April 1988
Vol 1-3

TS·2068 UP·DATE

the user's NEWS

New Sections

***** COMPUTING WITH LARKEN DISK *****

***** THE DISK DRIVE ORPHANAGE *****

The Millenia K
The TOS DISK

****

***** Major Softwares This Issue *****

** Feature Program: "BUDGET" by: Bob Mitchell

** Five New Disk Managers by: Bob Hartung

****

*** P L U S ***

Umteen articles on programming.

TS-2068 UP-DATE
1317 Stratford Ave, Panama City, FL 32404
904 871 4513

Z88.
THE FACTS BEHIND A NEW BREED OF PERSONAL COMPUTER.

The Z88 is smaller than an A4 pad, weighs less than 2lb, and measures only ⅞" thick. No other portable personal computer offers so much, in so small a package.

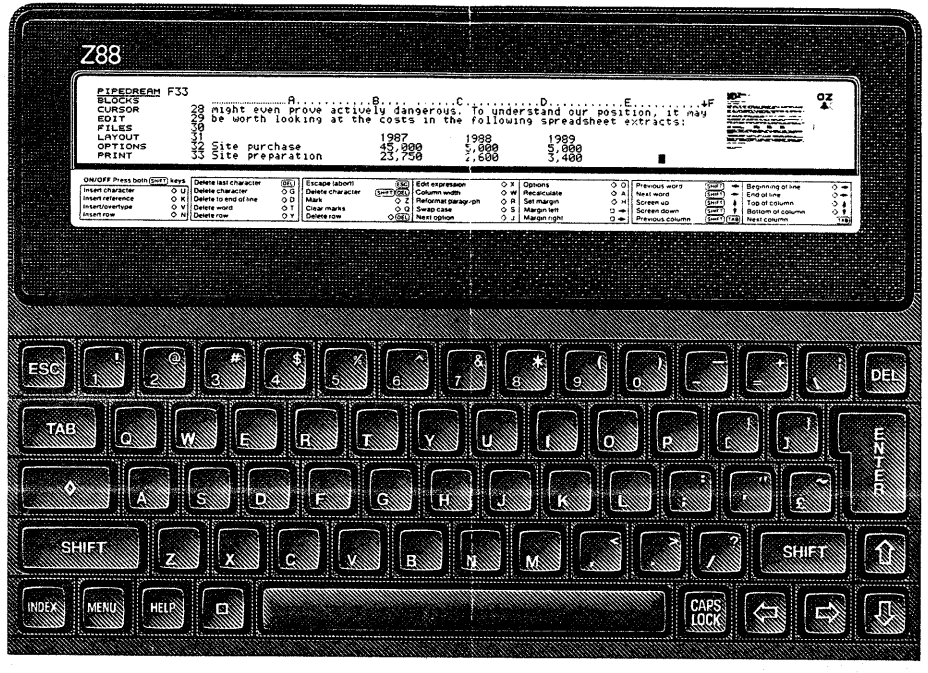
And at so small a price — $479.95

Memory
The Z88 comes with 32K of RAM built in, of which around 20K is available – enough for about 2,000 words of text. This is easily expandable to 416K by simply slotting in additional RAM packs (see back cover). With the introduction of soon-to-be-released 1 Mbyte packs, total RAM of 3 Mbytes will be available – enough to hold the complete works of Shakespeare.

Keyboard and display
The Z88 has a full QWERTY keyboard with virtually silent, short-travel keys. It includes special function keys, such as ◇, □, INDEX and MENU which make it exceptionally easy for novice users to quickly find their way around the sophisticated built-in software.

The screen is a state-of-the-art supertwist LCD, which provides a massive improvement on normal displays in both contrast ratio and viewing angle – providing a clear, sharp read-out. The screen's format is 8 lines×106 characters, and provides a large working area of 8×94.

The remainder of the screen area consists of sections which hold menu details (including all the Z88's commands); a unique page map (which shows you where you are on a complete page); diary date; filing and directory details; and operating system details which give you information on the Z88's status (such as battery strength).



Software
The Z88 comes with a built-in suite of applications software, which is exceptionally capable, yet extremely easy to use.

The operating software, OZ, allows you to switch between, for example, word processing and spreadsheet applications – without the delay of loading separate disk-based software into the computer's memory. With the Z88, it's just a matter of pressing a couple of keys.

Built-in applications software includes *Pipedream*, which features word-processing, spreadsheet and database functions.

Word Processing Facilities include block marking and movement, searching and replacing, and multi-column layout. The display is fully WYSIWYG (What You See Is What You Get), showing bold, italics, underlining, centering, and dynamic page breaks as they appear on the printed page. Each page of the document is represented by a page map which shows the layout of the text.

Spreadsheet Invaluable for financial planning, this application allows you to define a series of formulae within cells (where a row and a column meet) and then to enter a series of numerical variables for manipulation. It allows profit and loss, cashflow and budgets to be quickly and easily arrived at.

Database This works on the same principle as spreadsheet – so data being stored for later retrieval is held in a known cell where it can, for example, be manipulated into ascending sequence.

Diary The Z88 diary software is a free-form log that can be configured in any way you choose, within the constraints of the Z88's memory. It's particularly useful when used with the Z88's calendar and clock to find dates and set alarms.

Popdowns include a powerful calculator, clock, calendar and alarm. A printer editor allows you to configure Z88 codes to match a wide range of printers, and to store printer definitions if you're using more than one type.

Resident programming language is BBC BASIC with assembler. Communications to the outside world are achieved using a VT52 terminal emulation mode or the Import/Export facility which allow transmission and receipt of data files between the Z88 and other PCs.

Power supply
The Z88 runs on four commonly available AA batteries, which provide around 20 hours of active computing, or a year on standby mode. Switching off puts the Z88 into standby; all data in RAM stays live, drawing minimal power from the Z88's batteries. And when the batteries need to be changed, a special capacitor (or the mains adapter) preserves the data.

The Changing Complexion of TIMEX-SINCLAIR Computing

Seen at the Orlando 1988 Winter Fest

This issue of Up-Date was delayed a week to report upon the Orlando 1988 Winter Fest. There were quite a few subscribers to UP-DATE in attendance and several attendees became new subscribers. From my standpoint the Fest was a success. The display room was filled most of the time. Most area user groups were represented. The C.A.T.S. group was busy taking orders for their TS-2068 public domain software assemblage on tape. ($9.00 to C.A.T.S., PO Box 467 Fairfax, VA 22039). Representing the Western USA was Tim and Stephanie Woods (Time Designs Magazine), and the Central USA was represented by the staff of SyncWare News and attendees from the Michigan user group.

The rains came but the spirit of the fest was not dampened. Then Sunday afternoon the Sun came out for those who wanted to visit Disney World and the other attractions. A lot of merchandise was sold by the vendors. A+ Computers came prepared to sell out their stock of QL's and began selling the QL for $75.00 (including software!). Then by Saturday afternoon the price went up to $89.00 in order to keep from being sold out too early. Sharps Inc. had the table next to A+ and was doing a land office business in QL accessories and books. The strangest phenonemon was to see a person buy a QL for $75.00 and then turn to Sharps to buy a Trump Card for $300.00! It just "sounds" strange as the Trump contains more than the equilivant of $800.00 of IBM XT enhancements, including up-grade to 896K of memory.

The TS-2068 tables were also doing brisk business and almost everything sold out. Displays were set up to demonstrate all of the TS-2068 Disk Drive systems and Larken's new 256K Ram Disk Cartridge. Eric Johnson sold a bunch of new TS-2068 computers obtained from the Timex repair facillity. He also sold quite a few that had faults of one kind or another. All of the TS-2068's did not sell out, so if you need one, contact Eric at the address given later in this article. Available are TS-2068's that have been checked out good and some that have bugs. I believe that the prices are about $75.00 for the O.K. ones and $35.00 for the ones that have flaws.

Bill Pederson showed his (prototype) 20 Mbt EXTRA MEMORY Bank Switching RAM for the TS-2068 (thats 20 mega bytes!). Hopefully sufficient interest will develop that Bill will develop the system and offer it to the users. (William Pederson, 1120 Merrifield S.E., Grand Rapids, MI 49507).

TIMEX SINCLAIR GROUP EXPANDING! The complexion of the Timex Sinclair user group has changed during the past year. We have had an influx of 5-8000 new Sinclair QL users, while keeping constant numbers of TS-2068 users. Of course the TS-1000 and 1500 users are a solid group who mostly have complete software libraries. Interest remains strong in the TS-2068 due largely to the availability of three new disk drive systems for the TS-2068 and some new and more capable software. (See reviews of three TS-2068 Disk Drive systems in the January 88 issue of TS-2068 UP-DATE Magazine).

THE SINCLAIR Z88: Hold your hats! Here comes the neatest little PORTABLE computer that the world has seen! One was available at the Time Designs table to stroke and examine. Forget any comparison with other Sinclair products. This is New Concepts. The Z88 is a TRUE PORTABLE requiring no extra devices or dangling wires. It operates for 35 hours on enclosed AA Cell Batteries, holds memory data non volatile practically forever, has a pop up LED display, and Built in software. When the switch is turned on a pop-up Menu appears for the built in software ensemble. The Z88 has a built in EPROM programmer and one can pop in a small cartridge to down load the existing program to EPROM. Then the EPROM cartridge can later be plugged in and operated.

The Sinclair Z88 is a business person's delight and a sales person's dream. Also, I believe that many personal users will want this neat little computer. The dealerships in the USA are being organized and a few dealers have the Z88. Once the Z88 hits the USA market in full force, I believe it will capture a large chunk of the market for PORTABLE Lap Top computers. The Sinclair Z88 will fly! The price of the Z88 is about $479.00. It's nearest competitor runs about twice that. My opinion is that we will see the Z88 price lowered to just under $400.00 as the year progresses, which will cause intolerable pain to competing Portables and lap tops, which are grossly overpriced. There is room in the USA market for the Sinclair Z88.

With 5-8000 new QL users, the continuing TS-2068 "die hard" users who are now upgrading to Disk Drive, and the (just now) release of the Sinclair Z88, the Timex Sinclair user group is actually expanding. The convertability of syntax between the TS-2068 and the older TS-1000's helps to sustain the durability of the TS-1000 group. While some suppliers have abandoned the Timex Sinclair users others have increased their stock and intensified their efforts. As new Z88 and QL dealers come in the support picture should improve.

The TS-2068 users continue to have excellent cottage industry support and new software is constantly being produced. The clubs are busy in the conversion of public domain software to disk drive application and expanding the software functions of these abbreviated programs. Interestingly, about 1/4 of the "die hard" TS-2068 users also have a QL. Some have said that they are building their QL system in affordable stages while keeping the TS-2068 as their primary system.

"THE NORTH AMERICAN TIMEX SINCLAIR ASSOCIATION": The Florida User Groups, who organized Winter Fest, came to Orlando with a plan to organize a National Timex Sinclair Association. Quickly "National" changed to "North American" a more appropriate name which includes the Canadian user groups. In particular the Ontario and Vancouver user groups are very active and strong. A North American Association will continue (in name) the friendly, cooperative, and common interest "existing situation". The idea of a North American Association imediately took hold. A working group was established. Both Time Designs Magazine and TS-2068 UP-DATE Magazine volenteered to support the association with editorials and page space for communications.

The success of the North American Association is almost guaranteed from the start. Eventually as the Association is formed a new Association publication is likely to be started. The names of the other coordinating officials will be announced. But as of now, Eric and Mary Lynn Johnson, 249 N. Harden Ave., Orange City, FL 32763, are the key persons. The two had major parts in organizing the Orlando Winter Fest. TS-2068 UP-DATE Magazine highly recommends that ALL Timex Sinclair users join the new North American Timex Sinclair Assocation and participate in this venture that will serve to promote the interests of the users, the Clubs, the suppliers, and provide information exchange.

Eric Johnson is the SYSOP of the BBS which was set up as the communications center for the Fest. This BBS will be continued and will become the communications center for information, suggestions, and volenteers for work in the organization of the North American Timex Sinclair Association. The BBS telephone number is: 904 775 0093 (300 baud, 8, 1, none). Lately this BBS has been sabotaged by a pestering hacker, but will be back on line WHQ (when he quits). Keep on trying. Whether you can reach the BBS or not, your interest cards to the persons given will be a help.

IN SUMMARY: The Orlando Winter Fest was a success. Timex Sinclair Computing, instead of waning, is actually on a sharp up-swing. There are several thousand new QL users and another Sinclair Z88 user group is about to come into the fold. The Z88 group has the prospect of becomming a huge user group. New disk drive hardware and software for the TS-2068 are now available, which adds to the "staying power" of this viable and enthusiastic group. The new Association will bring more communications between Clubs, the users at large, and the supporting industries. With the enthusiasm being exhibited for the QL by the users, Amstrad may be enticed to resume production of the Sinclair QL. They will miss a great opportunity if they dont.

System Variables and Number Storage

Appendix D of the TS-2068 user manual gives four pages of "System Variables", which most users kinda ignore. These variables are used by the TS-2068 operating system to control things, and we users can use the vars to advantage. The most confusing thing about the tables is the "two byte numbers", which leaves us out in the cold climate of mis-comprehension. So, let us learn how to store a number greater than 255. A "8 bit" computer places a limit upon "how large a number" it can manage "in just one byte (memory address)" in memory. You can "LET a=999999" and the six digit number will STORE in the variable area. But, internally in the computer, that takes several bytes of memory. The minimum amount of memory required to store a number larger than 255 is two bytes, and is stored by POKE(ing) a smaller number to two consecutive addresses. EXAMPLE: POKE 60000,1 and POKE 60001,1. Now let us see what that means, in so far as "what number LARGER than 255" has been stored. The computer formula for finding out would be, PRINT PEEK 60000 + 256 * PEEK 60001. Try it. After some practice POKING two consecutive MEMORY ADDRESSES, and then PEEKING them back with the formula, you will find that resultant of each increase in a number poked to the HIGH address will be 256 digits higher, and the resultant of each increase in a number poked to the LOW address will be ONE digit higher.

The actual reason that a 8 bit computer can only STORE a number SMALLER than 256 in ONE memory address is that there are only 16 "address lines" to the Central Processing Unit (CPU). What is the MAXIMUM factor of 16? You are correct if you said "256". "But, you said that the computer is limited to the maximum number 255 in one memory address"?? That is correct, because 255 actually represents 256 digits, INCLUDING ZERO. Count on your pinkies. One finger is 1, but a closed fist is 0. In computing, zero has as much significance as 1000. Elementary, eh Watson? Now what is the HIGHEST number that can be stored in TWO memory addresses? Find out by POKING 255 to two addresses and using the formula. Never did get to them SYSTEM VARIABLES, did we! We will deal with them in the programming section.

Whither Goest, TS-2068 UP-DATE?

This April issue of UP-DATE marks the third of a four issue year, and a time for a decision for the second year. I am having fun publishing UP-DATE and learning a bit with each issue. But it is a lot of work! I hope that you like what is being done because it has cost me about $160.00 more per issue than gross income. The fun part isn't quite worth the $53.00 per month loss which could be buying a new Z88. So, I'll have to quit (nasty word) or increase the price of UP-DATE subscription to about $15.00 per year to break even. I thought about cutting the page count to about 16 pages, but I dont like little newsletter size publications. Do you? I had much rather INCREASE page count than to decrease the content. By the way, I just subscribed to Quantum Levels, a smaller magazine with a higher subscription cost. It will be worth it.

So, UP-DATE will continue with a increase in subscription price to $15.00. I hope that you dont begrudge me a break even. I'll try to add enough improvements to make up for the extra three bucks. UP-DATE will concentrate upon broadening the coverage of these topics: Public domain software- I have begun to assemble a library of PD programs and will cull out the dogs and publish only the programs that have been checked out, with edited changes for disk syntax (lotsa work!). We need to have more information about ROM ROUTINES that can be "called from BASIC". ROM BUGS in the bank switching functions must be explored. Telecomputing needs further detail. I am looking for a program to use "between persons" so that you can call someone else and the two of you begin exchange of data and programs without having to go through a BBS.

Other topics to explore are: Building projects. Right now we need a BUSS extension that has a 5V power supply to power external devices. Many of us are loading down the 5V supply with too many add-ons. Other building projects are needed. Then "interfacing" is a broad subject that has not been the subject of much writing. How about the idea of using your old TS-1000's as slave devices to do processing while you are doing something else with the TS-2068? More information needs to be published about how to program EPROMS to use in Dock Bank cartridges. And, of course, we must continue to cover all of our disk systems and bring on more DISK software. Patching up old cassette programs is like bear hunting with a BB gun. UP-DATE invites writers to assist in these areas.

QL Support

An analysis of TS-2068 users reveal that about 1/4 already own a Sinclair QL, and more will purchase the QL if the supply lasts. UP-DATE will not dilute the support of TS-2068 users at all, but will begin to cover the QL in the October 88 issue. Actually more than 1/4 of the subscribers have both TS-2068 and QL computers and are using the more familiar TS-2068 as their primary system as their QL learning curve progresses and as they build up their QL systems. Coverage of the QL will be secondary and may either stay that way or increase as the subscriber base demands. While no date is set, sometime in the future UP-DATE will look at the numbers and perhaps begin covering the Sinclair Z88. If the Z88 is as hot as I think it may be, and the price comes down under $400.00, many of us will be getting one and will want publication support.

Even before starting UP-DATE I was always interested in reading about the Spectrum and the QL, though I did not own either. Now I want to know more about the Z88 and how it is accepted. I believe that most UP-DATE readers will be interested in such articles. Who knows when the Zapper will get your TS-2068 and force you to change? We wont quit (nasty word) then, will we? Anticipating such time your scribe bought a QL at the Orlando Fest. My wife thinks I'm crazy, and with good reason as I have 4 TS-1000's, 3 TS-2068's, a TS-1500, a C-128 (ugh), and an old Heath. Maybe it's my background from an old manufacturing family that made machine screws for over a century. There were always too many loose screws and a few nuts in the family. Just imagine someone trying to settle the estate of a machine screw and orphan computer collector!

Speaking of the QL, Bill Fisher, 419 Parkwood Drive E, Orange Park, FL 32073 is about to start a new QL publication called "QL USA". So, UP-DATE's coverage of the QL will be laid back, serious stuff and tips for programming from our experienced QL users. Programs for the QL on disk will be offered from time to time (Issue Disks), which brings up a point. TS-2068 issue disks have been selling pretty good, but to the same group. It seems that once a person buys one he is hooked and wants the next disk. Each disk contains at least one long program and several utilities. Normally the one long program would sell for about $19.95. For $16.00 the issue disk is a bargain. AND, if anyone isn't satisfied--money back! "Mail Merge" (Oct Issue Disk) is the best program of it's type that I've used. "DOSDEX" (Jan issue) is even better.

FD-68 and LARKEN SUPPORT

I apoligize to the FD-68 and Larken Disk users for being broken down in support. My TS-2068 + FD-68 + LKDOS Cartridge, ALL zapped together! Dan Elliott fixed the 68, but the other peices took a slow boat to China and haven't been returned. I WILL get back into FD-68 and Larken disk support even if I have to buy new controllers. Now WHY did all of that break at the same time? Dan Elliott said that the voltage reg fried, placing 6+ volts throghout the Vcc line. Several memory chips fried. This makes me believe that there was too much tagged on to the external bus. In addition, I had the Parallel printer intfc, and the ZSI/O RS-232 interface on there. From now on I will try to keep only the essential items on the rear deck port. Dan's address is: Dan Elliott, RT-1, Box 117, Cabool, MO 65689. $15.00 plus parts for TS-2068.

Help me Format this Magazine!

Now it is YOUR decision time. I am fiddling with the print format of UP-DATE to try to find a more readable arrangement. Please compare the first type font and size with this page. The older presentation was 15 point type font with compressed spacing between characters, 52 characters per line and 56 lines per page. A page provided 5824 characters. This format is about 16 point type, 47 characters per line, 78 lines per page, with normal spacing between characters and between

lines, but with smaller type. This combination produces 7332 characters per page, 1508 more than the other format, or a 26% increase in page data. I'll mix up the format of this issue to let you decide between the styles.

I just got my glasses up-dated and I like this style better because the characters in the words are better spaced. But some of you may be straining to focus in the smaller print. If UP-DATE is printed in this style, 30 pages will contain about the amount of data in 38 pages of the other style. If we go to this style, listings will be in larger print and 32 chr lines. The way this new style is achieved is; I use 10 pt Pica on a legal size page and then reduce it to 76% for the masters. What I'm trying to do is to increase the content of UP-DATE and stay within 30 pages, and improve readability at the same time. You are invited to drop Up-Date a card and give your preference. Elaborate if you wish,or just say "New Style or OLD Style".

Keeping UP with Sir Cleave

Sir Cleave, that intrepreneur designer of computers, wrist watch TV's, steam powered aircraft, and hair growing lotion, recently spent five hours in a hospital in Spain where he personally directed the world's first implant of supplementary memory into a human brain, his own. He directed the precise operation while viewing the area of surgery via prisms and mirrors. 256 gbts (giga bytes) of non volatile protein memory cells were implanted along with a enzyme that he invented for bonding artificial protein cells to the natural cells of the brain. Departing from his normal habits, Sir Cleave's new system does not require additions to its I/O port to operate. Upon emerging from surgery, Sir Cleave struck up a conversation in Cantonese with a visiting intern from China.

Later Sir Cleave said that all of the World's languages and dialects were pre-programmed before implant. But he admitted to having tongue flutter when practicing changing from the dialect of the New Guinea natives to that of one Eskimo tribe of northern Manitoba. He said that this particular Eskimo dialect, to be spoken correctly, requires the speaker to be on the verge of shivvering, and the tempertures in Spain is not that cold. Never the less, it appears that periphreal devices will be needed after all to correct some ROM bugs already found. The trouble is, Sir Cleave has locked himself into a strange situation (for him). He cannot easily abandon this project. (April Fool!)

TS-2068 Publication Support

There are some excellent small publications that are worthy of a look. Send $2.00 to any of these to get a sample issue. Then you can decide whether or not to subscribe. QZX Magazine is oriented toward Amateur radio operators who use Sinclair computers (2025 O'Donnell Drive, Las Cruces, NM 88001). CATS Newsletter, published by the CATS group, is a good club newsletter (PO Box 467, Fairfax Station, VA 22039). QL USA is a new QL Newsletter (419 Parkwood Dr. E., Orange Park, FL 32073). D-FW Data Expansion is a good newsletter published by the Dallas / Fort Worth user group (4424 Geddes Ave, Fort Worth, TX 76107). ZX Appeal, a newsletter published by the Vancouver Club (2006 Highview Place, Port Moody, BC, Canada V3H 1N5).

SINCUS NEWS, published by the New York group (1229 Rhodes Rd., Johnson City, NY 13790). DATSN is a newsletter published by the Detroit user group (PO Box 614, Warren MI 48090). The Plotter, is published by the CCAT/S group in Oregon (1419 1/2 Street, Oregon City, OR 97045). TIMELINEZ, by the San Francisco area group (6615 Clifford Drive, Cupertino, CA 95014). SMUG BYTES is by the Milwaukee group (SMUG, PO Box 101, Butler, WI 53004). LISTing is the Long Island New York group newsletter (5 Peri Lane, Valley Stream, NY 11581). FDD Newsletter, 43307 Chambers Road, Horseheads, NY 14845, is Aerco FD-68 and CP/M orientatd, by Ron Havlen. SINC LINK is a good newsletter by the Ontario User Group, PO Box 7274, Stn. A, Toronto, Canada M5W 1X9.

TS-2068 Cottage Software Support

Fairware, Jack Dohany, 390 Rutherford Ave., Redwood City, CA 94061. Jack is a subscriber. Jack's cottage software business, has a catalog of various useful software titles. To get the catalog send $1.00 to Jack. Now WAIT! Should a catalog be free? It costs to answer correspondance and send free stuffin and our suppliers are operating on a low budget. You'll save the buck on your first order. Other software suppliers are: Chia-Chi Chao, 73 Sullivan Drive, Morago, CA 94556: RMG Enterpizes, 1419 1/2 7th St., Oregon City, OR 97045: S & K Enterprizes, 2107 SE 155th, Portland, OR 97223.

Some larger Cottage industries supporting the TS-2068 are: Lempke Software 2144 White Oak, Wichita, KS 67207: BYTE POWER, 1748 Meadowview Ave, Pickering, Ontario L1V 3G8: John Mathewson, 1852 Appleford St., Gloucester, Ontario K1J-6T8. One last mention for this issue: Grey & Clifford, PO Box 2186, Inglewood, CA 90305 kinda specializes in Telecom hardware and software. I killed my wallet with them, buying a Modem, Specterm 64, ZSI/O, and I'm just now getting well. UP-DATE endorses all of these suppliers as honest dealers who stand behind everything they sell. All of the larger dealers will keep you on their mailing list if you order something each year. But you will be glad if you try the smaller ones also.

I had an interesting talk at the Fest with Bill Pederson, the engineer who designed the electronics flight control system for the F4 Phantom aircraft. Bill probably has the most knowledge of any person about the SYSCON managemnt area of the TS-2068, which is the Memory Bank Switching routines. I have invited Bill to do a complete series of articles for UP-DATE about using extra memory, and to include some building projects. One of his prototype boards contains the extra 5V power supply (mentioned earlier) to power external devices. And he has external devices! How about 20 megabytes of extra RAM, and a re-designed control system for the extra memory! I'd settle for less, but better management without ROM bugs. See you in the pages.

BUDGET
BY: Bob Mitchell, Willodale, Ontario
Provided to TS-2068 UP-DATE Magazine
(Revision dated 880229)

Editor's Note: Bob Mitchell is retired from two professions, a career in the Canadian Armed Forces, and a subsequent career in Geodetic Survey. Presently Bob lives in beautiful Ontario Canada and "Snow Birds it" to Florida during the Winter months. Bob has worked on and purified this major software over a period of about six years, and now gives us this outstanding software which represents countless hundreds of hours of work. It is a masterpiece.

I never anticipated publishing such a long key in project in the pages of UP-DATE, and I would not except for these strong factors: 1. Though many, the program lines are short and easy to key in. 2. "Budget" rates a 10 in the category of usefulness. 3. Budget rates another 10 for educational value in two areas: a. Basic Programming techniques. b. It is a excellent tool for learning to compile basic programs. Finally, 4. Budget is easily used with all disk drives systems, as well as cassette. The program itself has great possibilities for re-dedication to other applications besides its purpose as a Budget software. The spreadsheat data is presented in excellent format and the subject categories can be changed to track other topic areas. This BUDGET software is a real barn burner worth the hours it will take to key in the progam lines. And there are great follow-on things to come.

Because the program listing is long, UP-DATE provides a short appendix at the end of the text to help with the key-in project. Take your time and do the key in over a time period of at least a week, saving the partially completed program often. It will be fun and relaxing that way. While keying in, ponder the program syntax and try using direct commands to see what the syntax actually does. UP-DATE will take questions and publish explanations in the issues to come. If the key in seems tedious, just remeber, Bob has worked on this sofware off and on for six years, and we get it all in a neatly packaged gift. Now I'll turn you over to Bob to get you going.

This BUDGET program is in three parts. First the long program in BASIC is given. When the listing is keyed in the program will work right away by RUN. BUDGET is really a Spread Sheet software that has the purpose of planning a annual household budget and then tracking the expenditures, providing various comparisons, and displaying or printing the data. The information products may be printed out with the TS-2040 printer. Follow on parts of the program will be given to allow print out of the data products with Centronics printers.

The second part of the BUDGET program, to be given in the July issue of UP-DATE, is a customized Disk Drive "Loader/Manager". This Loader program is not essential for operating the Basic program given in this issue and thus will be given in a follow on article. The Loader program customizes the Compiled Basic program to operate with disk drive. There will be versions for Oliger Disk, Larken Disk, and syntax changes for Aerco FD-68, RAMEX, and TOS Disk. One more program addition will be given to provide the spreadsheet print out with wide carriage printers. The Basic program given in this issue prints a 32 column product with the TS-2040 printer and with Oliger disk via the LET/p=o feature.

This issue deals with the main software titled: "budget.BJ", which is programmed entirely in BASIC. But, there is a unique feature. The program is set up for COMPILING into Machine code by use of the Compiler program "Timechine". So, you will see the strange syntax "REM !" installed just after the line number of some program lines. This is the syntax needed by Timechine for transforming the Basic Program into a Compiled Machine Code program. If you've never done this, dont worry about it because the program will run without compiling it, just slower. Then after you have the program running will be the time to begin the adventure of Compiling the Basic Program. So, go ahead and type in the long listing. Save the program often to either Cassette or disk. When finished, then pick up the operational procedures given next.

PROGRAM OPERATIONS

1. Type RUN to begin operations. When the BUDGET menu appears in a sort of pull-down menu format, follow the instructions on the screen and SET DATE (in international format: YYMMDD) and then SET BUDGET YEAR (just YY).

2. Try entering some amounts into the program. Move the cursor to "ENTER PLANNED" and press ENTER. Follow the prompts on the screen and enter some dummy planned data (ie, how much you plan to spend during the 12 months for each of 18 categories. When you complete this dummy run of the annual budget, it will be time to see the results in various formats.

3. In sequence, press: DISPLAY SUB MENU > PLANNED > JANUARY. (In this tutorial, sequential key presses are separated by a ">".) Prompts at the bottom of the PLANNED BUDGET FOR JAN-APR gives three choices:

a. Continue with Display Mode.

b. GO TO the Main Menu.

c. Copy the Spreadsheet to the 2040 printer. If your printer is not on the program will detect this and tell you to turn it on, then the COPY routine will work.

4. Try a few commands then go to the Main Menu. If there is an error, such as "number too big" (you wont see this report in the compiled program); you will be back in the loader. (Note: Since we do not yet have the LOADER program, type GO TO 1470 when any stoppage occurs).

Main Program Features

5. Let's go over some of the main features of this program. First, the line of BASIC that you will want to customize is 590. Edit this line down to the bottom of the screen and change it manually according to the folowing criteria.

6. There are 18 categories and you should set the names carefully to meet your needs over several years. (There is space for up to 10 letters in each of the category names.) If you change the category names from year to year, you will find it difficult if not impossible to make meaningful comparisons for budget planning.

7. Try entering some "actuals", ie, expenditures. Set the exchange rate if you are dealing with foreign currency. Just enter the proper rate. Enter "1" for local currency. Press JANUARY > GROCERIES and a "worksheet" appears to allow you to enter all of your bills. You can concatenate several entries in each category by entering them at the same time (eg, 134.35 + 27.09 + 48.76). Press "Y" for more entries until you have all of your bills for JANUARY entered and then press "N". The totaling then takes place and is slow in comparison with the compiled version.

8. Now inspect what you have done via the DISPLAY MENU. Notice that all data are right justified and all are rounded to the nearest whole number, no room for pennies! (The right justification routine slows the BASIC version down and is one reason for providing the set up for compiling.). Your data will be stored in decimal (Floating Point) notation even though it is rounded to the nearest whole number for screen display and for printing with the TS-2040 printer. Storage in floating point is essential so that the calculations performed by the program will produce accurate results, totals, etc. Rounded whole numbers are acceptable for budget display, planning, and tracking.

9. The ADD-ON programs will be given in the July Issue of UP-DATE. One addition will retrieve the real values and print them on a wide printer in full decimal format. Another will be the Loader which is the Disk Manager. I'll also provide a more detailed explanation of the BASIC program with emphasis on the parts with useful sub routines. Also there will be a HELP routine which you can put on disk and call up from an expanded loader: this "HELP" will give suggestions on allocating different types of expenditures to the 18 categories.

10. Try COMPOSITE for January Actuals. You will get a YEAR-TO-DATE (YTD) comparison with variances and planned Year-End Totals. The

variances show the differences between the corresponding planned and actual values. If you have spent more than your budgeted amount, the variances will be negative values, and these will be shown in flashing figures (in RED on a colour screen).

11. Take a look at CATEGORIES. Then try GRAPHS > BAR GRAPH > ACTUALS > JANUARY >. Try the table option and you will get a YTD percent table showing the percentage for each category against the total. The percentage total will seldom add up to exactly 100% (due to rounding).

12. Get back to GRAPHICS via the MENU and try the pie chart. Sometimes, depending upon how close the pie segments are, the category codes (A-R) will overwrite one another on the pie chart. I left this that way so that I could keep the pie chart as large as possible. Another view of the data is shown when you call for BAR charts.

13. Now BREAK and make a direct SAVE of the BASIC program to Disk or Cassette. SAVE with a starting line of 1470. The program, when re-loaded will present the main menu and the previously constructed data will be intact for either review or for entry of new data to continue planning and tracking of the annual budget.

Coming Additional Features of BUDGET

Next issue will bring the ancillary programs previously mentioned, plus the procedure for COMPILING the Basic program using the TIMECHINE Compiler program. This application of Timechine represents a case of putting the compiler through its paces. If you dont yet have the "Up-Dated version of TIMECHINE, it is available from several sources including: NOVELSOFT, 106 Seventh St., Toronto, Ontario, Canada M8V 3B4 ($19.95 US plus $3.00 S&H).. Be sure to specify "The AMENDED VERSION that handles D and E routines". This excellent compiler allows you to compile almost any slower Basic program into compiled machine code to greatly improve operating speed.

Editors Note: To get the advantage of having a faster operating program that has been compiled with Timechine, you have two options: 1. Buy the Timechine software and use it to compile this program. This is the "educational way". 2. The April UP-DATE Issue Disk will contain ALL programs mentioned (The BASIC un-compiled program, plus the compiled program, plus the additions, "Loader and Centronics Printer annex". See the yellow pages for ordering.

Bob Mitchell, Willodale, Ontario

UP-DATE Appendix to BUDGET

There are several program lines that start with < REM ! >. This is the syntax for the Timechine Compiler program which does not interfere when you run the program in its BASIC form. DO enter these codes whether you intend to compile the program or not. This is so that the TYPE IN CHECK POINTS given next will be correct. To enter the REM ! syntax, "first omit the REM !" and type the rest of the line of programming. Then back up the cursor to the line number and type REM !.

KEY IN Accuracy Check

Accuracy check points are given as follows: After keying in the lines given in the left column, type CLEAR, then PRINT FREE. If your typing is correct the figure in the right column will be presented on screen. If the screen figure and the number given in the column do not match, then there are type in errors.

LINE FREE	LINE FREE	LINE FREE	LINE FREE
100-38385	1500-33346	2900-29260	4300-25601
200-38163	1600-32926	3000-29081	4400-25305
300-37896	1700-32621	3100-28820	4500-25025
400-37601	1800-32353	3200-28612	4600-24434
500-37328	1900-32145	3300-28415	4700-24150
600-36641	2000-31840	3400-38086	4800-23616
700-35988	2100-31620	3500-27836	4900-23132
800-35362	2200-31316	3600-27460	5000-22820
900-34823	2300-31133	3700-27264	5100-22327
1000-34580	2400-30672	3800-27039	5200-21854
1100-34622	2500-30369	3900-26723	5300-21343
1200-34041	2600-30124	4000-26322	5400-21143
1300-33821	2700-29793	4100-26051	
1400-33625	2800-29574	4200-25773	

Last line-5420=21093

"budget.BJ" Bob Mitchell

```
  10 REM **Change DATA line 590
         --Category Names--
        to suit your own
        needs**
  20
  30 REM !USR 32000
  40 REM !INT +g,cf,cf2,gf,sf,d,
n,l,lo,p,cr,c,st,x1,y1,r1,gt,gc,
j,pe,py,px,k,e,o,o1,l
  50 REM ! LIST
  60 REM ! LPRINT
  70 REM !LEN $<=15
  80 REM ! OPEN #
  90 DIM c$(32)
 100 GO TO 420
 110 IF s$(1)="." THEN LET s$="0
"+s$
 120 FOR j=1 TO LEN s$
 130 IF s$(j)="." THEN GO TO 160
 140 NEXT j
 150 LET s$=s$+"."
 160 IF s$(LEN s$-1)="." OR s$(L
EN s$)="." THEN LET s$=s$+"0"
 170 IF s$(LEN s$-1)="." THEN LE
T s$=s$+"0"
 180 RETURN
 190 STOP
 200 LET o=1: LET o1=0
 210 PLOT px*8-o,176-((py+1)*8)
 220 DRAW 0,-pe*8-o
 230 DRAW pw*8+o*2,0
 240 DRAW 0,pe*8+o*2
 250 DRAW -pw*8-o*2,0
 260 DRAW 0,-o
 270 IF o1 THEN RETURN
 280 LET o=3: LET o1=1: GO TO 21
0
 290 STOP
 300 READ pe,py,px,k
 310 LET pw=0: FOR i=1 TO pe: RE
AD e$: LET e=LEN e$: IF pw<e THE
N LET pw=LEN e$
 320 PRINT AT py+i,px;e$: NEXT i
 330 PRINT AT py+1,px; PAPER 2;
OVER 1;c$( TO pw)
 340 PRINT AT k,px; OVER 1; PAPE
R 4; INK 7;c$( TO pw)
 350 GO SUB 200
 360 LET k$=INKEY$
 370 IF k$=CHR$ 10 THEN GO SUB 6
50
 380 IF k$=CHR$ 11 THEN GO SUB 7
00
 390 IF k$=CHR$ 13 THEN RETURN
 400 PAUSE 30: GO TO 360
 410 STOP
 420 DIM y(18,12): DIM a(18,12):
DIM t(18,12): DIM m(18,12): DIM
e(18,12): DIM u(18,12)
 430 LET exch=1: LET sf=0
 440 DIM m$(12,3)
 450 DIM b$(18,10)
 460 DIM t$(3,11)
 470 LET t$(1)="PLANNED"
 480 LET t$(2)="ACTUALS"
 490 LET t$(3)="COMPOSITE"
 500 LET n$="BUDGET"
 510 LET x$="": LET y$="    "
 520 LET t3=0
 530 LET m$(1)="JAN": LET m$(2)=
"FEB": LET m$(3)="MAR": LET m$(4
)="APR"
 540 LET m$(5)="MAY": LET m$(6)=
"JUN": LET m$(7)="JUL": LET m$(8
)="AUG"
 550 LET m$(9)="SEP": LET m$(10)
="OCT": LET m$(11)="NOV": LET m$
(12)="DEC"
 560 RESTORE 590: FOR i=1 TO 18:
 READ o$: LET b$(i)=o$: NEXT i
 570 GO TO 1470
 580 DATA INT 20,0,7,3,"CATEGORI
ES"," "
 590 DATA "CAPITAL","CLOTHING","
DENTAL","ELECTRIC","GAS","GIFTS"
,"GROCERIES","HOUSING","INSURANC
E","LEISURE","LIQUOR","MEDICAL",
"MISC.","PERSONAL","PHONE","PROP
.TAXES","TRANSPORT","VACATION"
 600 RETURN
 610 DATA INT 14,3,9,6,"MONTHS",
" ","JANUARY","FEBRUARY","MARCH"
,"APRIL","MAY","JUNE","JULY","AU
GUST","SEPTEMBER","OCTOBER","NOV
EMBER","DECEMBER"
 620 RETURN
 630 DATA INT 8,5,7,8,"BUDGET MA
IN MENU"," ","ENTER ACTUALS","EN
TER PLANNED","SET DATE","DISPLAY
 SUB-MENU","SET BUDGET YEAR","QU
IT"
 640 STOP
 650 LET k=k+1: IF k<=py+pe THEN
 PRINT AT k,px; PAPER 4; INK 7;
OVER 1;c$( TO pw)
 660 PRINT AT k-1,px; PAPER 0; I
NK 7; OVER 1;c$( TO pw)
 670 IF k=py+pe+1 THEN LET k1=k:
 LET k=py+3: PRINT AT k1,px; PAP
ER 0; INK 7; OVER 1;c$( TO pw)
 680>PRINT AT k,px; OVER 1; PAPE
R 4; INK 7;c$( TO pw)
 690 RETURN
 700 LET k=k-1: IF k>=py+3 THEN
PRINT AT k,px; PAPER 4; INK 7; O
VER 1;c$( TO pw)
 710 PRINT AT k+1,px; PAPER 0; I
NK 7; OVER 1;c$( TO pw)
 720 IF k=py+2 THEN LET k1=k: LE
T k=py+pe: PRINT AT k1,px; PAPER
 0; INK 7; OVER 1;c$( TO pw)
 730 PRINT AT k,px; OVER 1; PAPE
R 4; INK 7;c$( TO pw)
 740 RETURN
 750 FOR i=1 TO 12: LET s=0: FOR
 d=1 TO 18: LET s=s+a(d,i): LET
m(d,i)=s: NEXT d: NEXT i: RETURN

 760 FOR d=1 TO 18: LET s=0: FOR
 i=1 TO 12: LET s=s+a(d,i): LET
t(d,i)=s: NEXT i: NEXT d: RETURN

 770 LET s=0: FOR i=1 TO 12: LET
 s=s+a(d,i): LET m(d,i)=s: NEXT
i: RETURN
 780 LET s=0: FOR i=1 TO 12: LET
 s=s+a(d,i): LET t(d,i)=s: NEXT
i
 790 LET s=0: FOR i=1 TO 18: LET
 s=s+t(i,12): LET sf=s: NEXT i:
RETURN
 800 FOR i=1 TO 12: LET s=0: FOR
 d=1 TO 18: LET s=s+e(d,i): LET
y(d,i)=s: NEXT d: NEXT i: RETURN

 810 LET s=0: FOR i=1 TO 12: LET
 s=s+e(d,i): LET y(d,i)=s: NEXT
i: RETURN
 820 LET s=0: FOR i=1 TO 12: LET
 s=s+e(d,i): LET u(d,i)=s: NEXT
i: RETURN
 830 FOR d=1 TO 18: LET s=0: FOR
 i=1 TO 12: LET s=s+e(d,i): LET
u(d,i)=s: NEXT i: NEXT d: RETURN

 840 LET s=0: FOR i=1 TO n: LET
s=s+m(18,i): LET sb=s: NEXT i: L
ET s$=STR$ sb: RETURN
 850 LET s=0: FOR i=1 TO n: LET
s=s+y(18,i): LET sa=s: NEXT i: L
ET s$=STR$ sa: RETURN
 860 LET s=0: FOR i=1 TO n: LET
s=s+y(18,i): LET sa2=s: NEXT i:
RETURN
 870 LET s=0: FOR i=1 TO n: LET
s=s+m(18,i): LET sa3=s: NEXT i:
```

```
 880 GO TO 1470
 890 RETURN
 900 DIM k(18): FOR i=1 TO 18: L
ET k(i)=t(i,n)/t3: NEXT i: RETUR
N
 910 DIM k(18): FOR i=1 TO 18: L
ET k(i)=u(i,n)/t3: NEXT i: RETUR
N
 920 LET s$=STR$ ss: RETURN
 930 LET s$=STR$ sss: RETURN
 940 LET s$=STR$ pc: RETURN
 950 IF gf THEN RETURN
 960 LET cr=cr+1
 970 IF st=16 THEN GO SUB 1300
 980 IF st=0 THEN GO SUB 1140
 990 IF st=2 THEN GO SUB 1160
1000 IF st=4 THEN GO SUB 1180
1010 IF st=6 THEN GO SUB 1200
1020 IF st=8 THEN GO SUB 1220
1030 IF st=10 THEN GO SUB 1240
1040 IF st=12 THEN GO SUB 1260
1050 IF st=14 THEN GO SUB 1280
1060 IF ss<0 THEN PRINT TAB (c-L
EN s$+1); FLASH 1; PAPER 2;s$;
1070 IF ss>=0 THEN PRINT TAB (c-
LEN s$+1);s$;
1080 IF lo THEN RETURN
1090 LET c=c+5
1100 IF st=4 OR st=10 OR st=12 O
R st=14 THEN RETURN
1110 LET n=n+1
1120 IF cr=4 THEN LET n=p: RETUR
N
1130 GO TO 950
1140 LET ss=INT (a(d,n)+.5)
1150 GO SUB 920: RETURN
1160 LET ss=INT (m(18,n)+.5)
1170 GO SUB 920: RETURN
1180 LET ss=INT (t(d,n)+.5)
1190 GO SUB 920: RETURN
1200 LET ss=INT (e(d,n)+.5)
1210 GO SUB 920: RETURN
1220 LET ss=INT (y(18,n)+.5)
1230 GO SUB 920: RETURN
1240 LET ss=INT (u(d,n)+.5)
1250 GO SUB 920: RETURN
1260 LET ss=INT (t(d,n)-u(d,n)+.
5)
1270 GO SUB 920: RETURN
1280 LET ss=INT (t(d,12)+.5)
1290 GO SUB 920: RETURN
1300 RETURN
1310 GO TO 1470
1320 LET cf2=1
1330 RESTORE 580
1340 GO SUB 300
1350 GO SUB 580
1360 LET d=k-2
1370 CLS : GO SUB 300
1380 GO SUB 610
1390 LET n=k-5
1400 INPUT ("enter value for ";B
$(D)'"for ";m$(n)),a(d,n)
1410 INPUT "another change ? y/n
  "; LINE q$
1420 IF q$<>"Y" AND q$<>"N" THEN
 GO TO 1410
1430 IF q$="Y" THEN GO SUB 1770:
 GO TO 1320
1440 IF q$="N" THEN GO SUB 1770:
 GO SUB 1850: GO TO 1470
1450 STOP
1460 REM ! OPEN #
1470 CLS
1480 BORDER 0: PAPER 0: INK 7: B
RIGHT 0: CLS
1490 POKE 23658,8
1500 LET cf=0: LET cf2=0: LET gf
=0
1510 PRINT AT 2,0; INVERSE 1; IN
K 2; PAPER 7;"DATE: ";x$;"   BUD
GET YEAR: 19";Y$(2 TO 3)
1520 PRINT AT 16,0; INVERSE 1; I
NK 3; PAPER 7;"USE PULL DOWN MEN
US AS FOLLOWS: MOVE BAR CURSOR U
P AND DOWN      USING CS 6/7 KEYS
 THEN [ENTER]. "
1530 LET pw=0
1540 RESTORE 630
1550 GO SUB 300
1560 LET k=k-7
1570 IF k=4 THEN GO TO 1920
1580 IF k=2 THEN GO TO 1640
1590 IF k=1 THEN GO TO 3370
1600 IF k=3 THEN GO TO 3630
1610 IF k=6 THEN STOP
1620 IF k=5 THEN GO TO 1880
1630 IF k=0 THEN GO TO 1470
1640 CLS
1650 RESTORE 1660
1660 DATA INT 4,16,6,19,"PLANNED
 DATA ENTRY"," ","Whole year","C
hange one entry"
1670 GO SUB 300
1680 LET k=k-18
1690 IF k=1 THEN CLS : GO TO 171
0
1700 IF k=2 THEN CLS : GO TO 132
0
1710 FOR d=1 TO 18
1720 FOR n=1 TO 12
1730 PRINT AT 20,0;"ENTER ";b$(d
),"Planned Budget FOR ";m$(n)
1740 INPUT a(d,n)
1750 CLS
1760 NEXT n
1770 PRINT FLASH 1;"Totaling"
1780 GO SUB 770: GO SUB 780
1790 PRINT AT 20,0;b$(d)'t$(1);"
 for ";y$;" is ";":$";t(d,12)
1800 IF cf2 THEN PAUSE 100: CLS
: RETURN
1810 PAUSE 100
1820 CLS
1830 IF d=19 THEN GO TO 1710
1840 NEXT d
1850 PRINT FLASH 1;"Grand Totali
ng": GO SUB 750: GO SUB 790
1860 IF cf2 THEN PAUSE 100: CLS
: RETURN
1870 GO TO 1470
1880 CLS
1890 INPUT "ENTER Budget Year (Y
Y)"' LINE y$
1900 LET y$=" "+y$+" "
1910 GO TO 1470
1920 CLS
1930 BORDER 0: PAPER 0: BRIGHT 0
: INK 7: CLS
1940 LET cf=0
1950 RESTORE 1970
1960 GO SUB 300
1970 DATA INT 8,5,7,8,"DISPLAY M
ENU"," ","MAIN MENU","PLANNED","
ACTUALS","COMPOSITE","CATEGORIES
","GRAPHS"
1980 LET k=k-7
1990 IF k=1 THEN GO TO 1470
2000 IF k=2 THEN GO TO 2050
2010 IF k=3 THEN GO TO 2740
2020 IF k=6 THEN GO TO 3800
2030 IF k=5 THEN GO TO 3010
2040 IF k=4 THEN GO TO 2410
2050 CLS
2060 PAPER 0: BORDER 0: INK 7: C
LS
2070 RESTORE 610
2080 GO SUB 300
2090 LET n=k-5
2100 IF n>9 THEN LET n=9
2110 LET lo=0
2120 CLS
2130 LET p=n
2140 PRINT AT 0,0;c$
2150 PRINT AT 0,0; INVERSE 1;y$;
t$(1); INVERSE 1;TAB 13;m$(n);TA
B 18;m$(n+1);TAB 23;m$(n+2);TAB
28;m$(n+3)
2160 FOR d=1 TO 18
2170 LET cr=0
```

```
2180 PRINT TAB 0; INVERSE 1;" ";
2190 PRINT TAB 2;b$(d);
2200 LET c=15
2210 LET st=0
2220 GO SUB 950
2230 IF d=19 THEN GO TO 2160
2240 NEXT d
2250 PRINT AT 19,0;"TOTALS  ";
2260 LET cr=0
2270 LET st=2
2280 LET c=15
2290 INVERSE 1
2300 GO SUB 950
2310 INVERSE 0
2320 PRINT #1;AT 0,0; PAPER 2;"E
NTER>"; PAPER 0;" 1=CONT    2=ME
NU   3=COPY"
2330 LET k=CODE INKEY$-48
2340 IF k=1 AND NOT cf THEN GO T
O 1920
2350 IF k=1 AND cf=1 THEN GO TO
1470
2360 IF k=1 AND cf=3 THEN GO TO
3400
2370 IF k=2 THEN GO TO 1470
2380 IF k=3 AND IN 251<>126 THEN
 INPUT #1: PAUSE 30: COPY : GO T
O 2320
2390 IF k=3 AND IN 251=126 THEN
PRINT #1;AT 0,0;"TURN ON PRINTER
": BEEP .2,30: PAUSE 30: GO TO 2
380
2400 GO TO 2320
2410 CLS
2420 PAPER 0: BORDER 0: INK 7: C
LS
2430 LET lo=0
2440 RESTORE 610
2450 GO SUB 300
2460 LET n=k-5
2470 CLS
2480 PRINT AT 0,0;m$(n);
2490 PRINT AT 0,13;"PLAN";TAB 18
;"ACT";TAB 23;"VAR";TAB 28;"PLAN
"
2500 PRINT t$(3);TAB 13;"YTD";TA
B 18;"YTD";TAB 23;" $ ";TAB 28;
INVERSE 1;y$
2510 FOR d=1 TO 18
2520 LET cr=0
2530 LET c=16
2540 PRINT TAB 0; INVERSE 1;" ";
2550 PRINT TAB 2;b$(d);
2560 LET st=4: GO SUB 950: LET s
t=10: GO SUB 950: LET st=12: GO
SUB 950
2570 LET st=14: GO SUB 950
2580 NEXT d
2590 GO SUB 840
2600 PRINT TAB 0;
2610 LET s$=STR$ INT (sb+.5)
2620 PRINT TAB 0;"TOTALS";TAB (1
6-LEN s$+1);s$;
2630 GO SUB 850
2640 LET s$=STR$ INT (sa+.5)
2650 PRINT TAB (21-LEN s$+1); IN
VERSE 1;s$;
2660 LET ss=(sb-sa)
2670 LET ss=INT (ss+.5): GO SUB
920
2680 IF ss>=0 THEN PRINT TAB (26
-LEN s$+1);s$;
2690 IF ss<0 THEN PRINT TAB (25-
LEN s$+1); FLASH 1; PAPER 2;s$;
2700 LET ss=sf
2710 LET ss=INT (ss+.5): GO SUB
920
2720 PRINT TAB (31-LEN s$+1); IN
VERSE 1;s$
2730 GO TO 2320
2740 CLS
2750 PAPER 0: BORDER 0: INK 7: C
LS
2760 LET lo=0
2770 RESTORE 610
2780 GO SUB 300
2790 LET n=k-5
2800 IF n>9 THEN LET n=9
2810 CLS
2820 LET p=n
2830 PRINT AT 0,0;c$;AT 0,0; INV
ERSE 1;y$;t$(2); INVERSE 1;TAB 1
3;m$(n);TAB 18;m$(n+1);TAB 23;m$
(n+2);TAB 28;m$(n+3)
2840 FOR d=1 TO 18
2850 LET cr=0
2860 PRINT TAB 0; INVERSE 1;" ";
2870 PRINT TAB 2;b$(d);
2880 LET st=6
2890 LET c=15
2900 GO SUB 950
2910 IF d=19 THEN GO TO 2840
2920 NEXT d
2930 PRINT AT 19,0;"TOTALS  ";
2940 LET cr=0
2950 LET c=15
2960 LET st=8
2970 INVERSE 1
2980 GO SUB 950
2990 INVERSE 0
3000 GO TO 2320
3010 BORDER 0: PAPER 0: INK 7: C
LS
3020 LET st=16
3030 RESTORE 580
3040 GO SUB 300
3050 LET d=k-2
3060 LET cr=1
3070 CLS
3080 PRINT "CATEGORY DATA"; INVE
RSE 1;y$; INVERSE 0,,,
3090 PRINT INVERSE 1;b$(d);TAB 1
2;"PLAN";TAB 17;"ACT";TAB 22;"VA
R"
3100 FOR m=1 TO 12
3110 PRINT TAB 0;m$(m);
3120 LET lo=1
3130 LET s$=STR$ INT (a(d,m)+.5)
3140 LET c=14
3150 GO SUB 950
3160 LET s$=STR$ INT (e(d,m)+.5)
3170 LET c=19
3180 GO SUB 950
3190 LET s$=STR$ INT ((a(d,m)-e(
d,m))+.5)
3200 LET c=24
3210 GO SUB 950
3220 NEXT m
3230 LET lo=1
3240 PRINT 'TAB 0;"TOTALS";
3250 INVERSE 1
3260 LET s$=STR$ INT (t(d,12)+.5
)
3270 LET c=14
3280 GO SUB 950
3290 LET s$=STR$ INT (u(d,12)+.5
)
3300 LET c=19
3310 GO SUB 950
3320 LET s$=STR$ INT ((t(d,12)-u
(d,12))+.5)
3330 LET c=24
3340 GO SUB 950
3350 INVERSE 0
3360 GO TO 2320
3370 BORDER 0: PAPER 0: INK 7: C
LS
3380 PRINT AT 18,0; PAPER 2;" Cu
rrency conversion factor "' PAPE
R 0;"'1' for local."'"Exchange R
ate for other."
3390 INPUT "[ENTER conversion fa
ctor]",exch
3400 LET cf=3
```

```
3410 BORDER 2: PAPER 0: INK 7: C
LS
3420 RESTORE 610
3430 GO SUB 300
3440 LET n=k-5
3450 RESTORE 580
3460 CLS : GO SUB 300
3470 LET d=k-2
3480 LET ss=INT (e(d,n)*100)/100
3490 CLS : PRINT AT 3,1;b$(d);"
for ";m$(n);" = ";ss
3500 INPUT "ENTER new expense."'
exp
3510 LET exp=exp*exch
3520 LET ss=INT (exp*100)/100
3530 PRINT AT 5,1;"New expense=
 ";ss
3540 LET bal=e(d,n)+exp
3550 LET e(d,n)=bal
3560 LET ss=INT (bal*100)/100
3570 PRINT AT 7,1;"Total       =
 ";ss
3580 PRINT #1;AT 1,0;"More entri
es? Y/N"
3590 LET z$=INKEY$: IF z$="" THE
N GO TO 3590
3600 IF z$="N" THEN CLS : PRINT
AT 10,10; FLASH 1;"Totalling": GO
 SUB 800: GO SUB 830: GO TO 1470
3610 IF z$="Y" THEN PAUSE 30: CL
S : GO TO 3450
3620 GO TO 3580
3630 CLS
3640 INPUT "ENTER Date (YYMMDD)"
, LINE x$
3650 GO TO 1470
3660 CLS
3670 PRINT "ENTER Month #"
3680 INPUT n
3690 IF n<1 OR n>12 THEN CLS : G
O SUB 890: GO TO 3670
3700 CLS
3710 FOR d=1 TO 18
3720 CLS
3730 PRINT AT 6,0;"ENTER ";b$(d)
,"Actuals for ";m$(n)''
3740 INPUT e(d,n)
3750 PRINT e(d,n)
3760 PAUSE 20
3770 NEXT d
3780 CLS : PRINT FLASH 1;"Totali
ng": GO SUB 800: GO SUB 830
3790 GO TO 1470
3800 POKE 23658,0
3810 LET x1=64: LET y1=88: LET r
1=64
3820>CLS
3830 RESTORE 3860
3840 GO SUB 300
3850 LET gt=k-18
3860 DATA INT 4,16,6,19,"GRAPH T
YPE MENU"," ","PIE CHART","BAR G
RAPH"
3870 RESTORE 3900
3880 CLS : GO SUB 300
3890 LET gc=k-18
3900 DATA INT 4,16,6,19,"DISPLAY
 MENU"," ","PLANNED","ACTUALS"
3910 RESTORE 610
3920 CLS : GO SUB 300
3930 LET n=k-5
3940 IF gc=1 THEN GO SUB 870: LE
T t3=sa3: LET g$=t$(1)
3950 IF gc=2 THEN GO SUB 860: LE
T t3=sa2: LET g$=t$(2)
3960 IF gt=1 THEN GO TO 3980
3970 IF gt=2 THEN GO TO 4970
3980 POKE 23658,0: BORDER 0: PAP
ER 0: BRIGHT 0: INK 7: CLS
3990 CIRCLE x1,y1,r1
4000 IF t3=0 THEN PRINT AT 10,5;
"No data available";AT 12,5;"Pre
ss ENTER for menu": PAUSE 0: GO
TO 1470
4010 LET m1=r1/8
4020 GO SUB 900: DIM z(18)
4030 IF gc=1 THEN FOR i=1 TO 18:
 LET k(i)=t(i,n)/t3: NEXT i
4040 IF gc=2 THEN FOR i=1 TO 18:
 LET k(i)=u(i,n)/t3: NEXT i
4050 PLOT x1,y1
4060 DRAW 0,r1
4070 FOR i=1 TO 18
4080 LET z(i)=k(i)*360*.017453
4090 NEXT i
4100 DIM w(18)
4110 LET w(1)=z(1)
4120 FOR i=2 TO 18
4130 LET w(i)=z(i)+w(i-1)
4140 NEXT i
4150 GO TO 4620
4160 FOR i=1 TO 17
4170 IF w(i)>=0 AND w(i)<=.5*PI
THEN GO TO 4320
4180 IF w(i)<=PI AND w(i)>.5*PI
THEN GO TO 4280
4190 IF w(i)>PI AND w(i)<=1.5*PI
 THEN GO TO 4240
4200 LET w(i)=2*PI-w(i)
4210 PLOT x1,y1
4220 DRAW -SIN w(i)*r1,COS w(i)*
r1
4230>GO TO 4340
4240 LET w(i)=w(i)-PI
4250 PLOT x1,y1
4260 DRAW -SIN w(i)*r1,-COS w(i)
*r1
4270 GO TO 4340
4280 LET w(i)=PI-w(i)
4290 PLOT x1,y1
4300 DRAW SIN w(i)*r1,-COS w(i)*
r1
4310 GO TO 4340
4320 PLOT x1,y1
4330 DRAW SIN w(i)*r1,COS w(i)*r
1
4340 NEXT i
4350 LET y2=21-y1/8: LET x2=x1/8
4360 FOR i=1 TO 18
4370 IF gc=1 AND t(i,n)=0 THEN G
O TO 4520
4380 IF gc=2 AND u(i,n)=0 THEN G
O TO 4520
4390 IF o(i)>=0 AND o(i)<=.5*PI
THEN GO TO 4510
4400 IF o(i)>.5*PI AND o(i)<=PI
THEN GO TO 4480
4410 IF o(i)>PI AND o(i)<=1.5*PI
 THEN GO TO 4450
4420 LET o(i)=2*PI-o(i)
4430 PRINT AT y2-COS o(i)*m1,x2-
SIN o(i)*m1;CHR$ (i+64)
4440 GO TO 4520
4450 LET o(i)=o(i)-PI
4460 PRINT AT y2+COS o(i)*m1,x2-
SIN o(i)*m1;CHR$ (i+64)
4470 GO TO 4520
4480 LET o(i)=PI-o(i)
4490 PRINT AT y2+COS o(i)*m1,x2+
SIN o(i)*m1;CHR$ (i+64)
4500 GO TO 4520
4510 PRINT AT y2-COS o(i)*m1,x2+
SIN o(i)*m1;CHR$ (i+64)
4520 NEXT i
4530 IF gc=1 THEN FOR i=1 TO 18:
 PRINT AT i+1,17; INVERSE 1;CHR$
 (i+64); INVERSE 0;b$(i);
4540 IF gc=1 THEN LET s$=STR$ IN
T t(i,n): PRINT TAB (31-LEN s$+1
);s$: NEXT i
4550 IF gc=2 THEN FOR i=1 TO 18:
 PRINT AT i+1,17; INVERSE 1;CHR$
 (i+64); INVERSE 0;b$(i);
4560 IF gc=2 THEN LET s$=STR$ IN
T u(i,n): PRINT TAB (31-LEN s$+1
);s$: NEXT i
4570 LET s$=STR$ INT t3: PRINT A
T 20,17; INVERSE 1;"Total"; INVE
RSE 0;TAB (31-LEN s$+1); INVERSE
 1;s$; INVERSE 0
```

```
4580>PRINT AT 0,17; INVERSE 1;"
 YTD   ";m$(n);y$
4590 PRINT AT 1,17; INVERSE 1;c$
( TO 15)
4600 PRINT AT 1,19;g$
4610 GO TO 4680
4620 DIM o(18)
4630 LET o(1)=z(1)/2
4640 FOR I=2 TO 18
4650 LET o(I)=z(I)/2+w(I-1)
4660 NEXT I
4670 GO TO 4160
4680 PRINT #1;AT 0,0; PAPER 2;"E
NTER>"; PAPER 0;" 1=MENU   2=TAB
LE   3=COPY"
4690 LET z=CODE INKEY$-48
4700 IF z=1 THEN GO TO 1470
4710 IF z=2 AND gc=1 THEN GO SUB
 900: GO TO 4760
4720 IF z=2 AND gc=2 THEN GO SUB
 910: GO TO 4760
4730 IF z=3 AND IN 251<>126 THEN
 INPUT #1: PAUSE 30: COPY : GO T
O 4680
4740 IF z=3 AND IN 251=126 THEN
PRINT #1;AT 0,0;"TURN PRINTER ON
": BEEP .2,30: PAUSE 30:: GO TO
4730
4750 GO TO 4680
4760 CLS : PRINT AT 0,0; INVERSE
 1;c$;AT 0,0;y$;t$(gc); INVERSE
1;TAB 15;"Amount";TAB 23;"Percen
t"
4770 LET tp=0
4780 FOR I=1 TO 18
4790 LET pc=INT (k(I)*10000)/100
4800 LET ss=INT (t(I,n)*100)/100
4810 LET sss=INT (u(I,n)*100)/10
0
4820 LET tp=tp+pc
4830 IF gc=1 THEN PRINT INVERSE
1;CHR$ (I+64); INVERSE 0;b$(I);
4840 IF gc=1 THEN LET s$=STR$ ss
: GO SUB 110: PRINT TAB (20-LEN
s$+1);s$;
4850 IF gc=1 THEN LET s$=STR$ pc
: GO SUB 110: PRINT TAB (28-LEN
s$+1);s$;
4860 IF gc=1 THEN PRINT TAB 31;"
"
4870 IF gc=2 THEN PRINT INVERSE
1;CHR$ (I+64); INVERSE 0;b$(I);
4880 IF gc=2 THEN LET s$=STR$ ss
s: GO SUB 110: PRINT TAB (20-LEN
 s$+1);s$;
4890 IF gc=2 THEN LET s$=STR$ pc
: GO SUB 110: PRINT TAB (28-LEN
s$+1);s$;
4900 IF gc=2 THEN PRINT TAB 31;"
"
4910 NEXT I
4920 LET st3=INT (t3*100)/100
4930 PRINT INVERSE 1;TAB 0;"Tota
ls =>";m$(n);
4940 LET s$=STR$ st3: GO SUB 110
: PRINT INVERSE 1;TAB (20-LEN s$
+1);s$;
4950 LET s$=STR$ tp: PRINT INVER
SE 1;TAB (28-LEN s$+1);tp;TAB 32
4960 GO TO 4680
4970 CLS
4980 DIM q(18)
4990 LET gmax=-10000: LET min=10
000
5000 IF gc=1 THEN GO SUB 5310
5010 IF gc=2 THEN GO SUB 5360
5020 LET loc=58
5030 LET sca=0: IF (gmax-min)<>0
 THEN LET sca=(80-12)/(gmax-min)
5040 PRINT AT 21,7;"ABCDEFGHIJKL
MNOPQR"
5050 FOR I=0 TO 8 STEP 4: LET zz
=INT (.5+min+((gmax-min)/8)*(8-I
)): PRINT AT 11+I,5-LEN STR$ zz;
zz: NEXT I
5060 FOR I=8 TO 8+8*10 STEP 4: P
LOT 57,I: DRAW 255-112,0: NEXT I
5070 FOR I=1 TO 18
5080 FOR I=0 TO 4
5090 IF NOT q(I) THEN LET loc=lo
c+1: GO TO 5130
5100 PLOT loc,8: DRAW 0,12
5110 LET loc=loc+1
5120 DRAW 0,(q(I)-min)*sca
5130 NEXT I
5140 LET loc=loc+3
5150 NEXT I
5160 IF gc=1 THEN FOR I=1 TO 9:
PRINT AT I,0; INVERSE 1;CHR$ (I+
64); INVERSE 0;b$(I);
5170 IF gc=1 THEN LET s$=STR$ IN
T t(I,n): PRINT TAB (14-LEN s$+1
);s$: NEXT I
5180 IF gc=2 THEN FOR I=1 TO 9:
PRINT AT I,0; INVERSE 1;CHR$ (I+
64); INVERSE 0;b$(I);
5190 IF gc=2 THEN LET s$=STR$ IN
T u(I,n): PRINT TAB (14-LEN s$+1
);s$: NEXT I
5200 IF gc=1 THEN FOR I=10 TO 18
: PRINT AT I-9,16; INVERSE 1;CHR
$ (I+64); INVERSE 0;b$(I);
5210 IF gc=1 THEN LET s$=STR$ IN
T t(I,n): PRINT AT I-9,(30-LEN s
$+1);s$;
5220>IF gc=1 THEN PRINT AT I-9,3
1;" ": NEXT I
5230 IF gc=2 THEN FOR I=10 TO 18
: PRINT AT I-9,16; INVERSE 1;CHR
$ (I+64); INVERSE 0;b$(I);
5240 IF gc=2 THEN LET s$=STR$ IN
T u(I,n): PRINT AT I-9,(30-LEN s
$+1);s$;
5250 IF gc=2 THEN PRINT AT I-9,3
1;" ": NEXT I
5260 PRINT AT 10,0; INVERSE 1;c$
;AT 10,9;"Total: ";INT t3
5270 PRINT AT 0,0; INVERSE 1;c$
5280 PRINT AT 0,6; INVERSE 1;"YT
D ";m$(n);" ";y$;" "; INVERSE 0;
g$
5290 GO TO 4680
5300 RETURN
5310 FOR j=1 TO 18
5320 LET q(j)=t(j,n): IF q(j)>gm
ax THEN LET gmax=q(j)
5330 IF q(j)<min THEN LET min=q(
j)
5340 NEXT j
5350 RETURN
5360 FOR j=1 TO 18
5370 LET q(j)=u(j,n): IF q(j)>gm
ax THEN LET gmax=q(j)
5380 IF q(j)<min THEN LET min=q(
j)
5390 NEXT j
5400 RETURN
5410 REM ! CLOSE #
5420 CLEAR : RANDOMIZE USR 100:
SAVE "budget.BJ" LINE 10
```

SDOS AUTO_DEX, MARK_MOVE, AND VERI_DISK

Bob Hartung, 2416 N. Co. Line, Huntertown, IN 46748

In the January issue of SDU I gave the listing for a routine able to store and access from a single disk an index of more file titles than most of us will ever get around to using (no. tracks/disk X no. files/disk). This DOSDEX routine is based on an adapation of the versatile SDOS menu-loader created by Roelof Mulder with several modifications by John Oliger.

The machine code for DOSDEX and other utilities listed below is that used in the menu-loader except that if it is poked into a 39-byte first-line REM the 8th byte must be an 8 instead of the 14 in the original listing which utilizes a fast FOR/NEXT definition. For those who have the menu-loader routine with the code source in a line 10 q$ definition, after creating a 39-byte line 1 REM the code may be moved from q$ to the REM by using LISTING 1. If starting from scratch the code may be put into a DATA line and READ used for the pokes.

Rather than "let your fingers do the walking" through the pages of DOSDEX, the AUTO-DEX in LISTINGs 2 and 3 (SEARCH and FILE_DEX) provides an automatic search for a given title among all those saved on a disk as c$ arrays. It provides the option of a full-title search when the search word is padded out to a length of 10 by characters or spaces, or a wild-card search can be made for any set of 1-9 consecutive characters found in a title. If the wild-card search word is to be found at the very beginning of the file title, entering one leading space before the 1-9 character search word will allow a much faster search to be made as is also done for a full-title search.

The main drawback to using a full-title search is that the search-word must be entered exactly as the title you want to find. You could turn your printer on and key (LET /P=0) to print out a hard copy of the titles as the data disk is made up--just in case your memory fails. But then that defeats the purpose of an automatic search if you have to eyeball your way all through the printout or else use DOSDEX to find how to enter the title!

Anyway, here's how it works, if you're interested. The AUTO_DEX search routine is saved to file 0 by (RUN 9999) and FILE_DEX is saved to file 1, also by (RUN 9999). If you want a full display of individual files in FILE_DEX save mode, start with the listing as given for DOSDEX in the January SDU, and change line 526 to correspond with LISTING 3 which displays only the disk number, format title, and prompts. Lines 600-650 can be omitted in either case. The mod for line 526 is to to save as DATA c$() the catalog info from each file disk in a numbered sequence.

After these programs have been saved to your AUTO_DEX disk, label your file disks numerically if you have not already done so for DOSDEX and key (LOAD) with the AUTO_DEX disk in the drive. At the prompt, key (4) to load FILE_DEX. Insert file disk 1 in the drive and key an upper-case (C) for CAT. Return the AUTO_DEX disk to the drive and key upper-case (S) to save the directory data, and so on. As for DOSDEX, you could easily adapt the routine to use one drive for the index disk and the second drive to catalog your file disks.

To make an auto-search, key (LOAD) with the AUTO_DEX disk in place and any key except (4). At the search-word prompt, if an exact-title search is to be made, make your entry, then add spaces to pad out its length to 10. If you hold down the space-bar for an auto-repeat, any extra spaces added to the title will be truncated to the correct length. Precede any wild-card set of 1-9 characters with one space if that character set occurs at the very beginning of the targeted file title. If the set occurs elsewhere in the file title the leading space is omitted. Then enter a null-string (ENTER) if it is a Basic listing, or (token-word) plus (n) or ($) to denote the type of file.

When a match is found with the search-string title, the file number and the formatted title of the required disk is displayed. When that disk is inserted the file may be loaded from it. In a search for a full-title or a leading character-set the drive will seldom stop running from one index file to the next but a random wild-card search will take a little longer. The string-slicing will not allow TIMACHINE to compile the wild-card search, but those who are into machine code and want the ultimate in speed could adapt Tom Woods' Profile search routine to do it.

Roelof has also written another very useful utility he calls Extractor that among other things moves selected files from one disk to another. This method is a much safer way of recovering unused disk space even when an erase-and-recover routine is available. When one sector is being re-copied to another on the same disk there is always the risk of a glitch corrupting part if not all of the disk data. While not nearly as elegant as Roelof's Extractor, the MARK-MOVE routine (LISTING 4) adapted from his SDOS menu-loader provides the essentials to format a target disk and move to it the files selected from a source disk. Keying (SPACE) will mark a file for moving, a zero will unmark it, any other character will advance the cursor, and (ENTER) will commence the moves.

LISTING 5 is yet another adaptation of the SDOS menu-loader which uses the VERIFY function to check the integrity of each file on a disk. It might be in order here to discuss some DOS facts of life that, human nature being as it is, probably all of us are aware of but have neglected at times. To be on the safe side--and we're talking about all magnetic media data storage now--it is recommended practice to use a three-tiered approach. This calls for

at least two back-up copies, either disk and disk, or disk and tape, for any files that we don't want to lose. The original is then used only when an update is added or when both the working and backup files become corrupted or lost. It is also good practice to periodically check these files, especially if loading errors are encountered on any disks done at about the same time or after the backups.

By their very nature, all magnetic recording media (including audio and VCR as well as DOS) begin losing some of their playback signal strength, particularly in the higher frequencies, from the instant any recording is made upon them. This occurs at a diminishing rate somewhat like an inverse logarithmic progression or "half-life" process. As this "clipping" approaches the level expressed in percentage on all labelled name-brand disks, it happens more slowly but, unfortunately, never quite stops at zero.

Of course, many other factors could cause loss of data. These include possible instantaneous damage by magnetic fields such as a speaker or nearby AC cord, or static voltage potentials such as those on the face of a TV or monitor CRT or even on your fingertips if you have walked across the room or shuffled your feet on a carpet in low-humidity conditions. Creasing or folding a disk, dropping something upon it or writing on the sleeve with too much pressure, or touching the exposed surface can ruin it. There is also the possibility of dirt on the drive-heads or dust, heat and humidity hastening a disk's demise. Tobacco smoke is dangerous to the health of data as it is to the user's. Because of greater data-density per unit of media area, quad density systems are more sensitive to all these factors than double density systems, particularly when the less-costly DD disks are used with a QD system, which most of us do at times.

As John Oliger noted in the first issue of SDU, SDOS users have an advantage over users of some systems because we get an audible warning whenever a disk sector fails to save/verify or load and must be re-tried. However, even with SDOS an infrequently-used file or backup disk might drop below a critical clipping level without our being aware of it. If ever a data failure is encountered on a disk that previously showed no problems, unless you have done so periodically it is time to check all disks formatted at or before the time when that one was done.

Unless you are turned on by such things as watching paint dry or keying in VERIFY /"name" file-type for every file on every disk, a routine like VERI-DISK will take some of the tedium out of doing this. After loading VERI-DISK, insert a file disk and then press any key to CAT it and start the verifying sequence. Any failed files will be displayed by name and number in the order they are stored on the disk.

Since the VERIFY function tries only one time before returning an error report when even one item of data fails, if it is done in time it is usually possible to re-save or move a file that doesn't verify because of clipping. Because after extended non-use all magnetic media also loses coercivity, i.e., it takes a "set" and becomes less responsive to recorded signals impressed upon it, it is good practice to move the entire contents to another freshly-formatted disk. AFTER MAKING SURE ALL FILES HAVE BEEN MOVED SUCCESSFULLY this can become your new backup or work-disk, or else the original disk may then be re-formatted and the files moved back to it from the intermediate storage disk.

When the clipping level has become critical, data may still load OK on the drive used to save it but not on another, so it is a good idea to note the drive number on each disk label. If a disk will not load at all, John Oliger's routine for restoring directory files (SDU issue 1) may work to salvage some if not all the files in instances where track 0 has become corrupted.

LISTING 1

```
  1 REM 123 *39-byte MC is poked here* 23456789
  5 LET a=VAL "PEEK 23635+PEEK 23636*256+5"
 10 LET q$="(Load original menu-loader listing that has a
q$ definition. Delete all lines except this one. Replace
first + CHR$ VAL "14" with + CHR$ VAL "8" in the
definition. Add the lines for this routine and (RUN).)"
 20 LET d=1
 30 FOR n=a TO a+38
 40 POKE n, CODE q$(d): REM To use a DATA line, READ p:
POKE n,p
 50 LET d=d+1
 60 NEXT n
100 DATA 205,10,0,42,75,92,17,8, etc. : REM Not needed to
transfer from q$
```

LISTING 2 (SEARCH)

```
  1 CLEAR : LET n=1
  5 PRINT #0;"Key 4 TO SAVE DATA index files  OR any othe
r key for AUTO_DEX": PAUSE 0: IF INKEY$="4" THEN LOAD /"FI
LE_DEX"
 10 LET d=1
 20 DIM c$(178,20)
 30 LOAD /"1" DATA c$()
100 INPUT "Search-word: 1-9 CHR$ wild-card (plus 1 leadin
g space if start  of title)  OR pad to ↑ arrow if full tit
le-";s$'"""" DATA n DATA $ CODE ABS VAL ?",t$: IF LEN s$)
10 THEN LET s$=s$( TO 10)
110 LET t=0: LET f=0: LET f=(1 AND t$=" DATA n")+(2 AND t
$=" DATA $")+(3 AND t$="CODE ")+(4 AND t$="ABS ")+(5 AND t
$="VAL ")
115 IF s$(1)=" " THEN LET s$=s$(2 TO ): LET s=LEN s$: LET
t=1: GO TO 140
120 LET s=LEN s$
130 IF s<10 THEN GO TO 180
140 FOR /178
```

```
 150 IF c$(n,1)=" RESTORE " THEN GO TO 400
 160 IF c$(n, TO s)=s$ AND CODE (c$(n,11))=f THEN LET k=n:
 GO TO 600
 170 NEXT
 175 GO TO 400
 180 LET p=1
 190 FOR k=1 TO 178
 195 IF c$(k,1)=" RESTORE " THEN GO TO 400
 200 PRINT #0;AT 0,0;d
 205 FOR /10-s
 210 IF c$(k,n TO n+s-1)=s$ AND CODE (c$(k,11))=f THEN GO
TO 600
 230 NEXT
 240 NEXT k
 400 LET d=d+1: ON ERR GO TO 1000: LOAD /STR$ d DATA c$():
 ON ERR RESET : GO TO 140*(s=10 OR t)+180*(s(10 AND NOT t)
 600 PRINT s$;TAB 11;t$;" = search$"''c$(k, TO 10);" ";t$;
" is file #";k,"on disk #";d;" -- ";c$(178, TO 16): PRINT
#0;"Insert disk & key ENTER TO LOAD or any other key TO RE
PEAT": PAUSE 0: IF CODE INKEY$()13 THEN GO TO 10
 605 LET a=CODE c$(k,11): LET d$=c$(K, TO 10): IF NOT a TH
EN LOAD /d$
 610 IF a=SGN PI THEN LOAD /d$ DATA n()
 620 IF a=VAL "2" THEN LOAD /d$ DATA n$()
 630 IF a=INT PI THEN LOAD /d$CODE
 640 IF a=VAL "4" THEN LOAD /d$ABS
 650 IF a=VAL "5" THEN LOAD /d$VAL
1000 ON ERR RESET : PRINT s$;" ";t$; FLASH 1;" NOT FOUND "
: GO TO 10
2000 STOP
9999 CLEAR : SAVE /0
```

LISTING 3 (FILE_DEX)

```
   1 REM 1234567890123456 MC goes here 123456789
   2 CLEAR : DIM c$(VAL "178",VAL "20"): LET s=SGN PI: LET
 o=NOT PI: LET d=s
   5 LET a=VAL "PEEK 23635+PEEK 23636*256+5": LET c=INT (a
/VAL "256"): POKE VAL "23549",VAL "195": POKE VAL "23550",
a-(c*VAL "256"): POKE VAL "23551",c: LET fi=USR VAL "23549
": LET n$=c$(VAL "178", TO VAL "16")
 300 CLS : IF n$(LEN n$)=" " THEN LET n$=n$( TO LEN n$-s):
 GO TO VAL "300"
 320 PRINT AT o,o;" Disk#";d-s;": ";n$;#o;"Key: NEXT CAT S
AVE #";d
 510 LET a$=INKEY$: IF a$="" THEN GO TO VAL "510"
 522 IF a$="C" THEN OPEN #2,"P": CAT : CLOSE #2: GO TO INT
 PI
 526 IF a$="S" THEN LET d=d+s: SAVE /STR$ (d-s) DATA c$():
 GO TO INT PI
 528 IF a$="N" THEN INPUT " INPUT NEXT #";d: GO TO INT PI
 530 GO TO VAL "510"
9999 CLEAR : SAVE /"FILE_DEX" LINE 2
```

LISTING 4 (MARK_MOVE)

```
   1 REM 1234567890123456 MC goes here 1234567890
   2 CLEAR : DIM c$(VAL "178",VAL "20"): LET s=SGN PI: DIM
m$(VAL "177",s): LET f=VAL "4": LET t=VAL "10": LET o=NOT
PI: INK VAL "7": PAPER o: BORDER o: CLS
  10 DIM f$(VAL "6",s+s): LET f$(s+s)=" DATA n": LET f$(IN
T PI)=" DATA $": LET f$(VAL "4")="CODE ": LET f$(VAL "5")=
"ABS ": LET f$(VAL "6")="VAL "
  20 PRINT AT o,t;"FILE-MOVER";#o;"Place source disk in DR
IVE 0 &  target disk in DRIVE 1          Key any CHR$ TO C
AT source disk or key ENTER TO NOT CAT ": PAUSE o: IF CODE
 INKEY$()VAL "13" THEN OPEN #2,"P": CAT : CLOSE #2
  30 INPUT "": PRINT #0;" TO FORMAT target disk? y/n": PAU
SE o: IF INKEY$="y" THEN INPUT " INPUT FORMAT  ↑ title to
 arrow";a$: LET /d=1: FORMAT /a$: LET /d=0: OPEN #2,"P": C
AT : CLOSE #2
  40 CLS : PRINT #o;"Key:SPACE to mark CHR$ to review0 to
revise ENTER TO START MOVE"
 100 LET a$="": DRAW INK s;VAL "255",o: DRAW INK s;o,VAL "
175": DRAW INK s;VAL "-255",o: DRAW INK s;o,VAL "-175"
 200 LET a=VAL "PEEK 23635+PEEK 23636*256+5": LET c=INT (a
/VAL "256"): POKE VAL "23549",VAL "195": POKE VAL "23550",
a-(c*VAL "256"): POKE VAL "23551",c: LET fi=USR VAL "23549
": LET row=s+s: LET col=VAL "9": LET n$=c$(VAL "178", TO V
AL "16")
 300 IF n$(LEN n$)=" " THEN LET n$=n$( TO LEN n$-s): GO TO
 VAL "300"
 320 PRINT AT o,VAL "16"-(LEN n$/VAL "2"); OVER s; INK VAL
"5";n$;AT o,VAL "8"; OVER s;"_____": PAPER o
 405 LET L=o: LET f=s: LET c=INT (fi/VAL "18"): LET dif=IN
T ((fi/VAL "18"-c)*VAL "18"+VAL ".4"): LET loop=VAL "17"
 410 LET q$="p": LET it=s: IF loop)=fi THEN LET loop=fi: G
O TO VAL "425"
 415 FOR i=s TO c: FOR m=o TO loop: PRINT AT row+m,t-f;a$;
AT row+m,t-LEN STR$ it;it;m$(it);c$(it, TO t);" ";f$(CODE
c$(it,VAL "11")+1): LET it=it+s: NEXT m: GO SUB VAL "500":
 NEXT i: FOR i=s+s TO VAL "19": PRINT AT i,t-f;a$: NEXT i:
 IF NOT dif THEN GO TO VAL "410"
 425 IF f THEN FOR m=o TO dif-s: PRINT AT row+m,t-f;a$;AT
row+m,t-LEN STR$ it;it;m$(it);c$(it, TO t);" ";f$(CODE c$(
it,VAL "11")+1): LET it=it+s: NEXT m: IF loop)=fi THEN LET
 f=o
 427 IF NOT f THEN LET it=fi+s
 430 GO SUB VAL "500": GO TO VAL "410"
 500 FOR L=o TO m-s: PRINT AT row+L,col; INVERSE s;")": IF
 q$()"" THEN FOR a=s TO PI↑PI: NEXT a: LET q$=""
 510 LET a$=INKEY$: IF a$="" THEN LET q$="p": GO TO VAL "5
10"
 512 IF a$="0" THEN LET m$(it-m+L)=" ": PRINT AT row+L,col
+SGN PI;" "
 515 IF a$=" " THEN LET m$(it-m+L)="■": PRINT AT row+L,col
+SGN PI;"■"
 520 IF a$=CHR$ VAL "13" THEN GO TO VAL "600"
 530 PRINT AT row+L,col-s-s;"   ": NEXT L: LET a$="
          ": RETURN
 600 BORDER VAL "7": PAPER VAL "7": INK o: CLS : INPUT "So
urce & object disks in drives?Key ENTER TO START MOVE ";a$
: CLS : PRINT FLASH s;" MOVING!  DO NOT STOP till end  "
 601 FOR M=s TO VAL "177"
 602 IF c$(M,s)=" RESTORE " THEN GO TO VAL "700"
 605 PRINT AT t,t;"File: ";m
 610 LET d$=c$(M, TO t): LET a=CODE c$(M,VAL "11"): IF NOT
 a AND m$(M)="■" THEN MOVE /d$
 615 IF a=s AND m$(M)="■" THEN MOVE /d$ DATA
```

2.5

```
 620 IF a=VAL "2" AND m$(M)="■" THEN MOVE /d$ DATA $
 630 IF a=INT PI AND m$(M)="■" THEN MOVE /d$CODE
 640 IF a=VAL "4" AND m$(M)="■" THEN MOVE /d$ABS
 650 IF a=VAL "5" AND m$(M)="■" THEN MOVE /d$VAL
 660 NEXT M
 700 PRINT AT o,o,,AT t,t; FLASH 1;" MOVE OVER ": INPUT "K
ey ENTER FOR NEW CAT "/"    Any CHR$ STOP ";a$: IF a$="" T
HEN RUN
 710 STOP
9999 CLEAR : SAVE /"MARK_MOVE" LINE 2
```

LISTING 5 (VERI_DISK)

```
   1 REM 1234567890123456 MC goes here 123456789
   2 CLEAR : DIM c$(VAL "178",VAL "20"): LET s=SGN PI: LET
 f=VAL "4": LET t=VAL "10": LET o=NOT PI
  10 DIM f$(VAL "6",s+s): LET f$(s+s)=" DATA n": LET f$(IN
T PI)=" DATA $": LET f$(VAL "4")="CODE ": LET f$(VAL "5")=
"ABS ": LET f$(VAL "6")="VAL "
  20 PRINT AT o,t;"VERI_DISK";#o;"Place NEXT disk in DRIVE
, then  any key TO VERIFY ": PAUSE o: OPEN #2,"P": CAT : C
LOSE #2
 200 LET a=VAL "PEEK 23635+PEEK 23636*256+5": LET c=INT (a
/VAL "256"): POKE VAL "23549",VAL "195": POKE VAL "23550",
a-(c*VAL "256"): POKE VAL "23551",c: LET fi=USR VAL "23549
": LET n$=c$(VAL "178", TO VAL "16")
 210 ON ERR GO TO 800
 230 FOR m=s TO VAL "177"
 240 IF c$(m,s)=" RESTORE " THEN GO TO VAL "700"
 250 PRINT #o;AT o,o;"File: ";m
 600 LET d$=c$(m, TO t): LET a=CODE c$(m,VAL "11"): IF NOT
 a THEN VERIFY /d$
 610 IF a=s THEN VERIFY /d$ DATA
 620 IF a=VAL "2" THEN VERIFY /d$ DATA $
 630 IF a=INT PI THEN VERIFY /d$CODE
 640 IF a=f THEN VERIFY /d$ABS
 650 IF a=VAL "5" THEN VERIFY /d$VAL
 660 NEXT M
 700 ON ERR RESET : FOR m=s TO f: BEEP .2,RND*35: BEEP .1,
RND*15: NEXT m
 710 PRINT #o;AT o,o;n$; FLASH 1;" VERIFY OVER "; FLASH 0,
"Note failures--any key FOR NEXT ": PAUSE o: RUN
 720 STOP
 800 PRINT "Failed at #";m;" ";c$(m, TO t);" ";f$(CODE c$(
m,VAL "11")+1): NEXT m
9999 CLEAR : SAVE /"VERI_DISK" LINE 2
```

ADDENDUM: Since sending the above mss. to Bill, response to the DOSDEX article indicates some were confused by the instructions. Sorry about that! If the 8th byte in your MC listing or the 8th definition in q$ is already an 8, don't change any of the code to POKE it into the line 1 REM for DOSDEX or the above routines. Please note that in listings 3, 4, and 5, as well as DOSDEX, the respective values in lines 5 and 200 must be as given above for these listings, NOT as they are in the menu-loader. The original menu-loader values were set up to point to the VARS location of q$ in RAM, and so must be re-calculated if this q$ location is to be used for the code instead of the REM.

The Issue Disk

The UP-DATE ISSUE DISK is a valuable tool to verify all of your key-in work. And you have all of the programs and utilities to use while you wait for the time needed to devote to key-in projects (maybe next month). Maybe you wont have the time, ever. It usually takes me about two weeks of work to put together the issue disk and to verify all of the programs and utilities. Then it is time to start work on the next magazine issue. UP-DATE has become a full time job and I hope that you can see the effort expended in a gradual improvement in the contents. As the October issue begins to support the QL users I will try to hold TS-2068 coverage down to 30 pages, although that is difficult to do when the "IN BASKET" contains such outstanding articles as those submitted by the authors for the April 88 issue.

The LARKEN DISK Section

"BUDGET", by Bob Mitchell, came in to UP-DATE as a Larken Disk software. I kinda switched it to a "universal software", by nothing more than writing syntax. Part 2 of this software will be given in the July issue of UP-DATE and will give the "Loader Disk Manager" for LARKEN, OLIGER, and AERCO disk systems. UP-DATE has a treasure trove of excellent Larken Disk articles to present. They had to be placed in Back Log because of a equipment break-down (I couldn't verify the programs.). These articles and programs will be presented in the July issue and will take up about 8 pages.

TS-2068 DISK DRIVE SUPPORT

We TS-2068 users are currently supported by two dedicated Disk Drive Hardware engineers. John Oliger (The Oliger Co.) and Larry Kenny (Larken Electroncis) form the nucleus of our disk drive system support. Aerco seems to have developed other interests which has slowed further development of the Aerco FD-68 DOS. Maybe Aerco will see fit to finish the DOS? The Larken SKDOS Cartridge can give FD-68 users a finished DOS for use with the FD-68. In the meantime, both Larry Kenny and John Oliger are continually busy up-dating their disk drive systems and providing new hardware. UP-DATE salutes these two dedicated supporters. Both deserve our business. And, both individuals are eager to support their customers and the TS-2068 "user family", with detailed information in the pages of UP-DATE. Larken Electronics and The Oliger Co., together, are the BEEF of our TS-2068 Hardware support. Trust them, and look for their products.

DENSE PACK BASIC

A Method to Conserve Memory and Speed Up Basic

Most Computers use variations of the Dartmouth BASIC Language as their operating systems. The assembly language routines in ROM are designed to facilitate the Basic language as used by the operator and the Basic Editors of the computers are designed to process the particular variation of syntax used. Generally, the differences in syntax are easily learned by a user who has some experience operating any computer. So, once one learns the syntax one can begin to program the computer. This is quite an advancement over the way programming was done in the 60's, when a computer was programmed by punching holes in a card to be fed through a card reader.

So, things are now neatly arranged. We have the BASIC language to use and the computer is programmed to accept Key Words that the operator inputs from a keyboard and then Branch to a fixed ROM routine to execute the operator's desires. The operating system is FIXED in ROM. The BASIC language is just "semi-fixed" and has some flexibility. One of the flexible features has to do with mathematics expressions. The Central Processing Unit (CPU) of the computer has an Annex called "The Arithmetic Logic Unit" (ALU) which processes mathematics expressions and provides a Resultant. When a math expression is included in Basic Programming, the ALU solves the expression and the CPU stores the resultant.

If, within a program line, the expression "LET a=20*40" is used, the ALU solves "20*40" to allow the variable a to be assigned as 800. Almost all commonly used math expressions can be sent "IN BASIC" to the CPU. "LET c=2*PI*r" is acceptable as syntax to the basic intrepeter, and the CPU calls upon it's ALU to solve the math before it assigns a value to var c. Then if we use other math equations most will be acceptable. How about Boolean Logic expressions? Why not? Boolean Logic is the basis for the logic of most of the IC chips in the computer (AND Gates, OR Gates, NOT Gates, etc.). The CPU itself is largely a Boolean Logic device.

What is BOOLEAN anyway? Well, most of our younger set believe that computers sprang up as a new invention during the 70's. Actually most of the principles of computing extends back more than a hundred years. What sprang up was new manufacturing techniques of packaging electronics components. George Boole (1815-1867) gave us the logic used in computers. Mr. Boole spent a lifetime integrating two separate sciences, LOGIC and Mathematics. Then Mr. Einstein made great use of Mr. Boole's works and added to it. Before, Logic was considered to be literal, and mathematics was limited to the factoring of numbers. Boolean logic uses such expressions as AND, OR, NOT, in both a math and relational Logic sense.

Boolean logic can greatly shorten the way we express ourselves in Basic programming. Boolean logic can also speed up the execution of Basic programming. When Boolean expressions shorten the literal expressions in Basic, the programming does not require as much memory for storage. Most College courses in Computer Programming ignore Boolean because it is really an advanced Math discipline, and to introduce it in a programming class would require more semester time. Yet, it does not require extensive study to use simple Boolean expressions to great advantage.

Dense Pack Basic employs Memory and Time Saving techniques integrated with Basic Programming. Dense Pack is not a new language, but is a "method of programming", using math as logic, and existing memory saving techniques. It is reasonable to conclude that if a line of Basic programming can be reduced in Byte length by as much as 70%, the programming in the line will execute faster. Supporting this thesis is the fact that arithmetic functions executed by the ALU are the fastest operations performed by the computer.

The October issue of UP-DATE discussed "Pseudo Hex", a table of variables that substitutes for numbers. Most programmers use the principle, but in a un-organized manner. The pseudo hex table was designed to assist in remembering the variables used to represent numbers 1 through 20. These are the most used numbers in computer programs. To review, Pseudo Hex uses a double character variable "o", where "oa=1, ob=2, oc=3, to ou=20". A person tends to quickly learn that "oe=5". In the learning phase, one can count on fingers, using the alphabet to intrepet the variables. The use of such a variable system quickly becomes habit to programmers.

The Pseudo Hex table is constructed in program lines. Then GO TO the first line initializes the variables to memory. A good technique to use with disk drive systems is to then SAVE the Variables to disk where the vars table can be re-loaded to a CLEARED Basic program. When that is done, the program lines may be deleted. The memory cost of initializing the pseudo hex variables is usually recouped within the first 20 lines of a basic

program. A basic program of 20K in length is typically reduced to about 16K in length by using the variables table.

Now we will get into the meat of Dense Pack as it employs Boolean Logic and math operators in Basic Programming. The best way to start is with an example. I'll give a typical conventional program line after a MENU which has 9 optional electives. Normally the menu electives will be listed 1 through 9 and the operator touches a number key to make a selection. Ordinarily, a INKEY$ prompt would be used to assign the key touched to a numbered variable such as "z". The typical program lines after would be: 100 IF z=1 THEN GO TO 500 - 102 IF z=2 THEN GO TO 600 - 104 IF z=3 THEN GO TO 700 - etc, etc, until "9 IF THEN LOGIC lines are programmed". Then there would be a "Key Lock" line as: 118 IF z<1 OR z>9 THEN GO TO 90 (the menu).

The dense pack presentation would be only ONE line of programming instead of 10 lines. Vars would be used for each of the small numbers. The result would be a reduction of 9 lines of programming and 75% of memory required to store the programming. The equilivant programming in Dense Pack would be: 102 GO TO (z=oa)* 500 +(z=ob)* 600 +(z=oc)* 700 +(z=od)* 800) +(z=oe)* 900 +(z=of)* 1000 +(z=og)* 1100 +(z=oh)* 1200) +(z=oi)* 1300 +(z<oa OR z>oi)* 90.

The example dense pack (single) line requires only 25% of the memory needed to store the conventional 10 lines of the first example, and 9 fewer program lines. I will break down one of the IF THEN conditionals. "GO TO (z=oa)* 500". The operator is "GO TO" and is used only once for the 10 evaluations. "IF and THEN" are implied for each of the 10 conditions that are inclosed in (brackets). The term means: IF z=1 THEN GO TO 100. When the program line is processed ALL 10 of the conditions are evaluated as a single expression, where with conventional IF THEN conditions each program line is evaluated and a false condition would be ignored. The dense pack line operates faster and conserves 75% of program memory.

Now we will explore some more Dense Pack examples. I'll set up a program line with a pair of prompts. One will opt to solve the Circumference of a circle, and 2 will opt to solve the Area of a circle. 10 LET r=9: INPUT "<1> Circumference or <2> Area";a: LET y= (2*PI*r AND a=1) + (2*PI*r 2 AND a=2). That little routine lacks a trap for wrong key hit. Conventional IF THEN statements may be mixed with Dense Pack logic expressions, as: 100 IF a<10 THEN PRINT ("yes" AND a<6) + ("no" AND a>5): IF a<3 THEN GO TO (a<1)* 50 +(a=1)* 100 +(a=2)* 200). In that example IF a is smaller than 10 the print statement will execute and print "yes" if a is smaller than 6 or "no" if a is greater than 5. Then the last GO TO will execute only if a is smaller than 3.

Here is an actual dense pack line used in one of my programs. vars lx=maximum printer line, qq=center of page, tb=tab, ps=print style elected, xo=existing max printer line, lo=existing line length, and ll=line length elected. A matching line "ma" is computed: 8022 LET lx=(80 AND ps<3) + (96 AND ps=3) + (136 AND ps>3): LET qq= INT (lx/2+ .05): LET ma= INT (.5+lx*lo/xo): GO SUB 8088: LET tb=INT ((lx-ll)/2+.5 - 8088 PRINT AT 10,2;"Key in line length"''TAB 2;lx;"=Max ";ma="Match": INPUT ll: CLS RETURN. I left out the pseudo hex vars for easier follow through. These two lines are not intended to dazzle, but to illustrate how many IF THEN conditionals, plus computations, GO SUBS, and LET statements can be integrated into one dense pack line. Dense pack lines are "program packages".

As one gets into Dense Pack, the program lines become easier because they are independant "whole functions". Fewer GO TOs and GO SUBS are used to pick up other sub-routines, and "fall throughs" to next program lines are greatly reduced. Sometimes one tends to get too engrossed and does a blunder like my printing a listing of the "J-vars utility package" in the October 87 issue. Looking back, it is a most difficult listing for one to key in. To give you a better feel for the usefulness of Dense Pack, one of my programs was reduced from 28K of program length to about 12K. Then more functions could be added. All of the Key Words of any computer can be used as operators for Dense Pack programming.

RULES. 1. A Key Word can only operate on one variable within a set of brackets. <GO TO (a=1)* 100 +(b=1)* 200> would not work, because two evaluators are used "a and b". 2. ALL conditions should be accounted for. LET a= (10 AND b=2) +(30 AND b=4). In this example, the conditions of b<2, b=3, and b>4 are not accounted for. As a line is evaluated, and an un-accounted for condition is present, the CPU takes off hunting. Often this will result in the execution of an un-intended line near the end of the program. Use limiters such as "<" and ">" to account for all possible conditions. 3. Some mixes of AND and OR within the same brackets will not be intrepeted in the BOOLEAN sense, but in the literal. Only experimenting will proof the programming. Have fun with Dense PacK!

2.8

A Practical Study of SYSTEM VARIABLES

Put Them to Use with BASIC

Appendix D of the user manual list the memory addresses of RAM where most of the TS-2068 System Variables are stored. The left column "notes" tells whether the variable number is larger or smaller than 255. The significance of this is that system variable numbers larger than 255 require two memory addresses for storage. If the variable is stored in two addresses, the formula: PRINT PEEK (low adr) + 256 * PEEK (high address) returns the actual system variable number. We stared this discussion in the editorial section. Now let us examine some of the system variables.

PEEK K STATE and LAST K: These two system vars are used together by the TS-2068 ROM to read the keyboard. Then WHY are the vars in RAM? Well, the codes stored in ROM cannot be changed and a Keyboard READ requires the ability to store changing codes. Then how can we use these two system variables? There are two ways. One is to construct a little Machine Code loop routine to use the system vars to read the keyboard. The other way is to do it with a BASIC loop. Why do this when the INKEY$ function is already available? The answer is the INKEY$ function does not employ the Debounce and Error Detection routines in ROM, while K State does.

The following BASIC programming will read the keyboard and assign the "key struck" to "a$". 150 LET a$="": POKE 23611,220 - 151 IF PEEK 23611<220 THEN GO TO 151 - 152 LET c= PEEK 23560 - 153 IF c<32 THEN GO TO 160 - 154 LET a$=a$+ CHR$ c - 155 POKE 23611,220 - 156 GO TO 151 - 160 STOP. Explanation: At line 150 we start by initializing a$, then RESETING the system var FLAGS. This places the system in a wait state. Next at line 151 we form a loop to loop as long as FLAG remains in a wait state.

When a key is struck, the error detection and debounce routine in ROM processes the key value and if no error exists, the code of the key is placed at address 23560. In line 152 we pick up the code number of the key struck and assign it to var "c". Then at line 153 we have a ESCAPE so that any non printable key, such as ENTER, will escape the loop to process the results. At line 154 the character key struck is assigned to a$. Line 155 resets FLAGS, and then line 156 loops back to another keyboard scan at line 151. Two more variations of this PEEK K STATE routine, for PROMPTS in basic are given in the October issue of UP-DATE.

The above programming can be reduced to about three total lines by using chained commands. Another variation, using the OLIGER SAFE FOR/ counter is: 150 POKE 23611,220: FOR /65535: IF PEEK 23611<221 THEN NEXT - 152 LET c= PEEK 23560: POKE 23611,220: IF c<32 THEN GO TO 160 - 153 LET a$=a$+ CHR$ c: NEXT - 160 STOP.

REPDEL and REPPER: Want to perk up your keyboard and make it frisky? Page 262 of the user manual gives these two "single byte" system vars. Play around with POKES to the two addresses to get variations of the speed of REPEAT KEYS, and the DELAY between repeat key actions. How about making the keys Chirp? Do that by POKE to sys var PIP (23609). Try about 10. Then the sys var DF SZ lets you enlarge the prompt area at the bottom of the screen. For kicks, put on a whole screen and then BREAK and POKE 23659,15. Wow: now the prompt area is 15 lines which leaves only 9 lines at top!

OTHER SYSTEM VARS: We will skip some of the more obvious SYSVARS and deal with some more useful two byte vars. For these it will help to refer to page 254 of the user manual to visualize the TS-2068 memory map. On page 254, note that PROG is a line drawn without a specific address. This is because the BASIC PROGRAM, which normally begins at address 26710, may be intentionally changed. The System Var PROG is contained in two addresses, 23635 and 23636. Try PRINT PEEK 23635 + 256 * PEEK 23636. You should get 23710.

We will now find a nook in memory to stash a tiny MC Utility to shift the Basic program UP in RAM. The MC utility can go anywhere in RAM, but lets put it in a most unlikely place. Look on page 255 of the user manual. At the top of the page is a group of blocks. All of these are important places where "functions" are stored. But one can be used for multi purpose functions. The "printer buffer" is used only for holding data for printing something to paper with the TS-2040 printer, and is a ideal place to stash Machine code when the TS-2040 printer is not to be used.

So, we will do some slick shenanigans. A basic utility will be constructed that contains a tiny Machine Code routine. The utility will put it's own MC routine into the Print Buff area, and then Execute it. The result will be a "selective starting address" of any Basic program to which the utility is appended. In simplese, the utility will move a basic program UP in memory and also move PROG so that the CPU will know where to go to operate the Basic program.

A Utility that uses System Variables

The following utility really should be in the EXTRA MEMORY section, because it is so useful be able to MOVE a BASIC program to a specific STARTING MEMORY ADDRESS. It is a take off from the "MOVE" utilities given in the January issue of UP-DATE. But, we are discussing the use of SYSTEM VARIABLES, and this utility does a good job of demonstrating the use of system variables PROG, and VARS. PROG, of course, is the starting address of the first byte of the first line of the BASIC PROGRAM. VARS is the starting address of the first byte of the variables area. To get a good mind's picture of these two system variables, refer to page 254, left block. The Basic program normally is arranged so that it's first byte starts at address 26710. Then PROG is 26710. The Variables area (VARS) is ever moving as the basic program is added to by line numbers or changes of programming within the lines.

So, the system variable VARS is a very important tool. We can use it to find out the "LAST BYTE" of a BASIC PROGRAM. Since VARS begins with the "next byte after" the basic program ends, then all we have to do is to subtract "1" from VARS (when we find VARS). Now look at page 263 and about 1/3 down we see that VARS is given as address 23627, and it is a 2 byte variable. So <PRINT PEEK 23627 + 256 * PEEK 23628> will return the memory address where the variables area starts. Then subtract 1 and that will be the TOP address of the BASIC program. Next, we look on page 263 and find that PROG is given as a 2 byte variable starting at Address 23635. Then <PRINT PEEK 23635+256 * PEEK 23636> will return the address of the STARTING byte of the Basic program. Subtract the product of the two formulas and you have the TOTAL Number of BYTES of the Basic Program.

It is useful to be able to re-arrange the Memory Map of the computer when you want to. One need is to place a Basic program at the "starting address of a Chunk of memory" when we want to MOVE the program into the dock bank. A good discussion on this subject is in the EXTRA MEMORY section of the January issue. Another useful purpose is to place the Basic program UP in memory to make space for machine code utilities "under the basic program". This would be useful if operating a program in the dock bank, because there would be about 4K of wasted space down there when PROG is moved to Chunk 3. But sometimes the rationale is more complex than the act of doing it, so let us get to the action. Since the program itself is a shortie, I'll ~~double space the listing and~~ make notes at the side. I'll call the program "mov" in a REM line so you'll know what I'm referring to.

```
  5 REM "mov" Basic & Vars Memo
ry Management
  10 INK 7: BORDER 1: PAPER 1: C
LEAR : LET a=PEEK 23635+256*PEEK
 23636: PRINT "This Basic Progra
m begins at theMemory adr of: ";
a
  20 LET b=PEEK 23627+256*PEEK 2
3628: PRINT '"The last byte of t
his Program isat adr: ";b
  30 LET c=b-a: PRINT '"The prog
ram length is: ";c
  40 PRINT '"To Move the Program
 UP in MemoryType CONT ENTER": S
TOP
  50 INK 7: BORDER 1: PAPER 1: C
LS : INPUT "Input PROG ADR?";pg:
 LET a3=pg-a: LET a2=INT (a3/256
): LET a1=a3-(a2*256): RESTORE 5
0: FOR x=23296 TO 23304: READ y:
 POKE x,y: NEXT x: RANDOMIZE USR
 23296: DATA 33,85,104,1,a1,a2,1
95,187,18
  60 CLEAR
```

Note VAR

Works AS IS with All Disk and Cassette Systems

Line 10 PEEKS SYS VAR PROG and assigns the address to var a.

Line 20 PEEKS the SYS VAR VARS and subtracts 1 to get the last byte address of the basic program.

Line 30 assigns the address of the last byte of ram to var c.

Line 40 waits for your decision.

Line 50 will be explained by steps. First an input is requested for the START Address where you want to move the prgram. Your input address is assigned to the var pg.

Next, var a3 is assigned "pg-a" which is the total bytes of the move- UP in memory. The number assigned to a3 will next be broken into the two numbers to use as POKE addresses for a "two byte" number storage.

Next var a2 will be the "poke number for the high address". Var a1 will be the number to poke to the low address.

Next is the RESTORE to initialize the DATA group later in the same line.

Next is a counter to place a MC routine in the Print Buffer area. Note that vars a1 (low adr poke) and a2 (high address poke) are placed in the data group. This MC program MOVES PROG up in memory to the address that was INPUT at the beginning of line 50.

The MC program is a variation of "MOVE PROG" given in the January issue. It can be used repeatedly to step a program up in the memory map. Try inputs of 30000 first, then 35000, 40000, etc. SAVE it to auto run at line 10.

3.0

Screen Files In Strings
— By: Bob HARTung —

In January SDU Bill discussed the use of SCREEN$ files as a means of conserving RAM in programs such as SMART TEXT. Each SCREEN$, of course, takes 6912 bytes in disk file space or two cylinders in SDOS. The following demo listing shows how up to the maximum of 704 characters in a 32-column text screen such as a menu may be stored in a string, then printed back on-screen either from RAM storage or after retrieval from disk or tape files.

No graphics or UDGs are recognized by the SCREEN$ (x,y) function but this can be used to advantage by having a subroutine to set up screen borders, etc. then retrieving screen files and using PRINT OVER 1; a$ as many times as you wish to change the text. PRINT AT x,y; a$(a TO b) could also be used to alter specific parts of a text display. It does take awhile to chew away at a screen initially and put each character in the string, but once this is done and the file is saved as DATA a$() the access time is for only one cylinder of disk space. Because only about 1/10 the access time is required as compared to a SCREEN$ it would even be feasible to sequentially access a series of such files with tape data storage if RAM capacity was running short in a lengthy program. Use your imagination for other applications.

SCREEN STRING DEMO

```
  1 LET P=1
  5 DIM A$(704)
 10 FOR N=1 TO 704
 20 PRINT CHR$ (RND*74+48);
 30 NEXT N
 35 PRINT #0;"Original display transfer to A$"
 40 FOR V=0 TO 21
 50 FOR H=0 TO 31
 60 LET A$(P)=SCREEN$ (V,H)
 65 LET P=P+1
 70 NEXT H
 80 NEXT V
 90 CLS
100 PRINT A$;#0;"A$ printed from VARS file"
105 PAUSE 100
110 SAVE /"SCREEN" DATA A$()
115 CLEAR
120 PRINT #0;"Re-loading data just saved"
125 LOAD /"SCREEN" DATA A$()
130 LET a$( TO 17)="                ": REM Demo of how b
lank  spaces in screen data will not  overprint graphics o
r UDGs in  border when OVER 1 is used
135 PRINT "   BORDER HERE   "
140 INPUT "": PRINT OVER 1;AT 0,0;A$;#0;"A$ from disk fil
e to VARS prtout"
150 PAUSE 300
160 STOP
200 REM GO TO 120 to repeat disk file display
9999 SAVE /"SCREEN" LINE 1
```

The Mysterious "DEF FN and FN" Functions

One Monday morning at the Timex Literature Department Mary was given a task that would normally been assigned to Gus, the Mathematician. Gus had his usual Monday morning problem. Now Mary majored in Greek Literature, not Greek Mathematics symbols. But, never the less, Mary was given the task of writing the part of Appendix A of the TS-2068 User Manual to explain such functions as ABS, ACS, ATN, "DEF FN", and "FN". Thus explains the reason why many TS-2068 users have not ever used the DEF FN and FN functions. The User Manual has good prose, but lacks some in detail.

The two functions DEF FN and FN are used together like bread and butter to store MATH FORMULAS in memory and then SOLVE the problems by INPUT of the missing factor. Take for example the formula C=PI*D for solving the circumference of a circle when d (diameter) is known. To set the formula into memory, a line of programming would define the formula as, DEF FN c(m)=PI*d. Or, to convert METERS to INCHES, the formula I=39.37*M would be defined in a program line as, DEF FN i(m)=39.37*m. The formula is in the DEF FN structure, and the SOLUTION is achieved by using the FN function, as PRINT FN i(m), or LET a= FN i(m), to assign to a variable.

The following little program demonstrates one way of using the DEF FN and FN functions.

```
 1 REM ** Using DEF FN and FN **
10 PRINT "Convert Meters to Inches"
20 DEF FN i(m)=39.37*m
30 INPUT "Diameter in Meters? ";m
40 PRINT FN i(m)
60 LET d= FN i(m)
70 PRINT "Compute Circumference"
70 DEF FN c(m)= PI *d
80 PRINT FN c(m)
```

SYNTAX: DEF FN is followed by a identifier, "i"in the first case, and "c" in the second case. Then the designator letter is inclosed (m). This also can be any letter. On the right side of "=" comes the formula to be solved. The un-known factor in the formula, "m" in the first case, and "d" in the second case, are fed into the formula by INPUT prompt. Line 40 gets the value computed with the "PRINT FN i(m)". Line 50 demonstrates that the FN solution may be directly assigned to a variable. This value of d is used in the second formula to compute the circumference of a circle.

The DEF FN formulas may be placed in a line of

programming to be initialized as one initializes a variables table (with GO SUB line #). The DEF FN formulas need not be repeated once they are initilized. Variable Not Found report will result if there are insufficient inputs for the formula to be solved. BJ

Jack In the Redwoods Heard From

Hello Jack! This SYNX critter is one thing we've been a 'lookin fer! A way to sneak by the syntax checker. Now we can put foreign disk syntax into a program line, save to cassette and send to a friend who dont know how to be a dunin it.

Or, how bout changing the syntax in a TS-2068 program to QL Basic, then shipping it out as a ASCII file to the QL sitting over on the next table? Heck, some of us may be writing IBM programs on the 2068. Thanks Jack, we'll find a use for it. Now why did ya disguise it by sayin its fer AERCO Disk users? Some of us are but others aint and it appears that it dont make no difference nohow which kind of disk we're using. And while I gotcha, why dint you mention that you have a lot of good software for the TS-2068, and anybody can get yore catalog for a Buck, which will be recouped on the first order?

Jack Dohany 390 Rutherford Ave Redwood City, CA 94061
(415) 367-7781
FEB. 22, 1988

*** SYNX ***

SYNX is a 46-byte relocatable MC routine for AERCO DISC users.
SYNX is short for SYNTAX CHECKER SWITCH.
SYNX is given into the public domain.

SYNX allows you to turn off the BASIC syntax checker when writing or editing BASIC lines... and to turn it back on. Syntax checking during program execution remains in effect.

Why? Perhaps you want to write a BASIC program that can be used on a non-AERCO disc system... perhaps on ALL disc systems. You may want to write a line like this:

```
500 SAVE *"TEST": REM for Zebra disc
```

Well, you CAN'T write it because it will fail syntax checking. But with the syntax checker turned OFF, you CAN write it.

SYNX works by changing some locations in the BASIC operating system... normally in ROM, but in RAM with AERCO DISC. A more detailed explanation is beyond the scope of this document.

Assuming you have the code on disc or tape as a .BIN or CODE file, it can be loaded wherever there's no conflict with other software. Let us say you want to load it at Loc 64000:
CAT "SYNX.bin",64000 or LOAD "SYNX" CODE 64000 will do it.

To SAVE the code:
MOVE "SYNX.bin",64000,46 or SAVE "SYNX" CODE 64000,46

*** HOW TO USE IT ***
Assuming the code is at 64000:
RANDOMIZE USR 64000 turns syntax checker OFF.
RANDOMIZE USR 64002 turns syntax checker back ON.

Here is the SYNX code as a decimal listing, ready to be poked into memory however you wish:

64000	24	64010	255	64020	24	64030	255	64040	0
64001	27	64011	0	64021	221	64031	1	64041	243
64002	24	64012	0	64022	197	64032	4	64042	237
64003	18	64013	205	64023	225	64033	0	64043	176
64004	0	64014	13	64024	1	64034	9	64044	251
64005	0	64015	13	64025	11	64035	17	64045	201
64006	0	64016	253	64026	0	64036	76		
64007	253	64017	54	64027	24	64037	14		
64008	54	64018	0	64028	5	64038	1		
64009	0	64019	255	64029	197	64039	9		

Of course, if you get a single number wrong, the silly computer is likely to crash when you attempt to use SYNX. So... SAVE before testing.

32

*** END ***

A REVIEW OF a Aerco FD-68 Utilities Program Disk

By: Syd Wyncoop

This is a nifty set of disk utilities for the Aerco FD-68, by Chia-Chi Chao. The program "Disk File Manager" will be reviewed. This program allows individual files to be copied, singlely or in groups, instead of the entire disk, as in the familiar Aerco command, <MOVE "A:=B:",>. Other functions: .. normal disk catalog: ..detailed disk catalog, which shows track and sector information: ..listing of occupied tracks on the disk: ..checking disk for bad sectors.

Having the ability to copy individual files is obviously very handy and has several less obvious side benefits. These additional benefits could all be grouped as one, more efficient disk operation. As you use a disk, after many read/write operations, there are lots of 'reclaimed' sectors lurking about and 'holes' in the directory tracks. When you copy individual files onto a freshly formatted disk, the files are placed on the disk in consecutive sectors and the directory does not contain any erased entries. This greatly speeds disk access and loading, particularly as file size increases. The Aerco disk copy function does just that, it copies all disk information to the new disk. This is very useful for debugging and/or recovery of a 'trashed' disk. You should always attempt direct disk writes, such as recovery, from a copy, NOT the original disk.

The detailed disk directory function provides you with the information contained in the directory which is usually not available to you. This is autostart line numbers, starting addresses, as well as track and sector information. This is similiar to reading a tape header. You can use the track/sector information to see how disorganized the disk is. Listing the occupied tracks also gives some indication of disk disorganization and how full the disk is. Also, as above, this information is useful for debugging and recovery purposes. Best of all, this information can be redirected to a printer, either the TS2040 or an 80 column printer! From my perspective, the check disk option has one serious drawback, it should have been a non-destructive test. As it stands, you must test all disks prior to use, as all data will be destroyed by the disk check. This is because the disk check effectively re-formats the disk. I should clarify, the disk check works fine. However, since I have encountered only about 6 bad disks in a library of over 700, I see little need to check them. Therefore, I usually wait until a problem occurs, at which time I want a non-destructive disk check. This is not bright on my part (and frustrating when a problem does develope) but it is easier and has proven to be effective.

As a bonus, the disk contained a nice Boot.Bas file loader program. You use the (mouse?) arrow keys to point to the file you wish to load. Autorun Basic programs may be loaded with or without autorunning them. Disk File Manager should be considered a very useful addition to your library of Aerco utilities. It is easy to use and it works!

Ed note: Disk File Manager ($13.00) is available from CHAI-CHI CHAO, 73 Sullivan Drive, Moraga, CA 94556-1209. Chai-Chi has several other Aerco FD-68 utilities, all inexpensively priced. Send SASE for product list.

A Universal Treatment for Bugs in the Program Area

I want to call your attention to John Oliger's article (see "Problems in using the new V2.4 MERGE Command"), because it has universal application. The scenario is, a Basic program picks up garbage that is stuck within the program space that it occupies in memory (PROG to VARS). It's kinda like one walking through the woods and not realizing that chiggers are finding a warm bod to invade. Junk stuck in the program area is just as hard to get rid of as them pesky chiggers. You can't see um and they're stuck like they're a piece of ya.

When you SAVE a Basic program that contains junk (chiggers) the junk SAVES right along with the program (at least chiggers eventually die). How does junk invade a Basic program? Well, one way is via a "bad load", where a loading error occurs after some, or all of a program has loaded. You've seen this in program loads that aborted, and when you try LISTing the program, it scrolls and scrolls with a screen full of "?" question marks. Thats junk (chiggers), pardner, and the whole mess is in the program area of memory. The computer is trying to list the program but the last program line contains junk that extends all the way up to the end of memory and the systems vars VARS has not been set. Of course such a gross invasion of the program area by bugs cannot easily be recovered from.

What John is talking about in his article is a "lesser invasion", kinda like intestinal parasites. The program lives and the patient might not even know that they're in there. Funny how a program can live a normal life while hosting such parasites indefinitely. But eventually there comes a time when the buggers are detected. In this case they were discovered when someone tried to MERGE to the space where the critters had set up housekeeping.

John gives us a very useful procedure to rid a program of bugs that have invaded the program area. I'm highlighting it here because it has universal usage. Now there are many different kinds of bugs. Sometimes you can get rid of um with RAID and other times you have to take somethin fer it. In this case John shows us how to cut um out. Disgusting, them parasites, eh?

Book Review: Dick F. Wagner
Title: Epson, Epson, Read All About It!
Authors: Julie Knott & Dave Prochnow
Price: $14.95
I.D. ISBN 0-201-11640-5
Publisher: Addison-Wesley Publishing Co. Inc.
Pages: 275 Date: 1985

The authors have developed a guide for Epson and Epson work-alike (compatable) dot matrix printers that provides the user with more information on printer applications than most manuals give. Where printer manuals often devote a part page on each printer code and sometimes a program to explain its operation, these authors provide uses and application programs.

This book explains such mundain things as the proper way to load in the paper, discusses interfaces, explains what happens on printer initialization, how to program for different computers, and many other pertinent subjects. Within each chapter a table of new function codes used in that chapter is presented. This table shows the CHR$ codes and the ASCII codes corresponding to each action. LPRINT and CHR$ codes are used in all examples

The book is divides into 4 parts and 11 chapters as follows:

Chapter 1: Printing with dots
Part 1: Selecting printers and connecting to computers
  Viewing the printer field
  Joining of computer and printer
Part 2: Epson printer capabilities
  Character pitch
  Character sets
  Downloadable & custom characters
Part 3: Epson dot graphics
  Plot & graphics
  Printhead placement
  Programmable Graphics
Part 4: Patching software and hardware to Epson
  Installing software
  Joining hardware
Appendicies: Function codes
  ASCII codes
  Dip switch summary
  Character style chart
  Epson work-alike conversion chart

Some interesting programs given include a printer test, typewriter(print as you type), print out the International character sets, print double height characters, design print fonts, design custom graphics, setting margins, horizontal and verticle tabs, graphic line designs, circles, pie chart, and a program (GMAKER) to create graphics in any of seven graphic modes. There are various short demonstration programs for specific control codes. Progragms are in MBASIC and duplicated in Applesoft BASIC but are easy to convert to Sinclair BASIC. Screen dumps are not covered.

One subject not not discussed but important to some computers is how to cancel an unwanted line feed caused by the computer cable connection and software. The secret is to disconnect the printer pin 14 cable which eliminates the AFXT low signal. In some cases the dip switch does not seem to control the line feed.

About 70 pages are devoted to various appendicies. An interesting table covers the function codes used by Epson and compares codes for C. Itoh 8510BP and 8510SCP, Olivetti PR2300, and Star Micronics Gemini 10X printers. Character shapes used by these printers plus Epson FX, LQ, RX, and MX models are given.

While Part 4 would be the least useful part of the book, it does give some interesting details on specific products not useful to TS users. The reader can compare word processors available to TS users with a few used with other computers. Ours seem to be very comparable to the big name programs. About 70 pages are devoted to this subject.

Soft back and ample margins make this book easy to use at the computer. This is the only reference book I have seen devoted to Epson and work alike printers.

17

PROBLEMS IN USING THE NEW V2.4 "MERGE" COMMAND

It has come to my attention that several JLO SAFE users have been having some trouble using the new SAFE "MERGE" command under certain conditions. This note will detail the results of my investigation into this problem, as requested by several Oliger Disk System owners.

I have had reports from a few users that indicated a problem may exist in SAFE causing the MERGE command to sometimes not work. The users reported that SAFE would act like it had MERGED the Basic code, return with an "OK" report, but when they would LIST to look at the additional code it would simply not be there. Repeated attempts to MERGE the BASIC into the same program would give the same result. However, when asked for a sample disk that contained the programs that resisted MERGE, nobody could provide me with the sample for one reason or another. Repeated attempts to cause the problem myself would always fail. I couldn't get it NOT to work!

Finally one customer called with a report of the problem, and he had it doing it right then, before our very eyes (and ears)! He told me he would make a copy of the disk having the problem and get it off to me to work with.

I recieved the disk and quickly went to work. The main program on the disk contained Bill Jone's (are you listening Bill?) "SMART DAISY" Basic program, and the Basic code to MERGE was a ten line or so eprom programming subroutine that started at line 9980. The last line in SMART DAISY was line 9520. I loaded the main program and then tried to merge the programming subroutine. As reported, the drive and SAFE ACTED like it had been merged but I sure couldn't see it there!

Further investigation found that I could not get a good merge after Smart Daisy was loaded even after the CLEAR command was used and even when Smart Daisy itself was removed using the DELETE , command!!! I found that <PRINT FREE> would return only 38289 bytes free after SMART DAISY had been loaded and then removed with CLEAR and DELETE , but on power up before any loads the computer had 38652 bytes free. I had somehow LOST 363 bytes of memory by loading SMART DAISY and then getting rid of it!

After running HOT Z AROS to find out just what was going on, I found that the pointers to the variables area and program area were not the same as they are when the computer is first powered up. Looking at where the computer thought the program was showed a lot of gibberish and portions of one of the screen copy routines used on the cassette supplied with the Oliger Printer Interface.

I checked out several copies of Bill Jone's SMART TEXT and found that EVERY ONE of them from the very first was missing those same 363 bytes. They ALL had the same gibberish where there should have been program lines at the end of the basic.

Evidently somewhere along the line (very early) Bill's program had been corrupted, either through glich, cassette data error, stray POKE(s), or whatever. The MERGED code is actually there, but Basic can't find it to LIST or RUN because the gibberish before it in memory confuses it.

Although all of Bill's Smart Text series programs seem to have this same corruption, there are likely to be other programs found that are also like this. Because of the way Basic programs are stored, saved, and loaded on the 2068, once something like this occurs and the program re-saved (using either cassette or disk), it will forever exist because as far as the computer is concerned it IS a part of the Basic program, even though it can't use it or LIST it.

I hope I haven't bored everyone to sleep by now. I've attempted to go through the entire process of how I try to resolve a problem like this reported to me. Sometimes I beat my head against the wall all day long and never get a problem solved. If you have a real strange problem with a certain command or function in SAFE just some of the time, PLEASE put it on disk as a sample and send it with your letter. It can REALLY be a big help!

So, how do you know if a program has this problem? If you cannot use the SAFE MERGE command with it then it is very likely. But the REAL test is to use the command <PRINT FREE> immediately after power up and again after the program is loaded and the removed using <CLEAR :DELETE ,>. If the numbers don't match you can be fairly certain that the program does contain corruption (unless it had self-run and used the CLEAR N command). I will now detail step by step how to fix this:

1) FIND AMOUNT OF PROGRAM MEMORY CORRUPTED:

A) Write down result of the statement <PRINT FREE> immediately after computer power-up. (38652 on standard 2068 w/o any cartridges installed)

B) Load corrupted program. Save variables to disk with the command <SAVE /"vars" VAL>. Clear both program and variables from the computer with the command line <CLEAR :DELETE ,>. Now write down result of the statement <PRINT FREE>. (38289 for Smart Text)

C) Subtract the number written down in 1B above from the number written down in 1A. Write this number down. (363 for Smart Text) This is the amount of memory corrupted.

2) FIND CORRECTED BASIC VARIABLES POINTER:

A) Re-load corrupted program and remove the variables again with the <CLEAR> command.

B) Enter the command <PRINT PEEK 23627+ PEEK 23628*256>. Write down this number.

C) Subtract the number obtained in 1C from the number you just wrote down in 2B. Write down the result.

3) FIND NEW VALUES TO POKE:

A) Divide the number obtained in 2C by 256. Round down to the next whole number and write this down. (MSB)

B) Multiply the number just written down in 3A by 256. Subtract this product from the number obtained in 2C. Write down the result. (LSB)

4) SET THE VARIABLES POINTER CORECTLY AND CLEAR THE CORRUPTION:

A) Set the variables pointer with the command <POKE 23627,LSB: POKE 23628,MSB> with "LSB" and "MSB" replaced by the numbers written down in 2B and 2A respectively.

B) Clear the corrupted memory by using the <CLEAR> command again.

5) GET IT ALL BACK ON DISK:

A) Re-load back in the old variables with the command <LOAD /"vars" VAL>.

B) Re-save the repaired file with the regular <SAVE /"Filename"> command.

The "cleaned-up" program should now allow MERGE to operate and be shorter to boot!

John L. Oliger 2/88

SAFE QUESTIONS AND ANSWERS

This column will answer questions of general interest about the Oliger 2068 Disk I/F and JLO SAFE DOS. If you have questions you feel other users of this system would like to see answered, send them to me at: 11601 Whidbey Dr. Cumberland, IN 46229. Because my schedule can at times be simply too demanding to always find the time, I cannot promise to ALWAYS have this column ready for each new issue of Safe Disk Up-date. I will only do the best I can...

-John T. Oliger

What is the latest version of SAFE and what new features does it provide?

The current version of SAFE as of today (1/25/88) is V2.41. The last commands added to SAFE were the MERGE command, the SAVE //"Filename" command, and the NMI "N"EW key function. The MERGE command is a simplified but very fast merge that appends additional Basic code and variables onto an existing Basic program. The SAVE //"filename" command is an alternate method of saving to disk that by-passes the "FILE EXISTS!! 5 SECONDS TO ABORT" warning message. It is intended for use where a file overwrite is expected and the delays caused by the warning message are not desired. The NMI "N" key function allows a complete reset of the computer very easily w/o having to turn off your computer. It is like adding a reset switch without physically installing the hardware.

How much does it cost to update my SAFE V2 eprom?

I charge $5.00 if you return a SAFE eprom to be recycled or $10.00 if you do not. This is for all lawful owners of SAFE V2. SAFE V1 owners can upgrade to the latest SAFE V2 by sending $10.00 in addition to the above. This is a one time payment for the V2 software. I feel the $5.00 or $10.00 is about the cost for post/pack/handling and possibly the cost of another eprom. Thus, once you have SAFE V2, you are NOT charged for it again. All prices listed above include postage. If you wish defaults of other than 40 track, double sided, 6ms head step, ASCII COPY /, then you need to specify when ordering.

I own the Oliger 2068 Disk Interface W/JLO SAFE and use an AERCO printer interface with the Olivetti PR2300 ink jet printer. I have tried to use the built in SAFE printer driver and screen copy routines with my set-up without success. Can you help?

The Olivetti PR2300 printer and Aerco printer interface, used together, are NOT directly compatible with the printer driver and screen copy routines built into SAFE. The Oliger 2068 Printer I/F and Olivetti or most other combinations of Aerco I/F and printer work without a hitch. The trouble with the Olivetti/Aerco/Oliger Printer driver combination is caused by the way the Aerco I/F itself requires the software to finish toggling the printer's STROBE NOT line by reading the port again immediately after writing something to it, which the Oliger software does not do. The Oliger I/F does not require the software STROBE NOT read toggle. The fix for this is a simple modification to the Aerco I/F board itself, which will make it self-toggle and as such cause it to work with any combination of driver/printer you may have.

To perform this modification to your Aerco Printer I/F:

1)Cut the pc board trace going to pin 11 of the 74LS10 chip, right by pin 11.
2)Install a jumper wire from the 74LS10 pin 11 to the 74LS02 pin 5.

I have the latest version of SAFE (V2.41) and find that sometimes the new "N"ew NMI reset function does not work, but other times it does. It seems it usually won't work after it has previously been used. Is something wrong with my board or new SAFE eprom?

There is nothing at all wrong with your hardware or firmware. This is just a peculiarity in using this function in the way the hardware is designed. It is caused by holding the NMI button down too long when pressing it for the new reset. The correct procedure for using the new reset function is to hold down the "N" key and then to give the NMI button a short press (just a tap) to cause the reset. If the NMI button is held too long, the reset function will simply not work the next time it is tried, unless an error or anything that activates SAFE happens before then. When "locked-out" like this, no harm other than it not working again will occur.

After a <BREAK> key press during disk operations, I find the only way to get back to default settings is to turn both the 2068 and disk drive off then back on again. We need a way to clear SAFE and anything the 1770 is doing without losing everything. Any ideas?

You're assuming incorrectly here. If you BREAK during a disk operation and a return to Basic is made, SAFE has already taken care of all this as much as it can! SAFE is in good shape to do whatever you want, and has sent a command to the 1770 disk controller to abort whatever it is doing. What you are seing happen is that the 1770 chip itself HAS stopped everything it was doing, EXCEPT that it will keep the drive motor turned on until it sees several rotations of the disk. If the drive selected does not contain a disk or has its door open, the 1770 will leave the motor on forever because it hasn't seen the disk turn its allotted number of times. All you need to do is insert ANY disk into the drive or use the <LET /D=n> command to select a drive that IS on-line with a disk installed. Then the 1770 will see the disk turn a few times and stop the motor. Any write operations you might have started and broke into WILL NOT OCCUR because SAFE aborted them and told the 1770 to abort too. ALL disk systems that use the motor control function of the 1770 have this same quirk, because this is how the 1770 itself does it. There is NO software command to stop the motor, just let it see the disk turn and it will stop itself!

I get occasional "TOOTS" when using the MOVE/ command within a Basic program, but I don't seem to when I use it in direct mode. It appears to toot when the source light is on, as if SAFE is trying to read before the head has fully settled. What's wrong?

Your disk set up is marginal if you get these warning toots, period. Try relocating your drives and/or rerouting the disk controller cable. If you tried the slowest head step speed and it STILL does this, then the problem is NOT in SAFE trying before it should. Most problems of this sort are found to be caused by interference and are usually cured by rearranging the drive and cable's placement. The problem occurring within a program rather than in immediate mode is coincidental. SAFE has no idea if the statement is immediate or within a

program and the exact same code is executed either way. More likely you do the move more often using the program you mention so the odds of the error occuring is greater.

You say SAFE will allow from 10 to 255 tracks per side of the disk? What kind of drive, if any, allows more than 82 tracks per side?

None of them do, at least right now. But if tomorrow the drive manufacturers doubled the amount of tracks allowed per side from 80 to 160, SAFE would be able to use the extra space immediately! It happened before when they went from 40 track to 80 track drives. It is VERY possible it will happen again in the future as drive manufacturers become better at making the disk stepper motors and heads. As far as I know, SAFE is the ONLY DOS around that is flexible enough to allow this. Most others would require substantial revisions if it would be possible at all!

I see no real benefit in having the variables load in along with the MERGEd program, and I think it would be better if the LOAD /"name" VAL command cleared out the old variables before loading in the new. Any comments on this?

Because of the way MERGE was added to the SAFE code in the eprom, and because a regular Basic program DOES contain variables in a normal SAVE on the 2068, it used MUCH less eprom code space to have MERGE load in the variables too, rather than just the program. Any subroutine saved to disk to later be merged into a master program should have its variables removed by using the CLEAR command before saving to disk, unless these variables will be needed by the main program. The reason LOAD /"name" VAL doesn't erase existing variables is to make the command more flexible. Use can always CLEAR and the load a VAL type file to get only the new variables the way it was implemented. If it always cleared the old variables you would HAVE NO CHOICE and flags used by a running program that might be needed would be lost.

SAFE supports several different printers with its COPY/ command. I feel that we don't need several different copy routines, but only one. Why don't you remove these routines to open up more code space to implement new commands?

Most users of this disk interface feel that only one or maybe two of the several copy routines are needed in SAFE. Only trouble is, each feels that the one that supports his printer is the one that's needed, not the other. The next guy thinks it should be a different one. It would be much too complex to install a different copy routine in each different SAFE eprom I program. This would waste more time than if I devoted the same energy to making portions of the code more efficient and opened up a little room this way. The simple fact is that most people LIKE the ease of using the COPY / command or the NMI copy function. I do myself! If I took it out, I feel there would be a definite loss...

What is the purpose of the slide switch (SW1) right next to the NMI button on Disk Board B? Only reference I have come across is step 5 on page 4 of the Oliger Disk I/F User's Manual.

This switch is only on the "REV A" disk board B. It was intended to be used to switch between the two 8K "halfs" of a 27128 eprom. The idea was to have my own JLO SAFE DOS on one half (active when switched in the right "SA" position) of a 27128 eprom and Ray Kingsley's OKDOS (active when switched to the left "OK" position) in the other half. If switched into SAFE from OKDOS, you were to use the RESTORE /S command to initialise SAFE without a reset. Ray was going to have a similar command to initialise OKDOS when going from SAFE. Only trouble with all this is that Ray never did finish his OKDOS, and its been a year and a half since I've heard from Ray at all. I am sure now we will NEVER get a complete OKDOS from RAY, which is really a shame because he had most of the core of the DOS complete, needing only to tie in the DOS to Basic. Oh well, we all go on. I believe Ray didn't think he would get back financially what he would have needed to put into the DOS in his time, to make it worthwhile. He may NOT have...

SAFE BUGS that are not BUGS- ed

Seems that I have a blank page to fill, so will give this gem to the SAFE SDOS section. Ole John must dread getting letters from my house, but he's right prompt on answering them. Some of the questions that he has answered in this section came from you know who. And you know what? John does have answers! The rest of this will outline a problem that 80 track disk user may have encountered as I did. What happend was, I got a Disk$ back that had several files glitched up. Checking the balance of my pre-recorded disks, I found all of the 40 track disks were OK, but ALL of the 80 track disks had the same glitched files. Now that aint good for me reputation! Thankfully only one of them 80 track buggers had been sent out. This sneaky bug only occurs when the 80 track disk "is more than half filled".

I got my Master 80 track disk out and began to copy it. Now my drive set up is Three 80 Track disks as drive 0, 1, and 2, with a 40 track as drive 3. So it is easy to use drive 0 and 1 for copying with the MOVE/ function. How do I copy 40 track disks? Well I just remove the DATA cable from the SAFE controller, and plug on the data cable of another drive system that has 4 40 track drives and use MOVE/.

But, using the Master 80 track disk, the first MOVE/ resulted in the same files being glitched. Funny thing, when I LOADED a program from the Master disk and did this number: LET/d=1: SAVE/"title", the warning toot would sound, then the program would SAVE A-OK. So, your scribe did MANUAL transfers of programs to about 10 disks. Ah Ha! I figgers, I got ole John now! So I packed up a copy of the master and a glitched up disk and sent them off to him. John's reply came bouncing back. Now I'll quote:

1. The trouble you are having with MOVE/ is caused by how this command itself works. MOVE/ is a carry over from SAFE V1. It depends upon the Number of Tracks to be MOVED and the Number of Sides to be MOVED "as set with the LET/t=n and LET/s=n commands". The MOVE/ is using the DEFAULT value of 40 as is set in the SAFE EPROM. So, it is moving only the first 40 tracks to the destination drive. But, the entire Catalog, and the Pointers are being moved "because they are stored on track zero". But the DATA that is being pointed to will not be there after track 40.

2. The cure is easy. Just preceed your MOVE/ command with LET/t=80 when MOVING/ between your 80 track drives. You can think of the MOVE/ command as being dumb because you have to tell it to disregard the default parameters (40 track and 2 sides), while MOVE/"file name" is smart and can figure this out. I need to explain this better when I revise the manual. END QUOTE.

Durn! I thought I had him! But ain't John nice? He could have just told me I was dumb instead of saying his MOVE/ was dumb. Oh well! Funny thing is, I had made a DISK COPY program that repeatedly uses MOVE/ to copy disks between any track combination of drives AND I INCLUDED the LET/t= and LET/s= commands. Then when doing the MOVE/ in the direct mode the LET/ was left out. I didn't think I needed the LET when operating with two drives having te same number of tracks.

I have some more space to fill so will do it with programming. Here is a way of using the disk catalog as a MENU to select a program to load. <100 CAT/: INPUT "Key in TITLE of Basic Program";a$: LOAD/a$ >. You can embellish this with optional selection of a different drive, as: 100 INPUT "Enter drive number";a: LET/d=a: CAT : INPUT "<1> Basic or <2> Code Program?";b: INPUT "Enter Title only";a$: If b=1 THEN LOAD/a$ > <110 IF b=2 THEN LOAD/a$ CODE: GO TO menu >. In both examples, a INPUT prompt is printed to the Catalog Screen at the bottom area and you are using the catalog data for information.

Here's another bug that ain't a bug. Quite often I'll FORMAT a 40 track disk to 41 tracks in order to cram more programs or data into the disk. Once I tried MOVE/ to copy such a disk, and it didn't work. The reason was that the destination disk was formatted to 40 track. Thats like trying to walk on two straight lines at the same time. Incidentally, you CAN FORMAT a 40 track disk to more than 42 tracks, say 50. The disk directory will show the DISK FREE space as if you have 50 tracks, but the figure will be erroneous. All that I have ever been able to stuff into a 40 track disk is the contents of 42 tracks. The diskette manufacturers only guarantee the space for 40 tracks. On 80 track disks one can usually format 82 tracks (80 guaranteed). The practice of formatting extra tracks isn't advised because the read - write head is extending beyond the disk drive mechanical range as well. 41 and 82 might be safe enough, but beyond that is pushing it for a little bit more data space. Why not wait until one of them new 250 track disks comes out. Then we can use the range of SAFE that John provided, though the disk capability did not exist.

***** TS-2068 COMPUTING WITH LARKEN DISK *****

BY LARRY KENNY

** LKDOS Language Discussions

** Questions and Answers

** Machine Code Routines

** Ram Disk

** Integrating Disk Drive Systems

The LARKEN DISK Section

Mr. Larry Kenny, RR-2, Navan, Ontario, Canada K4B 1H9, is the owner of LARKEN ELECTRONICS. Larry is an Engineer and has developed the Larken Disk Drive System for the TS-2068. Other significant developments by Larken is the abbreviated Disk Driver Cartridge which interfaces to other Disk drive system. This is called "The LKDOS Cartridge" and it provides a "Dual DOS" when used with either the Oliger SAFE or the Aerco FD-68 disk drive controllers. One more significant development by Larry is the Larken RAM DISK, an extra memory device which "acts as a disk drive" for storage of data.

Larken Electonics is one of our most valuable Cottage Industry supporters of the TS-2068 and we need Larry to continue to develop the hardware additions that only an engineer who knows the TS-2068 can provide. This section in UP-DATE is devoted to providing information about the Larken Disk System(s) in the same manner as we provide the section titled "Computing with SDOS, by John Oliger. These two Cottage Industries, The Oliger Co., and Larken Electronics, form the nucleus of our TS-2068 Disk Systems support because of their continuing up-dates of their systems and their constant quest to produce new and inovative hardware systems.

This section is provided as a forum to support all aspects of the Larken Electronics system users. Larry has promised his inputs, and already UP-DATE has had outstanding article support by two avid Larken Disk users, George Chambers and Bob Mitchell. The more who joins in, the better the support. The Larken Disk systems began late, after two others had quit, leaving the users without support (see orphans). Just now, the TS-2068 users are finding out about what Larken Disk has to offer, to both the first time disk user and to users of other disk systems who want a inexpensive "second DOS".

Accessing LKDOS from Machine Code by Larry Kenny

The dos is easy to access from machine code using the internal jump table starting at address 120 in the cartridge.
The jump table is a list of 20 or so entry points to the most often used sub-routines in the dos.
Any changes or different versions of the dos in the future, do not affect these addresses, so utility programs dont need to be modified incase a different version of Lkdos is ever used.
Any machine code program (or Basic program) written for Lkdos is compatible with any Lkdos cartridge useing a different disk interface, (Aerco, Ramex or Oliger).
The routines can be used to load or save complete files or just load or save 1 block etc.
A complete list of all the Subroutines and variables in the Lkdos is supplied with the Disk Editor package. ($10.00)
Note - all numbers in the programs below are in decimal

********** Two Small Utility Programs for LKDOS **********

```
-- Addresses and dos routines used by  FREEBL and MOVCAT --
 data   EQU 8328    - Address of data in the disk buffer
 curtrk EQU 8221    - block number for TRACK to seek

-- Commonly used subroutines in the Lkdos jump table --
 SAVEBF EQU 120     - Save the buffer to current block
 LOADBF EQU 123     - Load the buffer from the cur block
 TRACK  EQU 126     - Restore to track 0 the seek (curtrk)
```

FREE BLOCKS

This routine allows you to find the number of free blocks with out displaying the catalog.
It will return the number of free blocks in the BC register, so it can be accessed from basic with PRINT USR FREEBL or LET x=USR FREEBL .

```
00010 FREEBL DI
00020        CALL 98          - turn on the Lkdos cartridge
00030        LD A,0
00040        LD (curtrk),A    - move the disk drive to
00050        CALL TRACK       - block 0  (side 0 track 0)
00060        CALL LOADBF      - load disk to buffer
00070        LD C,0           - set counter to 0
00080        LD HL,data       - set HL to start of track-map
00090 USED   INC HL           - first block always used anyway
00100        LD A,(HL)
00110        CP 255           - 255 marker means 'end of track-map
00120        JR Z,EXIT        - if so then exit
00130        CP 245           - 245 marker means 'block used'
00140        JR Z,USED        - if used then move on to next one
00150        INC C            - must be a free block,update count
00160        JR USED          - continue searching for free blocks
00170 EXIT   LD B,0
00180        LD A,(98)        - turn cartridge off
00190        EI
00200 RET
```

MOVE CATALOG to UPPER RAM

This routine loads the catalog from the disk, and then moves it from the dos ram to address 50000 where you can search or examine it from basic.

```
00010 MOVCAT DI
00020        CALL 98          - turn on cartridge
00030        LD A,0           - move the drive head to
00040        LD (curtrk),A    - block 0
00050        CALL TRACK
00060        CALL LOADBF      - load disk to buffer
00070        LD HL,data       - move the catalog
00080        LD DE,50000      - up to high memory
00090        LD BC,5090
00100        LDIR
00110        LD A,(98)        - turn off the cartridge
00120        EI
00130 RET
```

***** THE DISK DRIVE ORPHANAGE ****

FEATURING

THE RAMEX MILLENIA K AND TIMEX OF PORTUGAL

DISK DRIVE SYSTEMS

"Let us feed the good Orphans"

The Hungry Orphans

There are two TS-2068 Disk Drive Systems that are still in use despite the fact the their manufacturers have discontinued support of the systems. These are The RAMEX MILLENIA K, and the TIMEX of PORTUGUAL Disk drive systems. Information sources for these disk drive systems are scarce and are drying up as time goes by.

UP-DATE will publish every bit of information about these "orphan" disk drive systems that we can lay our hands on, because few articles can be found in other publications to support the users. To make space for supporting these systems we will go to a small print that may not be comfortable reading. We want INFORMATION VOLUME about these systems. The users will appreciate that more than they will "scarcity but larger print". Take a tip- each of the pages will "blow up" to exactly four pages of letter size paper that can be cut for a notebook. Any copy shop can do that for you for about a dime a page.

As is the case in this issue, much of the programming given in "Disk Specific" articles consists of useful routines that are adaptable to all other disk systems and cassette use, when one sifts out the disk syntax. So, all other TS-2068 users BE ALERT and dig out your binoculars to catch the universally usable programming.

UP-DATE invites all inputs to the "Orphanage Disk" section by clubs and individuals who want to help keep these systems alive. In particular, the users need information about REPAIR SOURCES. They also need EXCHANGE of PROGRAMS. ALL useful information received will be published (albiet, small type). Input articles should be on letter size pages and typed in ordinary PICA 10 CPI type, which will be reduced to what you see in the next pages. The goal is MAXIMUM INFORMATION!

This issue of UP-DATE presents a good source for RAMEX (SPDOS) information in Munson Cookayne. Munson is also a source for software. So you can think of the package of articles by Munson as being dual purpose, FREE information and Source information. Viva la RAMEX Users!! Viva la TOS Disk Users!! Both groups contribute to Viva la TS-2068 Users!!

A LITTLE ABOUT ME
by: Munson H. Cockayne, Jr. BSCS

Well you know my name but can you pronounce it? The last name is pronounced like "oocaine". It seems that in middle English "coc" was pronounced "coe" (long O) as is witnessed by the liqueur known as "Coc's" that is still popular in England. Enough about that.

I am a native Floridian, second generation. I have written several articles and letters for TDM, SPDOS SURF, and now TS 2068 UPDATE. Among my credentials is a BS in Computer Science and I am working on my MS at present. You may or may not have heard of the University of Central Florida (a state university) but that is where I go to school and offers one of the two PHD programs in Computer Science in the Southeast. Being a graduate of this program is my long term goal.

My first computer is a ZX81 kit that I got by mail order. Putting that one together and seeing it work really got me started on a new career. I never thought that I would become a computer scientist simply because of project kit!

Some of you have bought software and hardware from the small company my brother, Robert, and I run. It is called K.I.S.S./computers. There are three (even smaller) divisions:

1) K.I.S.S./softstuff (software).
2) K.I.S.S./hardstuff (hardware).
3) K.I.S.S./research (from whence the hardware and software come).

Most of the money earned goes to the third department to keep the TS line alive and thereby keep our business going. It makes sense to us and is a great source of fun. Yes, some people derive fun from painstaking work. We seek to fill the niches too small for the "big guys" to get into (not enough profit).

Computers are a big part of my life but my mainstay in life is my family. My wife, Chee, is everything I always thought a wife would be and then some. Our three children (Tina, Sandi and Trey) delight us with their company.

My philosophy is simply, "Life is an adverture. Make it the best sort for yourself and any you meet along the way." I am sure that there are many of you with essentially the same philosophy. My favorite catch phrase is "Keep It Simple Stupid! ".

I hope that was brief enough to keep you from going to sleep. Bill promised that he would edit it for me.

SHELL
by: Munson H. Cockayne, Jr. BSCS

I am sure that some of you read my article on this subject in TDM but I am also sure that you will find this expansion of the concept a useful one. The listing in this article is a 'shell'. It allows you to COPY or ERASE any or all of the files on a disk.

Let me pause to define the term 'shell'. A shell is a wrap around the Disk Operating System, DOS, just like a clam's shell wraps around it. Only, in this case, the shell not only hides the contents, but it also makes the contents simpler to use.

Now for the program, the first part asks you what you want to do in a series of questions. You never even have to hit enter.

The second part of the program creates a sequential file using the screen channel. This redirects the screen output to a sequential disk file. While this part runs, only disk activity will be noted until the end where the filenames are read back into a string array. Here you will hear rapid keystrokes as the input comes from the disk file.

The next part lists the filenames to the screen. As you mark files for action, they are converted to inverse video. When you are done, the requested action is taken on the files you indicated.

In operation, the number keys, 5 to 8, are used to control the cursor. The direction corresponds to the arrows above the keys. The "M" key marks a filename for action. The "U" key unmarks a file. The "N" key causes the next page (if all the files won't fit on one page) to be displayed. The "D" key tells the program that you are through marking files. A safety is installed before an ERASE command is followed. Just indicate your decision by pressing "Y" or "N".

This operation is very simple. After all, the operation of a shell should be simple or it isn't easier to use than DOS. It should accent the power of the DOS not take away.

Lines 1 and 2 are for those of you who have SPDOS V1.1 and wish to make this an AUTO execute file. All others may omit these lines.

```
  0 REM                          **********************************
                                 *                                *
                                 *             SHELL              *
                                 *       COPYRIGHT @ 1987         *
                                 * Munson H. Cockayne, Jr. BSCS   *
                                 *                                *
                                 **********************************
```

THE HISTORY OF THE RAMEX MILLENIA K
by Munson H. Cockayne, Jr. BSCS

There once was a little orphan computer named TS 2068. Like all orphans, it was not responsible for being an orphan. In fact, it was well designed and a pretty little thing. However, its parent, Timex, decided that raising the infant through proper marketing and support would not be a financially rewarding experience. So it was abandoned before its first birthday. Timex is alive and marginally sound but I for one have not been able to use their watches since that fateful day.

Along came the carpet baggers. Ramex was just one of many of this specie. They sought to exploit the poor orphan by making grandeous additions. When the orphan was unable to support their standard of living, they abandoned their addition.

None of these wonderful people ever made an effort to simply expand their arena of endeavor the way so many other computer oriented companies were doing. No, if they couldn't have all their eggs in one basket, they would put them all in another basket and too bad about the old basket.

Maybe this is a little cynical but that is the way I see it. Too many trying to make too much money from too few. Well now it is just us enthusiasts. We can and will keep the orphan alive until there are no more parts to fix them. Maybe by then we will know enough to be able to design replacement parts ourselves. Companies are proving every day that chips can be reverse engineered without breaking any patents. As long as we support each other and don't get greedy, we can and will keep this computer alive!

```
  1 ON ERR GO TO 1: INPUT "Press ENTER";c$
  2 ON ERR RESET
 10 PRINT #1; "1) ERASE"'"2) MOVE"
 20 PAUSE 0: LET c$= INKEY$
 30 INPUT ;: PRINT #1;"The files are on drive?": PAUSE 0: LET
r$= INKEY$ : LET d1= CODE r$-48: IF d1>4 OR d1<1 THEN GO TO 520
 40 IF c$ ="2" THEN INPUT ;: PRINT #1;"To drive?": PAUSE 0: LET
r$= INKEY $ : LET d2= CODE r$-48: IF d2<1 OR d2>4 THEN GO TO
540
 50 INPUT;
100 DIM t$(143,12)
110 CLS
120 PRINT #4: OPEN #2," CAT ": PRINT d1
130 PRINT #4: CAT "",: PRINT d1
140 PRINT # 4: CLOSE #2
150 PRINT #4: OPEN # 5," CAT ": PRINT d1
160 LET cnt=0
170 FOR i=1 TO 10: INPUT #5;f$: NEXT i
180 INPUT #5; f$
190 IF f$(2)=" CAT " THEN GO TO 220
200 LET cnt=cnt+1
210 LET t$(cnt)=f$(2 TO )
220 INPUT #5;f$: IF LEN f$=0 THEN GO TO 250
230 INPUT #5;f$
240 GO TO 180
250 PRINT #4: CLOSE #5
260 PRINT #4: ERASE " CAT ",: PRINT d1
600 POKE 23658, 8: REM Shift to all caps mode
1000 FOR p=1 TO cnt STEP 40
1005 FOR i=0 TO 19
1010 IF p+i <= cnt THEN PRINT t$(p+i);
1020 IF p+i+20 <= cnt THEN PRINT TAB 16;t$(p+i+20);
1030 PRINT 1035 NEXT i
1040 GO SUB 2000
1050 CLS
1060 NEXT p
1100 FOR i=1 TO cnt
1120 IF CODE t$(i)=20 THEN IF c$="1" THEN GO SUB 3000
1130 IF CODE t$(i)=20 THEN IF c$="2" THEN PRINT #4: MOVE t$(i,3
TO ),"": PRINT d1,d2
1140 NEXT i
1890 POKE 23658,0: REM Shift to normal mode
1900 IF c$="2" THEN PRINT #4: CAT "",: PRINT d2
1910 IF c$="1" THEN PRINT #4: CAT "",: PRINT d1
1990 GO TO 10000: REM Just stops without stop error report
2000 LET column=0
2010 LET row=0
2020 GO SUB 5000
2030 PAUSE 0: LET r$= INKEY$
2035 LET index=p+(column=16)*20+row
2040 IF r$="7" THEN IF row THEN GO SUB 5100: LET row=row-1
```

Cont top left

Shell

```
2050 IF r$="6" THEN IF index+1 <= cnt THEN IF row<19 THEN GO SUB
5100: LET row=row+1
2060 IF r$="8" THEN IF index+20 <= cnt THEN GO SUB 5100: LET
column=16
2070 IF r$="5" THEN GO SUB 5100: LET column=0
2080 IF r$="N" THEN RETURN
2090 IF r$="D" THEN LET p=143: RETURN
2100 IF r$="M" THEN IF CODE t$(index) <> 20 THEN LET t $(index)=
CHR$ 20+ CHR$ 1+t$(index)
2110 IF r$="U" THEN IF t$(index,1)= CHR$ 20 THEN LET t$(index)=
t$(index,3 TO )+" "
2120 GO SUB 5100
2130 GO TO 2020
3000 PRINT #1;"ERASE: ";t$(i,3 TO );"?": PAUSE 0: LET r$= INKEY$
: INPUT ;
3010 IF r$="Y" THEN PRINT #4: ERASE t$(i,3 TO ),
3020 RETURN
5000 PRINT AT row,column; OVER 1; FLASH 1;" "
5010 RETURN
5100 PRINT A T row,column;t$(p+(column=16)*20+row)
5110 RETURN
```

There are a few notable limitations to this little shell. Line 100 DIMensions t$ to 143 strings. Even though there are 144 filenames possible with SPDOS, our little CAT file will use one of these. It is skipped by line 190 since it will not exist by the time we start marking files.

Another less obvious limitation is that it cannot MOVE files that really crowd memory because this BASIC program is also tieing up a fair amount of memory on its own. I still find it to be a great little utility when I want to move several files. The memory problem just doesn't crop up that often for me.

If there are any questions/problems, write to:

342 Trotter Court
Sanford, FL 32773

FRACTIONS: HARD FOR YOUNGSTERS AND A PROBLEM FOR PROGAMMING

by: Munson H. Cookayne, Jr. BSCS

Those of you who have had to help your children with fractions in arithematic have probaby wished you had a driller for them. The following program can provide that driller and its two major subroutines can be used as a basis for your own program.

The Greatest Common Multiplier (GCM) routine uses an old trick to quickly find the GCM to enable finding the reduced form of a product and to enable adding and subtracting. It only uses two compound lines, 100 and 110, to accomplish this feat.

The program is self explanitory when RUN and I have placed comments where I thought a pointer might help. In line 9, CONSTANT is assigned a value of 10 but it can be changed as needed for your youngster. It sets the upper value limit for the numerator and denominator. In line 1990, the problem loop is set to 25 iterations and this can be changed as you see fit.

The scores are saved in a sequential file for your viewing any time. This file has to be started with the command line:

```
PRINT #4: OPEN #5,"scores":: PRINT #5;"SCORES":PRINT  #4:  CLOSE
#5
```

This will initialize the file to contain the word 'SCORES'. The second program will read this file to the screen or to a printer when you want to see it.

```
  1 REM
                    @1986 K.I.S.S./softstuff
                  Munson H. Cookayne, Jr. BSCS

  2 ON ERR GO TO 7: PRINT #4: OPEN # 5,"scores"
  3 PRINT #4: OPEN # 6,"temp"
  4 INPUT #5;a$
  5 PRINT #6;a$
  6 GO TO 4
  7 ON ERR RESET : PRINT  #4:  CLOSE  #  5:: PRINT  #4:  ERASE
"scores",
  9 LET CONSTANT=10: REM THIS  SETS  UPPER  LIMIT  OF  FRACTION
COMPONENTS
 10 GO TO 1500
 99 REM
    THIS ROUTINE (LINES 100 TO 110) RETURNS THE GREATEST COMMON
DIVISOR THAT EXISTS FOR THE PAIR OF INTEGERS IN M AND N.

 100 IF  NOT  INT n THEN  LET god= ABS m: RETURN
 110 LET t= INT (m-n* INT (m/n)+.005): LET m=n: LET n=t: GO TO 1
00
 199 REM
```

ConT →

```
    THIS ROUTINE PRODUCES TWO  DISSIMILAR INTEGERS FOR MAKING A
FRACTION.
 200 LET N= INT ( RND *CONSTANT)+1
 210 LET D= INT ( RND *CONSTANT)+1
 220 IF N=D THEN  GO TO 200
 230 RETURN
 299 REM
THIS ROUTINE PRINTS A FRACTION  AT THE COORDINATE X,Y [UPPER
RIGHT HAND CORNER] USING NUMINATOR AND DENOMERATOR.

 300 PRINT  AT X,Y+2*( LEN  STR$ NUMERATOR=1)+( LEN  STR$ NUMERA
TOR=2);NUMERATOR
 310 REM  PLOT 112,88: DRAW 24,0
 311 PLOT Y*8,175-(X+1)*8+1: DRAW 24,0
 320 PRINT  AT X+1,Y+2*( LEN  STR$ DENOMINATOR=1)+( LEN  STR$ DE
NOMINATOR=2);DENOMINATOR
 330 RETURN
1499 REM
                    THIS ROUTINE IS THE MAIN BODY
                    FOR THE FRACTION ARITHEMATIC
                              PRACTICE.
1500 CLS
1505 POKE 23692,255
1510 PRINT  AT 21,0;"YOU ARE TO PERFORM DESIGNATED   OPERATIONS
ON THE FOLLOWING     FRACTIONS.  ADDITION,          SUBTRACTION
, MULTIPLICATION,    AND DIVISION WILL ALL BE       PRESENTED R
ANDOMLY."
1520 PRINT ''"PRESS A KEY TO CONTINUE"
1530 PAUSE 0
1540 CLS
1550 RANDOMIZE
1570 LET RIGHT=0
1990 FOR I=1 TO 25: REM  THIS IS PROBLEM LOOP (25 ITERATIONS)
2000 DRAW 255,0: DRAW 0,175: DRAW -255,0: DRAW 0,-175
2001 BORDER 4: INPUT ;
2002 BEEP .1,20: BEEP .2,10
2010 PRINT  AT 1,11;"FRACTIONS"; AT 2,6;"A PRACTICE SESSION"
2020 PLOT 0,141: DRAW 255,0
2100 GO SUB 200
2110 LET X=10: LET Y=14: LET NUMERATOR=N: LET DENOMINATOR=D: GO
SUB 300
2140 LET N1=N: LET D1=D
2150 GO SUB 200
2160 LET X=13: LET NUMERATOR=N: LET DENOMINATOR=D: GO SUB 300
2170 LET N2=N: LET D2=D
2190 PLOT 112,52: DRAW 24,0
2191 PLOT 112,51: DRAW 24,0
2200 LET FUNCTION= INT ( RND *4)+1
2201 IF FUNCTION=1 THEN  GO SUB 3000
2202 IF FUNCTION=2 THEN  GO SUB 3100
2203 IF FUNCTION=3 THEN  GO SUB 3200
2204 IF FUNCTION=4 THEN  GO SUB 3300
2210 INPUT  AT 1,0;"ENTER NUMERATOR"; AT 0,0;NUM
2220 INPUT  AT 1,0;"ENTER DENOMINATOR"; AT 0,0;DENOM
```

↓

```
2225 IF ((FUNCTION=2) AND (N1/D1<N2/D2)) THEN  LET AN=-AN
2230 IF (NUM=AN AND DENOM=AD) OR (NUM=AN/GCD AND DENOM=AD/GCD) T
HEN GO TO 5000
2240 GO TO 6000
2800 PRINT #1;"PRESS A KEY TO CONTINUE"
2810 PAUSE 0
2820 CLS
2990 NEXT I
2991 PRINT #6;RIGHT
2992 PRINT #4: CLOSE # 6
2993 PRINT #4: MOVE "temp","scores"
2994 STOP
3000 PRINT  AT 8,15;"ADD"
3010 LET N=D1: LET M=D2: GO SUB 100
3015 LET N=N1*D2
3020 LET M=N2*D1
3025 LET AN=M+N: LET AD=D1*D2
3030 LET N=AN
3040 LET M=AD
3050 GO SUB 100
3060 RETURN
3100 PRINT  AT 8,12;"SUBTRACT"
3110 LET N=D1: LET M=D2: GO SUB 100
3115 LET N=N1*D2
3120 LET M=N2*D1
3125 LET AN= ABS (M-N): LET AD=D1*D2
3130 GO TO 3030
3200 PRINT  AT 8,12;"MULTIPLY"
3210 LET AN=N1*N2
3220 LET AD=D1*D2
3230 GO TO 3030
3300 PRINT  AT 8,13;"DIVIDE"
3310 LET AN=N1*D2
3320 LET AD=D1*N2
3330 GO TO 3030
5000 LET RIGHT=RIGHT+1
5010 LET X=17: LET DENOMINATOR=DENOM: LET NUMERATOR=NUM: GO SUB
300
5020 LET N=NUM: LET M=DENOM: GO SUB 100
5030 IF GCD <> 1 THEN  LET DENOMINATOR=DENOM/GCD: LET NUMERATOR=
NUM/GCD: LET Y=20: PRINT  AT 17,18;"=": GO SUB 300
5040 GO TO 2800
6000 BEEP .5,-10
6010 PRINT  AT 20,13; BRIGHT 1;"WRONG"
6015 PRINT #1;"PRESS A KEY FOR RIGHT ANSWER": PAUSE 0: INPUT ;
6020 LET X=17: LET DENOMINATOR=AD: LET NUMERATOR=AN: GO SUB 300
6030 PRINT  AT 17,5; FLASH 1;"RIGHT =>"
6040 LET M=AD: LET N=AN: GO SUB 100
6050 IF GCD <> 1 THEN  LET DENOMINATOR=AD/GCD: LET NUMERATOR=AN/
GCD: LET Y=20: PRINT  AT 17,18;"=": GO SUB 300
6060 GO TO 5040
```

==========

: 4

"CHECKER"
A SHORT LESSON ON SPDOS
by: Munson H. Cockayne, Jr. BSCS

In this short lesson on SPDOS, I am going to reveal one of the many ways that SPDOS can be used by assembly programmers. This is not to imply that if you are not an assembly programmer, you cannot use this article or the program herein contained. To the contrary, you may find it very useful.

The SPDOS system call is one that checks for a filename's existance on a disk. If you try to load a file that is not on your disk, SPDOS will create an error that cannot be trapped by the usual ON ERR GOTO. Try it! It can leave you with a locked up computer, forever trying to GOTO and always ERRing.

The only satisfactory solution to this situation is assembly. There is a hook code call that can be made to SPDOS. Some of you may be familiar with some of the hook routines if you are assembly programmers and were once SURFers. But let's get on with the program.

According to the SPDOS Newsletter #2, there is a section of memory set aside by SPDOS to hold headers for files much the same way BASIC uses the DE and IX register pairs to hold the address of a file header and its length. The address of this header area is fixed at E420 hex or 58400. The map of this area is as follows:

E420	Filetype (one byte)
E421	Filename (ten bytes)
E42B	Filelength (two bytes)
E42D	Start (two bytes)
E42F	Zeros (15 bytes)

The start bytes are the starting address of a byte file, the first line to execute for a BASIC program, or zeros for any other type of file. The filetype can be 0 for BASIC or 3 for a byte file. (I will leave the other types for another article or for you to figure out.)

The hook doesn't work unless you tell it what you want to do. For the function in which we're interested, the B register communicates the function desired. The hook we will use is:

B = 13 This checks for the file on the disk

With these prerequisites met, we make the call to E490 (58512). That is where the hook code is. Now we go for the program.

```
ADDR CODE          MNEMONIC      COMMENTS


F400 2A5D5C   MAIN LD HL,(chad) get BASIC pointer
F403 7E       FIND LD A,(HL)    get the character there
F404 FE24          CP 24        is it "$"?
F406 2817          JR Z,VSTR    yes, jump to $var evaluator
F408 FE22          CP 22        is it """?
F40A 23            INC HL       point to next char
F40B 20F6          JR NZ,FIND   not """, loop
F40D E5            PUSH HL      save pointer
F40E 01FF0A        LD BC,0AFF   set counters
F411 7E       GETC LD A,(HL)    get char of string
F412 0C            INC C        increment count
F413 23            INC HL       point to next char
F414 FE22          CP 22        is this one """?
F416 2802          JR Z,DOLN    yes, jump to length evaluator
F418 10F7          DJNZ,GETC    no, loop to max of ten char
F41A 0600     DOLN LD B,00      clear counter
F41C E1            POP HL       get start of string
F41D 185F          JR DOHD      jump to do header
F41F 2B       VSTR DEC HL       point back one char
F420 7E            LD A,(HL)    get var name
F421 E6DF          AND DF       convert it to upper case
F423 F5            PUSH AF      save it on stack
F424 2A4B5C        LD HL,(vars) get start of vars area
F427 ED5B595C LOOP LD DE,(elin) get end of vars area
F42B 1B            DEC DE       point back one
F42C F1            POP AF       restore converted char
F42D E5            PUSH HL      save current pointer
F42E A7            AND A        clear carry flag
F42F ED52          SBC HL,DE    check for end of vars
F431 E1            POP HL       restore current pointer
F432 2874          JR Z,ERR2    end of vars, quit to ERR2
F434 BE            CP (HL)      is this the string we need?
F435 F5            PUSH AF      stack the converted char
F436 2831          JR Z,DOIT    yes, we got it so go do it
F438 7E            LD A,(HL)    not right so lets check for type
F439 CB7F          BIT 7,A      is bit 7 set?
F43B 2012          JR NZ,BIT6   yes, test more
F43D CB6F          BIT 5,A      is bit 5 set?
F43F 2806          JR Z,ARAY    no, ths is an array
F441 110600  #VAR  LD DE,0006   this is simple # var
F444 19            ADD HL,DE        so point past it and
F445 18E0          JR F427              loop back
F447 23       ARAY INC HL       point to next byte
F448 4E            LD C,(HL)    get the
F449 23            INC HL            byte count
F44A 46            LD B,(HL)              into BC
F44B 23            INC HL       point past byte count
F44C 09            ADD HL,BC    point past var
F44D 18D8          JR LOOP      loop back
F44F CB77     BIT6 BIT 6,A      is bit 6 set?
F451 200C          JR NZ,MORE   no, test some more
F453 CB6F          BIT 5,A      is bit 5 set?
F455 28F0          JR Z,ARAY    no, test more
F457 23       NAME INC HL       point past this byte
F458 7E            LD A,(HL)    get byte
F459 CB7F          BIT 7,A      is bit seven set?
F45B 20E4          JR NZ,#VAR   yes, go increment past value
F45D 18F8          JR NAME      loop
F45F CB6F     MORE BIT 5,A      is bit 5 set?
F461 28E4          JR Z,ARAY    no, then it is an array
F463 011300        LD BC,0013   increment
F466 09            ADD HL,BC         by 19 bytes
F467 18BE          JR LOOP                and loop
F469 23       DOIT INC HL       get byte
F46A 4E            LD C,(HL)         count for this
F46B 23            INC HL                 variable
F46C 46            LD B,(HL)                   into registers
F46D 23            INC HL       point past this count
F46E F1            POP AF       restore char
F46F C5            PUSH BC      copy count to
F470 D1            POP DE            second register pair
F471 E5            PUSH HL      stack pointer
F472 210A00        LD HL,000A   set to max length
F475 A7            AND A        clear carry
F476 ED52          SBC HL,DE    test length
F478 E1            POP HL       restore pointer
F479 3003          JR NC,DOHD   if not too long then do header
F47B 010A00        LD BC,000A   else, set length to ten
F47E 1121E4  DOHD  LD DE,E421   point to disk header area
F481 79            LD A,C       save count
F482 EDB0          LDIR         move string
F484 67            LD H,A       put count in register
F485 3E0A          LD A,0A      set limit
F487 94            SUB H        get number of spaces needed
F488 2807          JR Z,SKIP    skip if none needed
F48A 47            LD B,A       transfer count again
F48B 3E20          LD A,20      put space in accumulator
F48D 12       ADSP LD (DE),A    add it to end of string
F48E 13            INC DE       increment string pointer
F48F 10FC          DJNZ,ADSP    repeat till count expires
F491 3E00     SKIP LD A,00      clear accumulator
F493 62            LD H,D       transfer pointer
F494 6B            LD L,E            to HL
F495 13            INC DE       point to byte past pointer
F496 77            LD (HL),A    clear the first byte (prime pump
F497 011100        LD BC,0011   set counter
F49A EDB0          LDIR         zero fill
F49C 0613          LD B,13      set request to check for file
F49E CD90E4        CALL HOOK    check for file
F4A1 010100        LD BC,0001   set return to true value
F4A4 D8            RET C        and return if file found
F4A5 0E00          LD C,00      else, set to false
F4A7 C9            RET          and return
F4A8 CF       ERR2 RST 08H           do error
F4A9 01             ERROR 2           #2 report
F4A9          HOOK EQU E490     HOOK is 58512
```

This utility must exist somewhere in memory. A REM statement of 170 free bytes at the beginning of a BASIC program will do nicely but it can be put anywhere you want as it is relocatable.

This utility can handle simple string variable references or constants only. I did not intend to include TS function dispatcher in this discussion and, instead, I made the little lookup routine. It does not handle string array references!

Here is an example of a use for this routine.

```
10 REM "CHECK" IS A VARIABLE CONTAINING THE ADDRESS OF THE FIRST
BYTE OF THE PROGRAM ABOVE
1000 IF USR CHECK THEN PRINT #4: LOAD "YOUR PROG"
1010 IF NOT USR CHECK THEN PRINT "YOUR PROG"; " NOT ON THIS
DISK!"'''PUT RIGHT DISK IN DRIVE AND PRESS ENTER": PAUSE 0: GO
TO 1000
```

Or try this:

```
10 REM "CHECK" IS AS ABOVE
1000 INPUT "ENTER TITLE OF FILE YOU DESIRE: ";LINE T$
1010 IF USR CHECK THEN PRINT #4: LOAD T$
1020 IF NOT USR CHECK THEN PRINT T$;" NOT ON THIS DISK!"'''PUT
RIGHT DISK IN DRIVE AND PRESS ENTER": PAUSE 0: GO TO 1000
```

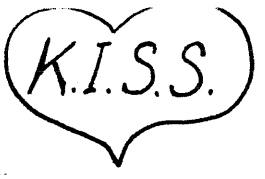
Remember, the first string reference after the call to CHECK must be the filename that you are seeking as in lines:
1000 and 1010 in first program fragment
1010 and 1020 in second program fragment

Here is a little program in BASIC to load the code.

```
  10 LET START = 62464: REM START can be any convenient address.
        62464 is the start of a 1k buffer in SPDOS.
1000 FOR A = START TO START + 169: READ X: POKE A,X: NEXT A
9000 DATA 42, 93, 92, 126, 254, 36, 40, 23, 254, 34, 35, 32,
246, 229, 1, 255, 10, 126, 12, 35, 254, 34, 40, 2, 16, 247, 6,
0, 225, 24, 95, 43, 126, 230, 223, 245, 42, 75, 92, 237, 91, 89,
92, 27, 241, 229, 167, 237, 82, 225, 40, 116, 190, 245, 40, 49,
126, 203, 127, 32, 18, 203, 111, 40, 6, 17, 6, 0, 25, 24, 224,
35, 78, 35, 70, 35, 9, 24, 216, 203, 119, 32, 12, 203, 111, 40,
240, 35, 126, 203, 127, 32, 228, 24, 248, 203, 111, 40, 228, 1,
19, 0, 9, 24, 190, 35, 78, 35, 70, 35, 241, 197, 209, 229, 33,
10, 0, 167, 237, 82, 225, 48, 3, 1, 10, 0, 17, 33, 228, 121,
237, 176, 103, 62, 10, 148, 40, 7, 71, 62, 32, 18, 19, 16, 252,
62, 0, 98, 107, 19, 119, 1, 17, 0, 237, 176, 6, 19, 205, 144,
228, 1, 1, 0, 216, 14, 0, 201, 207, 1
```

Have fun with this and send your comments/questions to:

342 Trotter Court
Sanford, FL 32773



SPDOS SUPPORT

CUSTOM SOFTWARE

K.I.S.S./computers
presents
SPDOS UTILITIES V1.1

New Command	Function
PRINT #4'Make,#	Makes a REM statement with # bytes of free space.
PRINT #4'X	Eliminates every line that begins with a REM statement.
PRINT #4'Names	Reveals the names and types of all variables in the VARS area.
PRINT #4'Dump	Dumps all the information in the VARS area including location and contents.
PRINT #4'VERIFY	Is an IF ... THEN construct that tests for a filename on the disk.
PRINT #4'Renum	Is a line renumbering routine that allows you to select increment, block of lines to be renumbered, and starting line number.

NONE OF THE ABOVE ROUTINES USE ANY EXTRA MEMORY BEYOND THAT USED BY SPDOS. The SPDOS buffer holds these routines when called so they are compatible with all your existing programs!

Also included:

TRACKER, a full function sector editor and, CHECKER, a memory resident routine that checks for a filename on the disk.

PRICE: $11.00 pp on 5 1/4 inch DS/QD or SS/QD disk. Also available on cassette for transfer to your disk at $12.00 pp. (Specify media please) Full documentation on disk/cassette. MONEY BACK IF NOT SATISFIED!

Available ~~COMING SOON~~ Now

Alpha Taper — For Both

Is the disk filename organizer that puts your files in alphanumeric order (not case sensitive).
Is a disk to tape file MOVEr.

→ ADD $1.00 TO YOUR ORDER AND MENTION UP-DATE

ALPHA & TAPER AVAILABLE NOW

JUST ADD $1 TO YOUR ORDER AND MENTION THIS AD!

```
 1 REM
                    ©1986 K.I.S.S./softstuff
                 Munson H. Cockayne, Jr. BSCS

 10 CLS
 20 PRINT #4: OPEN #5,"scores"
 30 PRINT #1;"1) To screen"'"2) To printer": PAUSE 0: LET r$=
INKEY$ : INPUT ;: IF r$ <> "1" AND r$ <> "2" THEN GO TO 30
 40 IF r$="1" THEN  OPEN # 2,"s"
 50 IF r$="2" THEN  OPEN # 2,"p"
 60 ON ERR GO TO 120
 70 INPUT #5;a$
 80 PRINT a$
 90 INPUT #5;a$
100 PRINT ,a$
110 GO TO 90
120 ON ERR RESET
130 PRINT #4: CLOSE #5
140 CLOSE #2

Well, that's it for now. Questions to:
                      342 Trotter Court,
                      Sanford, FL  32773.
```

From Tom Simon Jr

```
  1 REM BE SURE THAT YOU HAVE S
NOTED UP SPDOS FIRST BEFORE YOU
RUN THIS PROGRAM
  2 REM THIS PROGRAM MOVES SCRE
EN $ FILES FROM THE JLO SYSTEM T
O THE SPDOS SYSTEM AND IS A DEMO
NSTRATION OF THE TWO SYSTEMS TOG
ETHER. I AM RUNNING THIS ON A TW
O DRIVE SYSTEM AND SELECT JLO DR
IVE #1 TO BOOT FROM BY PRESSING
1 AS I TURN ON THE MACHINE. SPDO
S IS ALWAYS LOADED FROM JLO DRIV
E #0 SO THAT SYSTEM WILL BE IN T
HAT DRIVE. COMMANDS OF EACH SYST
EM WILL DEFAULT TO THEIR RESPECT
IVE DRIVES.
  10 CAT : PAUSE 0: INPUT "Name
of screen";a$
  11 IF a$="DIR" THEN PRINT #4:
CAT "": PAUSE 0: GO TO 10
  12 LOAD /a$SCREEN$ : PAUSE 0
  13 PRINT #4: SAVE a$SCREEN$
  15 GO TO 10
```

. 6

The ISSUE DISK Program

The issue diskette is a integral part of TS-2068 UP-DATE. The proceeds of sales are divided to pay token renumeration to the authors of the programs presented in the magazine for key-in projects. UP-DATE takes $8.00 for expenses and divides the balance between authors. Although there has been no disbursements at this writing, a ledger is kept for each author and checks will be mailed as funds accrue.

Generally, the feature programs included in the issue diskette are original programs never before offered to the TS-2068 user group and not for sale from other sources. Most of the utilities included are also new, but similiar utility programs may have appeared in other magazines. Public Domain progrms may be added from time to time, in order to increase the value of the diskette. Such public domain programs will be carefully checked out for value and to insure that they operate without flaw.

UP-DATE tries to make the issue disk a $50.00 value in comparison with programs offered commerically for other computers. Sometimes, as in this April issue, the Larken users cannot use the 5 DOSDEX programs. In such cases fill in programs will be added to the Larken issue disk to make up the difference. The same applies to other system formats. Such fill in programs may be accompanied by documentation if needed. Whatever, UP-DATE guarantees "satisfaction or money back" on all issue diskettes. The old American custom "the baker's dozen" applies. Extras may be thrown in to make sure that you get more than you pay for.

****

The April 1988 Issue Disk

The April Issue diskette will be ready O/A May 1st and will include the Feature Software "Budget", plus the compiled program, plus the printer annex, plus the customized "Loader Manager" for either Oliger Safe, or Larken LKDOS system. Later when the Aerco Disk capability is re-gained, that format will also be available. Aerco Disk users may order the issue disk, but please DO NOT inclose payment. The disk will be mailed when ready and an invoice enclosed.

Other programs and utilities included in the April issue disk are: Five DOSDEX Manager programs (OLIGER FORMAT ONLY), MAIL MERGE (LARKEN FORMAT ONLY), the MOV utility (all formats), and the Screen String Demo program (all formats). The Mail Merge program for Larken Format is the same program given in the January issue, but converted to Larken format.

The issue disk is priced at $16.00 and is sent postpaid. Send orders to UP-DATE, address given on cover.

Continued from preceeding page.

R_C—External collector resistance
$r_{CE(sat)}$—Collector-to-emitter saturation resistance
r_d—Damping resistance
R_E—External emitter resistance
R_{EB}—Emitter-base junction resistance (assume 4 ohms average)
$r_{e_1e_2}$—Small-signal emitter-emitter on-state resistance (double emitter transistors)
r_i—Dynamic resistance at inflection point
R_L—Load resistance
R_θ—Thermal resistance
$R_{\theta CA}$—Thermal resistance, case-to-ambient
$R_{\theta JA}$—Thermal resistance, junction-to-ambient
$R_{\theta Jc}$—Thermal resistance, junction-to-case
r_T—Slope resistance
Si—Silicon
T—Temperature
T_A—Ambient temperature
T_C—Case temperature
t_d—Delay time
t_f—Fall time
t_{fr}—Forward recovery time
T_j—Junction temperature
t_{off}—Turn-off time
t_{on}—Turn-on time
T_{opr}—Operating temperature
t_p—Pulse time
t_r—Rise time
t_{rr}—Reverse recovery time
t_s—Storage time
TSS—Tangential signal sensitivity
T_{stg}—Storage temperature
t_w—Pulse average time
V_B—Base voltage (dc)
V_{BB}—Base supply voltage (dc)
V_{BC}—Base-to-collector voltage (dc)
V_{bc}—Base-to-collector voltage (rms)
v_{bc}—Base-to-collector voltage (instantaneous)
V_{BE}—Base-to-emitter voltage (dc)
$V_{BE_{sat}}$—Saturation voltage, base to emitter
V_{be}—Base-to-emitter voltage (rms)
v_{be}—Base-to-emitter voltage (instantaneous)
v_{BO}—Breakover voltage (instantaneous)
$V_{(BR)CBO}$—Breakdown voltage, collector-to-base, emitter open
$V_{(BR)CEO}$—Breakdown voltage, collector-to-emitter, base open
$V_{(BR)CER}$—Breakdown voltage, collector-to-emitter, with specified resistance between base and emitter
$V_{(BR)CES}$—Breakdown voltage, collector-to-emitter, with base short-circuited to emitter
$V_{(BR)CEV}$—Breakdown voltage, collector-to-emitter, voltage between base and emitter
$V_{(BR)CEX}$—Breakdown voltage, collector-to-emitter, circuit between base and emitter
$V_{(BR)EBO}$—Breakdown voltage, emitter-to-base, collector open
$V_{(BR)ECO}$—Breakdown voltage, emitter-to-collector, base open (formerly BV_{ECO})
$V_{(BR)E1E2}$—Breakdown voltage, emitter-to-emitter (double-emitter transistor)
$V_{(BR)GSS}$—Breakdown voltage, gate to source, drain short-circuited to source
$V_{(BR)GSSF}$—Breakdown voltage, forward voltage applied to gate-source, drain short-circuited to source
$V_{(BR)GSSR}$—Breakdown voltage, reverse voltage to gate-source, drain short-circuited to source
$V_{(BR)R}$—Reverse breakdown voltage
V_{B2B1}—Bias dc voltage between base 2 and base 1 (double-base transistor)
V_C—Collector voltage (dc)
V_{CB}—Collector-to-base voltage (dc)
$V_{CB(f)}$—Dc open-circuit voltage, floating potential, collector-to-base
V_{cb}—Collector-to-base voltage (rms)
v_{cb}—Collector-to-base voltage (instantaneous)
V_{CBO}—Collector-to-base voltage (dc), with emitter open
V_{CC}—Collector supply voltage (dc)
V_{CE}—Collector-to-emitter voltage (dc)
V_{ce}—Collector-to-emitter voltage (rms)
$V_{CE(f)}$—Dc open-circuit voltage, floating potential, collector-to-emitter
V_{CEO}—Collector-to-emitter voltage (dc), with base open
V_{CER}—Collector-to-emitter voltage (dc), with specified resistance between base and emitter
V_{CES}—Collector-to-emitter voltage (dc), with base short-circuited to emitter
$V_{CE_{sat}}$—Saturation voltage, collector to emitter
V_{CEV}—Collector-to-emitter voltage (dc), with voltage between base and emitter
V_{CEX}—Collector-to-emitter voltage (dc), with circuit between base and emitter
V_D—Off-state voltage (direct)
V_{DD}—Drain supply voltage (dc)
V_{DG}—Drain-to-gate voltage (dc)
V_{DM}—Peak off-state voltage
V_{DRM}—Peak off-state voltage repetitive
V_{DS}—Drain-to-source voltage (dc)
V_{DSM}—Peak off-state voltage, nonrepetitive
V_{DU}—Drain-to-substrate voltage (dc)
V_{DWM}—Peak off-state voltage, working
V_E—Emitter voltage (dc)
V_{EB}—Emitter-to-base voltage (dc)
$V_{EB(f)}$—Dc open-circuit voltage, floating potential, emitter-to-base
V_{EBO}—Emitter-to-base voltage (dc), with collector open
V_{EC}—Emitter-to-collector voltage (dc)
$V_{EC(f)}$—Dc open-circuit voltage, floating potential, emitter-to-collector
V_{EE}—Emitter supply voltage (dc)
V_F—Forward voltage (dc)
V_f—Alternating component of forward voltage (rms value)
v_F—Forward voltage (instantaneous)
V_{FG}—Forward gate voltage (direct)
V_{FGM}—Peak forward gate voltage
V_{FM}—Forward voltage, peak total value
$V_{F(RMS)}$—Forward voltage, total rms value
V_{GD}—Gate nontrigger (direct) voltage
V_{GG}—Gate supply voltage (dc)
V_{GQ}—Gate turn-off voltage (direct)
V_{GS}—Gate-to-source voltage (dc)
$V_{GS(off)}$—Gate-to-source cutoff voltage
$V_{GS(th)}$—Gate-to-source theshold voltage
V_{GSF}—Forward gate-to-source voltage (dc), of such polarity that an increase in its magnitude causes the channel resistance to decrease
V_{GSR}—Reverse gate-to-source voltage (dc), of such polarity that an increase in its magnitude causes the channel resistance to increase
V_{GT}—Gate trigger voltage (direct)
V_{GTRMIN}—Minimum gate trigger voltage
V_{GU}—Gate-to-substrate voltage (dc)
V_I—Inflection-point voltage
V_P—Peak-point voltage (double-base transistor)
V_{PP}—Projected peak-point voltage
V_{PT}—Punch-through voltage
V_R—Reverse voltage (dc)
V_r—Alternating component of reverse voltage (rms value)
v_R—Reverse voltage (instantaneous)
$V_{R(RMS)}$—Reverse voltage, total rms value
V_{RRM}—Reverse voltage, maximum recurrent
V_{RSM}—Reverse voltage, peak transient
V_{RT}—Reverse collector-to-base voltage, reach-through voltage
V_{RWM}—Reverse voltage, (peak) working
V_{SB}—Source-substrate voltage
V_{SS}—Source supply voltage (dc)
V_{SU}—Source-to-substrate voltage (dc)
V_T—On-state voltage, direct
V_{TMIN}—Minimum on-state voltage
V_{TO}—Threshold voltage
V_V—Valley-point voltage (double-base transistor)
V_Z—Regulator voltage, reference voltage (dc working voltage)
V_{ZM}—Regulator voltage, reference voltage (dc at maximum rated current)
y_{fb}—Small-signal, short-circuit forward transfer admittance, common base
y_{fe}—Small-signal, short-circuit forward transfer admittance, common emitter
y_{fs}—Small-signal, short-circuit forward transfer admittance, common source

Continued on next page.

continued

y_{ib}—Small-signal, short-circuit input admittance, common base
y_{ie}—Small-signal, short-circuit input admittance, common collector
y_{ie}—Small-signal, short-circuit input admittance, common emitter
y_{ob}—Small-signal, short-circuit output admittance, common base
y_{oc}—Small-signal, short-circuit output admittance, common collector
y_{oe}—Small-signal, short-circuit output admittance, common emitter
y_{rb}—Small-signal, short-circuit reverse transfer admittance, common base.
y_{rc}—Small-signal, short-circuit reverse transfer admittance, common collector
y_{re}—Small-signal, short-circuit reverse transfer admittance, common emitter
Z_m—Impedance, modulator freq. load
z_{rf}—Impedance, radio frequency
$Z_{\theta(t)}$—Transient thermal impedance
$Z_{\theta JA(t)}$—Transient thermal impedance, junction-to-ambient
$Z_{\theta JC(t)}$—Transient thermal impedance, junction-to-case
z_v—Video impedance
z_z—Regulator impedance, reference impedance (small-signal at I_z)
z_{ZK}—Regulator impedance, reference impedance (small-signal at I_{zk})

Machine Screw Tap and Clearance Drill Sizes

Type	Tap Drill	Clearance Drill	Type	Tap Drill	Clearance Drill
0-80	3/64	50	10–24	25	13/64
1–64	53	47	10-32	21	13/64
1-72	53	47	12-24	16	7/32
2-56	50	42	12-28	14	7/32
2-64	50	42	1/4-20	7	17/64
3-48	47	36	1/4-28	3	17/64
3-56	45	36	5/16-18	F	21/64
4-40	43	31	5/16-24	1	21/64
4-48	42	31	3/8-16	5/16	25/64
5-40	38	29	3/8-24	Q	25/64
5-44	37	29	7/16-14	U	29/64
6–32	36	25	7/16-20	25/64	29/64
6-40	33	25	1/2-12	27/64	33/64
8-32	29	16	1/2-13	27/64	33/64
8-36	29	16	1/2-20	29/64	33/64

Volume Measure

1000 cubic millimeters	= 1 cubic centimeter
1000 cubic centimeters	= 1 cubic decimeter
1000 cubic decimeters	= 1 cubic meter

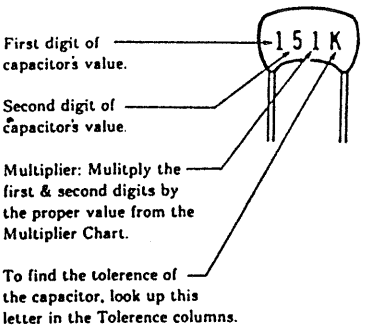
Liquid Measure

10 milliliters	= 1 centiliter
10 centiliters	= 1 deciliter
10 deciliters	= 1 liter

Weight Measure

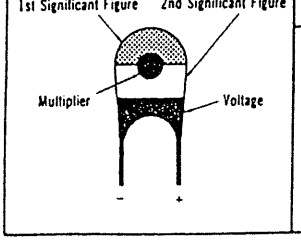
10 milligrams	= 1 centigram
10 centigrams	= 1 decigram
10 decigrams	= 1 gram
10 grams	= 1 dekagram
10 dekagrams	= 1 hectogram
10 hectograms	= 1 kilogram
1000 kilograms	= 1 metric ton

CODE FOR FILM AND MICA CAPACITOR



MULTIPLIER		TOLERENCE OF CAPACITOR		
FOR THE NUMBER	MULTIPLY BY	10pF OR LESS	LETTER	OVER 10pF
0	1	±0.1pF	B	
1	10	±0.25pF	C	
2	100	±0.5pF	D	
3	1000	±1.0pF	F	±1%
4	10,000	±2.0pF	G	±2%
5	100,000		H	±3%
			J	±5%
8	0.01		K	±10%
9	0.1		M	±20%

TANTALUM CAPACITORS
(CAPACITANCE GIVEN IN μF)



COLOR	1st SIGNIFICANT FIGURE	2nd SIGNIFICANT FIGURE	MULTIPLIER	RATED DC VOLTAGE
Black	·	0	1	10
Brown	1	1	10	·
Red	2	2	100	·
Orange	3	3	·	·
Yellow	4	4	·	6.3
Green	5	5	·	16
Blue	6	6	·	20
Violet	7	7	·	·
Gray	8	8	0.01	25
White	9	9	0.1	3
Pink	·	·	·	35

Capacitor Color Codes—

Continued on next page.

S . N . U . G .
Sinclair Northamerica Users Group

During one of the organizational meetings of the recent Sunstate Timex/Sinclair Winterfest '88, the idea of a National organization for the advancement of Sinclair computing came up. It was decided that since we had developed a "core group" that was dedicated to promoting Sinclair computing, We would attempt to lay the groundwork for such an organization. It was also mentioned that the greater the amount of time from the departure of Timex from the computer industry, the less of an active market would result. Since we would have users from across the nation at the fest, it would be an ideal time to make our plans known. So, the Sinclair Northamerica Users Group, or SNUG (a name submitted to us by John Cushran, and later modified by Bill Jones) was starting to come closer to reality.

WHAT IT IS

The intent of SNUG is to provide a forum for exchange of ideas. It would be a source of information, such as a listing of active members, active Users Groups, Sinclair specific Bulletin Boards, an active library of Public Domain software, and a listing of available shareware and freeware. Later on we hope to propose an industry wide standard of hardware and software compatability.So as to not to have to reinvent the wheel, and to do this in the shortest amount of time, we going to try and use an already established National group, such as CORSA (Corvair Owners Assn.) as a model to base our group on. SNUG would act as an umbrella Organization, with Regions being developed to tie in with established groups in those areas.

WHAT IT ISN'T

It is the intent of the organizers NOT to infringe or supercede any already established User Group or Vendor.It is intended to show some strength to the industry that Sinclair is not dead, and the mere fact that we can get this Organization together will prove that we can stick together and grow and prosper. We look at this as an enhancement to activities that have been planned on. Hopefully a Northamerican Calendar of events could be established to help co-ordinate any future plans and events. It is not designed to take anything away from anyone.

WHAT TO DO

We need the support of EVERY SINGLE SINCLAIR USER ! Whether you reside in Canada, the U.S., or Mexico, or for that matter anywhere, we need to know how you feel and what you want in this Organization. This is your opportunity to be heard ... your comments, critisms, complaints, or praises. What we have here is nothing more than an idea. Nothing at this time is set in concrete. We are more than open to suggestions. To make it work, these ideas of yours have to be forwarded immediately. We are putting on a time limit until June 30,1988. If there is no support, then we will not proceed further. If there is imput, we will update on a monthly basis to whoever will put the information in print. So, lets hear from you soon!!

TEMPORARY CONTACT FOR INFORMATION :

MEL NATHANSON

7515 ARBORDALE DRIVE
PORT RICHEY, FLORIDA 34668

(813) 863 - 5552

or

MARY LYNN JOHNSON
190 HICKORY WOODS CT
UNIT 3C
DELTONA, FL 32765

(305) 860-2465

** SUPPORTING PUBLICATIONS ** Time Designs, Syncware News, TS-2068 UP-DATE, Quantum Levels.

** Please show your support with INTEREST CARD for Mail List.

TS-2068 UP-DATE
the user's NEWS
Take it to the Limit
A Quarterly Magazine for the users
TS-2068 UP-DATE
1317 STRATFORD AVE.
PANAMA CITY, FL 32404