Table of Contents

NOTES FROM MISOSYS is a publication of MISOSYS, Inc., P0 Box 4848, Alexandria VA 22303.

THE BLURB

This is the third issue of NOTES FROM MISOSYS - later than anticipated, as usual. For those of you who are new to NOTES, it is our way of continuing the support for each of our products. MISOSYS firmly believes in product support. We are making every attempt to produce products of extremely high quality and excellent value for your dollars - quality in the software, the documentation, and the support. That's the kind of environment we strive for with the product itself. The NOTES publication is just one example of after sale support to the customer. It is issued on an as-required basis with a goal of three issues per year.

We still have copies of NOTES Issue I and Issue II. These are available for purchase. If you do not have one of the issues and wish to obtain one (or both for that matter), please send us $2 per issue [$3 if non-US] and request the issue or issues of your choice.

Speaking of notes, I have decided to correlate each issue of NOTES with "DISK-NOTES". DISK-NOTES will contain all of the FIX files and all of the programs that are contained in NOTES. It will be released on 40-track double density LDOS 5.x - TRSDOS 6.x media. If you want to get the DISK-NOTES corresponding to an issue of NOTES, submit a blank diskette in a mailer with $2 [$4 outside North America] and request the issue. Alternatively, remit $6 [$10 outside North America] and I will supply the disk and mailer. Getting DISK-NOTES will probably save you the time and energy of keying in lines of code. This is just another service to our customers.

The next thing that we usually discuss in NOTES is a summation of our new products. The last mailing of our catalog to all customers was in November of last year. We recently mailed our new 84-1 catalog to all new customers. The catalog added information on our two new products, ADE at $50 and IFC at $30. To give all of our customers the benefit of this info, NOTES will repeat that info at the end of THE BLURB. The catalog also describes LC 1.2 in more detail (i.e. adds the additional functions to the list). The LC section in this issue also covers this material.

The one additional item that is new is our Daisywheel II proportional spacing filter (DW2PS/FLT). This item is so new that it was implemented after our catalog went to press. This filter supports proportional spacing on the Tandy Daisywheel II. Any program that provides printer output to the standard *PR device will work with this filter. This entire publication was prepared with SCRIPSIT and our DW2PS filter. The filter supports BOLDFACE on the DW. It is also transparent to control codes. Our retail price is $25 on this neat item. Check with us on its availability if you are interested.

We are also making our imprinted blue vinyl 3/4" 3-ring binder available for purchase. This high-quality binder holds about 75 sheets of 60lb stock. It has a tab holder on the spine and a pocket on the inside front cover. The price for the MISOSYS 3/4" 3-ring BINDER for 8-1/2" x 11" sheets is $10

If you are into the Model 4, you should investigate our PROfessional software series. All products that are named beginning with "PRO" are for you. We currently have quite a selection of top-notch software for use with this DOS. There is PRO-ADE, a two-tier directory structure; PRO-LC, a C-language compiler and Macro-assembler; PRO-CREATE, a Macro-assembler; PRO-CESS, a load module maintenance utility; PRO-CON80Z, a utility to translate 8080 source files to Z-80; PRO-CURE, a utility to transfer files from CP/M media to LDOS; PRO-HELP, a help facility of reasonable size; PRO-IFC, an interactive file control tool; PRO-MACH2, a file allocation package that lets you put-it-where-you-want; PRO-GENY, a collection of four unique support utilities; PRO-PaDS, a partitioned data set utility; PRO-ZCAT, a disk cataloger; PRO-ZGRAPH, a pixel graphic editor and printing

package; and the PROGRAMMER'S GUIDE. We also have the VRHARD hard disk driver package for both Model I/III LDOS 5.1 and LDOS 6.x.

By the time you are reading this issue of NOTES, our TRSDOS to CP/M file transfer utility should be available. LCOPY is written by Richard N. Deglin. The package includes a version for use with CP/M+ on the Model 4 and with the MAX-80 under CP/M or CP/M+ [other machine versions may be implemented]. If you have need to transfer files from TRSDOS media to CP/M+, contact us about LCOPY's availability and pricing.

If you are into the old reliable Model I/III (including the Max-80), MISOSYS has an excellent selection of software products. Virtually all of our PRO series are available in Model I/III packages. In fact, we even have a few packages that are not available yet in the PRO series. Let me say that if you are currently using Microsoft's M-80 assembler which generates relocatable code files (/REL) or FORTRAN-80, COBOL-80, or BASCOM, you need MLIB. MLIB is a librarian for creating, building, managing, and maintaining relocatable libraries. MLIB, as supplied by MISOSYS, picks up where Microsoft left off. Since Microsoft has never supplied their librarian as is done with the price-inflated CP/M packages, TRS-80 users have never had the librarian capability. You can get it now with MLIB - as well as get readable documentation which describes the REL format. MLIB works with LDOS or TRSDOS.

When you read through this issue of NOTES, you will find offers to update CONVCPM, ZSHELL, ZGRAPH, LC 1.1, and PRO-CREATE 4.1a. Other products may have patches identified to either correct known bugs or add improvements. Unless otherwise specified, any time that you want to return a diskette to get the most recent copy of your version release (i.e. 1.1, 1.2, 1.3, etc. and not 1.1 to 2.1), instead of applying the patches yourself, you may return the diskette in a protective mailer with $5 per diskette.

Now let me repeat my statements of the last issue concerning sending diskettes in the mail. Radio Shack makes a disk mailer (Cat No 26-1317) which a lot of you use for mailing 5-1/4" diskettes. This is a good mailer. Somewhere I have seen the statement that the mailer is designed for ONE, repeat ONE diskette. Why is this so? There appears to be sufficient depth for a single diskette in the interior of the mailer. When you try to squeeze more than one disk into the mailer, the constant over-pressure tends to compress the edges of the diskette. A diskette compressed in this manner sometimes has difficulty in freely turning in the diskette jacket because of the jacket's pressure on the media. Other types of mailers - both homemade and commercial - may also exhibit this tendency.

Sometimes I dread offering an update. Why? Because I get diskettes returned in packaging from plastic boxes taped shut with yards of tape to plain envelopes with absolute nothing to protect the diskette. I have received reports from users getting an update returned where the diskette is unreadable. Invariably, the difficulty is caused by the inability of the diskette to turn stemming from the jacket being compressed. Therefore, whatever you use to mail diskettes, make sure that the wrapping does indeed protect your diskettes. There is no reason for you to have to incur a delay in getting a needed and desired update - as well as no need for us to have to duplicate a second time.

Now then, for those that happen to get a diskette that appears to be unreadable because the diskette does not turn in the jacket, let me remind you of the ball point pen trick (Bill Schroeder first clued me in to this). Hold the diskette on edge and gently rub the barrel of a pen along the edge. Do this for all four sides of the diskette. Invariably, the edge compression will abate and the diskette will turn freely.

Another problem that has come up concerns only our non-US retail customers and the payment for merchandise ordered with foreign checks [a foreign check is a check that is not drawn against an American bank]. Since our commercial bank has started to deduct an excessive fee to clear foreign checks, we will no longer accept checks drawn on a foreign bank as payment for retail orders. Our foreign customers are urged to use either MasterCard or VISA credit card accounts, International Postal Money Orders, foreign bank checks/drafts drawn against an American bank, or cash sent certified and registered (we would prefer any of the first three methods over cash).

Speaking of our non-US customers, are you aware that MISOSYS products are available directly from dealers outside of the United States. By arranging your purchase direct from the dealer in your country, you may find much faster service and less hassle from customs. The following are MISOSYS authorized dealers:

COMPUTEREISE
55 Elizabeth Street
Brisbine, Queensland - 4000
AUSTRALIA

JSOFT
P.O. Box 1437
Winnipeg, Manitoba R3C 2Z4
CANADA 204-942-0963

E. C. Data, Inc.
Tormevangsvej 88
Birkerod DK-3460
DENMARK

L. A.& H. A. Lewis
10 Rudd Place
Blackett, New South Wales 2770
AUSTRALIA

Peter Caritato & Assoc., Ltd.
18, Kolonaki Square
106 73 Athens
GREECE 01/3619379

MOLIMERX LTD.
1 Buckhurst Road, Town Hall Square
Bexhill-on-Sea, East Sussex
ENGLAND [0424] 220391/223636

JMG Software International
710 Upper James Street
Hamilton, Ontario L9C 2Z8
CANADA 416-389-6086

TELEBYTE COMMUNICATIONS CORP.
20 Tanglewood Drive
Nepean, Ontario K2H 6P3
CANADA 613-828-9432

The time has come to address an issue of policy. We have received some requests for special deals when a customer wants to purchase a software package for both a Model I and a Model 4 (or a Model III and a Model 4). We have finally reached a decision to provide some degree of accommodation for requests of this nature, If you are registered with Model I/III or a PRO-series product, you can purchase the EQUIVALENT alternate package without documentation (i.e. diskettes only) for 75% of the package price. The alternate package must be equivalent. For example, PRO-LC release 1.2 and LC release 1.2 are equivalent packages. PaDS and PRO-PaDS are equivalent packages. DSMBLR III and PRO-DUCE are equivalent packages. MSP-01 and PRO-GENY are NOT equivalent packages. Orders placed in this manner must include the registration number of which the registration card must be in our file. Alternatively, if you want to purchase a Model I/III package and the equivalent PRO package at the same time, you can order both program diskettes and one set of manuals while saving 25% of the second package price.

Now for the specials. When we issue NOTES, we provide special pricing for certain products for a limited period of time, This practice is to reward our loyal customers with the opportunity to stock up on quality products at tremendous savings. These specials will be strictly for our retail customers reading NOTES FROM MISOSYS. These specials will last from now until July 31st, 1984 (non-US customers have until August 30th). Please note that the price does not include shipping. Our normal shipping charges apply.

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$            SPECIAL PRICES             $
$            UNTIL 07/31/84             $
$                                       $
$   EDAS or PRO-CREATE .........$75     $
$    [note: Z-80 QRC not included]      $
$   CMDFILE or PRO-CESS ........$25     $
$                                       $
$   MLIB.........................$25     $
$                                       $
$   PaDS or PRO-PaDS ...........$25     $
$                                       $
$  You must mention NOTES to qualify    $
$                                       $
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Here's one last special offer. You should all be familiar with Karl Hessinger's work. Karl has written a number of packages for us. For at least nine months now Karl has been hacking away at a disk editor. In fact, he started working on it before LSI announced FED II. The Disk EDitor [or DED for short] is a sector-oriented screen editor that gives you control of editing a disk on a cylinder/sector basis instead of a file basis. The catch is that DED's command syntax and screen display are virtually identical to FED version I. The biggie with DED is that it is written in C using the LC compiler. Versions of DED are available for the Model 4, Models I/III under LDOS, and the MAX-8O. DED comes on a 40-track double density LDOS disk and includes the COMPLETE SOURCE CODE. So how do you get DED? It's easy. From now until August 31st, 1984 you qualify for a copy of DED by purchasing any one of the following programs: ADE, GRASP, IFC, LC, MACH2, ZCAT, ZGRAPH, ZSHELL or any of their equivalent PRO-series packages. Only one copy of DED to a customer. If you qualify you must ask for your free copy of DED. Don't want to "qualify" but still want DED? Almost as easy, send one diskette in a Radio Shack mailer with return postage and we will dupe you a copy of DED and return your package to you. Again, this offer is good until August 31st, 1984. After that date you may just have to buy DED.

Don't forget that we are also open to receive small articles concerning the use of any MISOSYS product. We will make every attempt to publish your item in the next issue. Also, contributed programs are acceptable.

For those newcomers to the C language, you might just learn a little if you get ahold of Paul Chirlian's latest book, INTRODUCTION TO C, by Matrix Publishers, Inc. Paul has a knack for keeping down to earth in his examples and discussions of C code. I'll modestly mention that LC was one of the compilers used to prepare the text [although MISOSYS is still misspelled in the book's preface].

Speaking of books, if you happen to have a few hundred bucks burning a hole in your pocket and want to get your very own copy of THE SOURCE, it should be available from Logical Systems. THE SOURCE is the complete commented source code to their 6.2 release of LDOS (alias TRSDOS 6.2 and LS-DOS 6.2). It's published in three volumes (the system, the library, the utilities) at $95 per book or a special price for all three. Now the set is not for everyone. I would recommend it for software developers, system houses, and the curious. I would also recommend it for libraries and educational institutions teaching operating system development. Check with LSI at 414-355-5454.

Before I close THE BLURB, I have two last things to mention. Our customers appear to have responded favorably to the colorful diskettes we now use to distribute software packages. For some

time now we have used the dark blue for Model I/III software and bright red for Model 4 software. We are also planning to use bright yellow for Model 2000 packages. These diskettes are packaged by Cenna Technology. Other colors are available. If you are interested in purchasing blank diskettes in clear plastic boxes of ten, we will make these available. Give us a call or drop a line if you wish to order these diskettes. They can be available in the following styles. RAINBOW PACK: two each of red, orange, yellow, green, light blue; BUSINESS PACK: two each of dark blue, grey, maroon, tan, brown; COLOR PACK: one each of the above colors; also available in ten-box of one color including white and lavender. Our price will be $28 plus shipping (and sales tax where applicable). The diskettes are certified for one sided double density 40 tracks.

The other item is an interest on our part to obtain our customers' opinions of the smaller 5-1/2 by 8-1/2 documentation packages that appear to be prevalent in the MS-DOS world. Tandy is also using this size for Model 4 software. I believe that the reasoning behind the smaller packages has been that they are easier to store in bookcases. They also do not take up as much space on a desk when open. On the negative side, they cannot display as much information on a page unless the print is reduced. If you have occasion to call us or write us, keep this question in mind and give us your opinion.

Now that all the business stuff is out of the way, there are some of you out there who are looking for an update on Stacey. As I write this, Stacey is almost 11 months old. She weighs in about 20 pounds - is tall and thin. Thus, she always seems to be needing clothes that are for a three month older child. She has seven teeth (three lower and four upper) and is working on her fourth lower tooth. She could still use some more hair. Being blond, it really takes some time to notice it. I'm not sure if her typing has improved. Witness this: ---------> e| ntxxvvvvvvvbbcc <---------. Now that vertical bar must have been a toughie!!! Stacey and Brenda spent Easter/Passover in Florida while I was left home to work on the house. Stacey was taken to see her grandparents, her great grandparents, and some cousins along with uncles and aunts.

NOTICE: MISOSYS, inc. expects to be moving to larger quarters between June 21st and June 30th. Our new telephone number will be 703-450-4181. Our new mailing address will be:

MISOSYS, Inc.
P.0. Box 239
Sterling, VA 22170-0239

That's it for this issue. Stay tuned to this channel.

ADE and PRO-ADE by Karl A. Hessinger and Roy Soltoff

If you are using large-capacity storage devices and are always running out of file slots, you need assistance. If your files remain relatively static, our Partitioned Data Set utility is most likely the solution to your problem. However, if your files are dynamic, ADE comes to your rescue. With ADE, you can emulate smaller-sized "floppy" drives on your larger drives. ADE creates files that look like disk drives. The ADE driver interfaces these "floppy files" to the DOS via a drive number. ADE supports as many "floppies" as you can fit on your physical disk drive. With each additional ADE floppy, you gain from 48 to 252 file slots. Look at the ease of invoking the ADE drive emulator driver by responding to some very basic questions:

```
How many "slots" for this driver <1-8> ? 4
Slots already in use <.-.-.-> Enter slot number ? 1
What is the drive emulation filespec? > floppy1:7
Enter the drive size <5,8> ? 5
Single or Double density <S,D> ? d
Enter the number of sides <1,2> ? 1
Enter the number of cylinders <35-96> ? 40
```

ADE then proceeds to invoke the DOS formatter which is used to create the directory information. Each "floppy" is created using the ADE/DCT driver and its brief series of questions. The driver is totally configurable. Once the driver is installed, a utility program called ADE/CMD is available to manage your "floppy" linkage. Existing "floppies" may be linked to the slot table or the existing table linkage may be altered to connect an emulated floppy to the DOS drive table. For example, invoking ADE/CMD displays the slot linkage:

```
Slot        Drive       Filespec
===         ====        =======
1             4         FLOPPY1/ADE:7
2                       not linked
3                       ADE1/ADE:7
4                       not linked
<M>ove, <L>ink, <A>saign, <U>nlink, <D>isable or <E>xit ? 1,4,ade2:7
```

Notice the input response of "1,4,ade2:7" which links slot 4 to the ADE2:7 file. ADE's command entries and operands may be strung together as illustrated or entered individually via prompts. Multiple ADE/CMD commands may even be entered directly from DOS level.

ADE "Floppies" are part of your system. DEVICE and FREE commends illustrate the transparency of these emulated floppies:

```
:4   [FLOPPY1  ]  5" Floppy #0,   Cyls= 40, Dden, Sides=1, Step  6ms, Dly= 1S
:5   [ADE1     ]  5" Floppy #2,   Cyls= 40, Dden, Sides=1, Step  6ms, Dly= 1S

Drive :4 FLOPPY1    03/14/64      Free Space =  174.00K/   180.00K Files = 126/128
Drive :5 ADE1       03/10/84      Free Space =  139.50K/   180.00K Files = 104/128
```

ORDERING INFORMATION:

ADE: For the TRS-80 Models 1/111/4 with LDOS 5.1.
PRO-ADE: For use with LDOS/TRSDOS Version 6 [i.e. Model 4].

IFC and PRO-IFC by Karl A. Heminger

IFC stands for Interactive File Controller and that's just what it does - IFC controls a drive's files interactively! IFC is great for managing the files that collect on a drive. It presents you with a scrollable sorted directory on your video display screen and a collection of operations to tag/untag selected files in the list. With IFC, you can copy, delete, rename, or list files - all interactively and based on your specifications for tagging. The more files you keep, the more you need IFC!

Enter [FC and tell it what drive to work with. Something similar to the following display of files will be presented:

```
Drive : 2   Name : TRSDOS61   Free :  6K          Tagged :     0K
================================================================
==>  BACKUP/CMD:2                  10-Feb-84              3K      :
     BASIC/CMD:2                   10-Feb-84             22K      :
     BASIC/OV1:2                   10-Feb-84              3K      :
     CLICK/FLT:2                   10-Feb-84              1K      :
     COMM/CMD:2                    10-Feb-84              3K      :
     FORMAT/CMD:2                  10-Feb-84              5K      :
     KSM/FLT:2                     10-Feb-84              1K      :
     MEMDISK/DCT:2                 10-Feb-84              3K      :
     PATCH/CMD:2                   10-Feb-84              3K      :
```

That "==>" file cursor points to the current file and can be positioned to any file. This "current" file can then be copied, deleted, listed, or renamed as easily as pressing a key. The IFC-list operation pauses as the listing fills the display screen. IFCLI5T can also be used as a stand-alone program.

Any file in the list can be tagged simply by moving the file cursor to point to the file and press the <T>ag command. A mass of files can also be tagged according to attributes such as MODIFIED, INVISIBLE, or PaDS files. Tagging can even be invoked according to an ambiguous file specification commonly known as a WILDCARD. In addition, you can even specify a second disk drive end tag those files that are on both drives thereby providing an OLD/NEW tagging specification. The copy, delete, and rename IFC commands can be invoked on the mass of tagged files. This operation really improves the file maintenance capabilities available at your fingertips.

Forget the command nomenclature? Help is available on-line! As easy as typing <ENTER>, [or four other characters to make it easy], IFC gives you:

```
A    - Again, retag files          C    - Copy current file
O    - Tag old files               D    - Delete current file
T    - Tag current file            L    - List current file
U    - Untag current file          R    - Rename current file
W    - Wildcard tag/untag
+1*  - Tag by attributes           E    - Exit to DOS
==   <M>ass functions  ==          F    - Free space on drive
C    - Copy files                  H    - Display help
D    - Delete files                Q    - Execute DOS command
R    - Rename files                S    - Select new drive

                    Press any key to continue
```

ORDERING INFORMATION:

IFC: For the TRS-80 Models I/III/4 with LDOS 5.1.
PRO-IFC: For use with LDOS/TRSDOS Version 6 [i.e. Model 4].

CMD-FILE/PRO-CESS Version 2

Let me start this column with a patch. Incidentally, if you Model I/III users think that you missed CMDFILE1/FIX, don't think that The first patch was applied before CMDFILE was released.

```
CMDFILE2/FIX - 01/11/84 - Roy Soltoff - Applied 520025
. This fix corrects the "New transfer address"
. function of the WRITE menu command.
.
DOF,30=CD 64 EA 34 OF 6B CD D1 68
. WAS  06 04 7E CD D1 66 23 10 F9
D17,EC=3E 02 CD 01 68 3E 02 CD 0l 68 3A 0E 6B C3 01 68
. WAS  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
. End of patch

. PROCESS3/FIX - 01/11/84 - Roy Soltoff - Applied 520051
. This fix corrects the "New transfer address"
. function of the WRITE menu command.
.
D0D,DC=CD A3 45 3A 3E 46 CD D4 43
F0D,DC=06 04 7E CD D4 43 23 10 F9
D15,F4=3E 02 CD D4 43
F15,F4=00 00 00 00 00
D15,FD=3E 02 CD D4 43 3A 3D 46 C3 D4 43
F15,FD=00 00 00 00 00 00 00 00 00 00 00
. End of patch
```

Those of you with DSMBLR (or PRO-DUCE) can easily use CMDFILE in most cases to reorganize those CMD programs that have non-sequential load blocks. After processing the CMD program, the screening text data of the disassembler will not get confused.

CON80Z/PRO-CON80Z

As reported in the last issue of NOTES, Bernd Jung of Dusseldorf, West Germany reported a few problems with the CON80Z program that apparently slipped by our testing. Two of the five bugs can be fixed by patches; however, since three of the five cannot, I decided to correct my source and reassemble. Therefore, we are shipping CON80Z version 1.1 and PRO-CON80Z version 1.1. If you are currently using a 1.0 version copy of CON80Z and wish to obtain the corrected version 1.1, please return your master diskette in a protective mailer. There will not be a charge for the update to your disk.

CONVCPM/PRO-CURE

The current version of CONVCPM and PRO-CURE is version 2. This release expands the supported CP/M media types over version 1.x. We now support the transfer of files from the following CP/M single-sided diskette formats. No double-sided formats are supported. All formats are 5-inch mini-floppy, 35-or 40-track, unless specifically noted otherwise.

1) Cromemco Z-2 double-density
2) DEC VT-180 double-density
3) Heath/Zenith H89 single-density, soft-sectored
4) Heath/Zenith Z-100 double-density
5) Holmes VID80 double-density
6) IBM Personal Computer CP/M-86 double-density
7) Kaypro II double-density
8) LNW80 double-density

9) Lobo MAX-80 5-inch double-density
10) Lobo MAX-80 8-inch double-density
11) Memory Merchant Shuffle Board double-density
12) Montezuma Micro Model 4, double-density, 256-byte sectors
13) Morrow Micro Decision double-density
14) NEC PC-8001A double-density
15) Omikron Mapper I single-density
16) Osborne One single-density
17) Standard IBM-3740, 8-inch single-density, CP/M format
18) Xerox 820-1 single-density
19) Xerox 820-2 double-density
20) TRS-80 Model 4 CP/M+ (same as #6)

If you currently have the CONVCPM version 1.x and want to upgrade to the CONVCPM version 2.x, please return your CONVCPM master diskette with $20. A version 2 package will be returned to you. Note that there is no offer to upgrade from CONVCPM 1.x to PRO-CURE.

Sometime after the first release of Montezuma Micro's CP/M (our M4 code) for the Model 4, they changed the sector interleave. Since our CONVCPM 2.x supported the original M4 interleave, it doesn't work on the new interleave. The following patches update CONVCPM and PRO-CURE to the new M4 interleave. Note from your registration number whether the patch has already been applied.

```
. CONVCPM1/FIX - 02/24/84 - Applied 320155
. PATCH CONVCPM/CMD USING .......
. This fix adapts CONVCPM for the revision to
. Montezuma Micro's 2.2 CP/M third release
D13,F5=02 03 06 07 0A 0B 0E 0F 12 13 16 17 1A 1B 1E 1F 22 23 04 05 08
09 0C 0D 10 11 14 15 18 19 1C 1D 20 21 24 25
. WAS =02 03 0A 0B 12 13 1A 1B 22 23 06 07 0E 0F 16 17 1E 1F 04 05 0C
0D 14 15 1C 1D 24 25 08 09 10 11 18 19 20 21
. End of patch

. PROCURE1/FIX - 02/24/84 - Applied 320023
. PATCH PROCURE/CMD USING...
. This fix adapts PROCURE for the revision to
. Montezuma Micro's 2.2 CP/M third release
D12,CC=02 03 06 07 0A 0B 0E 0F 12 13 16 17 1A 1B 1E 1F 22 23 04 05 08
09 0C 0D 10 11 14 15 18 19 1C 1D 20 21 24 25
F12,CC=02 03 0A 0B 12 13 1A 1B 22 23 06 07 0E 0F 16 17 1E 1F 04 05 0C
0D 14 15 1C 1D 24 25 08 09 10 11 18 19 20 21
. End of patch
```

It has been discovered that the diskette format used on the Model 4 for the Tandy release of CP/M+ is identical to t he PC's CP/M 86 diskette. Thus, if you want to transfer files from a Model 4 CP/M+ diskette, set PRO-CURE (or CONVCPM) to the PC disk type.

The following are two patches for the CVTEXT program [there's two for Model I/III under LDOS 5.x and two for TRSDOS 6.x). CVTEXT51 and CVTEXT61 correct an error abort if the input file ends on a record boundary (i.e. if it is missing the trailing X'1A' EOF byte). This situation can occur in CP/M text files, since the X'1A' is necessary only if the file does not end on a record boundary. CVTEXT52 and CVTEXT62 are applied to detect if the input and output filespecs are the same, which can't be allowed for obvious reasons.

```
. CVTEXT51/FIX - Applied 05/09/84 effective 320162
. patch CVTEXT/CMD - LDOS 5.1.x version ONLY!
```

```
. correct EOF detection if input file ends on record
. boundary (i.e. file does not have trailing X'1A')
.
D00,8E=C3 99 54 00 00
D02,74=CD 13 00 CA 54 52 FE 1C CA 66 52 C3 7B 52
. eop

. CVTEXTS2/FIX - Applied 05/09/84 effective 320162
. patch CVTEXT/OMD - LDOS 5.1.x version ONLY!
. Detect attempt to use same file for both input and output
.
D00,51=CD A7 54
D02,82=CD 32 53 21 29 54 11 61 54 1A FE 20 38 06 BE C0 23 13 18 F5 7E
FE 20 D0 21 C8 54 CD 67 44 C3 30 40
D02,A3="Input and output can't be same file!"
D02,C7=0D
. eop

. CVTEXT61/FIX - Applied May 9, 1984 effective 320044
. patch CVTEXT/CMD - TRSDOS 6.x version ONLY!
. correct EOF detection if input file ends on record
. boundary (i.e. file does not have trailing X'1A')
.
D00,8E=C3 98 32 00 00
F00,8E=3E 03 EF 20 26
D02,73=3E 03 EF CA 54 30 FE 1C CA 65 30 C3 7A 30
F02,73=00 00 00 00 00 00 00 00 00 00 00 00 00 00
. eop

. CVTEXT62/FIX - Applied May 9, 1984 effective 320044
. patch CVTEXT/CMD - TRSDOS 6.x version ONLY!
. Detect attempt to use same file for both input and output
.
D00,51=CD A6 32
F00,51=CD 31 31
DO2,81=CD 31 31 21 28 32 11 60 32 1A FE 20 38 06 BE C0 23 13 18 F5 7E
FE 20 D0 21 C7 32 3E 0A EF 3E 16 EF
F02,81=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
D02,A2="Input and output can't be same file!"
F02,A2=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00
D02,C6=0D
F02,C6=00
. eop
```

Three additional patches are available that enhance CONVCPM and PRO-CURE by providing additional formats. These are: (1) CONVCPMP/FIX and PROCUREP/FIX to enable reading of Lobo CP/M+ 5-1/4" (35/40 track) and 8" 512-byte formats. These REPLACE the Lobo CP/M 2.2 256-byte formats, so if a user wants to be able to read both, he/she needs to keep both a patched and an un-patched version around; (2) CONVCPM8/FIX and PROCURE8/FIX to enable reading of Lobo 5-1/4" 77/80 track format. This REPLACES the Lobo 5-1/4" 35/40 track format, so if a user wants to be able to read both, he/she needs to keep both a patched and an un-patched version around. This patch can be applied BY ITSELF for the CP/M 2.2 format; for the CP/M+ format, it REQUIRES the application of patch (1) above.; (3) CONVCPMM/FIX and PROCUREM/FIX to add Pickles & Trout Model II CP/M 2.2 SSDD 8" format. The new parameter keywords are 'PTMOD2' or 'PT'. This last patch has been applied to the released version starting May 9th, 1984. The first two patches follow; however, the third is too long to expect you to key in. Thus, if you need the "M" patch, send for DISK NOTES.

```
. CONVCPMP/FIX - patch CONVCPM/CMD for Lobo CP/M+
. 512-byte formats - R. N. Deglin - 04/20/84
.
. 5-1/4" 35/40 track
D0E,B0=28 00 02 04 09 09 00
. 8" 77 track
D0E,D1=44 00 02 04 10 10 00
. eop

. CONVCPM8/FIX - patch OONVCPM/CMD for Lobo CP/M 2 and 3
. 5-1/4" 77 and 80 track formats - R. N. Deglin - 04/20/84
.
D0E,B7=04 10 00 08 80 00
D0E,C8=02 08
. eop

. PROCUREP/FIX - patch PROCURE/CMD for Lobo CP/M+
. 512-byte formats - R. N. Deglin - 04/20/84
.
. 5-1/4" 35/40 track
D0D,87=28 00 02 04 09 09 00
F0D,87=24 00 01 02 11 08 01
. 8" 77 track
D0D,A4=44 00 02 04 10 10 00
F0D,A4=3C 00 01 02 1D 09 02
. eop

. PROCURE8/FIX - patch PROCURE/CMD for Lobo CP/M 2 and 3
. 5-1/4" 77 and 80 track formats   R. N. Deglin - 04/20/84
.
D0D,8E=04 10 00 08 80 00
F0D,8E=03 08 00 04 40 00
D0D,9B=02 08
F0D,9B=01 10
. eop
```

DSMBLR/PRO-DUCE

This issue has one patch for the disassembler III. It eliminates some obscure code that hung around from version II. It also corrects a bug in not properly detecting the source and destination diskettes being the same.

```
. DSMBLR36/FIX - 04/06/84 - Patch to Model I/III DSMBLR
D02,09=00 00 00 00
. Was FE 4D 28 0D @ 55C1H
D0C,0A=00 00 00 00
. Was FE 4D 28 05 @ 5F9AH
D11,48=9A
. Was  99 @ 64C4H
. End of patch

. PRODUCE5/FIX - 04/06/84 - Patch to Model 4 DSMBLR
D01,BD=00 00 00 00
F01,BD=FE 4D 28 09
D0A,B5=00 00 00 00
F0A,B5=FE 4D 28 05
D0F,B5=0B
F0F,B5=0A
. End of patch
```

EDAS VERSION 4.1/PRO-CREATE

The following patches are for Model I/III EDAS and XREF. Don't forget that the /FIX files are included on the DISK-NOTES.

```
. EDAS418/FIX - 01/31/84 - Applied 821048 - EDAS Version 4.1a.
. This fix corrects EDAS so that it prints the same number of
. lines on each page (the first page had one extra line).
D04,8F=FF; Change reset to -1 from 0
.; WAS 00
D25,97=3D C2 BF; Alter PAGE pseudo-OP to 0 reset
.; WAS B7 C2 BD
. End of patch

. EDAS419/FJX - 02/13/84 - Applied 821066
. This corrects EDAS418/FIX for Model III users.
D00,87=10; WAS 11 @ X'5258'
D00,8A=C0; WAS BE @ X'525B'
D00,8F=BD; WAS BE @ X'5260'
D04,48=00; WAS 34 @ X'5877'
D04,88=21 CF 56 34 7E FE 38 C0 3E 00 77
. WAS  3A CF 56 FE 38 C0 3E 0O 32 CF 56 @ X'5BB7'
D0D,FB=D0 56 AF 77 2B 3E 00 77
. WAS  CF 56 AF 77 23 77 3E 00 @ X'650A'
D12,87=21 CF 56 C3 BF 58; WAS ZEROES @ X'6986'
D25,97=B7 C2 86 69; WAS PREVIOUSLY PATCHED TO 3D CA BF 5B
. END OF PATCH

. EDAS420/FIX - 04/06/84 - Applied 821116
. Fixes the "1" command after it was "broken" by EDAS419/FIX
PATCH EDAS (D08,69=BD:D08,76=BD)
. The byte values were 88

. XREF5A/FIX - 01/31/84 - Applied 821048 - XREF/CMD Version 4.1a.
. This fix corrects XREF so that it doesn't generate 256 spaces
. on a line containing a label that is 15-characters long.
. A second bug where an I/O error was erroneously displayed is
. also corrected with this fix.
D02,3F=CD 4D 59 00 00
.; WAS CD 99 58 10 FB
D06,DO=04 05 C8 CD 99 58 10 FB C9
.; WAS 00 00 00 00 00 00 00 00 00

. XREF5B/FIX - 03/30/84 - Applied 821113
. Patch corrects bug of some symbols occasionally missing references
PATCH XREF (D01,64=E6 0F 4F)
. Byte string was 4F 06 00
```

The following patches are for the Model 4 release of PRO-CREATE

```
. PCREAT04/FIX - 02/13/84 - Applied 820153 [PRO-LC 920091]
. PATCH to PRO-CREATE Version 4.2a.
. This fix corrects EDAS so that it prints the same number
. of lines on each page (the first page had one extra line).
. PATCH EDAS USING ...
D04,34=00
F04,34=34
D04,89=21 CF 34 34 7E FE 38 C0 3E 00 77
F04,89=3A CF 34 FE 38 C0 3E 00 32 CF 34
D0E,48=D0 34 AF 77 2B 3E 00 77
F0E,48=CF 34 AF 77 23 77 3E 00
```

```
D02,52=21 CF 34 C3 52 3A
F02,52=00 00 00 00 00 00
D26,24=1B 38
F26,24=50 3A
. End of patch

. PCREAT05/FIX - 02/04/84   Applied 820153 [PRO-LC 920091]
. PATCH TO PRO-CREATE Version 4.2a release of EDAS
. Corrects strange behavior when symbols are multiply defined.
. This error was introduced in V4.2a when something else was fixed
D1D,13=79 32 A9 52 7E F6 10 77 F2 A5 52 CB 76
F1D,13=7E F6 10 77 F2 A7 52 CB 76 79 32 98 52
D1D,27=0C
F1D,27=0E
D1D,30=12
F1D,30=14
D1D,32=20 0F 0A F6 40 02 E1 E1 11 82 4C C3 78 48 F6 80 77 0E 00
F1D,32=0E 00 20 0F 0A F6 40 02 E1 E1 11 82 4C C3 78 48 F6 80 77
. End of patch

. PCREAT06/FIX – 02/04/84 - Applied 820153 [PRO-LC 920091]
. PATCH TO PRO-CREATE Version 4.2a release of EDAS
. Minor enhancement of PCREAT03/FIX.
D01,C5=CD 13 38
F01,C5=3E 3C EF
D02,4A=1A E6 40 C8 3E 3C EF C9
F02,4A=00 00 00 00 00 00 00 00
. End of patch

. PCREAT07/FIX - Applied 820153 [PRO-LC 920091]
. Patch inhibits the B command from resetting bit-6 of X'EF65'
D0D,57=00 00 00 00 00
F0D,57=2A 97 3A CB 56
. End of patch

. XREF6A/FIX - 01/31/84 - Applied 820153 [PRO-LC 920091]
. PATCH XREF/CMD Version 4.1a.
. This fix corrects XREF so that it doesn't generate 256 spaces
. on a line containing a label that is 15-characters long.
. A second bug where an I/0 error was erroneously displayed is
. also corrected with this fix.
D03,EC=CD 73 37
F03,EC=3E 1A EF
D06,BA=4F 3E 1A EF C9
F06,BA=00 00 00 00 00
D02,02=CD 6A 37 00 00
F02,02=CD A5 36 10 FB
D06,B1=04 05 C8 CD A5 36 10 FB C9
F06,B1=00 00 00 00 00 00 00 00 00

. XREF6B/FIX - 03/30/84 - Applied PRO-CREATE 820168, PRO-LC 920112
. Patch corrects bug of some symbols occasionally missing references
PATCH XREF (D01,27=E6 0F 4F:F01,27=4F 06 00)
```

Now that the patches are out of the way, let's turn to some interesting material. Carltan Ulbrich had a problem with character case since he was using "THE PATCH" on his Model I. I suggested that Carlton apply the patch to inhibit Extended Cursor Mode (ECM). This patch was listed in NOTES Issue I. Canton reports that...

"I have followed your suggestion and it seems to work perfectly. That is, my Model I with THE PATCH now accepts upper and lower case with EDAS 4.1. In addition, it switches between Lower Case Converted and Lower Case accepted modes. This information may be of some use to you in the future in case you ever encounter another individual who persists in programming on the ancient and antique TRS-80 Model I. Thanks again for your help."

Robert J. Newton, a customer of long-standing, always seems to discover the "weird" problem. Bob presented me with the fact that EDAS prints one more line on the first page than succeeding pages. In addition he states, "I seem to receive an extra line feed after the last software form feed after every multi-page -LP assembly, so that the paper creeps up." I investigated both of Bob's problems and came up with some surprising findings. Excerpts from my response to Bob follows.

"It takes a person who has been with me for a long time to casually mention something that may not have been mentioned before but is nevertheless quite important. I am talking about the point raised in your latter of December 27th concerning EDAS printing one more line on the first page than it does on subsequent pages. Have you ever mentioned that before? In any event, I decided to check that out. Sure enough, you are correct. In fact, I even went so far as to come up with a patch (almost a one-byter) to correct the problem." [the patch turned out to be three patches: EDAS418, EDAS419, and EDAS420. These are listed above.]

"Now I cannot see where this situation would cause an extra line after every multi-page printout. There may be a clue, however. I had always thought that the extra line problem was rectified. After checking into the matter, I find that it was indeed corrected with EDAS48/FIX dated 12/09/82. EDAS48/FIX was published in NOTES FROM MISOSYS, Issue I; however, as it turns out, the FIX published in NOTES was EDAS48/FIX dated 10/27/82 - WHICH WAS A DIFFERENT FIX!!! Apparently I goofed and never got the 12/09/82 fix published. This was most likely due to the confusion prevailing with two fixes having the same number - a situation which never should have happened and was the root cause of the error. Here is the one-byte patch that was the 12/09/82 EDAS48/FIX:

```
D19,8A=BA; was B7"
```

This fix was applied starting with number 820318. If your registration number precedes 820318 for Model I/III EDAS 4.1, please apply this fix.

The following is an interesting letter from Jeffrey McLean concerning EDAS used with the LNW-80 Model II.

"…I discovered that EDAS would not work properly with it. When ever I would press the SHIFT @ key to pause a listing, the program would abort to the LDOS Ready prompt with the message, 'SYS ERROR'. I traced this problem to the fact that the LNW ROM has NOP's in the place of the @CTL vector. Since the LNW-80 model II has the option of RAM in the address space from zero to X'3700', I wrote a program to move the ROM to RAM and patch it up. I am enclosing a copy of that program; you may want to pass it along if any one contacts you with a similar problem."

When I looked at the program, I commented to Jeffrey that I thought the program should have a DI before switching out the ROM and an EI after restoring everything to normal with the patched low-memory RAM. Jeff said, "There are no DI or EI instructions in the program because they are not needed. When I select RAM, I output a 6 decimal to the memory management port, 31 decimal. This sets bits one and two. According to LNW documentation, setting bit one of this port will enable RAM from zero to X'3700'. Setting bit two will turn off the maskable interrupts. I've tested this; it works.

Later on, I output a 130 decimal, which sets bits one and seven and resets bit two. Bit seven write protects the RAM from zero to X'2FFF'.

Jeff also added code to put a jump to the DEBUG entry point in the NMI vector at 66H. He says it is extremely useful when debugging assembly language programs. This could be deleted from the code if you do not want it.

```
; LNW80II/ASM
; This program was written by Jeffrey McLean.
; Revised on February 3, 1984
; This program will move the ROM of the LNW 80 model two
; to RAM and then patch it so that EDAS (and probably
; many other programs) will work on it.
@EXIT   EQU     402DH
@ABORT  EQU     4030H
@DSPLY  EQU     4467H
HIGH$   EQU     4049H
DEBUG   EQU     400FH
        ORG     5200H
START   XOR     A
        OUT     (31),A      ;Make sure ROM is switched in.
        OUT     (254),A     ;Turn off reverse video
        LD      A,(HIGH$+1)
        CP      90H
        JR      NC,CONT1
        LD      HL,MESSAGE
        CALL    @DSPLY
        JP      @ABORT
CONT1   LD      SP,(HIGH$)
        LD      HL,0
        LD      DE,EOP
        LD      BC,30O0H
        PUSH    HL
        PUSH    DE
        PUSH    BC
        LDIR                ;Copy ROM to user memory
        POP     BC
        POP     HL
        POP     DE
        LD      A,6
        OUT     (31),A      ;Select RAM from 0 to 12K
        LDIR                ;Move ROM into RAM at 0 to 12K
; Patch the ROM so EDAS will work correctly.
; Fix the @CTL vector
        LD      HL,FIX
        LD      DE,23H
        LD      BC,5
        LDIR
; Fix the initialization for level two basic
        LD      DE,87H
        LD      B,2
LOOP1   LD      A,(HL)
        LD      (DE),A
        INC     HL
        INC     DE
        DJNZ    LOOP1
        LD      DE,8FH
        LD      A,(HL)
        LD      (DE),A
        INC     HL
        LD      DE,66H
```

```
         LD       BC,LENGTH
         LDIR
         LD       A,130
         OUT      (31),A
         JP       @EXIT
MESSAGE  DB       'Insufficient memory to fix ROM!',13
FIX      OB       OC5H,6,4,18H,1EH,2DH,1,OC3H
LEN1     EI
         JP       DEBUG
LENGTH   EQU      $-LEN1
EOP      EQU      $
         END      START
```

Jeffrey had a few other problems that he brought to my attention. First, he could not get the MEM parameter to work according to the documentation on page 2-1 of the manual. Second, on rare occasions, when entering the P command or the A command, the computer would 'freeze up'. Pressing the ENTER key or any other key would cause the computer to continue with the requested operation.

I responded... "You are correct in that the EDAS manual does indeed state a use for the "protected" memory being for in-memory assemblies. Unfortunately, that has not been valid for a long time and the two related statements have not been removed from the manual. I will get that corrected on the next printing. [The MEM parameter was originally implemented to provide a means of protecting high memory from use by EDAS when it was not protected via HIGH$. This function matches the M= parameter used in BASIC. For some unknown reason, the EDAS manual contains some erroneous statements concerning the use of the MEM parameter to set up space for an in-memory assembly. I don't think that any version of EDAS ever performed in this manner; therefore, the offending remarks will be removed from the manual. Jeffrey raised another point that EDAS does not guard against your entering a MEM value that exceeds the current value of HIGH$. Since patch space is at a premium, I won't implement a patch to inhibit that but will wait until the next re-assembly of EDAS.]

There could be a rare occasion when a PAUSE (SHIFT-@) would be entered prior to a command. EDAS uses the KFLAG scanner in LDOS for its BREAK-PAUSE detection. When EDAS cycles for its command entry, it calls a routine to reset the BREAK and PAUSE bits of KFLAG$. This routine performs a sequence of RESET, pause for about 1/2 second, then tests to ensure the bits are still reset. If they are set again, then it repeats the sequence. If by chance you happen to tap the PAUSE after the test, the PAUSE condition will get locked in. This would be rare since the window of time where the pause would be registered is extremely small.

As a last note, I find it surprising that you would be surprised at receiving a response to a "bug" report [note: Jeffrey was surprised that I took the time to respond to his bug reports and he commented thusly to me]. If I had ever written to other software firms concerning any problem whatsoever and not received a response, I would immediately cease doing any business with that firm. As far as I am concerned, that is part of the service that must be provided to a customer. Maybe that's what separates the good guys from the bad.

The next item came from Knute Johnson. "... I thought I would send along my solution to the EDAS paging problem [Knute uses PRO-CREATE, the Model 4 version of EDAS]. My printer is a Radio Shack Line Printer VIII. It does not have built in form feed control. When a top of form code is sent to the printer, it prints a splotch [Note, a splotch is not to be confused with a splat. A splat is a one-syllable way of saying "asterisk" (you know, the character "*") I can't print a splotch on the

Daisywheel II that is used to print NOTES; however, I suspect that a splotch may be the same character printed for an invalid code on my DMP-500 (see page 30 of the DMP manual). Thus, a splotch sorta looks like a Roman numeral 10 or two equilateral triangles connected on one of the vertices.] The easiest way to handle that problem is to use the forms filter's soft form feed [PR/FLT for Model I/III]. The lines per page must be set to some number higher than EDAS' page eject count, I use LINES=66. Now a new problem shows up. Every time a new line is sent to the printer via the forms filter the line count is incremented. So when you print something from EDAS that doesn't send a top of form character when it's done (e.g. 'T' command or an aborted print) the forms filter thinks part of a page has been printed. Now if you send an assembly listing to the printer, the forms filter will form feed when the lines per page count is reached and EDAS will form feed some lines later, none of them in the right place. This totally messes up your listing. So to solve this problem I wrote the attached program to send a reset top of form control code to the forms filter. It is loaded into the user patch space so it can be executed with EDAS' 'Z' command. The program has a loader which finds the user patch space and stores the program there. It is ORGed at 8000H so you can use the 'Q' command from EDAS (I wouldn't have any text in the buffer when you do this). [Model I/III users would have to apply a patch to the CMD file as the 'Q' command is not as powerful as the Model 4 'Q' command.] If you think this program might be of any use to anyone else, feel free to give them a copy."

```
;       TOFZ
;       Program to send top-of-form code to
;       forms filter with EDAS 'Z' command.
@PRT    EQU     6
TOF     EQU     6
        ORG     8000H
START   LD      HL,ZCMD
        LD      DE,(3609H)
        LD      BC,LENGTH
        LDIR
        RET
ZCMD    LD      A,@PRT
        LD      C,TOF
        RST     40
        RET
LENGTH  EQU     $-ZCMD
        END     START
```

Joe Williams, in Brussels with the military, supplied me with a great idea for using EDAS - something other than as an assembler. It turns out that Joe is a big KSM fan; he needed a reasonably efficient way to input and maintain KSM files. This was especially true since Joe developed a patch to Model III KSM/FLT to provide key stroke multiplication for A-Z and shifted A-Z. Joe submitted the patch and accompanying article covering a major use for enhancing SCRIPSIT to LSI for publication in the LSI Journal. I therefore won't go into the entire procedure; but I will cover a little about Joe's use of EDAS. He writes...

"The method I propose is as follows: first, using EDAS, create a /CIM file, e.g. a file without /CMD file load modules (with the /KSM file extension, if you like) using the -CI switch which allows an easy mixture of standard ASCII control codes, alpha-numerical codes, and extended 'ASCII codes' (those above code 128). Also using EDAS, the [source to the] /KSM file can be easily edited. (It is much easier to understand this if you take a look at the ESCRIPT/ASM file... [An excerpt of this file is shown later.]

Such a file allows the user to create one stroke commands, assigned to any alphabetical key, which will delete a word, scroll the entire screen up (or down), insert a space, ... (There are certainly other ways to create such a file. The LED Text Editor might serve, but since I don't have it, I can't be sure. The LDOS BUILD command and SCRIPSIT itself are not so easy to use because, without going into detail, they don't handle extended-ASCII codes or codes mixed with text very well. I would argue that EDAS is hard to beat."

For the entire text of Joe's article, look to a future edition of the LSI Journal. If they don't get around to printing it by the next issue of NOTES, I will ask Joe for a disk copy of his article so I can print it in NOTES. An abbreviated copy of his KSM source file follows. Note that for a full standard KSM, there would be an entry for each of the alphabetic characters A through Z.

```
;IMPORTANT:      Assemble using the -CI switch:
;A FILENAME/KSM -CI
;
;CLR = CLEAR KEY
;CTL = <SHIFT><DOWN ARROW>
INSERT           EQU         0B1H ;CLR <1>
LINE             EQU         0B2H ;CLR <2>
DELETE           EQU         0B3H ;CLR <3>
TAB              EQU         09H  ;CTL <I>
TO_END_OF_TEXT   EQU         1AH  ;CTL <Z>
SPACE            EQU         20H  ;<SPACE>
CR               EQU         38H  ;<;>
SHIFT_ENTER      EQU         1DH  ;<SHIFT><ENTER>
STOP             EQU         0DH  ;ENDS EACH KEY ENTRY
;
;-------
;<CLEAR> KSM file
;-------
AA       DB       INSERT,STOP      ;INSERT ONE CHARACTER
BB       DB       BLOCK,STOP       ;BLOCK MODE
CC       DB       DELETE,TO_END_OF_TEXT,STOP
DD       D8       DELETE,STOP      ;DELETE ONE CHARACTER
EE       DB       PARA,STOP        ;PARAGRAPH MARKER
;
Sheets:  Letter Size and long drafts -- Epson Printer:
II       DB        '>* Sheets: Standard 8 1/2 x 11 '
         DB        '>J=N TM=2 BM=51 PL=55 LM=10 '
         DB        'RM=72 LS=1 PF=2',CR
         DB        '>*VC=Y',CR
         DB        '>$27,71',CR
         OB        '>* Use for a long draft:',CR
         DB        '>*BM=57 PL=61 LS=2',CR,STOP
;
LL       DB       SHIFT_ENTER,'T=5 15 25 35 45 55',CR,STOP
;
```

GRASP

Don't forget that the first diskette of fonts is still available. Nine fonts are included - CLASSIC, BIGCHAR, COMPUTER, Anglo-Saxon, Anglo-Saxon BOLD, Irish, Irish BOLD, Coptic, and Egyptological. If you are a registered GRASP user and want to obtain this disk of fonts, please send $10 and request GRASPF1. The $10 charge covers the cost of the disk, mailer, processing, and shipping. MISOSYS will continue to seek character fonts to collect for building GRASPF2. I am indebted to R. W. Odlin and Karl A. Hessinger for their generosity in making these fonts available.

HELP/PRO-HELP

A query from Fred Keebler concerning a problem with PRO-HELP disclosed a bug on investigation of his problem. After analyzing the disk files that Fred sent, I discovered that there is a bug in the HELPTXT/BAS program. The last part of LINE 450 should be changed from "&H28" to read "&H26". The &H26 represented the high order byte of the transfer address which should have been X'2600', not X'2800'.

You may well puzzle as to how I developed the HELP and HELPB files in spite of that bug. Well, I used the MAP method. In this method, the HELPADD program always forces the value to X'2600' regardless of the output of HELPTXT/BAS. Thus, by using MAP, it masked the bug in HELPTXT/BAS.

LC/PRO-LC

The big news here is the PRO-LC Version 1.2 update. On March 31st, a letter was sent to PRO-LC Registered Users. If you have PRO-LC and did not get the letter, then you must have registered after the letter went out. The contents of that letter are repeated here.

MISOSYS has completed the implementation of the floating point library for the PRO-LC users. This library includes all of the float (single precision) and double precision functions identified in your PRO-LC user manual. In addition, we have also added the fseed() function to provide for user initialization of the frnd() random number seed.

We have also significantly enhanced the LC standard library through the implementation of the input scan functions and standard UNIX low-level file I/O routines as well as many other functions [isalnum(), isspace(), ispunct(), isprint(), iscntrl(), isascii(), btoi(), itob(), otoi(), itoo(), checkc(), chkx(), getx(), putx(), cleareof(), clearerr(), close(), creat(), fdown(), fpup(), ferror(), fflush(), open(), read(), seek(), scan_(), scanf(), fscanf(), sscanf(), sprintf(), ungetc(), ungetch(), unlink(), and write()].

Because of the extensive library improvement, we want to assure ourselves that the correct PRO-LC owner obtains this update. Therefore, we are asking that you return you PRO-LC master diskette to: MISOSYS, Inc., Attn: PRO-LC update, PO Box 4848, Alexandria, VA 22303.

The update is free of charge to the PRO-LC owners since we have delayed in forwarding the floating point library. Thank you for your consideration.

Now then, since the LC/LIB was extensively revised, an LC 1.2 version is also now available. It adds all of the above functions except the chkx(), getx(), and putx() functions which are pertinent to banked RAM only [getx() and putx() get and put a character to a RAM bank while chkx() checks for the availability of a bank]. The charge for this update is based on the original purchase date of your LC by the original purchaser and is as follows:

Anytime after May 15, 1984	No charge
From April 1, 1984 through May 14, 1984	$5.00
Anytime prior to April 1, 1984	$15.00

To obtain the update, send your LC master disk enclosed in a protective mailer to the address identified above (use Attn: LC update). Anyone claiming a charge of less than $15 must enclose a copy of their invoice if the LC was purchased from other than MISOSYS. For those customers requesting the return shipment to outside of North America, please remit an additional $2.00.

Let me expand a little on the functions that are included and changes to existing functions. First, the low-level UNIX-type file I/O functions; open(), close(), read(), write(), and seek() should operate exactly as described in K&R. Don't forget that seek() is not the same as lseek(). Since LC doesn't support longs, I implemented seek(). An additional feature of seek() is that it provides as a return code, the current position of the file: the block position for a block seek, and the block offset position for an offset seek. Creat() functions according to K&R except that it uses only the "all others" field of the pmode argument to establish the protection level under LDOS (or TRSDOS 6). The low-level file I/O functions reference a file via a FILE DESCRIPTOR (FD) not a file pointer. There is a similarity between an FD to the data set reference numbers found in FORTRAN. FD's are numbers such as 0, 1, 2, ... There is a relationship between FD's and FP's. The two functions, fpup() and fdown() convert a file descriptor to a file pointer and vice versa respectively. The fflush() function can be used to write all buffered output to disk as well as update the directory's EOF while ferror() is used

to obtain the last error code encountered on a file stream. Unlink() operates per K&R to delete a file (i.e. KILL or REMOVE). It will not permit you to unlink a file that is currently open under LC.

The added iswhat() functions round out the character type tests as suggested in Plum's book, LEARNING TO PROGRAM IN C. In fact, I tested the functions with his program listed on page 3-15 of the text.

The checkc() function should be very useful for anyone developing communications programs in LC. This function will scan a file stream for the availability of a character. In order to make the function portable across all files, it actually operates by issuing a DOS @GET. If a character is fetched, it is stuffed back into the file stream via the ungetc() function. Speaking of ungetc(), it supports a one-character buffer per file stream to allow you to push one character back into a file stream. Ungetch() is the same as ungetc(stdin).

Btoi() and its converse, itob() perform conversions between binary numbers in ASCII and ints. Otoi() and itoo() are similar but operate on octals. They round out the decimal and hexadecimal conversion functions existing in LC 1.1.

The various scan functions support d, o, x, b, h, c, s, and f or e conversions. Scan operates per K&R. Notice that it supports float (and double) input conversions. The print functions have also been updated to directly convert floating point values. However for now, the output format is fixed at standard scientific notation in the format, sd.ddddddEsee. The ftoa() and dtoa() routines convert to this fixed format as well. This standardizes the output format of floating point numbers across all releases of LC. At some future time, we will provide a C function to format floats. Note that the sprintf() function is also included in release 1.2.

Turning to our customer letter file, Larry Lewis of Australia writes, "In the documentation for LC you note the method for adding a user library. However, you omit to mention that the user must amend LCMACS/ASM to have the default invocation of:

```
@_USERLIB     EQU     FALSE
```

This is required for programs that DO NOT use the user library. I hope that this may be of assistance to other users of LC."

Larry is correct in his statement so long as you have added the following code to LC/ASM in order to search the library:

```
            IF          @_USERLIB
*SEARCH     USER/LIB
            ENDIF
```

On the other hand, if you change the conditional from "IF" to "IFDEF", then you should not need to add the initialization to LCMACS. The IFDEF conditional is evaluated TRUE if the label in the operand field has been defined regardless of the value of the definition. Now if you use IFDEF, DO NOT insert any initialization to LCMACS or the SEARCH will always occur. I can't say why we chose to not use the IFDEF for the FPLIB and INLIB searches when putting together the first release of LC.

Mike Lewis of Cheshire, England, has sent in the following program which can be used to SYSRES multiple system modules from a single command line. It essentially generates a JCL file which it then invokes with a cmdi() function. I know that some of you may have tried to SYSRES

system modules from an LC program via the cmd() function; however that cannot be done. I'll tell you why later. First, Mike's RES/CCC.

```
/* << RES - Ldos Sysres Utility >> */
/* 'res n1 n2 n3' etc creates a file called
   resfile/jcl with SYSRES statements for
   overlays n1 n2 etc. This is then DO'ed! */
#asm
   COM '<LDOS Multiple SYSRES Utility by Mike Lewis - Sep 1983>'
#endasm
#include stdio/csh
#option INLIB
#option FPLIB OFF
#option FIXBUFS ON
#option REDIRECT OFF
#option MAXFILES 1
#define T1 '**** LDOS Multiple SYSRES Utility ****\n"
#define T2 "Copyright (C) Mike Lewis, Sep 1983\n"
#define T3 "Written in LC - a 'C' implementation from MISOSYS\n"
#define T4 "Assembled with EDAS IV from MISOSYS\n"
#define T5 "LC Library, Copyright 1982 James J. Frimmel\n\n"

main (argc,argv)
int argc;
char *argv[];
{ int c, fp;
   char buf[100] ;
   puts ("\x1c\x1f") ; puts (T1);
   /* title */
   puts (T2);
   puts (T3);
   puts (T4);
   puts (T5);
   if (argc < 2 )
   {
      puts ("\nFormat is: RES n1 n2 n3 ...\n");
      exit (0);
   }
   fp = fopen ("resfile/jcl","w")
   while (argc>1)
   {
      buf[0] = 0x0;
      ++argv;
      strcat (buf, "SYSTEM (SYSRES=");
      strcat (buf,*argv);
      strcat (buf,")\n")
      puts ("Submitting .. ");
      puts (buf);
      fputs (buf,fp);
      --argc ;
   }
         puts ("\n");
   fputs (". press <ENTER> ......",fp);
   fputs ("\n//alert (1,0,7,0)\n",fp);
   fclose (fp);
   cmdi ("do resfile/jcl\n");
}
```

Now then, why can't you SYSRES a system module from an LC application via the cmd() function. The cmd() function operates by moving the executing program to high memory by reducing the unused memory to zero (see LC PROGRAM MEMORY MAP page 5-6 of the LC manual). It then sets HIGH$ to the new next lowest address and sets up an interface to @EXIT and @ABORT. Cmd() then issues the requested command. The interface to @EXIT and @ABORT results in a return to the cmd() function at the completion of the command. If you invoked SYSRES and the DOS carried out the request, it would place the system module prior to the revised HIGH$ set by cmd(). After return to cmd(), HIGH$ is restored to the original value; thus the system module is no longer protected and may in fact be overwritten by the LC application as it is restored to its executing memory region.

David Lamkins, of Massachusetts, brought up a few points in his letter concerning LC and PaDS as well as ZSHELL. He says...

"I have a couple of very short items that you may wish to pass along to other users in the next issue of NOTES. First, when making an LC-compiled program part of a PDS, be sure to invoke the program without an intervening blank between the PDS name and the member name. This way, the argument parser will still work properly. If you use a blank before the member name, you'll find that the PDS name alone gets parsed out as a command line argument. Second, if you use ZSHELL to redirect I/O for LC-compiled programs, specify '#option KBECHO OFF' as well as '#option REDIRECT OFF'. If you don't, you'll get a copy of the standard input interspersed with the desired output. The ZSHELL documentation doesn't mention this."

LC is all over the Globe. Roger Merritt, a C'er from Thailand, had some interesting words concerning the Shell sort as demonstrated in my SEESHELL program that was in NOTES, Issue I. Roger writes...

"In the first issue of NOTES you gave a very nice little demo of the Shell sort, which you called 'seeshell'. I have been working on a disk cataloging program to send to the LC Interest Group and got diverted to a consideration of sorting. For my program I copied the quicksort routine from Kernighan & Plaugher's SOFTWARE TOOLS, but I had some bugs and had to spend more time studying the darned thing than I really wanted to. As it happened, when I finally found the bug it was ridiculously simple and stupid (aren't they all?), but in the process I had written a demo program to try to follow what Quicksort was doing. Actually, what I did was take your seeshell, deleted shell(), and copied my routine in its place.

As I expected, Quicksort was a lot faster than shellsort, and by about the proportion I expected based on Donald Knuth's analysis in Vol. 3. That is, your shellsort's time was proportional to N**1.5 while Ouicksort was proportional to N*logN (where 'log' means 'log to the base 2'). However, Knuth indicated shellsort could be considerably improved 'to about N**1.36) by proper selection of the gap.

By this time, I had gotten curious enough to give it a try, because Quicksort compiled to quite a lot more code. I modified shellsort to select the gap according to Knuth's recommendation. I was appalled at the result.

By my very rough timing, the original seeshell takes about 26-26 seconds to sort 1024 characters. The unmodified version of Quicksort takes about 12 seconds. The modified shellsort only takes about 14 seconds.

I suppose that you, a professional, won't be surprised at this result, though you may be interested by the magnitude of the difference [actually Roger, a professional can't possibly know everything about everything – I indeed found your findings quite useful and deserving a pat on the

back]. However, you might like to pass the modification on to the readers of NOTES (economical, too -- at a rough estimate Quicksort compiles to about 500 bytes more code). It also has the prestige of having been used as the example by Kernighan & Ritchie, and of course by you. The modification isn't very complex, and the improvement is truly remarkable, but I suspect that few of the people who purchase the LC compiler will encounter it.

Perhaps I should mention that I wouldn't have taken time to explore this area except for an article by George Blank in Softside magazine a few years ago in which he criticized the Shell-Metzner sort on this very ground. However, the program he gave was in BASIC, and the language makes it hard to follow the algorithm."

Roger went on to say that he was trying to get a disk containing the Quicksort and Shellsort versions to the LCIG. I follow this discussion on sorting with the updated copy of SEESHELL - actually a version that does an ascending sort followed by a descending sort. This version of Seeshell has been modified to function on a Model 4 using PRO-LC. The example also illustrates using a function for the ordering test to support both ascending and descending sorts.

```
/* seeshell/ccc by Roy Soltoff modified by R.A.Merritt
   according to recommendation in Art of Computer Pro-
   gramming, Vol 3, Sorting & Searching, by Donald R.
   Knuth.
*/
#include stdio/cah
#option args OFF
#option redirect OFF
#option maxfiles 0
#option fplib
#define CRT 0xf800  /*Use  0x3c00 for Model I/III */
#define SIZE 1920   /* Use 1024 for Model I/III */
main()
{
   char *v, v1[4], v2[4];
   int i,n,c, ascend(), decend();
   /* CRT address and screen size */
   v = CRT; n = SIZE;
   cls();
/* Use following ASM code for PRO-LC on Model 4 only */
#asm
   DI
   LD     A,(78H)
   AND    0FCH
   OR     2
   OUT    (84H),A
#endasm
   itof(94,v2);   /* set random upper */
   for (i=0; i<SIZE; ++i)
       {
       frnd(v1,v2);  /* get random number */
       *v = ftoi(v1) + 32; /* set char 32-126 */
       ++v;
       }
   shell(CRT,n,ascend);
   for (i=0; i<10000; i++) ;
   shell(CRT,n,decend);
   c=getchar();
   cls();
   exit(0);
}
```

```
shell(v,n,order) /* sort v[0].. .v[n-1] */
char v[];
int n, (*order)();
{
   int gap, i, j, temp;
   /* set gap as recommended in Knuth, Vol 3, p. 95 */
   for (gap = 1; gap < n; gap = (3 * gap) +1)
      ;  /* set gap larger then n, then back up two steps */
   gap = (--gap)/3;
   gap = (--gap)/3;
   /* now gap has been initialized */
   for ( ; gap > 0; gap = (--gap)/3 )
      for (i=gap; i<n; i++)
         for (j=i-gap; j>=0 && (*order)(v[j],v[j+gap]); j-=gap)
            {
            temp=v[j];
            v[j]=v[j+gap];
            v[j+gap]=temp;
            }
}

cls()
{  puts("\x1c\x1f");  }
ascend(v1,v2)
char v1,v2;
{  if (v1 > v2) return(1); else return(0); }
decend(v1, v2)
char v1,v2;
{  if (v2 > v1) return(1); else return(0); }
```

While we're on the subject of sorting, I share with you the following information donated by Joe Williams, our C'er from Brussels. Joe also supplied the informative article on using EDAS to input and maintain KSM files.

"I am enclosing the C program for a simple sort which has the advantage (not shared by either quicksort or the shell sort, for example) of being stable. 'Stable' here means that the listed items are sorted in sequence; that is, if you sort an alphabetized list by a key other than the one used to alphabetize it, the newly sorted list will still remain in alphabetical order within the groups formed by sorting with the new key. (This sort is not necessarily the most efficient stable sort available, but it is certainly one of the simplest. The shellsort is based on it and improves its speed, but shellsort is not stable.)

As implemented, the sort will also sort on a field, as well as the whole line or the first x characters of a line, and does not distinguish between upper or lower case. The sort is embedded in an input and output routine that comes almost straight out of chapter 5 of K & R. Using the sort with this routine (which, by the way, could be sped up by pre-allocating string space), you can sort SCRIPSIT files by line or by parts of a line (using redirection, of course). This allows you to use SCRIPSIT as a simple, but flexible in-memory data base manager. The program will serve to sort any list (or words, or names, or addresses, etc.), and is another good example of how easy it is to use C to write useful, multipurpose utilities."

```
/*  sortesc. .2/29/84. .ajw */
/*  K & R, page 1:6 - modified to handle fields and to */
/*  ignore case distinctions. Also, */
/*  uses a straight insertion sort (see Knuth) which is */
/*  a stable sort. See sort for an explanation of the */
/* function's arguments. */
```

```
/**/
#include stdio/csh
#option INLIB
#define LINES 500 /* max lines to be sorted */
#define FAIL 0xffff
/**/
main(argc,argv)
   unsigned argc; char *argv[];
   {
   char *lineptr[LINES]; /* pointers to text lines */
   int nlines; /* number of input lines to read */
      /* IMPORTANT: nlines is to BASE 1 */
   if ((nlines=readlines(lineptr,LINES)) >= 0)
      {
      sort1(lineptr, nlines, argc, argv);
      writelines(lineptr,nlines);
      }
   else
      printf("Input too big to sort!\n");
   }
/**/
#define MAXLEN 255 /* MAXLEN-1 = max line length */
/**/
readlines(lineptr, maxlines) /*read input lines and */
   char *lineptr[]; /* ret # of lines, */
   int maxlines; /* to BASE 1 (not 0) */
   {
   int len, nlines; /* len & nlines are BASE 1! */
   char p, *alloc(), line[MAXLEN];
/**/
   nlines = 0; /* nlines counts to BASE 1! */
   while (( len = getline(line,MAXLEN)) >0)
      if (nlines >= maxlines)
         return(-1);
      else if ((p = alloc(len)) == NULL)
         return(-1);
      else
         {
         line[len-1] = '\0'; /* converts \n to NULL */
         strcpy(p,line);
         lineptr[nlines++] = p;
         }
   return(nlines),
   }
/**/
getline(s, lim) /* get line into s, ret length */
   char s[]; /* lim-1 = max length line */
   int lim; /* length returned is BASE 1! */
   {
   int c, i,
   for (i=0, i<lim-1 && (c=getchar())!=EOF && c !='\n'; ++i)
      s{i] = c;
   if (c == '\n')
      { s[i] = c;  ++i; }
   if (s[i-1] != '\0' || i != 1) /* eliminate final */
      s[i] = '\0'; /* null byte in */
   else i = 0; /* a SCRIPSIT file */
   return(i);
}
/**/
writelines(lineptr, nlines) /* write output lines */
   char *lineptr[];
   int nlines;
```

```
   {
   while (--nlines >= 0)
      printf("%s\n", *lineptr++);
   }
/**/
sort1(v,nlines,argc,argv)
/* Straight insertion sort (stable) Knuth III, p. 81 */
unsigned argc;
char *v[], *argv[]; /* into increasing order */
int nlines;
   { int i, j; char *temp;
   for (i=1; i < nlines; i++)
      for (j = i-1; j >=0; j--)
         {
         if (fldcmp(v[j], v[j+1), argc, argv) <= 0)
            break;
         temp = v[j];
         v[j] = v[j+1];
         v[j+1] = temp;
         }
   }
/*  fldcmp. .2/25/84. .ajw */
/*  Compares fields in s1 and s2 using and returning
   same values as strcmp. Field placement is the pos
   and count of strmid. */
/* Case distinctions are ignored. */
/* This version takes none, one, or two arguments */
/* No arg:    whole line is sorted. */
/* One arg:   sort lines on the first X number of chars
              indicated by the argument */
/* Two args:  sort lines on the field indicated by the
              arguments. First argument = start of the
              field, base 0. Second argument = the length
              of the field. */
/**/
fldcmp(s1,s2,argc,argv)
unsigned argc; char *argv[], s1[], s2[];
/**/
   {
   static char fld1[MAXLEN], fld2[MAXLEN];
   int pos, count; /* pos and count in strmid */
   int errcode;
/**/
   if (argc == 1)
      { allupper(s1); allupper(s2); return(strcmp(s1,s2)); }
   else if (argc == 2)
      {
      pos = 0;
      count = atoi(argv[1]);
      strmid(fld1,s1,pos,count);
      strmid(fld2,s2,pos,count);
      allupper(fld1); allupper(fld2);
      return(strcmp(fld1,fld2));
      }
   else if (argc == 3)
      {
      pos = atoi(argv[1]);
      count = atoi(argv[2]);
      strmid(fld1,s1,pos,count);
      strmid(fld2,s2,pos,count);
      allupper(fld1); allupper(fld2);
      return(strcmp(fld1,fld2));
      }
```

```
   else abort("0, 1, or 2 arguments\n");
   }
/**/
abort(msg) char *msg;
   {
   fputs(msg,stderr);
   putc(EOL,stderr);
   exit(FAIL);
   }
allupper(s) char s[];
   {
   int p; char toupper();
   for (p=0; s[p] != NULL; p++)
      s[p] = toupper(s[p]);
   return;
   }
```

The next topic provides an answer to the problems sometimes experienced which stem from LC's use of the BREAK key to indicate EOF from the keyboard. A few people have raised the question as to why the keyboard seems to lock up for getchar() after the BREAK key has been pressed. For each open file stream (and don't forget that character devices such as the keyboard input and video output are files to LC), LC maintains a flag byte in the File Control Area (FCA). Within this byte, there is an FOF flag that is set once an EOF is detected from the file on a request for a character input (other conditions can cause this bit to be set such as an error detected on an output file). The getc() function (and getchar() invokes getc()) first tests the EOF flag. If EOF is set, then getc() does an immediate return with EOF as the returned value. Thus, once you depress BREAK in response to a getc() or getchar(), EOF will always be subsequently returned.

Under release 1.2, a function called 'cleareof()' has been provided which can be used to reset the EOF flag bit. Also, under version 1.2, another option called 'BREAK' has been added which is used by getc() to inhibit the BREAK key from being treated as an end-of-file from all input devices. For the moment, until you obtain LC release 1.2, let me pass along the cleareof() function. It may be used to reset the EOF condition once detected.

```
cleareof(fp)
{
#asm
*M
       CALL   @GETFV        ;Get the FCA pointer
       SCF                  ;Prepare for error
       JR     Z,$?1         ;Go if file closed
       BIT    5,(HL)        ;Cannot cleareof a file
       JR     NZ,$?1        ;  if the error bit set
       RES    6,(HL)        ;Reset the EOF flag
       XOR    A             ;Clear the CF
$?1    SBC    HL,HL         ;Set 0=no error, -1=error
#endasm
}
```

An 'RET' statement is not needed as the closing brace automatically generates an RET. The argument, fp, is the file pointer returned from an fopen() request or stdin in the case of getchar(). Using the cleareof() function, one could rewrite an input as follows:

```
if ((choice = getchar()) == EOF) cleareof(stdin);
```

You may remember the useful MBCONV program submitted and printed in the last issue of NOTES. This program converted a Model I/III compressed BASIC program for Model 4 use. Well John Harrell liked it enough to make it better. John was the one that wrote that in-depth review of LC which appeared in 80 MICROCOMPUTING. Rather than repeat MBCONV here, I will just include it in DISK-NOTES. A copy is also available from the LCIG.

Speaking of the LCIG, let me mention a reminder. Don't forget that there is an LC Interest Group chaired by Earl C. Terwilliger, Jr. Contact Earl for details at 647 North Hawkins Ave., Akron, OH 44313 (USA). I believe that there are presently 5 to 6 disks of public domain LC software.

With the release of LC 1.2 and its fixed scientific format for floating point numbers, it is necessary to revise the floating point example shown in the LC manual (page 4-44 of the First and Second editions). In the Fahrenheit to Celsius conversion table printing, you can no longer print 8 characters for the float result. This is so because the new float format is precisely 13 characters long. The easiest solution at present is to delete the line:

```
ftoa(celsius,celsius_str);
```

and change the subsequent line to read:

```
printf("%-10.3d %e\n",fahr,celsius);
```

If you want to keep the ftoa(), then you must change the printf() to specify a "%13s" in lieu of the "%-8.8s" and increase the size of celsius_str to 14 from 8 in the statement: "char celsius_str[8];".

On another front, if you have been writing LARGE programs in LC without using separately compiled modules, you probably have run up against a problem with the assembly. EDAS has a kludgy end-of-file test to permit it to work with source files that do not have a CTL-Z as the last byte (such as could occur when using SCRIPSIT or LED). Because the assembler's editor itself can not possibly create a source file that can exceed 64K, the test limits itself to testing only the two low-order bytes of the EOF triad. If, by chance, you have written an LC program that generates an ASM source file exceeding 64K, you will have trouble assembling it. There are two ways out of this dilemma. The first is to separate your one program into two or more modules. Then compile them separately and link them together via *GET's. You will have to study the LC manual on separately compiled modules.

The other way is to patch the assembler to remove the kludgy EOF test. The following patch to EDAS (Model I/III) will remove the test.

```
PATCH EDAS (D01,C3=18)
```

The value was X'38'. So to restore the test, patch the byte back to X'38'. The equivalent patch for PRO-CREATE (Model 4) is as follows:

```
PATCH EDAS (D01,4E=18:F01,4E=38)
```

A word of caution here. If you are going to access the Model I/III LC 1.1 FP/LIB, do not use the patch. In order to reduce the length of FP/LIB to save a granule of space, the members are stored without the CTL-Z. This was possible because of the EOF test performed by EDAS. If you patch EDAS to remove the test, you will have a problem in accessing the FP library. This restriction was removed when LC 1.2 was released.

C'erious Subjects - by Jim Frimmel

Hiya! We're going to deal with a limitation in LC. Only one-dimensional arrays are allowed by LC, but fortunately there are ways around this.

To get an idea of how a multi-dimensional array can be simulated in an LC program, think about how a two-dimensional array would be stored in memory. Normally, the first element, a[0][0], would be stored at the lowest location. Each subsequent element, a[0][1] through a[0][MAX], is stored in subsequent memory locations. If we assume that our compiler doesn't worry about word boundaries, we could use a pointer to get to elements in the array. Given the array declaration:

```
char m [ISIZE][JSIZE];
```

we can use a character pointer, p, to get to any particular element of the array. If "i" is the major subscript and "j" is the minor subscript, the expression:

```
any_element = p[ i * JSIZE + j];
```

will cause any_element to be assigned the value of the element, m[i][j]. This technique can be expanded to an arbitrary number of subscripts. However, there is no guarantee that other implementations of C will store the elements of a multi-dimensional array in contiguous memory, especially if the elements are structures.

Take a look at listing 1 which is a program that implements a simple two-dimensional array called 'a'. The basic method is to write a function which can access a one-dimensional array in a way that simulates a multi-dimensional array. Using this function looks rather like using a BASIC multi-dimensional array, since the name of the "array" is followed by subscripts in parentheses separated by comma's. Note, however, that the function returns the ADDRESS of the element specified, NOT the VALUE. This is so that assignment expressions can be used. Another way to do this would be to write two separate functions to access the element: one to "set" the value and the other to "get" the value.

```
LISTING 1
/*** example of a multi-dimensional array implemented
   with functions - simple case - Jim Frimmel 5/10/84
***/
int a_raw[1000]; /* the 'raw' vector */
int *a(i1,i2);
{ /* returns address of element of array a[100][10] */
   return  a_raw+i1*10+i2;
}

main()
{ /*   let's fill the array with the offset into the
       array for each element */
   int i,j,k;
   for (i=k=0; i<=99; ++i)
      for (j=0; j<=9; ++j)
         *a(i,j) = k++;

   /* now dump the values to the standard output */

   for (i=0; i<=99; ++i)
      {  printf("\n%4.4d: ",i);
         for (j=0; j<=9; ++j)
```

```
            printf("%4.4d ", *a(i,j));
        }
    }
```

In listing 1, the actual space for the array is declared in line 4. In lines 5-8, the function is defined that returns the address of an element, given two subscripts. Note that `"i1"` is multiplied by 10 to adjust for the innermost dimension of ten elements.

The body of the program consists of two similar loops. The first loop fills the array with numbers; the second loop prints out these numbers, showing that which was placed into each element is the same as what was retrieved. The indirection operator (asterisk) is used to access the array element in both loops since a() only returns the address of the element.

By writing a function with which to access a data structure, we are actually using a modern technique called "object-oriented design". This technique calls for "hiding" the details of a data structure by defining functions which are the exclusive method for accessing the data structure. In this way the actual method of storing the data is not important to the programmer using these access functions. Besides simplifying a program's design, object-oriented design allows the actual storage method used to be changed without affecting the programs that access the "object".

This concept of object-oriented design has been built right in to a computer language called Ada. The Department of Defense spent lots of our taxpayer dollars inventing Ada to the be the standard language used by DoD and its contractors. In the Ada language, a series of functions to access a certain data structure or machine facility is called a PACKAGE. An Ada package does not just consist of the access functions; an integral part of a package is an "interface" section. This tells both user and system how to access the functions in the package. In this way Ada can be used to write "self-documenting" programs.

Let's look now at what a "package" could look like In C. For our example let's assume that you are a teacher and you want to use your system to compute average grades of your students. To do your calculations, you want to load all student grades into a two-dimensional array called "grade". Since you are using LC, you can't have a two-dimensional array directly, so you choose to write functions that will simulate the two-dimensional array. You mumble something under your breath, but start coding anyway. However, as you continued writing, you started to like the way things were looking. Listing 2 shows what you came up with.

```
LISTING 2
/**
       GRADES DATABASE PACKAGE - by Elsie U. Zerr
**/
/*** variables used in this package - all static ***/

static int *gr_base; /* points to base of array */
static unsigned n_students, n_tests;
/**
       gr_init() - initialize the grades database
          This function tells the package how to dimension
          the array and how much storage to reserve, based
          on the number of students and tests.

       Returns:  0 if no errors, -1 if not enough space.
**/
gr_init(num_students, num_tests)
{   n_students = num_students;
```

```
    n_tests = num_tests;
    gr_base = alloc(n_students * n_tests * 2);
    if (gr_base == NULL)
        return -1; /* return -1 if error */
    else return 0; /* no error */
}
/**
    get_grade() - return the grade for a given student
                  and test.
**/
get_grade(student,test)
    unsigned student, test;
{   /* first, check the range of subscripts */
    if (student >= n_students || test >= n_tests)
        return -1; /* bad subscripts */
    return gr_base[ (student * n_tests) + test ];
}
/**
    set_grade() - set the grade for a given student test.
**/
set_grade(student, test, value)
    unsigned student, test;
(   /* first, check the range of subscripts */
    if (student >= n_students || test >= n_tests)
        return -1; /* bed subscripts */
    gr_base[ (student * n_tests) + test ] = value;
    return 0; /* no errors */
}
```

Since you didn't have to explicitly declare the array, the program did not have to be limited to a particular size of a dimension. The function gr_init() provides the ability to define the dimensions each time the program is run. Also, the memory space for the array could be dynamically allocated using alloc() so that all of available memory could be used instead of limiting the size of the array at compile time. Since you hadn't read this article yet, you wrote separate functions for storing and retrieving an element. This turned out later to be an advantageous approach, as we'll see.

Being an ambitious and capable teacher, soon you have become the principal of a large high school. Thus, you were working on converting the original grading program to be able to handle a larger number of students. The original version could only handle 5,000 student-tests since available memory above the program size was about 10K. This was adequate for 500 students with 10 tests to average. There were now 1,500 students to grade, more than was possible in memory. Just when you were ready to go out and blow your meager budget on a model 2000 with more RAM, you had a brainstorm, "Why not store the working array on disk?", you posed. Looking over your program, you found that the conversion was made very simple due to having access to the student grades provided by functions in a package. Listing 3 shows the result. Notice that the random I/O functions available in the LC 1.2 upgrade are used to seek() to positions in a file.

```
/**
       GRADES DATABASE PAGAGE - by Elsie U. Zerr
       version II - now can handle up to 16,000
       student tests by storing on disk.
**/
/*** variables used in this package - all static ***/
int gr_file; /* file of student grades */
unsigned n_students, n_tests;
```

```
/**
   gr_init() - initialize the grades database
       this function tells the package how to dimension
       the array and how much storage to reserve, based
       on the number of students and tests.

   Returns:  0 if no errors, -1 if a disk error occurs.
**/
gr_init(num_students, num_tests)
{  char buf[256]; int i;
   n_students = num_students;
   n_tests = num_tests;
   if ((gr_file=creat("grades/wrk", 6) == EOF)
       return -1;
   /* now initialize the work file */
   for (i = n_students*n_tests; i>=128; i-=128)
   {  if (write(gr_file, buf, 256) != 256);
      {  close(gr_file);
         unlink("grades/wrk");
         return -1;
      }
   }
   if (i>0) write(gr_file, buf, i*2);
   if (gr_base == NULL)
      return -1; /* return -1 if error */
   else return 0; /* no error */
}
/**
   get_grade() - return the grade for a given student
                 and test.
**/
get_grade(student,test)
   unsigned student, test;
{  /* first, check the range of subscripts */
   int i;
   if (student >= n_students || test >= n_tests)
      return -1; /* bad subscripts */
   seek(gr_file, (student*test)*2, 0);
   read(gr_file, &i, 2); /* read the test score
   return i;
}
/**
   set_grade() - set the grade for a given student test.
**/
set_grade(student, test, value)
   unsigned student, test;
{  /* first, check the range of subscripts */
   if (student >= n_students || test >= n_tests)
      return -1; /* bad subscripts */
   seek(gr_file, student*test*2, 0);
   write(gr_file, &value, 2); /* save test score */
   return 0;/* no errors */
}
```

Looking very pleased with yourself, you rushed over to see the computer music teacher. When asked by her how you thought of such a neat programming idea, you exclaimed, "Elementary, my dear Olivia!"

Until the next Notes, ciao and God bless you all!

MACH2/PRO-MACH2

The MACH2 package is an excellent product to give you personal control of the placement of files on a disk by the DOS - whether LDOS 5.1.x or TRSDOS 6.x.x. MACH2 contains four utilities: MAPPER provides a map of file utilization by disk granule; ALLOC lets you put files on the disk according to your specifications; for a directory of files, CALC provides a tabular listing of granule requirements covering all media formats supported by the DOS; HANDY is a fast space allocator that uses its own optimum space algorithm to generate files with a minimum number of extents.

Out of the set of programs, we have found one minute bug in CALC. Therefore, fix your LDOS 5.x version of CALC by applying the following patch:

```
PATCH CALC (D02,A8=00)
```

This patch was applied on April 5, 1984 starting with 540041. If you are using PROMACH2, use this patch on CALC:

```
PATCH CALC (D02,B5=00:F02,B5=23)
```

This patch was also applied on April 5th, 1984 starting with 450021.

PaDS/PRO-PaDS Version 1.0

Let's start the PaDS column with a few patches. To begin with, when I recently worked on the 1.2 release of PRO-LC, I noticed that each library member had one extra byte after the CTL-Z (CTL-Z is used for an end-of-file character in assembler source). The extra byte would not cause any adverse behavior for any operation; however, it would add one byte per module. In a 100-member LC library, that extra 100 bytes could cause the file to grow another granule of disk space. I therefore traced the problem and discovered a jump NC instruction that needed changing to a jump NZ. This problem exists only in PRO-PaDS, NOT the Model I/III release of PaDS. The following direct patch corrects this problem in PRO-PADS.

```
. PPADSC/FIX - Applied starting 220085
. Patch directly from DOS Ready via:
PATCH PDS.PDS (D20,8B=20:F20,8B=30)
```

The following "D" patch is again only for PRO-PADS. Invoke the "D" patch procedure after applying PPADSC/FIX. Apparently, some patch code that was applied to PDS prior to its release never got into the source code used to derive PRO-PADS. The patch dealt with the updating of the PDS directory after a PDS(PURGE) operation was performed. To correct this bug, it is necessary to re-build your PRO-PADS file. A Job Control Language (JCL) procedure called REDOPPAD/JCL is included on the DISK NOTES that will accomplish all that is necessary. The JCL performs the following procedure [note, #s# refers to a source drive which will contain the PROPADS members, #d# refers to a destination drive which contains a working copy of the current PROPADS file].

1) Copy the PURGE member out of PRO-PADS via

```
PDS(C) PDS(PURGE) PURGE:#s#
```

2) Apply the PPADSD/FIX as follows:

```
PATCH PURGE:#s# PPADSD/FIX

. PPADSD/FIX - 04/05/84 - Applied 220085
. PATCH TO PROPADS(PURGE)
```

```
X'2920'=CD 01 2C
X'2C01'=2B 2B 2B 2B 2B 36 00 21 00 30 C9
. END OF PATCH
```

3) Copy the remaining members out of PROPADS via the commands:

```
PDS(C) PDS(DIR) DIR:#s#
PDS(C) PDS(LIST) LIST:#s#
PDS(C) PDS(COPY) COPY:#s#
PDS(C) PDS(APPEND) APPEND:#s#
PDS(C) PDS(KILL) KILL:#s#
PDS(C) PDS(RESTORE) RESTORE:#s#
PDS(C) PDS(BUILD) BUILD:#s#
```

4) Remove the old working copy of PROPADS via the command:

```
REMOVE PDS/CMD.PDS:#d#
```

5) Finally, rebuild the PROPADS command file with the statements:

```
!BUILD:#s# PDS:#d# (M=10)
!APPEND:#s# DIR:#s# PDS.PDS:#d#
!APPEND:#s# LIST:#s# PDS.PDS:#d#
!APPEND:#s# COPY:#s# PDS.PDS:#d#
!APPEND:#s# APPEND:#s# PDS.PDS:#d#
!APPEND:#s# KILL:#s# PDS.PDS:#d#
!APPEND:#s# RESTORE:#s# PDS.PDS:#d#
!APPEND:#s# PURGE:#s# PDS.PDS:#d#
!APPEND:#s# BUILD:#s# PDS.PDS:#d#
```

PaDS users have probably at some time or another come across the "Load file format error" message when attempting to invoke a PaDS member with the DEBUG active. Let me explain why this occurs. When the PaDS' Front End Loader (FEL) program is loaded and takes control, it needs to scan the command line for the member name then search the member directory for a match. The FEL makes use of the knowledge that since the system just loaded it, the system's Command File Control Block is still accurate for the PaDS file's access and the system I/O buffer contains the second sector of the file. The FEL alters the NRN offset byte to point to the beginning of the member table (the value is predetermined at the FEL's assembly time) so that it can commence reading of the member table. Unfortunately, if DEBUG is active, the system loads the DEBUG module (SYS5/SYS) which then takes control first. The system uses the same I/O buffer to read SYS5/SYS; thus, when control is passed to the FEL, the I/O buffer no longer contains the second sector of the PaDS file.

There is an easy solution to the problem - short of rewriting the FEL. If you SYSRES the SYS5/SYS module [via a SYSTEM (SYSRES=5)], the system does not have to use the I/O buffer to "load" the DEBUG module. Therefore, if you want to "debug" a PaDS member, just SYSRES the debugger first.

Sidney Bloom, of Maryland writes, "Just a few comments on the PDS. I assume there is a technical reason which mandates the second ')' convention when adding information to a PDS command member for execution, i.e., 's(d) :1', as opposed to 's(d :1', which does not work. It would save time to have a space automatically substituted for the second ')'.

Also, could the '(' be changed to a lower case character, such as '@'? This too would facilitate entry of the PDS parameter."

Sidney's request seemed straightforward enough. I took a look at the Front End Loader and came up with the following patches to accommodate his request [NOTE: these patches are not numbered and will not be applied by MISOSYS to any PaDS disk]:

```
. PaDS patch to not require closing ")"
D00,47=21 38; was 0D 28
D17,44=21 38; was 0D 28

. PaDS patch to change the initial "(" to "c"
D00,32="c"; was "("
D17,2F="c"; was "("

. PRO-PaDS patch to not require closing ")"
D00,4D=21 38
F00,4D=0D 28
D1C,F4=21 38
F1C,F4=0D 28

. PRO-PaDS patch to change the initial "(" to "c"
D00,32="c"
F00,32="("
D1C,D9"c"
F1C,D9="("
```

Now if you have a PaDS file that you created using PDS(BUILD), you can make the changes to it by applying just the "D00," (and "F00," for Model 4 TRSDOS) lines of each respective patch.

Speaking of Front End Loaders, I have received reports that a few of you "experienced" programmers have been designing your own FELs. That's exactly what I had in mind when I implemented the LOADER parameter of the PDS(BUILD) module. If you take a look at the PaDS documentation, it states that the LOADER parameter is used to initialize the PDS with a Front End Loader (FEL) module other than the standard FEL supplied with the PDS utility. Scott Loomer used a slightly patched standard FEL for his Model I/III HELP utility. Jim Kyle has a slew of FELs that we will get into later. I have also been advised that one of my customers in France has combined the LDOS "FILTER" code with a FEL to implement a filter PDS.

Because of the interest in FELs, I am going to supply the source code for the standard FELs in PaDS and PRO-PaDS. First, If you have both types of machines and the PaDS utility for just one, you can use the other FEL to BUILD partitioned data sets for use with the other system. Second, once I document the FEL, it will be easier for the not-so-experienced programmers to create their own FELs. The source code exists in a conditional form suitable for assembly by EDAS. Change the conditional label, DOS6, from -1 to 0 to switch between assembling a PRO-PaDS or a PaDS version respectively.

```
;PDSFEL/ASM - PDS Front End Loader - 03/14/83
          TITLE    '<PDS Loader - Ver 1.0>'
;*=*=*
;         PDS Front End Loader
;
;         Copyright (C) 1981 by Roy Soltoff
;
;         This program performs a search of the MEMBER
;         table for a match to the PDS Member requested.
;         Works with LDOS 5.0.x and 5.1.x & 6.x.x
DOS6      EQU      -1
CR        EQU      13
          IF       DOS6
LOADOFF   EQU      17
          ORG      2600H
          ELSE
LOADOFF   EQU      12
```

```
@GET     EQU      13H
@ABORT   EQU      4030H
@ERROR   EQU      4409H
@LOAD    EQU      4430H
@RUN     EQU      4433H
@DSPLY   EQU      4467H
         ORG      5200H
         ENDIF
;
         COM      '<Copyright (C) 1981 by Roy Soltoff>'
;
PAGE1    EQU      $<-8
ENTRY    LD       A,(HL)            ;Check for '('
         INC      HL                ;Bump to probable member
         CP       '('               ;Start of member?
         JR       Z,GOTBGN          ;Member required...
         LD       HL,MBRREQ$
;
         IF       DOS6
         LD       A,12              ;@LOGOT
         RST      40
         LD       A,21              ;@ABORT
         RST      40
         ELSE
         CALL     @DSPLY
         JP       @ABORT
         ENDIF
;
GOTBGN
;
         IF       DOS6
         LD       A,101             ;Get the flag pointer
         RST      40
         ENDIF
         PUSH     DE                ;Save FCB pointer
         LD       DE,MEMBER         ;Point to save area
         PUSH     DE
         LD       B,9               ;Max of 8+ending char
GETMBR   LD       A,(HL)            ;P/u char
         CP       CR                ;End of line?
         JR       Z,EXITGET
         INC      HL
         CP       ')'               ;End of member?
         JR       Z,EXITGET
         SET      5,A               ;Adapt to lc
         LD       (DE),A
         INC      DE
         DJNZ     GETMBR
BADMBR0  JP       BADMBR            ;Name is too long!
EXITGET  LD       A,9
         SUB      B
         JR       Z,BADMBR0         ;Name too short
         LD       B,A               ;Xfer compare length
CKSPA    LD       A,(HL)            ;Skip past trailing
         INC      HL                ;  spaces to next
         CP       ' '               ;  hard "character"
         JR       Z,CKSPA
```

```
          DEC       HL
          LD        (LINPTR+1),HL     ;Save ptr to INBUF$
          POP       HL                ;Member entry
          POP       DE                ;FCB
          PUSH      DE
          POP       IX                ;FCB to IX
;*=*=*
;         PAGES*LOADADR + proglen + COM + FNAME + TRAADR
;*=*=*
          LD        (IX+5),PAGES*4+FELLEN+35+8+4&0FFH
PARSE     PUSH      HL                ;Save member start
          PUSH      BC                ;Save compare len
          CALL      GETBYT            ;Get type byte
          CP        0EH               ;End of table
          JR        Z,NOFIND          ;Go if end of table
          CP        0CH               ;Member entry?
          JR        NZ,FMTERR         ;Go if load format error
          CALL      GETBYT            ;Get length byte
          LD        C,A
PARSE1    CALL      GETBYT            ;Get member char
          DEC       C                 ;Dec the field length
          CP        (HL)              ;Is it a match?
          INC       HL                ;Bump pointer
          JR        NZ,PARSE2         ;Exit if no match
          DJNZ      PARSE1            ;  else loop for length
          DEC       C                 ;Adj for reading the
          DEC       C                 ;  rest of the field
          DEC       C
          JR        Z,PARSE4          ;Don't read any more if 8
          LD        B,C               ;  chars already tested
PARSE3    CALL      GETBYT            ;Loop for trailing field
          DJNZ      PARSE3
PARSE4    POP       BC                ;Clean up the stack
          POP       HL
          CALL      GETBYT            ;Grab the ISAM #
          LD        B,A              Set up for link
          CALL      GETBYT            ;Read to end of field
          RLCA                        ;Ensure a program file
          JR        C,FMTERR          ;Go if data file
          CALL      GETBYT
LINPTR    LD        HL,$-$            ;P/u ptr to INBUF$
          PUSH      HL                ;Place on stack (RUN)
          PUSH      DE                ;Save FCB
          IF        DOS6
          LD        H,(IY+26)         ;P/U pointer to SVC table
          LD        L,77*2            ;Point to @RUN vector
          LD        A,(HL)            ;  & fetch it
          INC       L
          LD        H,(HL)
          LD        L,A
          ELSE
          LD        HL,(@RUN+1)       P/u RUN vector
          ENDIF
          LD        DE,9              ;Index past CALL @LOAD
          ADD       HL,DE
          EX        (SP),HL           ;Simulate (RUN RET)
          PUSH      HL                ;Put back FCB (LOAD)
```

```
         IF       DOS6
         LD       H,(IY+26)         ;P/U pointer to SVC table
         LD       L,76*2            ;Point to @LOAD vector
         LD       A,(HL)            ;  & fetch it
         INC      L
         LD       H,(HL)
         LD       L,A
         ELSE
         LD       HL,(@LOAD+1)      ;P/u load vector
         ENDIF
;
         LD       DE,LOADOFF        ;Point to LOADER vector+1
         ADD      HL,DE
         PUSH     HL                ;Set ret to POP DE ir) LOAD
         DEC      HL
         LD       A,(HL)            ;P/u hi-order LOADER
         DEC      HL
         LD       L,(HL)            ;P/u b-order LOADER
         LD       H,A
         INC      HL
         INC      HL                ;Pt to ISAM stuff field
         LD       E,(HL)
         INC      HL
         LD       D,(HL)
         EX       DE,HL
         LD       (HL),B            ;Stuff member needed
         EX       DE,HL
         LD       DE,7              ;Point to CALL LDR01
         ADD      HL,DE
         PUSH     IX
         LD       E,(IX+5)
         DEC      E                 ;Adj for increment
         LD       D,(IX+4)          ;P/u hi-order buffer
TABBGN   EQU      $-1
         JP       (HL)              ;Go to LOADER
PARSE2   LD       B,C               ;Not in this field
PARSE5   CALL     GETBYT
         DJNZ     PARSE5
         POP      BC
         POP      HL
         JR       PARSE
GETBYT
;
         IF       DOS6
         LD       A,3               ;Get a byte
         RST      40
         ELSE
         CALL     @GET              ;Get a byte
         ENDIF
;
         RET      Z
         DB       1                 ;Pass the code thru
FMTERR   LD       A,34              ;Load file format error
         DB       1
BADMBR   LD       A,19              ;Illegal file name
         DB       1
NOFIND   LD       A,24              ;File not in directory
```

```
IOERR    OR       40H
;
         IF       DOS6
         LD       C,A
         LD       A,26              ;@ERROR
         RST      40
         ELSE
         JP       @ERROR
         ENDIF
;
MBRREQ$  DB       'PDS Member required!',CR
MEMBER   DB       '        '
FELLEN   EQU      $-ENTRY
PAGES    EQU      $<-8-PAGE1+1
         END      ENTRY
```

The purpose of the FEL is to 1) extract the member specification from the command line, 2) search the partitioned data set member table for a match to the entered memberspec, then 3) interface back to the operating system loader routine for positioning to and loading the requested member. The following outlines the various sub-modules of the standard FEL.

sub-module	function
ENTRY-GOTBGN	Checks for mandatory left parenthesis.
GOTBGN-EXITGET	Extracts member specification from command line.
EXITGET-PARSE	Skips past spaces; Saves command line pointer.
PARSE-LINPTR	Searches the PaDS directory for the first match.
LINPTR-TABBGN+	Interfaces to the system loader.
PARSE2-GETBYT	The rest of PARSE-LIMPTR code.
GETBYT-END	Miscellaneous I/O and error handling.

I believe that in the above code, the only statement that I will add some detail on is the following.

```
LD    (IX+5),PAGES*4+FELLEN+35+8+4&0FFH
```

Recall from the first paragraph of this issues PaDS column, "The FEL alters the NRN offset byte to point to the beginning of the member table (the value is predetermined at the FEL's assembly time) so that it can commence reading of the member table." Well this is the instruction that does the job. Note that the NRN offset is relative byte 5 of the FCB. Register IX is pointing to that FCB, thus the instruction alters the NRN offset value. Now what value do we want to place in the NRN offset? When the FEL takes control, the system's I/O buffer contains the sector that holds the last byte of the FEL's transfer address. Since the system loader uses sector I/O, the NRN offset is always going to be zero when the FEL takes control. This offset value needs to be changed so that it points to the first byte following the transfer address. This value can be calculated by determining the quantity of bytes that the FEL itself takes up. Let's break down the summation in the instruction's operand field and see just how this value is predetermined at assembly time. Since each load block record adds a 4-byte header and EDAS generates 256-byte load blocks, "PAGES*" calculates the number of bytes taken up by load block headers. "FELLEN" adds in the byte length of the FEL itself. The "COM" statement takes up 35 bytes (33 for the message and 2 for the header. The NAME header that is the first part of the

file takes 8 bytes and the transfer address record is 4 bytes. If all of these values are added together, you then strip the high order byte and the result is the NRN offset. If you make any changes to the FEL, the calculation achieved by this instruction will still be accurate.

One subject concerning the PaDS that Is still mysterious to some users is the concept of the /MAP file. Before I discuss MAP files, let me cover the steps that are taken by PaDS when appending a file to a partitioned data set.

An append operation is requested from a command such as,

```
PDS(A) source-file pdsfile.pds
```

After parsing the command line, the pdsfile is opened and tested to ensure that it is a PaDS file. Its member and ISAM directory tables are then read into memory. Next, the file that is to be appended is opened [If this file is a MAP file, different steps are taken]. The name to be used for the MEMBER specification is taken to be the file name field of the source-file specification. Thus, the member name is predetermined and is always going to be the file name of the appending file. This file is then read into memory and examined to determine if its an executable program file or not. Three items must pass the CMD teat for the file to be treated as an executable program file: the extension must be /CMD, the first character must be a X'05' or an X'01', and the fourth from last character must be the transfer address record indicator of X'02'. This last test means that CMD files that have an incorrect end-of-file indicator in the directory will not be considered to be CMD files! You can examine this aspect by listing the file in hex using the LIST command of the DOS. The last four characters should be "02 02 LL HH", where "LL HH" are the low and high order bytes of the program's entry point. CMDFILE or PROCESS can be used to fix any CMD program with an invalid EOF. The file is then appended to the PaDS. If it is a CMD file, the transfer address record is modified to convert it into a form used by the DOS for detecting the end of an ISAM member during its invocation. The PaDS directory is then updated in memory then written back to the PaDS file.

If you take a look at the PPADSD/FIX step 5, be aware that the eight append invocations force eight loadings of the append module, eight readings of the directory, eight files appended, and eight writings of the directory because the append module is invoked eight times. If we use a MAP file to append the files, It would look like this:

```
DIR/CMD,DIR,2600
LIST/CMD,LIST,2600
COPY/CMD,COPY,2600
APPEND/CMD,APPEND,2600
KILL/CMD,KILL,2600
RESTORE/CMD,RESTORE,2600
PURGE/CMD,PURGE,2600
BUILD/CMD,BUILD,2600
```

Each line specifies the name of a file that is to be appended to the PaDS, a name to be used as the member specification [the same as the file name in this case], and an entry point or transfer address. This MAP file could be used for the appending by being invoked with the command:

```
!APPEND REDOPPAD/MAP PDS.PDS (MAP)
```

assuming the map file was named "REDOPPAD/MAP". Two things are evident here. First, using MAP, it is possible to have the member name be different from the file name since the member name is explicitly specified rather than being implicit in the case of not using MAP. Second, the transfer address must be known since it has to be entered on the MAP line. Another thing that may not be

evident is that the MAP method will result in a faster operation where a number of files are to be appended [as in this case]. For instance, in this example, the append module will be invoked once. The PaDS file's directory will be read once. Each file to be appended will be read and then written to the PaDS. The directory will be updated only in memory. The directory will be written once. Thus, file I/O - which is the slowest part of the operation - will be minimized.

The PaDS manual states, "If multiple entry points are required, then you must prepare a MAP file which is used to append the member to the PDS." What does this mean, "multiple entry points"? NOTES Issue I discussed this topic in the case of the COPY and APPEND library commands. Another concept of this is related to data files - or to be appropriate, assembler source routines stored in a PaDS for access via the *SEARCH directive of EDAS. You may have occasion to develop an assembler routine that can be entered at more than one label. As a for instance, consider a routine that unpacks an 8-byte value. It can be used to unpack a 16-byte value by being called twice. You could have code such as the following:

```
PUT16     LD        A,H
          PUSH      HL
          CALL      PUT8
          POP       HL
          LD        A,L
          JP        PUT8
```

This takes the high-order then low-order bytes of the 16-byte value and passes each in turn to the PUT8 routine. You could save 3-bytes of code by having the PUT8 routine as part of the PUT16 routine but still available for the 8-byte conversion. This would look like the following:

```
PUT16     LD        A,H
          PUSH      HL
          CALL      PUT8
          POP       HL
          LD        A,L
PUT8      ............
```

If this combined routine is to be part of your assembler source library, it must be appended with a MAP. You see this file has two entry points, PUT16 and PUT8. They don't relate to addresses the way they do for CMD files. However, the MAP entry MUST have a value in the transfer address field. So you dummy an entry with any value, a zero will suffice. An appropriate MAP line will look like the following:

```
PUTBYTES/ASM,PUT16,0,PUT8,0
```

Another reason for using a MAP to append a file is to provide it with more than one member name. For instance, let's say you were using a small bring-up file data base application written using LC. This was for a small office where you had a few people using it. One program would suffice by being in a PaDS with a member name for each person. Thereby, the program could determine who was accessing it strictly by the program's member name which was obtainable via LC's argv retrieval. The application could then automatically select the data for that individual. I'm sure that you can come up with other uses for multiple names (actually called aliases).

Jim Kyle was mentioned earlier as someone working on his own front end loaders. Jim also passed along a PDS/JCL file that he uses to simulate UNIX-style directories and subdirectories via PDS. The JCL entries will extract, update, or create appropriate nested PDS files to hold his commercial utility-

package archives You can DO the file with no parameters to get immediate on-line assistance. Due to the length of this file, only an extract of it to give you some flavor of his work will be printed here. The entire file is part of DISK NOTES.

Jim uses partitioned data set files named according to a convention. He uses an extension of PDS. The filename actually is composed of two parts – a tag field and an extension field that contains the extension of the files that have been appended to the PaDS [apparently Jim segregates his PaDS files so they each hold only members of a given file type]. The following is an abbreviation of the PDS/JCL file found on DISK NOTES.

```
//. PDS/JCL Library - 12/13/83 - jwk
//. Must call with ONE of these labels:
//. DO PDS (@GET/@PUT/@UPD/@FROM/@TO/@MAKE1/@MAKE2...
//.   @PACK/@GETHLP)
. NO SYSTEM JCL GENERATED!!!!
//QUIT
@GET
//. PDSGET/JCL - 12/12/83 - jwk
//IF -C+-T+-N+-D
//. USAGE: do pds (@get,t=tag,c=ext,n=filename,d=drive)
//QUIT
//END
PDS(C) #T##C#/PDS(#N#) #N#/#C#:#D#
//EXIT
@PUT
//. PDSPUT/JCL – 12/12/83 - jwk
//IF -C+-T+-N
//. USAGE: do pds (@put,t=tag,c=ext,n=filename)
//QUIT
//END
PDS(A) #N#/#C# #T##C#/PDS.PDS
KILL #N#/#C#
//EXIT
@UPD
//. PDSUPD/JCL - 12/12/83 - jwk
//IF -C+-T+-N
//. USAGE: do pds (@upd,t=tag,c=ext,n=filenaae)
//QUIT
//END
PDS(K) #T##C#/PDS.PDS(#N#)
PDS(P) #T##C#/PDS.PDS
Y
PDS(A) #N#/#C# #T##C#/PDS.PDS
KILL #N#/#C#
//EXIT
```

THE PROGRAMMER'S GUIDE

MISOSYS began publishing THE PROGRAMMER'S GUIDE TO LDOS/TRSDOS VERSION 6 back in August of 1983. The GUIDE [not to be confused with THE HITCHHIKER'S GUIDE] is designed to be used by the TRSOOS Version 6 assembly language programmer. The GUIDE, in its 214 pages, covers the procedures to be followed when writing assembly language programs for use with version 6 TRSDDS.

For those that have already acquired a copy of the GUIDE, there are a few corrections to make. The following are a list of all of the corrections made to the First Edition 1983.

1) Pages 3-47 and 3-48 was revised on September 13, 1983 to change all references of "IX" to "IY". If your book does not have the revised page, contact MISOSYS and request the new page.

2) page 2-32, change the statement beginning with "+3 - Flag to indicate" to read "+3 - Contains the receive character code interpreted as BREAK." Note: I suggested to LSI that they change the COM/DVR module to interpret a value of NULL to indicate that no checking for BREAK/PAUSE/ ENTER conditions should be done if the BREAK character code value is set to NULL. This was implemented in release 6.2.0.

3) Page 6-139, change VIDEO POKE register B value from "B => 1", to 'B => 2".

4) Page 6-139, SET CURSOR POSITION; change the statement, "A <= Will ..." to read, "A <= Will contain the error code if an error was encountered."

5) Page A-150, paragraph 4; change the phrase "256 load bytes respectfully" to read "256 load bytes respectively".

6) Page 4-64, sample code; change label HASHMPW to HASHSPEC, and change label PSWDPTR to SPECPTR.

7) Page 5-90, sample code; move the two lines "LD A,@REW; RST 40" to precede the line "LD HL,0".

8) Page A-160, sample code; change the line "DB MODDCB-BEGIN-5" to read as follows: "DB MODDCB-ENTRY-5".

9) Page A-190; the statement, "CRESP DB 0" should precede the statement "DW CPARM+1".

10) Page A-191; Remove the statement, "END TRAP".

Now that the misprints are corrected, let me turn to a lighter side. I am happy to report that Jim Ross was the first person to decipher the "words" written on the open book as depicted on the front cover of THE PROGRAMMER'S GUIDE. Jim called in his findings on March 2, 1984. I am sad to say that Jim has so far been the ONLY one to realize that there was a hidden message! We may write Serious Software, but folks, we're not serious all the time'

SOLE

In the last issue, I challenged all those that were using SOLE with two sided disk drives, to come up with a patch to BACKUP to support a mirror image backup of the SOLEd two-sided system disk. One response said it was easy if I changed SOLE2 to format track zero on both sides in single

density. Actually, if you think about it in detail, that scheme will not work. Follow my analysis as discussed in the last issue.

John Blair suggested the three step procedure of 1) lockout track zero on the source disk; 2) backup source to destination; 3) backup the BOOT/SYS. We don't run with two-sided drives so I can't test his procedure; however, since it inhibits BACKUP from touching track zero during the mirror image copy phase, it should work. Of course, locking out track zero first requires you to use either DEBUG on the directory or some other utility (like FED). Since I asked for a patch to BACKUP that would then be transparent to the user, John's suggestion works but is not the answer I was looking for.

Jeffrey McLean did his homework! Although he worked up a modification to QFB instead of to BACKUP, since QFB is supplied as part of LDOS 5.1.4, his actions are perfectly acceptable. Jeffrey's procedure, outlined in a letter to me, follows. Since the coding may be lengthy, the code has been included on the DISK-NOTES.

"I am writing to you to inform you of some changes I have made to SOLE1 and QFB/CMD. The changes to QFB will allow it to work with double sided disks that have been SOLEd.

Included with this letter you will find screening data for QFB/CMD, a partial listing of SOLE1/CMD with my changes to it marked on it. One of the changes to SOLE1 is to correct a hitherto undiscovered bug, one is needed for my fix to QFB, and the rest are purely gratuitous.

Starting out with SOLE1, the first change is to fix a potentially disastrous bug. When SOLE1 checks to see if gran one of track zero is in use, it does not check to see if gran two is in use. Since that second gran is annihilated when track zero is reformatted, if for any reason gran two is used without gran one being used, the file occupying that gran would be lost. This change will have SOLE1 check both grans one and two to see if they are allocated. A second change is to the code that allocates all of track zero. I have changed it so that the three grans on the second side of the disk remain unallocated.

Moving on to page two, the only change here is to an error message, so that the message more accurately describes the detected error condition for the change on page one.

On page three, I have changed the code that locks out track zero so that the three grans on the second side of the disk remain usable . Only gran two on side one is locked out now. A second change on this page is used in conjunction with my fix for QFB. This is to set bit seven of GAT byte X'CD'. According to LDOS documentation, this bit is not used for anything; so I am using it to indicate that the disk has been SOLEd.

Moving on to page one of the QFB listing, the only change on this page is to save GAT byte X'CD' from the source disk.

On page two, I have changed some EQU statements that came between load blocks to DS statements. This is to preserve the correct amount of storage when the program is assembled with the changes made.

Page three contains the actual code to filter out the read request for sectors ten through seventeen of track zero on a SOLEd disk. Since this same code is used by the program for reading sectors from both the source and destination disks, the first few lines check the A register, which will contain an X'FF' if the read request is for the source disk.

On page four, I have changed some more inter-block EQU statements to DS statements, and reserved one byte of storage for GAT byte X'CD'.

I have so far used the fixed QFB to create nine duplicates of three double sided SOLEd disks. I then used COMP to compare the source and destination disks. and every byte was the same on both disks. You may want to pass this information on to other people who are trying to use SOLE on double sided disks.

SOLE1/ASM, Page 1: Change code starting at 5223H to read as follows:

```
        PUSH    AF
        LD      A,(HL)
        PUSH    AF
        AND     6
        JR      Z,M525C
        POP     AF
        OR      7
        LD      (HL),A
        POP     AF
        CALL    M5347
        JR      NZ,M5257
        LD      HL,M52B6
```

SOLE1/ASM, Page 2: Change message at label "M532E" to read as follows:

```
M532E   DB      'Track zero, granule one or two in use!',13
```

SOLEl/ASM, Page 3: Change patch code at label "M5347" to as follows:

```
M5347   LD      L,60H
        LD      A,(HL)
        OR      4
        LD      (HL),A
        PUSH    HL
        LD      L,0CDH
        LD      A,(HL)
        OR      128
        LD      (HL),A
        POP     HL
        JP      M5239
```

QFB/TXT:

```
$5627-562C,#562D,$562F-5634,#5635,$5637-563C,#563D,$563F-5644,#5645
$5647-564C,#564D,$564F-5654,#5655,5657
$5701-5710,$5711-577F,$5780-57A8,$57A9,57B8,$57B9-57CD,$57CE-57E4
$57E5-5806,$5807-582A,$582B-5857,$5858-587C,$587D-5898,$5899-58B6
$58B7-58D8,$58D9-5906,$5907-591B,$591C-593B,$593C-5952,$5953-595C
$595D-5966,$5967-59E8,$59F8-5E0F,$5E10-5E27,$5E28-5E3F,$5E40-5E57
$5E58-5E6F,$5E70-5E87,$5E88-5EAD,$5EAE-5ED6,$5ED7-5EE6,$5EE7-5EFB
$5EFC-5F4E,$5F4F-5F60,5615,5619,561D,5FA1-5FB4,609A-60A3,5CAA-5CE9
5CEA-5D43,5D44-5D88,5D89-5DF7,5412-5414,566E-5675,5676,5677,5678
5679,567A,567B,567C-567D,567E,567F,5680,5681-5682,5683-5684
5685-5686,5687-5688,5689,568A,568B,56BC.
```

QFB/ASM Page 1: After "LD HL,(M63CC)" between label "M5303" and label "M5335", add the two lines: "LD A,H" and "LD (SAVEGAT),A".

QFB/ASM Page 2: Change EQUates from label "M568C" to read:

```
M568C   DB      0
M568D   DS      2
```

```
M568F     DS        1
M5690     DS        1
M5691     DS        80
M56E1     DS        32
M5701     DB        'Parameter error',0DH
```

QFB/ASM, page 3: Change code at label "M5BBF" to read as follows:

```
M5BBF     LD        A, (M568A)
          OR        A
          JR        Z,M5BD2
          PUSH      AF
          CP        0FFH
          JR        NZ,NOSKIP
          LD        A,(SAVGAT)
          BIT       7,A
          JR        A,NOSKIP
          LD        A,D
          OR        A
          JR        NZ,NOSKIP
          LD        A,E
          CP        10
          JR        C,NOSKIP
          CP        18
          JR        NC,NOSKIP
          POP       AF
          JR        M5BCD
NOSKIP    POP       AF
          CALL      M5C6A
SKIP      JR        Z,M5BCD
          CP        6
          RET       NZ
M5BCD     CALL      M5FEB
          XOR       A
          RET
```

QFB/ASM, Page 4: Change EQUates and add as follows:

```
M5CAB     DS        1
M5CA9     DS        1
SAVEGAT   DS        1
M5CAA     DB        0AH,07H,00H,05H,01H,06H
```

Remember, the above changes are available on the DISK-NOTES diskette. The source to SOLE1/ASM is also included. Implementing this procedure will require you to have access to DSMBLR Version III and an assembler (preferably, EDAS).

ZCAT and PRO-ZCAT

This disk cataloger is great for your LDOS 5.x or TRSDOS 6.x disk collection. Once you go through the efforts of cataloging your files, you will find that putting your hands on that file needed right now will be an easy task. I know of at least one PRO-ZCAT user who has about 100 diskettes in his collection cataloged with this application. I know this because of a report that the second screen display for the directory of disks is off by one line [it takes a catalog of over 70 or so disks to even require two screens on a model 4]. Thus, the following patch to PRO-ZCAT corrects this.

```
PATCH ZCAT (D0B,46=04:F0B,46=05)
```

ZGRAPH and PRO-ZGRAPH

First let's repeat the update notice for ZGRAPH version 4.x. Karl Hessinger has developed an enhanced version of ZGRAPH. Version 5 is written completely in assembly language - it is fast. It includes an improved circle generating routine [improved for blinding speed]. It supports a "fill" function that fills in all pixels of a boundary. This operation is similar to the "paint" operation under graphic BASICs. Version 5 uses all available RAM for buffer space. With the ability to store 15-25 screens and save all the screens into a single file, Karl added another utility which "plays" the screen images in a timed succession. Karl added magnification and reduction functions to ZGRAPH so that you can take a rectangular block of the screen image and either expand it or shrink it. You get nine levels of expansion and two modes of reduction. Version 5 supports the Radio Shack DMP-series of printers - including the DMP-2100 in addition to the Epson printers.

If you presently own Model I/III ZGRAPH version 4 and wish to upgrade to Model I/III ZGRAPH version 5, you can do so by returning your ZGRAPH master diskette to MISOSYS. There will be a $20 charge. In return, you will get an updated diskette and a brand new version 5 ZGRAPH manual.

It turns out that not all DMP printers support the 1/216" line feed mode. To be specific, the DMP-200 printer does not. The RSBINCAT program makes use of that mode; thus, the DMP-200 does not properly print the images. The problem was due to the use of the 1/216" line feed spacing mode which was assumed to be available on all of the DMP-xxx printers. However, that line feed spacing mode is not supported by the DMP-200. Therefore, instead of sending 16 of the 1/216" feeds, the patch changes RSBINCAT to send 5 of the 1/32" feeds. This equates to 15/216" in lieu of 16/216"; thus, the paper movement is short by 1/216". if you look closely at a graphics print, you may notice a slight darkened row of dots. This is caused by a slight overprinting of the bottom row of one pass by the top row of the subsequent pass. If you change to 6 1/32" feeds, there will be a whitespace gap of 2/216". 1 believe that 15/216" spacing will be better.

The DMP2005A/FIX is for the LDOS 5.1.x Model I/III version of RSBINCAT while the DMP2006A/FIX is for the TRSDOS 6.x version. Apply the proper patch (one for Model I/III the other for PRO-ZGRAPH) to the RSBINCAT program so that it can print the graphics /BIN files on a Radio Shack DMP-200 printer.

```
. DMP2005A/FIX - 01/31/84
. Patch for RSBINCAT - LDOS 5.1.x Version (Models I/III)
. This fix allows RSBINCAT to support the Radio Shack DMP-200
. printer which does NOT support 1/216" line feed. After the
. patch is applied, the DENSE parameter is ineffective since
. DENSE requires use of the 1/216" line feed paper movement.
D00,95=18; Deactivate DENSE parameter
.; WAS 28
D02,DE=05; Change to 5 feeds of 1/32"
.; WAS 10
D03,2D=32; Change escape sequence to 1/32" from 1/216"
.; WAS 33
End of patch

. DMP2006A/FIX - 02/23/84
. Patch for RSBINCAT - TRSDOS 6.X Version (Models 4, etc)
. This fix allows RSBINCAT to support the Radio Shack DMP-200
. printer which does NOT support 1/216" line feed. After the
. patch is applied, the DENSE parameter is ineffective since
. DENSE requires use of the 1/216" line feed paper movement,
D00,69=18
F00,69=28
```

```
D02,BE=05
F02,BE=10
D03,09=32
F03,09=33
. End of patch
```

ZSHELL

The following is a suggestion from David Lamkins concerning the use of ZSHELL to redirect I/O for LC-compiled programs. "...specify '#option KBECHO OFF' as well as '#option REDIRECT OFF'. If you don't, you'll get a copy of the standard input interspersed with the desired output."

While I'm talking about LC and ZSHELL in the same breath, don't forget the if you have ZSHELL installed and wish to invoke an LC-compiled program using its I/O redirection (assuming you didn't compile it with REDIRECT OFF), you can pass the command line through ZSHELL by starting it with a double quote ["]. This was mentioned in NOTES II.

David also brought another problem with ZSHELL to our attention, He inadvertently entered a text file specification in lieu of a program for a piped command line. This resulted in a crash that was beyond the control of ZSHELL since it relates to an inherent and long-lasting bug in LDOS version 5. The following scenario describes the chain of events that occurred.

ZSHELL was installed and directed the pipe files to a disk drive (default of drive 0). ZSHELL did not request that the pipe files, Z0/PIP and Z1/PIP be allocated with any file space. The command line was parsed and ZSHELL set up for piping. It did this by redirecting the *DO output to the first pipe file, Z0/PIP in that case. It also set up linkage to the system and set an internal pipe flag. ZSHELL then invoked the @CMNDI system call with the revised command line (just the text filespec). LDOS acted on the @CMNDI and invoked @RUN of the "program". @RUN invoked @LOAD. LDOS attempted to load the file and detected a load-file-format-error (since it was not a CMD file). The @LOAD routine set the abort flag and then passed the error back to @RUN which then passed control to the @ERROR system call (with the flag set for @ERROR to exit to @ABORT).

The @ERROR module is in SYS4/SYS. It transmitted the error message string through @LOGOT (which sends it to both @DSPLY and any active JOB LOG). Since @DSPLY sent the character string a byte at a time via @PUT to the video DCB, this video output was redirected by ZSHELL to the pipe file - the same as if the output came from the assumed executing program. ZSHELL used @PUT to write the output. The LDOS device handler sensed the request to be a file reference and passed control to the character I/O file access routines. When the first BYTEIO PUT request was issued, the file access routines had to get the first granule of disk space allocated. Under LDOS 5.1, disk space allocation is performed by the SYS8/SYS module - which was installed in the overlay area. [The LDOS file access routines have a check which re-installs SYS1/SYS after returning from SYS8/SYS only if SYS1 was previously resident. In this case, SYS4 was previously resident so nothing was done to restore it.] After @LOGOT completed the output of the message string, it returned - supposedly to SYS4/SYS which originally invoked @LOGOT. Unfortunately, SYS4 was not resident but SYS8 was. Since now code in SYS8 was executing that should not be, the exact result would be unpredictable - other than it could crash.

Now the system cannot reinstall modules other than SYS1 if SYS8 got flopped in since the other modules are not necessarily re-entrant. This is so with SYS4 - the error string and return address are stored in its data area - part of the overlay.

The scenario is repeatable under one other circumstance. If a JOB LOG is active and routed to a disk file with unallocated space and @ERROR issues a message string via @LOGOT, the SYS0 conflict is also evident. One thing that could be done is to warn the users about this system bug. The other thing to do is advise them to pre-allocate space for JOB LOG files and the pipe files, in the case of ZSHELL. Thus, if the pipe files are pre-allocated, the problem will not materialize. The next release of ZSHELL will pre-allocate the pipes.

CONTRIBUTIONS

The CONVDOS program provided in Issue II has been improved so that the REMOVE parameter works without requiring you to re-establish the logical drive. Rather than repeat the hex listing, the new file as well as its assembler source is included on the DISK NOTES, Issue III. CONVDOS is used to be able to read files off of a DBLDOS or NEWDOS80 double density disk.

The LDOS community has long relished the LSCRIPT patch to Model I SCRIPSIT which provided useful features (such as a directory command, direct output of printer codes, use of LDOS type-ahead, keyboard command revamping allowing use of KSM, and more). Well since I was the individual responsible for putting all of that together, I decided to implement a patch to Model 4 SCRIPSIT which supported those types of features. This patch turns SCRIPSIT into MSCRIPT, since it is implemented by MISOSYS. It consists of three patches: MPATCH1, MPATCH2, and MPATCH3.

What does MPATCH provide? First, it provides a <BREAK> "Q" command that gives you access to all TRSDOS 6 library commands. With this you can invoke the directory command, free, device, remove, and other library commands. In fact, any utility that executes and uses only the library region of memory can also be accessed via "Q RUN program".

MPATCH also defaults the file extension to "/SCR" on <L>oad and <S>ave commands as well as default to "/TXT" on ASCII loads and saves. MPATCH provides improvements to the keyboard activity by removing SCRIPSIT's internal type-ahead thereby utilizing the system's type-ahead solely. Optionally, you can also revamp the SCRIPSIT command keys to the top row to allow use of TRSDOS 6.x KSM function.

MPATCH provides the ">$" and ">#" support as follows:

">$" allows you to transmit a series of byte values directly to the printer during the printing phase of SCRIPSIT. The byte values are identified as decimal entries separated by commas. Thus, the control line:

>$27,10,27,13

will output the four bytes, "ESCAPE LF ESCAPE CR", to the printer. The ">$" control can be used anywhere a format control line can be specified.

">#" allows you to temporarily suspend print output until a keyboard entry is made. Depression of <SHIFT><CLEAR> will abort the printing after a pause. This control may be useful to provide a pause for changing type wheels. Where the Model I/III LSCRIPT patch supported direct print output of key strokes with the ">#" control, MSCRIPT does not.

MPATCH is applied to Model 4 SCRIPSIT version 01.00.00. It consists of three files: MPATCH1/FIX, MPATCH2/FIX, and MPATCH3/FIX, To implement the patch, use the TRSDOS 6.x

PATCH facility. Apply the first two fixes. If you want to change to the upper row of keys, apply the third fix. Due to the length of these fix files, they will not be printed (you really would not want to key them in). Therefore, they will be available only on the DISK NOTES. Additionally, an MPATCH documentation file will be on DISK NOTES.

Another item for the Model 4 users is a program that can be used to change the code values generated by the three functions keys. FKEY can change either the unshifted or shifted key values. FKEY displays the current values (after any changes have been made). FKEY also permits you to reset the keys to their default values. FKEY executes totally within the library memory region so it can be invoked from any program that permits the invocation of library commands. Programs such as BASIC which restrict you to DOS library commands only, may still grant you access to FKEY via the SYSTEM verb if you invoke FKEY via the RUN library command [see, there really is a valid use for RUN]. From BASIC, for example, you can change the F1 key to generate the carat (used for exponentiation and generated via <CLEAR><;>) with the syntax:

```
SYSTEM"RUN FKEY (F1="^")
```

Thereafter, depressing F1 will generate the carat. Please note that you cannot use FKEY to allow a function key to emulate the <SHIFT><@> function of PAUSE. Since the DOS specifically scans (SHIFT> and <@> to detect the PAUSE function, even though another key may generate the code value of X'60', it will NOT be detected as PAUSE. Both the source and object code files for FKEY are on DISK NOTES. The source listing follows for those that may just want to look at the code.

```
;FKEY/ASM - 05/03/84
;*=*=*
;Program to set function key codes in TRSDOS 6.x
;         Use: FKEY (DEFAULT,F1=ddd,S1=ddd,...
;         Sn = shifted Fn. Formats: Fn="c"; Fn=ddd; Fn=x'hh'
;         Released to public domain by Roy Soltoff – 04/13/84
F1        EQU       0
S1        EQU       1
F2        EQU       2
S2        EQU       3
F3        EQU       4
S3        EQU       5
          ORG       2600H
FKEY      PUSH      HL                  ;Save command pointer
          LD        HL,HELLO$
          LD        A,10                ;@DSPLY
          RST       40
          POP       HL
          LD        DE,PRMTBL$          ;Get parameter values
          LD        A,17                ;@PARAM
          RST       40
          JP        NZ,PRMERR
          LD        DE,KIMOD$           ;Locate *KI driver data area
          LD        A,83                ;@GTMOD
          RST       40
          JP        NZ,KIERR
          LD        HL,32               ;Index to function key table
          ADD       HL,DE
          PUSH      HL                  ;Save on stack
          LD        IX,PRMTBL$+4        ;Test if user wants to
          LD        A,(IX+F1*6)         ;  default to system values
          OR        (IX+S1*6)
          OR        (IX+F2*6)
          OR        (IX+S2*6)
```

```
          OR        (IX+F3*6)
          OR        (IX+S3*6)
          JR        NZ,CHGKEY
DPARM     LD        BC,0                ;Init DEFAULT=OFF
          OR        B
          OR        C
          JR        Z,CHGKEY            ;Go to p/u current key settings
MOVKEYS   LD        HL,DEFKEY           ;Move new function key
          POP       DE                  ;   code table
          PUSH      DE                  ;Save pointer to table
MOV6      LD        BC,6
          LDIR
          POP       DE                  ;Recover table pointer
          LD        HL,KEYS$-1          ;Index to start of buffer
          LD        B,3                 ,Loop for 3 keys
DSPLP     PUSH      BC
          LD        BC,6
          ADD       HL,BC               ;Point to next field
          CALL      GETKEY              ;Unpack unshifted
          CALL      GETKEY              ;Unpack shifted
          POP       BC
          DJNZ      DSPLP
          LD        HL,KEYS$
          LD        A,10                ;@DSPLY
          RST       40
EXIT      LD        HL,0
          RET
GETKEY    LD        A,(DE)              ;P/u key code
          INC       DE
          LD        C,A
          LD        A,98                ;@HEX8
          RST       40
          INC       HL
          RET
CHGKEY    LD        DE,DEFKEY
          POP       HL                  ;Point to KI's table
          PUSH      HL                  ;Save for update
          LD        B,6                 Init for six tests
          LD        DE,DEFKEY-1         ;Point to start of table
KEYLP     CALL      PARSE
          DJNZ      KEYLP
          JR        MOVKEYS
PARSE     LD        A,(IX)              ;P/u response code
          INC       IX
          LD        L,(IX)              ;P/u vector
          INC       IX
          LD        H,(IX)
          INC       IX
          INC       IX
          INC       IX
          INC       IX
          INC       DE                  ;Bump pointer to table
          OR        A
          RET       Z
          BIT       7,A
          JR        NZ,PARSE2
          BIT       5,A
          RET       Z
          LD        A,(HL)
          INC       HL
          LD        H,(HL)
          LD        L,A
PARSE2    LD        A,(HL)              ;Get the new value
```

```
          LD        (DE),A              ;Replace with new value
          RET
KIERR     LD        HL,KIERR$
          DB        0DDH
PRMERR    LD        HL,PRMERR$
          LD        A,12                ;@LOGOT
          RST       40
          LD        HL,-1               ;Init for error
          RET
PRMTBL$   DB        80H
          DB        0A2H,'F1',0
          DW        F1*2+PARMS
          DB        0A2H,'S1',0
          DW        S1*2+PARMS
          DB        0A2H,'F2',0
          DW        F2*2+PARMS
          DB        0A2H,'S2',0
          DW        S2*2+PARMS
          DB        0A2H,'F3',0
          DW        F3*2+PARMS
          DB        0A2H,'S3',0
          DW        S3*2+PARMS
          DB        57H,'DEFAULT',0
          DW        DPARM+1
          NOP
DEFKEY    DB        81H,91H,82H,92H,83H,93H
PARMS     DW        0,0,0,0,0,0
HELLO$    DB        10,'FKEY - Change or reset Model 4 function keys'
          DB        10,'Public Domain - Written by Roy Soltoff'
          DB        ', All rights reserved.',10,13
KIMOII$   DB        '$KI',3
PRMERR$   DB        'Parameter error!',13
KIERR$    DB        'Can''t locate *KI driver',13
KEYS$     DB        'F1 = xx,yy; F2 = xx,yy; F3 = xx,yy',13
          END       FKEY
```

The following suggestion was contributed by James O. Mullin. James writes, "...I use the last sector of a file for whet you might call internal documentation. When some question comes up, rather than having to dig out a notebook or a piece of paper where the patch or other information has been written down, I just look on the last sector, using FED or FED II. In fact, on the Electric Pencil, I have added an extra sector which serves no purpose except more documentation. I have found that when you copy the file from one diskette to another, the extra information or garbage on the last sector goes along. Thus, I never lose my documentation. Also, I have in some instances added a name or version number to a piece of software, so that when I go to the last sector I can see which version I am using.

Now with two contributions designed strictly for the Model 4 user, let's provide something for the old trusty Model I. Mike Kaizen, of Rockville MD, writes, "Enclosed is a little program you may want to use in the next issue of NOTES. I wrote it for use with my remote terminal so that I could have the break and pause features which are not available with the LDOS RS232 driver [Mike may have been referring to an earlier RS232 driver - the RS232T driver provides a BREAK parameter that supports the KFLAG BREAK-PAUSE bits] I use task slot zero to sample the RS232 interface for 01H or 60H and set the keyboard flag. All credit, however is due to your articles in the LDOS QUARTERLY and your sample filter program in the LDOS manual. It is relocatable to high memory and will list with MEMDIR in MSP-01." The following is a listing of Mike's KBFLAG filter. The source and object code files are included on DISK NOTES.

```
;This program permits the user to use break and pause from a
;remote terminal when using the TRS-80 MODEL I.
;CONTROL A replaces the break and ` replaces the shift @.
;XON/XOFF are also implemented. Control S will pause the system
;and Control Q will restart it.
;After activating the RS232 driver and doing the appropriate linking
;or routing, type KBFLAG to activate the use of break and pause.
;THIS PROGRAM IS LOADED AT THE TOP OF MEMORY AND RESETS HIGH$.
;The file name will be readable in MEMDIR available from MISOSYS as
;part of the MSP-01 package.
;THIS PROGRAM IS BASED ON ARTICLES BY ROY SOLTOFF
;of MISOSYS on relocation, the kbflag and task control.
;***** Written by Mike Kaizen - Rockville, Md. - April 1983 *****
          TITLE       <KBFLAG>
HIGH$     EQU       4049H
@EXIT     EQU       402DH
@ADTSK    EQU       4410H
USTOR$    EQU       4DFEH
KFLAG$    EQU       4423H
          ORG       5200H
BEGIN     PUSH      DE
          LD        BC,LAST-ENTRY
          XOR       A
          LD        HL,(HIGH$)
          SBC       HL,BC
          LD        (HIGH$),HL
          PUSH      HL
          ADD       HL,BC
          LD        (BUFFHI),HL
          POP       HL
          INC       HL
          DI
          EX        DE,HL
          LD        HL,ENTRY
          LDIR
          EI
          LD        HL,(HIGH$)
          INC       HL
          LD        (USTOR$),HL
          LD        DE,USTOR$
          LD        A,0
          CALL      @ADTSK
          POP       DE
          JP        @EXIT
ENTRY     JR        START
BUFFHI    DW        0
          DB        START-TITLE
TITLE     DB        'KBFLAG'
START     LD        A,(4423H)
          AND       0F0H
          LD        (4423H),A
          IN        A,(0EAH)
          BIT       7,A
          RET       Z
          IN        A,(0EBH)
          AND       7FH
          CP        01H
          JR        Z,SETBRK
          CP        60H
          JR        Z,SETPSE
          CP        13H
          JR        Z,INTR
          RET
```

```
SETBRK    LD        A,(KFLAG$)
          SET       0,A
          LD        (KFLAG$),A
          RET
SETPSE    LD        A,(KFLAG$)
          SET       1,A
          LD        (KFLAG$),A
          RET
INTR      IN        A,(0EAH)
          BIT       7,A
          JR        Z,INTR
          IN        A,(0EBH)
          CP        11H
          JR        NZ,INTR
          RET
LAST      EQU       $
          END       BEGIN
```