EDMONTON
99'er
COMPUTER
USERS'
SOCIETY

99'er Online

DECEMBER 1984

P.O.Box 11983
Edmonton, Alberta
Canada T5J 3L1

TO:

99'er ON LINE is the news letter of the Edmonton 99'er Computer User's Society published ten times a year. All material contained in this news letter may be published in other news letters provided that source and author are identified unless otherwise stated. We welcome correspondence from all TI User Groups and will extend source credit courtesy in **99'er ON LINE.**

CORRESPONDENCE: News letter editor: BOB PASS, 59 LABELLE CR, ST. ALBERT, ALBERTA, CANADA T8N-[illegible]. All other correspondence: EDMONTON 99'er COMPUTER USER'S SOCIETY, P/O BOX 11983, EDMONTON, ALBERTA, CANADA T5J-3L1

OFFICERS: PRESIDENT--BILL CANNON, VICE PRES--PAUL HELWIG, TREASURER--EVAN SMITH, SECRETARY--SUSAN LIVINGSTON

REGULAR MEETINGS: Regular meetings of the Edmonton User's Group are held on the second Tuesday of each month on the 3'th floor of the General Services building of the University of Alberta from 7:00 till 10:00 PM and are open to all members in good standing. Non-members may attend their first meeting free of charge. The Executive Committee meets monthly. Members may attend these meetings as observers or to address a particular issue. Arrange with one of the officers listed above if you wish to attend.

ADVERTISING: Commercial advertising space is available in this news letter at the following rates: FULL PAGE--$20.00, HALF PAGE--$15.00, 1/4 PAGE--$10.00. Discuss your commercial needs with Paul Helwig at the next meeting or write to the P/O Box above. Members may advertise their personal computer related items for free but are asked to limit their ads to about 20 words. Mail your ads to the EDITOR'S ADDRESS or hand it to him at the general meeting; newsletter deadline 15'th of the month.

MEMBERSHIP FEES: FAMILLY---12 MONTHS, $20.00, 6 MONTHS, $15.00. STUDENTS---12MONTHS,$15.00, 6 MONTHS, $10.00.

THE EDMONTON 99'er COMPUTER USERS' SOCIETY

Date / /

New Membership Application	☐
Renewal of Membership	☐
Change of Address	☐

NAME ____
last first initial

ADDRESS ____

CITY ____________________ PROVINCE or STATE ____________________

POSTAL or ZIP CODE ______________ TELEPHONE ____________________

Single or Family membership	$20	(12 months)	☐
	$15	(6 months)	☐
Student membership (must have student I.D. number)	$15	(12 months)	☐
	$10	(6 months)	☐
Out of Town (newsletter only)	$20	(12 months)	☐
	$15	(6 months)	☐

* Please note that all membership money should be in CANADIAN FUNDS.

* May we release your name, address and telephone number on a club roster, to companies and other clubs?

YES ☐ NO ☐

FOR OFFICE USE ONLY

Date Filed / / Membership Number ____________________

Amount Tendered $ __________ Authorized Signature ____________________

GUEST SPEAKER

by: Bob Pass

At our November meeting, we had the pleasure of welcoming Mr. Martin Kratz, a graduate law student who is presently articling before the Alberta Bar. Martin's specialty in law is focused on computer security, copyrights, and licensing of software. At this time, I would like to thank Martin on behalf of our group for taking the time on a nasty evening to speak to us. Judging from the questions from the floor, it is obvious that all present appreciated Martin's presentation. Martin expressed intrest in speaking again to us next spring as he could not adequately cover the entire agenda he had prepared nor everyone's questions. We would certainly like to hear more about this very interesting field. Again, thank you Martin for an excellent evening.

During Martin's presentation, I jotted down some of the more interesting points which I would like to pass along to those who could not get to the meeting. The first part of his topic was on License Agreements. These are usually found inside the manuals that accompany your software that severly limit your rights in the use of the product. There is often a notification that if you opened the package, it is presumed that you have read the License Agreement and agree to all terms by the act of your opening the package. Quite often this is INSIDE the sealed package, but that doesn't seem to matter! Briefly, Martin explained that these agreements mean that you have purchased only the right to use the material; you own nothing, not even the disk or manuals! Some companies require you to register the number of the machine you will be using the product on. Just because you purchase one copy, you cannot make copies to use on your other systems. Most companies allow you to make a back-up copy for your own use while others insist that you purchase a back-up from them at a nominal fee. If you make subsequent modifications to the software to do a particular task, you are still bound by the agreement in all ways. Should you decide that the product is really not the best thing since sliced bread, you may in most cases (read your agreement) sell the product to someone else provided that you sell EVERYTHING including the original documentation and diskettes plus all of your back-ups and modifications. Again, according to the agreement, if you are not happy with the product you probably cannot get recourse from the manufacturer or seller unless you can prove that the product was miss-represented. The manufacturer or dealer is not responsible if your software causes you financial loss or legal liability through your application of the product.

Mr. Kratz next spoke on Canadian copyright law which came into being in 1923 and has not been changed since. Consequently, there is much confusion as to how this ancient code is to be applied to computer software, especially since our laws are different from the US code which we hear more of due to the sensationalism of some of the trials and lawsuits south of the border. Recent cases in Canada have resolved that software is protected under our code. Moreover, Martin explained that there is an agreement between Canada and the USA to honor each other's copyrights. In Canada you do not have to copyright your material to be protected. In fact, if your product contains trade secrets, you should NOT copyright as you will be laying bare your secrets to anyone nosey enough to look. You can cover your interests in Canada by establishing the fact that you did it first. This can be done by providing a copy of your work to a lawyer, notary, a software escrot house (for a small fee), or some other professional who can swear to the contents and date you wrote the material. Should you publish the material in the "public domain" (such as newspapers, etc), you still retain your full rights. Should you develop an improved or enhanced application based on someone else's idea, your changes must be SIGNIFICANT before you could gain full rights to the material. You would be most advised to enter into an agreement with the original author and cut him in for a fair share.

If you feel that the product you purchased is not working properly, establish communications immediatly with the vendor or manufacturer. Retain all sales slips and correspond as much as possible in writing so that dates and times are established. It is in your interest to document everything possible or at least have creditable witnesses to verbal exchanges.

Lastly, Martin spoke at length on PIRACY and it's consequences. He explained that there is an attitude among computer users that it is OK to swap programs, especially the more costly ones as the manufacturers are obviously making a fast buck and besides who will find out any how! Well that may be so in some cases but there are an awfull lot of software writers and companies that are bankrupt or soon will be because pirates have effectively destroyed the market. This forces successfull companies to charge top dollars initially because they know that they will have about six months to recover their investment before the market has walked the plank. So, on one hand we have authors who can't afford to turn out innovative products. On the other hand, sooner or later an author (or company) will have had enough and will lash out at the first pirate he can find. If that happens to be some poor schmuck who picked up something neat from a friend's brother-in-law in Kippers Flippers Newfoundland, well that's too bad! This guy has had it right-up-to-here and he wants to nail someone and get enough publicity to discourage others. At the very least, he could be had for copyright infringement. If the product is worth more than $200, the hapless thief could be contemplating his deeds from a cell: that could be the result of a trial occuring in Ontario right now. A chilling thought, isn't it?

nova·COMPUTERWARE
52 AIRPORT ROAD
EDMONTON, ALBERTA T5G 0W7

(403) 452-0372

TI-99/4A

SOFTWARE & HARDWARE

THIRD PARTY PRODUCTS

Extended Basic	$139.95
Widget	$49.95
Super Sketch	$99.99
TI-Writer	$139.95

Sanyo Electonic Typewriter
AE-335 (TI-Compatible) $899.99

Parallax Interface $149.95

10% Discount
on Educational Software

OPEN 1:00 - 5:00
TUESDAY - SATURDAY

NEXT MEETING AND EXECUTIVE NOTES

The next regular meeting will be on Tuesday, December 11'th at 7:00 PM in room 349 in the U of A General Services building on 116'th Street. The executive meeting will be on Tuesday, December 18'th at the same time and place.

This month, we feature a cassette tape deck cleaning and maintenance workshop presented by our own Bob Burley. He will have several different units to demo and he will sell them at a very reasonable cost.

Those of you who did not pick up their membership card at the last meeting will recieve it in the mail along with this news letter. Don't throw away that envelope! Look inside!

Executive members and assistants are asked to prepare a resume of their duties and bring it along to the next exec meeting. Identify areas where you need help.

CONTEST We need a letterhead for official Society correspondance. Printed stationery costs too much and besides, why deprive our members of a chance at fame and noteriety? Your letterhead must be capable of being printed on a dot matrix printer. You may choose any of the TI languages and you can use the top two inches of the page. Our president has promised a prize for the most acceptable entry. He won't tell us what it is but last time it was a sack of printer ribbon dust!

Our one and only member in Saskatoon, Francis Gaston, wrote to say that he and 12 fellow Saskatoonies have initiated a TI user's group. He included a copy of their newsletter and it looks like they are off to a fine start. Best of luck to you and keep in touch.

We received a letter from Holt, Rinehart, and Winston of Canada Ltd. They are looking for authors (programs or microcomputer oriented pieces) who lie undiscovered in user groups across Canada. If you have something that you believe is worthy or an idea that could be developed if the price was write (pun), send a query letter to:

Ian Chadwick,
Software Editor,
Professional and Trade Division,
Holt, Rinehart and Winston Ltd.,
55 Horner Ave.,
Toronto, Ont.,
M8Z-4X6
(416) 255-4491

FOR SALE

SOURCERER & WITNESS $20 EACH.
ORIGINAL DOCUMENTATION AND MEDIA.
CALL SCOTT AT 928-3874 AFTER 6 PM.

RETURN TO PIRATE'S ISLE $20
SUPER SKETCH GRAPHICS TABLET FOR TI $80
VIDEO CHESS MODULE $40
CALL PAUL 432-0613 EVENINGS

LITTLE REM

by: Tom Hall & Michal Jaegermann

Picture this: you've found a program on a bulletin board that you've been looking for for a long time, and you've just finished downloading it to your system when you realize that you **STILL** have to type the whole thing into the machine before it will run. You think, "Gee, I wish there was a way to just convert this file into a program without having to type it all in!"

Well, now there is! The following program comes from the newsletter of the Lehigh 99'er Computer Group in Allentown, Pennsylvania and, with an improvement added by Michal Jaegermann, we present this little gem to you.

The beauty of this little program is its approach to the problem: instead of worrying about tokenizing the file, it simply writes each line to a display variable 163 file, and inserts a "!" at the beginning of each line so that the rest of that line is treated as a REMark. After you run the program, you will have to merge in the newly-created file and manually remove each "!" (which is a **LOT** easier than having to type in the whole thing!), and the computer will do the work of tokenizing it when you save it back to disk!

Michal has added a couple of words of warning about minor alterations of the file which you should keep in mind when getting ready to use this program. The original program could only handle a maximum of 80 bytes per line; Michal's modification extends that capability to the full length of the merge file record length of 163 bytes (minus 2 bytes for the line number and 1 byte for the CHR$(131)), but in order to get the merged file formatted correctly, you must pay particular attention to columns 78-80 of each line. If there's no text in this area, the you can forget about that line. If a word in a line ends **EXACTLY** at column 79 or 80, but is not the last word associated with that line number, then you should insert a space at the very beginning of the following line. If the last word of a line number ends in either column 79 or 80, then you should insert a blank line as the very next line of the file. This will ensure that the following line is not simply tacked on to the end of the preceding one. The program will ignore the blank line in outputting to the merge file. The program will also ignore a line which does not begin with a valid number and is not a continuation of the previous line, so if you want to, you can liberally sprinkle comment lines throughout the file, and this program will skip them as it processes the file. Unfortunately, if your program file has very long lines, you will not be able to translate the complete line, since the fact that the line is treated as a REM means that it is not tokenized; therfore each character that you can see is a byte used, and a program line in a merge file can be no more than 163 bytes, including line number.

```
   1  DISPLAY  AT(5,4)ERASE ALL:"INPUT FILE": :TAB(10);"DSK1.":
:TAB(4);"OUTPUT FILE ": :TAB(10);"DSK1."

   2      ACCEPT      AT(7,13)SIZE(-12):I$      ::      ACCEPT
AT(11,13)SIZE(-12):O$   ::    OPEN   #1:"DSK"I$   ::   OPEN
#2:"DSK"O$,VARIABLE 163

   3 LINPUT #1:L$ :: IF LEN(L$)>78 THEN LINPUT #1:M$ :: L$=L$M$

   4 S=POS(L$," ",1):: ON ERROR 7 ::  N=VAL(SEG$(L$,1,S)):: ON
ERROR 6 :: A=INT(N/256):: A$=CHR$(N-A6):: PRINT : :L$

   5 PRINT #2:CHR$(A);A$;CHR$(131);SEG$(L$,S+1,139-S);CHR$(0)::
GOTO 3

   6 PRINT #2:CHR$(255);CHR$(255):: CLOSE #2 :: END

   7 ON ERROR 6 :: RETURN 3
```

EDITOR'S CORNER

Your not so trusty Editor (me) slipped in a couple of errors last month to see who was awake! Michal Jaegermann obviously was as he picked up the following:

In column 2 the 5'th line in screen 3 of the FORTH program reads in part, " 4 10 A." whereas it should be " 4 10 AT .", exactly the same format as the " 3 E AT ." in the line above. I believe this would make some difference in how the prgram runs!

Also, in the X-Basic program listing in column 8, I missed a "GOTO 130" at the end of line 130. Add it immediately after the last two colons in that line. Many thanks Mike.

Last month I tried a new format to get 64 characters into each column of the newsletter to format the FORTH listings as they would appear on screen. This caused some of the right margins to be clipped in reproduction so I've reverted back to the previous format.

My thanks to all of you who have been sending excellent articles for this newsletter. Without you folks, this would be pretty dull reading!

PACMAN

by: Francis Gaston

PACMAN is one of the new software cartridges available from Atarisoft. Although the name may induce a consumer to purchase this module, I would recommend a demo before plunking down the green. In my opinion, I was not impressed with this product and would rate it 7 out of 10.

Primarily, this module succeeds in simulating the Atari 2600 version of PACMAN in terms of speed and capabilities. In the first 6 screens, it may be a little bit slow but what is lacking in speed is compensated for by the cunning of the goblins! The subsequent screens advance in intelligence and speed. At no time, however, does the game approach the arcade version.

Joysticks are required and beware that some non TI sticks will not operate Atari games for some queer reason. (I can't figure why not as the joystick port is conventional TTL logic).

The game has over 19 screens with higher scoring opportunities as you progress. It has the same format as Munchman with energisers, bonus nuggets, and the familiar Blinky, Pinky, Inky, and Clyde. Levels of difficulty can be set at the beginning of the game. A pause can be generated by pressing the space bar (far better than diving for the "P" as in Munchman!) so you can raid the fridge, etc. High score is retained throughout the session.

This module was obtained from the USA for $33.00 CDN (taxes, duties, and exchange included); local retail is $59.95. Although I would prefer a faster version, it still has merit in that you can still obtain good quality software for our systems.

broadmoor stationers ltd.

- Office Supplies
- Office Furniture
- Typewriter Rentals
- Typewriter Repairs
- Business Machines
  - Typewriters
  - Calculators
  - Cash Registers
- Photo Copy Service
- Rubber Stamps

SALES, SERVICE & LEASING

broadmoor stationers

165 Athabascan Avenue Sherwood Park

Mon. - Fri. 9:00 – 5:30 pm Sat. 10:00 - 4 p.m.

SHARP 464-4343 SCM SMITH-CORONA

BASIC PROGRAMMING

by: Bob Pass

This month's article covers graphics and how to program them in TI Basic. First of all, I will discuss how the computer displays graphic characters on the screen of your monitor or TV then I will describe how you can control this procedure through programming. Finally, there will be a short program that demonstrates the principals of the three articles I have written to date.

Let's start with a description of the "canvas" you have to work with - namely the TV screen. The computer, because of memory constraints, dictates the dimensions of the grapic screen available for your use. In the case of the TI-99/4, the screen has a size of 24 rows by 32 columns or 24x32=768 separately controlable screen elements. The machine has the capability (and you via programming) of controling each of these elements. Each screen element may contain just one character. However, each screen element can be further subdivided into an eight by eight grid of individually controlable points or "pixels" (an acronym for PICture ELement). Thus, you can control upto 768x64=49,152 pixels on the screen. From here on, I will call each of the 768 screen elements a "character" and each of the 64 picture elements that comprise a character will be called a "pixel".

The character is of course defined by the 64 pixels contained within it's boundaries. Further more, each character will have a foreground and background. If a pixel is turned on, it becomes the foreground of the character, a pixel that's turned off will be in the character's background. You can assign any one of 16 colors to both the foreground and background pixels of a character and you can control which pixels are on or off. Additionally, the entire screen can be assigned one of the sixteen colors. All of this is done with just three Basic commands: CALL SCREEN, CALL COLOR, and CALL CHAR.

At this point, you should read pages II-73, II-75, & II-76 of the User's reference manual.

To summarise the above, you have 768 character positions on the screen arranged in 24 rows of 32 characters with each character further divided into an 8 by 8 matrix of pixels. Thus, you have a screen comprised of 256x192=48,052 separately controlable dots or pixels. How does your 99/4(A) administer all of this screen image yet still be able to run your programs as fast as it does? Well, the secret is that your machine actually has two processors in it! Most of you are aware that there is a TMS9900 processor chip inside your cosole that runs all programs. Also there is a TMS9918A Video Display Processor (VDP) chip in there which is responsible for the screen image among other things. When something is to be displayed on the screen, the CPU (the 9900 chip) writes the data into a memory area called VDP RAM in tables called the Screen Image Table and the Character Description Table. The VDP chip scans through these tables continuously and uses the information to modulate the signal to the monitor or TV thus producing the image on the screen. The following paragraphs discribe the Screen Table and the Character Description Table and how you can control them.

The Screen Image Table starts at hex >0000 and contains 768 bytes of 8 bits each. Each byte corresponds to a screen position and contains the ASCII code number of the character to be displayed at that position. Addresses >0000 to >001F contain the character codes for the first row, addresses >0020 to >003F contain the codes for the second row, etc. Since each byte consists of 8 bits and must contain a complete ASCII code, there are only 256 unique characters possible (see last month's article on strings), ie ASCII codes 0 thru 255. From this, you can resolve that the only way to get something onto the screen is to give it a character number and then place that into the Screen Table in the place that corresponds to where it is to

appear on the screen. This is done with the basic commaands CALL HHAR or CALL VCHAR. Note that these commands specify all the parameters identified above; a row, a column, and a character number. The CPU chip can mathematically convert the row and column numbers into the VDP RAM address that will recieve the character code specified. Additionally, there is a fourth parameter in these commands that provide the programmer a short cut; you can specify how many characters are to be displayed starting from the position specified. Now, look at the command CALL GCHAR and see if you can figure out how the computer can retrieve the character code displayed at a screen position. It should be apparent that again the row and column numbers are used to generate the RAM address and then the character code at that address is read instead of written.

Simply having the character code numbers in the screen table is not enough if we are to have full graphics capabilities; we must also be able to control each of the 64 pixels that make up the character. This can be done using the CALL CHAR command which changes the pixel pattern of printable characters. This command works in the following way. In VDP RAM there is a second table called the Character Description Table. It starts at address >0800. Since once again there are 8 bits to each byte in RAM, and each character's description consists of 8 rows of 8 pixels, we will need 8 bytes of ram to describe each of the 256 characters. Thus this table is 2048 bytes long. The first 8 bytes describe character 0, the next 8 describe character 1, etc. When the VDP scans the screen table, it gets the character number to be displayed. This number is then used to index the character description table to get the actual pixels to be displayed. The CALL CHAR command simply updates this table to your specifications. (When you first turn on your system, the CPU auto matically loads this table with the "Standard Character Set" which is located in a ROM chip). Before the VDP writes this pattern to the screen however, it must determine which colors to use. Once again, the character number read from the screen table is used to index into another VDP RAM table which contains the color code numbers for the foreground and background of that character. This table is customised via the CALL COLOR command. The VDP can now use the color information from this table to color the "on" (1) (foreground) and the "off" (0) (background) bits from the character description table to be displayed at the position being read from the screen image table! Simple isn't it?

I am not going to go into any detail about any of the commands that control graphics as the descriptions in your User's Reference Guide are fairly explicit. The CALL CHAR command will likely give the most trouble but I suggest that the best way to learn how to do it is to DO IT! Before long, you will find that it will be second nature to you. By the way, the long string of data you put into the CALL CHAR command is the actual hex code that is written into the character description table in VDP RAM which is about as close as you will get to assembly language programming in TI BASIC. Many other micro's require nearly ALL graphic routines to be written in assembly so next time you are grumbling about all those CHAR, HCHAR, & VCHAR commands, just be thankfull you didn't buy an APPLE!

Now I'll pass along a program that demonstrates many of the items discussed in these articles to date. Before you run it, see if you can figure out what will happen. If you can, you will find that you will have a dandy sub-routine that you can use in some of your programs; the heart of it is in the GOSUB. As an example of neat programing, this is not the best. I have tried to illustrate as much of the past three articles as possible and what may seem clumsy was put there for a reason. For instance, in line 510 the concatenation operator (&) is illustrated to not only show how it is used but also to center the text that will be displayed. Try taking lines 510 & 530 out to see what I mean. When you run this program, keep the names to less than 28 characters else it will "bomb"; try names of different lengths to see how the program handles the information. Next month, if time permits, I'll talk about how to develop some good programming habits that will make things easier for you when your creation won't run.

```
100 REM  ****************************
110 REM  * PROGRAM TO DEMONSTRATE     *
120 REM  * GRAPHICS, STRINGS, AND     *
130 REM  * USE OF NUMERIC VARIABLES.  *
140 REM  ****************************
150 CALL CLEAR
160 PRINT "PUSH DOWN ALPHA LOCK"::
170 REM
180 REM  UPPER LEFT
190 CALL CHAR(96,"FF80BFA0AFA8ABAA")
200 REM
210 REM  UPPER HORIZONTAL
220 CALL CHAR(97,"FF00FF00FF00FF00")
230 REM
240 REM  UPPER RIGHT
250 CALL CHAR(98,"FF01FD05F515D555")
260 REM
270 REM  RIGHT VERTICAL
280 CALL CHAR(99,"5555555555555555")
290 REM
300 REM   LOWER LEFT
310 CALL CHAR(100,"AAABA8AFA0BF80FF")
320 REM
330 REM  LOWER HORIZONTAL
340 CALL CHAR(101,"00FF00FF00FF00FF")
350 REM
360 REM   LOWER RIGHT
370 CALL CHAR(102,"55D515F505FD01FF")
380 REM
390 REM  LEFT VERTICLE
400 CALL CHAR(103,"AAAAAAAAAAAAAAAA")
410 REM
420 CALL COLOR(9,11,2)
430 CALL SCREEN(16)
440 FOR X=1 TO 8
450 CALL COLOR(X,13,1)
460 NEXT X
470 REM
480 REM   START OF MAIN PROGRAM
490 REM
500 INPUT "YOUR FIRST NAME PLEASE ":NAME1$
510 NAME1$=" "&NAME1$
520 INPUT "YOUR SECOND NAME ":NAME2$
530 NAME2$=" "&NAME2$
540 SIZE1=LEN(NAME1$)
550 SIZE2=LEN(NAME2$)
560 IF SIZE1>SIZE2 THEN 590
570 SIZE=SIZE2+2
580 GOTO 600
590 SIZE=SIZE1+2
600 COL=16-INT(SIZE/2)
610 ROW=9
620 REM  FANCY CLEAR SCREEN
630 CALL VCHAR(1,1,32,768)
640 REM  DISPLAY BORDER
650 CALL HCHAR(ROW,COL,96.1)
660 CALL HCHAR(ROW.COL+1,97,SIZE)
670 CALL VCHAR(ROW-1,COL,103,5)
680 CALL VCHAR(ROW-6,COL,100,1)
690 COL=COL+SIZE
700 CALL HCHAR(ROW.COL,98,1)
710 CALL VCHAR(ROW-1,COL,99,5)
720 CALL VCHAR(ROW-6,COL,102,1)
730 ROW=ROW+6
740 CALL HCHAR(ROW,COL-SIZE+1,101,SIZE-1)
750 DISP$=NAME1$
760 LONG=SIZE1
770 ROW=11
780 GOSUB 900
790 DISP$=NAME2$
800 LONG=SIZE2
810 ROW=13
820 GOSUB 900
830 DISP$="PRESS ENTER TO STOP"
840 LONG=LEN(DISP$)
850 ROW=24
860 GOSUB 900
```

```
870 CALL KEY(0,K,S)
880 IF K<>13 THEN 870
890 STOP
900 REM  SUBROUTINE
910 COL=16-INT(LONG/2)
920 FOR X=1 TO LONG
930 FIG=ASC(SEG$(DISP$,X,1))
940 CALL HCHAR(ROW,COL,FIG,1)
950 COL=COL+1
960 NEXT X
970 RETURN
```

SORTING OUT THE SCOTT ADAMS ADVENTURE HINT BOO

by: Tom Hall

If you finally got frustrated with trying to solve the Scott Adams Adventure series, and in desperation purchased a copy of his Official Hint Book, you might find the following extended basic program helpful. It greatly simplifies the process of locating clues in the format used in the hint book. All you have to do to this program is to add the data statements. These consist of the dictionary for each adventure as printed in the Scott Adams Hint Book. The first two data statements should be the adventure number and title, and then simply type in each word in the dictionary, in the same order as it appears in the book. After the last word in the list, add one final data statement, the "@" symbol. This tells the program that it's reached the end of the list.

What I did was to type the data statements as a separate file, beginning with line number 500. I then saved the data statements in merge format, so all you have to do is to load this program, then merge the data statements, and run the program. Then simply type in the number sequence for each clue, and when finished, just hit the <ENTER> key, and the clue will be printed on your screen. To terminate the program, simply hit the <ENTER> key without typing in a number. This way you can use the program with any of the adventures covered in the Scott Adams book!

```
100 !ADVENTURE HELP PROGRAM
110 !BY TOM HALL
120 !EDMONTON 99'ER USERS GROUP
130 !
140 !A PROGRAM TO USE WITH THE SCOTT ADAMS HINT BOOK
150 !
160 CALL CLEAR
170 DIM A$(25),D$(300)
190 READ C$,T$
190 DISPLAY AT(2,3):"CLUES FOR ADVENTURE
#";C$:TAB(14-INT(LEN(T$)/2));T$
200 I=1
210 READ D$(I):: IF D$(I)="@" THEN 22 ELSE I=I+1 :: GOTO 210
220 CX$="" :: FOR I=1 TO 25
230 ACCEPT AT(8,1)BEEP:A$(I):: IF (A$(I)="")*(I=1)THEN CALL
CLEAR :: END ELSE IF A$(I)="" THEN 260
240 CX$=CX$&A$(I)&" "
250 DISPLAY AT(12,1): : : : : : : : : : :CX$: : : :: NEXT I
260 DISPLAY AT(8,1):""
270 X$="" :: FOR N=1 TO I-1
230 X$=X$&D$(VAL(A$(N)))&" " 290 DISPLAY AT(12,1):X$: : : : : :
: 300 NEXT N
310 GOTO 220
```

FOUNDATION 128K CARD REVIEW

by: Danny Ochitwa

For those people who are thinking of adding more than 32K of memory to your TI-99/4A you're in luck! A company called FOUNDATION makes a card that fits into the TI Peripheral Expansion box and gives the user 128K bytes of random access memory to use. In order to better understand the function of the 128K card, a little knowledge of the TI architecture is required. The TMS9900 microprocessor (the processor that your TI uses) like other microprocessors can only directly access 64K bytes of memory. In the TI-99/4A the lower 32K (from 0 to 32,000) is used by the system. There are only 256 bytes of CPU usable RAM there; the rest is taken up by ROMS, DSR'S etc. The upper 32K (from 32,000 to 64,000) is your 32K memory expansion card which of course you can use. (NOTE: the 16K of memory inside the computer is in the Video Display Processor memory space and is not directly addressable by the TMS9900). The FOUNDATION card comes with four banks (chunks) of 32K bytes of RAM and a Device Service Routine in ROM. The first 32K bank behaves exactly like your TI 32K memory expansion card, the other three 32K banks (aprox. 96K) are inactive and are not seen by the computer. A small assembly language routine contained in the TI MINI-MEMORY MODULE could switch in and use any one of the four 32K banks thus using a total of 128K bytes of RAM. This process is known as bank switching and therefore the total amount of memory space that the microprocessor sees at any given moment is never more than 64K. For those of you who are jumping up and down in your seats and yelling "What good is this card to me, I don't know assembly anguage!!!", NEVER FEAR the Device Service Routine is HERE!. The FOUNDATION card comes with a DSR in ROM, which contains two assembly language programs, the first of which is a Disk Emulator Program (DEP for short). This program allows TI BASIC, XTENDED BASIC, and just about any other applications program to use the extra memory, by alloting the first 32K bank as your normal 32K of memory, and the remaining 3 banks as 32K, 32K, and 24K respectively as a psuedo RAM disk; the last bank is only 24K because the DSR uses 8k for house-keeping leaving a total of 88K of RAM to work with. The second program contained in the DSR is a small Memory Manager Module program (MMM for short) to manage the files and programs contained in the RAM disk. The DEP has various ways in which it stores programs and files. You can specify each of the 3 banks as MEM96A, MEM96B or EM96C (remember MEM96C is only 24K bytes) or you can specify the entire 88K by MEM96, or you can specify each of the three banks individually by DSKX.<filename>. As you can see you can only store three programs or files in the RAM disk, one program or file per bank, or you can store a program or one large file in MEM96 using all 88K bytes. Here are some command examples that you could use in EXTENDED BASIC; let's say that you have a program or data in memory called TEST, you can:

SAVE MEM96, SAVE MEM96A (or B or C), SAVE DSKX.TEST, SAVE DSKX.TEST,MERGE, OLD MEM96, OLD MEM96A (or B or C), OLD DSKX.TEST, MERGE DSKX.TEST, RUN "DSKX.TEST", OPEN #1:"MEM96A" (or B or C), etc.

The Memory Manager Module (MMM) program allows you to delete, rename or clear any or all programs or files from the RAM disk; in addition it gives the file type and length. There is one other point that I should mention: when you first turn the computer on, you must initialize the RAM disk by issueing the command DELETE "MEMINIT". The word DELETE is not used in the same way as DELETE "DSK1.<filename>". The word DELETE is simply used to call in the DSR ROM; you are not deleting anything. To use the Memory Manager Module you would issue the command DELETE "MMM", then press FCTN BACK to return to the calling program.

In addition to TI BASIC and EXTENDED BASIC, the FOUNDATION card can be used with a variety of application software, such as TI-WRITER, EDITOR/ASSEMBLER, MULTIPLAN, TERMINAL EMULATOR II etc. Using the card with TI-WRITER, you could store a file in each of the three banks, for example DSKX.LETTER, DSKX.STORY, & DSKX.MYFILE. Each of these files could be worked on during the course of a word processing session and later dumped to disk for permanent storage. The idea here is to use the greater access speed of Ram vs. Disk. Some application programs suggest that for ease of operation, two or more disk drives should be used. In this instance, the FOUNDATION card can easily substitute for a second drive and one package that illustrates this is TI-EDITOR/ASSEMBLER. Using the RAM disk, it is possible to store the source code, object code and listing in each of the three banks. An example of this might be to store the source code in DSKX.SOURCE, assemble and store the object code in DSKX.OBJECT and print a listing to DSKX.LIST; the only disk access needed is for the Editor and Assembler programs. When you have finished writing and debugging your assembly language program you can transfer the source and assembled codes to disk for permanent storage. Unfortunately not all software works as well with the RAM disk as EDITOR-ASSEMBLER and TI-WRITER. MULTIPLAN and the TI DISK MANAGER module both have one major drawback: you can only specify DSK1, 2 or 3; you can not specify DSKX, therefore, using these modules, you can print to the RAM disk but not store files or programs directly to it.

An interesting discovery was made concerning the transfer of DIS-FIX 80 type files from disk to RAM disk; you can use the PRINT option of the EDITOR/ASSEMBLER module to do it. When you get the FILE NAME prompt you can type DSK1.filename, and then on the DEVICE prompt you can type DSKX.<filename>. This method can be used to transfer assembly language programs to the RAM disk and have them always on hand, thus eliminating the need to load them from disk every time you want to use them.

The FOUNDATION card is said to work with all TI and most third party hardware and software, but two exceptions have been found. The first is TI's TEXT-TO-SPEECH program which comes on diskette and the second is the new CORCOMP Disk ontroller Card. The latest word from CORCOMP is that a fix is on the way. Access time for the RAM disk seems to be on average two to four times faster than from a disk drive. This means that a program which takes 12 seconds to load from a disk will only take bout 3 to 6 seconds to load from he RAM disk.

I have tried to give a comprehensive look at the FOUNDATION 128K Memory Card, to help the reader decide if this piece of hardware is a worthwhile investment. Some people might claim that because they have not even begun to use up all the available memory they have, why should they invest in more? In my view, if you run any large application programs or do any serious programming, it is not how much memory you use but how you can use your available memory that counts. There are most likely many more programs that could make excellent use of the RAM disk and some third party companies ave already begun to design their hardware and software to use this extra memory.

If you were to order the FOUNDATION card it would cost you about $330.00 (duty, tax, etc. included). This is about the same price as a disk drive, which would give you greater permanent storage but would not be as flexible for program development. I should point out that the RAM disk is not meant to replace a disk drive but can substitute for one in many applications. If you are seriously thinking of upgrading your TI, the addition of the FOUNDATION 128K card is an excellent choice. Together with the 16K of memory in the console, your TI will have a total of 144K of memory, putting it on par with some of the best home computers n the market today!

CUSTOMIZE YOUR TI FOURTH SYSTEM

by: Tom Hall & Michal Jaegermann

As you become more comfortable with FORTH, the time will come when you are ready to do some customizing to your system disk. Again, it can't be emphasized too often: ALWAYS KEEP AT LEAST ONE, AND PREFERABLY TWO, BACKUP SYSTEM DISKS, so that if something goes wrong and a major part of the system is lost, you'll still be able to go back and recover at least part of your work.

Depending on the type of work you do with FORTH, you'll find there are certain routines that you use far more frequently than others. In most instances you'll be using at least one of the editors, and if you happen to prefer the bit-map editor (the one that uses tiny characters and lets you see the entire screen on your monitor without having to window left and right), you'll eventually get tired of the long wait while the system loads the editor from the source screens on your disk.

There is a routine on your FORTH disk that will allow you to save parts of your system in a format that will load in a fraction of the time normally required to compile it from the source listings on your system disk. This routine is located on screen #83 of your system disk, and is called BSAVE. What BSAVE does is to write a binary image of the current contents of memory to the disk plus some information required by FORTH to allow it to restore a number of system variables that enable it to be called, loaded and properly linked by the word BLOAD. This is a great time-saver, since the system won't have to compile the routines again, because it is saved in a machine-readable form and is ready to be executed as soon as it is loaded from disk. What basically happens when you BSAVE something is this: the entire contents of the area of memory that you specify with the BSAVE command are saved, just as they are, onto a FORTH screen also specified with the BSAVE command. Later, when you BLOAD that same information your memory is restored to the same condition as it was when the data was originally BSAVEd. This is an important point to remember, because whatever you have in memory at the moment you BLOAD a previously BSAVEd screen will be replaced by the BLOADed information.

Another advantage of BSAVE is the amount of disk space used. Since binary format is much more efficient than readable ASCII text, the BSAVEd version of your application will require much less disk space than the original source listing.

Perhaps an example of using BSAVE will make this clearer. Let's say that you have loaded the 40-column editor (the one that requires windowing left-to-right), and you would like to put this on your disk in such a way that it is automatically loaded every time you boot your FORTH disk. We'll assume that you have just loaded FORTH and you now have the menu and the words TI FORTH on your screen. At this point you should issue the command -EDITOR. This will load and compile screens 34-38 from your system disk, along with screen #33 (-SYNONYMS). As soon as the "ok" prompt appears, you're ready to call up BSAVE. You can now type -BSAVE and this will load screen #83 for you. Before you actually BSAVE anything, there are a few points you should keep in mind. The main thing to remember is that there should be no "holes" in memory created by BLOADing anything, so the best bet, at least for your initial attempt, is to save everything currently in memory. If you load -DUMP you have a command called VLIST, which displays a list of every word currently compiled in your FORTH system vocabulary. The listing begins with the last (most recent) definition loaded, and proceeds downward until you reach the FORTH kernel itself. You will see a word called TASK. This definition is actually a dummy, since it contains no executable instructions, but merely marks the boundary line between the FORTH kernel (the basic operating system contained in the file FORTHSAVE), and the "extras" which are loaded from screens.

When you use the BSAVE command, the first piece of information required is a starting point, usually the address of a word in your VLIST. The word TASK is a convenient word to use, because it marks the end of the kernel. BSAVE will save everything from the memory location you specify right up to HERE, which is a word that returns the address of the next available space in the dictionary for a new definition. (In other words, if you were to define a new word at this moment, it would be located at the address pointed to by the current value put on the stack by HERE.)

Okay, so we've loaded our EDITOR, and are now ready to save it with BSAVE. We know that the editor was originally stored in screens 34-38 on the system disk, so we may as well use screen #34 as the starting point of the BSAVEd editor. We've also decided to use the word TASK as the cutoff point. So: how do we find the address of TASK? We use the symbol "'" (pronounced "tick"), followed by the word TASK. This will place the address of the parameter field of the word TASK on the stack, and this is the first part of the information BSAVE needs to have in order to do its thing. The next piece of information is where to store the information, and that, as we've already decided, will be screen #34. So the command to do all of this becomes:

```
' TASK DECIMAL 34 BSAVE <enter>
```

When the BSAVE command is complete, there is a number left on the stack; this number is the number of the next available screen immediately following the last screen used by the BSAVEd data.

The best way to discover how to make BSAVE work for you is to experiment. Make an extra copy of your system disk, and try various combinations using BSAVE. Make sure that you don't overwrite anything on your system disk that you might need later on.

Now that we've BSAVEd the editor, we want to have this BSAVEd routine automatically loaded when FORTH is booted. To do this, put the command "34 BLOAD" into screen #3. As soon as the FORTH kernel is loaded, it automatically loads and executes any instructions in screen #3, so if you place the BLOAD command there, each time you load FORTH, your BSAVEd routines will also be automatically loaded!

****

You can skip the following information at first reading, but you should read it after you've gotten the hang of BSAVEing and BLOADing.

****

The procedure described above provides a "fool-proof" method leading to a binary file with "no holes" in memory. Actually, if you were converting a disk for distribution then it would be possible to start BSAVEing from ' MENU LFA instead of ' TASK and the results would be exactly the same. This is because the link field address of the word MENU is the first address which was not loaded directly from disk but was compiled by FORTH from source screens. But the difference between ' MENU and ' TASK is a mere two bytes, so probably isn't worth the bother. On the other hand, you could also start your procedure with, say, (hex) >A000 on stack and you would get a file which after BLOADing would have exactly the same effect on memory as the previous one, only much bigger. If you would like to know how many screens you need for a binary format file BEFORE its creation type the following, replacing <starting address> with its value: HERE <starting address> 1000 / 1+ . <enter> and what will appear is the number of screens you need for your file. (Yes Virginia, this is DECIMAL 1000, 3E8 in HEX.)

Please bear in mind that when you were performing BSAVE, lets say, of the 40 column EDITOR, you converted into binary format all contents of memory at that moment. This means that not only EDITOR words but also every word **compiled** from screens 3 (welcome screen), 20 and 21 (CONDITIONAL LOAD), 33 (SYNONYMS) and at least 83 (BSAVE by itself) all became a part of the BSAVEd file. As a result, the source listings for all of these definitions now may be removed. If they are compiled before you BLOAD your file back they will be clobbered anyway by the contents of your file and they are only delaying loading. If they are compiled after - then you are defeating the purpose of BSAVE and you will get a lot of "ISN'T UNIQUE" messages.

These considerations do not apply to actions which are executed on screens you are replacing. They are not in memory, so they will be not reflected in your binary file. If you still want them to be executed you have to leave them in their proper places. Of course at this point you will have to remove all references to those screens which now no longer exist, otherwise very strange things are likely to happen.

If you choose to BSAVE the 64 column EDITOR then together with it you will also load, at least, the same routines as before, i.e., MENU, CLOAD, and the information in screen #3 (without the other editor), -CODE, -TEXT, -GRAPH, -GRAPH2 and -SPLIT -- in other words, a **lot** of screens. On the other hand, most of these routines are quite useful by themselves and they do not have to be bundled with the 64-column EDITOR. But you do not need to keep them all in one binary file. Of course you may save everything from ' TASK but what a waste of disk space and loading time. The better way is to create a chain of binary files, each one picking up the job at the point where it was left by the previous one. The editor loading word -64SUPPORT may be redefined into something along this pattern:

```
: -64SUPPORT 20 BLOAD 40 BLOAD 37 BLOAD 22 BLOAD ;
```

Everything will be ok as long as the last BLOADed file is the last one which was BSAVEd for that particular application (the order will have to be the same here as it was when you did the BSAVE if you have many words to load your application back). The simplest and universal way to find the starting address for the next BSAVE is to load everything (and nothing else) which has to be loaded before you load the application. Now execute the word HERE which will leave on stack your starting address. If you are not going to perform any fancy footwork on the stack before you BSAVE then you even do not have to print it out. But if you will feel more comfortable - display it on your screen, load your application and BSAVE from that address. BSAVE always will finish with HERE (a new one!). Do not forget that you can always FORGET (pun intended) part of the contents of your memory before you execute HERE to find the starting address. For example, you can execute FORGET MENU, which is pretty extreme but can be used for our purposes instead of ' TASK. This opens possibilities of creation of "overlay structures" (translation from computerese - loading a big application in pieces which reuse the same part of memory to perform different tasks, like "TI-WRITER"), replacing one editor with another without creating conflicts and so on.

Finally let us note that at least one of our binary files contains an image of BSAVE itself which cannot be used for anything else except BSAVEing. Since BSAVE's usefulness is limited to one particular function, why have it included in your BSAVEd application? There is a very simple method for saving everything you want except BSAVE. This will save you memory (around 200 bytes), but also, which is maybe even of greater importance, can make a difference of a whole screen (4 sectors) of disk space depending on the overall size of your memory image. One fly in the ointment. You have to modify Screen #83 and the definition of BSAVE itself. Do this as follows: Squeeze the layout of screen #83 to get an extra two lines. In line 1 change DECIMAL into HEX and push the colon definition down in order to have space in lines 2 and 3. In those lines type the following: HERE 3D00 ! CURRENT @ 3D02 ! LATEST 3D04 ! CONTEXT @ 3D06 ! CONTEXT @ @ 3D08 ! VOC-LINK @ 3D0A !. Next in the colon definition replace every (this is important! every!) occurrence of HERE with 3D00 @, every occurrence of CURRENT @ with 3D02 @ and so on. (It means that CONTEXT @ @ is replaced with 3D08 @). Change 29801 into 7469

(this is "ti") and 1000 into 3E8. You may also add after [illegible] in the last line ." Next screen is #" . (the last dot is [illegible] dot - not period). FLUSH it and you are done.

How to use it? Let us assume that you're starting with the application which will be BSAVEd, but which is not yet compiled into memory. Execute HERE and load the whole application. Only after that load screen #83 with BSAVE and execute BSAVE immediately (starting address is very kindly waiting on stack). If you are going in this very moment to create the next binary file then FORGET BSAVE, prepare memory and perform the whole procedure once again. This version of the utility has to be ALWAYS the last thing loaded. But your binary files will never be aware of the existence of BSAVE.

Now happy BSAVEing! And remember: "When everything else fails read the instructions". TI FORTH documentation in some places is surprisingly good (after fourth reading).

MERRY

Christmas

from

THE GAMES GANG

FAMILY COMPUTERS

435-4636

9872-63rd AVENUE, EDMONTON, ALBERTA T6E 0G6