MICROPOLIS USERS GROUP

MUG Newsletter #29 - December 1982

****

USING OVERLAY FILES

by Burks A. Smith of DATASMITH
Box 8036, Shawnee Mission KS 66208

There appears to be a misconception among MUG readers that MDOS will not allow loading a file that would overwrite the system. On the contrary, many files overwrite the system, most notably BASIC and DISKCOPY. The system does not protect itself, but has built-in facilities that prevent a user from accidentally overwriting the system. If you want to overwrite the system on purpose there are also facilities for that.

Two types of files are capable of overwriting the system without creating an error. They are system files themselves (Type 14 Hex) and "Executable Overlay Files" (Type 0C Hex). Overlay files are a special type of user file that can be loaded and executed anywhere in memory you please, even if they overlay the operating system. The only "catch" is that you can't just load an overlay file, you must execute it too. This is due to the fact that if a program overlays the system, it must be prepared to take over control of the computer itself. In practice, however, it is a simple matter to overlay the system with a "patch" without actually executing the file.

To create an overlay file, assemble your overlay code with an ORG assembler directive putting it exactly where it needs to go. We are assuming here that your overlay does not actually damage the system, but simply modifies it in a constructive way. The last statement in your assembly language source program should be an END pseudo-op, as usual, but remember that END may also have an operand that specifies an address to begin execution of the program. Usually it is the address of the beginning of the program, but in the case of an overlay patch this is not appropriate. The execution address specified with END should be the warmstart address of the system, which if 04E7 Hex. Therefore, the last statement in your source code should be END 04E7H. This will cause control to be transferred to warmstart when the program is loaded, simply re-displaying the sign on message and putting you back where you were.

The assembler only creates type 08 files, which can not overlay the system. After assembly you must change the file type to an overlay file using the MDOS command: TYPE "FILENAME" 0C. After this is done the overlay file may be loaded simply by typing its filename from the MDOS > prompt or by typing LINK "FILENAME" from BASIC. Since warmstart is the execution address of the overlay program, control is immediately returned to MDOS or BASIC, as the case may be. This technique is particularly useful for configuring MDOS or BASIC for different printers.

.....

SORTING BASIC ARRAYS (PART 1)

by Burks A. Smith of DATASMITH
Box 8036, Shawnee Mission KS 66208

This will be the first of a few articles on standard sorting techniques that can be used to sort string arrays in a BASIC program. With this installment, the environment and logic of the sorting algorithm will be presented along with a program written in BASIC that illustrates the logic used. In future articles a program written in 8080 assembly language will be developed that can be called as a function from any BASIC program. Using the identical logic as the BASIC program presented here, the assembly language function can sort an array that fills most of a 64K machine's memory in only a few seconds.

The sort program is designed to sort a one-dimensional string array in ascending order on any number of keys. For simplicity of programming, especially in assembly language, data in the array is assumed to be in IBM punched card format. That is, each element in the array is of fixed length and represents a "record" of data composed of several "fields". Since each field is at a fixed position within the record, there are no delimiters used. Fields are extracted by knowing their starting position and length, using the MID$ function in BASIC.

The creation of a record in fixed field length format, while not the normal Micropolis way of doing things, it is really rather easy to do using BASIC. For numeric data, write the data to disk or the array using the FMT function. This always generates a fixed-length string with a fixed decimal point and the number right justified, which is perfect for sorting. For strings, use the TAB function to make sure each string starts at a known field. A PUT statement to a disk drive has exactly the same result as a PUT statement to a printer, so the technique for producing a tabular report is the same. If you want to use the sort on data recorded in a different format, it is a relatively simple matter do a conversion in the sort program or as a separate utility program.

There are dozens of ways to sort data and some are so complicated that only the author can understand them. I prefer a technique known as "Shell's Method", named for the author. It isn't very difficult to understand and it does a good job of sorting, taking advantage of lists that are "almost" sorted and speeding up the sorting of lists in ramdom order.

Two pointers are used to indicate which elements of the array are to be selected for comparison. As the program gets underway, the Bottom pointer points to the beginning of the list and the Top pointer points to the middle of the list. These two elements are compared, and if the element pointed to by the Bottom pointer is "greater than" the element pointed to by the Top pointer, the two elements are swapped.

Each pointer is then incremented by one to maintain a constant increment between pointers and the process is repeated until the Top pointer reaches the end of the list. A swap counter counts the number of swaps made. If there have been no swaps made by the time the Top pointer reaches the top of the list, the top Pointer becomes half of its former starting value and the process is repeated. Otherwise the Top pointer is reset and another pass is made. In the BASIC program illustrated, the Top pointer is not a separate variable, but the sum of the Bottom pointer and the increment value.

The sort program keeps comparing and swapping elements until the increment between the two pointers is one and the entire list can be scanned without the need to move any of the elements. Therefore, the list is sorted.

In many cases, sorting on one key such as a last name is not sufficient. Since many people have the same last names, a second key, first names, is used in the case where the primary keys compare equal. The sort program uses the starting position and length of all the specified keys in an array, with the first key being the most important. If an equal comparison should be made on one key, the program tries the next key, repeating the process until an unequal comparison is made or all the keys are used up. The program illustrated uses an array to store the starting position and length of any number of keys.

The program illustrated is designed to act as a subroutine in a larger program. All sort param-

eters are variables, so the program will sort any size array on any number of keys. This is, of course, limited by the amount of memory your computer has at its disposal. It is the responsibility of the main program to dimension and fill all arrays, and to set the values of I% (number of keys) and N% (number of records to sort. The program illustrates a simple routine that reads, sorts, and rewrites a file. We assume that the file is small enough to fit in memory and contains only three fields per record: last name (L$), first name (F$), and account number (N).

To be continued.

```
10 ! **** SORT ****    Written by Burks A. Smith
20 !
30 ! BASIC PROGRAM TO SORT ARRAY A$
40 ! DATA IS EXPECTED TO BE IN FIXED FIELD LENGTH
   FORMAT.
50 ! ARRAY K% HOLDS START POSITION AND LENGTH OF
   ALL KEYS.
60 !
70 ! A$( ) - ARRAY HOLDING STRINGS TO SORT
80 ! K%(X,0) - KEY X START
90 ! K%(X,1) - KEY X LENGTH
100 ! I% - NUMBER OF KEYS
110 ! J% - BOTTOM POINTER (TOP POINTER IS J%+K%)
120 ! K% - INCREMENT BETWEEN POINTERS
130 ! N% - NUMBER OF ELEMENTS IN ARRAY
140 ! O% - NUMBER OF SWAPS
150 ! X% - TEMP KEY START
160 ! Y% - TEMP KEY LENGTH
170 ! X$ - TEMP STRING FOR SWAPPING
180 !
200 ! SAMPLE PROGRAM TO SORT A DATA FILE
210 OPEN 1 "DATAFILE" END 300
220 DIM A$(SIZE(1),250), K%(2,1)
230 N%=0
240 GET 1 L$,F$,N
250 A$(N%)=LEFT$(L$+REPEAT$(" ",20),20): ! FIX
   LENGTH
260 A$(N%)=A$(N%)+LEFT$(F$+REPEAT$(" ",20),20)
270 A$(N%)=A$(N%)+FMT(N,"ZZZZZ")
280 N%=N%+1
290 GOTO 240
300 N%=N%-1: I%=2
310 K%(1,0)=1: K%(1,1)=20: ! KEY 1
320 K%(2,0)=21: K%(2,1)=20:! KEY 2
330 GOSUB 1000: ! SORT
340 ! < PGM USES SORTED DATA >
350 CLOSE 1
360 STOP
370 !
1000 ! < SORT ARRAY A$ SUBROUTINE
1010 !
1020 K%=N%: ! SET INCREMENT AT SIZE OF LIST
1030 K%=K%/2: ! HALVE INCREMENT
1040   PRINT
1050   PRINT ">";K%;"INCR";TAB(15);"SWAPS:";
1060   J%=0: O%=0: ! ZERO BOTTOM POINTER &
       SWAPCOUNT
1070     FOR L%=1 TO I%: ! DO FOR ALL KEYS
1080       X%=K%(L%,0):  Y%=K%(L%,1): ! CURRENT KEY
           START AND LENGTH
1090       IF MID$(A$(J%),X%,Y%) > MID$(A$(J%+K%),X
           %,Y%) THEN 1110
1100     NEXT L%: GOTO 1140 : ! ALL KEYS EQUAL
1110       ! <SWAP ARRAY ELEMENTS
1120       X$=A$(J%+K%): A$(J%+K%)=A$(J%):
           A$(J%)=X$
1130       O%=O%+1: ! INCR SWAPCOUNT
1140   J%=J%+1
1150   IF J%+K%<=N% THEN 1070: ! LOOP TILL END
1160   PRINT O%;
1170   IF O%>0 THEN 1060: ! RESCAN UNTIL NO SWAPS
1180 IF K%>1 THEN 1030: ! REPEAT UNTIL INCR IS 1
1190 PRINT
1200 RETURN
```

.....

SUBMIT

by Carl J. Singer
6049 N. Morgan St., Alexandria, VA 22312
Phone (703) 354-2904

SUBMIT is a utility program for Micropolis PDS vs. 4.0. It permits commands and executable files to be batched together and processed automatically, much like an expanded CP/M SUBMIT.COM. It is called as follows:

[unit:]SUBMIT "[unit:]filename"

where filename is the name of a file created in the MDOS editor and saved to a disk currently mounted in the selected drive. Each line of filename (except lines starting with '*' or ';', which are disregarded) should be a single MDOS command or an executable program line. Example:

```
100 SCRATCH "HENRY"
110 ASSM "$SAM" "HENRY" "EC"
120 TYPE "HENRY" 18
130 SCRATCH "1:$SAM" 140 SCRATCH "1:HENRY"
150 FILECOPY "$SAM" 1
160 APP "HENRY" 1
```

When preparing the file in the editor, be sure to disable assembly language tabs with a TAB 1 1 1 command.

Any executable program used by SUBMIT must be one which returns to MDOS after its completion. This includes Micropolis utilities with types 14 to 1B, and any other program of such type designed to return to MDOS, except SUBMIT itself. SUBMIT cannot call itself, nor can it call an overlay file (e.g., BASIC, DISKCOPY) because overlay files destroy MDOS, and there would be nothing to return to.

SUBMIT will not execute a program which loads into any part of the area occupied by SUBMIT itself or the file of commands which it calls. (This file is loaded into memory immediately after SUBMIT). However, it cannot protect itself against overlaid data from programs which may use all of contiguous memory. Therefore, if at all possible, SUBMIT should be located above contiguous memory, say in the space above a ROM or any other break in RAM. The object code furnished places SUBMIT at F800 hex, and reserves F800-FFFF for SUBMIT and the called command file. If you wish to use another area, just change ORIGIN and HIBLOCK in the table of equates near the start of $SUBMIT, and reassemble.

Generally, the command file will be relatively short, so that HIBLOCK probably does not have to be more than 400H above ORIGIN. If the file turns out to be too large for the space reserved, a FILE TOO BIG message will be returned. SUBMIT uses RST 3 and RST 4 as traps, keeping it from conflicting with DEBUG, which uses RST 6.

Errors encountered during the loading of SUBMIT itself or the called command file will cause an error return to MDOS. Later errors (encountered during execution of the commands) will abort the particular command being executed, returning an error message, and SUBMIT will go on to the next command. After completing its task, SUBMIT will ring the console bell and return to MDOS.

If you wish to use the same SUBMIT "file" command on a series of disks, that command need be issued only once. Subsequently, the command EXEC ORIGIN (where ORIGIN is the load address of SUBMIT, as previously described) may be used. This procedure obviates the need for the system to load SUBMIT and "file" more than once.

A little ingenuity will develop shortcuts for many applications. Suppose you have a new RES on drive 0, and you want to transfer it to all your other system disks. First, create and save the following Editor file, which we shall call TRANSRES.

```
100 TYPE "1:RES" 0
110 SCRATCH "1:RES"
120 APP "RES" 1
130 INIT 4
140 EXEC F800 (or wherever SUBMIT has been loaded)
```

After returning to MDOS, issue the command

```
LOAD "FILECOPY"
```

Now insert the first destination disk on drive 1 and enter SUBMIT "TRANSRES". The first copy will be made and the program will pause after the INIT command with the prompt ARE YOU SURE? Replace the disk on drive 1 with the next one, hit RETURN, and continue doing this until you're finished. Now you'll have to reset and reboot to get back to the system -- a small price to pay for having done the job the easy way!

You say you have only one disk drive? Don't despair; do it this way:

Bring up the system with a disk that has SUBMIT and the new RES on it. Call the Editor and create a different TRANSRES as follows:

```
100 INIT 4
110 TYPE "RES" 0
120 SCRATCH "RES"
130 SAVE "RES" 2B1 1598 3
140 EXEC F800 (or wherever SUBMIT has been loaded)
```

After you have SAVEd this version of TRANSRES and returned to MDOS via the DOS command, enter

```
SUBMIT "TRANSRES"
```

When the ARE YOU SURE? prompt comes up, remove the original disk and start feeding the disks on which you want the new RES. Hit RETURN immediately after loading each new disk.

Don't try this second method with MDOS because the copy of MDOS currently in memory will still have the trap branches in it.

```
            FORM 51
* PROGRAM $SUBMIT
*
            LINK 'SYSQ1'
            LINK 'SYSQ2'
*
ORIGIN      EQU  0F800H
HIBLOCK     EQU  0
SYSADDR1    EQU  2025H
SYSADDR2    EQU  20B8H
@RFILESECTOR EQU 1A8EH
*
            ORG  ORIGIN
*
COLDSTART   MVI  A,0C3H
            STA  COLDSTART
            LXI  H,WARMSTART
            SHLD COLDSTART+1
            LDA  @NASCPAR
            DCR  A
            JNZ  @DISKERROR-2
            MVI  C,0
            CALL @TRANSFILENAME
            MVI  B,0
            LDA  @DRIVEN0
            MOV  C,A
            LXI  H,@FILEBUFFER0
            CALL @OPENFILE
            JC   @DISKERROR
            CALL @RFILEINF
            JC   @DISKERROR
            MOV  A,B
            ANI  0FCH
            CPI  4
            MVI  A,17
            JNZ  @DISKERROR
            LXI  H,SUBFILE
            LXI  D,HIBLOCK-SUBFILE
            MVI  B,0
            CALL @LOADDATA
            JNC  TOOBIG
            CPI  2
            JNZ  @DISKERROR
            CALL @CLOSEFILE
            JC   @DISKERROR
            MVI  A,0DFH
            STA  @MDOSEXECUTIVE
            STA  @MDOSRETURN
            MVI  A,0E7H
            STA  SYSADDR2
WARMSTART   LXI  H,SUBFILE
READFILE    DCR  M
            JZ   RESTORE
            INX  H
            PUSH H
            MVI  C,20H
            CALL @SCAN
            CALL @SKIPSPACE
            SHLD FLPTR
            POP  H
            CPI  '*'
            JZ   READFIL1
            CPI  ';'
            JZ   READFIL1
            CALL @CCRLF
            CALL @NLINEOUT
            LHLD FLPTR
            INX  H
            CALL SYSADDR1
RETURNADDR  CC   ERRMSG
            LHLD FLPTR
READFIL1    MVI  C,0DH
            CALL @SCAN
            INX  H
            JMP  READFILE
*
RESTORE     MVI  A,31H
            STA  @MDOSEXECUTIVE
            STA  @MDOSRETURN
            MVI  A,0AFH
            STA  SYSADDR2
REST1       MVI  B,7
            CALL @COUT
            JMP  @MDOSEXECUTIVE
*
TOOBIG      CALL @CLOSEFILE
            JC   @DISKERROR
            LXI  H,MSG1
            CALL @NLINEOUT
            JMP  REST1
*
TRAPSTACK   LXI  SP,@STACK
            XRA  A
            LXI  H,RETURNADDR
            PUSH H
            PCHL
*
CHECKFILE   LXI  H,@FILEBUFFER0
            CALL @OPENFILE
            JC   DSKERR
            LXI  D,1
CKADDR      PUSH D
            CALL @RFILESECTOR
            JC   DSKERR
            XCHG
            LHLD @FILEBUFFER0+30
            MOV  A,H
            ORA  L
            JZ   OKTOLOAD
            DAD  D
            LXI  D,ORIGIN
            CALL @COMPARE
            JC   LOADERR
            POP  D
            INX  D
            JMP  CKADDR
*
OKTOLOAD    POP  D
            CALL @CLOSEFILE
            JC   DSKERR
            XRA  A
            JMP  SYSADDR2+1
*
LOADERR     MVI  A,19
DSKERR      CALL ERRMSG
            JMP  TRAPSTACK
*
ERRMSG      CALL @ERRORMES
            CALL @CLOSEFILES
            RET
*
MSG1        DTH  'FILE TOO BIG'
```

```
FLPTR       DW   SUBFILE
SUBFILE     EQU  $
*
            ORG  24
*
            JMP  TRAPSTACK
*
            ORG  32
*
            JMP  CHECKFILE
*
            END
```

SUBMIT is available on MUG Library Disk 9.

.....

ZAP
===

by Carl J. Singer
6049 N. Morgan Street Alexandria, Va 22312
Phone (703) 354-2904

This program retrieves any series of consecutive tracks (e.g., a file) from the disk without reference to the directory. A series of consecutive sectors may also be written to the disk, also without directory recourse.

The program is called as follows:

[unit:]ZAP <unit number>

where <unit number> is the drive to be examined or modified (default=0).

A reply of G during the terminal dialog will send the user to the "GET" routine, and P will send him to the "PUT" routine.

If data is to be obtained from the disk, the starting track number and the number of tracks are entered and the data is dumped to memory starting at 3000H. If "full sector" is specified, each sector starts at a 512-byte boundary (3000H, 3200H, 3400H, etc.) and is 266 bytes long, including the ten-byte sector leader. The remainder of the 512-byte segment is filled with bytes set to 78 hex (lower-case x).

NOTE: Each MDOS sector consists of 268 bytes, of which the last 256 are data bytes. The ten bytes following the track and sector numbers are here referred to as the "ten-byte leader." A description of the functions of these bytes can be found in lines 2090- 2320 of Manderson's MDOSDOC, on Library Disk 17. In Manderson's notation, these are bytes 2-11. Note that Manderson's description of the first and third bytes of the ten-byte leader (his bytes 2 & 4) is inaccurate. The first byte of the leader is a backward pointer identifying the track immediately preceding this one in the sequence of tracks assigned to the file. The third byte is as described only if the file did not arise (even indirectly) by means of a FILECOPY, in which case it is zero.

If "data only" is specified, each sector is 256 bytes long and the file is dumped to memory without gaps.

ZAP also retrieves sector leaders separately without the accompanying data file.

If "track leaders only" are specified, each 16-byte line (starting at 3000H) will have the track number followed by the ten-byte leader of the first sector of each track. The line's last five bytes are not significant.

If "all sector leaders" are specified, each 16-byte line will have the track and sector numbers followed by the ten-byte sector leader. The last four bytes of the line are not significant.

ZAP also enables one to "PUT" sectors on the disk without recourse to the directory. This feature must be used carefully so that each file remains consistent with its directory description. The replacement sectors must be 266 characters long, consisting of a ten-byte leader and 256 data bytes.

The first sector must start at 3000H, the second at 3200H, etc., each sector starting at a 512-byte boundary.

This is the same format as that produced by the GET command when F(ull) sectors are specified. A disk sector can readily be changed by GETting the track containing the sector, making the changes to the memory image by means of the ENTR command, getting ZAP back with the APP command, and PUTting the 16 sectors of the track back on the disk.

CAUTION: Be sure to have a back-up of any disk on which "PUT" is used, in case something goes wrong!

Since writing on the disk destroys what is being over-written, an ARE YOU SURE? query is made, and if any reply other than Y or y is made, the write is aborted.

ZAP is available on MUG Library Disk 9.

.....

CRYPTO1
=======

by Julius C. Martin
Greenwood, WV 26360

CRYPTO1 is a program for encoding and decoding files, and is an adaptation of Ralph Roberts' BYTE magazine program (April '82). I have changed it to provide for a choice of input and output and to work in Micropolis BASIC without (known) bugs. Plus I've added some error trapping. The program prompts for your actions. See the BYTE article for Roberts' comments; see also the REMarks in the program.

Run the CRYPTO1 program using the password WIZARD, and the encoded file, TESTCODE. Select the Screen output option. You'll see both the encoded and the decoded text.

Next, try the same (or your own) password on the file THREEDEEP. If you succeed, then Roberts and I have failed. But take heart. After trying this, and perhaps some coding of your own, run the bonus program, BREAKER.

You should also be aware that the method used makes CRYPTO1 encoded text open to attack by normal letter frequency checking on an individual record basis. As an example of the individuality of the records, run CRYPTO1 on THREEDEEP in the file decode mode after using Breaker to find the key to record one. Even though the first record is decoded, other records will not be.

CRYPTO1 is available on MUG Library Disk 24.

.....

DISK CATALOG
============

by Julius Martin
Greenwood WV 26360

INTRODUCTION

This program was written to provide myself with a means of keeping track of the few disks I have and the programs/files on them. Not having CP/M (a trade mark of Digital Research) and not wanting to purchase one, if I could find one for the Micropolis system, I decided to do-it-myself. Not too long after that, the MUG published a way to access the DIR and Buzz Rudow presented MUGgers with a Disk Catalog program.

Why add another Disk Catalog program to the library? I believe in making the computer work while I

do the thinking. My approach is a little different from Buzz' (no pun intended) and frees me from worrying about disk sequencing and the like. No program since the first one is completely original and every program is subject to improvement. With that in mind, here are some details about DISKCAT.

PROGRAM OPERATION

With the program disk in drive #0, load and run <DISKCAT>. The program will introduce itself, present a 'menu', and prompt your actions. The sub-programs <DISKCATA/D> and <DISKCATS/P> allow you to add new filenames to the 'Catalog' or delete files which have been scratched from a disk. You may access the catalog and put its contents on your monitor or printer.

The disks to be cataloged may be logged into the computer in random order. The program will organize the information for you and will keep track of what file was on which disk.

This is accomplished by means of a memory-mapped screen and the use of an assigned disk number in the disk directory. The disk number is assigned by using CREATE in MDOS, or OPEN in BASIC, to place a filename on the disk, consisting of the term

-DISK.

plus the disk 'number' (see below for numbering). The program will find this filename and add its associated disk 'number' to each filename from the disk directory. E.g., this is a good disk number:

-DISK.312

One or two disk drives may be used. If two are used, one could be used to load the Program disk while the other could hold the Catalog disk. The program can work with only one drive as written.

Automatic backup of the Catalog is provided. When first a catalog is 'created', a backup (initially empty) file 'OLDCATALOG' is also created. Whenever there is a DISKSAVE of the catalog, 'OLDCATALOG' becomes 'CATALOG' while 'CATALOG' switches with 'OLDCATALOG'. (In a multi-catalog system, 'CATALOG' would be the assigned catalog 'name' and the backup would have the prefix 'OLD'.)

DISK NUMBERING SCHEME(S)

Using ideas from many 'catalogs' and adding my own, I am currenting using a numbering system which flags commercial programs with an S and a dash number or S with two numbers; for example, S-2, S23. Other disks receive numbers in the range 100-999, according to their content. Another possible numbering system, if you have a large number of files, is A00 thru Z99.

This allows 2600 disk 'names'. Since Mod II's can handle a total of 75 files per disk, the number of possible filenames is 195,000; more than we'll probably ever need. (One track was used for the disk number.)

At the moment, the program is DIMmed to hold 250 filenames (actually 251) with the memory I have available. Thus, my catalogs are also limited to 250 filenames. All sorting, printing, etc., is done based upon filenames in a memory array. Your capacity may be different. Try for 250 names to start.

To overcome the 250 filename limitation, multiple catalogs can be used. You might want to group your disks in some way; perhaps by storage box or major category. A 'name' of some sort would be used to identify a given catalog.

See MODIFICATION Section in the disk files for further information.

ADDITIONAL INFORMATION

The following control bytes are used in this program:

HEX	Decimal	Purpose
11	17	Home Cursor
17	23	Up Cursor
0C	12	Clear screen, home cursor

NOTE: The Sorcerer has a 1920-byte video screen starting at F080 Hex (-3968 Decimal).

The printer routines are set up for the EPSON MX-80. The codes used are shown in remarks. The print size and layout are based on the print sizes for the EPSON.

See the programs DISKCAT, DISKCATA/D, and DISKCATS/P for more documentation. Changes may be made to the program listings to allow more than one catalog. The assumption remains that the catalog 'name' used is limited to seven (7) characters.

I hope you will find DISKCAT useful. Feel free to modify it to work for you!

CATALOG is available on MUG Library Disk 24.

.....

LIBRARY ADDITIONS

The following are additions to various library disks which were made during November. The listings show only the additions. See the full library catalog which accompanied last month's newsletter for the previous contents.

I realize that all entries are not adequately explained. I'm working on that. Again, if members who have received MUG disks wish to send short explainations of the purpose of undocumented programs on the library disks, I will surely use the input.

MUG MDOS Library Disk 06, Revision 08, NOV 82
SYSTEM UTILITIES - 30 tracks

NAME	TYP	RV	SZE	CAT	DATE	AUTHOR/DESCRIPTION
LINE57	SYS	00	010	EDT	1182	Singer, C.
	An enhansed version of the Micropolis LINEEDIT.					
LINE57.DOC	DOC	00	02E	EDT	1182	Singer, C.

.....

MUG MDOS Library Disk 09, Revision 04, NOV 82
SYSTEM UTILITIES - 33 tracks

NAME	TYP	RV	SZE	CAT	DATE	AUTHOR/DESCRIPTION
$SUBMIT	SRC	00	00B	UTL	1182	Singer, C.
SUBMIT	SYS	00	005	UTL	1182	Singer, C.
SUBMIT-DOC	DOC	00	017	UTL	1182	Singer, C.
	Allows multiple calls to MDOS functions by way of a precomposed LINEEDIT file. Similar to CP/M's SUBMIT.					
ZAP	SYS	00	006	UTL	1182	Singer, C.
	Reads and writes any series of consecutive tracks from a disk without reference to the directory.					
ZAP-DOC	DOC	00	010	UTL	1182	Singer, C.

.....

MUG MDOS Library Disk 14, Revision 02, NOV 82
DATA BASE MANAGERS - 24 tracks

NAME	TYP	RV	SZE	CAT	DATE	AUTHOR/DESCRIPTION
INVENTORY	BAS	00	00D	DBM	1182	Riding, G.
	Packs three logical records to each physical sector, i.e., 3648 inventory items per MOD II disk.					
MAIL-LIST	BAS	00	01D	DBM	1182	

.....

MUG MDOS Library Disk 19, Revision 04, NOV 82
GAMES - 33 tracks

NAME	TYP	RV	SZE	CAT	DATE	AUTHOR/DESCRIPTION
STARTREK1	BAS	00	051	GAM	1182	Cole, H.

MUG MDOS Library Disk 22, Revision 01, NOV 82
H/W, HAM & COMMUNICATIONS - 20 tracks

NAME	TYP	RV	SZE	CAT	DATE	AUTHOR/DESCRIPTION
CLOCK.DOC	DOC	00	00C	H/W	1182	Helm, E. Documentation for the following set of six programs which implement a MSM 5832 Real Time Clock.
SWRITECLCK	SRC	00	005	H/W	1182	Helm, E.
WRITECLOCK	OBJ	00	002	H/W	1182	Helm, E.
SREADCLOCK	SRC	00	007	H/W	1182	Helm, E.
READCLOCK	OBJ	00	002	H/W	1182	Helm, E.
STIMECHECK	SRC	00	002	H/W	1182	Helm, E.
TIMECHECK	OBY	00	002	H/W	1182	Helm, E.

MUG MDOS Library Disk 23, Revision 03, NOV 82
BUSINESS - 29 tracks

NAME	TYP	RV	SZE	CAT	DATE	AUTHOR/DESCRIPTION
REIIAP	BAS	00	02A	RET	1182	Chaft, A. Real Estate Income Investment Analysis.

MUG MDOS Library Disk 29, Revision 01, NOV 82
SCIENCE AND ENGINEERING - 16 tracks

NAME	TYP	RV	SZE	CAT	DATE	AUTHOR/DESCRIPTION
DIFFEQ	BAS	00	007	MTH	1182	Cole, H.
DIFFEQ.DOC	BAS	00	018	MTH	1182	Cole, H.
SIMEQ	BAS	00	005	MTH	1182	Cole, H.
SIMEQ.DOC	BAS	00	006	MTH	1182	Cole, H.
INTEGRAL	BAS	00	004	MTH	1182	Cole, H.
INTGRL.DOC	BAS	00	008	MTH	1182	Cole, H.

MUG MDOS Library Disk 32, Revision 01, NOV 82
BASIC SUBROUTINES - 21 tracks

NAME	TYP	RV	SZE	CAT	DATE	AUTHOR/DESCRIPTION
RP2	BAS	00	002		1182	Riding, G. This, and the following three files, show the arithmetic of packing and de-packing multiple logical records to the Micropolis 250-byte physical disk sector.
RP3	BAS	00	002		1182	Riding, G.
RP4	BAS	00	002		1182	Riding, G.
RP5	BAS	00	002		1182	Riding, G.
DATAENTRY	BAS	00	002		1182	Raney, M. A set of routines which force user to enter proper responses.
D-ENTRYDOC	SRC	00	013		1182	Raney, M.
SORT.BAS	BAS	00	008		1182	Smith, B. A Shell sort routine. See newsletter #29.

MUG MDOS Library Disk 34, Revision 01, NOV 82
GAMES - 25 tracks

NAME	TYP	RV	SZE	CAT	DATE	AUTHOR/DESCRIPTION
CODEBRKR	BAS	00	00D	GAM	1182	Cole, H.
CODEMAKER	BAS	00	009	GAM	1182	Cole, H.

LINE57 (LINEEDIT version 5.7)

by Carl J. Singer
6049 North Morgan Street, Alexandria, VA 22312
Phone (703) 354-2904

INTRODUCTION

LINE57 is an extension of LINEEDIT, using all of the enhancements afforded by Robert Manderson's LINEEDIT5. Five new commands have been added:

```
PAGE [l] [s] [m]
COPY L1 L2 L3 [n]
MOVE L1 L2 L3 [n]
DMOVE L1 L2 L3 [n]
CCHANGE [L1] [L2]
```

In addition, all of the MDOS commands which have not been preempted by LINE57 commands (the latter group consists of LOAD, SAVE, MOVE, and PROMPT) are now available directly from LINE57.

PAGE

When using LISTP or PRINTP to print several pages from the current text file on the system printer, a PAGE command preceding LISTP or PRINTP enables the user to produce proper margins at the top and bottom of each page. The form of the command is

PAGE, PAGE <lines>, PAGE <lines> <skip>, or PAGE <lines> <skip> <margins>.

<lines> is the number of Editor lines to appear on each page of a listing before moving to the next page.

<skip> is the line number count (not number of lines) to be skipped when moving from one page to the next. <skip> is normally zero except in multicolumn listings.

<margins> is operative only on systems where the FORMFLAG byte at 4C8 is zero (printer does not accept formfeeds); otherwise it is ignored. See page 2-28 of the manual for a description of FORMFLAG. <margins> is set equal to the number of linefeeds to be issued between the bottom line of one page and the top line of the succeeding page -- i.e., the sum of the top and bottom margins. The default value is six.

If the FORMFLAG byte has been set to a non-zero value (printer accepts formfeeds), PAGE m causes the output of a formfeed after each m lines. Thus top and bottom margins of any size can be obtained on any printer with any size of paper.

The use of the second parameter (skip) is best explained by means of an example:

Suppose the current text file is numbered by tens, and it is desired to print out lines 1000 to 4500, two columns per page, 60 lines per column, on standard 11-inch paper. The first column on page 1 will have lines 1000 to 1590; the second column, lines 1600 to 2190. The first column on page 2 will have lines 2200 to 2790, etc. We begin by issuing a PAGE 60 600 command (600 = line numbering increment times lines per page times [number of columns minus 1]). Now all the first columns will be correct when we issue a LISTP 1000 4500 command.

After the first columns of all three pages have been printed by the above command, we set the printer margin for the second column, back up the paper to the beginning, (if your printer supports reverse vertical tabs, see the note at the end of this documentation) and issue the command LISTP 1600 4500. This will print all the second columns correctly.

The parameters set by a PAGE command remain in effect until changed by another PAGE command. PAGE without a parameter resets LINE57 to its original condition with no printer page controls.

The PAGE command has no effect on commands other than LISTP and PRINTP.

COPY

A block of lines in the current text file can be copied to another part of the file by using the COPY command. The form of this command is:

COPY line#1 line#2 line#3 [renumbering increment]

Line#1 is the number of the first line of the block to be copied, and line#2 is the last line. If these two numbers are equal, the block consists of only one line. Line#3 is the number of the destination line AFTER which the block is to be deposited. All 3 line number parameters must be specified.

After the copy has been made, the file will be renumbered automatically. If a renumbering increment has been specified, say 'n', the renumbering obtained will be the same as if RENUM n n had been called. If no increment is specified, n=10 is assumed.

The destination line number may be any line number in the text file which is not inside the block to be copied. For this purpose, the first and last line numbers of the block are not considered to be inside the block. Thus COPY 100 200 200 is a valid command, but the command COPY 100 200 150 will produce the error message OVERLAP, and no copy will be made.

If there is insufficient memory to hold the larger text file resulting from the copy, the message NOT ENOUGH RAM will be displayed, and no copy will be made.

MOVE

A block of lines in the current text file can be moved to another part of the file by using the MOVE command. The form of this command is

MOVE line#1 line#2 line#3 [renumbering increment]

The MOVE command operates just like the COPY command, except that the moved block disappears from its original position, leaving the file exactly as long as it was before the MOVE.

Nevertheless, a MOVE requires as much memory as a COPY, because at some time during the execution of the MOVE, the block will exist at both the source and the destination. Thus, the combination of a large textfile and a large block to be moved may result in the NOT ENOUGH RAM message. In this case, use the DMOVE command.

DMOVE

From the user's point of view, the DMOVE command works just the same as the MOVE command, except that disk unit 0 must be engaged. When DMOVE is used, the block to be moved is written to disk, and later loaded back into the correct place in the current text file. With the DMOVE command, no additional memory is required for the move. Of course, there will have to be enough free space on the disk to accept the temporary file later scratched by the system.

MOVE should always be tried before DMOVE, since MOVE is executed much faster than DMOVE. Only if a NOT ENOUGH RAM message is displayed, should DMOVE then be used.

CCHANGE

CCHANGE stands for Controlled Change. This command is exactly the same as the CHANGEALL command, except that the user can interactively specify whether each line containing the match string should be changed or left as is. This allows the user to step through the file and selectively change certain strings. When a line containing the match string is found, it is displayed on the console and the user receives a prompt CHANGE ?. If it is desired that the line be changed, a Y is typed for yes; otherwise, an N is typed. The user can exit at any time by typing a Control-C.

Universal Match Characters

The universal match character used in SEARCHes and CHANGEs is set to a question mark (?) when LINE57 signs on, just as in LINEEDIT. However, the user can change it to any other printable character by entering MATCH "X", where X is the new universal match character. In this way, question marks may be searched for or changed. Entering MATCH with no parameter disables the universal match character by setting it to a null.

APPEND

A minor change has been made in the APPEND command. Its form is now

APPEND "filename" [renumbering increment].

If a renumbering increment is specified, the file will be renumbered using it; otherwise the increment 10 will be used.

Short Form Commands

The following shortened forms of the commands may also be used:

LONG FORM	SHORT FORM
CCHANGE	CC
CHANGE	C
CHANGEALL	CA
COPY	CO
DELETE	D
DMOVE	DM
EDIT	E
LIST	L
LISTP	LP
LOAD	LO
MOVE	M
PRINT	P
PRINTP	PP
RENUM	R
SEARCH	S
SEARCHALL	SA

LINE57 still fits in one track on the disk (barely), and is a lot more powerful and useful than the original LINEEDIT. Enjoy it!

Replacement Pages for Manual

A set of manual replacement pages (2 index plus 12 text) embodying all of the described changes to LINEEDIT, is available from me at the above address for $4 ($6 outside of North America).

Multi-column Printing

NOTE: For multicolumn printing of long source code files, it would be nice, after you've printed one column, to have the printer - rather than your poor tired wrists - roll the paper back to the first page. I have provided a means of doing this if your printer supports reverse vertical tabs.

Locations 38F3 to 38F8 contain space for a string up to five bytes long plus a terminating zero. Replace the existing string (1B 59 5F 00 00 00 for an NEC Spinwriter) with the string required by your printer for its longest reverse vertical tab. For my printer, this is 63 lines - almost a page. Now the command RV n will cause the printer to perform this tab n consecutive times. Proper choice of n will get you back to page 1, ready to set up for printing the next column.

This string change can be performed permanently by means of ZAP. Call ZAP d, where d is the drive on which the disk with LINE57 on it resides. Now get the disk directory (track 0) and find LINE57. Immediately following the ten bytes assigned to the filename is a byte which tells you which track that file starts on. This byte has the high order bit set high, so the actual track number (in hex) is

UTILITY PACK #1
==========

Utility Pack #1 is a collection of versatile general purpose routines that can speed program development and make life easier for every programmer and computer user.

The package includes:

* GREP: Searches a list of files for the specified string. Each occurance is displayed as well as the file it was in.

* CMP: Compares two files and displays the differences.

* WORD: Searches a specified file for a string surrounded by alphanumerics.

* TOKEN: Searches a specified file for a string surrounded by white space.

* UNIQ: Removes duplicate adjacent lines in a disk file.

* RM: Erases a list of files.

* AR: Archiver. Puts many files into one large file which has its own directory.

* SUM: Performs a checksum of a list of files.

* WC: Counts the number of words and lines in a specified file.

* SORT: In RAM variable length record shell sort. These invaluable routines can save hours for every programmer and computer user. Keep Utility Pack #1 with your system at all times.

Requires: 24K CP/M, most disks formats available. Also available for CP/M 86 and MS DOS. From Super-Soft.

.....

UTILITY PACK #2
==========

Utility Pack #2 is a collection of general purpose programs that can speed program and system development. Many were originally developed for in house use. Most of these programs feature I/O redirection to console or file. The packet includes:

* TR: Translates one user defined set of characters in a list of files to another set. Character set specification is versatile and may occur either on the command line or be redirected from a file.

* RPL: Replaces every occurance of one user defined string in a list of files with another string.

* DC: Powerful postfix desk calculator program featuring 13 digit precision, transcendental functions and 10 registers.

* DIFF: Compares two source files and displays the minimum number of differences using a backtracking algorithm. Output may be redirected to a file or console.

*PR: A multi-column print formatter. Prints a list of files on multi-column format with headings, page numbering and date. Output may be directed to console, printer, punch or file.

* SLEEP: Causes processing to pause for a specified period of time. Useful in batch processing.

* INUSE: Displays "IN USE" message in block letters on the console.

* CAT: Concentrates a list of files.

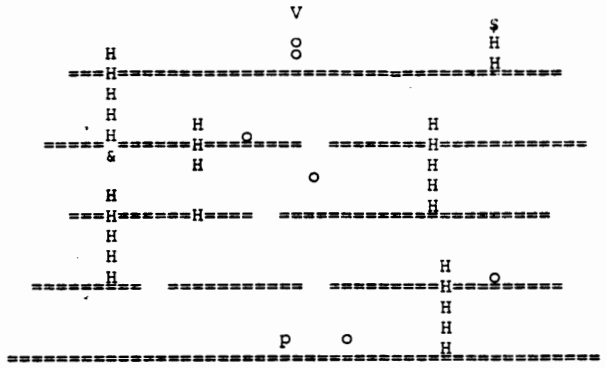
* DATE: Maintains a current date file used in conjunction with PR.

* SPLIT: Splits a file into smaller chunks of specified length.

Requires: 32K CP/M, most disk formats available. Also available for CP/M-86 and MS DOS. From Super-Soft.

.....

LADDER
======



The latest CP/M action game is LADDER. The idea is to move your person across and up the series of floors and ladders, without getting crushed by a rock.

This game also uses the A/Z (up/down) and O/P (left/right) configurations. In addition, the space bar makes your man jump.

When you finally reach the treasure (the $), you get a new screen layout. Neither Lynn nor I can get past the fifth layout. How many points you get depends on how long it takes you to get to the treasure. You also get points for jumping the rocks, and for grabbing the "&"s. LADDER is a distinctly different game from CATCHUM.

As it was with CATCHUM, there are adjustments you can set for the speed of the game, the use of the bell or buzzer, and for having the program make wise-cracks to you.

NOTE

Both LADDER and CATCHUM require CP/M, an 80 x 24 display, and a directly addressable cursor. Vector Graphic Mindless Terminals must be driven by a 3.0 or later monitor. If, when you first turn on your VG, you just get an "*" displayed, you have a version below 3.0.

LADDER lists for $40, is available from DAMAN for $20, or for $19 if you wish the cash discount (check or money order). Add $1 postage to North America, $3 elsewhere.

.....

CATALOG

CATALOG is a master disk cataloging system for CP/M based 8080/Z80 systems. It includes a COMPARE program for checking "like" files found through cataloging.

CATALOG builds and maintains a compressed master data base containg information relevant to each file on each disk. Generating and updating this data base requires only information regarding what disk drive to read and what ID number to assign to the disk. CATALOG permits users to enter short notes for each file and disk in the data base. Data base query accepts filenames, filetypes, "wild cards", partial filenames or disk numbers as search directives.

The information displayed or printed by CATALOG is the most complete available. Disks are displayed complete with the date they were last entered in the data-base, and the space used. File displays include filename, filetype, file size, disk numbers containing that file, user entered notes, and for CP/M 2.x, the user number, system status, and read-only status. A quick summary of all disks is also available which includes disk number, date last entered in the data base, space used, and user entered disk notes.

One seemingly inescapable problem accompanying the addition of disks to an existing system, or the purchase of a disk based microcomputer system, is a gradual increase in the number of disks needed to contain the various purchased and generated files, and a corresponding decrease in a users ability to remember which disk contains a seldom-used file. A supplementary problem also develops; the what-on-earth-is-this-file problem. Most users have dealt with these problems head-on, resulting in stacks of hand-written or printer generated CP/M DIR listings with hastily scribbled notes about the intended purpose of each file. Needless to say, these stacks of paper quickly become unwieldy and disorganized, and updating them becomes a monumental task.

CATALOG has been developed to free the microcomputer user from this drudgery by allowing a means of automatically keeping track of those wandering files. Under user control, CATALOG reads the appropriate disk directories and builds a permanent data-base of pertinent file and disk information, including user entered notes or identifying data for each file and disk. Once created, this data-base need only be infrequently updated by causing CATALOG to re-read the directories of any desired disks.

Having generated a data-base, CATALOG may be used in an interactive mode or may generate one of two formats of CP/M List device output. One output format is an alphabetical listing of file names, grouped by file-type, complete with sizes, and identifying those disks which contain each file. The other format is a disk-by-disk listing which displays the user-entered notes. Used interactively, CATALOG may be used to quickly locate any file or disk and display the data pertinent to that file or disk. CATALOG supports the full range of CP/M "wild" card file specifications.

Additional CATALOG features include the entry of short notes for each file and disk, an option to erase the old data-base and start afresh (switching sort parameters to produce a strictly alphabetical listing if desired), and to erase all references to a particular disk and display the highest disk ID currently contained in the data-base.

The COMPARE program doesn't just check the name and size, as do some "compares". It checks every byte, and prints the differences, if found. This is very useful when you have several versions of a development program on separate disks, and you need to scratch one for the file space.

CATALOG is a product of SRX.

SPECIFICATIONS

Minimum hardware configuration: 24K CP/M 8080/Z80 system with two disk drives

Disk ID labels: Decimal numbers in the range of 1 to 255

Maximum number of Unique Disks: 255

Maximum number of files of any one input disk: Depends upon CP/M memory size; 900 for 24K systems - 1400 for 32K system, etc.

Sort method: Operator selectable at data-base (re)creation time for either FILENAME.TYP sort or TYP.FILENAME (default) sort

Access/Update Time: Totally dependent on size of data-base and type of disk drives. Less than 30 seconds for a 14K data-base consisting of 400 files, each having a user note averaging 15 characters in length, run on an 8" Shugart type single density drive

Merge criteria: Files are entered as separate data-base entries if any of the following criteria are met:

- Different FILENAME.TYP
- Different file size
- Different file attribute bits set in FILENAME.TYP (CP/M 2.0 and later)
- Different USER numbers (CP/M 2.0 and later)

User Notes: 62 Characters maximum per file
62 Characters maximum per disk

SUMMARY OF COMMANDS

[DISKS] - List a summary of all disks contained in the data-base.

[DISKS rrr] - List a summary of all disks in the range specified by rrr.

[ERA nnn] - Erase all references to disk nnn (nnn = 1-255).

[EDIT nnn] - Enter disk and file notes for disk nnn (nnn = 1-255).

[EXIT] - Warm boot back to CP/M.

[FIND xxxxxxxx] - Locates a filename containing the subset xxxxxxxx.

[LAST] - Displays the highest disk number presently contained in the data-base.

[NEW] - Erase old data-base and restart.

[NEW S] - Erase old data-base and restart with swapped sort parameters.

[PRINT] - Preceeds legal list-type command, e.g., PRINT rrr. Diverts list-type output to CP/M List device.

[PRINT=m] - Change nominal printer width to m (60-132).

[d:] - Update data-base by reading directory from disk in drive d: (d = A-P) and identify it as disk nnn using the directory entry -xxxxxxx.nnn.

[d:nnn] - Update data-base by reading directory from disk in drive d: (d = A-P) and identify it as disk nnn (nnn = 1-255).

[filename.typ] - Locate all occurrences of filename.typ and list them along with the disk numbers on which they reside. CP/M wild cards are supported.

[rrr] - List the contents of disk(s) rrr with user supplied notes.

FILEFIX

FILEFIX(TM) is a program for recovering erased files, protecting deleting, and renaming files, as well as forging multiple user links to a single CP/M file.

FILEFIX can perform several different operations on your CP/M directory. The directory can be viewed in detail, accidentally erased files will be identified and may easily be recovered. All operations are performed on the directory itself; data in the actual files will not be altered.

With FILEFIX you may:

1) View your CP/M directory block allocation map

2) Display your files in short form - including ERASED files

3) Display your files in long form with block and sector status

4) Display your disk status completely.

With FILEFIX it's easy to PROTECT a file and CLEAN erased files or FORGE multiple user links to the same file. You may also give several files with the same name, UNIQUE NAMES.

FILEFIX is menu driven and easy to use for the novice as well as the experienced CP/M user. Your FILEFIX disk will not only allow you to recover erased files in the directory - several utilities are provided which allow you to SCROLL or SHOW textfiles one screenful at a time, ENCRYPT files with a one or two letter password, COPY files, RENAME files, VERIFY files and determine the CPU processsor type in your computer. Experienced hackers will enjoy the DUMP facility which includes both HEX and ASCII values. GO may be used to branch to any memory location and IOBYTE allows the user to reassign the four logical devices to various physical devices by changing the IOBYTE HEX value.

Everyone will eventually erase a file accidentally. With FILEFIX you do not spend valuable time starting all over because you have "mistake insurance".

FILEFIX works with CP/M 2.2 and is written by Allen Miller.

.....

ENCODE/DECODE II

Encode/Decode II is a sophisticated coding system for CP/M. The coding techniques used include:

* Transposition
* Inversion
* Traveling XOR's (element relative)
* XOR's with hash generated by user combination

The file is coded block by block. Hence, should equipment error cause a bad data write, decoding error will be local only to that block.

Essentially, one codes files when they are not needed and decodes the files when access is required. Access is inhibited in two ways. First, there is a user defined password that is needed just to execute Encode/Decode II. Second, the user defined combination is needed to decode a file. There are 10,000,000,000 possible combinations.

Accidental decoding is virtually impossible. In fact, we can not even decode a file without its combination! This means solid security.

Also, because each file can have its own combination, multiple security levels are possible. That is, simply assign sets of files the same combination. In this way, each person can be given clearance to only those files that are permitted.

Requires 32K CP/M. Also available for CP/M-86 and MS DOS. From SuperSoft.

.....

ANALIZA II

ANALIZA II is an expanded and "smarter" version of the famous "Eliza" program. ANALIZA II, following in the tradition of its predecessor, imitates a psychiatrist. The user (patient) interacts with the program in a question and answer fashion. ANALIZA II asks questions based on previous responses by the user.

As a simulation, ANALIZA II is very complete. For example, if the user becomes belligerent, ANALIZA II responds with remarks intended to stop this. If the current dialogue is repetitious or dead ended, ANALIZA II prompts the user with new subjects until a new dialogue is achieved. The simulation is so complete that the user even receives a "bill" at the end of the session.

ANALIZA II performs its magic through the use of sophisticated "cracking" algorithms used to break down the patient responses. It looks for key words and phrases in order to construct a reply based on the user's input. A list of subjects is kept in memory, so the longer the session, the more surprises ANALIZA II has in store for the user.

ANALIZA II also stores the user subjects on disks. This means that sessions can be continued day to day, and that the program will become more knowledgeable about each patient.

ANALIZA II provides an excellent example of Artificial Intelligence that will certainly shock those unfamiliar with current computer software technology. It makes an excellent addition to both office and home computer systems.

From SuperSoft. Requires 48K CP/M. Also available for CP/M-86 and MS DOS.

.....

DISK-EDIT

DISK-EDIT is a screen-oriented disk editor designed as a DDT for disk files. DISK-EDIT gives you complete access to all the raw information on your disks; it will let you examine or alter files that cannot be examined with a normal text editor.

DISK-EDIT loads one 1024 byte segment of a disk or file into its internal memory buffer. It displays a window into that buffer on your screen. This window has, in effect, two panes side-by-side. Through the left-hand pane you see the hexadecimal representation of each byte in the buffer that lies within the current window. Through the right-hand pane you see the ASCII representation of those same bytes. A column on the far left-hand side of the screen provides the byte location of the bytes appearing on the screen.

The DISK-EDIT program has two editing modes: hex and ASCII, and you can toggle back and forth between the two (each is contained in its own "window pane"). Changing a byte in one, let's say hex, at a given location will produce a corresponding change in the ASCII character at the same location. Any changes on the screen will also be reflected in the buffer.

Once you are in the DISK-EDIT window, you have a full range of text editing commands at your control, including forward space, back space, next line, previous line, view next screen, view previous screen, beginning of file, end of file, string searching, write to disk, and several others.

DISK-EDIT comes complete with a terminal definition program, and it is configurable for any disk, including hard and double-sided. A uniquely powerful debugging tool, we feel that DISK-EDIT is an invaluable utility for every programmer.

DISK-EDIT is produced by SUPER-SOFT, requires 32K CP/M, and is also available for CP/M-86 and MS DOS.

.....

obtained by subtracting 80H. Thus B4 (for example) becomes 34, which is 52 decimal.

After finding the track on which LINE57 resides, call APP d, which brings back ZAP. Now GET the proper track, being sure to specify (F)ull sectors, and DUMP 4AFD 4B0C. If you got the right track, the string 1B 59 5F 00 00 00 should be there, starting at 4AFD.

If everything is O.K., call ENTR 4AFD, and insert your own string, making sure that it is terminated by a zero. Once more, call APP d, and do a PUT to the right track, for 16 sectors. This will change LINE57 on the disk.

You can, of course, make this permanent change of LINE57 in a simpler way, but it will then occupy two tracks (rather than one) on the disk. Perform the following sequence:

```
LOAD "LINE57"
ENTR 38F3
Enter the correct reverse vertical tab code sequence
SCRATCH "LINE57"
SAVE "LINE57" 2B00 3B27 14
```

LINE57 is available on MUG Library Disk 6.

.....

FORTH UPDATE

For all you MUGies who purchased Acropolis' FORTH, I have been told by George Shaw (Owner of Shaw Labs, the producer of A-FORTH) that you should receive your update within two weeks. This was on November 15, so you should have received your new FORTH disk before you receive this. If not, give me a call and I'll see what I can find out.

.....

CLASSIFIED

FOR SALE: Micropolis 1015 MOD II disk drive. Has all electronics, but no power supply. Same as 1021 MOD II. I am using 4 of these myself & have had no problems. First $200 takes it.

Call (913) 686-2491 after 4:PM C.S.T. Carl Colvin, 108 W. Nichols, RT. 3, Spring Hill, KS 66083

.....

FOR SALE: One Processor Tech SOL with 48K user memory, SOLOS monitor, 1K video memory, 1K system RAM. One Micropolis MOD II disk drive with S-100 controller. One Heath H9 Terminial. The equipment is like new. Make me an offer.

Pete Eversole - (608) 784-9750 days, (608) 788-6677 nights. P.O. Box 1777, LaCrosse WI 54601.

.....

FOR SALE: Pearl III (Program Generator). Lists for $750, sell for $350. 56K CP/M required.

John Jeddeloh, 2207 N.E. 12 Ave., Portland OR 97212 Phone (503) 287-3997.

.....

WANTED: Communication with, and help from, a member who has reasonable access to a SAAB dealer. I need a few parts for mine and there are no dealers in Panama.

Paul Boon, PSC Box 356, APO Miami FL 34002.

.....

```
**************************************************
*         DAMAN'S SALE LASTS UNTIL 12/15         *
*      12 1/2 % OFF DAMAN'S NORMAL LOW PRICE     *
*     SEE LAST MONTH'S NEWSLETTER FOR DETAILS    *
**************************************************
```

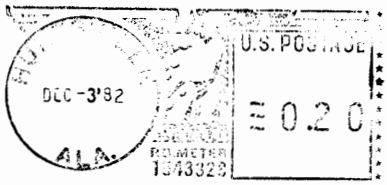
Published Monthly by the MUG
Subscription rates:
U.S., Canada, Mexico; $18/year: Other, $25/year

FIRST CLASS MAIL

FIRST CLASS MAIL

MICROPOLIS USERS GROUP

Buzz Rudow, Editor
604 Springwood Circle
Huntsville AL 35803
(205) 881-1697



FIRST CLASS MAIL