M

WINNIPEG

Newsletter

March's Newsletter

The Winnipeg 99/4 User Group is a non-profit organization formed to meet the needs of Manitoba based Texas Instruments users. The content of this publication does not necessarily represent the view of the Winnipeg 99/4 User Group. This newsletter is one form of communication to keep Manitobans up on Texas Instruments Computers and its clones.

Next General Meeting - Date : April 3rd, 1986
Time : 7:00 P.M.
Place: Winnipeg Centennial Library
2nd Floor, Assembly Room

Executive 1986:

President:	Jim Bainard	334-5987
Treasurer:	Bill Quinn	837-7758
Newsletter Editor, Book and User Programs Librarian:	Mike Swiridenko	772-8565
Contributing Editor:	Paul Degner	586-6889
Inter-Group Representative and Newsletter Publisher:	Dave Wood	895-7067
ASSISTANT NEWSLETTER Publisher:	HANK Derkson	
Systems Co-Ordinator:	Rick Lumsden	253-0794
Educational Co-Ordinator:	Sheldon Itscovich	633-0835
Public Domain Librarian: 822 Henderson Hwy.	Gordon Richards	668-4804
Module Librarian:	Peter Gould	889-5505

Mailing Address: NEWSLETTER EDITER
WINNIPEG 99/4 USERS GROUP
P.O.B. 1715
WINNIPEG, MANITOBA
CANADA, R3C 2Z6

BBS #: (204)-889-1432 SYSOP: Charles Carlson
HOURS: CLOSED PERMANENTLY, during FEBUARY!

EDITORIAL COMMENTS:

Welcome back to another edition of the newsletter. This month I've inserted some interesting items from some back issues of the Clubline-99 magazine. These items include an article about Program Files using the Editor/Assembler, two TI-Forth screens of speech words, and an Extended Basic program which will print nifty little diskette labels. Hank Derkson passed along a little program written by a "FREEWARE" author by the name of Barry A. Traver. I'll let you figure out what it does. Paul provides the second part of a two part article on computer interrupts. I talk about program control structures in the XBasic Helpfile, and present a review of a book Paul loaned me, "Cracking the 99/4A", in the reviews column. I also have to appologize because I have to put off my promised article on software design until another month.

If you have a review, user hints, or helpful programming tips, get them to me for the next newsletter. The deadline that I have set for submissions is one week before the date of the group's meeting. Thanks go to all who have submitted items for this issue of our newsletter.

MISCELLANIA:

Miscellaneous news and reminders.

R.I.P.-The SMORGAS BOARD closed last month. Charles Carlson sold his modem. It was nice while it lasted.
Last meeting the tape of the Chicago Faire was not shown as the video machine did not make it to the meeting. Hopefully, this month someone can coax it out from hiding.
Good news on the widget that Sheldon Itscovitch has been working on. It is finally operational! If interested in a six slot widget contact Sheldon.
Steve Zabarylo has built a working 32K expansion for console system. Steve will make this expansion for others. If interested talk to Steve. A 32K expansion without disk drive allows a memory area for user written assembly language programs. To access this memory area you would need a Mini-Memory module or XBasic Module. The MM module and line-by-line assembler lets you program directly in assembly language, while XBasic would allow you to use Load and Peek to put assembly code in this area.

READER RESPONSE:

Thanks goes to Brian Lesko for pointing out to me an article in the March 1986 issue of Compute! from which the following exerpt comes. The article is entitled d 'The Future of Mass Storage' by Selby Bateman, Features Editor.

"Some computer experts believe that by the year 2000, the days of magnetic computer data storage may be only a historical footnote. Major advances in the use of low power lasers in audio and video players are being quickly applied to computer technology. One of the hottest consumer electronics items in recent years is the audio compact disc (CD). And later this year, computer users will get a chance to see what laser technology can do when linked with a computer - virtually *any* computer - as a CD-ROM storage device.
The basic principle of CD ROMs is similar to the audio CD. A low-power laser beam reads microscopic pits that have been burned into the disc itself. These pits - representing a series of ones and zeros - contain the data that in a magnetic medium would be formed by the arrangement of magnetic particles. The 4.7-inch CD-ROM discs contain a whopping 550 megabytes of data per disc. The first applications are likely to be encyclopedias, such as the nine-million-word *Academic American Encyclopedia,* a 21-volume reference work that fits on just a quarter of one CD-ROM disc.
The biggest problem with CD-ROM technology at this point is that the devices are read-only. Unlike the magnetic particles on a megnetic floppy or hard disk, once the pits are burned into the surface of a CD, they can't be altered. But that limitation is already being challenged in the labs.
Sony recently announced a *writeable* 12-inch optical disc system that can hold up to 3.2 gigabytes of information. The disc is composed of two metalic elements sealed in a polycarbonate plastic. The laser beam writes information on the disc by turning the elements into an alloy which has different reflective properties. 'This direct-seal method is more reliable and less costly than melt-type or bubble formation methods, which form gases during the writing process,' Robert Mesnik, Sony Information Products product manager. 'The direct-seal method has a simple structure with no air spaces which can cause degradation of information over time.'
This is a form of WORM (Write-Once, Read-Mainly) storage technology which offers high-densty storage options for a variety of markets. The next step, however, is to create an optical technology that allows a laser to repeatedly write information on the same disc. Although not yet fullydeveloped, an eraseable, reuseable 5 1/4-inch optical disc has been announced by Maxell for distribution in 1987. But for now, CD-ROMS will remain read-only reference and archival storage devices.
One of the first CD-ROM models debuting in 1986 is Toshiba's XM-1000 drive, which will be able to access digital computer dataand also play music - that is, it will be both an audio accessory and a computer peripheral. The unit will have a storage capacity of 600-680 megabytes and may enter the rettail market at close to $1,000. Sony will also market its CDU-1 CD-ROM player in 1986."

LANGUAGES

BASIC
XBASIC
FORTH
ASSEMBLY
C99
PASCAL
LOGO

LANGUAGES

```
*************
*JFHLDLISBV*
*BPOZACBAFP*
*PGYCDASOPV*
*OYSUSSRNYR*
*IAIIETXXOZ*
*PRCMHNZBVZ*
*ELBCCFQAEN*
*WLYVLDVS9C*
*YWCUQWVI9J*
*TVIVLOKCCL*
*************
```

QUID PRO QUO

[illegible]

The lion hasn't returned! Maybe we should send out a posse? Noah, where are you? On a serious note, Charles Carlson has guillotined his Volksmodem leaving millions and millions of TIers without a home. Therefore I would suggest a fundraiser to 'Raise The Smorg'. In any case, I can get more work done now during the evenings so I guess it can't be all that bad. Two other BBSs that support the TI community are Tronica Information Network (9438517) and Tec Voc Computer Services (8857921). T.I.N is operated by the folks at Tronica on McPhillips. Don Ponchinko, System Operator, is running a TBBS program in a IBM PC supporting 20 megabytes of storage with 300/1200/2400 bps access. There is a one time fee of twenty dollars which I guess is to pay for TBBS enhancements such as a multi-user system soon to be released. TBBS has a TI download section containing over a megabyte of programs for you to tap. T.V.C.S. on the other hand costs you nothing to access. Just logon and read the bulletin and it will tell you how to gain access to the system. I'm the grand TI poobah there and currently the only TI user on the board, just imagine the great email I can have sent to myself! I will start doing massive injections of programs to T.V.C.S as currently the bare minimum exists! I hope to see you all on either of the two boards!

Briefs:

The final chapter of Bill and the plight of his interruptable interrupts!

INTERRUPTS cont'd

The second data word, which for reset will be >0002, contains the memory location of the first instruction of the interrupt routine's program coding. The processor will begin program execution at the address contained in >0002 immediately after the workspace changeover.

TI refers to the memory addresses containing the new workspace and program location addresses as "context vectors". You may have seen this term used to explain the "bullwhip" instruction (Branch and Load Workspace Pointer - BLWP). The functions of the bullwhip instruction and interrupt are identical except that the BLWP is encountered within a program while interrupts are generated outside of the program. Both return to the original program via the Return with Workspace Pointer (RTWP) instruction.

There are sixteen levels of interrupts available on the TMS 9900 microprocessor. Each level has its own allotted context vector at an address predetermined by the silicon substrate of the microchip. Each level is also a priority, which allows more important program routines to take priority over less important ones; so one interrupt can actually "interrupt" another interrupt. The interrupt priorities of the TMS 9900 are numbered 0-15; the lower the number, the higher the priority.

The programmer is able to decide what priority of interrupts will be acted upon by the processor by using the Load Interrupt Mask Immediate (LIMI) instruction. Within the status register that is finely etched upon the TMS 9900 silicon are four status bits that form the interrupt mask. The programmer set/resets these bits via an LIMI instruction to tell the processor which interrupts to act upon and which to ignore. The processor constantly samples the interrupt priorities of equal or higher precedence as specified by the mask.

Again using the pizza delivery example, suppose that when deliveries fall behind more than thirty minutes, the driver has been instructed to ignore green colored flares and only act upon red flares. When the critical time delay is reached, the driver puts on a pair of green-lensed glasses. Now he won't be able to see the green flares (nor green traffic signals for that matter, but that's not the point), but he will still see the red ones, which perhaps may indicate the store has been robbed and he is needed at once. The green glasses would be analogous to the status register interrupt mask.

The 99/4 is set-up to use interrupts numbered two or lower. All external interrupts are done by means of the 9901 microcircuit soldered within the console. The versatile chip is called a Programmable System Interface (PSI). The PSI is capable of recognizing its own set of interrupt levels, but all of these are still presented to the processor as level two interrupts. I'm not going to give you an excessive amount of information on the PSI in this article, I just want you to have a noding acquaintance with the interrupt PROCESS within your system. The PSI, however, is of great use to the advanced Assembly language programmer and responds to Communications Register Unit (CRU) instructions such as SBO, SBZ, TB, STCR, and LDCR. The Editor/Assembler manual doesn't tell you all that the PSI is capable of doing for you; I had to get a copy of the 9901 designer manual to learn all its secret powers. The PSI will provide an advanced interrupt capability to the user who has a solid back ground in computer hardware and needs to use his 99/4 for external control. However, there is one interrupt that the novice user can access with a very simple piece of hardware that he can fabricate for under two dollars.

Reset is the most powerful interrupt in your home computer. However, it is of limited used to you because its context vector is in ROM, frozen into a factory chosen state of existence. Oh!, if only we were allowed to change those two memory words at >0000 and >0002, what powers we could command! (CE. You can now do this with the Gram Cracker device available from Millers Graphics.) Luckily, there is a very powerful and very useful interrupt which has a context vector that resides in RAM; it is called the "Load' interrupt. Load has the second highest priority, topped only by reset. Like Reset, Load interrupts on Level 0, but the system can differentiate between the two and Load uses memory words >FFFC and >FFFE as its context vector.

Load and Reset share the quality of being the only two interrupts that are non-maskable. The processor will always jump to attention whenever Load or Reset tickles its interrupt control lines and the interrupt mask is powerless to stop them.

What this means to you is that for $2 you can have supreme control over your computer, being able to snatch control away from whatever program is being executed, be it the BASIC interpreter or a Command Module. Let me give you a practical example of this.

Suppose you are writing/debugging a program and for some reason it keeps "locking up" the console. In the past, you had to turn off the main console switch and scratch your head--the "lock-up" can be most difficult bug to find. Imagine instead of resorting to the on/off switch you pressed a switch that generated the Load interrupt and brought up the debugger. Now you can check around in memory to see what happened right when your console was locked up.

Another example will show that even those of you who don't program could make good use of a Load generator. You could press the Load button and get a printout of the monitor screen even if you were using a Command Module that lacked a print capability! That could come in very handy. What does it take to achieve this miraculous power? Assuming you have the memory expansion, which you need to give you memory addresses >FFFC and >FFFE, all you need is a connector to fit your I/O port (which is the one your peripherals attach to), a switch and a soldering iron.

I call my $2 Load generator the Gronos GROMbuster. It is simply a normal open switch soldered to pins 13 and 21 of a connector that mates to the I/O port connector. If you can find a 44 pin connector for the I/O port, it will probably cost you at least $5. I couldn't get one at my local electronics surplus store, so I bought a 32 pin connector and hack sawed one of the ends off to make it fit onto the I/O port. I used a Cherry microswitch that straddled pins 13 and 21 with just a little bending, soldered it into place and then moulded some epoxy around it to add a little strength. Looking straight on at your I/O port, pin 1 is on bottom left, pin 2 is top left, pin 3 is directly right of pin 1, etc. Works Great! (pd. A year ago I assembled a Load interrupt switch where I mounted it on the computer but it could have easily been mounted on the Voice Synthesizer.)

For the combined cost of a 75 cent connector, a 50 cent switch and a few pennies worth of epoxy, I have a device that has saved me a great deal of debugging time and has allowed me to learn many more 99/4 secrets. Oh, I guess I left out one important part; the software. Here is a short example to demonstrate the format:

```
REF VMBW
ST LI Ø,3Ø2
LI 1,TX
LI 2,4
BLWP @VMBW
LIMI 2
JMP $
TX TEXT 'TEST'
WS BSS 32
AORG >FFFC
DATA WS,ST
END
```

To test your GROMbuster, assemble this program and load it into memory, use "Quit" to return to the title screen. Press the GROMbuster switch and "TEST" should appear near the middle of the screen.

If you want to use the debugger with GROMbuster, assemble the following code:

```
INT CLR @>FFFC DISA
LWPI MYWS
CLR RØ
DEC RØ
JNE $-2 WAIT FOR CONTACT
* TO BREAK CLEANLY
* --
* DO YOUR DEBUGGING HERE-INTERRUPTED
* CONTEXT IS IN R13-R15 OF MYWS
* --
STWP RØ
MOV RØ,@>FFFC
RTWP RETURN TO INTERRUPTED PROGRAM
MYWS BSS 32
AORG >FFFC
DATA MYWS,INT
```

Thanks to C.J. Daly for this one. He says it functions very much like breakpoints and you can return to the interrupted program with the 'Q' command. Previous attempts were subject to mechanical switch bounce and thus would not properly store the information needed to return to the original program via the RTWP instruction but this one solves that. The key to this trick is the first instruction (at INT) which clears the workspace pointer word of the Load interrupt vector, thereby causing interrupts after the first one to discard their contexts harmlessly into ROM.

This concludes my explanation of how interrupts are used in general within the 99/4A.

Grapevines are wonderful and thanks to the Ottawa User Group for getting the following from C.A.U.G Alert.

The following information is from the Victoria (B.C., Canada) 99'er Group's August newsletter.

Over two years ago Johan Vanimschoot had spoken with me about a large high quality widget type device for more than three modules. The concept was a six or eight slot expansion box with line drivers and a buffered selector for each module. This was to remove the Navarone Widget's worst features, limited expansion and noisy switching.

This idea has hung around and a couple of months ago he and I were talking about it, and I mentioned that I had read about supposed software existing in the TI for multiple pages of cartridges. This built in software would supposedly work with the proper extra hardware. Lending credence to this information was a Millers Graphics newsletter (The Smart Programmer) refering to their address decoding required. Johan and I had both had a screen with "REVIEW MODULE LIBRARY" come up from time to time during assembly language development. We talked about the hardware problem and devised a simple scheme to test how the software worked (and if it would work at all).

I loaned Johan a wire wrap and a breadboard prototyping system I had built for the cartridge port and with considerable effort Johan managed to locate some 36 pin sockets.

The software does work! This what appears to be possible:

The TI menu comes up as it usually does and an extra selection is added "REVIEW MODULE LIBRARY". If this option is chosen the next available cartridge page is displayed on the menu as if it were the only cartridge plugged in and the option "REVIEW MODULE LIBRARY" is also displayed. This action continues in a loop thru all modules until a selection is made of an application.

Now this is nice, no need to flick switches and up to 16 modules could be available in a monster box. But there is more!

The GPL (Graphics Programming Language) system is designed so that with this hardware the built in software allows one cartridge to access the devices and calls in another. This allows for example, console basic to access all of the plugged in modules call routines and device names at one time. "MINIMEM1", "MINIMEM2", and "SPEECH" and CALL PEEKV, POKEV, LOAD, etc. are all available from basic.

TI Forth can access "MINIMEM1" and "MINIMEM2" and "SPEECH" as devices with no need to switch anything or to modify any console hardware. The operating system in the console handles all accesses transparently.

This was built in from day one with the 99/4 and is on my pre 1983 black and silver console. I don't know for a fact if this on the newer models, but I suspect it is.

The software during the module library selection finds only those pages that contain grom or grom and rom combined. The slots with rom only (third party stuff) do not come up on the menu. Much like the post 1983 consoles.

The Victoria company Osram Industries is currently developing an inexpensive "Super Widget" to take full advantage of this in console software.

Rumours abound on Timeline. Michel.A14C2, of Montreal, Quebec, says a local TI BBS sysop, Tony Hackett, is in the process of adding a 20 megabyte device to his computer. Though Michel was not very specific on this, it does sound like a RAM card and possibly piggybacking his Myarc 512K card.

Another little message popped on my screen when I was on Timeline. It's from Tom Hall of the Edmonton Users Group. The subject relates to the current buzzword, Freeware.

At our monthly group meeting last night, a rather heated discussion arose on the topic of freeware. It seems that a number of our group have trouble with the "free" part -- and the point was made quite strongly that freeware does *NOT* mean that the program is free; it merely means that you are free to try it out before you pay for it, and only your honor and integrity will force you to destroy a program for which you have no use. On the other hand, if you find a program useful, you should in all fairness send some token sum to the author, if for no other reason than to let him know that his work is appreciated. It should be remembered that in many cases the authors of freeware are merely attempting to save themselves some of the costs involved in the more traditional forms of software distribution, i.e., packaging, delivery to vendors, etc., and this is why their programs can be offered to the end user at such a low price. Perhaps it might be a good idea for us to consider that issue here. I'm sure there are a few "professional" software developers who are using this service, and I'm sure just about all users have some opinion on the subject of freeware. I don't know what it's like with other computers, but there is sufficient abuse of the freeware concept within the TI community to cause some developers to withdraw their software from the freeware market, and grumblings of discontent have been heard from still others. If we want to continue to see the quality of software we've grown accustomed to in the past, we're going to have to seriously re-evaluate our obligations to those individuals who are still committed to the TI and are demonstrating that commitment by their continuing development of first-rate software.

The following CALLs were given to us by Teresa Delaney of Dubuque, IA. Thanks Teresa!

Call Load(-31572,P). 0>=P>=255 to vary keyboard response.

Call Load(-31744,P). 0>=P>=15 for continuation of last sound (0=Loud 15=Soft).

Call Load(-31748,P). 0>=P>=255 to change cursor rate and response tone rates.

Call Load(-31788,160). Blank screen must press key to activate.

Call Load(-31794,P). 0>=P>=255 to change time for Call Sound.

Call Load(-31806,0). Normal operation.

Call Peek(-31808,P,Q). P Q - Double random numbers (0 to 255). Needs RANDOMIZE first.

Call Load(-31860,4). Go from xbasic to console basic. NEW afterwards.

Call Load(-31860,8). Auto run of DSK1.LOAD.

Call Load(-31868,0). No "RUN" or "LIST" after a function clear is used.

Call Load(-31873,P). 3>=P>=30 screen column to start at with a PRINT.

Call Load(-31879,P). 0>=P>=255 timer for VDP interrupts every 1/60 sec.

Call Load(-31884,P). 0>=P>=5 change keyboard mode.

Call Load(-31962,32). Return to title screen.

Call Load(-32700,0). Clears screen for an instant.

I hope most of you enjoy reading my column! In case you don't or have suggestions on how to improve this column, please send your letters to the Readers Response column of this newsletter in care of the Newsletter Editor. Next month I hope to get into Clint Pulley's c99 language. I have just bought Jack J. Purdum's C Programming Guide inorder to understand this new language and hopefully pass on some of my newly acquired knowledge of c99 to you the readers so stay tuned!

REVIEWS:

This column presents reviews of materials that may be of interest to the user. The views expressed are the opinions of the reviewers, exclusively.

BOOKS:

Thanks goes to Paul Degner for loaning me "Cracking the 99/4A" (subtitled:'Serious fun for the home computer enthusiast...') for the following review. "Cracking the 99/4A" was written by Brian Prothro and is published by Midnight Express, Box 26941, Austin TX 78755.

Review: by Mike Swiridenko.

Well we rarely get to see a new text for the TI-99/4A, so a review of this work should be of some interest.
Cracking the TI-99/4A is basically a collection of several types of programs prefaced with several brief tutorials of some fundamental programming techniques. The programs range from Games such a checkers and Othello to Assembly Routines to manipulate disk files. The tutorials cover such things as Structured Programming, Linked Lists, User Friendliness, and Logical Operators.
The Programs are written for a full range of systems; Basic sytem to TEII with speech to full systems with Editor/Assembler package. The programs themselves perform quite well, and the Assembly Language routines are welcome although there weren't enough of them to satisfy my own wants. If you order the program disk, as Paul did, you won't have to key in all of the programs in the book. A nice choice for busy people.
The tutorials were informative and centered mainly on the topics of structured programming and user friendliness. The tutorials walk you through the development of a mailing list program and show you how a basic idea can be developed into a more sophisticated working program. Linked lists are covered also, but one drawback to the discussion of this usefull concept was the lack of diagrams that would aid greatly in the understanding of what exactly linked lists are. Finally logical operators are discussed. Logical operators are quite usefull and direct the basic sequence of operation of all programs. It is noteworthy that this topic was included in this book.
Mr. Prothro, in his forward, states, "Wheter you are just learning BASIC, or are well on your way to understanding programming, this book offers you the opportunity to get to know your 99/4A. Often the best solutions to your programming needs are learned by studying how others have solved their own programming problems. For the most part, this book is not a BASIC tutorial, but offers an opportunity to learn while exploring finished programs, as well as add to your software library."
"Cracking the 99/4A" may not be a BASIC tutorial, and it may contribute to one's software library, but I feel that this book would be of more value to the average home computer programmer if Mr. Prothro had presented more than the one program (a mailing list program) in a tutorial manner. A person may learn a lot from going through a working program, but programming is best learned when you are shown in a stepwise manner how it can be done. Not only does a tutorial approach to programming keep interest up it relieves the tedium of keying in a long program by breaking it up with discussion, and in some cases program modifications. The discussion may highlight a particular aspect of the program design, or an interesting programming technique. Modifications may suggest enhancements to the program or a solution to a part of the program not yet implemented. Both contribute to the understanding of how a program works.
In summary "Cracking the 99/4A" is different from most of the 'program' books around, because it gives the reader an brief tour of the design of a mailing list program, as well as a several other programs for the reader to key in.

PROGRAMMING HELP FILE:

The purpose of this column is to present, to the user, techniques that will be useful in the writing of programs for the TI-99/4A home computer. If you can provide some prgramming insight that might be useful to someone, please, feel free to pass it on to me, and I'll get it into the next newsletter.

BASIC/EX-BASIC:

This month I discuss the IF-THEN-ELSE, FOR-TO-NEXT, GOTO, ON-GOTO, and ON-GOSUB statements.
It has been proven mathematically (by Bohm and Jacopini) that all programs may be written using only three statement types. These three types of statements are:

1.- Sequential.
2.- Decision.
3.- Repetitive.

The sequential statements include assignements and other data manipulation statements. These statements are relatively straight forward, and are often performed one after another. The other two statement types determine 'flow of execution' of a program and are what this discussion is about.
Decision statements alter the flow of a program by allowing choices to be made. Choices in a computer system involve yes/no types of answers. The yes/no or TRUE/FALSE answers are the result of logical comparisons which are sybolic of questions such as, "is the value of variable TEST equal to one?", or "has an key been pressed?", and so-forth.
TI Basic and XBasic have several decision type statements for the programmer to use. These statements are the IF-THEN-ELSE, ON-GOTO, and ON-GOSUB statements.
In the IF-THEN-ELSE statement a logical test is placed between the IF and the THEN parts. If the test is TRUE the statements following the THEN will be executed. If the test is FALSE the statements following the ELSE are performed. In TI-Basic only a line number may appear after the THEN and ELSE parts. The line numbers indicate statements to which execution will pass as a result of the test made. XBasic allows multi-line statements as well as nested IF-THEN-ELSE statements. Nested control structures are another topic which I will discuss in another newsletter. The ELSE part of a IF-THEN-ELSE statement may be omitted. In this case a FALSE test result will result in the statement on the line immediately after the IF-THEN statement to be executed. This is called an 'implied else'.
The ON-GOTO and ON-GOSUB statements provide additional decision making power. They are similar in structure to each other, but have a subtle difference in their operation. Both require a value to select one of a choice of line numbers that follow the statement, but the ON-GOSUB statement will jump to a subroutine and return to continue execution from the line following the ON-GOSUB statement, while the ON-GOTO statement will continue exectution from the line number selected.

THE ON-GOTO and ON-GOSUB STATEMENTS LOOK LIKE THIS:

```
100 IF VALU=ZERO THEN 210
[redacted]
110 ON VALU GOTO 130,140,150,140
120 PRINT "VALU is greater than 4"

190 IF VALU=ZERO THEN 210
200 ON VALU GOSUB 1000,1500,1600
205 PRINT "ON-GOSUB returns here."
210 PRINT "end of ON-GOTO/GOSUB example"
```

The value of VALU selects the corresponding line number from the list following the statement. VALU may not be zero in value, and any VALU that is greater than the number of line numbers in the ON-GOTO/GOSUB will result in execution continuing with the next program line.

The FOR-TO-NEXT statement is an example of a repetitive control statement, commonly called a 'loop'. A repetitive structure repeatedly performs some set of statements until a desired condition is met. The condition, like the choice test of an IF-THEN-ELSE, is again a logical comparison. (Logical conditions play an important part in programs.) In the case of the FOR-TO-NEXT loop the comparison is made between the FOR-TO-NEXT index and the limit. Once the index exceeds the value of the limit the condition test evaluates to a TRUE value, the loop ends, and execution continues from the line following the NEXT part.

The FOR-TO-NEXT statement consists of the FOR-TO part, the NEXT part, and some statements that are placed between the FOR-TO part and the NEXT part. The statements are said to form the body of the loop. A typical FOR-TO-NEXT loop looks like the following:

```
100  FOR INDEX=1 TO LIMIT
110  PRINT "INDEX IS NOW ";INDEX
115  PRINT "Other statements execute here."
120  NEXT INDEX
```

Notice that between the FOR and the TO the index of the FOR-TO-NEXT is assigned an initial value, and that the name of the variable used as the index is repeated after the NEXT. The NEXT must match up with the index of the FOR-TO part otherwise the loop will not function. When the loop is first encountered the index is compared to the value of the limit. If the index is greater than the limit the statements within the loop are not executed. If the index is less than the limit the body of the loop will execute. When the NEXT is reached the index is increased by an increment value of one and again compared to the limit. If the index is still less or equal to the limit the statements within the body of the loop will execute again. Thus the body of the loop is executed repeatedly until the index exceeds the value of the limit. When the index becomes greater than the limit the loop ends and the statement immediately following the NEXT part will execute.

You may alter the increment value and/or the way in which the index is compared to the limit by the use of a STEP value. If you specify a negative step value the index must be less than the value of the limit before the loop will end. If you specify a STEP value then that value is added to the index after every repetition of the loop body. The STEP value may be negative or positive in value. If no STEP value is specified then the index is increased automatically by one. A FOR-TO-NEXT with a STEP value specified is shown below.

```
100 FOR INDEX=1 TO LIMIT STEP 2
110 PRINT "The index of this loop (";INDEX;") is increasing by two."
120 NEXT INDEX
```

There are times when you want to leave a FOR-TO-NEXT loop before its index reaches its limit. A good example of this is when you are doing a search for some item in a list. In a situation such as this you would use a IF-THEN statement to test if the desired item was found, ie.- IF FOUND THEN GOTO 200. This introduces the final statement that I will discuss, the GOTO statement.

GOTO statements are usefull in certain places in a Basic language program, but misuse of this statement can result in a program that is very hard to understand making changes to that program extremely difficult. A GOTO merely alters the 'flow of execution' by causing the program to execute from the line number specified in the GOTO. One place where you would want to use the GOTO is in statements that are part of a THEN or ELSE of an IF-THEN-ELSE statement. An example of an IF-THEN-ELSE using GOTOs is shown below.

```
100 IF TEST THEN 150
110 PRINT "This is the ELSE part of the IF TEST statement."
120 PRINT "The program will get here if TEST is FALSE in value."
130 PRINT "We now GOTO a statement past the THEN part of the IF TEST statement."
140 GOTO 170
150 PRINT "The program gets here if the TEST is a TRUE value."
160 PRINT "The ELSE part of the IF TEST statement is passed over, and is not executed, in this case."
170 PRINT "The program continues from here after executing a THEN or ELSE part of the IF TEST
statement."
180 PRINT "This is the end of this example."
```

Another use for the GOTO is in loops other than the FOR-TO-NEXT type. Examples of those loops are:

```
100  GOTO 100  ! an infinite loop. No way to execute other statements after this one is encountered.
200  CNT=0
210  IF CNT>100 THEN 250
220  CNT=CNT+1
230  PRINT "Count is now ";CNT
240  GOTO 210
250  PRINT "Out of Count loop."

90   OPEN #1:"CS1.DATAFILE",INPUT,INTERNAL,VARIABLE 32
100  IF EOF(1) THEN 150
110  INPUT #1:A$
120  PRINT A$
130  PRINT "This is a record from a file."
140  GOTO 100
150  PRINT "All records in the file have been read."
160  CLOSE #1
```

That concludes my discussion of the IF-THEN-ELSE, FOR-TO-NEXT, GOTO, ON-GOTO, and ON-GOSUB statements.

5. RUN PROGRAM FILE

by Don Cook

Once a program has been perfected in object code using the 3. LOAD AND RUN option , it usually is advantageous to convert it to machine code using the SAVE utility program on your E/A disk . A program file which is executed through the 5. RUN PROGRAM FILE option can take less storage space on the disk and load faster . To gain these features , the constraints listed below should be observed .

1. The program must be contained in a continuous memory space in high or low memory expansion . e.g. Part of your program can't be at memory >3000 to >3400 and the rest at >B000 to >C000 .

2. If your program uses low memory expansion , it can only use locations >2F00 to >3F00 (4K) since the utility routines , SAVE program and REF table use the remaining low memory . There should be no restriction on using high memory expansion from >A000 to >FFF0 .

3. Try to avoid reserving memory expansion space using BSS and insted use EQU . IF your program is in high memory , use low memory for your workspace or any memory buffers and keep track of what memory you have reserved . e.g. Use MYWS EQU >2F00 Instead of MYWS BSS 32 This example could save 32 bytes of disk storage space .

4. You must DEF the first , load and last locations of your program as SFIRST , SLOAD and SLAST so that the SAVE program can recognize them .

5. The first statement in your program must be an executable statement . (i.e. not DATA or TEXT) For example :

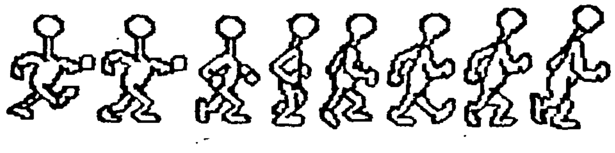
```
       DEF  SFIRST,SLOAD,SLAST
SFIRST
SLOAD  LWPI MYWS
       JMP  START
MYWS   EQU  >2F00        Workspace at >2F00->2F1F
INSTR  TEXT 'PRESS SPACE TO QUIT'
START  LI   R0,2         Screen column 3 , row 1
       SWPB R0
       MOVB R0,@>8C02    LSB of VDP RAM location
       SWPB R0
       MOVB R0,@>8C02    MSB of VDP RAM location
       LI   R1,INSTR
SHOW   MOVB *R1+,@>8C00  Send byte to VDP RAM
       CI   R1,START     Last byte ?
       JNE  SHOW
       CLR  R12          CRU address >0000
TSPACE TB   4            SPACE key pressed ?
       JEQ  TSPACE
       BLWP @0           QUIT
SLAST  END
```

6. To store the program above as a program file , assemble it and load it using the 3. LOAD AND RUN option . Then , clean the dust off your E/A disk part B , load the SAVE utility program and run the program called SAVE . It will prompt you for the name you wish your program file saved as . When it is finished , select the 5. RUN PROGRAM FILE option and test your program file .

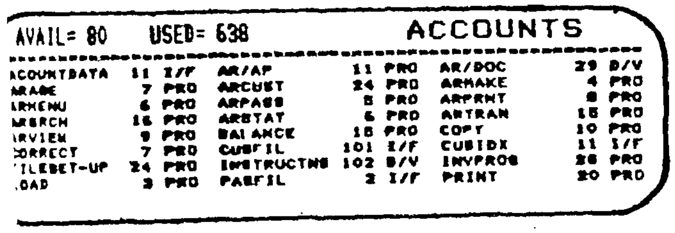
7. The 5. RUN PROGRAM FILE option does not automatically load the utility routines such as VMBW , KSCAN , DSRLNK etc. You must either create your own equivalent routines or load these utilities at the beginning of your program . The machine code for these utility routines is located in the E/A command module GROM starting at location >7000 . A method of loading these routines is listed below .

```
START LI   R2,3          Branches , Routines & REF
      LI   R1,>7000      GROM location
      MOVB R1,@>9C02     MSB of GROM location
      SWPB R1
      MOVB R1,@>9C02     LSB of GROM location
GROM0 LI   R3,4          :
      LI   R4,MYWS       : Put # of bytes in R0 ,
GROM1 MOVB @>9800,*R4+   : memory location in R1
      DEC  R3            :
      JNE  GROM1         :
GROM2 MOVB @>9800,*R1+   GROM value to memory exp.
      DEC  R0
      JNE  GROM2         Last memory location
      DEC  R2
      JNE  GROM0
```

At GROM location >7000, the first two bytes are the number of machine code bytes to load and the next two bytes indicate where to load the code in memory expansion. The first group of code is the subroutine branch locations; the second group is the subroutines code and the last group is the REF table identification. END



```
100 ! ********************
110 ! *    DISKLABEL       *
120 ! *  by ROBERT NEAL    *
130 ! *  T.I. USERS OF     *
140 ! *   WILL COUNTY      *
150 ! * ADAPTED TO EPSON   *
160 ! * &ROLAND PRINTERS   *
170 ! *  by TOM ARNOLD     *
180 ! * CHANNEL 99 USERS   *
190 ! ********************
200 DIM PN$(127),SZ$(127),PT
$(127)
210 TYPE$(1)="D/F" :: TYPE$(
2)="D/V" :: TYPE$(3)="I/F" :
: TYPE$(4)="I/V" :: TYPE
$(5)="PRO"
220 OPEN #1:"PIO"
230 PRINT #1:CHR$(27)&CHR$(6
5)&CHR$(6);:: !*** SETS LINE
 FEED TO 6/72 INCH ***
240 PRINT #1:CHR$(15);!*** P
UTS PRINTER INTO CONDENSED P
RINT
250 DISPLAY AT(2,1)ERASE ALL
:"          DISKLABEL":"
    =========": :"
  by Bob Neal":"         Vers
ion 2.0": :RPT$("c",28)
260 DISPLAY AT(9,1):"Avail=2
91 Used= 67 DISKNAME":RPT$("
=",28):"DLABEL 20 PRO DL
ABEL 27 PRO":"LOAD   15 PRO
LDATA  25 D/F":RPT$("c",28)
270 DISPLAY AT(20,1)BEEP:"Pl
ace Disk To Be Labeled in Dr
ive #1 Then Press Any Ke
y" :: ST=1
280 CALL KEY(0,K,ST):: IF ST
=0 THEN 280
290 OPEN #2:"DSK1.",INPUT ,R
ELATIVE,INTERNAL
300 FOR X=1 TO CNT
310 PN$(X)="" :: SZ$(X)="" :
: PT$(X)=""
320 NEXT X
330 CNT=0
340 INPUT #2:A$,J,J,K
350 IMAGE " ###### ####  ###
## ####               ########
##"
360 PRINT #1,USING 350:"AVAI
L=",STR$(K),"USED=",STR$(J-K
),CHR$(14)&A$
```



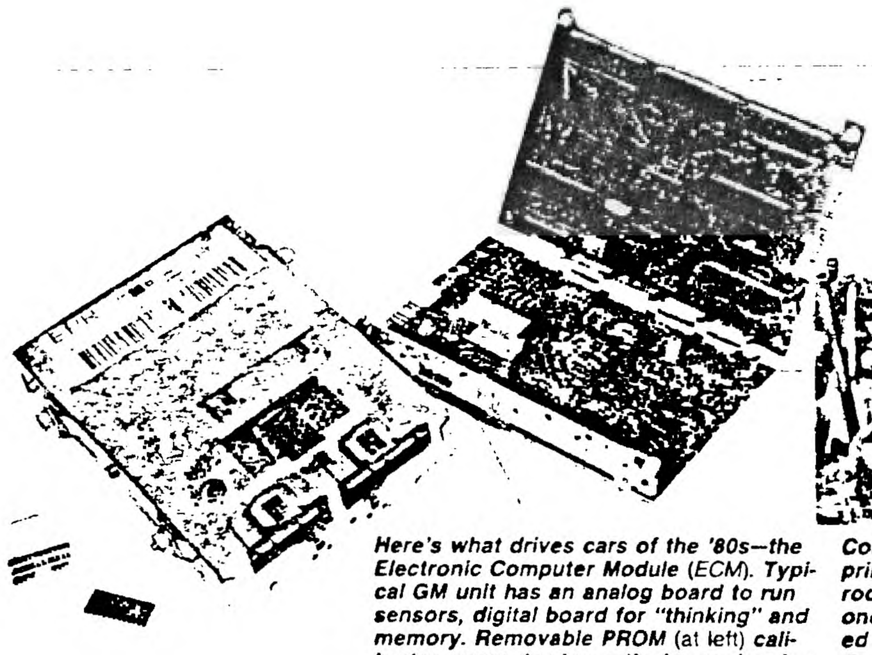
```
AVAIL= 80    USED= 638                   ACCOUNTS
----------------------------------------------------------------------
ACOUNTDATA  11 I/F  AR/AP         11 PRO  AR/DOC     29 D/V
ARAGE        7 PRO  ARCUST        24 PRO  ARMAKE      4 PRO
ARMENU       6 PRO  ARPASS         5 PRO  ARPRNT      8 PRO
ARSRCH      16 PRO  ARSTAT         6 PRO  ARTRAN     15 PRO
ARVIEW       9 PRO  BALANCE       16 PRO  COPY       10 PRO
CORRECT      7 PRO  CUSFIL       101 I/F  CUSIDX     11 I/F
FILESET-UP  24 PRO  INSTRUCTNS   102 D/V  INVPROG    26 PRO
LOAD         3 PRO  PARFIL         2 I/F  PRINT      20 PRO
```

```
370 PRINT #1:CHR$(27)&CHR$(8
3)&CHR$(1);:: PRINT #1:RPT$(
"=",58)
380 LC=2
390 FOR X=1 TO 127
400 INPUT #2:A1$,A,J,K
410 IF LEN(A1$)=0 THEN 460
420 PN$(X)=A1$ :: SZ$(X)=STR
$(J):: SZ$(X)=RPT$(" ",3-LEN
(SZ$(X)))&SZ$(X)
430 A=ABS(A):: PT$(X)=TYPE$(
A):: IF A=4 AND K=254 THEN P
T$(X)=TYPE$(5)
440 CNT=CNT+1
450 NEXT X
460 CLOSE #2
470 FOR X=1 TO CNT STEP 3
480 IMAGE ########## ### ###
  ########## ### ###  ######
#### ### ###
490 IF LC>9 THEN 500 ELSE 53
0
500 PRINT #1:"":""
510 IMAGE "
                      ########
##"
520 LC=2 :: PRINT #1,USING 5
10:CHR$(14)&A$ :: PRINT #1:C
HR$(27)&CHR$(83)&CHR$(1)
;:: PRINT #1:RPT$("=",58)
530 PRINT #1:CHR$(27)&CHR$(8
3)&CHR$(1);:: !*** PUTS PRIN
TER IN SUBSCRIPT MODE **
*
540 PRINT #1,USING 480:PN$(X
),SZ$(X),PT$(X),PN$(X+1),SZ$
(X+1),PT$(X+1),PN$(X+2),
SZ$(X+2),PT$(X+2):: LC=LC+1
550 NEXT X
560 FOR X=1 TO 11-LC :: PRIN
T #1:"" :: NEXT X :: PRINT #
1:CHR$(27)&CHR$(84)! ***
LAST PART RESETS SUBSCRIPT,
MAY NOT BE NEEDED ON GEMINI
570 DISPLAY AT(20,1)BEEP:"CA
TALOG ANOTHER? (Y/N)":"":""
580 CALL KEY(3,K,S):: IF S=0
 THEN 580
590 IF CHR$(K)="Y" THEN 270
ELSE IF CHR$(K)="N" THEN 600
 ELSE 570
600 CLOSE #1 :: END
```

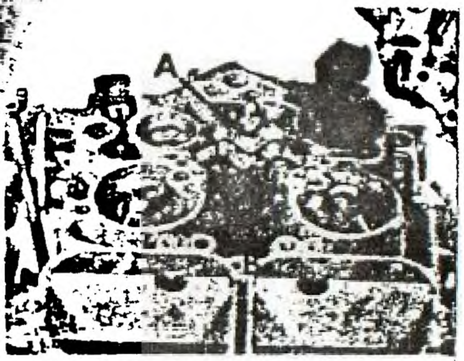
FORTH:

```
SCR #36
  ( Speech words for TI FORTH )  BASE->R  HEX  : Z ;  : SPSTART ;
  ( Adapted from Wycove FORTH by Clint Pulley)
  ( SPEECH? RETURNS TRUE IF SPEECH SYNTHESIZER IS ATTECHED )
  ( ADDRESS SAY speaks the word or phase with address given by
                Editor/Assembler Manual or LIST-VOCAB)
  ( >HELLO and >IDENT speak appropriate things)
  ( LIST-VOCAB lists the resident vocab with DECIMAL addresses)
  : SPWR ( n ---) 9400 C! ;
  CODE SPRDC 0460 , 8320 , ( branch to read code on 16-bit bus)
  : SPRD ( --- n )  [ HERE 18 + ] LITERAL 8320 0E CMOVE
    SPRDC 8349 C@ ; ( Read from speech ROM)
  ( Speech Read routine, moved to >830)
  D820 , 9000 , 8349 , ( MOVB @SPCHRD,@>8349 )
  1000 , 1000 , 1000 , ( NOP's for delay )
  045F ,               ( NEXT  )
  R->BASE    -->

SCR #37
  ( Speech Words Screen 2)  BASE->R  HEX
  : SPAD ( addr --- ) 5 0 DO 10 /MOD SWAP 40 OR SPWR LOOP
    DROP 1 0 DO LOOP ;  ( Load speech address)
  : RDSP ( addr --- n ) SPAD 10 SPWR SPRD ; ( Read speech data)
  : SPEECH? ( --- flag ) 0 RDSP 0AA = ; ( Test speech attached )
  : SAY ( adr --- ) SPAD 50 SPWR ; ( using resident vocabulary )
  : >HELLO 351A SAY ;
  : >IDENT 2D19 3793 6551 3A32 6DDE SAY SAY SAY SAY SAY  ;
  : RDSPW ( addr --- word value ) DUP 1+ RDSP SWAP RDSP 100 * + ;
  : LIST-TREE -DUP IF DUP RDSP OVER + 1+ DUP RDSPW MYSELF OR
    DUP 4 + DUP 1+ RDSPW SWAP RDSP SWAP 7 D.R SPACE SWAP OVER OVER
    - 1 DO 1+ DUP RDSP EMIT LOOP DROP PAUSE IF ABORT ENDIF
    2+ RDSPW MYSELF ENDIF ;
  : LIST-VOCAB 1 LIST-TREE ;
  ( PAUSE by pressing  any key, ABORT by pressing CLEAR )
  >HELLO >IDENT    R->BASE
```

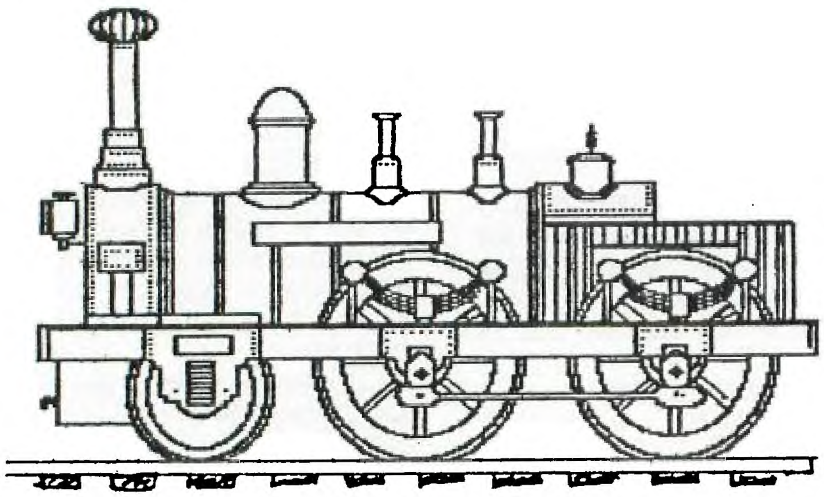


Here's what drives cars of the '80s—the Electronic Computer Module (ECM). Typical GM unit has an analog board to run sensors, digital board for "thinking" and memory. Removable PROM (at left) calibrates computer to particular car/engine.



Computer-controlled Quadrajet regulate primary fuel flow by pulsing metering rod bar (A) up and down 10 times a second with a solenoid, pushing spring-loaded metering rods (B) down into jets. Rest of carb is mechanical.

```
100 !  LOAD/TRICK PROGRAM
110 !  "FREEWARE" BY BARRY A. TRAVER, 835 GREEN VALLEY DRIVE, PHILADELPHIA, PA 1
9128 (PHONE: 215/483-1379)
120 ON BREAK NEXT :: CALL CLEAR
130 PRINT "* READY *": : :: M$="" :: CALL HCHAR(24,2,62):: R=24 :: C=3 :: CTR=0
140 CALL KEY(0,K,S):: CTR=CTR+1 :: IF K=8 THEN PRINT "I DON'T DO BACKSPACES.": :
 : :: GOTO 130
150 IF S<>0 THEN CALL HCHAR(R,C,K):: C=C+1 :: M$=M$&CHR$(K)
160 IF M$="OLD" THEN PRINT "OLD?": :"WHY OLD?  WHY NOT NEW? ": : : :: GOTO 130
170 IF M$='NEW" THEN PRINT "NEW?": :"NOW THAT'S MORE LIKE IT!": : : :: GOTO 130
180 IF M$="RUN" THEN PRINT "RUN?": :"WHY RUN?  WHY NOT WALK?": : : :: GOTO 130
190 IF M$="CALL" THEN PRINT "CALL? ": :"NOT THAT STUPID CALL FILES   ROUTINE AGA
IN!": : : : :: GOTO 130
200 IF LEN(M$)>4 THEN PRINT :"YOU'RE TOO ROUGH--I'M GOING   TO LOCK UP ON YOU!"
:: GOTO 240
210 IF CTR=2 THEN CALL HCHAR(R,C,30)
220 IF CTR=4 THEN CALL HCHAR(R,C,32):: CTR=0
230 GOTO 140
240 GOTO 240
```



1827 STEPHENSON STEAM LOCOMOTIVE

This was one of the first steam locomotives to operate in the U. S.

Picture drawn by Malcolm Johnson on the GRAPHX program.

99- SUBSCRIPTION -99

CLUBLINE-99
P.O. BOX 1005, STATION A
HAMILTON, ONTARIO
CANADA. L8N 3R1

Yes I would like to subscribe to CLUBLINE-99

Subscription Rates are high as the postage cost with packaging is more than the production cost of the Magazine.

(Please Print Clearly)

NAME ______________________________

POSTAL ADDRESS ______________________________

____

____

SUBSCRIPTION RATE "INDIVIDUAL"

	MAGAZINE ONLY	MAGAZINE WITH MONTHLY DISK
CANADA	$30 for 12 issues	$120.00
U.S.A.	$25 U.S. Money	$95.00 U.S. Money
OVERSEAS	$35 Canadian Money	$130.00 Canadian Money

This magazine could be available from your club for $2.00 Canadian per issue. The monthly disk should also be available from your club at $6.00 Canadian or less.

VISA

Bill my ☐ VISA ☐ MasterCard

☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐
Account No

Phone No. ______________

Signature ______________ Exp Date ______

☐ Check or Money Order Enclosed

MasterCard

```
FILE:  SWAP/SELL
DATE:  3/6/86
TITLE: SWAP/SELL

 INDEX
 0 = PAGE #
 1 = ITEM
 2 = TYPE
 3 = DSCRPTION
 4 = PRICE
 5 = PHONE
```

0	1	2	3	4	5
1	BEST SOFTWARE	CASSETTE	SOFTWARE	$10.00	586-6889
2	CAR WARS	MODULE	SOFTWARE	$20.00	668-1781
3	DOW-4 GAZELLE	CASSETTE	SOFTWARE	$20.00	[illegible]-6889
4	EGYPT GRPH ADV	CASSETTE	SOFTWARE	$10.00	[illegible]-6889
5	HUNT THE WUMPUS	MODULE	SOFTWARE	$20.00	668-1781
6	MDVL GRPH ADV	CASSETTE	SOFTWARE	$10.00	586-6889
7	PUBLICATIONS	MAGAZINES	BACK ISSUES ECH	$1.00	586-6889
8	SIGNALMAN MARK3	MODEM	HARDWARE	$100.0	586-6889
9	SUNDIAL ISL 1&2	CASSETTE	SOFTWARE	$10.00	586-6889
10	TEACH SELF XBAS	CASSETTE	SOFTWARE	$15.00	668-4804
11	TEACH SELF XBAS	CASSETTE	SOFTWARE	$18.00	888-1346
12	TEACH SELF XBAS	CASSETTE	SOFTWARE	$20.00	668-1781
13	TEACHSELF BASIC	CASSETTE	SOFTWARE	$17.00	668-4804
14	TEACHSELF BASIC	CASSETTE	SOFTWARE	$20.00	632-4967
15	TI FORTH PKG	DISK	SOFTWARE	$40.00	895-7067
16	TREASURE HUNT	DISK	SOFTWARE	$10.00	586-6889
17	WIZARD&PRINCESS	DISK	SOFTWARE	$30.00	586-6889
18	LAST WORD ON TI	BOOK	INFORMATION	$10.00	895-7067
19	DUAL CASSETTE	CABLES	HARDWARE	$18.00	668-4804
20	TI-99/4A	COMPUTER	HARDWARE	$60.00	669-4804
21	MINI-MEMORY	MODULE	SOFTWARE	$30.00	668-1781
22	TI LOGO II	MODULE	SOFTWARE	$90.00	489-3127
23	ADVENTURE	MODULE	SOFTWARE	$55.00	489-3172
24	TI SYSTEM	UNIT	HARDWARE	$775.	489-3172
25	TI-SYSTEM	COMPLETE	H/W+S/W	$960.	255-8915

DISK LABELS

by Tom Arnold

Every once in a while I come across a program that really impresses me. Disk Label by Robert Neal is one of those. It is one of those rather simple programs that performs like magic and is really very useful. This program catalogs your disks, which isn't so special, however it prints out the catalog on a 3 1/2" x 7/8" label! Now this is really handy. Simply place the disk in the disk drive and press a key. The label is printed out, ready to stick on your disk. This makes updating your disks really easy. The print is in compressed subscript, which isn't the easiest to read but neccessary to get long listings on the label. I want to thank Bob for this most useful program and bet that you, the reader will find this one of the most used programs in your library.

NEWSLETTER EDITER
WINNIPEG 99/4 USERS GROUP
P.O.B. 1715
WINNIPEG, MANITOBA
CANADA, R3C 2Z6

TIERS EDMONTON
P.O. Box 11953
EDMONTON, ALBERTA
T5J 3L1