THE NATIONAL NINETY-NINER

VOL III - NO. 2 - FEBRUARY, 1986

THE 99ER'S ASSOCIATION
3535 SO.H ST., #26
BAKERSFIELD, CALIF. 93304
(805) 397-4361

DON VEITH - EDITOR
COMPUSERVE ID #: 72257,3671

CREATED FOR TI 99/4A HOME COMPUTER OWNERS

TABLE OF CONTENTS

THE NATIONAL NINETY-NINER

VOL III - NO. 2 - FEBRUARY, 1986

THE 99ER'S ASSOCIATION
3535 SO.H ST., #26
BAKERSFIELD, CALIF. 93304
(805) 397-4361

DON VEITH - EDITOR
COMPUSERVE ID #: 72257,3671

CREATED FOR TI 99/4A HOME COMPUTER OWNERS

ANNOUNCEMENTS

FAS-Tran

By William Harms - Pomona (Ca) Valley Users Group

FAS-Tran is a program I wrote (with a little help from my friends) for anyone to use on the TI-99/4A. It is a **Public Domain** program with no copyright. I enjoyed writing it and did learn a few things.

It's called) FAS-Tran (for fast checkbook totals (or any other transactions) where you desire totals on each "category" or type of item. It's good for budgeting and income tax totals. It has growth potential into a utility for many related applications. You can setup your own categories easily and then enter the data. Data is entered in four (4) categories:

(1) Check number (or Transaction No.) - six digits maximum

(2) Date - Any six digit number in the MMDDYY pattern (zeroes are substituted for days and months with only one digit).

(3) Category Code - A two digit number.

(4) Amount - Usually expressed in dollars ($), up to eight digits maximum.

The data entry is fast and error free. There is a net pay explosion technique to automatically enter recurring amounts. D/V-80 file conversion is available if your data is already stored in files of this type on disk. Different types of listings of all the checks in a file are available including a printout/display/D/V-80 file showing the total amount for each category in the file and all the checks/transactions.

Totals may be saved in the SYLK format to load the data into a Multiplan spreadsheet. The program has a spreadsheet type option. The monthly totals of transactions, by category, can be loaded into a 12 month matrix/file that will provide totals for each month and year-to-date totals. If the budget option is used, variances to the budget may also be listed.

The current program is limited to 350 transactions for the RAM sort, 3600 for the disk sort, and 99 individual categories. Since the Checks File is updated in the append mode, you can easily recap your checkbook weekly, monthly, or whenever you need to review the totals.

Instructions are included in two files on the disk named $FA-INSTR1 and $FA-INSTR2. The instruction files are stored in a DIS/VAR-80 format and may be printed with TI-Writer or the Editor Assembler's Print Option. The program requires Extended Basic, a Disk Drive, and Expansion Memory. A printer is highly recommended. To obtain a copy of the program, send your request to the program's author. The details on how to obtain the programs are outlined below:

(1) If your disk drives are double sided, only one disk will be required. If your system has single sided, single-density drives, two disks will be required to obtain the programs.

(2) Forward the diskette(s), formatted in a SD/DS format (Bill cannot handle DS/DD format) for one disk or SS/SD if you are forwarding two disks, to the author at the address listed below.

(3) You must forward sufficient postage to insure return of the diskette(s). Any package NOT CONTAINING the proper postage will never return. The program is free, not the postage for the return of your disks.

(4) Take a moment and include two extra labels. One with your return address and one with Bill Harms home address. Bill will simply have to copy the programs to your disk(s), place your postage on the diskette mailer plus the extra labels and drop it into the mailbox.

Contact the author of FAS-Tran at:

FAS-Tran Programs
C/O BILL HARMS
6527 HAYES COURT
CHINO, CA 91710

SOFTWARE LIBRARY RELEASES

We are happy to announce the release of two more disks of software from THE 99'ERS ASSOCIATION Library. Disks 002 and 003 are ready for release. We had to bypass the release of programs last November due to our Librarian's heavy schedule. Keith will now be able to handle your request for releases from the Library. Please format your disks prior to mailing them. Each disk of programs being released is in a SSSD format. You may forward a single disk formatted in either SSDD or DSDD which will handle all the programs being released. If your system operates with SSSD diskettes, two disks will be required to obtain the programs.

As we have previously requested, PLEASE include the return postage and two extra labels with your return address and that of Keith Felix, our Librarian, inside your diskette mailer. We would appreciate it if your 99'ERS ASSOCIATION Subscriber Number was included in the enclosed correspondence. If you are a recent subscriber, you may also obtain Library Release 001 at the same time you obtain releases 002 and 003. Please state in your note to Keith what format you used to setup your diskette(s) and the Software Library Releases you wish to have returned on your disk(s). You may contact our Librarian at:

99'ERS SOFTWARE LIBRARY
C/O KEITH FELIX
P.O. BOX 32421
SAN JOSE, CA 95152

NUTS 'N BOLTS DISK NO. 2

Jim Peterson, better known as Tigercub Software, has done it once more. Another disk chock full of utility subprograms (108). All of the programs are saved in the MERGE format to allow each one may be incorporated into your program by simply typing MERGE and the filename. The utilities saved as subprograms simply require a CALL statement with the proper parameters to invoke their use in a program. Each subroutine was saved with high line numbers to prevent their overwriting the lines in your program. The subroutines are sequenced so they will not overwrite each other or THOSE ON NUTS 'N BOLTS DISK No. 1. The contents of Disk No. 2 include 21 screen font routines, 21 screen display routines, 3 joystick routines, 13 math routines including every base conversion, 5 screen graphs and a printer graph, 16 programming utilities, 4 file handling utilities, 6 two-dimensional sorts and many more programs. The price for NUTS & BOLTS NO. 2 is $19.95. This price INCLUDES both shipping and handling charges. You may order both NUTS & BOLTS disks for $37.00 postpaid.

Jim also included information on an additional bargain available. Tigercub Software is making available a collection of 18 special disks which contain 5 or 6 of the Tigercub's programs. The programs on each disk all relate to one category. Some examples of the categories are Programming Tutor, Programmer's Utilities, Reflex and Concentration, Two-Player Games, Tigercub's Best, and many more. Each disk is priced at $12.00 each with the postage prepaid. The remainder of each disk has been filled with some of the finest public domain programs of the same category. To set the record straight, Jim stated that he is not selling the public domain programs, but has just included them as a bonus to each purchaser. The programs on each disk created by the "Tigercub" are greatly reduced from their individual catalog prices. You may obtain more details about the software discussed above by contacting Jim Peterson at:

TIGERCUB SOFTWARE
156 COLLINGWOOD AVE.
COLUMBUS, OHIO 43213

SST SOFTWARE REDUCES ITS PRICES

SST Software asked us to notify our readers of the reduced prices for their products. The prices were effective January 15, 1986. The price reductions represent the new list price for each product.

PRODUCT	LIST PRICE	NEW PRICE
PRE-SST PROGRAM	$30.00	
EXPANDED BASIC COMPILER	$49.00	$25.00
EXPANDED BASIC COMPILER WITH GRAPHICS PACKAGE	$59.00	$35.00

The firm may be contacted at SST Software; P.O. Box 26; Cedarburg, WI 53012; (414) 771-8415.

FROM THE MAILBOX

BLACKSHIP COMPUTER SUPPLY

This firm, in the postcard we received, states it has wholesale pricing on 5 1/4" diskettes. The firm's advertisement also offers power supply strips with surge protection, disk head cleaning kits, diskette notchers, and diskette storage boxes. Some disk prices are:

SIZE	DENSITY	1-50	51 +
5.25"	SS/DD	$.69	$.59
5.25"	DS/DD	$.79	$.69
PC FORMATTED			
AT	1.2 MB	$1.99	$1.89
3.5"(MAC)	1D	$1.99	$1.89

Contact the firm at P.O. Box 883362; San Francisco, Ca 94188; (415) 550-0512.

NO LABEL SYSTEM BY WEBER

You affix a plastic pocket to any floppy disk, write details about the disk's contents on a handy card, and then insert the card into the plastic pocket previously attached to the disk. Change the disk contents, or reformat it for new programs, simply write the new list of programs on a new insert card. The NL system ends the task of attaching and removing labels. It also provides a method for locating programs without running a disk catalog.

The price is $19.45,plus $2.00 for shipping, for 100 plastic pockets, 100 white and 100 color coded inserts. The system is also available for 3.5" and 8" diskettes. You are urged to call or write for a free sample or forward $5.00 for a Test Packet from Weber & Sons; P.O. Box 104, Dept PC-B; Adelphin, NJ 07710; or telephone (800) 225-0044.

ODDS 'N ENDS

LANGUAGES NOT INCLUDED IN THE WORLD OF COMMERCIAL PROGRAMMING

By Doug Bohrer And Ted A. Bear

APL, BASIC,COBOL, PILOT, FORTRAN, FORTH, PASCAL, RPG that are well known and (more or less) loved throughout the computer industry. There are numerous other languages, however, that are less well known yet still have ardent devotees. In fact, these little known languages generally have the most fanatic admirers. For those who wish to know more about these obscure languages -- and why they are obscure -- we present the following catalogue.

C - This language is named for the grade received by its creator when he submitted it as a class project in a graduate programming class. C- is best described as a "low level" programming language. In general, the language requires more C- statements than machine code instructions to execute a given task. In this respect, it is very similar to COBOL.

DOGO - Developed ay MIOT (Massachussettes Institute of Obedience Training). DOGO heralds a new era of computer literate pets. DOGO commands include SIT, HEEL, STAY, PLAY_DEAD, and ROLL_OVER. An innovative feature of DOGO is "puppy graphics", a small cocker spaniel that occasionally leaves deposits as it travels across the screen.

FIFTH - FIFTH is a precise mathematical language in which the data types refer to quantities. The data types range from CC, OUNCE, SHOT, and JIGGER to FIFTH (thence the name of the language), LITER, MAGNUM, and BLOTO. Commands refer to ingredients such as CHABLIS, CABERNET, GIN, VERMOUTH, VODKA, SCOTCH, BOURBON, CANADIAN, COORS, BUD, EVER-CLEAR, and WHAT_EVERS_AROUND.

The many versions of the FIFTH language reflect the sophistication and financial status of the user. Commands in the elite dialect include VSOP, LAFITE, and WAITERS RECOMMENDATION. The GUTTER dialect commands include THUNDERBIRD, RIPPLE, and HOUSE RED. The GUTTER dialect is a particular favorite of frustrated FORTH programmers who end up using this language.

LAIDBACK - This language was developed at the Marin County Center for T'ai Chi. Mellowness and computer programming (now defunct), as an alternative to the more intense atmosphere in the nearby Silicon Valley. The center was ideal for programmers who liked to soak in hot tubs while they worked. Unfortunately few programmers could survive because the center outlawed Pizza and Coco-Cola in favor of Tofu and Perrier. Many mourn the demise of LAIDBACK because of its reputation as a gentle and non-threatening language since all error messages were in lower case letters. For example, LAIDBACK responded to syntax errors with the message, "I hate to bother you, but I just cannot relate to that. Can you find the time to try it again?"

LITHP - This otherwise unremarkable language is distinguished by the absence of an "S" in its character set. Programmers and users must substitute "TH". LITHP is said to be useful worthd prothething. This language was developed in San Francisco.

REAGAN - This language was also developed in California , but is now widely used in Washington, D.C. It is the current subset of the international bureaucratic language known as DOUBLESPEAK. commands include REVENUE_ENHANCEMENT, STOCKMAN, CAP_WEINBERGER, MALCOMB_BALDRIDGE, CABINET, CHOP_WOOD, LAXALT and SCENARIO. WAIT and BURFORD have been removed from the commands while there is a current effort to add MEESE. The operating systems used is NEW_RIGHT and the designated memory is THE_RANCH. The compile SCENARIO is a compile with NANCY followed by a link with BONZO resulting in a SNOOZE. COMMIES (program bugs) are removed with the GRANADA command. A REAGAN program commences with LANDSLIDE AND terminates with SENILITY.

RENE -Named after the famous French philosopher and mathematician Rene DesCaters, RENE is a language used for artificial intelligence. The language is being developed at the Chicago Center of Machine Politics and Programming under a grant from the Jane Byrne Victory Fund. A spokesman described the language as, "Just as great as dis (sic) great city of ours." The center is very pleased with progress to date. They say they have almost succeeded in getting a VAX to think. However, sources inside the organization say that each time the machine fails to think it ceases to exist.

SATRE - Named after the late existential philosopher, SATRE is an extremely unstructured language. Statements in SATRE have no purpose; they just are. Thus SATRE programs are left to define their own functions. SATRE programmers tend to be boring and depressing and are no fun at parties.

SIMPLE - SIMPLE is an acronym for Sheer Idiot's Monopurpose Programming Linguistic Environment. This language, developed at Hanover College for Technological Misfits, was designed to make it impossible to write code with errors in it. The statements are, therefore, confined to BEGIN, END, and STOP. No matter how you arrange the statements, you cannot make a syntax error.

SLOBOL - SLOBOL is best known for the speed, or lack of it, of the compiler. Although many compilers allow you to take a coffee break while they compile, the SLOBOL compiler allows you to travel to Columbia to pick up the coffee.Forty-three programmers are known to have died of boredom sitting at their terminals while waiting for a SLOBOL program to compile.

VAL6OL - From its modest beginnings in Southern California's San Fernando Valley, VAL6OL is enjoying a dramatic surge of popularity across the industry. VAL6OL commands include REALLY, LIKE, WELL, Y'KNOW. Variables are assigned with =LIKE and =TOTALLY operators. Other operators include the California Booleans, AX and NOWAY. Repetitions of code are handled in FOR - SURE loops.

A sample program is shown below:

```
        LIKE, Y'KNOW (I MEAN) START
IF PIZZA            =LIKE BITCHEN AND
GUY                 =LIKE TUBULAR AND
VALLEY GIRL         +LIKE GRODY!$MAX(FERSURE)$$2
THEN

    FOR I =LIKE 1 TO ON$MAYBE 100
        DOWAH - (DITTY$$2)
        BARF(I) =TOTALLY GROSS(OUT)
    SURE

    LIKE BAG THIS PROGRAM
    REALLY
    LIKE TOTALLY (YA$KNOW)
    IM$SURE
    GOTO THE MALL
```

VALGOL is characterized by its unfriendly error messages. For example, when the user makes a syntax error the interpreter displays the message, "GAG ME WITH A SPOON!!!"

This bit of humor was copied form the Amarillo (Texas) 99/4 Users Group newsletter of November, 1985. They copied the article from an unspecified DEC Users Group newsletter.

ARTICLES

INTERRUPT NOTES (Part 1)

By Edgar Dobmann

An interrupt is an external means of interrupting the current sequence of operation which a computer is executing. Some interrupting schemes are software controlled (multi-tasking operating systems for example) while others are triggered by some kind of hardware mechanism. The TMS9900 provides for 2 non-maskable hardware interrupts and up to 16 levels of maskable interrupts. The system designer has a choice of whether or not to implement all of the maskable interrupts, but the 2 non-maskable interrupts are available on any system using the TMS9900.

All TMS9900 interrupts use two consecutive memory words (16 bits each) as the "interrupt vector". The first word identifies the Workspace area to be used by the interrupt handler and the second word identifies the entry address of the interrupt handler. When an interrupt occurs, the processor essentially forces a BLWP instruction to the interrupt vector. This passes the current Workspace Pointer, Return Address, and Status to the interrupt handler so that it can (if it chooses) return control of the processor back to the program that was running when the interrupt occurred.

The two non-maskable interrupts are called RESET and LOAD. You are already familiar with the RESET interrupt. This one is executed every time you turn power on the console. The power-on logic forces this interrupt to be generated so the computer will always be assured of starting at a known location. This interrupt causes the computer to run the Power Up routine which initializes VDP RAM, all peripheral controllers, and displays a startup menu. This is also the interrupt that is generated when you press the reset switch on a Widget.

The other non-maskable interrupt is also available on the TI-99/4A but it is not fully implemented in a standard console. The RESET interrupt vector uses locations >0000 and >0002 which are in console ROM. The LOAD interrupt vector uses locations >FFFC and >FFFE which are the last two words of addressable memory space. To make use of these, you of course need a means of getting access to these memory locations. One way is with any of the 32K memory expansions that are available for the /4A. The other way is to build your own external memory expansion with ROM or RAM. I once built a 24K ROM expansion module with these interrupt vectors stored in my ROM expansion. It worked just fine and I will describe it further in a future article.

In addition to the address space to store the interrupt vector, you will need some means of generating the interrupt. Fortunately TI has made this fairly simple for us. The LOAD pin of the TMS9900 processor is brought out to the I/O bus connector on the right side of the console and a 4.7 K ohm pull-up resistor exists on the computer mother board. This pull-up resistor keeps the LOAD signal at +5V under normal conditions. All that is required to force the interrupt is to somehow pull this line to ground momentarily.

The easist way to do this is with a small momentary pushbutton switch. Several suitable switches are available from Radio Shack (275-1547, 275-1571, or 275-1549). All that is needed is a single-pole single-throw (SPST) normally open (NO) momentary pushbutton switch of some type. If you use a SPDT switch like the 275-1549, be sure to wire up the normally open contacts.

The LOAD signal is on pin 13 of the I/O bus and all you need to do is connect a wire from one terminal on your switch to pin 13 and a wire from the other terminal of your switch to ground. Ground can be found on pins 21, 23, 25, and 27 of the I/O bus connectors. Any one of these 4 pins can be used for the ground connection. Then when you press your switch you will essentially be shorting pin 13 (LOAD) to ground. This will not cause any damage because of the 4.7 K ohm resistor which provides current limiting between the LOAD line and +5V power. In fact when you press your switch, the computer will execute a BLWP to locations >FFFC and >FFFE and if there is nothing stored there you will probably have to restart your computer so be careful about pressing the switch when you don't mean to.

This brings up the question of where to mount or install the switch. You can actually install it anywhere that you can gain access to the I/O bus. One frequently suggested place is the Speech Synthesizer. This is handy because you can either glue (or leave it dangling) the switch under the cover where it is out of the way unless you need it or you can drill a hole in the cover so it sticks up through the cover. Drilling the hole so the switch sticks up is handy but if you have little urchins who like to poke around while you are using the computer (as I do), you will probably want it a little less accessible.

Since I don't always have my Speech Synthesizer hooked up, I disassembled the "elephant paw" and soldered the switch to the I/O connector that plugs into the console or Speech Synthesizer. I made the wires long enough so that when I reassembled the unit, there were about 2 inches of wire sticking out of the elephant paw. These wires are underneath the flex cable and pressed up against it when I put the unit back together. The switch is soldered to the end of these wires and just kind of dangles under the flex cable. This keeps it out of the way but handily available whenever I need it.

If you are using some other means of system expansion besides the PEB (such as the freight trains, Micro Expansion Unit, etc), you can still mount a switch as I have described. If the manufacturer has kept his unit fully compatible, the pins will be as I have described here. The sketches show how you can locate the desired pins. Looking at the I/O bus on the right side of the console, even numbered pins are on the top row and odd numbered pins are on the bottom row. Pins 1 & 2 are on the extreme left and Pins 43 & 44 are on the extreme right. When soldering, be sure to use a low wattage iron (15 or 25 watts). Solder to the pins in a manner that will not interfere with the insertion of the device into the I/O connector. If you are not used to soldering delicate electronic devices, you should probably find a friend who is able to do this job for you.

If you like, you can solder a small capacitor across the terminals of the switch to help with debouncing. Something like a .001 or .01 microfarad capacitor should do just fine. Radio Shack types 272-154, 272-155, 272-156 are fairly small and are not polarity sensitive so they can be installed in either direction. The capacitor is not absolutely essential, but it may help improve the operation reliablity of your LOAD interrupt.

EXTEND THE USE OF TI-WRITER

By Allen Burt - England

TI-WRITER can be used for much more than just producing letters---a substitute for a typewriter. In the last article I described how to make use of the CONTROL "U" function in the Text Editor mode. This function can be used to extend the application of the system and to produce integrated documents of words and diagrams. For example, it is easy to show a Histogram (Bar Chart) like this:

Figure 1

This uses the CHR(124) obtained by using 'FUNCTION'[F and KEY 'A' for the verticles and the underline character CHR(95) (-[F and KEY 'U').

A useful tip when doing this type of exercise is that if you place the CHR(124)'s in the appropriate locations and wish to continue them downwards from the point indicated by the asterisk - just move the cursor down to the next line and press CONTROL [C and KEY '5' - this copies the line above onto that line. When you draw diagrams like this, it is better to insert a number of lines in order to have room to move around.

If you want to include a simple graph within your script, try doing this as shown in Figure 2 below:

Figure 2

A more sophisticated graph can be achieved using the above techniques. In the example I found the 'COPY' command very useful because having once obtained the required width - I only had to 'copy' down the required number of lines using [Control & KEY '5'. Remember that when you place a special set of codes at the start of the line, the space they occupy will not be recognized by the printer. That is the printed line will commence at the location of the first special code. This can place the numbers used in the graph in the wrong place. You have to enter your special codes at the point you wish the following characters to print. Thus what you see on the screen is not necessarily what you will get on the printout.

USING TI-WRITER TO DRAW A GRAPH

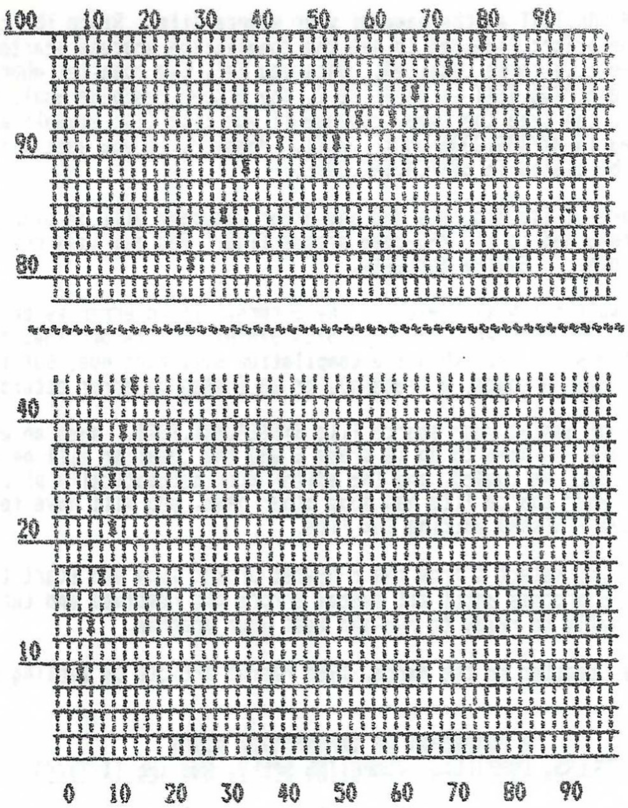

Figure 3

TI-WRITER can be used to draw graphs as Figure 3 above illustrates. The horizontal lines are achieved by setting the printer into an underline mode [CHR$(27);CHR$(45);CHR$(1)]. The line spacing is set to 7/72" [CHR$(27);"A";CHR$(7)] - this approximates to 1/10". If a CARRIAGE RETURN is placed at the point where the line should finish, the printer will draw a line to that point. The verticle lines are drawn by using CHR(124) - Function "A". As the printer normally prints at 10 characters to the inch, this will produce a grid of roughly 1/10" squares.

There are two points to watch using this procedure:

(1) If you do not want the underlining to start at the beginning of the printer line, the underline code must be placed at the the start of each line and cancelled at the end of each line before the carriage return. There is another means of achieving this and that is to set the left hand margin to the required position (on GEMINI printers this is CHR(27);"M";CHR(n) - n being the column to start printing . THIS CAN ONLY BE DONE USING THE PRINTER CODES, NOT BY SETTING TI-WRITER's TABS.

(2) The second point is that many printers do not align the characters in a bidirectional mode. YOU ARE ADVISED IN THE TI-WRITER MANUAL THAT FOR TABULATION IT IS ADVISABLE TO SET THE PRINTER TO A UNI-DIRECTIONAL PRINTING MODE.

Figure 4 below illustrates how a line will appear on the 4A screen.

```
90!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
^
!
! Actual point where the printing will start.
```

Figure 4

PASCAL NOTES

By Edgar Dohmann - JSC Users Group (JUG)

This month we will discuss the use of the Pascal Compiler. If you plan to compile your own Pascal programs, you should have at least two SS/SD (or equivalent) disk drives. While it is possible to configure a system that will allow you to perform a limited amount of compiling with only 1 SS/SD disk, I do not recommend it. The hassles and frustrations are not worth the results. Before you can compile a program, it must exist in source form as a text file. Regardless of what you plan to call the final compiled program, I recommend that you name the source file SYSTEM.WRK.TEXT until it is compiled and tested. While this is not absolutely necessary, it does allow more convenient use of some of the inherent P-System features.

From the command line of the P-System, press C for Compile to load the Compiler. The screen will display the message

Compiling...

while the Compiler is loaded. If a file called SYSTEM.WRK.TEXT exists, it will be automatically compiled and the P-code produced is saved as SYSTEM.WRK.CODE.
This is the first reason to use SYSTEM.WRK.TEXT as the name of your source file. Since the compiler automatically looks for a file with this name, it minimizes the effort required to get the compilation process started. If the compiler cannot find such a file, it will prompt you for the file to be compiled. Once you tell the compiler where the source or text file is located (for example if the source file MYPROG.TEXT is on drive 2, the proper response would be #5:MYPROG), you will be prompted for the target code file. You can type the same response if you like (for example a second response of #5:MYPROG will cause the target code to be placed in MYPROG.CODE on drive 2). However, if you just press <return> in response to the codefile prompt, the target code will be placed in SYSTEM.WRK.CODE.

While the compiler is running, a summary of its progress and any error messages will be displayed on the screen. When the compilation is finished, you will be returned to the P-System prompt line. You may then compile another program, test the program you compiled, or execute some other P-System activity.

Of course this assumes that your compilation finished without any errors. If an error is detected, the text surrounding the error is displayed and an error number is displayed. If the SYSTEM.SYNTAX file is on line, then a descriptive error message will be displayed. If the Q and L options are set, then the compilation will continue, but if the Q and L options are both off, you will be given the option of typing a space, a <return>, or an E when an error occurs.

This brings us to the second advantage of naming your source file SYSTEM.WRK.TEXT. When an error occurs and you want to correct your source file immediately, all you have to do is press E and the compiler will be exited and the Editor will be loaded. If SYSTEM.WRK.TEXT exists, it will be loaded, and the editor will automatically place you at the spot in your source file where the error was detected. If your source file has some other name, you will have to answer prompts both for loading the file and for saving it when you have finished editing the file.

After editing and saving your source file, select C from the P-System prompt line and start the compilation process over again. Repeat the editing and compiling process until all syntax errors are resolved and the compiler runs to completion without complaining. Then you will finally be ready to run and test your program.

We will continue our discussion of the Compiler in the March, 1986 Pascal article by looking further at some of the compile-time options available.

A POTPOURRI OF INFORMATION ABOUT ASSEMBLERS, COMPILERS, COMPUTING SPEED, AND THE TI-99/4A

By David Shultz, Steven Schwengels, Ron Tump
SST Software, Inc.

In 1983 we developed a basic compiler for the TI-99/4A. Before the coding could begin, we had to do an extensive study of the 4A console. Because of the limited memory of the console, we had to decide which commands were absolutely essential to a compiler, and which could be left out. We found out which commands were efficient, and which should be avoided. In this and future articles, we will share this information with you.

In this article we will first outline the development of programming languages and then present a study of computer times for T.I. Basic commands and the SST Basic compiled commands. Finally, we will present some examples.

Early computers evaluated, through hardware, a limited number of instructions. The programmer would code by writing a series of zeros and ones. This program, coded in what is known as machine language, would then be put into memory and executed. Machine language allows you to write very efficient programs. However, programmers found it difficult to code and debug in machine language. To make the process easier, programmers began using symbols for each machine instruction, which would later be translated into machine language. Such a symbolic (mneumonic) language is now called assembly language. These assembly languages have many of the same advantages and disadvantages of machine languages. However, it is somewhat easier to write and read programs in assembly language. Programs, known as assemblers, are used to translate code written in assembly language into machine language. The assembler often outputs the machine language translation (object program) onto a secondary storage device. Another program called a loader, places into memory the machine language translation of the user's program and then transfers control to it.

Later, high level languages were developed that were processed by programs called interpreters and compilers. Basic, for example, is often implemented by an interpreter. With BASIC, it is easier to write programs, but Basic is slow in execution. The reason is that the interpreter is a program that treats the user's program as data. That is, when a Basic program is executed the Basic interpreter must, every time a command is encountered, translate the command into machine language. As a result the same statement may be translated several thousand times in the same program.

To increase speed, compilers are often used in place of interpreters. Basic compilers are also available. Basic compilers convert a basic program directly into a machine language program, thus giving you the power and speed of machine language without actually having to write the program in machine language. The following steps are performed when a program is compiled and executed.

1. The program written in Basic (source language), which is called the source program, must be entered into the computer.

2. The translation program, called a compiler, is usually in the computer. The compiler uses the source program as data.

3. The compiler then translates the source program into a machine language program equivalent to the original program. This program , which is the output of the compiler, is called the object program. The object program often is stored on tape or disk for later use.

4. At execution time the object program is loaded into memory by a program called a loader, from a secondary storage device, and executed.

With a compiler a program needs to be translated only once. With an interpreter it must be translated each time it is executed. With an interpreter a statement may be translated thousands of times. Thus a compiler is preferred over an interpreter anytime speed is important.

The three of us were often using the TI-99/4A for scientific computation. However, we soon found that TI Basic was too slow for large problems. After experimenting with assembly language, we soon found that the 4A had a powerful processor. However, we also found it took a great deal of effort to code large problems. After checking with TI and discovering they were not going to write a Basic compiler, we decided to write our own. The result of our effort is the SST EXPANDED BASIC COMPILER.

To make the most efficient use of the TI-99/4A, we placed all floating point arrays into the TI console memory. The machine language code is placed into Mini-Memory or Expansion Memory. The SST EXPANDED BASIC COMPILER SYSTEM provides many of the advantages of both Basic and machine language. The SST SYSTEM is composed of three (3) main programs: a Compiler, a Loader, and an Editor. The Editor is designed to allow you to write and debug Basic programs using the editing and debugging features of TI Basic. Once the basic program is found to be correct, the Editor program is run and the Basic program is stored on disk for compiling. The compiler then translates the Basic code into machine language code and stores it permanently on disk. Anytime the program is to be run the loader will load it into memory and cause it to be executed.

While studying assembly language, we found that floating point computation is extremely slow. This is because the 4A works only in eight (8) byte floating point arithmetic. While this provides great accuracy, it also causes slow execution. This execution is slow, whether you access the floating point routines from TI Basic or assembly language. Therefore, to increase the speed and efficiency of your programs, we have included the option to code integer arithmetic. The TI processor has integer addition, subtraction, multiplication, and division commands. These instructions are extremely fast. In floating point, these operations must be done by subroutines, and are therefore slow. In addition, an integer variable requires only two (2) bytes, while a floating point variable requires eight (8) bytes. The GROM routines, like SIN or COS are extremely slow and little time is saved in Assembly Language over Basic. It may be worth the effort to write your own routines for these computations.

Table I contains a study of computing times for some TI Basic and SST EXPANDED BASIC commands in milliseconds. The method used to obtain the numbers is similar to that described by John Bow in the November, 1983 issue of the 99'ER MAGAZINE. The main difference is that we used a FOR LOOP of 30000 instead of 1000. A stop watch was used to record the times. As you can see from the table, the integer commands and graphic commands are fast, while the floating point and GROM routines are slow.

TABLE I

COMPUTING TIMES FOR TI BASIC AND SST BASIC OPERATIONS IN MILLISECONDS (MS)

OPERATION	SST	TI	% SST FASTER
ABS	0.11	7.5	68.2
CHAR	4.0	113.0	28.3
COLOR	0.32	47.0	147.0
GCHAR	0.32	47.0	147.0
VCHAR	0.32	48.0	150.0
IF	0.03	4.0	133.0
INTEGER +	0.03	5.0	166.0
INTEGER -	0.03	5.0	166.0
INTEGER *	0.19	6.0	31.6
INTEGER /	0.19	8.0	42.0
INTERGER ASSIGNMENT	0.03	3.0	100.0
FLOAT +	1.2	5.0	4.2
FLOAT -	1.2	5.0	4.2
FLOAT *	1.37	6.0	4.3
FLOAT /	3.5	8.0	2.3
FLOAT ASSIGNMENT	0.7	3.0	4.3
570 NEXT L@			
EXP	140.0	170.0	1.2
CALL POSITION	0.65	31.0	47.7

(SPRITE #,ROW,COLUMN)

Another example of how slow the GROM routines are is the involution routine (i.e., a&b). It turns out that a&b takes 382 ms, while the equivalent operation exp(b*log(a)) takes only 361 ms. That is, to save time you should use exp(b*log(a)), not a&b. If b is an integer, a&b takes only 67 ms or 2&4 takes 67 ms. However, 2*2*2*2 takes only 13 ms. Again, you should not use involution if b is an integer, but simply repeated multiplication.

To really decrease computing time, it is best to use as much integer arithmetic as possible. With the SST BASIC COMPILER, you can do integer arithmetic by simply using an % symbol after a variable name. Whenever the compiler sees an % symbol it knows it must do integer arithmetic. As an example of the speed of integer arithmetic, consider the following programs:

```
100 for I=1 TO 30000                100 LET I@=1
200 NEXT I                          110 LET J@=30000
300 STOP                            120 FOR I@=I@ TO J@
                                    130 NEXT I@
                                    140 STOP
TIME: 84 SECONDS                    TIME: 1.25 SECONDS
```

(NOTE: The SST Expanded Basic Compiler requires that you define all variables and constants at the beginning of the program. This is done to increase the efficiency of the compiler.)

Another example is one which appeared in the March, 1980, BYTE MAGAZINE. It is a program to generate prime numbers, and is often used as a benchmark program. The program was originally run in Basic on a TRS-80 computer. It took 8 hours, 12 minutes to check the first 10000 integers for prime numbers. We ran this program in TI Basic for over 4.5 hours and it was nowhere near completion. The compiled version took eleven (11) minutes and twenty (20) seconds to run. The following program was written to check the first 1000 integers.

SST BASIC PRIME PROGRAM

```
100 LET L@=6
110 LET E@=1
120 LET M@=1000
130 LET Z@=5
140 LET A@=1
150 LET N@=10
160 LET D@=1
170 LET B@=2
180 LET C@=2
190 LET A@=L@ TO M@
200 A@=A@+E@
210 D@=A@/C@
220 FOR Z@=B@ TO D@
230 Z@=Z@+E@
235 REM FOR T.I. BASIC
236 REM SHOULD BE
237 N@=INT(A@/Z@)
240 N@=A@/Z@
250 N@=N@*Z@
260 N@=A@-N@
270 IF N@<=0 THEN 300
280 NEXT Z@
290 PRINT A@
300 NEXT A@
310 STOP
```

TIME-TI BASIC: 1535 SECONDS VS. SST COMPILER: 18 SECONDS

The following benchmark program appeared in the January, 1985, issue of COMPUTE.: "MSX Is Coming" by Tom R. Halfhill. The program is a bubble sort.

```
100 PRINT "CREATING ARRAY"
110 DIM A(150)
120 FOR J=1 TO 150
130 A(J)=151-J
140 NEXT J
150 PRINT "SORTING"
160 EX=0
170 FOR K=0 TO 149
180 IF A(K)>A(K+1)THEN T+A(K):
A(K)=A(K+1): A(K+1)=T: EX=1
190 NEXT K
200 IF EX<>0 THEN GOTO 160
```

```
100 LET A$="CREATING ARRAY"
110 LETB$="SORTING"
120 DIM A@(150)
130 LET E@=0
140 LET J@=0
150 LET K@=1
160 LET L@=150
170 LET M@=151
180 LET AA@=0
190 LET AB@=149
200 LET X@=0
210 LET Y@=0
220 LET Z@=0
230 PRINT A$
240 FOR J@=K@ TO L@
250 A@(J@)=M@-J@
260 NEXT J@
270 PRINT B$
280 E@=AA@
290 FOR J@=AA@ TO AB@
300 X@=J@+K@
310 Y@=A@(J@)-A@(X@)
320 IF Y@<=0 THEN 370
330 Z@=A@(J@)
340 A@(J@)=A@(X@)
350 A@(X@)=Z@
360 E@=K@
370 NEXT J@
380 IF E@=00 THEN 400
390 GOTO 280
400 STOP
```

The times in minutes : seconds are:

SST EXPANDED BASIC COMPILED with integer arithmetic	0:31
SST EXPANDED BASIC COMPILED with floating point arithmetic	2:05
IBM PC	5:45
GOLDSTAR MSX	6:20
APPLE II PLUS	6:24
APPLE II PLUS	6:33
COMMODORE VIC-20	6:34
IBM PCjr	6:59
COMMODORE 64	7:02
COMMODORE 8032	7:16
TRS-80 COLOR COMPUTER	8:01
COMMODORE 16	8:35
COMMODORE PLUS 4	8:36
ATARI 800XL	8:55
ATARI 800	9:00
TI-99/4A BASIC	12:58

As you can see the SST EXPANDED BASIC COMPILER greatly improved the performance of the TI-99/4A. A compiler could also improve the times of the other computers. IN MARCH, 1983, 99er or HOME COMPUTER MAGAZINE today, there was an article called "Matrix Muncher", by Cheryl Whitelaw and the HCM Staff. The article contained a program to solve the simultaneous linear equations. We modified the program to handle up to 21 equations in 21 unknowns, and then compiled and ran the program to compare computing times. We ran one system of four (4) simultaneous equations and another of eleven (11) simultaenous equations. The 4x4 system took eight (8) seconds in T.I.

Basic and less than one second when compiled. The 11x11 system took 125 seconds in T.I. Basic and 11 seconds for the compiled program. The compiled program is over 11 times faster, not including the long pre-scan time in T.I. Basic. The SST EXPANDED BASIC program follows:

```
70 LET R$="INPUT N"
75 LET S$="INPUT A(I,J)"
80 LET T$="INPUT RHS"
85 LET U$="SOLUTION IS"
90 LET V$="ROW = "
95 LET W$="NO UNIQUE SLOUTIO
N"
100 DIM A(21,21)
110 DIM X(21)
120 DIM B(21)
130 DIM Z(21,21)
140 LET AB=1
150 LET AZ+1
155 LET Q=0
160 LET T+1
170 LET AA=1
180 LET N@=1
190 LET I@=1
200 LET A@=1
210 LET J@=1
220 LET K@=1
222 LET L@=1
224 LET T@=1
230 LET M@=1
240 LET AL@=1
250 LET CL@=1
260 CALL CLEAR
265 PRINT R$
267 REM INPUT # OF EQUATION
S
270 INPUT N$
275 PRINT S$
277 REM INPUT MATRIX
280 FOR I@=A@ TO N@
285 PRINT V$
288 PRINT I@
290 FOR J@=A@ TO N@
300 INPUT A(I@J@)
310 Z(I@,J@)=A(I@,J@)
320 NEXT J@
330 NEXT I@
335 PRINT T$
337 REM INPUT RIGHT
        HAND SIDE
340 FOR I@=A@ TO N@
350 INPUT B(I@)
360 NEXT I@
370 FOR L@=A@ TO N@
380 IF Z(L@,L@)<=0 THEN 384
382 GOTO 390
384 IF Z(L@,L@)>=0 THEN 670
390 Z(L@,L@)=AA/Z(L@,L@)
400 FOR K@=A@ TO N@
410 T@=K@-L@
420IF T@<=0 THEN 424
422 GOTO 430
424 IF T@>=0 THEN 500
430 Z(K@,L@)=Z(K@,L)$Z(L@,L
    @)
440 FOR M@=A@ TO N@
450 T@=M@-L@
460 IF T@<=0 THEN 464
462 GOTO 470
464 IF T@>=0 THEN 490
470 T+Z(K@,L@)$Z(L@,M@)
480 Z(K@,M@)=Z(K@,M@)-T
490 NEXT M@
500 NEXT K@
510 FOR M@=A@ TO N@
520 T@=M@-L@
530 OF T@<=0 THEN 534
532 GOTO 540
534 IF T@>=0 THEN 560
540 T+Z(L@,L@)$Z(L@,M)
550 Z(L@,M@)=Q-T
560 NEXT ,M@
570 NEXT L@
575 PRINT U$
577 REM SOLUTION VALUES
580 FOR I@=A@ TO N@
590 X(I@)=Q
600 FOR J@=A@ TO N@
610 T+Z(I@,J@)$B(J@)
620 X(I@)=X(I@)+T
630 NEXT J@
640 PRINT X(I@)
650 NEXT I@
660 GOT 900
665 REM SWITCH ROWS
670 AL@=L@+A@
680 FOR CL@=AL@ TO N@
690 IF Z(CL@,L@)<=0 THEN 694
692 GOTO 700
694 IF Z(CL@,L@)>=0 THEN 890
700 FOR M@=A@ TO N@
710 AZ=Z(L@,M@)
720 Z(L@,M@+=Z(CL@,M@)
730 Z(CL@,M@)=AZ
740 NEXT M@
750 AB=B(L@)
760 B(L@)=B(CL@)
770 B(CL@)=AB
780 GOTO 390
890 NEXT CL@
895 PRINT W$
900 STOP
```

In future articles we will discuss more of the features of the Expanded Basic Compiler including the differences between Extended Basic and SST Expanded Basic.

A CONDENSED DISK CATALOG PROGRAM

By John Moody

EDITOR'S NOTE: John has given permission for us to print the listing of his program. If you wish a copy of the program, forward a formatted SS/SD in a mailer with the return postage included to our letterhead address. I will place a copy of John's Condensed Disk Catalog on yourtr disk and return it by mail. Thank you. Don Veith, Editor.

```
100 . CONDENSED DISK MANAGER
110 . EXTENDED BASIC
120 . WRITTEN BY:
      JOHN MOODY
      409 HANSEL COURT
      FLORENCE,AL. 35630
      (205)766-5173
130 CALL CHAR(96,"FFFFFFFFFF
FFFEFCFCFEFFFFFFFFFFFFFFFFFF
FEFEFF7F3F3F7FFFFFFFFFFFFF")
.DISK
890 S=0 :: A=80 :: GOTO 1300
900 PRINT #2:"DISKNAME= ";A$
(0,0);
910 IF O=1 THEN PRINT #2:CHR
$(13):: GOTO 930
920 PRINT #2:TAB(A);"DISKNAM
E= ";A$(1,0)
930 PRINT #2:"AVAILABLE= ";F
(0,0);"USED= ";J(0,0)-F(0,0)
;
940 IF O=1 THEN PRINT #2:CHR
```

```
140 CALL CHAR(100,"FFC0BFA0A
0A0A0A0A0A0A0A0A0BFC0FFFF03F
D0505050505050505050505FD03FF"
).SCREEN
150 CALL CHAR(104,"000000000
0001F1010107F40405C407F00000
0000000F8080808FE02030302FE"
).PRINTER
160 CALL LOGO :: DISPLAY AT(
20,12):"PRESENTS" :: CALL SP
RITE(#4,96,2,24,112):: DISPL
AY AT(9,11):"CONDENSED": :TA
B(9);"DISK CATALOGER"
170 FOR I=1 TO 500 :: NEXT I
 :: CALL MAGNIFY(3):: CALL L
OCATE(#4,40,125,#1,112,78,#2
,112,126,#3,112,174):: FOR I
=1 TO 900 :: NEXT I
180 CALL DELSPRITE(ALL):: FO
R I=1 TO 8 :: CALL COLOR(I,2
,1):: NEXT I
190 . START
200 CALL CLEAR :: CALL SCREE
N(15):: DISPLAY AT(2,3):"CON
DENSED DISK CATALOGER"
210 DISPLAY AT(4,3):"1 = CAT
ALOG DISK": :TAB(3);"2 = DEL
ETE FILE": :TAB(3);"3 = QUIT
"
220 CALL KEY(0,K,S):: IF K=4
9 THEN 300 ELSE IF K=50 THEN
 2000 ELSE IF K=51 THEN 5000
 ELSE 220
300 . CATALOG DISK
310 CALL CLEAR
320 DIM A$(1,2),J(1,2),F(1,2
),B$(1,127),C(1,127),D(1,127
),E(1,127),TY$(5)
325 CALL MAGNIFY(4):: CALL S
PRITE(#4,96,2,24,24,#5,96,2,
24,88,#6,96,2,24,152,#7,96,2
,24,208)
326 DISPLAY AT(9,3):"1";TAB(
11);"2";TAB(19);"3";TAB(26);
"4"
330 DISPLAY AT(12,1):"NUMBER
 OF DISK TO CATALOG":TAB(8);
"(1/2) 2" :: ACCEPT AT(13,14
)SIZE(-1)VALIDATE("12"):O
335 TY$(1)="DIS/FIX" :: TY$(
2)="DIS/VAR" :: TY$(3)="INT/
FIX" :: TY$(4)="INT/VAR" ::
TY$(5)="PROGRAM"
336 ON ERROR 2900
340 DISPLAY AT(12,1):"DISK T
O CATALOG":TAB(8);"(1234) 1"
 :: ACCEPT AT(13,15)SIZE(-1)
VALIDATE("1234"):N
350 . OPEN DISK
360 OPEN #1:"DSK"&STR$(N)&".
",INPUT ,RELATIVE,INTERNAL
370 IF K=89 THEN T=1 ELSE T=
0
380 INPUT #1:A$(T,0),J(T,0),
J(T,0),F(T,0)
390 FOR I=0 TO 126
400 INPUT #1:B$(T,I),C(T,I),
D(T,I),E(T,I)
410 IF B$(T,I)="" THEN 420
415 NEXT I
420 CLOSE #1
430 IF O=1 OR O=0 THEN 500
440 DISPLAY AT(12,1):"READY
TO CATALOG 2ND DISK?":TAB(8)
;"(Y/N/R) "
450 O=0 :: CALL KEY(0,K,S)::
 IF K=89 THEN 340 ELSE IF K=
78 THEN 450 ELSE IF K=82 THE
N 180 ELSE 450
```

```
$(13):: GOTO 960
950 PRINT #2:TAB(A);"AVAILAB
LE= ";F(1,0);"USED= ";J(1,0)
-F(1,0)
960 PRINT #2:"FILENAME   SIZ
E  TYPE       P";
970 IF O=1 THEN PRINT #2:CHR
$(13):: GOTO 990
980 PRINT #2:TAB(A);"FILENAM
E  SIZE  TYPE
990 PRINT #2:"
              ---------- ---
              ";
1000 IF O=1 THEN PRINT #2:CH
R$(13):: GOTO 1020
1010 PRINT #2:TAB(A);"
                     -------
1020 FOR I=0 TO 126
1030 IF B$(0,I)="" AND B$(1,
I)="" THEN 1210
1035 IF B$(0,I)="" THEN 1105
1040 PRINT #2:B$(0,I);TAB(11
);D(0,I);TAB(16);TY$(ABS(C(0
,I)));
1050 IF ABS(C(0,I))=5 THEN 1
080
1060 X$=" "&STR$(E(0,I))
1070 PRINT #2:TAB(23);E(0,I)
;
1080 IF C(0,I)>0 THEN 1100
1090 PRINT #2:TAB(29);"Y";
1100 IF O=1 THEN 1200
1105 IF B$(1,I)="" THEN O=1
1110 PRINT #2:TAB(A);B$(1,I)
;TAB(A+10);D(1,I);TAB(A+15);
TY$(ABS(C(1,I)));
1120 IF ABS(C(1,I))=0 THEN 1
150
1130 IF E(1,I)=0 THEN 1150
1140 PRINT #2:TAB(A+23);E(1,
I);
1150 IF C(1,I)>0 THEN 1200
1160 PRINT #2:TAB(A+29);"Y";
1200 PRINT #2 :: NEXT I
1210 PRINT #2:CHR$(27)&CHR$(
64): : : :: CLOSE #2
1220 GOTO 200
1300 CALL CLEAR :: DISPLAY A
T(3,8):"FORMAT CHOICE" :: "1
= REGULAR CATALOG FORMAT": :
"2 = CONDENSED CATALOG FORMA
T"
1310 DISPLAY AT(9,1):"CHOICE
:" :: ACCEPT AT(9,9)SIZE(-1)
VALIDATE("12"):PM
1320 IF PM=13 THEN 1310
1330 ON PM GOTO 900,1500
1500 PRINT #2:"DISKNAME= ";A
$(0,0)&RPT$(" ",41);:: IF O=
1 THEN PRINT #2: :: GOTO 151
0 ELSE PRINT #2:"DISKNAME= "
;A$(1,0)
1510 FOR I=0 TO 126 STEP 3
1520 FOR Q=0 TO 1
1530 FOR W=0 TO 2
1540 IF O=1 OR B$(Q,W+I)=""
THEN PRINT #2:RPT$(" ",17)::
: GOTO 1550 ELSE PRINT #2:B$
(Q,W+I)&RPT$(".",11-LEN(B$(Q
,W+I)));:: PRINT #2,USING "#
###  ":D(Q,W+I);
1550 NEXT W :: PRINT #2:RPT$
(" ",10);:: NEXT Q
1555 IF B$(1,W+I)="" THEN O=
2
1560 IF B$(0,W+I)="" AND B$(
1,W+I)="" THEN 1580
1570 PRINT #2: :: NEXT I
1580 FOR Q=0 TO 1 :: FOR W=0
 TO 126 :: B$(Q,W)="" :: NEX
```

```
500 .OUTPUT
510 CALL CLEAR
515 CALL SPRITE(#8,100,2,128
,64,#9,104,2,128,160)
520 DISPLAY AT(12,1):"OUTPUT
TO SCREEN OR PRINTER?":TAB(
8);"(S/P) P" :: ACCEPT AT(13
,14)SIZE(-1)VALIDATE("SP"):X
$
530 IF X$="P" THEN 800
540 CALL CLEAR :: CALL DELSP
RITE(ALL)
550 T=0
560 IF K=67 THEN T=1
570 DISPLAY "DISKNAME= ";A$(
T,0):"AVAILABLE=";F(T,0);"US
ED=";J(T,0)-F(T,0)
580 DISPLAY "FILENAME   SIZE
TYPE      P":"
                 "---------- --
590 FOR I=0 TO 126
610 DISPLAY B$(T,I);TAB(11);
D(T,I);TAB(16);TY$(ABS(C(T,I
)));
615 IF LEN(B$(T,I))=0 THEN 7
00
620 IF ABS(C(T,I))=5 THEN 64
0
625 Q$=" "&STR$(E(T,I))
630 DISPLAY SEG$(Q$,LEN(Q$)-
2,3);
640 IF C(T,I)>0 THEN 660
650 DISPLAY TAB(27);"Y";
660 DISPLAY :: NEXT I
700 PRINT : : : : :
710 IF Q=0 THEN 740
720 DISPLAY AT(22,1):"PRESS
<M> FOR MENU";"PRESS <P> TO
PRINT" :: ACCEPT AT(23,19)SI
ZE(1)VALIDATE("MP"):X$
730 IF X$="" THEN 720 ELSE I
F X$="P" THEN 790 ELSE IF X$
="M" THEN 200
740 DISPLAY AT(22,1):"PRESS
<C> FOR NEXT DISK" :: Q=1 ::
 CALL KEY(0,K,S):: IF S=0 TH
EN 740
750 IF K=67 THEN 540 ELSE 74
0
790 CALL CLEAR
800 DISPLAY AT(12,1):"PRINTE
R DEVICE NAME: ":P$ :: ACCEP
T AT(13,1)SIZE(-20):P$
810 CALL DELSPRITE(ALL):: CA
LL CLEAR
820 DISPLAY AT(3,8):"PRINT C
HOICES:": :"1 = REGULAR": :"
2 = CONDENSED TYPE": :"3 = S
UPER CONDENSED TYPE"
830 DISPLAY AT(12,1):"CHOICE
: " :: ACCEPT AT(12,9)SIZE(1
)VALIDATE("123"):PM
840 OPEN #2:P$,DISPLAY ,VARI
ABLE 255
850 ON PM GOTO 855,860,880
855 A=40 :: GOTO 900
860 PRINT #2:CHR$(27)&CHR$(6
6);&CHR$(3);
870 A=80 :: GOTO 900
880 PRINT #2:CHR$(27)&CHR$(8
3)&CHR$(1)&CHR$(27)&CHR$(49)
&CHR$(27)&CHR$(15);
T N ::-NEXT Q
1590 PRINT #2:CHR$(27)&CHR$(
64):: CLOSE #2
1600 GOTO 200
2000 CALL CLEAR :: DISPLAY A
T(3,5):"DELETE FILES:" :: DI
SPLAY AT(12,1):"MASTER DISK
(1-4) 1"
2010 ACCEPT AT(12,19)SIZE(-1
)VALIDATE("1234"):N
2020 OPEN #1:"DSK"&STR$(N)&"
.",INPUT ,RELATIVE,INTERNAL
2030 INPUT #1:J$,Y,Y,N :: DI
SPLAY AT(15,1):"DISKNAME= ";
J$
2040 DISPLAY AT(16,1):"FILEN
AME: ".:: ACCEPT AT(16,11):X
$
2050 DELETE "DSK"&STR$(N)&".
"&X$
2060 CLOSE #1 :: GOTO 200
2900 CALL ERR(L,M):: IF L<>1
30 THEN RETURN
3000 DISPLAY AT(12,1):"CHECK
 DRIVE ";N;"OR DISK":"ERROR
DETECTED" :: FOR I=1 TO 500
:: NEXT I
3010 DISPLAY AT(12,1):"PRESS
 <ENTER> WHEN READY" :: CALL
 KEY(0,K,S):: IF K<>13 THEN
3010
3015 ON ERROR 3030
3020 CLOSE #1 :: RETURN 336
3030 CALL ERR(L,M):: RETURN
336
5000 CALL CLEAR :: CALL LOGO
 :: DISPLAY AT(19,10):"JOHN
MOODY":TAB(7);"409 NANSEL CO
URT":TAB(6);"FLORENCE, AL. 3
5630":TAB(9);"(205)768-5173"
5010 DISPLAY AT(10,11):"THAN
K YOU." :: END
9990 SUB LOGO
9995 CALL CLEAR
10000 CALL CHAR(108,"0701010
1010101010101131313B1F0FF0C
0C0C0C0C0C0C0C0C0C0C0C0B0B00
0").J
10010 CALL CHAR(112,"FF7F707
0707070707070707070FFFE0F
83C1C0E0E0E0E0E0E0E0E1C3CF8E
0").D
10020 CALL CHAR(116,"F870707
87C74767273717170707070F81F0
E0E1E3E2E6E4ECE8EBE0E0E0E0E1
F").M
10030 CALL SCREEN(5):: FOR I
=1 TO 8 :: CALL COLOR(I,16,1
):: NEXT I :: CALL MAGNIFY(4
)
10040 CALL SPRITE(#1,108,15,
96,1,0,10,#2,112,15,1,113,15
,0,#3,116,15,96,255,0,-15)::
 FOR I=140 TO 21 STEP -7
10050 CALL SOUND(-150,220#I/
20,6,440#I/20,9):: NEXT I ::
 CALL MOTION(#1,0,0,#2,0,0,#
3,0,0)
10060 CALL LOCATE(#1,96,65,#
2,96,113,#3,96,161):: DISPLA
Y AT(18,10):"SHOALS'99ERS"
10070 SUBEND
```

OUT OF ROOM

FOR THIS MONTH, FOLKS ....
THANKS.
DON & LUCI

THE 99'ERS ASSOCIATION
ATTN: LUCI VEITH
3535 SO. H ST., #93
BAKERSFIELD, CALIF. 93304

PLEASE MAIL CK./M.O. FOR SUBSCRIPTION CHOICE SELECTED ABOVE TO:

SUBSCRIPTION TYPE	AMOUNT	CHOICE
THIRD CLASS - BULK RATE	$12.00	
FIRST CLASS - US AND CANADA	$17.00	
FIRST CLASS - OVERSEAS	$22.00	

CITY: _____ STATE _____ ZIP _____

ADDRESS: _____

NAME: _____ DATE: _____

SUBSCRIPTION FORM
FOR "THE NATIONAL NINTY-NINER"
PUBLISHED BY THE 99'ERS ASSOCIATION

THE NATIONAL 99'ER
3535 SO. H ST., #26
BAKERSFIELD, CA 93304

FIRST CLASS

MR. BILL PECHNIK
1467 CARMI DR.
PENTICTON BC CANADA V2A 4R9