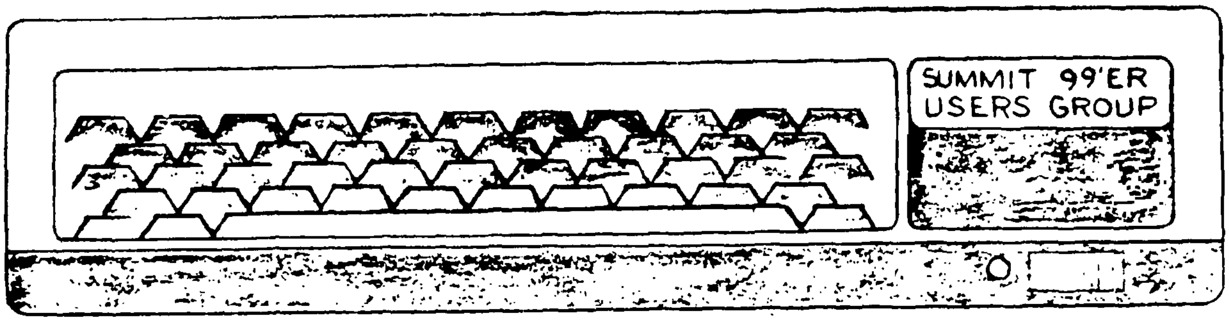


FEBRUARY 1985 Vol. 3 No. 2

This month's meeting will be held on Thursday, February 21st at Cuyahoga Falls High School at the corner of 4th and Stow Street in Room 413- Physic's Lab. The March meeting will be held on March 21st also. Please remember to sign in.

PROGRAM

This month's program will be on Fourth. Dan Fedak will be giving the demonstration.

BASIC CLASS

Rich will be teaching the basic class. Remember to bring your Blue book in to class.

MEMBERSHIP DUES

Those people whose membership expired in January will no longer receive this month's newsletter. Please remember to renew your membership dues. They are now $15.00 this year.

NEWSLETTER DEADLINE

The deadline for the March newsletter is March 9. We are in need of articles. This month's newsletter is made up almost entirely from other users group newsletters.

TIPS FROM THE TIGERCUB

#19

TIGERCUB SOFTWARE
156 Collingwood Ave.
Columbus, OH 43213

The entire contents of *Tips from the Tigercub* Nos. 1 through 14, with more added, are now available as a full disk of 50 programs, routines and files for just $15.00 postpaid!

Nuts & Bolts is a diskfull of 100 (that's right, 100!) XBasic utility subprograms in MERGE format, ready for you to merge into your own programs. Contents include 13 type fonts, 14 text display routines, 12 sorts and shuffles, 9 data saving and reading routines, 9 wipes, 8 pauses, 6 music, 2 protection, etc., and now also a tutorial on using subprograms, all for just $19.95 postpaid!

And I have about 140 other absolutely original programs in Basic and XBasic at only $3.00 each!(plus $1.50 per order for casette, packing and postage, or $3.00 for diskette, PPM) Some users groups charge their members that much for public domain programs! I will send you my descriptive catalog for a dollar, which you can then deduct from your first order.

Folks, I just can't afford to keep mailing out these Tips if you don't BUY something once in awhile! I am hearing from more and more groups - [illegible] on my mailing [illegible]. am having to r[illegible]ack. I am dropping t[illegible] groups which don't [illegible] any indication that [illegible] members ever get [illegible]see the Tips, and I'll have to cut further. If you do send me an order, or even ask for my catalog, mention your users group so I'll know there is someone still alive out there!

If you know of any schools in your area, especially elementary schools, that have TI-99/4As in the classroom, won't you please give me their address? I'll send them a free catalog.

Danny Michael has improved his graphics screen dump to include rotate and double size! It is in assembly, very fast, and runs out of XBasic, E/A module or Mini Memory. He has also written an assembly Neatlist program which lists an XBasic program to a printer in single line statements, indented, expanded, etc., very useful for debugging, setting up pre-scan, etc.

These are freeware, pay if you want and whatever you want. Just send an initialized disk for either one, or two disks (or SSDD or flippy) for both, in a returnable mailer with ENOUGH RETURN POSTAGE, to

Danny Michael,
Rt 9 Box 460
Florence, AL 35630.

John Hamilton of the Central Iowa Users Group will send you his 22-page boklet of "99 Tips" for the TI-99/4A, for just $4.00. The address is

John Hamilton,
4228 E. Clinton, Des Moines IA 50317.

I have been experimenting with [illegible]writer, and this issue of the Tips is being printed in 4 columns, right justified directly from the printer. Here's how -

Use TI-Writer, editor mode, in any line length you want. The first line should be .RM 27;FI;AD but don't use any other formatter codes. Don't indent paragraphs. Use some other character as a temporary substitute for any ^, @, & or # in the text. Don't include any program listings, yet.

Save the file as DSK1.TEXT. Print an edit copy. Then go into formatter mode. Select DSK1.TEXT to be printed, but instead of your printer spec, type DSK1.TEXT2. Your file will now be in 28-column format and right justified, and indented.

If the text is to include any program listings, run them through my 28-Column converter (see Tips #18), using the Editor option of that program.

Go back to TI-Writer editor and load DSK1.TEXT2. Merge in the program listings. Then PF to print file, but instead of a printer spec, type C DSK1.TEXT3. When it has printed to disk, LF the DSK1.TEXT3 and you will find that all control characters are gone.

Now for a bit of editing. Delete the 3 blank lines at the beginning, and the 6 blank lines that have appeared after every 60th line. Center the title by erasing with the space bar and retyping - do NOT use FCTN 2! Also replace any temporary characters with the ^, @, & or #.

You will print 4 columns of 60 lines per page, so the total lines in your file must be a multiple of 240. Add enough blank lines to the end of the file to reach that count.

Save that file back to disk as DSK1.TEXT3. Now go into XBasic, key in this program and RUN!

```
100 OPEN #1:"DSK1.TEXT3",INP
UT :: OPEN #2:"PIO",VARIABLE
 255 :: PRINT #2:CHR$(15);CH
R$(27);CHR$(69):: DIM B$(240
)
110 FOR A=1 TO 2 :: FOR B=1
TO 240 :: LINPUT #1:B$(B)::
NEXT B
120 FOR C=1 TO 60 :: PRINT #
2:TAB(10);B$(C);TAB(41);B$(C
+60);TAB(72);B$(C+120);TAB(1
03);B$(C+180):: NEXT C :: PR
INT #2:CHR$(27);CHR$(97);CHR
$(6):: NEXT A :: CLOSE #1 ::
 CLOSE #2 :: END
```

The A loop is for a 2-page printout of 480 lines, of course.

You can modify this routine to print in 2 or 3 columns, adjust the margins, change the type font or size, rewrite for your own printer, etc. And the column width can be anything you want, just change that .RM 27 in the first line of the text (don't forget that the left margin is set at 0, not 1).

If you want a 2-column page, you can dump the file back to disk instead, and then print it out of TI-Writer editor. Use this routine, modified as you wish.

```
100 !Opens a file TEXT3 of 2
40 lines 35 char long and co
nverts it into a file which
can be printed out of TI-wri
ter Editor as 2 pages in 2 c
olumns
110 OPEN #1:"DSK1.TEXT3",INP
UT :: OPEN #2:"DSK1.TEXT4",O
UTPUT :: DIM B$(120)
120 FOR A=1 TO 2 :: FOR B=1
TO 120 :: LINPUT #1:B$(B)::
NEXT B
130 FOR C=1 TO 60 :: PRINT #
2:"     "&B$(C)&RPT$(" ",38-
LEN(B$(C)))&B$(C+60):: NEXT
C :: FOR D=1 TO 6 :: PRINT #
2:" " :: NEXT D :: NEXT A ::
 CLOSE #1 :: CLOSE #2
```

It is best to run a program to set up your printer, and leave it turned on, before printing that file out of the Editor. It is not at all easy to imbed control characters in the file, because they affect the line in all columns and also shift the lines out of alignment.

I understand that there a couple of kids who wait every month for their dad to key them in a bit of nonsense from the Tigercub, so -

```
100 !KEYZAP - by Jim Peterso
n
110 DISPLAY AT(6,11)ERASE AL
L:"KEYZAP" :: DISPLAY AT(12,
1):"  Zap the Zprite by typ
ing the key in the correspon
dingposition on the keyboard
."
120 DISPLAY AT(24,10):"Press
 any key" :: CALL KEY(0,K,S)
:: IF S=0 THEN 120
130 RANDOMIZE
140 CALL CHAR(47,"817EA5B199
A5423C")
150 CALL CLEAR :: T=0 :: CAL
L FLASH(T)
160 CALL KEY(3,K,ST):: IF ST
=0 THEN 180
170 C=C+1 :: IF C=101 THEN 1
90 ELSE CALL KEYBOARD(K,T)
180 CALL MOTION(#1,25*RND-25
*RND,25*RND-25*RND):: CALL C
OINC(#1,#2,16,A):: IF A=0 TH
EN 160 ELSE CALL FLASH(T)::
GOTO 160
190 CALL DELSPRITE(ALL):: DI
SPLAY AT(12,9):"GAME OVER" :
: DISPLAY AT(14,9):"SCORE";T
 :: DISPLAY AT(16,9):"PLAY A
GAIN?"
200 CALL KEY(3,K,S):: IF S<1
 THEN 200
210 IF K=89 THEN C=0 :: GOTO
 150 ELSE END
220 SUB KEYBOARD(K,T)
230 IF FLAG=1 THEN 250 :: FL
AG=1
240 KEY$="1234567890=QWERTYU
IOP/ASDFGHJKL;"&CHR$(13)&"ZX
CVBNM,."
250 IF (K=47)+(K=61)+(K=13)T
HEN SUBEXIT ELSE X=POS(KEY$,
CHR$(K),1):: Y=ABS(X>11)-(X>
22)-(X>33)+1 :: R=Y*6 :: C=(
(X+(Y>1)*(Y-1)*11)*3)
260 CALL SPRITE(#2,42,16,R*8
-7,C*8-7):: CALL COINC(#1,#2
,16,N):: IF N=0 THEN SUBEXIT
270 CALL FLASH(T):: SUBEND
280 SUB FLASH(T):: FOR W=1 T
O 10 :: CALL SCREEN(16):: CA
LL SCREEN(8):: NEXT W :: CAL
L SPRITE(#1,47,2,1,1):: T=T+
1 :: DISPLAY AT(1,20):T :: S
UBEND
```

And here's another -

```
100 ! QUICK & DIRTY DOODLER
      by Jim Peterson
Use joystick #1. Press fire
button to change color or
pattern, Enter to clear the
screen.
110 DATA FFFFFFFFFFFFFFFF,FF
,0101010101010101,0000000000
0000FF,8080808080808080,01020
4081020408,8040201008040201,
FF818181818181FF
120 CALL CLEAR :: FOR J=1 TO
 8 :: READ CH$(J):: NEXT J
130 FOR CH=32 TO 136 STEP 8
:: FOR CN=CH TO CH+7 :: X=X+
1 :: CALL CHAR(CN,CH$(X))::
NEXT CN :: X=0 :: NEXT CH ::
 CALL CHAR(32,"0")
140 CALL SCREEN(16):: FOR S=
2 TO 14 :: CALL COLOR(S,S+1,
1):: NEXT S :: R=12 :: C=16
:: CH=33
150 CALL HCHAR(R,C,CH):: CAL
L FASTJOY(C,R,Q):: IF Q=18 T
HEN CH=CH+1+(CH=143)*110
160 CALL KEY(0,K,S):: IF K=1
3 THEN CALL CLEAR :: GOTO 15
0 ELSE 150
170 SUB FASTJOY(C,R,Q):: CAL
L JOYST(1,X,Y):: CALL KEY(1,
Q,S):: X=SGN(X):: Y=-SGN(Y):
: C=C+X+(C=32)-(C=1):: R=R+Y
+(R=24)-(R=1):: SUBEND
```

And a pretty one -

```
100 CALL CLEAR :: CALL SCREE
N(2):: FOR S=2 TO 8 :: CALL
COLOR(S,15,1):: NEXT S :: DI
SPLAY AT(12,7):"KALEIDOSQUAR
ES" ! by Jim Peterson
110 FOR CH=40 TO 136 STEP 8
:: FOR L=1 TO 4 :: RANDOMIZE
 :: X$=SEG$("0018243C425A667
E8199A5BDC3DBE7FF",INT(16*RN
D+1)*2-1,2)
120 B$=B$&X$ :: C$=X$&C$ ::
NEXT L :: CALL CHAR(CH,B$&C$
):: B$,C$=NUL$ :: NEXT CH
130 FOR S=2 TO 14 :: X=INT(1
5*RND+2)
140 Y=INT(15*RND+2):: IF (Y=
X)+(Y=8)THEN 140
150 CALL COLOR(S,X,Y):: NEXT
 S
160 AR,R,AVR,VR=1 :: AC,C,AH
C,HC=4 :: TT=24 :: XX,XT=13
170 FOR L=1 TO 12 :: T=TT ::
 XT=XX :: R=AR :: VR=AVR ::
C=AC :: HC=AHC
180 FOR J=1 TO XT :: X=INT(1
3*RND+2)*8+24 :: CALL HCHAR(
R,HC,X,T):: CALL HCHAR(25-R,
HC,X,T):: CALL VCHAR(VR,C,X,
T)
190 CALL VCHAR(VR,31-C,X,T):
: T=T-2 :: HC=HC+1 :: VR=VR+
1
200 NEXT J :: AR=AR+1 :: AVR
=AVR+1 :: AC=AC+1 :: AHC=AHC
+1 :: TT=TT-2 :: XX=XX-1 ::
NEXT L
210 IF INT(2*RND)<>0 THEN 23
0
220 FOR S=INT(12*RND+2)TO 14
 :: CALL COLOR(S,1,1):: NEXT
 S
230 FOR J=1 TO INT(20*RND+1)
:: S=INT(13*RND+2):: X=INT(1
5*RND+2):: Y=INT(15*RND+2)::
 CALL COLOR(S,X,Y):: NEXT J
240 CALL SCREEN(INT(15*RND+2
)):: ON INT(5*RND+1)GOTO 130
,160,220,230,240
```

The challenge in Tips #16 was - how can you store a hundred or more values of any size, positive or negative, integer or non-integer, even in exponential notation, without dimensioning an array or opening a file, and then link to another program with a RUN statement and recover those values - not by reading them from the screen? I had just one reply! Was it too easy, too hard, or doesn't anyone care? Anyway -

```
20591 SUB CHARSAVE2(CH,N)::
N$=STR$(N):: N$=RPT$("0",16-
LEN(N$))&N$
20592 IF POS(N$,".",1)=0 THE
N 20593 :: N$=SEG$(N$,1,POS(
N$,".",1)-1)&"A"&SEG$(N$,POS
(N$,".",1)+1,LEN(N$))
20593 IF POS(N$,"+",1)=0 THE
N 20594 :: N$=SEG$(N$,1,POS(
N$,"+",1)-1)&"B"&SEG$(N$,POS
(N$,"+",1)+1,LEN(N$))
20594 IF N<0 THEN N$=SEG$(N$
,1,POS(N$,"-",1)-1)&"F"&SEG$
(N$,POS(N$,"-",1)+1,LEN(N$))
20595 CALL CHAR(CH,N$):: SUB
END
```

And to recover the values -

```
20596 SUB READCHAR(CH,N):: C
ALL CHARPAT(CH,CH$)
20597 IF POS(CH$,"A",1)=0 TH
EN 20598 :: CH$=SEG$(CH$,1,P
OS(CH$,"A",1)-1)&"."&SEG$(CH
$,POS(CH$,"A",1)+1,LEN(CH$))
20598 IF POS(CH$,"B",1)=0 TH
EN 20599 :: CH$=SEG$(CH$,1,P
OS(CH$,"B",1)-1)&"+"&SEG$(CH
$,POS(CH$,"B",1)+1,LEN(CH$))
20599 IF POS(CH$,"F",1)<>0 T
HEN CH$="-"&SEG$(CH$,POS(CH$
,"F",1)+1,LEN(CH$))
20600 N=VAL(CH$):: SUBEND
```

Here's a jewel of a routine from Danny Michael, to avoid those lockups and other foul-ups that occur when you CALL INIT after you have already CALLed INIT -

```
CALL PEEK(8198,A):: IF A<>17
0 THEN CALL INIT
```

The best way to edit a program is to type NUM and the first line number, then Enter will take you through line by line with no danger of accidentally deleting a line. The edit functions will still work, and FCTN 4 gets you out of the NUM mode.

MEMORY FULL!

Jim Peterson

This article comes to us from Edmonton, Canada's users group, 1/85 issue.

BASIC PROGRAMMING: -CASSETTE DATA FILES

by: Bob Pass

Some of you may not be aware that you can use your cassette recorder to do more than just load or save programs. Your cassette can also store data files which can be read into the console by a running program, modified by the user, and saved for later reference. By learning to use the basic commands OPEN#, INPUT#, PRINT#, & CLOSE# you can open up new horizons with your TI 99/4A by being able to save & recall data from cassette.

One important point to get clear first is the concept of "Buffers". *The word "buffer"* is used to describe an area of computer memory (or hardware) that is used to temporarily store data that is to be written into or out of the computer. Buffers are required whenever the computer must talk or listen to another device which does not operate at the same speed or in the same manner as the computer does. For example, since you cannot type at computer speed, the keyboard on your machine uses a buffer to pass information to the processor. Similarly, a cassete tape recorder simply cannot handle data at computer speeds; consequently the computer must use a buffer to transfer information to the device. Briefly, a buffer is a block of memory of fixed size which is referenced by a numerical tag (you can have more than one buffer available). When data is to be transfered, the computer will load the buffer with data untill it is full. Then the same buffer is read by a device service routine at a speed wich is compatible with the output device. When the buffer is emptied, more data is written into it untill the data transfer is complete. An important point to realize is that the transfer of data from the buffer to the external device is automatically done only if the buffer is full. If the buffer is only partially loaded when your application program ends, this data could be lost unless you instruct the system to close all open files (buffers). This will cause the system to finish dumping the buffer to the cassette. The last data item is always an end of file marker.

When data is read back into the computer, the process is reversed with the computer looking for the end of file mark so that it knows when to stop reading the buffer and shut down the external device.

As mentioned earlier, the buffers have a numerical tag. In TI basic you can specify a tag from 1 to 255 with each buffer being distinct from any others by the tag number. Buffer number 0 is reserved for system use and is, in fact, the keyboard (and screen) buffer mentioned earlier. You can use more than one buffer at a time for different purposes; however the number of buffers that are open at the same time is limited to a default of 3. If you need more than three open buffers, use the CALL FILES(n) command where "n" is any number from 1 to 9. Note that this will limit you to a maximum of 9 open files or buffers at a time. The CALL FILES command must be used in the following way:

```
NEW
CALL FILES(n)
NEW
```

Now load your application program in the usual way and you will have the required number of files or buffers available. CALL FILES may not be used within a program; it must be entered in the command mode. Consequently, any program requiring more than three buffers must have the appropriate CALL FILES executed first. Each buffer that has been reserved occupies 519 bytes of RAM (except the first which takes up 1052 bytes) so it is wise to keep the required number of buffers as low as possible to conserve memory space.

Below is a short program that will allow you to set up & maintain a short telephone number list and save it to tape for later recall and/or modification. It can be easily modified to hold more data as you see fit. There is one important thing that should be mentioned:

DO NOT USE A PROGRAM TAPE TO STORE DATA!

You wouldn't be the first to overwrite a program with a data file. It would be wise to keep your data files on separate tapes, preferably one file per tape to avoid confusion.

OPEN# This command prepares the system to transfer data to an accessory device. The buffer number (the TI manual calls buffers "files") is specified by you as well as the device name (such as CS1) to which the data is to be written. Additionally, you must specify the structure of the data file to be written on the cassette. Untill you have become thoroughly famillıar with the TI User's Reference Guide and you have gained some experience working with cassette files, always specify "CS1", INTERNAL, SEQUENTIAL, FIXED for your file structure. Further more, you must tell the system the size of the data strings to be written (so that it will know how to read the data back later) by placing 64, 128, or 192 after the FIXED notation. You must plan the maximum length of each data item to be stored; if, for example you chose FIXED 64 in the OPEN# statement and then wrote a data item 70 characters long, the last 6 characters would either be lost or would overflow into the data of the next character string producing an unwanted concatenation or "trashed file". On the other hand, if your string was only 60 characters long, the system would automatically pad the string out to 64 with dummy characters which are stripped off when the data is recalled. Line number 260 from the program below contains the OPEN# statement. Notice that the size is 64 and that further more the data entry routine does not check for strings longer than 64 characters (see lines 430 to 460). By playing with this fact, you will be able to see what happens if you enter a very long string, saving the data to tape, and then reading it back.

CLOSE# This statement will cause the computer to empty the specified buffer number of pending data by completeing the transfer sequence. See lines 300 and [illegible]. To prevent corrupted files, always close your opened files under program control. Treat the OPEN# and CLOSE# statements like matched bookends. Do not place any statements between them that could cause a transfer of the program control out of the program block defined by these two statements. If you experience a program error message during a file transfer sequence, do not use FCTN QUIT as this will cause all data in the buffers to be lost. Instead, type BYE, RUN, NEW, OLD, SAVE, or LIST or else EDIT a line number; either of these actions will cause the buffers to be closed properly.

PRINT# This causes the system to transfer (print) data **FROM** the computer **TO** the device identified by and in the format specified by the OPEN# statement whose buffer number corresponds to that of the PRINT# statement. See line number 500.

INPUT# **This** statement is the opposite of the PRINT# statement. It reads data **INTO** the system **FROM** an external device. The buffer number must match the corresponding OPEN# which conditions the system as to what format the incomming data will be in. See line number 280. Notice the comma in this line and also in the PRINT# statement (line 500). This is a data element separator which tells the system where the end of each data block is located; ie, when to pad the string out to the size specified in the "FIXED" parameter of the open statement. If you used a semi-colon (;) here, the two data elements would be joined together.

I encourage you to enter this program and experiment with it. Once you understand how it works, I am sure you will find many more applications of this concept. For further reading, refer to your User's Reference Guide, pages II-118 through II-136.

```
THIS LISTING IS IN THE SAME
FORMAT AS IT WILL APPEAR ON
YOUR SCREEN UPON ENTRY.
THIS IS AN ASTERISK (*),
THIS IS A ZERO (0), AND THIS
IS A LETTER O.
THE FOLLOWING LINE IS A ROW
OF ALTERNATING DASHES AND
SPACES.
- - - - - - - - - - - - - -
100 REM ******************

110 REM * PHONE LIST *

120 REM ******************

130 REM TI BASIC
140 REM REQUIRES CASSETTE
RECORDER AND
150 REM CASSETTE CABLE.
160 REM DEMO OF CASSETTE     5
170 REM DATA STORAGE.
180 DIM NAME$(10),PHONE$(10)
190 CALL CLEAR
200 PRINT " PHONE LIST"::
::
210 PRINT "1-READ FILE FROM
TAPE"::"2-REVIEW AND ENTRY O
F DATA"::
215 PRINT "3.  SAVE FILE TO T
APE"::"4.  QUIT":::::
220 INPUT CHOICE
230 IF (CHOICE>4)+(CHOICE<1)
=-1 THEN 190
240 ON CHOICE GOTO 250,320,4
70,540
250 REM READ TAPE FILE
260 OPEN #1:"CS1",INPUT ,INT
ERNAL,SEQUENTIAL,FIXED 64
270 FOR N=1 TO 10
280 INPUT #1:NAME$(N),PHONE$
(N)
290 NEXT N
300 CLOSE #1
310 GOTO 190
320 REM ENTER DATA IN FILE
330 CALL CLEAR
340 PRINT "WHICH RECORD NUMB
ER"
350 INPUT ENTRY
360 IF (ENTRY>10)+(ENTRY<1)=
-1 THEN 330
370 CALL CLEAR
380 PRINT "ENTRY NUMBER ",EN
TRY," IS:":NAME$(ENTRY)::
390 PRINT "WHOSE PHONE # IS:
":PHONE$(ENTRY)::
400 PRINT "1-ENTER NEW NAME"
:"2-ENTER NEW PHONE NUMBER":
410 PRINT "3-TRY ANOTHER REC
ORD":"4-EXIT REVIEW AND ENTR
Y MODE"
420 INPUT CHOICE
430 IF (CHOICE>4)+(CHOICE<1)
=-1 THEN 370
440 ON CHOICE GOTO 430,450,3
30,190
450 INPUT "NAME? ":NAME$(ENT
RY)
460 GOTO 370
470 INPUT "PHONE #? ":PHONE$
(ENTRY)
480 GOTO 370
490 REM SAVE FILE TO TAPE
500 OPEN #1:"CS1",OUTPUT,INT
ERNAL,SEQUENTIAL,FIXED 64
510 FOR N=1 TO 10
520 PRINT #1:NAME$(N),PHONE$
(N)
530 NEXT N
540 CLOSE #1
550 GOTO 190
560 END
```

LIST OF BOARD MEMBERS AND THEIR HOME PHONE NUMBERS

President, Norm Sorkin	678-2360
Vice President,	
Librarian, Bert Haase	753-7846
V.P. Program, John Tuesday	644-2616
Secretary, Vicky Chrisman	784-0943
Treasurer, Betty Duncan	633-5217
Educational Director, Rich Williams	626-2423
Editor, Kathi Anderson	923-7530

This article comes to us from Atlanta's A9CUG, 1/85 issue.

**** ASSEMBLY LOADER ****

This is an assembly loader routine that automatically loads assembly programs with no program names.(Like Atarisoft games or some TI games) It will automatically load ANY assembly program including those with program names. After loading those with program names, it returns you to the Editor/Assembler. Just select option 4 (RUN) and type the program name. It will then run.

Another way to do it is just make an extended basic CALL LOAD(FILENAME), CALL LINK(PROGRAM NAME). Then the program will run automatically from extended basic. I tried to use an external REF to get my program to run the one you select with a program name, but both programs have to be already loaded for it to work. (Self-defeating isn't it?).

There are a few things you will have to change to adapt the routine to run the specific programs you want to use with this routine. After the listing, I'll explain what you'll have to change.

Here is the listing:

```
        REF DSRLNK,VMBW,VSBW,LOADER
        REF KSCAN
        DEF BEGIN
PABBUF  EQU >1000
PAB     EQU >F80
STATUS  EQU >837C
PNTR    EQU >8356
SAVRTN  DATA 0
TDATA   DATA >0005,PABBUF,>5000,>0000
        DATA >000B
        TEXT 'DSK1.TENNIS'
        EVEN
CDATA   DATA >0005,PABBUF,>5000,>0000
        DATA >000E
        TEXT 'DSK1.CENTIPEDE'
        EVEN
CLOSE   BYTE >01
MYREG   BSS >20
TEN     TEXT '1. TENNIS'
CEN     TEXT '2. CENTIPEDE'
BEGIN   LI R0,34
        LI R1,TEN
        LI R2,9
        BLWP @VMBW
        LI R0,98
        LI R1,CEN
        LI R2,12
KPREP   CLR R0
        MOVB R0,@>8374
        LI R4,>3100
        LI R2,>3200
        LI R3,>2000
KCHECK  CLR R1
        BLWP @KSCAN
        MOVB @STATUS,R5
        COC R3,R5
        JNE KCHECK
        MOVB @>8375,R1
        CB R1,R4
        JEQ TLOAD
        CB R1,R2
        JEQ CLOAD
        JMP KPREP
TLOAD   MOV R11,@SAVRTN
        LWPI MYREG
        LI R0,PAB
        LI R1,TDATA
        LI R2,>20
        BLWP @VMBW
        LI R6,PAB+9
        MOV R6,@PNTR
        BLWP @LOADER
        JMP CLOSEF
CLOAD   MOV R11,@SAVRTN
        LWPI MYREG
        LI R0,PAB
        LI R1,CDATA
        LI R2,>20
        BLWP @VMBW
        LI R6,PAB+9
        MOV R6,@PNTR
        BLWP @LOADER
CLOSEF  MOV R6,@PNTR
        MOVB @CLOSE,R1
        LI R0,PAB
        BLWP @VSBW
        MOV R6,@PNTR
        BLWP @DSRLNK
        DATA 8
        CLR R0
        MOVB R0,@STATUS
        MOV @SAVRTN,R11
        RT
        END
```

Well, that's the listing. Now I'll try to explain what you'll have to change to adapt the routine to your use. You'll want to change the program names from CENTIPEDE TENNIS to whatever you want. You don't have to just have 2 programs either, that was just for simplicity.

Back up in the beginning of the listing there are 2 symbols TDATA CDATA. To put your files in place of the 2 I used, you'll want to change the command TEXT 'DSK1.TENNIS' to TEXT 'DSK1.pfile name'. In the DATA statement directly above that, there is the statement DATA >000B. >000B is the length in characters of the file name DSK1.TENNIS in hexadecimal. Change that to however long your file name is including "DSK1.". For example, "DSK1.LOADER" would be 11 characters long, so the data statement would read DATA >000B and the text below it would read TEXT 'DSK1.LOADER'. To add more programs to the routine, copy the first line of DATA from either TDATA or CDATA(it's the same), put TEXT 'DSK1. file name' below it, put another DATA statement below it with the length of the filename below it, and put an EVEN statement below it(like in the program.)

There are a few other things you have to do to add more programs to the routine. Two lines after the KPREP symbol, you'll see a command LI R4,>3100 That's the ASCII code in hex for "1". You'll have to load registers 7-15 with the hex ASCII codes for 3-9. Next you'll have to put a CB R7,R1; R8,R1 ... with a JEQ following each one telling it to jump to a symbol you create. The symbol should have the commands just like us...
SYMBOLS TLOAD CLOAD do from MOV R11, @SAVRTN TO JMP CLOSEF. All of that should be under the symbol you create. That should be all you need!

Good Luck and I hope you enjoy it!
Jim Rice

This article comes to us from Penn Ohio users group, 1/85 issue.

FOR BEGINERS ARRAYS

The following is a list of past "FOR BEGINNERS" columns for those new to the group: Cassette recorder use, periphreal devices, word processing, cassette files, random numbers, finding errors in programs, string functions and artificial intelligence, user proofing programs, ASCII codes, and last issue I talked about Data statements. Please let me know what you need help with. I am running out of ideas. I am also trying to correct a tendency to get too advanced. In talking with members it seems most are still struggling with the basics. My feeling is once you get past this most programers learn best by programing and going back to the reference manuals. The articles mentioned above are available on request.

Well, I've encountered more than one individual with terminal confusion over arrays, and it is one of the few weak points in the reference guide as far as clarity of explanations go. I expect that by the time I finish writing this my sympathy for the authors of the manual will increase.

First, conceptually, what is an array? It is simply a list of items, each one with an address, or number. This is a one dimensional array. I would consider two or three dimensional arrays as poison until you get comfortable with one dimensional arrays. Actually they are simple, but I managed to get my understanding all gummed up, and since I am at least average in intelligence, others will likely have similar trouble.

Now that you understand that it is a list, you need to clearly understand that two different types of items can make up an array. That is numbers and strings. Remember a string is not for kites, but is what we call a group of characters.

A quick review of strings, as many folks get confused here, and is the source of many a program error. A word is a string. It is a sequence of letters. Also a sentence can be a string, a sequences of letters and spaces. Even the characters representing numbers can be in a string. You can even have a string where all the characters are numbers.

Because the computer handles numbers totally differently from strings you must always keep track of whether a sequence of characters, or a data item is a number or a string. Most of the commands in BASIC also include variables that are either numbers or strings. String variables always end with a "$", and functions that deal with strings can only have a string variable within them. For instance the command LEN(A$) must have a string within the parenthesis. In this case, the name of the string is A$. Note that LEN is not followed by a "$" because it returns a value that is a number. Compare this to CHR$(60). A number must be within the parenthesis, but because this command returns an answer that is a single character string, CHR is followed by a "$".

BACK TO ARRAYS. An item within an array is identified by a number that is within parenthesis following the array name. For instance to assign the position 1 a value in an array named B$ you do the following:
B$(1)="APPLE"
Note, since the array name is B$, you can only assign strings to this array. You may never assign a number to any position within this array. If the array name was B, then you could write:
B(1)=20
You pronounce this B of 1 equals twenty.
The computer lets you have arrays up to 10 in lengh. If you want a bigger array you simply tell the computer once at the beginning of the program, before the array name is ever used. You use the following command:
DIM B(100)
Where the number within the parenthese gives you the maximum size of the array you want. This is called dimensioning an array. In the users reference guide they give the the mathameticians delight, the general situation, which always confuses. To make the manual clear, there are one, two and three dimensional arrays. To dimension a two dimensional array you would give the command:
DIM C(100,2)
To dimension a three dimensional array you use the form:
DIM C(20,4,10)
These are specific examples of the general formula that is given in the users manual. You are not limited as to what specific integers you may use within the parenthesis, except by the computers memory. Incidently arrays seem to use up huge chuncks of memory. Apparently when you dimension an array a number of bytes are set aside for each array item, 10 or 20??? Some one out there experiment arround and let me know.

One other quirk, the computer starts numbering arrays at 0, so if you say B(10)=20 without a DIM statement, it won't work, because you have items 0-9 only. You can make the computer start counting with 1 by saying OPTION BASE 1 at the very beginning of a program, before any arrays are used or dimensioned.

Now an array member is used as a variable, either a numeric variable, or a string variable. They can take the following forms (any sort of expression can be within the parenthesis, so long as it reduces down to a number):
A(B)
A(1)
A(2*B)
A$(B)
APPLE$(4)
A(B/POS(C$,"D",2))
You use them the same way you would use any other string or numeric variable.

If you use A(X) as an array name, you can not use A as a numeric variable name in the program. Same for A$(X) and A$.

Now for an example of why one would want to use an array.

```
10 OPTION BASE 1
20 DIM A$(12)
30 FOR B=1 TO 12
40 READ A$(B)
50 NEXT B
60 INPUT "ENTER 1 TO 12":C
70 PRINT "YOU PICKED ";A$(C)
80 GO TO 60
90 DATA APPLE, ORANGE, PEACHES,
ORCHARD, BANANA, CAR, CONCANTENATION,
SILLY PUTTY
100 DATA CHAIR, DONKEY, THE LAST ITEM
```

This is a trivial example. All the program does is ask for a number input, then prints a string. Line ten tells the computer to number its arrays starting with 1 rather than 0. Line 20 Tell the computer the array is going to be bigger than the normally allowed 10 items, namely 12 items. Line 30 to 50 assigns strings to the array locations using a READ statement and DATA items. Line 60 asks which item you want, and then prints the item in the next command.

Well I hope this helps more than confuses. Once you understand these basic concepts, the users reference guide will be more understandable, and you will easily master multidimensional arrays.

One final way that you can go wrong. A1, A2, etc are not array names. They are simply variable names that happen to have a number character as part of the variable name. The power of an array comes from the fact that you can place a numeric variable name within the parenthesis, as was done in line 40 above.

Try it out, doing is the only way to learn when programing. Making errors is the best learning method, for programming anyway. -Frank Krautter

FOR LONG LIFE, KEEP YOUR FLOPPY DISKS CLEAN. In the year 2184 your great-great-great-great grandchildren could be using the same home computer floppy disks you update today - if you handle them with care Kid gloves aren't necessary, according to floppy disk experts at 3M, whose Scotch disks are guaranteed for literally millions of passes per track. The company estimates that if if you updated each track on a disk once an hour, they'd be going strong for some 200 years. But careless hands could cut a disk's life to a few short minutes. Follow these tips to keep your precious programs safe:

1. A magnetic field won't damage your disk, but it can erase your precious data. Be on the safe side - keep disks several feet away from monitors, television sets and even telephones.

2. Do what the pros do - make your home computer area a "CLEAN ROOM" with no soft drinks, no cigarette ashes, no chemical fumes - and that includes furniture polish!

3. Avoid extreme temperatures. Keep disks out of the sun, away from radiators and no where near a frosty window.

4. Please don't squeeze! Store disks properly in hard protective cases, don't write on the labels on disks inside their sleeves, and never put anything heavy on top.

5. Just in case, 3M advises you to always make at least one back-up copy of important material. Sad but true, the first hint you get that a disk is going is often a lost file!

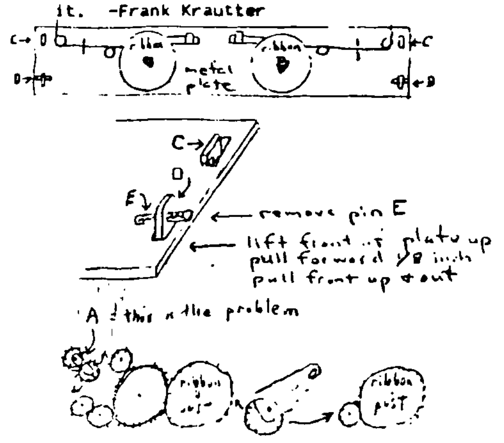
TAPE SYSTEMS

1. Follow the same rules!

This article was handed to me at the last meeting and I don't remember who gave it to me, so to help give credit where it's due please put your name on any article you submit to the newsletter.

REPAIR NOTES GEMINI PRINTER

Twice now, my ribbon has stopped turning while using the Gemini printer. This is after several years of trouble free heavey use. The problem seems to be that the paired gears on a lever, labeled A below gets stuck. All you need to do is to move it a bit and it is fixed. Once you know where it is you can likely repair it without taking the metal plate off that covers the gears. First time though you will likely need to remove the plate to see where it is. This is for the mechanically inclined, but it saved me having to send it off to the factory. First remove the ribbon. Then get the print head to the center of the printer so that it will be out of the way of posts F when you pivot the metal plate up and out. Do this while shutting the printer off while the head is moving, trial an error will get it out of the way. Then remove pin E with pliers. Because post C has a slot as diagramed the whole sheet of metal needs to be lifted up in the front so that it clears post D, then slide it forward about 1/4 or 1/8 of an inch to disengage it from post C, then the sheet is pivoted up on its back edge and lifted out. You will then see the gears as drawn below. An elegantly simple arrangement actually, though it looks a bit complicated. If you are not comfortable with mechanical things you may want to let someone else do it. -Frank Krautter



This article comes to us from Micro Report, 1/85 issue

FORTH TIP

-Reprinted from the Central Iowa Users Group Newsletter

For those of you who have a one disk drive system...this program is for you! This will be put in our library soon!

It will copy the entire contents of an original disk in only 3 passes! For example, if you had a disk with 20 programs or data files on it and used TI's DISK MANAGER, (and only one disk drive) it would take you about 8 minutes to copy all of the files. With two drives, it takes about 3 minutes. Now you can do it in 2 minutes! First, type and save this program.

To operate, load TI FORTH, and when the cursor appears, type COLD. The disk drive will "kick in" momentarily. Next, insert your disk with this program on it (drive #1), and type LOAD. this will load both screens automatically. The screen prompts will give you all of the instructions to proceed.

```
SCR #30
  ( 3 PASS DISK COPIER - DOUG SMITH 301-645-1432 )     : IT ;
  : CLS 16 SYSTEM ;  : VMBW 2 SYSTEM ;  : VMBR 6 SYSTEM ;
  0 VARIABLE AREA 15360 ALLOT 0 VARIABLE PL 0 DISK_LO !
  : TX CLS 5 11 GOTOXY ;
  : M1 TX ." INSERT COPY DISK-PRESS ANY KEY " KEY DROP ;
  : M2 TX ." INSERT  MASTER - PRESS ANY KEY " KEY DROP ;
  : M3 TX ." DONE - ENTER W TO COPY ANOTHER " ;
  : PR TX ." COPIER NOW READY-PRESS ANY KEY " KEY DROP ;
  : PU PL @ 20 + PL @ 5 + DO I BLOCK AREA 2 + I PL @ 5 + - 1024
       * + 1024 CMOVE LOOP ;
  : BU PL @ 5 + PL @ DO I BLOCK UPDATE LOOP M1 FLUSH ;
  : DR PL @ 10 + PL @ 5 + DO AREA 2 + I PL @ 5 + - 1024 * + I
       BUFFER 1024 CMOVE UPDATE LOOP FLUSH PL @ 15 + PL @ 10 + DO
       AREA 2 + I PL @ 5 + - 1024 * + I BUFFER 1024 CMOVE UPDATE
       LOOP FLUSH PL @ 20 + PL @ 15 + DO AREA 2 + I PL @ 5 + -
       1024 * + I BUFFER 1024 CMOVE UPDATE LOOP FLUSH ;    -->

SCR #31
  ( 3 PASS COPIER CONTINUED )
  : BPU PL @ 28 + PL @ 20 + DO I BLOCK 5120 I PL @ - 20 - 1024
       * + 1024 VMBW LOOP
       PL @ 28 + BLOCK 3072 1024 VMBW
       PL @ 29 + BLOCK 1122 1024 VMBW ;
  : BDR PL @ 25 + PL @ 20 + DO 5120 I PL @ - 20 - 1024 * + I
       BUFFER 1024 VMBR  UPDATE LOOP FLUSH
       PL @ 28 + PL @ 25 + DO 5120 I PL @ - 20 - 1024 * + I
       BUFFER 1024 VMBR  UPDATE LOOP
       3072 PL @ 28 + BUFFER 1024 VMBR UPDATE
       1122 PL @ 29 + BUFFER 1024 VMBR UPDATE FLUSH ;
  : PAS BPU PU BU BDR DR ;
  : CY  0 PL ! 30 / 1+ 0 DO PAS PL @ 50 > IF M3 ELSE M2 I 2+ .
       30 PL +! THEN LOOP ;
  : W  M2 89 CY ;
  PR W
```

This article comes to us from HUG, 1/85 issue.

--NEW FORTH--

LOADS FROM EXTENDED BASIC AND OTHER CARTRIGES!

by Bill Jones

Most HUGgers know Greg Goodwin as something of a patriarch to the Hoosier Users Group's FORTH interest group. He is also a professional programmer who spends his days writing assembly programs on the 'big boys' and comes home at night to relax by writing programs in FORTH on his TI. His popular KIBBIT graphics program in FORTH is well known among us and his programs frequently appear on the bulletin board.

Greg didn't stop there though. He wasn't satisfied with programming in a language that only people with the Editor/Assembler (or the new CorComp disk controller) could use. He picked the FORTH source code apart and came up with the modifications that made it possible to load the language through several TI cartriges. Greg has been able to make FORTH load from Extended BASIC and TI Writer. Now anyone with a disk drive and memory expansion can use FORTH even if they only have one of these cartriges!

I'm told that two different modified versions are used, one that loads from Corcomp, Minimem and the Editor/Assembler, and one that loads from the Extended BASIC and TI-Writer cartriges. With Extended BASIC, loading is done the same way as any other assembly language program file, then you CALL LINK and you're in FORTH! Imagine that as an autoload program. Greg tricked TI-Writer into thinking that FORTH is one of its utility programs.

Since it would be nice to combine BASIC and FORTH programs on our bulletin board, I asked if it would be possible to switch back and forth between the two languages. He said although he hadn't tried it yet, a routine similar to FORTH's MON word could possibly cold-start Extended BASIC. One hitch to that is that both BASIC and FORTH expect to load from disk 1.

Greg decided to release this version of FORTH to the users groups, and says that the Hoosier Users Group may send one copy of the disk to each group that asks, for $5. He cautions that he is not giving it over for public domain, but sends it to each group along with a limited license to produce copies for members of that group. It contains the FORTH that TI released to public domain with some exceptions. It loads with other cartriges and it has a fix for a bug in the graphics mode that TI never fixed. He also added routines to allow speech and sound.

This new addition to our library will open FORTH up to a second generation of TI FORTH users who, I'm sure, will find FORTH as fascinating and useful as I and many other FORTH enthusiasts already have. Thanks Greg!

This disk is, of course, free for the copying to active HUG members; other clubs wanting a copy may send $5 to the Hoosier Users Group at the address below.

HOOSIER USERS GROUP DISK
PO BOX 34334
INDIANAPOLIS, IN 46234

SUMMIT 99ers USERS GROUP
P.O.Box 3201
Cuyahoga Falls, Ohio 44223

USA 20c

AKRON, OH
FEB
16
PM
1985
443