               MOV TO NEXT ONE
        CPI     ' '             SPACE?
        JZ      GETARG          SKIP SPACES
        DCX     H               BACK UP TO FIRST CHARACTER
        CALL    DECIN           CONVERT NUMBER TO BINARY
        JC      NONUM           BAD NUMBER
        SHLD    MEMTOP          SAVE USER'S MEMORY TOP
NONUM   LXI     SP,USERFWA-8    LOWER STACK A BIT
        MVI     A,3
        LXI     B,0FFFFH
        SCALL   .CONSL          SET FULL CONSOLE WIDTH
        LXI     H,NAMRET
        LXI     D,DEFALT        POINT TO .NAME BUFFERS
        MVI     A,-1            USE SYSTEM CHANNEL (-1)
        SCALL   .NAME           GET EDBASIC DEVICE NAME
        LXI     H,BASIC         POINT TO "BASIC"
        LXI     D,DEFALT        AND DEFAULTS
        XRA     A
```

```
        SCALL   .OPENR          OPEN FOR READ
        JC      ERR             COULDN'T DO IT
        CALL    HERE            PUSH PC ONTO STACK
HERE    POP     H               GET CURRENT PC
        SHLD    LIMIT           SAVE MEMORY LIMIT
        LXI     D,-USERFWA
        DAD     D               SUBTRACT MEMORY START
        MOV     B,H
        MVI     C,0             (BC) = BUFFER SIZE
        LXI     D,USERFWA-8     PUT BASIC.ABS HERE
        XRA     A
        SCALL   .READ           READ IN BASIC.ABS
        CPI     1               GOOD READ (MUST BE EOF)
        JNZ     ERR             NO
        XRA     A
        SCALL   .CLOSE          CLOSE FILE

*       FIX USER MEMORY LIMIT

        LHLD    LIMIT           GET MEMORY LIMIT
        LDA     MEMTOP+1        GET USER'S MEMORY TOP PAGE
        ORA     A               ANYTHING ENTERED?
        JZ      NOUSRM          NO, USE LIMIT
        CPI     132Q            ADDRESS TOO LOW?
        JC      NOUSRM          YES, USE LIMIT
        MOV     B,A             SAVE PAGE IN B
        LDA     LIMIT+1         GET LIMIT PAGE
        CMP     B               ADDRESS TOO HIGH?
        JC      NOUSRM          YES, USE LIMIT
        LHLD    MEMTOP          ADDRESS OK, USE IT
NOUSRM  SHLD    S.SYSM          SET MEMORY LIMIT

*       PATCH BASIC TO JUMP TO EDITOR ON CONTROL-D

        MVI     A,303Q          JUMP INSTRUCTION
        STA     77105A          PUT IT IN BASIC
        STA     77131A          HERE ALSO
        LXI     H,EDITCH        EDIT CHECK ROUTINE
        SHLD    77106A          PUT IT IN BASIC
        LXI     H,DELSPC        GET SPACE DELETION ROUTINE
        SHLD    77132A          PUT IT IN BASIC
        IF      HDOS2.0
        JMP     112252A         BASIC ENTRY POINT
        ELSE
        JMP     112210A         HDOS 1.6 BASIC ENTRY POINT
        ENDIF

*       CHECK USER INPUTS FOR CONTROL-D, START EDIT IF SO

EDITCH  SCALL   .SCIN           CHECK CONSOLE FOR ENTRY
        JC      77076A          NO ENTRY, GO BACK TO BASIC
        CPI     4               CONTROL-D?
        JZ      EDIT            YES, START EDITOR
        JMP     77112A          NO, RETURN TO BASIC

*       EDITOR ENVOKED, GET LINE TO EDIT

EDIT    SCALL   .SCIN           CLEAR INPUT BUFFER
        JNC     *-2
        LXI     B,8181H
        XRA     A
        SCALL   .CONSL          SET CONSOLE TO CHAR MODE
        SCALL   .CLRCO          CLEAR CONSOLE
        CALL    $TYPTX          SAVE CURSOR POSITION
        DB      27,'n'+200Q
        MVI     B,4             SET COUNTER
        LXI     H,CURPOS        PUT CURSOR POSITION HERE
CURLP   SCALL   .SCIN           GET A CHARACTER
        JC      *-2
        MOV     M,A             STORE IT
        INX     H               INCREMENT POINTER
        DCR     B               DONE?
        JNZ     CURLP           LOOP UNTIL DONE
        CALL    $TYPTX          GO TO 25TH LINE
        DB      27,'x1'
ELINE   DB      27,'Y8 Enter line no. to edit:',240Q
        SCALL   .CLRCO          CLEAR CONSOLE
        LXI     B,0FFH
        XRA     A
        SCALL   .CONSL          SET CONSOLE TO LINE MODE
        LXI     H,BUFFER        PUT NUMBER HERE
GETNUM  SCALL   .SCIN           GET A CHARACTER
        JC      *-2
        MOV     M,A             STORE CHARACTER
        CPI     12Q             END OF ENTRY?
        JZ      GOTNUM          YES
        CPI     4               CONTROL-D?
        JZ      ABORT           YES, END EDIT
        INX     H               INCREMENT POINTER
        JMP     GETNUM          GET MORE
GOTNUM  LXI     H,BUFFER
        CALL    DECIN           CONVERT USER'S NUMBER
        JC      ABORT           BAD NUMBER
        PUSH    H
        POP     D               (HL) = (DE) = NUMBER

*       ENTER EDIT MODE

        CALL    $TYPTX          CLEAR 25TH LINE
        DB      27,'Y8                                                  '
        DB      '                                                       ',27,'Y8 '
        DB      27,'t'+200Q     ENTER KEYPAD SHIFTED MODE
        CALL    51066A          LIST USER'S LINE
        XRA     A
        LXI     B,8181H
        SCALL   .CONSL          SET CONSOLE TO CHAR MODE
EDITLP  SCALL   .CLRCO          CLEAR CONSOLE
        SCALL   .SCIN           GET ENTRY
        JC      *-2
        CPI     4               CONTROL-D?
        JZ      ABORT           IF SO, ABORT EDIT
        CPI     33Q             ESCAPE?
        JZ      CKINS           YES, CHECK FOR INSERT/DELETE
        CPI     177Q            DELETE?
        JNZ     NODEL           NO
        CALL    $TYPTX          DELETE, BACK UP AND DELETE
        DB      8,27,'N'+200Q
        JMP     EDITLP          CONTINUE EDITING
NODEL   SCALL   .SCOUT          PRINT CHARACTER
        CPI     12Q             RETURN?
        JNZ     EDITLP          IF NOT, CONTINUE EDITING
        JMP     XMIT            TRANSMIT EDITED LINE
CKINS   SCALL   .SCOUT          PRINT ESCAPE
        SCALL   .SCIN           GET CHARACTER AFTER ESC
        JC      *-2
        LXI     H,GOODKYS       GET GOOD KEYS
        MVI     B,5             NO. OF GOOD KEYS
CHKEYS  CMP     M               GOT A GOOD KEY?
        JZ      PRKEY           YES, PRINT IT
        DCR     B               DECREMENT COUNTER
        INX     H               INCREMENT POINTER
        JNZ     CHKEYS          CONTINUE CHECKING
        MVI     A,' '           BAD KEY, REPLACE W/SPACE
PRKEY   SCALL   .SCOUT          PRINT IT
        CPI     '@'             ENTER INSERT MODE?
        JZ      INSERT          YES
        CPI     'O'             EXIT INSERT MODE?
        JNZ     EDITLP          NO, CONTINUE NORMAL LOOP
        CALL    $TYPTX          KILL "IC"
        DB      27,'j',27,'Y n  ',27,'k'+200Q
        JMP     EDITLP
INSERT  CALL    $TYPTX          PRINT "IC" IN CORNER
```

```
        DB      27,'j',27,'Y nIC',27,'k'+200Q
        JMP     EDITLP

*       TRANSMIT 25TH LINE TO BASIC

XMIT    CALL    $TYPTX
        DB      27,'J'+200Q     TRANSMIT LINE
        LXI     H,0
DELAY   DCX     H               WAIT A BIT
        MOV     A,H
        ORA     L
        JNZ     DELAY
FIXSCRN XRA     A
        LXI     B,0FFH
        SCALL   .CONSL          SET CONSOLE TO LINE MODE
        CALL    $TYPTX
        DB      27,'v',27,'O'   SET KEYPAD UNSHIFTED, IC OFF
        DB      27,'Y n  '      KILL "IC"
        DB      27,'y','1'+200Q 25TH LINE OFF
        LXI     H,CURPOS
        SCALL   .PRINT          RESTORE CURSOR POSITION
        CALL    $TYPTX
        DB      27,'A'+200Q     MOVE CURSOR UP ONE
        LXI     B,0             CLEAR BC
        JMP     43124A          RETURN TO BASIC
ABORT   SCALL   .CLRCO  CLEAR CONSOLE
        JMP     FIXSCRN         FIX SCREEN, RETURN TO BASIC

*       DELETE TRAILING SPACES FROM COMMAND LINE

DELSPC  DCX     H               BACK UP TO LAST CHARACTER
        DCX     H
DELOOP  MOV     A,M             GET A CHARACTER
        CPI     ' '             IS IT A SPACE?
        JNZ     DDONE           NO, FINISHED
        DCX     H               BACK UP ONE MORE
        JMP     DELOOP          TRY AGAIN
DDONE   INX     H               MOVE PAST LAST CHARACTER
        MVI     M,0             TERMINATE LINE
        INX     H               FIX HL AS BEFORE
        POP     D               (DE) = LINE FWA
        MOV     A,L
        SUB     E               FIND LENGTH OF LINE
        ANA     A               CLEAR CARRY
        XCHG                    (HL) = FWA
        RET                     RETURN TO BASIC

*       CONVERT ASCII DECIMAL NUMBER AT ((HL)) TO BINARY

DECIN   LXI     D,0             CLEAR DE
        XCHG                    DE = POINTER, HL = ZERO
DLOOP   LDAX    D               GET A DIGIT
        SUI     '0'             REMOVE ASCII BIAS
        ANA     A               LESS THAN '0'?
        RM                      IF SO, DONE
        CPI     10              MORE THAN 10?
        CMC
        RC                      IF SO, RETURN WITH CARRY
        INX     D               MOVE TO NEXT DIGIT
        DAD     H               * 2
        PUSH    H               SAVE IT
        DAD     H               * 4
        DAD     H               * 8
        POP     B               BC = N * 2
        DAD     B               (N * 2) + (N * 8) = N * 10
        MOV     C,A             ADD IN LATEST DIGIT
        MVI     B,0
        DAD     B
        JMP     DLOOP           GET ANOTHER DIGIT

*       REPORT ERRORS

ERR     PUSH    PSW             SAVE ERROR CODE
        CALL    $TYPTX
        DB      12Q,7,'ERROR -',240Q
        POP     PSW             RESTORE ERROR CODE
        MVI     H,12Q           END ERROR WITH NEW LINE
        SCALL   .ERROR
        XRA     A
        SCALL   .EXIT           RETURN TO HDOS

*       CONSTANTS AND BUFFERS

GOODKYS DB      'DC@ON'         GOOD KEYPAD KEYS
CURPOS  DB      '    ',200Q     CURSOR POSITION
LIMIT   DW      0               MEMORY LIMIT
MEMTOP  DW      0               USER'S MEMORY LIMIT
BUFFER  EQU     *               PUT BUFFER HERE
NAMRET  RET
        DS      8
DEFALT  DB      'SY0ABS'
BASIC   DB      'BASIC',0

        END



*       ELOADR - B H BASIC EDITOR LOADER
*
*       ADAPTED BY P. SWAYNE, HUG  4-MAY-82
*       FROM T. J. EITEL'S RELOC PROGRAM
*
*       ASSEMBLE THIS FILE TO GET ELOADR.ABS, THEN MAKE
*       ONE FILE FROM THIS FILE AND BEDIT.ABS AS
*       FOLLOWS:
*
*       COPY EDBASIC.ABS=ELOADR.ABS,BEDIT.ABS

$MOVE   EQU     30252A
$ADJREL EQU     33175A

        XTEXT   HOSDEF
        XTEXT   HOSEQU
        XTEXT   ESVAL

        ORG     USERFWA
*
START   XRA     A
        SCALL   .LOADO          LOAD FIRST OVERLAY
        JC      ERROR
        MVI     A,1
        SCALL   .LOADO          LOAD SECOND OVERLAY
        JC      ERROR
        LXI     H,-1
        SCALL   .SETTP          FIND MEMORY TOP
        PUSH    H               SAVE IT
        SCALL   .SETTP          SET IT
        JC      ERROR
        POP     H
        LXI     D,-280H
        DAD     D               SUBTRACT EDITOR SIZE
        SHLD    LOADADR         SAVE AS LOAD ADDRESS
        LXI     D,-8            .NEGITIVE OFFSET
        DAD     D               .SUBTRACT 8
        SHLD    RELADDR         .LOAD - OFFSET
LOAD    MVI     A,-1            .CHANNEL -1
        LXI     B,256           .ONE SECTOR
        LXI     D,BUFFER        .INPUT AREA
        SCALL   .READ           .INPUT ONE SECTOR
        JC      ERROR           .ERROR ON INPUT
MOVE    LXI     D,BUFFER+8      .FIRST ADDR OF PROG
        LHLD    LOADADR         .PROGRAM AREA
        LXI     B,248           .MOVE 248 BYTES
```

```
        CALL    $MOVE           .MOVEM
        XCHG                    .ADDRESS TO DE
        LHLD    LENGTH          .PROGRAM LENGTH
        MOV     B,H             .# OF SECTORS
        MVI     C,0             .EVEN SECTORS
        MVI     A,-1            .CHANNEL -1
        SCALL   .READ           .INPUT REST OF FILE
        JC      ERROR           .ERROR ON INPUT
RELOC   LHLD    RELADDR         .RELATIVE OFFSET
        XCHG                    .OFFSET TO DE
        LHLD    RELINFO         .ADDR OF RELOC INFO
        DAD     D               .HL NOW ABSOLUTE
        MOV     C,E
        MOV     B,D
        CALL    $ADJREL
FINISH  LHLD    RELADDR         .OFFSET
        XCHG                    .OFFSET TO DE
        LHLD    STRTADR         .PROGRAM ENTRY
        DAD     D               .MAKE ABSOLUTE
        PCHL                    .GO TO PROGRAM
ERROR   MVI     H,120           .TRAILING CHARACTER
        SCALL   .ERROR          .DISPLAY ERROR
        XRA     A
        SCALL   .EXIT
RELADDR DW      0
LOADADR DW      0               .LOAD ADDRESS
BUFFER  EQU     *               .INPUT AREA
LENGTH  EQU     BUFFER+2        .PROGRAM LENGTH
RELINFO EQU     BUFFER+4        .RELOCATE INFO
STRTADR EQU     BUFFER+6        .PROGRAM ENTRY
BUFEND  EQU     BUFFER+256
*
        END     START
```

The MX-80 Once More

In last month's REMark, I presented a patch for the HUG or Heath MX80 device driver for the GRAFTRAX option. This patch was based on what I had heard, and not on actual experience. Since then I have obtained an MX-80 with GRAFTRAX Plus and the 8145 Serial Interface, and found that that patch does not work. I have developed a patch that does work, which should be made to either the HUG MX-80 device driver (from 885-1105) or Heath's LPMX80.DVD if you want to use the GRAFTRAX option. Make the patch using PATCH.ABS as follows (what you type is in bold print):

```
>PATCH

PATCH Issue #50.06.00.

File Name? LPMX80.DVD
Patch ID? IFOJIC
Prerequisite Code? IFBEIADPGEFFCF

Address? 3042
003042 = 345/311
003043 = 072/^D     (control-D)
Address? 3314
003314 = 346/76
003315 = 357/3
003316 = 315/^D
Address? 4127
004127 = 033/377
004130 = 104/^D
Address? ^D
Patch Check Code? EEJIAFBF

PATCH Issue #50.06.00.

File Name? ^D
```

If you are patching the HUG version, you will not be asked for the Patch codes, and the file you are patching (LPMX80.DVD) must not have the W or L flag set. Make the patch whether you entered last month's patch or not. After the patch is made, your driver will support the MX-80 or MX-100 with GRAFTRAX Plus (with an appropriate serial interface).

The switch settings in the MX-80 or MX-100 with GRAFTRAX are different from those shown in last month's article. The switches on the printer board should be set as follows:

SW1 settings

1	off	Normal print (uncompressed)
2	off	Not used
3	off	Paper out sensor active
4	off	Normal print (not italic)
5	off	Normal print (not emph.)
6	on	Buzzer on
7	off	Regular zero (not slashed)
8	on	Select fixed

SW2 settings -- all off

As I stated last month, you need a serial interface with at least a 2k buffer to use GRAFTRAX. Epson makes at least 3 different serial boards with 2k buffers. The one I had access to was model 8145. The switch settings for this board are as follows:

SW1 settings

1	off	Baud rate (4800)
2	on	Baud rate
3	off	Baud rate
4	off	Baud rate
5	on	Data ready flag control
6	on	Data ready flag control
7	on	Parity disabled
8	on	Even parity (ignored)

SW2 settings

1	off	8-bit word length
2	on	Reverse channel = Mark
3	off	Reverse channel = Mark
4	on	Reverse channel valid

All jumpers on the board should be at their factory settings. Having a 2k buf-

Vectored to page 39

Oooooopppsss, I Forgot!

Or the Missing Program

In the May 1982 issue of REMark, a program was omitted from the article "Split Byte Octal/Decimal Addition/Subtraction" by Robert G. Traub. My apologies to Mr. Traub - here is the program.

NS

```
* SPLIT OCTAL OR DECIMAL FROM SPLIT OCTAL SUBTRACTION
* PROGRAM. BY R.G. TRAUB
*
SCONSL  EQU     006377A         SET CONSOLE MODE
SCIN    EQU     001377A         INPUT ROUTINE
SEXIT   EQU     000377A         EXIT SCALL
SPRINT  EQU     003377A         STRING SCALL
SCOUT   EQU     002377A         CHARACTER SCALL
STACK   EQU     042200A         STACK DOWN FROM HERE
LF      EQU     012Q            LINE FEED
*       ORG     042200A
START   LXI     SP,STACK        SET STACK AREA
        XRA     A               CLEAR 'A'
        LXI     BC,001201A      CONSOLE MODE WORD
        DW      SCONSL          SET MODE NOW
*
* SHOW OPENING MESSAGE
*
BEGIN   LXI     HL,MESS1        STARTING MESSAGE
        DW      SPRINT
BEG1    LXI     HL,MESS6
        DW      SPRINT
*
BEG2    MVI     A,1             FLAG FOR SUBTRACTION FIRST
        STA     FLAG            EACH TIME THRU ROUTINE
        CALL    INPUT           READ KEYBOARD
        CPI     'H'             H FOR 'HDOS'
        JZ      EXIT            WARM REBOOT
        LXI     HL,MESS2        SECOND MESSAGE
        DW      SPRINT          SHOW IT
*
        CALL    OCTIN4          GET OCTAL NUMBER
        MOV     H,A             SAVE IT
        MVI     A,'.'           GET A PERIOD
        DW      SCOUT           SHOW IT
        CALL    OCTIN4          GET NEXT BYTE
        MOV     L,A             SAVE IN 'L'
        SHLD    NBUF            SAVE IT IN BUFFER
        LXI     HL,MESS3        THIRD MESSAGE
        DW      SPRINT
        CALL    INPUT           GET CHARACTER
        CPI     'D'             IS IT DECIMAL?
        JZ      DECIN           YES, GO GET
*                                CPI 'Q'
* SHOW HEADER FOR SECOND OCTAL NUMBER NOW        { JZ OCTIN
*                                                 JMP ERROR
OCTIN   LXI     HL,MESS8        GET MESSAGE FOR OCTAL NUMBER
        DW      SPRINT          SHOW IT
        CALL    OCTIN4          GET FIRST BYTE
        MOV     H,A             SAVE IT IN 'H'
        MVI     A,'.'           LOAD IN A PERIOD
        DW      SCOUT           SHOW IT
        CALL    OCTIN4          GET SECOND BYTE
        MOV     L,A             SAVE IN 'L'
        SHLD    NBUF+2          SAVE IN ANSWER BUFFER
        CALL    CRLF            SHOW A NEW LINE
        JMP     SUBANS          GO DO MATH
*
* ROUTINE TO SHOW FOUR SPACES ON CONSOLE
*
SPACE   MVI     B,0040          LOAD LOOP COUNT
        MVI     A,040Q          LOAD A SPACE
SPACE1  DW      SCOUT           SHOW THE SPACE
        DCR     B               SUB ONE FROM THE COUNT
        JNZ     SPACE1          DO AGAIN
        RET                     ALL DONE
*
* ROUTINE TO PRINT OUT THE ERROR MESSAGE AND START ALL OVER
*
ERROR   LXI     HL,MESS9
        DW      SPRINT          SHOW IT
        JMP     BEG1            START OVER
*
* ROUTINE TO CONVERT INPUT TO OCTAL BINARY NUMBERS
*
OCTIN4  PUSH    BC              SAVE THE BYTE
        MVI     B,000Q          CLEAR 'B'
        CALL    INOCT3          GET FIRST DIGIT
        CPI     004Q
        JZ      ERROR
        ADD     A               SHIFT IT
        ADD     A               THREE
        ADD     A               TIMES
        MOV     B,A             SAVE NEW BYTE IN 'B'
        CALL    INOCT3          GET NEXT DIGIT
        ORA     B               ADD TO FIRST
        RAL                     SHIFT IT THREE
        RAL                     TIMES
        RAL
        MOV     B,A             SAVE NEW TOTAL
        CALL    INOCT3          GET THIRD DIGIT
        ORA     B               ADD TO LAST
        MOV     B,A             STORE IN B
        MOV     A,B             GET IN 'A' NOW
        POP     BC              RESTORE BC
        RET                     DONE
*
INOCT3  CALL    INPUT           GET DIGIT
        SUI     060Q            REMOVE ASCII OFF-SET
        JC      ERROR           EDIT IF INCORRECT
        RET
*
* ROUTINE TO CONVERT INPUT TO DECIMAL BINARY NUMBERS
*
DECIN   LXI     DE,0
        LXI     HL,MESS5
        DW      SPRINT          SHOW IT
        CALL    CRLF
*
DECIN2  CALL    INPUT
        CPI     012Q            IS IT A 'RETURN'
        JZ      SAVDEC          IF YES, STORE NUMBER
        SUI     060Q            REMOVE ASCII OFF-SET
        JC      ERROR
        CPI     012Q
        JNC     ERROR
        LXI     HL,0
        DAD     D
        DAD     H
        DAD     H               HL = DE TIMES FOUR
        DAD     D               HL = DE TIMES FIVE
        DAD     H               HL = DE TIMES TEN
        MOV     E,A
        MVI     D,0
```

```
        DAD     D
        XCHG
        JMP     DECIN2
*
LINE    MOV     A,H             HIGH BYTE IN THE 'A'
        CALL    OCTOUT          CONVERT IT, SHOW IT
        MVI     A,'.'           LOAD IN A PERIOD
        DW      SCOUT           DISPLAY IT
        MOV     A,L             GET THE LOW BYTE
        CALL    OCTOUT          CONVERT, SHOW IT
        CALL    SPACE           SHOW FOUR SPACES NOW
        CALL    DECOI           SHOW THE DECIMAL EQIV.
        CALL    SPACE           SHOW ANOTHER FOUR SPACES
        CALL    BOUT            SHOW THE BINARY NUMBER
        CALL    CRLF            TURN UP A NEW LINE
        RET                     AND RETURN, ALL DONE.
*
SAVDEC  XCHG                    'DE' BACK IN 'HL'
        SHLD    NBUF+2          SAVE BUFFER
        CALL    CRLF            SHOW A NEW LINE
*
* ROUTINE TO DO THE SUBTRACTION
*
SUBANS  LDA     NBUF+2          GET 'LSB' OF SECOND
        MOV     B,A             STORE IN 'B'
        LDA     NBUF            GET 'LSB' OF FIRST
        SUB     B               SUBTRACT THEM
        STA     ANS+2           SAVE IN BUFFER
        LDA     NBUF+3          GET 'MSB' OF SECOND
        MOV     B,A             SAVE IN 'B' REG.
        LDA     NBUF+1          GET 'MSB' OF FIRST
        SBB     B               SUBTRACT WITH BORROW
        STA     ANS+3           SAVE IN BUFFER
*
* ROUTINE TO ADD THE TWO NUMBERS
*
        LHLD    NBUF+2          GET THE SECOND
        XCHG                    STORE IT IN 'DE'
        LHLD    NBUF            GET THE FIRST
        DAD     D               ADD THE TWO NUMBERS
        SHLD    ANS             SAVE BUFFER
*
* ROUTINE TO DISPLAY THE ANSWER ON THE CONSOLE DEVICE
*
ANSW    LXI     HL,MESS4        GET SCOUT MESSAGE
        DW      SPRINT          SHOW IT
DISPLY  LHLD    NBUF            GET THE FIRST
        CALL    LINE            DISPLAY IT IN THREE FORMS
        LHLD    NBUF+2          GET THE SECOND
        CALL    LINE            DISPLAY IT IN THREE FORMS
        MVI     B,045Q          LOAD IN COUNT OF 72 DECIMAL
DISP1   MVI     A,'-'           LOAD IN A '-' CHARACTER
        DW      SCOUT           SHOW IT 72 TIMES
        DCR     B               SUB ONE FROM COUNT
        JNZ     DISP1           I NOT ZERO, SHOW ANOTHER
        CALL    CRLF
        LDA     FLAG            GET FLAG CHARACTER
        CPI     0               IS IT ZERO YET?
        CZ      ADDANS          IF YES, DO ADDITION
*
        LHLD    ANS+2           ANSWER FOR SUBTRACTION
        MOV     A,H             HIGH BYTE 'A' REG.
        CALL    OCTOUT          CONVERT AND SHOW
        MVI     A,'.'           LOAD IN A PERIOD
        DW      SCOUT           SHOW IT
        MOV     A,L             GET LOW BYTE
        CALL    OCTOUT          CONVERT AND SHOW
        CALL    SPACE           SHOW TWO MORE
        CALL    DECOUT          SHOW DECIMAL RESULTS
        CALL    SPACE           SHOW TWO MORE
        CALL    BOUT            SHOW IN BINARY
        CALL    CRLF
        CALL    CRLF
        LDA     FLAG            GET FLAG
        CPI     0               IS IT ZERO?
        JZ      BEGI            IF YES, EXIT NOW
        MVI     A,0             SET FLAG TO ZERO
        STA     FLAG            FOR ADDITION ANSWER
        LXI     HL,MESS7        GET ADDITION ANSWER
        DW      SPRINT          SHOW IT
        JMP     DISPLY          SHOW ADDITION ANSWER
*
ADDANS  LHLD    ANS             GET ADDITION ANSWER
        SHLD    ANS+2           SAVE IN BUFFER
        RET
*
* ROUTINE TO PRINT OUT ANSWER IN BINARY FORMAT
*
BOUT    MOV     A,H             GET HIGH BYTE
        CALL    BINOUT          CONVERT IT TO '1' OR '0'
        MVI     A,040Q          GET A SPACE
        DW      SCOUT           SHOW IT
        MOV     A,L             GET LOW BYTE
BINOUT  PUSH    B               SAVE THE BC REGISTERS
        MVI     B,010Q          COUNT OF 8 IN B REG
BIN1    RAL                     SHIFT BIT INTO CARRY
        PUSH    PSW             SAVE MODIFIED BYTE
        MVI     A,'1'           LOAD IN AN ASCII 'ONE'
        JC      BIN2            SHOW IF IT IS A '1'
        MVI     A,'0'           OTHERWISE GET AN ASCII 0
BIN2    DW      SCOUT           SHOW IT
        POP     PSW             GET MODIFIED BYTE
        DCR     B               SUBTRACT ONE FROM COUNT
        JNZ     BIN1            DO NEXT DIGIT IF NOT ZERO
        POP     B               RESTORE 'B'
        RET                     OTHERWISE ALL DONE
*
* ROUTINE TO CONVERT OCTAL BINARY TO ASCII DIGITS AND DISPLAY
*
OCTOUT  ORA     A
        PUSH    BC
        MVI     B,003Q          LOAD LOP CONT IN 'B'
OCT1    RAL                     SHIFT IT
        RAL                     THREE
        RAL                     TIMES
        PUSH    PSW             SAVE ON STACK
        ANI     007Q            ISOLATE LS THREE BITS
        ADI     060Q            ADD ON ASCII OFFSET
        DW      SCOUT           SHOW IT
        POP     PSW             BYTE BACK NOW
        DCR     B               SUB ONE FROM LOOP
        JNZ     OCT1            AGAIN IF 'B' NOT ZERO
        POP     BC              OTHERWISE ALL DONE
        RET                     SO RETURN
*
* CONVERTS DECIMAL BINARY TO ASCII PACKED NUMBER
*
DECOUT  LHLD    ANS+2           GET ANSWER IN HL
DECO1   PUSH    HL              SAVE ON STACK
        LXI     BC,10000        LOAD 'BC' WITH 10,000
        CALL    SUBTR           SUBTRACT AND DISPLAY
        LXI     BC,1000D        LOAD BC WITH 1,000
        CALL    SUBTR           SUBTRACT AND DISPLAY
        LXI     BC,100D         LOAD BC WITH 100 DECIMAL
        CALL    SUBTR           SUBTRACT AD DISPLAY
        LXI     BC,10D          LOAD BC WITH TEN DECIMAL
        CALL    SUBTR           SUBTRACT AND DISPLAY
        MOV     A,L
        ADI     060Q            ADD IN ASCII OFF-SET
        DW      SCOUT           SHOW ON CONSOLE
        POP     HL
        RET
```

```
*
* ROUTINE TO PACK DECIMAL NUMBER
*
SUBTR   MVI     D,377Q          LOAD 'D' WITH -1
SUBTR1  MOV     A,L
        SUB     C               SUB LOW ORDER BYTE
        MOV     L,A             PUT BACK IN 'L'
        MOV     A,H
        SBB     B               SUBTRACT WITH BORROW
        MOV     H,A             RESULT BACK IN 'H'
        INR     D               INCREMENT COUNT
        JNC     SUBTR1          DO AGAIN IF NOT DONE
        DAD     B
        MOV     A,D             COUNT TO 'A' REGISTER
        ADI     0600            ADD IN ASCII OFFSET
        DW      SCOUT           SHOW ON CONSOLE
        RET
*
* ROUTINE TO INPUT A CHARACTER
*
INPUT   DW      SCIN            GET CHARACTER FROM HDOS
        JC      INPUT           IF NOT, TRY AGAIN
        ANI     177Q            STRIP PARITY
        RET                     ALL DONE
*
* ROUTINE TO SHOW A NEW LINE ON CONSOLE
*
CRLF    MVI     A,LF            GET RETURN CHARACTER
        DW      SCOUT           LET HDOS LOOK AFTER IT
        RET                     ALL DONE
*
* ROUTINE TO EXIT BACK TO HDOS
*
EXIT    XRA     A
        DW      SEXIT
*
MESS1   DB      012Q,'PGT 2.0 :OCTAL ADD/SUB PROGRAM:',012Q,212Q
MESS2   DB      'ENTER FIRST 16-BIT OCTAL NUMBER :',012Q,212Q
MESS3   DB      012Q,'ENTER "D" FOR DECIMAL OR "O" FOR OCTAL',212Q
MESS4   DB      012Q,'THE RESULTS OF THE SUBTRACTION ARE:',012Q,012Q
        DB      'OCTAL      DECIMAL      BINARY',012Q,212Q
MESS5   DB      012Q,'ENTER DECIMAL NUMBER :',012Q,240Q
MESS6   DB      'TO EXIT HIT "H" FOR HDOS OR ANY OTHER '
        DB      012Q,'KEY TO CONTINUE.',212Q
MESS7   DB      'THE RESULTS OF THE ADDITION ARE :',012Q,212Q
MESS8   DB      012Q,'ENTER SECOND 16 BIT OCTAL NUMBER',012Q,212Q
MESS9   DB      'OOPS TRY AGAIN',012Q,212Q
NBUF    DB      0,0,0,0         CLEAR FOUR SPACES FOR NUMBERS
ANS     DB      0,0,0,0         CLEAR FOUR SPACES FOR ANSWERS
FLAG    DB      1               FLAG STORAGE AREA
*
        END     START
```

Pat's Patches

The .NAME Bug

The following patch corrects the bug in the .NAME system call processor in HDOS, as reported in REMark issue #27, page 21. Enter the patch using the PATCH.ABS program supplied with HDOS. In the examples below, what you type is shown in bold print.

```
.NAME Patch for HDOS 2.0

>PATCH

PATCH Issue #50.06.00.

File Name? HDOSOVL0.SYS/DISP:12
Patch ID? IFOJIC
Prerequisite Code? IFBEIADPGEFFCF

Address? 2353
002353 = 322/341
002354 = 360/322
002355 = 002/361
002356 = 321/2
002357 = 311/341
002360 = 341/311
002361 = 021/^D      (control-D)
Address? 12263
012263 = 354/355
012264 = 002/^D
Address? ^D
Patch Check Code? CPMCHOMP

PATCH Issue #50.06.00.

File Name? ^D

.NAME Patch for HDOS 1.6

>PATCH

PATCH Issue #50.05.00.

File Name? HDOSOVL0.SYS/DISP:12
Patch ID? IFOJIC
Prerequisite Code? IFBEIADPGEFFCF

Address? 2305
002305 = 322/341
002306 = 312/322
002307 = 002/313
002310 = 321/2
002311 = 311/341
002312 = 341/311
002313 = 021/^D
Address? 11363
011363 = 306/307
011364 = 002/^D
Address? ^D
Patch Check Code? FHPNOOEI

PATCH Issue #50.05.00.

File Name? ^D
```

Another HTERM Patch

This patch is an alternate to the second part of the HTERM patch in issue #29. This patch allows all control characters in incoming data except carriage returns to be stored in HTERM's buffer. The resulting file will be in HDOS format, with only line feed characters at the end of lines.

Vectored to page 39

CP/M Disk Errors

The following program is an improved version of the ERRORS program from HUG disk 885-1212, which originally worked only under CP/M version 2.2.02. This version works under 2.2.02 or 2.2.03, and reports both hard sector and soft sector 5.25-inch disk soft (recoverable) errors if you have CP/M version 2.2.03. It also contains a better routine for printing binary numbers in decimal.

```
;*************************************
;*                                   *
;*            ERRORS                 *
;*            ======                 *
;*                                   *
;*   PRINTS # OF SOFT DISK ERRORS    *
;*       SINCE LAST COLD BOOT        *
;*                                   *
;*   RUNS ON H89, H8 (USES 8080 CODE), *
;*   32K MEMORY, REQUIRES CPM 2.2.   *
;*                                   *
;*      WRITTEN BY  R L GAZLAY       *
;*                                   *
;*           17 JULY 1981            *
;*                                   *
;*************************************
;
;       MODIFIED BY P. SWAYNE, HUG  6-APR-82
;
;
;       MISC EQUATES
;
CR      EQU     0DH
LF      EQU     0AH
;
CONOUT  EQU     2
PSTRING EQU     9
;
BASE    EQU     0
BDOS    EQU     BASE+5
        ORG     BASE+100H
;
MAIN:   LXI     H,0
        DAD     SP              ;FIND CP/M STACK
        LXI     SP,STACK        ;SET NEW STACK
        PUSH    H               ;SAVE OLD STACK
        LXI     D,CRLF
        MVI     C,PSTRING
        CALL    BDOS            ;PRINT CR, LF
        LHLD    BASE+1          ;BET BIOS ADDRESS
        LXI     D,33H-3         ;OFFSET TO VERSION NO. (.03)
        DAD     D
        MOV     A,M             ;GET VERSION NUMBER
        STA     VERS            ;SAVE IT
        LHLD    BASE+1          ;GET BIOS ADDRESS
        LXI     D,49H-3         ;OFFSET TO H17 ERROR COUNT (.03)
        CPI     3               ;VERSION .03?
        JZ      VERS3           ;YES
        LXI     D,4CH-3         ;OFFSET TO ERROR COUNT (.02)
VERS3:  DAD     D
        MOV     E,M
        INX     H
        MOV     D,M             ;(DE) = H17 ERROR COUNT
PHERR:  XCHG
        CALL    DECOUT          ;PRINT ERRORS
        LXI     D,MSG1          ;POINT TO MESSAGE
        MVI     C,PSTRING       ;READY TO PRINT STRING
        CALL    BDOS            ;PRINT H17 ERRORS
        LDA     VERS            ;GET VERSION NUMBER
        CPI     3               ;VERSION 3?
        JNZ     EXIT            ;IF NOT, FINISHED
        LHLD    BASE+1          ;ELSE, GET BIOS ADDRESS
        LXI     D,4BH-3         ;OFFSET TO H37 ERROR COUNT
        DAD     D
        MOV     E,M
        INX     H
        MOV     D,M             ;(DE) = H37 ERROR COUNT
        XCHG
        CALL    DECOUT          ;PRINT H37 ERRORS
        LXI     D,MSG2
        MVI     C,PSTRING
        CALL    BDOS            ;PRINT SOFT ERROR MESSAGE
EXIT:   LXI     D,LBMSG         ;GET "LAST BOOT" MESSAGE
        MVI     C,PSTRING
        CALL    BDOS            ;PRINT IT
        POP     H               ;GET OLD STACK
        SPHL                    ;SET IT
        RET
;
;       PRINT NUMBER IN (HL) IN DECIMAL
;
DECOUT: PUSH    B
        PUSH    D
        PUSH    H               ;SAVE REGISTERS
        LXI     B,-10           ;RADIX FOR CONVERSION
        LXI     D,-1            ;SUBTRACTION COUNTER
DX:     DAD     B               ;SUBTRACT 10
        INX     D               ;INCREMENT COUNTER
        JC      DX              ;REPEAT UNTIL OVERFLOW
        LXI     B,10
        DAD     B               ;ADD RADIX BACK IN ONCE
        XCHG                    ;(DE) = DIGIT, (HL) = NUMBER/10
        MOV     A,H
        ORA     L               ;DONE?
        CNZ     DECOUT          ;CALL RECURSIVELY UNTIL DONE
        MOV     A,E             ;GET CHARACTER TO PRINT
        ADI     '0'             ;ADD ASCII BIAS
        MOV     E,A             ;PUT RESULT IN E
        MVI     C,CONOUT
        CALL    BDOS            ;PRINT DIGIT
        POP     H
        POP     D
        POP     B               ;RESTORE REGISTERS
        RET
;
VERS    DB      0               ;SAVE VERSION NO. HERE
;
MSG1:   DB      ' H17 SOFT DISK ERRORS',CR,LF,'$'
MSG2:   DB      ' H37 SOFT DISK ERRORS',CR,LF,'$'
LBMSG:  DB      'SINCE LAST COLD BOOT.'
CRLF:   DB      CR,LF,'$'
;
        DS      64              ;STACK SPACE
STACK   DS      0
;
        END     MAIN
```

Vectored from page 4

about the circumference. The kit comes with a sealed high access one byte laser read/write device which can be mounted anywhere about the circumference. The circular tape, is then installed on a user supplied windmill. Benchmark tests show that in a 98 mile an hour wind, typical load time for a 3000 line program is inversely proportional to tensile strength times the power of the clock speed. (It's best if the clock runs on 12 volts.) To round out the kit, 65,535 easy-to-peel-off write protect tabs are supplied. Special 'no parity' stickers are also an option. A double density version is expected to follow.

HOW TO GET THERE

From O'Hare Airport: The hotel provides complimentary limousine service every 15 minutes from O'Hare which is just 10 or 15 minutes away. Just call from the phone located near the baggage claim level in each terminal building. **Downtown Chicago:** Kennedy Expressway West to O'Hare Field turnoff. Exit at River Road South. It's the large bronze colored building on your left. **From the North:** Tri-State (I-294) south into the Kennedy Expressway Chicago turnoff. Exit at Cumberland North and pass over the expressway and re-enter Kennedy Expressway to the O'Hare Field turnoff. Exit at River Road South. **From the South:** Tri-State north (I-294 I-190) to O'Hare Field, exiting at Mannheim Road South. Pass over the expressway to enter Kennedy Expressway at the Chicago East sign. Exit at River Road South. **From Midway Airport:** Take a cab. Or if renting a car, drive out of the airport and turn right and proceed to I-294 (about 20 minutes), right on I-294 (North bound) to O'Hare, exiting at Mannheim Road South as explained above.

If you are driving in Friday afternoon or evening from the East, South or West, you probably should pick up the Tri-State (I-294) as soon as possible to avoid the out-going rush hour traffic. If coming in from the East, do not use the Kennedy Expressway. Go around the South side of the city and use I-294. Saturday morning, the traffic will be pretty light. Have a safe trip!

Just got a call from Robert Todd... He will be at the conference ready to copy SIG/M and CPMUG disks for a couple bucks each if you bring a FORMATTED diskette. (See page 15 of last months REMark.)

:JB:

WILL THIS PLACE EVER BE THE SAME? Hyatt Regency O'Hare.

The Flying Huggies!!!

Alan Bose, President
Taildragger Flyers, Inc.
2514 Essex Court
Saint Joseph, Michigan 49085

The response by HUG pilots to NAVPROG-seven has been fantasic. Comments have come from students to commercial pilots. One commercial pilot writes, "I am extremely pleased with NAVPROG-seven, and other pilots I've shown it to are equally impressed." Another writes, "This program really meets the needs of the general aviation community. I myself have a commercial license with multi-engine and instrument ratings and find your program extremely useful." Someone even suggested forming the "Flying Huggies" to trade data & information on various areas of the country. I'm sure when Ed Heath was out flying his Parasol on the west side of Chicago back in the 20's, he never imagined anything like this. Well, in the meantime, here are the latest upgrades to make sure everyone is flying with the most current programs.

"East is least and West is best....." Any pilot should be able to recite that ditty by heart. To the pilot it means that his magnetic course is found by taking the true course and either subtracting easterly magnetic variation or adding westerly variation.

Since the agonic line (zero magnetic variation) practically runs through Benton Harbor, we flatland Midwestern pilots rarely have to take it into account. But a line of programming was inadvertantly left out of two of the NAVPROGseven programs that could be very mportant to Western pilots (where 10-15 degrees of easterly variation is common).

Add the following line to NAVPROG7.BAS:

```
625 IF V$(I)="E" THEN V(I)=-V(I)
```

Edit the following RNAVREF.BAS line to remove the GOTO statement:

```
600 V(K)=V5:V$(K)=V1$:
                EL(K)=E5:GOTO 610
```

Also add the following line to RNAVREF:

```
605 IF V$(K)="E" THEN V(K)=-V(K)
```

AIRINPUT.BAS which also uses magnetic variation in the RNAV functions is OK and properly takes easterly variation into account. East coast pilots should make the changes to their programs also, in case their travels should take them west of Benton Harbor.

If your database contains checkpoints very closely spaced (less than a mile apart), AUTONAV.BAS may attempt to solve for the great circle distance between them. When it does there's a 50-50 chance that an Illegal Function Call will result unless double precision is used for variable P in lines 1630 & 1640. The following changes to AUTONAV will solve it:

```
1630 P#=COS(P2/U)*COS(P4/U)
1640 Q=P#*COS(ABS(A)/U)+
          COS(ABS(B1)/U)-P#:IF Q...etc
```

The same formula is used in AIRINPUT .BAS, NAVPROG7.BAS and RNAVREF.BAS, but presents no problem since it is unlikely that you will attempt to plot a great circle route from one end of the runway to the other.

The NAVPROGseven system as distributed by HUG is optimized for a minimum system with 48K of memory. Its index will handle close to 250 navigational identifiers before running out of string space.

If you have more than 48K, you can increase system capacity by changing the CLEAR statements at the beginningof AIRINPUT.BAS, AIRALPHA.BAS, AIRROUTE .BAS and AUTONAV.BAS. With 64K, and leaving approximately 4K free for MBASIC to use, system capacity is increased to almost 2000 checkpoints. The change is the same in each program, only the line numbers are different:

In AIRINPUT edit line 20,
in AUTONAV edit line 30,
in AIRROUTE edit line 70, and
in AIRALPHA edit line 70 to read:

```
X=FRE(0):CLEAR X-3000:WIDTH...etc.
```

Edit the following AIRROUTE line to read:

```
1140 PRINT...etc:CLEAR 1000:LOAD@
"NAVPRO7.BAS",R
```

This automatically allocates almost all remaining memory for string space to be used by the index. Note that if you have a lot of RNAV cross-references, more free space is required in AIRINPUT.BAS. Consult your MBASIC manual for the correct use of the FRE and CLEAR commands. Note also that NAVPROGseven searches the entire index looking for duplicate entries; therefore, with 2000 checkpoints on file, access time will be slowed considerably.

The following upgrade to the AUTONAV.BAS program allows the pilot to limit route selection to VOR-to-VOR enroute navigation only. It also narrows the block of airspace to be searched for possible

enroute checkpoints, decreasing the amount of time required to lay a route between your departure & destination.

Edit the following AUTONAV line to read:

```
580 NEXT J:XT=XT+2:NT=NT-.5:
          XN=XN+1:NN=NN-1
```

Also add the following lines to AUTONAV:

```
515 PRINT FNC$(7,1);J1$;"VOR-to-VOR
    only? (Y or N) ";:X$=INPUT$(1):
    GOSUB 1870:PRINT X$:IF X$="N" THEN
    VN=1 ELSE IF X$ >"Y" THEN
    PRINT BL$:GOTO 515
590 FOR J=1 TO MD
592 IF VN=1 THEN 598
594 IF J=W(1) OR J=W(2) OR J=W(3)
        THEN 650
596 IF INSTR(FA$(J),"V")=0 THEN
        PR(J)=3:GOTO 650
598 IF LT(J)>XT THEN 640
```

NAVPROGseven has been modified to operate under CP/M and MBASIC 5.21 thanks to the work of Glenn Hassebrock of Decatur, Illinois and should be available soon through HUG on soft-sectored discs. The spouse and I took the old flying machine down to Decatur last weekend and met with Glenn. You CP/M pilots will be pleased to know that he has done an excellent job. Although you can take Glenn's CP/M version and put it on an H-37 (which has enough disc capacity to hold every VOR in the country), NAVPROG-seven's file management is still limited by how much free memory is available to hold the index of checkpoints.

It should be noted that NAVPROGseven was designed primarily for the general aviation pilot; not that he has to stick to one region of the country, or stay within a thousand miles of home. But it was not really designed for the ATP who traverses the country on a regular basis. That will have to wait for future versions.

A simple solution for the time being, since most flights tend to radiate outward from one's "home base", is to set up separate data discs for flights to the east and west of home. Pre-dividing the airspace around your home base into halves or quadrants will probably take care of 95% of all flights. A couple of notes for those of you who want to do this:

1) Be sure that all programs and aircraft data files on your original data disc are also on your duplicate, along with DISCID.DAT.

2) However you divide your airspace up, provide some 'overlap' towards the north. A great circle route in this hemisphere is always bowed toward the north pole. Flying a great circle to a point 1000 miles due east or west of your home base will take you 1-2 degrees north. How far north depends on your latitude; a pilot in Florida will require less overlap than a pilot in Maine.

3) As you edit your database, delete unneeded checkpoints before adding new ones to make the most efficient use of file & memory space.

Thanks to everyone for their comments and suggestions to help make NAVPROG-seven such a valuable piece of software for the general aviation community. And if you haven't earned your FAA Pilot Proficiency Wings for this year, please do it today -- it's well worth the effort. We want to make the Flying Huggies the safest pilots in the sky. Happy landings.

AB:

Vectored from page 32

fer in the serial interface is like having a despooler if your text is 2k or less. You are able to continue with what you were doing long before the printer finishes printing.

The set options on the patched driver work like they did before, except that a GRAFTRAX equipped printer does not allow compressed printing in the emphasized mode (compressed printing is canceled if it was set and you then set the emphasized mode). When you want to print graphics, you should set NOFORM, PAGE 0, and WIDTH 0. The PAGE 0 and WIDTH 0 settings allow you to print any number of successive lines or characters on a line. NOFORM prevents the driver from issuing a form feed when you close it.

PS:

Vectored from page 35

```
>PATCH

PATCH Issue #50.06.00.

Address? 50052
050052 = 376/376
050053 = 040/15
050054 = 330/310
050055 = 167/^D
Address? ^D

PATCH Issue #50.06.00.

File Name? ^D
```

Note: The old data will be all zeros if you have already made the other HTERM patch.

PS:

Heath
Users'
Group
Hilltop Road
St. Joseph MI 49085

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

POSTMASTER: If undeliverable, please do not return.

885-2030