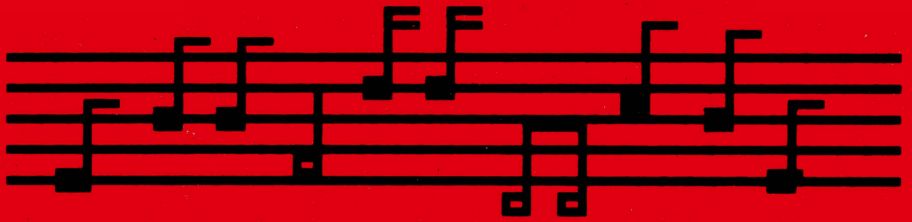

NOTES FROM MISOSYS

ISSUE I - May 1, 1983

NOTES FROM MISOSYS

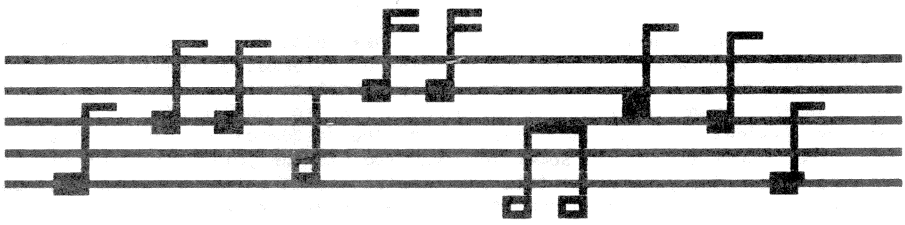

TABLE OF CONTENTS

NOTES FROM MISOSYS is a publication of MISOSYS, PO Box 4848, Alexandria VA 223Ø3.

THE BLURB

This is the inaugural issue of our "newsletter". For some time now, MISOSYS has been working towards publishing information bulletins for our customers. Coming up with a name for this publication has been a problem. We don't want to compete with the LSI publication, so "QUARTERLY" is out of the picture. Besides, we have no idea as to the frequency of this publication. The term, "newsletter", is sometimes overused to the point of loss of its meaning. Since the purpose of this publication is to bring together into one source, information on new products, hints and kinks (an old HAM term) on our existing products, patches and whatnot where necessary, and other programming tidbits, we have settled in on "NOTES FROM MISOSYS".

This first issue is being mailed via FIRST CLASS in order to ensure that those of you having moved since registered will still get this issue. Future mailings will probably be via BULK RATE which does not get forwarded by USPS. Therefore, if your address is incorrect on the label, please let us know of your revised address so that our data base may be updated prior to the next mailing. If you received this issue along with an order, then your registration of the associated product will get you into our data base.

In mid-April, we released two new products. One is the long-awaited and most requested product function of the past few years - release III of our disassembler. DSMBLR III provides the capability of DIRECTLY disassembling an object-code file (/CMD type). We did not want to stop just there. DSMBLR III also automatically partitions the output into two or more files of a size specified by the user. DSMBLR will also prompt to swap destination diskettes when the destination disk becomes full. The disassembler also accepts a data file of screening information which tells it to decompose selected regions as data. It constructs these regions as DW or DB statements as specified (literals are constructed as strings). This is truly an advanced product and will set the standard for disassemblers. DSMBLR III is priced at $40 (+$2 S&H). It will operate under LDOS 5.1 or TRSDOS (model I or III). Okay, you ask, what about all of us out here already owning a DSMBLR II? If you return your DSMBLR II cassette when placing an order for DSMBLR III, you will receive a $10 credit. The trade-in offer expires August 31, 1983.

The second new product is ZSHELL, a command line preprocessor. Inspired by LC and UNIX, ZSHELL adds additional flexibility to LDOS 5.1.x. You will have I/O redirection of *KI, *DO, and *PR controlled for the duration of a program's execution. No longer do you have to ROUTE, execute, RESET, and use the output file. A command line as simple as "DEVICE >STATUS/TXT" will automatically place the display output of the device command into the file, STATUS/TXT. Realize that this now lets you control the *KI input device just as easily. You also get piping so that "DIR :1 (A,I,S,P) |+ LSCRIPT" pipes what would have been the printer output into LSCRIPT. Imagine what you would have had to type to accomplish that feat with standard LDOS commands. You also get multiple commands on a line. For example, "LIB; DEVICE; DIR; FREE" can be typed to schedule the execution of all four commands. ZSHELL is priced at $40 (+$2 S&H).

Effective March 1, we also LOWERED the price of the CON80Z and MSP-01 utility packages. Each has been reduced from $50 to $40. Specifics are discussed under each packages "notes". We also have available at a special

price for a limited time, Volume II of "THE BOOK, ACCESSING THE TRS-8Ø ROM". This book presented an in-depth analysis of device input/output on the Model I computer. It details keyboard, video, printer, and cassette routines as well as provides a commented partial disassembly (comments by address). This book originally sold for $14.95. We have a limited quantity to be sold for $5 (+$1 S&H). Even if you have other books on the Model I, Volume II will be of benefit by the detailed treatment of its subject matter. Even if you have a Model III, the textual material will be advantageous. Get your copy while they last.

By now, you should be aware that LSI was exceedingly busy preparing its 6.Ø release of LDOS. The first machine to receive that advanced system was Tandy's Model 4. Yes, TRSDOS 6.Ø (TRSDOS is a trademark of the Tandy Corp.) is a licensed version of LDOS 6.Ø. LDOS 6.Ø is very different from LDOS 5.1; however, media format is virtually identical! LDOS 6.Ø is 1ØØ% SVC accessible and operates in low memory (Ø-3ØØØH). Thus, MISOSYS has been busy rewriting our product line for operation under LDOS 6.Ø. In order to avoid confusion with existing products, the 6.Ø line of comparable products have different names. For instance, the comparable disassembler version III is named PRO-DUCE III for operation under 6.Ø. Similarly, the 6.Ø comparable EDAS version IV is entitled, PRO-CREATE. We also have a comparable version of the partitioned data set utility entitled, PRO-PaDS. We expect to bring up CMDFILE, CON8ØZ, CONVCPM, GRASP, LC, MSP-Ø1, ZGRAPH, and ZSHELL as time permits. Some of these other products may already be available by the time you read this. So, if you are using a machine that is running LDOS 6.Ø and are looking for the features in software available in our product line, stay in close contact with us for availability. These new 6.Ø releases are separate products. Please do not ask for any "update" offers from our comparable 5.x products.

EDAS VERSION III

Although we no longer produce this version of the Editor Assembler, we will still be supporting it for a period of time. Some users are having a problem, depending on which operating system is being used, with pagination using XREF3 on the model III. The problem stems from Tandy's changeover from the printer driver paging from Ø through 66 to 1 through 67. They apparently never quite understood zero origin mathematics and the problem prevalent in the Model I which attempted to range the printer from Ø through 67 lines was "corrected" in the Model III by restarting the count of lines printed to 1. XREF3 initializes the counter to Ø the first time a printout is requested. Since succeding pages re-initialize the line counter to 1, 67 lines print on the first page and 66 on succeeding pages. The following patch to XREF3/CMD should correct this problem. The patch can be installed by whatever means you have at your disposal to patch CMD programs. Remember, if you have no "zapping" utility, you can assemble the patch as data and use CMDFILE to append the patch code. The patch shown is for the XREF3/CMD file released 11/9/81 or later. Data in brackets "[]" is relative sector and relative byte for the first byte of the patch.

```
. Patch to XREF3/CMD dated 11/9/81 or later
At X'5358', change from "32 29 4Ø" to "CD C7 5A" [RØØ, B87]
At X'5369', change from "42" to "43" [RØØ, B98]
```

At X'5AC5', change from "2Ø 2D 2Ø 4A 6F 62 2Ø 61" to "21 ØD 3C 32 29 4Ø 3D C9" [RØ7, BF4]

EDAS VERSION 4.1

Version 4.1 of EDAS was released September 1st of 1982. This version runs only under LDOS. Although not documented, EDAS 4.1 uses the Extended Cursor Mode (ECM) of LDOS. Since ECM was first introduced in LDOS 5.1.1, EDAS will not function with the 5.Ø series. Another problem is that folk's operating their machine remotely, usually cannot generate the ECM character set. EDAS 4.1 uses the LDOS 5.1.x KFLAG scanner to deal with PAUSE and BREAK. This is done so that TYPE-AHEAD will function properly. The KFLAG scanner does not exist in LDOS 5.Ø or other operating systems. The *SEARCH facility in EDAS 4.1 also makes use of a function existing only in LDOS. These are the primary reasons that EDAS 4.1 works only with LDOS. To help our users that are trying to operate remotely, we have developed a patch that can be applied to EDAS 4.1 to have it work without the Extended Cursor Mode - thus allowing it to function with remote access terminals. The patch is as follows:

```
. PATCH TO EDAS 4.1 TO INHIBIT ECM
DØØ,9B=C9; WAS CØ
DØØ,EA=8Ø; WAS 89
DØØ,F6=Ø8 18 ØA; WAS 91 92 94
DØØ,FC=19; WAS ØØ
DØ3,C1=19; WAS 94
DØ5,Ø2=ØØ; WAS 4Ø
DØ6,EB=5B; WAS '82
DØ6,EE=ØA; WAS 88
DØ7,96=18; WAS 91
DØ7,9A=Ø8; WAS 81
DØ7,AD=5B; WAS 82
DØ7,B1=ØA; WAS 88
DØ7,DØ=19; WAS 94
D11,D6=1B; WAS 92
D11,D9=Ø1; WAS 81
D12,5Ø=Ø8; WAS 81
```

The next series of patches are for various versions of EDAS 4.1. These patches have already been applied to released disks according to serial number. Check the serial number of your diskette (located on the diskette label) and compare to the application date and number identified in the patch. Apply only those you need. There is no harm in applying any patch to any EDAS 4.1 diskette. If your EDAS 4.1 diskette is relatively new, you may not even have to apply a single patch. Consult your LDOS manual under UTILITIES (PATCH command) for instructions in entering and applying these patches. If you choose NOT to apply the patches yourself but would rather return your EDAS 4.1 disk for application by us, there will be a $5 charge. The patch list starts from EDAS42/FIX since the first fix was applied prior to the EDAS4 release.

```
. EDAS42/FIX - Ø8/3Ø/82 - installed 82ØØ27-82ØØ43 & 82ØØ6Ø+
. This fix inhibits KEYIN from accepting the
. ESCAPE character which is only used in line edit.
```

```
.
DØ2,8F=CD ØB 58 FE 92 CØ 18 F8
DØ7,93=C6 59
. end of patch

. EDAS43/FIX - Ø8/3Ø/82 installed 82ØØ27-82ØØ43 & 82ØØ6Ø+
. This fix corrects the -SL assembly switch option
. for correctly displaying symbols that have
. a dollar sign ($) in positions 2-n
D19,FC=ØD 2Ø ØF FE 24 2Ø ØB 3E Ø2 8Ø 85 6F 8C 95 67 18
D1A,ØC=D5 7E CD 38 5B 23 1Ø F9 CD CB 82
. end of patch

. EDAS44/FIX - Ø9/Ø8/82 - installed 82ØØ63+
. This fix corrects the "*PREFIX" assembler command.
. *PREFIX can now be used within a macro to schedule
.   a change to the macro substitution string.
. This fix reduces the # of macro nesting by 1.
D19,24=CC; Correct pointers to MACNEST.
D28,77=CC
D28,92=CC
D2A,8Ø=CC
D2A,96=CC
D2A,8D=9D 57 Ø1 Ø7; Correct pointer to restore & length.
D28,8C=Ø7; Change nest move length to 7
D28,A2=CD B2 81; Add patch to shift one more substitution
D2B,17=11 9B 57 ED AØ C9; byte into current macro area.
. end of patch

. EDAS45/FIX - Ø9/Ø8/82 - installed 82ØØ71+
.  This patch inhibits EDAS from accepting null lines
.  read from a GET or SEARCH file with a blowup due
.  to MODSCAN parsing 256 characters.
D2E,1D=B8 81; At X'84AB' JP to patch
D2B,1D=1A B7 C2 9C 71 C3 95 84; Test line length
. end of patch

. EDAS46/FIX - Ø9/15/82 - installed 82ØØ77+
. Fixes macro processor not terminating on tab.
D28,D1=ØØ ØØ ØØ ØØ ØØ CD C6 81
. end of patch

. EDAS47/FIX - Ø9/15/82 - installed 82ØØ77+
. Corrects ABORT option if X command.
DØ2,23=CE 59
DØ7,34=CE,59
D19,96=ED 5B CE 59 14 2Ø
. end of patch

. EDAS48/FIX - 1Ø/27/82 - installed 82Ø181+
. This fix corrects EDAS47/FIX which corrected the ABORT
. option but caused eject of extra page and title.
. Note: the patch supercedes EDAS47/fix
DØ2,23=CE 59
DØ2,99=ED 5B CE 59 C9
```

```
D07,34=CE 59
D19,96=CD D0 59 7A B3 28
. end of patch

. EDAS49/FIX - 12/18/82 - Roy Soltoff
. This FIX corrects the error handling response if
. KI/DVR is not resident. Patched starting with 820338
.
D00,A0=67 44
D00,C0=0D
. end of patch

. EDAS410/FIX - 01/04/83 - Roy Soltoff
. This FIX corrects the symbol table addition on the
. statement: LABEL <TAB> ;COMMENT
. Patched starting with 820428
.
D12,55=FE 3B C2 63 72 4D C3 28 72
.; WAS 00 00 00 00 00 00 00 00 00
D1B,7E=28 08 C3 54 69 00 FE 3B 28 DE
.; WAS 20 04 FE 3B 28 DA FE 3B 28 C5
. end of patch

. EDAS411/FIX - 01/19/83 - Roy Soltoff
. This fix enables EDAS to trap symbols that are
. too long ( > 15) when appearing in the
. operand field of a statement.
.
D12,5E=79 FE 10 D2 1D 72 AF C3 E9 72
. WAS  00 00 00 00 00 00 00 00 00 00
D14,83=5D 69; WAS  E9 72
. end of patch

. EDAS412/FIX - 01/26/83 - Roy Soltoff
. Applied starting with 820428
. This fix corrects nested IFNE - ENDIF constructs
. Two conditionals were not parsed if within
. a conditionally false.
.
D1B,D9=30; WAS  2E
. end of patch
. EDAS413/FIX - 02/08/83 - Roy Soltoff
. Applied starting with 820471
. This fix corrects assemblies where *GET files
. contain source lines of 127 or 128 characters.
.
D03,0E=99; WAS  96
D2D,32=78; WAS  7A
D2D,6A=7B; WAS  7D
. end of patch
```

The next patch does not correct a bug. It enhances EDAS IV by speeding up the assembly of source code that uses MACROs. A notable improvement would be recognized by the LC users during the assembly phase. MACRO names are stored in the symbol table. EDAS always searches the symbol table starting

from the most recent entry. Since MACROs are usually entered near the beginning of a program, EDAS had to constantly search through the bulk of the table before matching up the MACRO name when a MACRO was invoked. This patch adds a second pointer which is updated only when a MACRO is added to the table. The MACRO search uses this new pointer; thus, MACRO search time is much shorter. Uniquely, as the length of the symbol table grows, this new method provides a greater percentage reduction over the unpatched EDAS. The sieve program discussed later under LC was assembled with a 14% decrease in assembly time!

```
. EDAS414/FIX - Ø4/18/83 - Roy Soltoff
. This patch increases the speed of assemblies
. when MACROs are involved. The greater the
. source files, the greater the speed increase.
. First applied #82Ø614
D12,68=73 23 72 2B ED 42 2B 22 CE 57 C9
. WAS  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
D12,73=22 B4 56 22 CE 57 C9; WAS ØØ ØØ ØØ ØØ ØØ ØØ ØØ
D17,E2=CD 72 69; WAS 22 B4 56
D1B,94=CE 57; WAS B4 56
D26,4F=CD 67 69; WAS 73 23 72
. End of patch
```

The most asked question concerning EDAS 4.1 (and asked question refers to someone suspecting a bug) concerns the display of "total errors". Now for the explanation of the "total errors" message.

The assembler has up to three phases (or passes) when assembling. The first phase assembles each instruction but provides no output. This phase is used to generate all addresses of symbolic labels. Note that any forward reference (referencing a label prior to its definition) results in an error stroke. Phase 2 is used to provide a listing - this phase can be suppressed with -NL. Phase 3 generates object code.

In EDAS version 4.1, the "total errors" is shown after either the listing pass or the object code pass. The error display is NOT suppressed if you inhibit both phase 2 and phase 3. Thus, the error count you get is the total number of forward references plus any other assembly errors. You cannot just "A-NL" and expect to see a proper error total. Perhaps EDAS should have inhibited the "total error" message if you suppressed both pass 2 and pass 3. In any event, suppressing both pass 2 and 3 serves no useful purpose. It is a carryover from early releases of EDAS 3.4 and other assemblers that perform a second pass on "-NL" to accurately get an error count while suppressing the display. Kim Watt (of Breeze/QSD) originally suggested that we completely inhibit the listing pass on "-NL" so that large assemblies (i.e. from multiple disk files such as an assembly of Super Utility) do not have to waste time reading files just to get an error total.

One other problem may exist with "large" programs or where a long editing session loads and deletes a number of files. If you haven't realized by now, EDAS 4.1 normally uses unnumbered files unless overridden by the user during the "W" command. Files are automatically numbered during the loading process. You may find that that a simple "d t,b" may not clear out the text buffer IF THE STATEMENTS ARE NOT IN LINE NUMBER ORDER! EDAS consecutively numbers statements on loading. Nothing resets the counter except the

SHIFT-CLEAR function, a reNumber command, or a fresh entry into EDAS. Therefore, if you continue to load, edit, and delete files, you may find that the line numbers wrap around past 65529 to ØØØØ3. It is important to occasionally reset the line counter by either entering SHIFT-CLEAR with an empty text buffer or renumbering the buffer when loaded with text.

We have occasionally been asked why not have a *GET assembler directive that only operates during the first pass. It would be useful to load EQUATE files. A file containing only EQU statements is often used; however, EDAS "wastes" time - not to mention disk wear - in reading the EQU files two or three times per assembly. Well, you can control this function more easily than EDAS. From Jim Frimmel (of LC noteriety) comes a suggestion of how to accomplish this wondrous feat. You can set up your own "pass" counter and conditionalize your *GET of the EQUate file. The following code does this:

```
PASS  DEFL  PASS+1
      IF    PASS.EQ.1
*GET  MYEQU
      ENDIF
```

The value of any undefined symbol is zero. Therefore, the first time this code is executed, the symbol PASS is set to a value of one and the *GET statement is assembled causing the EQUate file to be included. Subsequent passes set PASS to two and three respectively. Thus, the conditional is FALSE for all but the first pass. This, by the way, is another use for the DEFL pseudo-OP. If you use this procedure, you will lose the use of IFDEF and IFNDEF conditional pseudo-OPs for any symbol defined in the EQUate file. Now you ask why can't I use this on MACRO files (obviously after trying to no avail)? EDAS 4.1 has a few conditionals (IFDEF, IFNDEF, and IFREF) that require some hairy and elegant implementation. They make use of flag bits that indicate DEFINITION and REFERENCE on each pass of the assembler. EDAS restricts the redefinition of MACROS and makes use of these flag bits. If a MACRO file was only loaded once, the MACRO would show up as being undefined on subsequent passes because the DEFINITION bit needs to be set during each pass to define the macro.

PDS VERSION 1.Ø

PDS has been around since December 1981. It adds some interesting and useful capabilities to LDOS versions 5.Ø and 5.1. PDS stands for Partitioned Data Set and permits the bundling together of executable CMD files into one file - each individual file now termed a member of the PDS. PDS stores the members adjacent to each other and keeps a directory in the front of the one file. This winds up saving disk space as there is no need to maintain granule boundaries for each member. Small utility programs can sometimes take up only a few sectors of disk space. The average file will always waste a few sectors due to allocation in granule units (a granule is a contiguous quantity of sectors on a track - the quantity varies with the size and density of the disk). We have had reports of up to 25% reduction in storage requirements when maximum use is made of PDS-type files. Remember, PDS can also store data members that are accessible from PDS utilities.

PDS does its thing without using any high-memory space. It uses what we consider to be a very elegant process to accomplish its functions. When you BUILD a new partitioned data set, PDS adds a small program, termed the front end loader, to the beginning of the file during the initialization process. A member and ISAM directory immediately follow this loader. The loader executes from X'52ØØ' to X'52E1' - quite short. The loader makes use of a little known fact that when a program executes, the file control block used to access the program file is left in an OPEN state upon transferring control to the program (in this case the loader). The front end loader then parses the command line looking for a member specification. Finding one, it continues to read the program file as if it were data. It is thus reading the member directory. After it finds the member entry, it extracts appropriate loading information and interfaces back to the LDOS loader - which continues to load and execute the member requested. This is the reason that you cannot have DEBUG active during the execution of a un-protected PDS file - DEBUG would load after the front end loader and alter the FCB that the front end loader would need to access the directory. Although PDS itself is copyrighted, you may freely disemminate your own partitioned data sets that are created using PDS and have only this loader (plus your own members, of course).

Since its release, PDS has had surprisingly few bugs. Herewith are a few patches that correct known problems.

```
. PATCH TO PDS/CMD.PDS
. NOTE: This patch was applied quite some time ago
.       and is most likely in your copy. Check with
.       the LIST command to ensure your PDS is patched.
.       Patch corrects PDS(COPY) of members less than
.       one sector in length.
DØD,BC=11 EB 55 C8; WAS C8 11 EB 55
. End of patch
```

This next patch is a new one and corrects a problem of PDS(APPEND) when the file you are trying to append is a null file. This occurrence is obviously one where you would not want to append; however, you may inadvertantly try to append a null file and we surely would not want PDS(APPEND) to try to append it. Since this is an X-patch, you will have to patch PDS by first copying APPEND to workspace, patching the separate file, killing and purging the APPEND in the PDS, then appending the patched copy back to the PDS. Do this with a BACKUP copy of PDS. This is detailed as follows:

1. BUILD the PDSA/FIX file with:

```
. PDSA/FIX - PATCH TO THE APPEND MEMBER OF PDS
X'5499'=CD 3D 59
X'593D'=11 8A 58 2A A5 54 AF ED 42 CØ 21 4D 59 C3 2E 56
        ØA 41 7Ø 7Ø 65 6E 64 69 6E 67 2Ø 66 69 6C 65 2Ø
        69 73 2Ø 6E 75 6C 6C 21 ØD
. End of patch
```

2. Execute the command, "PDS(A) PDS(APPEND) APPEND"

3. Execute the command, "PATCH APPEND PDSA"

4. Execute the command, "PDS(K) PDS.PDS(APPEND)"

5. Execute the command, "PDS(P) PDS.PDS"

6. Execute the command, "!APPEND APPEND PDS.PDS"

Note specifically the exclamation point in step 6 - it is required! You now have corrected the PDS(APPEND) member.

A question frequently asked concerning PDS is the use of the MAP parameter. First let's explain what MAP is all about and why it is here in the first place. When PDS was being designed, one projected use of it was to store the libraries provided by LC - our C compiler. We had a distinct need to be able to load one member but provide more than one entry point to that member. This correlates to similar functions in the LIBrary modules of LDOS. For example, the COPY command and the APPEND command of LDOS both execute from the same member of SYS6. COPY and APPEND are two different entry points. The method of specifying multiple entry points to a single member is provided via the MAP option. Rarely will a user want to store a CMD file that has multiple entry points as a member of a PDS. Therefore, we weren't too specific concerning the exact syntax to use for this process - although the PDS documentation does show what a MAP entry should look like.

For a multiple entry member, there is only one object file to load yet each entry requires a name and an entry point. Thus, the syntax of a MAP record looks like this:

filespec,member1,traadr1,member2,traadr2,...

The first field specifies the name of the file that is to be loaded. "Member1" and "traadr1" is the first member name and its entry point. Subsequent fields identify each member name (another entry to filespec) and the respective entry point. For example, suppose you had a file called, COPY/CMD, which contained two entry points: APPEND at X'52ØØ' and COPY at X'52Ø3'. The MAP record to append such a file would be:

COPY,APPEND,52ØØ,COPY,52ØØ

If you are using the PDS file to store assembler libraries and you have multiple entry points to an assembler source code member, what do you do about the transfer address? There is no such thing for the source! Well, since the syntax requires a transfer address, we must enter something; however, it matters not what we enter - a zero will suffice.

While we are talking about libraries, our PDS users have questioned us on the reference in the documentation to using CMDFILE to extract LIBrary commands from SYS6 and SYS7 and append them into a PDS. This was mentioned to have user-constructed libraries. Unfortunately, we never really told you exactly how to do this feat. The LDOS manual mentions that CMDFILE recognizes the LIBrary files, SYS6 and SYS7, and can read in each member individually. Some time ago, Earle Robinson (of SoftERware) had written an article on extracting the individual LIBrary members. Exactly what happened to that information is unknown. In case you never saw it, let me bring you up to date.

NOTES FROM MISOSYS

If you execute a LIB command, you will see LIB <A> and LIB <B> commands displayed. LDOS 6.x users will also see LIB <C>. The names of each command represent the entries to members in SYS6 and SYS7 (also SYS8 for LDOS 6.x LIB <C>) respectively. The command interpreter which resides in SYS1 compares your command entry to a table which contains ISAM codes for each LDOS LIBrary command. It is these codes that are needed in CMDFILE to extract one of the LIBrary members. Rather than have you waste the time to search SYS1/SYS and decode the table, here is a list of the codes:

SYS6-LIBA	SYS6-LIBA	SYS7-LIBB	SYS7-LIBB	SYS8-LIBC
31-APPEND	41-LIST	51-ATTRIB	72-PURGE	B1-FORMS
32-COPY	81-LOAD	11-AUTO	A1-SYSTEM	B2-SETCOM
61-DEVICE	1E-MEMORY	33-BUILD	16-TIME	B3-SETKI
21-DIR	53-RENAME	17-CLOCK	1A-TRACE	A2-SPOOL
91-DO	63-RESET	13-CREATE	1B-VERIFY	
66-FILTER	64-ROUTE	15-DATE		
18-KILL*	82-RUN	14-DEBUG		
19-LIB	65-SET	71-DUMP		
62-LINK	A2-SPOOL	22-FREE		

* LDOS 6.Ø command name is "REMOVE"

What more can you do with PDS? MISOSYS generally has imaginative users. When it comes to PDS, the most imaginative has been Scott Loomer of MicroConsultants West. Scott has put together a package he calls "PDS Tutor" which not only presents a handful of non-typical uses, it also provides a technical treatise on the directory structure maintained by PDS. If you are interested in some unusual uses for PDS, get in touch with Scott at 315 Palomino Lane, Madison WI 537Ø5.

Incidentally, the April 1, 1982 issue of THE LDOS QUARTERLY had a piece on the various types of records in load modules. This information was in Roy's Technical Corner. It documented the load records in a PDS as well as the others typically used in LDOS.

ZGRAPH VERSION 4.Ø

The ZGRAPH package has been pretty solid. One minute bug crept in to the BINCONV/BAS program, though. Under certain conditions, the EDAS-compatible output file would be constructed wrong. It is necessary to correct four BASIC statements: lines 565, 61Ø, 78Ø, and 91Ø. The lines should read as follows:

```
565FORT1=1TO16:T2=TØ*16+T1:IFT2>LEN(A$)THENPRINT#2,CHR$(13);:GOTO57ØEL
SEPRINT#2,FNSS$(ASC(MID$(A$,T2,1)));:GOSUB61Ø:NEXTT1:PRINT#2,CHR$(13);
:IFT2<LEN(A$)THENGOSUB6ØØ:TØ=TØ+1:GOTO565
61ØIFT1<16ANDT2<LEN(A$)THENPRINT#2,",";:RETURN:ELSERETURN
78ØFORT1=1TO16:T2=TØ*16+T1:IFT2>LEN(A$)THENPRINT#2,CHR$(13);:GOTO79ØEL
SEPRINT#2,FNSS$(ASC(MID$(A$,T2,1)));:GOSUB91Ø:NEXTT1:PRINT#2,CHR$(13);
:IFT2<LEN(A$)THENGOSUB9ØØ:TØ=TØ+1:GOTO78Ø
91ØIFT1<16ANDT2<LEN(A$)THENPRINT#2,",";:RETURN:ELSERETURN
```

GRAPHIC SUPPORT PACKAGE

GRASP, by its nature of providing a special graphics printer driver, must interface directly to the printer port. This may be a problem if the machine you are using does not exactly correspond to the peripheral interfaces of the Model I or III. Such is the case with two machines - the Video Genie by EACA and the MAX-8Ø by Lobo Drives. The Video Genie is a Model I work-alike; however, instead of interfacing the printer through a memory-mapped address of X'37E8', the Video Genie uses a Z-8Ø port, X'FD'. On the other hand, the MAX-8Ø is a Model III work-alike under LDOS but uses memory-mapped address X'37E8' whereas its Model III counterpart uses port X'F8'. Why oh why do these "work-alikes" not work alike! Herewith are patches to ALTCHAR/DVR and ALTCHAR/CMD:

```
. PATCH FOR ALTCHAR/DVR FOR VIDEO GENIE
DØ5,5F=D3 5D ØØ; WAS 32 E8 37
DØ8,ØF=DB FD ØØ; WAS 3A E8 37
. End of patch

. PATCH FOR ALTCHAR/CMD FOR VIDEO GENIE
DØØ,35=D3; WAS 32
DØØ,38=FD; WAS E8
DØØ,3B=ØØ; WAS 37
DØ2,C2=DB FD ØØ; WAS 3A E8 37
. End of patch

. PATCH FOR ALTCHAR/DVR FOR MAX-8Ø
DØ8,19=32 E8 37; WAS D3 F8 ØØ
. End of patch

. PATCH FOR ALTCHAR/CMD FOR MAX-8Ø
DØ2,CC=32 E8 37; WAS D3 F8 ØØ
. End of patch
```

For the purists out there, there is a minor correctable problem with GPD/DVR. When the driver is installed, it does not initialize the first byte of the Device Control Block. If GPD was applied to the *PR device, then there was never a problem since the TYPE byte would already be properly established. The following patch forces the TYPE byte to an X'Ø6' in all cases.

```
. Patch to correct GPD/DVR version 5.1a to cause GPD
. to set the DCB type byte when installed on devices
. other than *PR. Use this patch only with GPD 5.1a.
. After patching, version number becomes 5.1b.
. New driver installed starting with #4ØØØ82.
DØØ,21=22 ØC 53 11 ØA 53 ØØ ØØ; WAS 7D 32 ØC 53 7C 32 ØD 53
DØØ,3Ø=C3 4E 52; WAS 23 DD E1
DØØ,4B=22 D1 52 11 CF 52 ØØ ØØ; WAS 7D 32 D1 52 7C 32 D2 52
DØØ,65=DD 36 ØØ Ø6 C3 32 52; WAS 21 CF 52 ED BØ FB C3
DØØ,94="b, 12 April 1983"; WAS "a, 22 April 1982"
. End of Patch
```

Incidentally, the GPD driver module used on the Model III interfaces to the ROM which access the printer via port X'F8'. Therefore, GPD is unusable on the MAX-8Ø. This should not present a problem for MAX users as the standard LDOS printer driver should be adequate for graphics use.

One thing that we had expected with GRASP was to find our users submitting character sets. It is known that some excellent character sets have been developed. Witness the review in 8Ø Microcomputing and the Hebrew and Greek character sets done by Charlie Knight. Also, some users local to the MISOSYS area have showed some interesting graphic results created from customized characters using GRASP. Well, here's the pitch. If you have developed an interesting and useful character set for use with GRASP, send us a copy to combine into a diskette of fonts. We will then make the fonts available to all our GRASP users.

SOLE - MODEL I DDEN BOOTING

SOLE has been without problems since its inception [assuming that the instructions were followed exactly]. Since SOLE was written prior to Tandy introducing their Model I double density controller adaptor, the original SOLE could not support Tandy's DDEN [if the controller handshaking was different from the other manufacturers]. Since the release of SOLE, we had developed a SOLE patch to support Tandy's controller. This patch was provided with the SOLE shipments starting July 28, 1982. If you have received SOLE prior to this date, then you need the patch. The patch is to be applied to SOLE2/CMD prior to using SOLE2. Patch is as follows:

```
. SOLE2RS/FIX - Ø7/28/82 - By Roy Soltoff
. This fix modifies SOLE2 to work with the Radio Shack
. double density adaptor modification.
. Apply with the command: PATCH SOLE2 SOLE2RS
DØ4,CC=ØF EE AØ 32 EE 37 C9 ØØ ØØ ØØ ØØ ØØ ØØ
. WAS  Ø7 Ø7 F6 FE 32 EC 37 3E DØ 32 EC 37 C9
. End of patch
```

We have come across one problem with the installation of SOLE that merits discussion. Step 7 of the SOLE implementation instructions states that, "You can place the diskette in the drive of your choice - it doesn't have to be drive Ø". This is in reference to the booting diskette just prior to running the SOLE2 program. SOLE2 can do its job on the booting disk in other than drive Ø. However, a slight snag develops if drive Ø is a slow stepping drive and the diskette is placed in a fast stepping drive for the SOLE2 process. SOLE2 installs a double density driver onto the boot track of the booting diskette. In doing so, it copies most of the Drive Code Table (DCT) data for the requested drive and will use this data for booting. Therefore, if the drive was set for 12ms or 6ms step rates but your zero drive can only handle 2Øms, the disk will not boot. The documentation should have stated that you can place the disk in the drive of your choice PROVIDED IT IS SIMILAR TO DRIVE Ø IN STEP RATE.

CON8ØZ - 8Ø8Ø TO Z8Ø TRANSLATOR

The early shipments of CON8ØZ did not have a patch applied that is needed to correct the translation of the JMP and JP instructions. Please check your disks and apply the following patch as necessary.

```
. PATCH TO CON8ØZ/CMD VERSION 1.Ø
DØ7,A4=Ø4; WAS Ø3
DØ7,B4=ØC Ø2; WAS ØØ Ø5
DØ3,8C=C3; WAS E2
DØ4,7D=ØØ; WAS C9
. End of patch
```

You should also be aware that CON8ØZ is now priced at $4Ø. If you do substantial 8Ø8Ø programming, or have access to the CP/M public domain library with extensive 8Ø8Ø source code, you may want to explore CON8ØZ.

CONVCPM - CONVERT FILES FROM CP/M

This utility is used to transfer files from selected CP/M media to LDOS media. The program handles standard 8" single density diskettes and 5-1/4" diskettes formatted 128-byte sectors with 18 sectors per track. The current version of CONVCPM is 1.3. This version was released starting with serial number 2Ø. If you have an earlier version, you may want to return it for an update. There will be a $5 handling charge.

A CONVCPM user in England requested information on modifying CONVCPM to handle 5-1/4" media that was formatted 16-single density 128-byte sectors per track. Others may be interested in the patch. It is as follows:

```
. PATCH TO CONVCPM VERSION 1.3 TO USE 16 SPT.
DØØ,EØ=1Ø; WAS 12 (# of bytes in TRANSLATE table)
DØ1,ØØ=27 1Ø; WAS 28 12 (DCT Bytes 8 & 7)
. End of patch
```

With the availability of CP/M on the MAX-8Ø and obvious CP/M installations for the Model 4, we may consider writing a version of CONVCPM to handle 5-1/4" 256-byte sectors - both single and double density. The stumbling block to this job is the non-standard CP/M media. If you have access to documentation specifying the exact configuration of such media, we would appreciate forwarding us a copy. What is needed is the OEM name, diskette format, sector numbering, sector translation table, and the length of the directory. Also, any deviation from standard directory data is important. Address such data to MISOSYS, c/o Software Development.

MSPØ1 - PARMDIR - DOCONFIG - DOAUTO - MEMDIR

If you didn't notice in THE BLURB, the price of MSPØ1 has been lowered by $1Ø effective March 1. This may now be the time to look into this very interesting package. DOCONFIG provides some interesting capabilities. Many

LDOS users have been attracted to the concept of configuring a system diskette. Because of that growing popularity, more users note that in order to change an existing configuration, you either have to issue RESET commands or re-BOOT the system while holding the CLEAR key (to inhibit the CONFIG/SYS file). Now issuing RESET commands is not always the greatest as not every high-memory module will be able to re-establish itself in the same spot. Therefore, upper memory is wasted.

It sure would be nice to be able to establish a configuration and change to another easily. Les Mikesell came up with a program called SYSGEN/CMD that could do just that. There were limitations to what SYSGEN/CMD could do. One of our customers wanted to be able to re-configure while running Job Control Language. SYSGEN couldn't do that. Another wanted to set up a JCL file to establish a particular configuration then "SYSGEN" it for later use without having to exit the JCL procedure. SYSGEN couldn't do that either. To satisfy these needs, we put together the DOCONFIG program. DOCONFIG is unique in that it provides these two functions. You can save or restore a configuration while within a JCL execution. DOCONFIG also interfaces to the LDOS "SYSTEM (SYSGEN)" module in SYS7/SYS so that any change to the data tables there will automatically be part of DOCONFIG. DOCONFIG also provides for saving the state of an LBASIC program execution while the program is executing so that it may be re-established at some future time.

PARMDIR is a very sophisticated data base program that uses the on-line directories as a data base. It was originally designed to generate JCL files based on directory interogatories. For instance, a massive RENAME of all /TXT files to /SCR could be easily established via JCL with one PARMDIR request without having to enter a myriad of "RENAME filespec1 to filespec2" commands. Here is an interesting PARMDIR command:

```
PARMDIR /asm:3 genasm:Ø (a="l ",x=";a $nam:2-nl;dt,b"
```

which is used to construct a JCL file to assemble all /ASM files resident on drive 3. All that is needed is to add the statement, "edas (jcl,abort)" to the resulting file. What that strange command line does is select all files on drive 3 with an extension of "ASM". It then outputs records to the file, GENASM/JCL:Ø. Each selected file generates three records: the first is "l filename/asm:3", the second is "a filenam:2-nl", while the third is "dt,b". Three records are output because of the semi-colons in the command statement. Other complex examples are at your fingertips.

PARMDIR had one problem reported to us where the drive being referenced was drive 7. Obviously, the bug would show up only by a hard-drive user (why didn't we catch that one?). In case you need to reference drive 7, apply the following patch (apply it even if you don't need to access drive 7).

```
. PARMDIR1/FIX - Ø3/Ø1/83
. This patch permits PARMDIR to address drive 7.
. A limitation is that files designated SYS in
. the directory of drive 7 will not be classed
.  as SYS files.
.  Applied Ø3/Ø1/83 - Starting 23ØØ61
X'5535'=CD B2 61; Call the patch
X'61B2'=ØF BØ 77 3C CØ CB A6 C9
. End of patch
```

CONTRIBUTIONS

We will attempt to make "NOTES FROM MISOSYS" more than just a presentation of patches. Our goal is to make it "newsy" and stock it with "freebies" - useful programs for our readers. What follows is a boldface filter for use with the Radio Shack DMP series of printers. We ripped the guts from our Daisy Wheel boldface filter andd rewrote what remained to deal with the boldface handshaking of our relatively new DMP-500 (27 31 to enable boldface, 27 32 to disable boldface). The filter has two distinct toggle characters used to alternate the boldface condition. If either toggle character is detected when bolding is off, it turns it on. If on, receipt of a toggle character turns off the bolding. A tilde ( ) **is used to toggle and** output a space in lieu of the toggle character. This permits its use in LSCRIPT text while still permitting right justification. The other toggle character is X'7F' entered by depressing <CLEAR><SHIFT><ENTER> (with the LDOS KI driver). A command such as:

filter *pr dmpbold (t="%",n=x'1e')

changes the toggle-with-space to a per cent sign and the toggle-with-nil to an X'1e'. The filter follows in HEX format and can be converted to an FLT file by using BINHEX/CMD.

```
05 06 44 4D 50 42 4F 4C 1F 1D 43 6F 70 79 72 69 67 68 74 20 28
43 29 20 31 39 38 32 20 62 79 20 4D 49 53 4F 53 59 53 01 02 00
52 D5 1A F5 E5 21 CB 52 CD 67 44 3A 25 01 FE 49 20 18 21 11 44
22 6D 52 22 9C 52 22 A8 52 21 8A 42 22 C6 52 21 54 44 22 2E 52
E1 11 5D 53 CD 76 44 C2 C2 52 F1 CB 5F C2 B6 52 CB 67 C2 BE 52
CB 4F CA BA 52 DD E1 DD 6E 01 DD 66 02 22 8D 53 21 7E 00 24 25
7E 20 01 7D 32 97 53 32 C6 53 21 7F 00 24 25 7E 20 01 7D 32 9B
53 32 CA 53 2A 49 40 22 80 53 DD E5 DD 21 D8 53 11 D7 53 B7 ED
52 44 4D 3E 05 DD 6E 00 DD 66 01 5E 23 56 EB 09 EB 72 2B 73 DD
23 DD 23 3D 20 EA DD E1 ED 5B 49 40 21 D7 53 01 5A 00 ED B8 ED
53 49 40 13 F3 DD 73 01 DD 72 02 FB C3 2D 40 21 22 53 DD 21 35
53 DD 21 4B 53 DD 21 11 53 CD 7B 44 C3 30 40 44 4D 50 2D 35 30
30 20 42 4F 4C 44 46 41 43 45 20 46 69 6C 74 65 72 20 2D 20 56
65 72 73 69 6F 6E 20 31 2E 30 0A 43 6F 70 79 72 69 67 68 74 20
31 39 38 33 2C 01 E4 00 53 20 62 79 20 52 6F 79 20 53 6F 6C 74
6F 66 66 0A 0D 50 61 72 61 6D 65 74 65 72 20 65 72 72 6F 72 21
0D 44 65 76 69 63 65 20 6E 6F 74 20 61 63 74 69 76 65 21 0D 4E
6F 74 20 61 6E 20 6F 75 74 70 75 74 20 64 65 76 69 63 65 21 0D
44 65 76 69 63 65 20 69 73 20 72 6F 75 74 65 64 21 0D 54 4F 47
47 4C 45 4F 52 54 20 20 20 20 20 4F 52 4E 55 4C 4C 20 20 5E 52
4E 20 20 20 20 20 5E 52 00 18 0A 00 00 07 44 4D 50 42 4F 4C 44
28 03 C3 00 00 F5 3E 00 B7 20 2F 79 FE 7E 28 23 FE 7F 28 03 F1
18 EB E1 0E 1F 18 04 E1 AF 0E 20 32 91 53 C5 0E 1B BF CD 8C 53
C1 BF 18 D4 CD A7 53 18 03 CD A2 53 0E 20 18 DA 79 FE 7E 28 EF
FE 7F 28 D9 FE 0D 20 CD CD A7 53 0E 0D 18 C6 AB 53 B2 53 B9 53
BE 53 D2 53 02 02 00 52
*CE
```

LC - C-LANGUAGE COMPILER

LC has been a big hit with our users. Just about everything being fed back from our users to date has been positive. A couple of points are worth mentioning here. Some users may be having trouble obtaining a copy of THE C PROGRAMMING LANGUAGE by Kernighan and Ritchie. Due to the growing popularity of the C language, Prentice Hall has been working hard keeping that book in print. Besides that, they raised the price last November. We keep a small supply of this publication in stock to satisfy our retail customers. We sell K&R retail for $18 plus $1 shipping.

Another point that needs mentioning is the tremendous job being done by Earl Terwilliger, Jr. with the LC Interest Group (LCIG). If you are an LC user and have not been in contact with Earl, you are surely missing out on some good stuff. The LCIG already has a handful of diskettes containing C programs in source code. These are all compatible with LC and are public domain - all that's necessary is to join the LCIG. Contact Earl C. Terwilliger, Jr., 647 North Hawkins Ave., Akron OH 44313.

LC was released in October of 1982. Since that time we have uncovered some bugs while our users have advised us on some others. In spite of these "minor" mishaps, we are very pleased with the quality of the package. LC constitutes a complete 35-track single density diskette. That's almost 85K of code. EDAS 4.1, which is included with the package, constitutes more code on another diskette. The LC/EDAS development system is a monumental work and we are proud of all those that have had a part in LC's birth. Jim Frimmel, the author of LC, has been busy researching reports of problems with the compiler or libraries. Jim is readying an update which will be provided FREE to our LC users. We hope to complete the testing of version 1.1 by the early part of May for shipment in mid-May. Therefore, you may want to start sending your master LC disk in a protective mailer to us. DO NOT RETURN THE EDAS DISKETTE (unless you want us to apply the EDAS4 patches at a charge to you of $5). We will regenerate your master diskette and return it to you in the mailer you use to send it to us. Please address the mailer to: MISOSYS, Attn: LC Update, PO Box 4848, Alexandria VA 22303-0848.

While the LC diskettes are being exchanged, you may want to ascertain the need of applying patches in the interim. We have installed a few patches to the LC diskette since its release. There is at least one additional patch that you may want to install. Confirm whether the following patches have been applied to your diskette and patch accordingly.

```
. LC1/FIX - 10/09/82 - PATCH TO LC/LIB.LC
. This patch corrects a problem in FCLOSE. It was
. added to serial # 920069 and from 920079 on.
D3F,67=42 43; WAS 48 4C
D3F,76=42 43; WAS 48 4C
. End of patch
```

Another change made on 10/15/82, was to correct the "#option ZVAR". This is correctable by changing the "$VAR" macro located in LCMACS/ASM to read as follows:

```
$VAR  MACRO #NAME,#SIZE
```

```
        $SORG
#NAME   EQU     $$STEMP
        IF      @_ZVAR
        DC      #SIZE,Ø
        ELSE
        DS      #SIZE
        ENDIF
        $PORG
        ENDM
```

Also change the first line of the file to read "; 1Ø/15/82".

There also was a patch applied to the compiler, LC/CMD, on 12/18/82 starting with serial # 92Ø218. If your LC is a prior number, you may want to apply the following patch:

```
. PATCH TO LC/CMD.LC TO FIX VARIOUS PROBLEMS.
. APPLIED 12/18/82 STARTING FROM 92Ø218.
. PATCH TO CORRECT EOF DETECTION.
D2C,Ø2=FF FF; WAS 84 ØE
. PATCH TO CORRECT FUNCTION RETURNING A POINTER VALUE
D5Ø,EB=BC; WAS CB
. PATCH TO CORRECT ALLOCATION OF LOCAL POINTER ARRAYS
DØE,8A=ØØ; WAS Ø1
. End of patch
```

Now that the patches are out of the way, let's examine what else iis locally correctable while you are awaiting the update. Three functions were inadvertantly omitted from the floating point library. These are FINT, FLOG, and FSGN. You may easily add them to any program that needs them by adding the following code to the LC/ASM file. Insert this code just before the "IF @_FPLIB" statement.

```
        IFREF FINT
FINT    LD      HL,ØB37H
        JP      @FNF
        ENDIF
        IFREF FLOG
FLOG    LD      HL,8Ø9H
        JP      @FNF
        ENDIF
        IFREF FSGN
FSGN    CALL    @SVS
        #GA     HL
        CALL    9B1H
        CALL    ØAEFH
        CALL    98AH
        LD      HL,(4121H)
        RET
        ENDIF
```

While we are talking about the Floating Point library, one other problem came to light recently. It seems that the use of FFIX causes the resulting application to crash - a most undesirable feature. No, we did not plan it that way. You can correct this malady by applying the following patch to

FP/LIB.LC.

```
. PATCH TO FP/LIB.LC TO CORRECT FFIX.
DØ5,2B=4C 44 Ø9 48 4C 2C 3Ø 42 32 36 48 ØD Ø9 4A 5Ø Ø9 4Ø 46 4E 46 ØD
. WAS  43 41 4C 4C Ø9 3Ø 42 32 36 48 ØD Ø9 4A 5Ø Ø9 4Ø 46 4E 46 ØD 1A
. End of patch
```

We must have had a problem in generating a small batch of LC disks. There were a few returned for problems - all in a small range of serial number 92Ø249. If you are having some extreme problems (like reboots and crashes when trying to run LC) and thought that it was your C program, it may be a bad diskette. If your serial number is close to 92Ø249 and you have severe problems with LC, hold on until receiving the update.

Before we go any further in identifying the additional problems reported to us that will be fixed in the update, let's turn to some examples of LC programming. The first is a good illustration of the Shell sort. The sort routine is essentially straight from K&R (page 58). However, what is sorted and the method of display is the useful part of the illustration. The SEESHELL program generates 1Ø24 random displayable characters in video RAM. It then proceeds to sort them directly in video RAM so that you can observe the status of the sort by eyesight.

```
/* seeshell/ccc by Roy Soltoff */
#include stdio/csh
#option args OFF
#option redirect OFF
#option maxfiles Ø
#option fplib
main()
{
   char *v, v1[4], v2[4];
   int i,n,c;
   v = Øx3cØØ;             /* set to CRT area */
   n = 64*16;              /* set to CRT size */
   cls();
   itof(94,v2);            /* set random upper */
   for (i=Ø; i<1Ø24; ++i)
      {
      frnd(v1,v2);         /* get random number */
      *v = ftoi(v1) + 32; /* set char 32-126 */
      ++v;                 /* bump pointer to next loc */
      }
   shell(Øx3cØØ,n);
   c=getchar();
   cls();
   exit(Ø);
}
shell(v,n)                 /* sort v[Ø]...v[n-1] */
char v[];
int n;
{
   int gap, i, j, temp;
   for (gap=n/2; gap > Ø; gap /=2)
      for (i=gap; i<n; i++)
```

```
            for (j=i-gap; j>=Ø && v[j]>v[j+gap]; j-=gap)
               {
               temp=v[j];
               v[j]=v[j+gap];
               v[j+gap]=temp;
               }
      }
      cls()
      { puts("\x1c\x1f"); }
```

We have gotten a few questions concerning the use of the Floating Point library. It is understandable! Although the manual describes the method of interfacing, the lack of examples is clearly a drawback. The first example in K&R coincidentally happens to be a problem requiring floating point. If you are new to C, you probably just bypassed that problem and continued on. There is really a great deal that can be done in integer arithmetic. The use of floating point as interfaced in LC does not necessitate a great deal of time; however, a few examples will obviously help you over the hurdle. The following is presented as one example of the Fahrenheit to Celsius program described on page 8 of K&R but written using the FP library in LC.

```
/* fctab - print Fahrenheit-Celsius table
   for f = Ø, 2Ø, ..., 3ØØ */
#include stdio/csh
#option fplib
main()
{
   int lower, upper, step, fahr;
   char celsius[4], fivedivnine[4], temp[4];
   char thirtytwo[4], celsius_str[8];
   lower = Ø;   /* lower limit of temperature table */
   upper = 3ØØ; /* upper limit */
   step = 2Ø;   /* step size */
/*
   calculate 5.Ø/9.Ø
*/
   atof("5.Ø",fivedivnine); /* float 5 */
   atof("9.Ø",temp);        /* float 9 */
   atof("32.Ø",thirtytwo);  /* float 32.Ø */
   fdiv(fivedivnine,temp);  /* calc 5.Ø/9.Ø */
/* */
   fahr = lower;
   while ( fahr <= upper )
   {
      itof(fahr,celsius);  /* float fahr */
      fsub(celsius,thirtytwo); /* fahr - 32.Ø */
      fmul(celsius,fivedivnine);  /* (5.Ø/9/Ø)*(fahr-32.Ø) */
      ftoa(celsius,celsius_str); /* result to ASCII */
      printf("%-6.3d        %-8.8s\n",fahr,celsius_str);
      fahr += step;
   }
}
```

Aside from setting up the floating point arithmetic as functions, we chose to take the division of 5.Ø by 9.Ø out of the scope of the "while" loop. If

"5.Ø/9.Ø" is kept within the scope of the while loop, the result will be slower execution time as the calculation is performed repeatedly (it is possible that an optimizing compiler would recognize the calculation to be a constant - perform it once then use the stored value).

ATTENTION LC USERS

The following are problems brought to our attention that have already been corrected by the preceding patches and changes or that will be remedied in release 1.1. IF YOU ARE AWARE OF ANY OTHER BUG, PLEASE CONTACT MISOSYS IMMEDIATELY:

1. Problem with fclose() most visible when trying to run the CAT/CCC demo program while concatenating two or more files.

2. The #option ZVAR did not work as specified.

3. The compiler could not detect the end of a /CCC source file unless the file was terminated with an X'1A'. This would present difficulties when using LED or LSCRIPT to prepare source files.

4. A function that returned a pointer value would not return the correct value.

5. Local pointer arrays would not have proper space allocated.

6. The function, fint(v1,v2), was omitted from FP/LIB.

7. The function, flog(v1,v2), was omitted from FP/LIB.

8. The function, fsgn(v1,v2), was omitted from FP/LIB.

9. The function, ffix(v1,v2), improperly executed.

1Ø. The atoi() function does not accept a leading plus sign prefixed to the integer string.

11. A memory block allocated with alloc() larger than 32,767 bytes would be incorrectly allocated, and could result in a bad crash if free() and alloc() are alternately called after allocation of a large block.

12. Comparisons of the form "(-32768 < 32767)" are improperly calculated.

13. The fprintf() and printf() functions can't handle the value, -32768. fprintf() and printf() would also not print correct values for hexadecimal numbers over 8ØØØH.

14. Problem with switch-case when multiple "case" statements appear on a line. Also, a case statement had to be the first statement within a switch statement body, or the result of the switch expression would be lost. The value is now retained even if a case is not the first statement. Furthermore, the "default" was required to be placed as the last part of the switch-case-default construct. This requirement will be eliminated

5. The fgets() function is removing the newline character (X'ØD') from the terminal end of the string contrary to K&R page 155.

16. Catastrophe exists if you specify both "#option FIXBUFS" and "#option MAXFILES Ø".

17. An expression such as:

```
*(ptr=ADDRESS) = Ø;
```

where ptr is a character pointer, resulted in a byte being stored, instead of a character.

18. Static functions were not allowed to have arguments. When the declaration of the arguments was encountered, LC would produce the message, "unmatched arguments".

19. The function, cursor() uses arguments reversed from the order specified in the LC manual on page 4-34 and B-1. The x,y values shown on page 4-35 are correct. The proper funtion invocation is: cursor(col,row);. Please correct your documentation.

2Ø. A pointer expression with the indirection operator outside of parentheses, such as this:

```
val = *(pfunc(x)+3);
```

would sometimes result in an extra call to @gint, resulting in a wrong value.

21. LC would generate a signed comparison when comparing two pointer expressions in certain cases.

22. The fopen() function would open an additional file beyond that specified in the MAXFILES option when #option FIXBUFS ON was specified. If the additional file was opened, a crash would result.

23. LC would abort without any error message at all if there was an error in opening LC standard I/O files (usually due to a large amount of high memory usage). It will now generate an error message.

24. The fgets() and gets() functions would lose one character if the maximum line size was reached in an input.

25. If you specified #option NOREDIRECT, LC also suppressed ARGS.

26. LC did not permit the reuse of an existing variable name within a new block (routine within braces). This was not considered to be a bug but an LC limitation. This restriction will be removed.

26. Finally, LC does not generate an X'1A' character in its output /ASM file.

We have had requests from LC users as to its speed compared to other compilers and languages. This is a tough question to answer. At best, we turn to a standard test program used in the industry and attempt to correlate the

results. A most often used test program is the classic, Sieve of Eratosthenes. This algorithm locates the prime numbers from one to some upper limit by elimination of multiples of prime values from a sequential list. This example is especially useful in light of an extensive test of the sieve processed under many compilers, languages, operating systems, and computers and discussed in a recent article appearing in BYTE magazine ["Eratosthenes Revisited - Once More Through the Sieve" by Jim Gilbreath and Gary Gilbreath, BYTE, January 1983].

The sieve program written in C was adapted to LC and executed on a standard Model III running LDOS (with type-ahead active). The execution time was 1Ø6 seconds. This compares to times of 19Øs (MMSFORTH 1.9 on Model I), 288Øs (BASIC on Model III), 478Øs (Disk BASIC Model III), 25.4s-53.2s (various C compilers running under CP/M on 4MHz Z-8Ø CPUs). The test program used for our timing was as follows:

```
/* Eratosthenes Sieve Prime Number Program in C */
/* Adapted from BYTE January 1983 */
#include stdio/csh
#option REDIRECT OFF
#option FIXBUFS
#option MAXFILES 1
#option INLIB
#define size 819Ø
   char time_of_day[9],flags[8191];
main() {
   int i,prime,k,count,iter;
   time(time_of_day);                        /* get start time*/
   printf("1Ø iterations starting -> %s\n",time_of_day);
   for(iter = 1; iter <=1Ø; iter++) {        /*do program 1Ø times*/
      count=Ø;                               /*prime counter*/
      for(i = Ø; i <= size; i++)             /*set all flags true*/
         flags[i] = TRUE;
      for(i = Ø; i <= size; i++) {
         if(flags[i]) {                      /*found a prime*/
            prime = i + i + 3;               /*twice index + 3*/
/*          printf("\n%d",prime);*/
            for(k=i+prime; k<=size; k+=prime)
               flags[k] = FALSE;             /*kill all multiples*/
            count++;                         /*primes found*/
         }
      }
   }
   time(time_of_day);                        /*ending time*/
                                             /*primes found on 1Øth pass*/
   printf("\n%d primes ending -> %s.",count,time_of_day);
}
```

Programming in C is really quite easy. It is a great language for writing all sorts of utility programs. As an example, we had a need to determine the length of the longest line in an assembler source file. This was needed to run down the problem corrected in EDAS413/FIX. The big question is not so much how to write it, but what language to use. BASIC would be very slow and kludgy. Assembler would be fast and a program could be thrown

together from piece parts. C was chosen because it would be very easy to do. Besides, the LC manual already had a program that opened files and read the character stream. A quick modification of COMPARE/CCC from page D-4 of the LC manual produced the following program:

```
/* linelen/ccc */
#include stdio/csh
int line, c1, longest;
FILE *fp1;
main (argc,argv)
    int argc, *argv;
{   if (argc!=2)      /* program name & file to read*/
    {   puts("Format error:  linelen file1 \n");
        exit();
    }
    line = Ø;  longest = Ø;  /* initialize variables */
    fp1 = getfile(*++argv);  /* get filespec */
    while ((c1 = getc(fp1)) != EOF ) /* until EOF */
    {   ++line;              /* bump line length */
        if (c1 == EOL)       /* test for longest at EOL */
            { if (line > longest ) longest = line;
              line = Ø;
            }
    }
        printf("Longest line is %d",longest);
}
getfile(fname)
    char *fname;
{   char *fp;
    if ((fp=fopen(fname,"r")) == NULL)
    {   printf("Open error - %-2Øs\n",fname);
        exit();
    }
    else return fp;
}
```

EPILOGUE

MISOSYS will be considering submissions for the next issue of "NOTES FROM MISOSYS". If you feel you have any item to contribute, please contact Roy Soltoff. Also, why not let us know what you think of these "notes" as an information source. The next issue will be published as soon as sufficient information is garnered.

Published by MISOSYS
P. O. Box 4848
Alexandria, VA 22303-0848