TIsHUG NEWS DIGEST

Focusing on the TI99/4A Home Computer

Volume 11, Number 8 September, 1992

Registered by Australia Post - Publication No. NBH5933

New Expansion System

For Cassette Based TI99/4A Users

Sydney, New South Wales, Australia $3

TIsHUG (Australia) Ltd
A.C.N. 003 374 383

TIsHUG News Digest

September 1992

All correspondence to:

P.O. Box 1089
Strawberry Hills, NSW 2012
Australia

The Board

Co-ordinator
Dick Warburton (02) 918 8132
Secretary
Terry Phillips (02) 797 6313
Treasurer
Geoff Trott (042) 29 6629
Directors
Rolf Schreiber (042) 85 5519
Russell Welham (043) 92 4000

Sub-committees

News Digest Editor
Bob Relyea (046) 57 1253
BBS Sysop
Ross Mudie (02) 456 2122
BBS telephone number (02) 456 4606
Merchandising
Percy Harrison (02) 808 3181
Publications Library
Russell Welham (043) 92 4000
Software library
Rolf Schreiber (042) 85 5519
Technical co-ordinator
Geoff Trott (042) 29 6629
TI-Faire co-ordinator
Dick Warburton (02) 918 8132

Regional Group Contacts

Central Coast
Russell Welham (043) 92 4000
Coffs Harbour
Kevin Cox (066) 53 2649
Glebe
Mike Slattery (02) 692 8162
Hunter Valley
Geoff Phillips (049) 42 8176
Illawarra
Geoff Trott (042) 29 6629
Liverpool
Larry Saunders (02) 644 7377
Northern Suburbs
Dennis Norman (02) 452 3920
Sutherland
Peter Young (02) 528 8775

Membership and Subscriptions

Annual Family Dues	$35.00
Associate membership	$10.00
Overseas Airmail Dues	A$65.00
Overseas Surface Mail Dues	A$50.00

TIsHUG Sydney Meeting

The September Meeting will be a full day tutorial starting at 10.00 am on 5th of September at Ryde Infant School, Tucker Street, Ryde.

Printed by
The University of Wollongong
Printery Services.

TIsHUG News Digest **ISSN 0819-1984**

Index

Title	Description	Author	Page No.
Algorithm design	Software hints	Christensen,Garry	9
Beginning Forth #18	Software hints	Raguse,Earl	20
Cadet console expansion	Hardware review	Christensen,Col	7
Clearinghouse BBS	General interest	Peterson,Jim	2
Communicators	BBS information	Mudie,Ross	5
Editor's comment	General interest	Relyea,Bob	1
Extended BASIC tips #20	Software hints	Swedlow,Jim	14
File protocol	Software hints	Schafer,Mark	6
Keeping track of petrol costs	Software hints	Takach,Ben	15
Newsletter update	General interest	Relyea,Bob	22
Regional group reports	General interest		23
Sprite tutorial	Software hints	McCormick,Mack	19
Techo time	MiniPE systems	Trott,Geoff	5
TI-Base tutorial #19	Data base	Smoley,Martin	17
TI-Bits #19	Software hints	Swedlow,Jim	13
Tips from the Tigercub #64	Software hints	Peterson,Jim	11
TIsHUG shop report	Club news	Harrison,Percy	4
TIsHUG software column	Club software	Schreiber,Rolf	3
Title screen for TI-Base	Data base	Gaskill,Bill	18
Treasurer's report	Club news	Trott,Geoff	5
Writing in machine code	Software hints	Banfield,J.E.	21

All articles appearing in this month's issue of the TND are available as text files on disk ready for the formatter. Newsletter editors please note that, if you wish to re-print any articles, contact us, stating which articles you are interested in and giving the date of the TND. These will be dispatched to you promptly at the cost of the media plus postage.

We have changed our postal address. From now on please use: PO Box 1089, Strawberry Hills NSW 2012.

Editor's Comment

by Bob Relyea

I have been asked by Dick to say two things this month. One is that we still could use more local contributions. Some of you have contributed a lot along the way and are still doing so. In addition to those very useful contributions we would like a bit more from the 'locals' to accompany what the others are doing. Remember, it does not have to something with a lot of depth as there are TIers at all levels in our club and anything you have to say would be of benefit to someone. Thank you. The other thing, which I have also mentioned in the Newsletter Update, is that the Melbourne group is alive and well. It does not look like they are publishing a newsletter but they are still active. That is always good news!

Clearing House BBS

by Jim Peterson, Tigercub Software, USA

At the Lima Multi-User Group Conference in 1990, the problem of dissemination of TI information was discussed. It has always been the custom for user groups to exchange newsletters, and to reprint articles from each others newsletters. With decreasing membership, it was becoming too expensive for some groups to maintain this exchange. Others were mailing them in bulk every few months, which delayed receipt of new information.

It was therefore decided to establish a Clearing House BBS, to which text articles could be uploaded and downloaded for rapid circulation. Irwin Hott, SYSOP of the Spirit of 99 BBS of the Central Ohio 99'ers, agreed to be the SYSOP, and the Central Ohio 99'ers assumed responsibility for establishing the BBS.

It was necessary to add a hard drive and other equipment to the existing BBS, in order to receive this large volume of files. To defray the cost, it was decided to charge participating user groups $30 for initial membership, and a lesser fee to defray maintenance costs in future.

The following user groups and individuals contributed- Lima 99/4A Group, Twin TIers User Group, Blue Grass 99/4 Computer Society, Tigercub Software, Atlanta 99/4A Computer Users Group, Philadelphia Area TI Users Group, Sacramento TI Modem Users Group, E. L. Edwards, Great Lakes Computer Group Inc., NEWJUG 99ers Group, Boston Computer Society TI 99/4A User Group, L-Town 99/4A User Group, S. Jean Hall, Cedar Valley TI User Group, and C.O.N.N.I. Of these, the Lima User Group contributed $200 and S. Jean Hall, C.O.N.N.I. and Tigercub Software each contributed $100.

Because of Myarc's unreliable support, we were reluctant to purchase their hard drive controller. The ESD corporation had announced a new hard drive controller to be soon available. We waited for it- and waited, and waited.

Finally in November of 1991 the Clearing House went into operation, with a MYARC HFDC loaned by Chuck Grimes. Irwin Hott, Chuck Grimes, Karl Romstedt, Ken Marshall and Dick Beery donated much time in getting the drive installed and operating, and in modifying Irwin's already highly-modified TIBBS to work with a hard drive.

Unfortunately, there were still further delays in announcing and publicizing the opening of the Clearing House, and it has still not been well publicized. However, a large number of articles have been uploaded and are available for downloading by those who have subscribed by becoming associate members of C.O.N.N.I. The Lima Users Group alone has contributed about 125 files, including Charles Good's articles about many rare and unreleased peripherals and software, Andy Frueh's software reviews, etc. The Bluegrass User Group has contributed about 15 articles by Mark Schafer, Steve Burns, and others, and recent articles from the C.O.N.N.I. newsletter are also on file.

I have uploaded all 67 of my Tips From The Tigercub, updated and edited and with obsolete advertising removed. I have also uploaded about 40 other articles I have written- Extended Basic programming tutorials, product reviews, TI world news, commentary, etc.

Additionally, I have uploaded many excellent articles from foreign newsletters which have not been widely distributed in this country. These include 26 contributions from the Brisbane User Group in Australia, written by Col and Garry Christensen, many of which would be of great interest to assembly programmers; several articles by Jan Alexandersson, from the Swedish newsletter (but written in English!) on assembly programming and the hidden commands in the PRK and Statistics modules, etc. and about 25 articles from the TI*MES of England, mostly by Stephen Shaw, on many subjects.

I have also uploaded numerous files from disks supplied to me in the past by the now- defunct Central Westchesters and by the K.C. 99ers, and another 37 files written by a prolific author, Jim Swedlow, for the User Group of Orange County newsletter.

And I have a stack of about 30 other disks full of articles which I will check, catalogue, archive and upload if I see any evidence that the board is getting enough use to justify the considerable time that it will take me to do so. All files in the Clearing House are archived to cut down on downloading time. Irwin is preparing a condensed catalogue of file descriptions for each library, which will also be archived and can be quickly downloaded for reference, rather than wasting long distance time in browsing through file descriptions.

User groups which have not joined the Clearing House are urged to consider doing so. Any individual TI user who would like access to this great collection of information is also welcome to join, for the same $30 fee.

And anyone at all is welcome to browse through the clearing House and see what we have to offer, although you will not be able to download if you have not joined us. Call the Spirit of 99 BBS at (614) 263-3412 and at the main menu enter O for other.

And finally, the board will gladly accept uploads of text files from anyone, and anyone who writes an article for a TI newsletter is urged to upload a copy to us. ○

continued from page 7

must be opened before printing takes place and the file should be closed when all printing has been completed. Some program lines to test the printer are:-

```
100 OPEN #1:"PIO"
110 FOR I=1 TO 10
120 PRINT #1:"THIS NUMBER IS";I
130 NEXT I
140 CLOSE #1
```

Connecting Peripherals

A TI Speech Synthesiser can be plugged into the I/O extension on the side of the CADET for those games and programs that support speech.

A Peripheral Expansion Box can also be connected to the I/O extension of the CADET. Files stored in RAM1 and RAM2 can then be archived to disk for permanent storage.

A Horizon RAMdisk can alternatively be attached through an interface card available separately.

When connecting a RAMdisk or the Peripheral Expansion Box to the CADET, two precautions are necessary. Firstly, as the CRU address of the CADET is fixed at >1000, any card having the same CRU address should be changed using its built-in dip switches. Secondly, the 32K memory expansion card must be removed from the PEBox. Although the 32K memory expansion in the CADET and the 32K card in the PEBox can co-exist peaceably, there will be a clash when RAM1 or RAM2 are accessed as these devices are paged in onto the normal 32K being used by the external memory expansion.

Finally

The computer can be switched off at any time while in the Editor, the Formatter, the Games Loader or the games themselves with no ill effects. BUT when using the Editor, be careful to save any document you have created onto one of the RAM devices first if it will be required for later use. ○

TIsHUG Software Column

by Rolf Schreiber

Software Releases for September

DISK A495 is the second disk in Earl Raguse's collection of miscellaneous Extended BASIC programs. The disk is SSSD and the programs on the disk run to three pages on the menu.

Disk Directory page 1

A QUIT
B PRINT ASCII CHAR
C CHANGE PRINT FONT
D SET PRINT MARGINS
E SET LINE FEED
F MAKE BAR GRAPH
G BELL TONES
H SPRITE DEMO
I TALKING TYPE
J CUSTOMIZE THE CURSOR
K UPSIDE DOWN!
L MY SECRET LOVE
M LOW SOUND TEST
N TWINKLE LITTLE STAR

Disk Directory page 2

A INVERSE VIDEO
B LINE TYPER
C MAKE LABELS
D WIPER DEMO
E PAKDEMO
F JUSTIFY DECIMALS
G WHITE ON BLUE
H THIS WONT WORK
I TAKE ME TO BALL GAME
J CATALOG DISK
K PRINT QUOTE STRINGS
L RS232 FILE TRANSMIT
M FILE TEST NOTE
N DISPLAY NOTE

Disk Directory page 3

A BASS NOTES
B LOW SOUNDS
C BICYCLE BUILT FOR 2
D SOUND TEST DEMO
E PRINT IN 2 COLUMNS
F A POCKET CANON
G HORIZON DEMO
H SCREEN WIPERS
I WAVES AND BIRDS
J SPRITE DEMO
K XB LISTING FORMAT
L PLAY FUNNY NIM
M NOISE DEMO
N WORMY STUFF

DISK A505 contains all the program and text files to accompany the series of tutorials on sorting, by Ron Brubaker, that we ran in the TND earlier this year.

A505 Diskname: SORTING Format: DSSD

Filename	Size	Type / Length	
BUBPOINT	4	PROGRAM	720
SHAHZSORT	12	PROGRAM	2797
SHPRINTS&D	13	PROGRAM	2837
SORT/1	23	DIS/VAR	80
SORT/2	25	DIS/VAR	80
SORT/3	39	DIS/VAR	80
SORT1	2	PROGRAM	214
SORT1/P1	2	DIS/VAR	163
SORT1A	2	DIS/VAR	80
SORT2	2	PROGRAM	190
SORT2/P1	3	PROGRAM	447
SORT2/P1DV	4	DIS/VAR	80
SORT2/P2	4	PROGRAM	621
SORT2/P2DV	4	DIS/VAR	80
SORT2/P3	4	PROGRAM	657
SORT2/P3DV	4	DIS/VAR	80
SORT2A	2	DIS/VAR	80
SORT3P	5	PROGRAM	780
SORT3P/DV	5	DIS/VAR	80
SORT3P1	4	PROGRAM	556
SORT3P2	4	PROGRAM	686
SORT3P2/DV	4	DIS/VAR	80
SORT4	25	DIS/VAR	80
SORT4/P1	4	PROGRAM	603
SORT4/P1DV	4	DIS/VAR	80
SORT4/P2	5	PROGRAM	989
SORT5	17	DIS/VAR	80
SORT5/P1	5	PROGRAM	956
SORT5/P1DV	6	DIS/VAR	80
SORT5/P2	6	PROGRAM	1033
SORT5/P2DV	6	DIS/VAR	80
SORT5/P3	5	PROGRAM	975
SORT5/P3DV	6	DIS/VAR	80
SORT5/P4	5	PROGRAM	1006
SORT5/P4DV	6	DIS/VAR	80
SORT6	54	DIS/VAR	80
SORT6/P1	10	PROGRAM	2158
SORT6/P1DV	11	DIS/VAR	80
SORT6/P2	11	PROGRAM	2544
SORT6/P2DV	13	DIS/VAR	80
SORT6/P3	10	PROGRAM	2210
SORT6/P3DV	11	DIS/VAR	80
SORT6/P4	12	PROGRAM	2693
SORT6/P4DV	13	DIS/VAR	80
SUPERTRACE	24	PROGRAM	5715

DISK A506 is a memory manager for the TI99/4A, with many powerful features. The program was written by Chris Winter, a programmer from Germany.

A506 Diskname: MEMORY MGR Format: SSSD

Filename	Size	Type / Length	
MM	33	PROGRAM	8192
MM-EPROG	32	PROGRAM	7708
MM-PAR	2	PROGRAM	188
MM-SOUND	11	PROGRAM	2456
MM-UPDATE	25	DIS/VAR	80
MN	33	PROGRAM	8192
MO	6	PROGRAM	1274

DISK A507 is all about implanting assembly language programs into runnable Extended BASIC programs. The disk was compiled by George Meldrum, who pioneered this procedure many years ago. Once AL (assembly language) programs have been implanted in this way, they can be loaded into the computer from either disk or cassette, using only "OLD DSK1.filename" or "OLD CS1".

A507 Diskname: IMPLANTING Format: SSSD

Filename	Size	Type / Length	
-README	13	DIS/VAR	80
BIG/SOURCE	11	DIS/VAR	80
BIGFOOT1XB	44	PROGRAM	10955
BIGFOOT2XB	43	PROGRAM	10718
CENTIPEDE	32	PROGRAM	7874
DEBUG	19	PROGRAM	4584
DEBUG/MAN	44	DIS/VAR	80
DEBUGX	67	DIS/FIX	80
SAVEMM	13	DIS/FIX	80
SOURCE	6	DIS/VAR	80

DISK A508 is called Introduction to the UCSD p-System and was written by Ron Williams of the Boston Computer Society. The disk is a hybrid, containing both text files suitable for viewing with TI-Writer, and information which is accessible by the p-Code system. Topics include the UCSD p-Code editor, p-System file management, and UCSD Pascal programming. This disk should be useful to anyone using the p-Code card on their TI99/4A, or attempting to program in Pascal.

continued on page 4

TIsHUG Shop with Percy Harrison

Hi! Well the response for coloured monitors was so overwhelming that we have now completely sold out of them. Unfortunately the source for these has now dried up and no further coloured monitors will be available through the shop so to those members who have missed out I offer our apologies. The feedback from those members who purchased monitors from us has been to confirm that they are exceedingly pleased with the quality and performance of the units and to date, touch wood, there have been no adverse complaints.

For those members who did not get along to the July meeting the bad news is that we have not had any response from Canada re the TIM/SOB Cards and so the Directors have decided to ask for a refund of the monies paid. Unless we can source an alternate supply or another equivalent 80 column card this will mean that the members who intended to use their colour monitor with the 80 column card will now have to purchase an interface card and kit to run it on their TI 99/4A. At present I have 7 only Interface Cards in stock and once these are sold the price will increase from $12.00 to $38.00 to cover the high production cost for a small run. Note this is the cost price to the club and will not return us any profit.

The Faire committee is currently organizing a TI Club T - Shirt which we hope to have available for sale at our October meeting. It is estimated that the price of these will be in the order of $12.00 to $15.00 each and will be available in three sizes: Small, Medium and Large so an early indication from members as to their requirements would be greatly appreciated in order that we can better forecast the number of each size required.

PRICE LIST.

Club Software Disks.

See Page 3 of the August TND plus following:

A475 Clubline 99 Vol 4 No.8 SSSD $2.00
A476 Clubline 99 Vol 5 No.5 SSSD $2.00
A494 XB #0 Money Money SSSD $2.00
A504 The Director SSSD $2.00
TC820 Health and the Human Body SSSD $2.00
TC860 Astronomy Disk #1 SSSD $2.00
TCC9 Tigercub Collection #9 SSSD $2.00

Hardware

AT Disk Control Card (DSDD Format) $150.00
5.25 Half Height Drive Double Sided $65.00
RS232 Card for PE Box $100.00
Modem Card (300Bd) for PE Box $60.00

Packaging and postage charges:

	Surface	Airmail
1 to 2 Disks	$1.90	1.90
3 to 9 Disks	$2.90	$3.60
10 to 20 Disks	$3.90	$4.80
TI Artist Plus	$3.00	$3.70
Display Master	$3.00	$3.70
TI Base	$3.00	$3.70
TI Sort	$3.00	$3.70
5.25 inch half-height drive (1.25 Kg)	refer to your local post office	

Bye for now.

continued from page 3

A508 Diskname: BOOKLET Format: SSSD

Filename	Size	Type / Length	
*README	4	DIS/VAR	80
NBOOT1	2	PROGRAM	238
NBOOT2	2	PROGRAM	32
NOBOOT1/S	3	DIS/VAR	80
NOBOOT2/O	3	DIS/FIX	80
NOBOOT2/S	12	DIS/VAR	80
P1	11	DIS/VAR	80
P10	23	DIS/VAR	80
P11	16	DIS/VAR	80
P12	16	DIS/VAR	80
P13	25	DIS/VAR	80
P14	12	DIS/VAR	80
P2	18	DIS/VAR	80
P3	14	DIS/VAR	80
P4	16	DIS/VAR	80
P5	22	DIS/VAR	80
P6	26	DIS/VAR	80
P7	26	DIS/VAR	80
P8	26	DIS/VAR	80
P9	15	DIS/VAR	80
PASCAL	54	DIS/FIX	128

Tigercub Releases for September 1992

TCC 10 is the tenth disk in Jim Peterson's Tigercub Collection. The following programs are menu driven out of Extended BASIC.

1 Snerk
2 Golden Squares
3 Dry Gulch
4 Plain of Jewels
5 Midnight Trail
6 Ranch War
7 Game of One or Two
8 Four in a Row
9 Golf
10 Home Runs
11 Isolation
12 JCL Squares

DISK TC-830 is a collection of physics programs, which should be of interest to high school students and teachers alike. The following programs can be selected from the menu.

1 Science Friction
2 Temp/Humidity Index
3 Windchill Factor
4 4-Stroke Engine Demo
5 Jet Engine
6 Race into Physics
7 Water Tank Simulation

DISK TC-850 is a collection of chemistry programs, which should also be of interest to high school teachers and students. The following programs can be selected from the menu.

1 Chemical Symbols
2 Table of Elements
3 Grunge on Chemistry
4 Hydrogen
5 Laboratory Calculator
6 Peptide HPLC
7 Periodic Table
8 Protein Predict
9 Chemistry Test
10 Nuclear Chemistry
11 GFW Calculator

DISK TC-890 is a collection of programs designed to make a teacher's job easier when it comes to working out class marks, and grading students at exam time. Extended BASIC is required, and allows for easy selection of any program from the menu.

1 Drill
2 Easy Grader

continued on page 23

Techo Time with Geoff Trott

MiniPE Systems

Bruce Boese bought a very neat system a few months ago. It consists of a MiniPE system built into a shallow box along with a power supply and 3.5 inch floppy disk drive which fits under the console. There is a ribbon cable connecting the I/O connector of the console to the MiniPE system. The MiniPE system consists of 32K memory, RS232, PIO and a disk controller. Unfortunately, Bruce has had a lot of trouble keeping the system running as he uses Extended BASIC all the time and it was slowly becoming harder to get Extended BASIC to start correctly. At the last meeting we had a look at it, as it was mis-behaving regularly enough to try and find out what the problem was.

The system worked well with any other cartridge plugged in, including the peripheral test module and Disk Manager 2 which allowed the disk drive to be tested. It also did not matter which Extended BASIC cartridge was used. It seemed to me that the problem had to be because of the ROM in the Extended BASIC cartridge and that it was being interfered with by some other memory in the MiniPE system. The only memory in the system which is enabled all the time is the 32K RAM. The other ROMs are only enabled when devices are used. If we took the 32K chip out (it was in a socket), Extended BASIC worked well. At this point I decided that I needed to look at the signals around the 32K memory to see if there were any not working fast enough. I thought it must be that the RAM chip was putting data on the bus at the same time as the ROM in the Extended BASIC module.

When I had the system at home, I looked at all the signals around the RAM and could not see any problems. Then I noticed that the OE(L) input into the RAM chip is connected to 0V. This means that whenever the chip is selected, the output of the RAM is put on the bus. This may be all right, but the processor puts out a signal, DBIN(H), which the processor expects to be used to put the data on the bus for it to read. That is, DBIN is a timing signal to ensure that the bus traffic is regulated correctly. Bruce's system is a bit different to most MiniPE systems in that it is connected to the console with a (short) length of ribbon cable, instead of plugging directly into the console which may make its timing more critical.

With an intermittent problem like this one, I thought it was worth adding an inverter and using DBIN(L) to ensure that the OE(L) input to the chip was asserted only when the processor determined that it should be. Fortunately, that seemed to fix the problem, at least as far as I was able to test. Extended BASIC worked every time for me and all the other disk tests worked well. I hope that Bruce also has no problem in that area. There was a bit of a hitch with the 3.5 inch drive, but that is another and a continuing problem with those cheap drives.

So, I guess if you are having intermittent problems with Extended BASIC and a MiniPE system, you may well be advised to ask to have this little modification made done to your MiniPE system. I will supply more details upon request. o

The Communicators

by Ross Mudie

Interested in the suggestion of a dinner on Saturday night of the faire? See the Secretary's suggestion in ALL NEWS file and tell Terry if you like the idea or not. I am very much in favour of the proposal. SYSOP o

Treasurer's Report

by Geoff Trott

Rolf and I have been working for the club for a number of years in various capacities. Rolf has been doing the layout of the TND since Shane gave it up. Shane did this job (mainly by himself) for six years and Rolf will have exceeded that effort by the end of this year. I think that he has done a marvellous job over that time, setting and keeping a standard which has not been approached by any other user group magazine. He and Bob Montgomery did the first year while I was away in 1987 and then Rolf and I did it on our own for a few years until Bob Relyea volunteered to join the team some years ago.

When we (from Wollongong) first started doing the job, we had the feeling that the people running the club did not trust us. Even with a few April Fool's jokes, I believe that we have always acted in a way to give the members as much enjoyment out of their magazine as we were able to. Fortunately, for the last few years, we have had very good relations with the directors, even becoming directors ourselves.

Over the last two years both Rolf and I have been finding that our work commitments have been making time to do justice to the magazine harder to find. I am also finding that I do not have time to do repairs as quickly as I would like, nor the time to write articles on software and hardware. As a result, Rolf and I have decided that we will not be doing the magazine after December 1992. We have informed our fellow directors of this decision and so they may well be asking for volunteers to take up this challenge.

I understand that Bob Relyea is still keen to do the editing, so that one, or better two, people are needed to produce the magazine. The steps that are required start by taking the printed output generated by Bob and laying it out into the basic form of the magazine. Then this layout is sent to a printer for copying and assembling. Finally, the finished magazines are put into envelopes with mailing labels attached and taken to the post office for mailing. There are about 180 magazines to be processed each month.

Income for July	$ 575.65
Payment for July	$ 780.20
Excess of expenses over income for July	$204.55

o

continued from page 19

```
START  LWPI MYREG
       CLR  @>8375        KEYBOARD DEVICE = 0. SCAN ALL.
       MOV  @VDP,R6
       LI   R0,>0400      LOAD  (BASE ADDRESS OF SPRITE DESCRIPTOR TABLE)
       LI   R1,HELI            SPRITE
       LI   R2,32                   DESCRIPTOR
       BLWP @VMBW                          TABLE
*
       LI   R0,>0300      LOAD  (BASE ADDRESS OF SPRITE ATTRIBUTE TABLE)
       LI   R1,SDATA           SPRITE
       LI   R2,6                    ATTRIBUTE
       BLWP @VMBW                          TABLE
*
       LI   R0,>0780      LOAD  (BASE ADDRESS OF SPRITE MOTION TABLE)
       LI   R1,SPEED           SPRITE
       LI   R2,4                    MOTION
       BLWP @VMBW                          TABLE
*
       LI   R1,>0100
       MOVB R1,@>837A     ONE SPRITE IN MOTION
*
LOOP   CLR  @STATUS
       BLWP @KSCAN
       MOVB @STATUS,@STATUS HAS KEY BEEN PRESSED?
       LIMI 2             ENABLE INTERRUPTS FOR AUTO MOTION
       LIMI 0             DISABLE INTERRUPTS SO VDP IS NOT AFFECTED ON READ/WR
       JEQ  LOOP
*
CHECK  INC  R6            R6 IS USED AS A COUNTER TO KEEP
       CI   R6,>01E4      TRACK OF WHICH MAGNIFICATION
       JLT  GO            LEVEL (1 TO 4) WE ARE ON.
       MOV  @VDP,R6
*
GO     MOV  R6,R0         LOAD R0 WITH DATA TO LOAD INTO VDP R1
       BLWP @VWTR         CHANGE THE VDP REGISTER
       B    @LOOP
       END
```

* For extra practice add a routine that shows the X and Y position of the SPRITE on the screen as it moves. HINT: Y location is 1st byte in sprite attribute list. X is the 2nd byte. Read them, convert to ASCII decimal and redisplay with appropriate text. Who will be first? o

File Protocol

by Mark Schafer, USA

This article is way too late. About 10 years too late. Texas Instruments should have read this article before they made TI Writer! They are using the wrong file format, and you probably are, too.

Let's look at how files are stored on disk. No, I am not going to get technical and tell you how you can use sector editors to look at and/or change things in a file. This will be a general approach. First, you should be aware that disks are divided into sectors. Catalogue programs usually tell you how many of those you have free on a disk. One whole sector is used for each file to tell the computer everything it needs to know about it before it tries to read or write to it (the header sector). But I am not going to focus on that, either. I am looking at the part of the disk on which the content of the file is stored.

Now how many file formats are there? Only counting the ones Extended Basic can manipulate on its own, there are four:

Internal/Variable, Internal/Fixed,
Display/Variable, and Display/Fixed.

With any of these file types the computer never splits a record between two sectors. If there is not enough room on the current sector to store the next record, it will start it on the next sector. This is key to my discussion here.

Starting with variable-length files, the maximum number of characters you can get in one record is 254 (253 in internal in BASIC). There are 256 characters in each sector. As a quick technical excursion, the computer uses the other two characters as a length byte and an end-of-record marker (why it needs both is beyond me). Anyway, you can optionally specify the maximum record length when you open a file (in any language; that is the beauty about this article: it applies to ALL programming languages.) TI Writer files, for instance, are in DIS/VAR 80 format, so they are limited to 80 characters per record.

Get ready because here comes one of my key points! Why limit the record length to 80 characters when you can have 254? Well, you might say it takes up less disk space. Wrong! Each record in a variable-length file takes up only as many bytes as it needs to. In other words, if you changed every DIS/VAR 80 file to DIS/VAR 254, nothing would change. It would still take up the same amount of space and take the same amount of time to load. Do not do that because TI Writer will not be able to read them.

So, you might say, why reserve space for each record you will never use? After all, TI Writer only puts 80 or fewer characters in each record; even if the format were DIS/VAR 254, the extra characters would probably never be used. The point is there is no reason not to. If they had done it this way, DIS/VAR 254 would be the standard that DIS/VAR 80 has become. Then all programs that used DIS/VAR files would be able to read each other's files, although, they may not be able to make sense out of them, but at least the option would be available.

So if you use DIS/VAR or INT/VAR, you should only use 254. The only reason not to now is to maintain compatibility with other programs. And that is unfortunate because that is a good reason. That is what I mean about being too late. Oh, well, if compatibility is not important for a particular program you are working on, then go with 254. This will give you the ability to add more characters to the record should it prove necessary later on without having to change the record length.

Yes, I am also going to attack fixed-length record files. Now the maximum is 255 characters (254 in internal in BASIC). The length byte and the end-of-record byte are not needed since all records are the same length. And since that is the case, the number of records in each sector is fixed. So in most cases, there will be unused bytes in every sector. We need to try to reduce the number of unused bytes without taking up more sectors.

Obviously, FIXED 255 does not work because you would only get 1 record per sector no matter how small. I will just give this one to you and tell you the reason afterwards. Take 256 and divide it by the number of bytes you plan to have in each record, and drop the decimal. Then take this number and divide it into 256, and drop the decimal again. This is what you should fix the length at. For those of you who are not mathematically inclined, let me give you a list of numbers: 1 through 19, 21, 23, 25, 28, 32, 36, 42, 51, 64, 85, 128, and 256. You cannot use 256 since the computer cannot represent that number in a single byte, so you have to use 255 instead. If the length you want to fix is not on the list, move up to the next higher number that is. So if you want to use INT/FIX 43, 43 is not on the list, so you would use INT/FIX 51 instead.

Let me explain. You see with 43 bytes per record, you would get 5 records per sector on the disk. That would amount to wasting 41 bytes per sector (trust me). If you use 51 instead, you still get 5 records per sector but only waste 1 byte per sector. Same argument as above against "...but I am reserving more space than I need." You will have the ability to add more characters later on without taking up more space or changing your file format. This will not work out every time; you may have to change the file format some time, but there is no reason to use 43 when 51 costs you no more disk space. The same argument applies to every number not on the list.

This is very important for the numbers between 128 and 255. If you use anything in that range, you will only get 1 record per sector. As an alternative to using 255, you might look for a way to reduce the number of characters in each record especially if you are only a little over 128. If you can get it down to 128, you will not believe the space you will save! Suddenly, you will get two records per sector taking only about half the space!

This rule also applies to variable-length record files. As a case in point, I had a program which used a variable-length record file with 64 records. I put so much space in each record, that the file took up the full 65 sectors (one more for the header sector). So I split it into two files, so I could get more records per sector. On one file I get two records per sector; I get three on the other, so the two files combined now take up only 48 sectors! If I had read this article before, they would both be INT/VAR 254 files. So remember even variable files can benefit from the number list given above. If you can lower the MAXIMUM number of characters per record in files with consistently long records, you can save serious disk space. But you would still use VARIABLE 254.

In fixed-length records, the most efficient numbers are the powers of 2 (128, 64, 32, etc.) because they use every byte of a sector. This is why archived files are in INT/FIX 128 format. It wastes no space in the fewest reads.

Whereas all the above applies to any language, in assembly language you have the option of using PROGRAM files which use every byte up to the last sector, and are therefore the most efficient (not to mention quicker load time). So in assembly language, you might want to think about this format for program-specific files. This is the way fractals are stored in FRACTAL EXPLORER.

o

CADET Console Expansion

by Col Christensen, Brisbane

The CADET is a low cost system of expanding an unexpanded, cassette only TI-99/4A into both a word processor, complete with printer port, and an assembly games machine. These are the two main functions for a computer in the home.

The CADET is guaranteed for 12 months from the date of purchase against manufacturing defects but not against physical or other misuse by the owner. Defective units must be forwarded to Colin Christensen, 17 Centaur Street, Redcliffe, Queensland 4020, phone (07)284-7783 and repairs will be effected as soon as possible.

The back-up battery should last for three years or more. It is an inexpensive lithium type commonly used in small calculators and available from electronics component stockists or from some chemists.

Getting Started

The CADET plugs into the I/O port on the right side of the TI-99/4A so that the printer port of the CADET faces the rear of the computer. The console must be switched off while connecting or disconnecting the CADET. The CADET should not be used on a console with built-in 32K memory expansion.

When the computer is switched on, the normal colour bar screen and then the selection screen show on the screen. If a module has been inserted in the module port, the selection screen will give more than one choice. e.g.

1. TI BASIC
2. (Name of the module)

The inbuilt software is invoked through system CALLS typed in when in the BASIC or EXTENDED BASIC computer environment.

CALL WP

To load the Word Processor, type the command, CALL WP, then press the ENTER key. This part of the word processor is referred to as the Editor and is basically the original version of the TI-Writer editor. This version is not quite as versatile as the Funnelweb Editor used more universally today but ROM space limitations forced the use of the shorter, original TI program.

You will need a copy of the TI-Writer User's Manual as a guide to the use of the word processor or, alternatively, I have available printed copies of a tutorial on its use. I will supply on request a free copy to each purchaser of a CADET or they can be purchased at $5 per copy posted.

By using the TI-Writer command SaveFile (SF), text files created can be saved to two device names on the CADET. The larger, RAM1, has a capacity of 24K bytes. It will store up to 5 full pages of close typed text, being about the full capacity of the TI-Writer text buffer. The devicename, RAM2, has 8K bytes of memory, sufficient for the average sized letter.

Both RAM1 and RAM2 devices are housed in a battery backed memory chip and their contents remain intact even while the computer is switched off or the CADET remains idle. Files can be loaded into the computer memory text buffer from either of the devicenames using the TI-Writer command LoadFile (LF) and entering the appropriate devicename. A text file stored on RAM1 or RAM2 cannot be deleted in the normal TI-Writer way. If a new text file is saved to that devicename, the old flags and pointers are overwritten and the old file will not be accessible any more.

CALL FO

For the Text Formatter, type the command, CALL FO then press the ENTER key. This call loads from the ROM (Read Only Memory) a copy of the original TI text formatter with a few modifications to effect improvement. One is the elimination of the page form feed at the beginning of printing and the other reduces the blank lines printed at the top of each page from three to one.

The formatter program is also loaded automatically when exiting from the Editor. Pressing "Q" to quit, then "E" to exit, will bring up the Formatter option screen. The QUIT key (FCTN[=]) which was disabled when in the Editor is now active and can cause a computer reset if pressed a few times. The only way of QUITting from the Editor is through the Formatter.

CALL GL

For the Games Loader, type the command, CALL GL from one of the BASIC environments then press the ENTER key. A few screen prompts serve as a guide to use of the tape recorder. Each games program is stored on tape as a group of files consisting of from one to five files. If a number of programs have been recorded sequentially on tape, the beginning of each program is indicted by a period of silence. There is no period of silence between each file in a program.

For those with disk drives, the disk supplied with the CADET contains this document as a text file and also the program, TAPEITSAV. This program allows you to save assembly games to tape in a form suitable for use by the Games Loader. Only assembly programs that normally run in the Editor/Assembler option 5 environment or Funnelweb option 3 (RUN PROGRAM) are suitable. Some of these programs are copyright and some may be freeware. Use them only if you have paid for the original copyright programs or, out of conscience, have contributed to the freeware author.

Printing

In BASIC or Extended BASIC, output can be directed to the printer port using normal programming code. Only DISPLAY/VARIABLE type records can be sent to the printer, though, which covers 99% of normal needs. This includes listing programs in one of the BASICS to the printer.

The printer port responds to the print devicenames of PIO or PIO.LF or PIO.CR. It is capable of data output only, no provision being made for data input. From the Editor, the PrintFile command should use the devicename PIO while the Formatter needs to use in almost all cases the devicename PIO.LF. If this name does not drive the printer correctly, try PIO.CR.

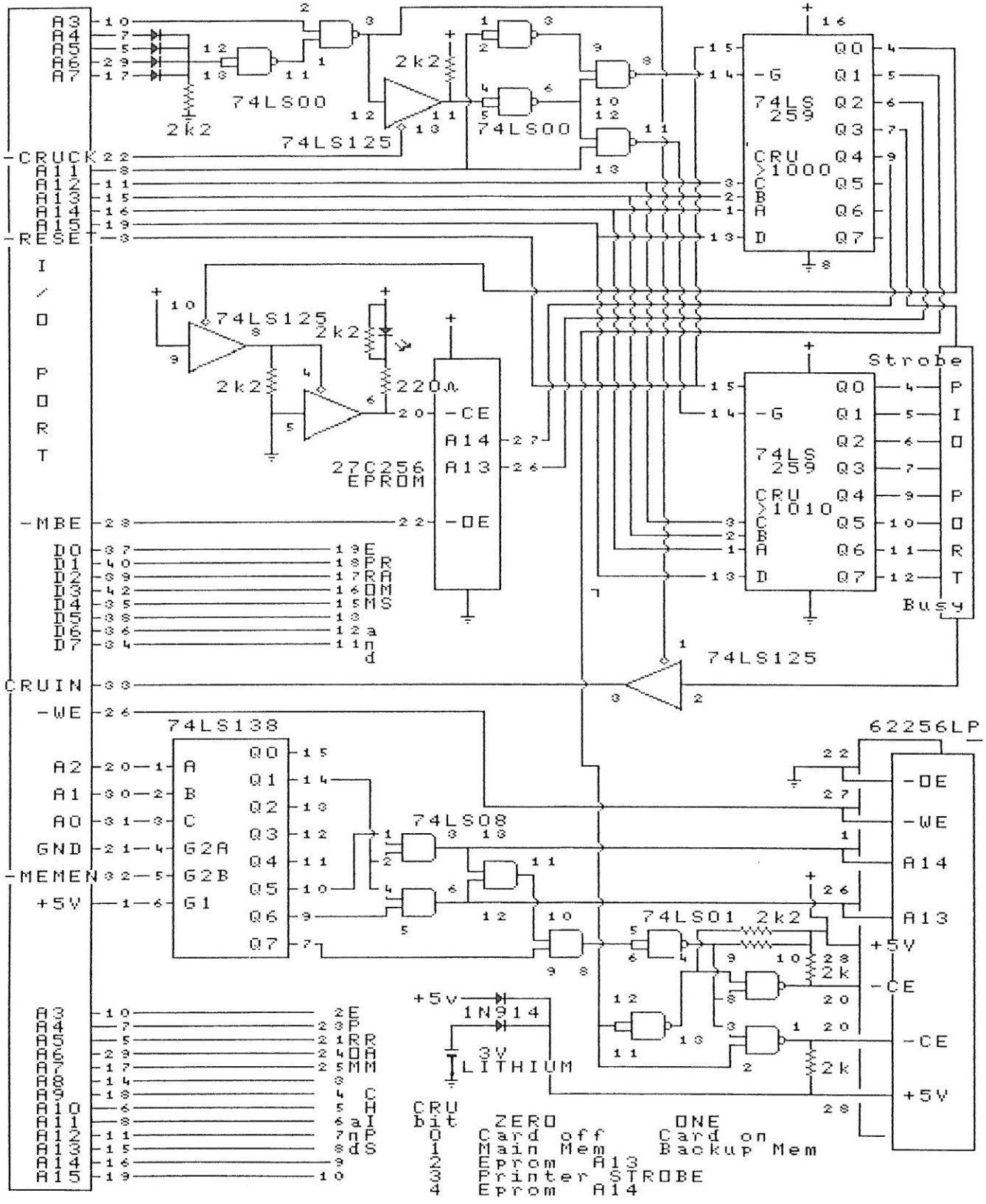
The printer cable is usually made of a 16 way flat ribbon up to about one and a half metres long with the following plug connections at either end. The CADET PIO port needs a 16 pin IDC plug and the printer requires a 36 pin Centronics connector.

CADET		Printer
1	Strobe OUT	1
2	Data	2
3	Data	3
4	Data	4
5	Data	5
6	Data	6
7	Data	7
8	Data	8
9	Data	9
10	Busy IN	11
11	GND	20
12	+5v through 10 ohms	
13	Spare	
14	Spare	
15	+5v through 1K ohms	
16	Gnd	20

To print from BASIC or Extended BASIC, normal file programming commands are used. A file to the printer

continued on page 2

CADET Console Expansion Schematic



Algorithm Design

by Garry Christensen, Brisbane

We will start with a definition:

ALGORITHM - a systematic procedure for solving a problem or accomplishing some end.

Programming some tasks on the computer is easy. Many short programmes can be written quickly and with little fore-thought. This is not always the case. The more complicated the problem, the more involved the solution. There comes a stage when the programmer must sit down and work out both how he will approach the problem and how the programme will work.

An ALGORITHM is usually the result of this work. The algorithm is probably the most important part of any programme yet it is so often ignored.

Once the algorithm has been written, the programme is much easier to write and frequently will have fewer bugs. That does not mean that algorithms only apply to computing. An algorithm can apply to any situation where a result is required. Consider the algorithm for finding the partner of your dreams using a dating service:

```
Send money.
Wait for reply from dating service.
Contact date.
Go out with selected partner.
Repeat until suitable partner is found.
```

This process is far from perfect as you may go broke before getting married but the algorithm still exists.

The first part of algorithm design is the analysis of the problem. In simple cases, the answer is obvious but sometimes it may require some thought to determine the solution. Consider also that the solution may not be unique. There may be other ways of doing the same thing. Before even touching the keyboard, try to determine that the method you have chosen is the best.

The second part is to determine the steps that will lead to the solution. This part involves a method called top-down development or step wise refinement.

Start with the steps written in very broad terms (generally using English) then refine each step into a series of more detailed steps. Each of these steps can then be further refined, and so on until the steps are close enough to the programming language to allow easy conversion. Do not start to write programme code in your algorithm until the very last step.

Let's consider the algorithm necessary to instruct a robot to make a cup of coffee. The solution is simple so lets define it in broad terms.

```
1 Boil the water.
2 Put coffee in the cup.
3 Put water in the cup.
```

These steps are obviously to complex for a robot that has no experience in the human world so they need to be defined further.

```
1 Boil the water.
   1.1  Fill the kettle.
   1.2  Plug in the power.
   1.3  Turn on the power.
   1.4  Wait until the water has boiled.
   1.5  Turn off the power.
```

Refine the steps still further.

```
1 Boil the water.
   1.1 Fill the kettle.
       1.1.1 Place kettle under tap.
       1.1.2 Turn on the tap.
       1.1.3 Wait until the kettle is full.
       1.1.4 Turn off the tap.
```

Continue through all the steps using this method until a full list of detailed instructions is produced.

As you can see, this is an easier way to end up with detailed instructions than simply starting at the top and working straight through. This method of refinement can continue until the instructions are something that the robot can perform. These are called primative operations. In computer programming, the primative operations are the statements that make up the language.

There are two types of traps that I should point out at this stage. The first is called the "first catch your rabbit bug". We have so far assumed that there is a kettle for the robot to use and that there is a tap available. Nowhere have we told the robot to "find a kettle" or "go to the tap".

The second trap is called the "initial state of things bug". What will the robot do if the water is already turned on. When told to "turn on the tap" it may screw it out of the wall.

The secret to writing an algorithm is to keep the following in mind:

1. Start at high levels (use English to express very broad steps).
2. Concentrate on refining one step at a time.
3. Do not start writing programme code until the steps are approaching the level of the programming statements.

Let's now work through a couple of examples.

I will use the selection of all the prime numbers between 1 and 100 as an example. (A prime number cannot be evenly divided by any number other than 1 and itself, eg 7). Problem: How do I determine whether a number is prime.

The simplest answer is to divide it by all the numbers less than the number in question and see if any produce a whole result. That is a valid method but not very efficient.

If the number is 50, dividing it by any number greater than 25 cannot produce a whole result (50/26 = 1.something). If the number is called N, we need only divide by all the numbers up to N/2.

To test all the numbers from 1 to 100, we need to test them all in this method. The algorithm development follows:

```
1 Set N to 1
2 Is it prime?
3 If so then print it.
4 Add 1 to N
5 Go to step 2 if N is not greater that 100.
```

Steps 1, 4, and 5 are very close to a basic statement so the next step of refinement will convert them to code. The others need a little work.

```
1 FOR N=1 TO 100
2 Is N prime?
   2.1 Set D to 2
   2.2 Is N/D a whole number?
   2.3 If so, go to step 4 (not prime).
   2.4 If not, add 1 to D.
   2.5 If D is less that half of N, then go to
       step 2.2.
3 PRINT N (this step is jumped over if not prime).
4 NEXT N
```

Further refinement is not needed.

```
1 FOR N=1 TO 100
2 Is N prime?
```

```
  2.1 FOR D=2 TO N/2.
  2.2 IF N/D=INT(N/D) THEN step 4
  2.4 NEXT D
3 PRINT N
4 NEXT N
```

The programme:

```
100 FOR N=1 TO 100
110 FOR D=2 TO N/2
120 IF N/D=INT(N/D) THEN 150
130 NEXT D
140 PRINT N
150 NEXT N
```

While this algorithm demonstrates the best way to build up an algorithm, it falls short in one point. The approach to the problem is not necessarily the best. Stop reading here and try to find a more efficient way to find prime numbers.

Did you think of one? Follow the next algorithm through. You will notice that I have used indenting to separate the levels of detail. If you want only a brief outline, read the parts of the algorithm that are not indented. The further indented, the more detailed.

```
REM  This method sets up an array of flags for each
number between 1 and 100.
   1 Initialize the array
      1.1 Dimension array
      1.2 Set array pointer to 1
      1.3 Set array element to 0
      1.4 Increment the pointer
      1.5 Repeat until pointer in greater than 100
   2 Remove the non-prime numbers
      2.1 Set N to 2
      2.2 Remove all multiples of N up to 100
          2.2.1 Set pointer to value of N.
          2.2.2 Check that this value has not been
                removed by a previous pass
          2.2.3 If it has then go to step 2.4
          2.2.4 Add N to value of pointer.
          2.2.5 Set array element to 1.
          2.2.6 Go to 2.2.4 and repeat until pointer
                is greater than 100.
      2.3 Increment N.
      2.4 Repeat until N is greater that 50 (100/2)
   3 Print the remaining numbers.
      3.1 Set pointer to 1
      3.2 Does the flag indicate that the number is
          prime?
      3.3 If not them go to step 3.5
      3.4 Print the value of the pointer.
      3.5 Increment the pointer.
      3.6 Repeat until end of array.
```

The programme will look like this.

```
100  DIM FLAGS(100)
110  FOR I=1 TO 100
120  FLAGS(I)=0
130  NEXT I
140  FOR N=2 TO 50
150  IF FLAGS(N)=1 THEN 190
160  FOR I=2*N TO 100 STEP N
170  FLAGS(I)=1
180  NEXT I
190  NEXT N
200  FOR I=1 TO 100
210  IF FLAGS(I)=1 THEN 240
220  PRINT I
230  NEXT I
```

As you can see, the algorithm is easy to follow and you will have to believe me when I say that it made the programming much easier. I tried it first without an algorithm and the result was both longer and less efficient.

It seems that the computer science proverb is correct: 'The sooner you start coding, the longer it will take to write a programme that is correct'.

To give you a bit of practice at algorithm design, I have included several problems. Write algorithms for the following problems. One way to check the answer is to continue on and write the programme. You might try to write the programme first, then solve it using step-wise refinement. Compare the answers. Are they the same?

1. Write an algorithm for placing a phone call.
2. Include in the algorithm for question 1 the possibility of busy signal, no dial tone and no answer.
3. Write an algorithm to test if a string is a palindrome. (A palindrome is a word or sentence that reads the same forward as backward, ignoring the spaces. eg RADAR or A MAN A PLAN A CANAL PANAMA)
4. Devise an algorithm for a programme that will ask for numbers to be input until the value 0 is entered then print the largest number that was input.
5. Write an algorithm for a programme that will ask for a input of a single positive number then print the number with the order of the digits reversed. eg. 13542 results in 24531. O

continued from page 20

```
CR #60
 0 (FORTH TALK TEST 12/1/86 E RAGUSE ) HEX
 1 : TALK 4 0 DO 4 SRC DUP F000 AND 4 SRL 4000 + 9400 ! LOOP
 2   4000 9400 ! 5000 9400 ! DROP ; : T TALK ;
 3 : SRT 351A T 1 WAIT 7875 T 556E T 767D T 6489 T 1A42 T 56B3 T ;
 4 : NR 3793 T 1830 T 4AAB T 7D99 T 56B3 T ;
 5 : WN 7875 T 7A11 T 71BE T 1A42 T 57C1 T 26CB T ;
 6 : DK 3793 T 2480 T 4AAB T 3C4F T 3850 T 3793 T 1CD9 T
 7   2480 T 3A7A T ;
 8 : IM 3793 T 1830 T 61C6 T 67B6 T 71BE T 556E T 4AAB T 7717 T ;
 9 : GS 3793 T 327E T 3850 T 3793 T 473D T 3793 T 473D T ;
10 : HG 6153 T 34E5 T 767D T 2FFC T ;
11 : TI 6696 T 2D19 T 3A32 T 56B3 T ;
12 : TK SRT 2 WAIT NR 2 WAIT WN 2 WAIT DK 2 WAIT IM 2 WAIT
13   GS 2 WAIT HG 3 WAIT TI 1 WAIT CLS 10 8 AT ." CHARGE!!!" CH ;
14 : TLK CLS 6 9 AT ." ENTER <TALK> TO HEAR ME TALK" ;
15 : TALK TK ; DEC TLK QUIT
```

Until next time

HAPPY FORTH TALKING. O

continued from page 14

For those of you with 32K and a disk drive, Multiplan has more features. 99-CALC is worth getting if for no other reason than to see what can be done with the 4A's 16K of VDP RAM. 99-CALC is also much easier to master.

HOW TO ORDER: Send a disk and postage paid, self addressed return mailer to:

Phil Barnes
24631 Via San Fernando
Mission Viejo, CA 92692

Phil will send you a flippy back with some good public domain software on the flip side.

If you write Phil for information, be sure and send an self addressed, stamped envelope (SASE) if you want a reply.

Phil asks for a donation of $10 if you like the program. It is well worth it!

MISTAKES DO HAPPEN

The copy of SIDE*PRINT that I distributed at the May UGOC meeting had a BUG in line 270. For those of you who got a copy there, the error is:

```
270 GOSUB 420 :: IF E=90 THEN 240 :: IF E<
>70 AND E<>83 AND E<>84 THEN 270 :: C=-(E=
70)-2*(E=84):: IF C THEN ON ERROR 520 :: O
PEN #@:P$ :: DISPLAY AT(21,@):"Printing ";
A$(A):Y$: : ELSE CALL CLEAR :: PRINT A$(A): :
```

END NOTES

This is the last of the Extended Basic series. O

Tips from the Tigercub #64

by Jim Peterson, Tigercub Software, USA

Back in the days of David Ahl's Creative Computing magazine, when computers were too expensive for hardware hacking and had memories too small to run much of a program, the emphasis was on "recreational computing", and the British TI'ers carry on that tradition. A recent issue of their excellent TI*MES newsletter had this challenge - write a program to set up a circle of any chosen number of objects; starting at one, count them off by 10's, removing every 10th object. What are the numbers of the last two left?

This is my solution. It is not the best one, but it does show how strings can be used to perform maths.

```
100 INPUT "NUMBER?":N
110 FOR J=1 TO N :: N$=N$&CHR$(J):: NEXT J :: IF N<10
    THEN 140
120 N$=SEG$(N$,11,255)&SEG$(N$,1,9):: IF LEN(N$)>9 THEN
    120
140 FOR J=1 TO 10 :: N$=SEG$(N$,2,255)&SEG$(N$,1,1)::
    NEXT J :: N$=SEG$(N$,1,LEN(N$)-1):: IF LEN(N$)>2
    THEN S40
150 FOR J=1 TO 2 :: PRINT ASC(SEG$(N$,J,1)):: NEXT J
```

Which reminds me that I forgot to give you the answer to that short CALL SOUND puzzler in Tips #62. A CALL SOUND, even with a positive duration, will be interrupted by a BEEP.

Here is a bit of nonsense I worked up from an idea by Tim Brooks. Save this by SAVE DSK1.BUGS,MERGE . Then when you get a chance, load one of your friend's favourite programs, add this to it by MERGE DSK1.BUGS, and in the middle of the program somewhere put a line with CALL BUGS.

```
32000 !@P+
32001 SUB BUGS
32002 CALL CLEAR :: CALL CHARSET :: CALL DELSPRITE(ALL):
: CALL SOUND(225,220,0):: PRINT "*ERROR 4 IN LINE 150" :
: PRINT "*BUGS IN PROGRAM"
32003 CALL SCREEN(8):: FOR A=1 TO 500 :: NEXT A :: A$(1)
="997E3CFF3C7EBD99" :: A$(2)="DB3CBD7E3CFFBD99" :: X=1 :
: CALL CHAR(96,A$(X))
32004 RANDOMIZE :: CALL MAGNIFY(2):: FOR T=1 TO 2 :: FOR
A=1 TO 20 :: X=X+1+(X=2)*2:: CALL CHAR(96,A$(X)):: FOR
 D=1 TO 20 :: NEXT D
32005 CALL SPRITE(#A,96,2,195,240*RND,-5,0):: NEXT A ::
NEXT T :: CALL CLEAR :: CALL DELSPRITE(ALL):: SUBEND
```

Here is a puzzle game for you brainy types. I worked it up from a game by Jack Sughrue -

```
100 ! PSYCHO by Jim Peterson
110 CALL CLEAR :: RANDOMIZE :: CALL SCREEN(2):: FOR S=1
TO 12 :: CALL COLOR(S,2,16):: NEXT S :: CALL VCHAR(1,31,
31,96):: CALL KEY(3,K,S)
120 RANDOMIZE :: Y$(1)="+" :: Y$(2)="-" :: Y$(3)="x" ::
Y$(4)="/"
130 CALL VCHAR(1,3,32,672):: D$="" :: Y(0),X=INT(10*RND+
5)
140 DISPLAY AT(2,11):"PSYCHO ":" Enter P(lus), (M)inus,
   (T)imes or (D)ivided by"
150 FOR J=1 TO 4 :: Y(J)=INT(10*RND+5):: Z(J-1)=INT(4*RN
D+1)
160 IF Z(J-1)=1 THEN X=X+Y(J):: GOTO 180 ELSE IF Z(J-1)=
2 THEN X=X-Y(J):: GOTO 180 ELSE IF Z(J-1)=3 THEN X=X*Y(J
):: GOTO 180
170 IF X/Y(J)=INT(X/Y(J))THEN X=X/Y(J)ELSE Z(J-1)=INT(3*
RND+1):: GOTO 160
180 NEXT J :: R=6 :: FOR J=0 TO 3 :: DISPLAY AT(R,12):Y(
J):: R=R+2 :: NEXT J :: DISPLAY AT(R,12):Y(4)
190 DISPLAY AT(R+1,12):"____ " :: DISPLAY AT(R+3,12):X
200 FOR J=0 TO 3 :: D$=D$&STR$(Y(J))&Y$(Z(J)):: NEXT J :
: D$=D$&STR$(Y(4))&"="&STR$(X):: FOR J=1 TO 4
210 ACCEPT AT(J*2+5,12)SIZE(1)VALIDATE("PMTD"):A$ :: IF
A$="" THEN 210
220 ON POS("PMTD",A$,1)GOSUB 270,280,290,300
230 DISPLAY AT(J*2+4,12):"" :: DISPLAY AT(J*2+6,12):Y(J)
240 NEXT J
250 IF Y(4)=X THEN DISPLAY AT(19,9):"RIGHT!" :: GOTO 260
ELSE DISPLAY AT(19,9):" WRONG! OFF BY";ABS(X-Y(4)):: D
ISPLAY AT(21,3):D$
260 DISPLAY AT(23,2):"PLAY AGAIN? Y/N" :: ACCEPT AT(23,1
8)SIZE(1)VALIDATE("YN"):Q$ :: IF Q$="N" THEN CALL CLEAR
:: STOP ELSE 130
270 Y(J)=Y(J-1)+Y(J):: RETURN
280 Y(J)=Y(J-1)-Y(J):: RETURN
290 Y(J)=Y(J-1)*Y(J):: RETURN
300 Y(J)=Y(J-1)/Y(J):: RETURN
```

Someone uploaded the New Testament books of the Bible to DElphi, probably ported over from IBM files. They included a program to break them into individual verses and another to display them on screen. Neither program worked properly, so I wrote this one to do it right.

```
100 CALL CLEAR :: CALL SCREEN(16):: FOR J=1 TO 12 :: CAL
L COLOR(J,2,1):: NEXT J :: DISPLAY AT(2,8):"BIBLE READER
" !by Jim Peterson
110 DIM I$(127),L$(24)
120 DISPLAY AT(24,1):"DRIVE #?" :: ACCEPT AT(24,10)VALID
ATE(DIGIT)SIZE(1)BEEP:DN:: CALL CLEAR :: ON WARNING NEXT
130 X=0 :: OPEN #1:"DSK"&STR$(DN)&".",INPUT ,RELATIVE,IN
TERNAL :: INPUT #1:N$,A,A,A
140 INPUT #1:F$,A,B,C :: IF LEN(F$)=0 THEN 160
150 IF C=80 AND ABS(A)=2 THEN X=X+1 :: I$(X)=F$ :: DISPL
AY AT(X+(X>23),1-(X>23)):STR$(X);" ";I$(X):: GOTO
140 ELSE 140
160 DISPLAY AT(23,1):"Read file #" :: ACCEPT AT(23,12)VA
LIDATE(DIGIT):FL :: IF FL<1 OR FL>X THEN 160
170 CLOSE #1 :: OPEN #1:"DSK"&STR$(DN)&"."&I$(FL),INPUT
:: CALL CLEAR :: DISPLAY AT(3,1):"Press any key at the b
eep" :: X=0
180 LINPUT #1:M$
190 IF POS(SEG$(M$,1,5),":",1)=0 THEN 220
200 IF FLAG=0 THEN FLAG=1 ::
 GOTO 220
210 X$=M$ :: GOTO 250
220 IF T$<>"" THEN M$=T$&" "&M$ :: T$="" :: GOSUB 320 EL
SE GOSUB 320
230 IF LEN(T$)>27 THEN M$=T$ :: T$="" :: GOSUB 320 :: GO
TO 230
240 IF EOF(1)<>1 THEN 180
250 IF T$<>"" THEN X=X+1 :: L$(X)=T$ :: T$=""
260 CALL SOUND(1,500,8)
270 CALL KEY(0,K,S):: IF S=0 THEN 270
280 FOR J=1 TO X :: DISPLAY AT(9+J,1):L$(J):: NEXT J ::
FOR J=10+X TO 24 :: DISPLAY AT(J,1):"" :: NEXT J :: X=0
290 IF X$<>"" THEN M$=X$ :: X$="" :: GOSUB 320 :: GOTO
230
300 IF EOF(1)<>1 THEN 180 ELSE IF X>0 THEN 250 ELSE CLOS
E #1 :: CALL SOUND(1,500,5)
310 CALL KEY(0,K,S):: IF S=0 THEN 310 ELSE 100
320 IF LEN(M$)<29 THEN X=X+1 :: L$(X)=M$ :: RETURN
330 IF SEG$(M$,28,1)=" " THEN X=X+1 :: L$(X)=SEG$(M$,1,2
8):: T$=SEG$(M$,29,255):: RETURN
340 IF SEG$(M$,29,1)=" " THEN X=X+1 :: L$(X)=SEG$(M$,1,2
8):: T$=SEG$(M$,30,255):: RETURN
350 P=27
360 IF SEG$(M$,P,1)=" " THEN X=X+1 :: L$(X)=SEG$(M$,1,P-
1):: T$=SEG$(M$,P+1,255):: RETURN
370 P=P-1 :: IF P>1 THEN 360 ELSE X=X+1 :: L$(X)=SEG$(M$
,1,28):: T$=SEG$(M$,29,255):: RETURN
```

Files ported over from IBM lack carriage returns, which can be a problem if you want to do any editing. I think this tinygram will do a good job of adding CRs to any text file which has centered headers and indented paragraphs.

```
100 DISPLAY AT(3,4)ERASE ALL :"CARRIAGE RETURN ADDER":"":
" This tinygram program will add carriage returns to any
 text file which has centered"
110 DISPLAY AT(8,1):"headers and indented paragraphs."
120 DISPLAY AT(12,1):"Input filename?":"DSK" :: ACCEPT A
T(13,4):IF$
130 DISPLAY AT(15,1):"Output filename?":"DSK" :: ACCEPT
AT(16,4):OF$
140 OPEN #1:"DSK"&IF$,INPUT :: OPEN #2:"DSK"&OF$,OUTPUT
150 LINPUT #1:M$
160 IF M$="" THEN PRINT #2:CHR$(13):M$;ELSE IF ASC(M$)<3
3 THEN PRINT #2:CHR$(13):M$; ELSE PRINT #2:"":M$;
170 IF EOF(1)<>1 THEN 150 ELSE CLOSE #1 :: CLOSE #2
```

Note that the program does all its work in line 160!

When text files are reformatted to a shorter line length, using the Funlweb Formatter, there are often long gaps at the ends of the lines, or between words if Fill and Adjust is used, due to long words which would have been hyphenated if the text had been originally typed in the shorter length. This little program will reformat text (containing carriage returns) to any shorter length and allow you to optionally hyphenate words which do not fit at the end of a line.

```
100 CALL CLEAR :: CALL SCREEN(5):: FOR SET=0 TO 12 :: CA
LL COLOR(SET,2,16):: NEXT SET
110 CALL CLEAR
120 DISPLAY AT(12,1):"Input filename?":"DSK" :: ACCEPT A
T(13,4)BEEP:IF$ :: OPEN #1:"DSK"&IF$,INPUT
130 DISPLAY AT(15,1):"Output filename?":"DSK" :: ACCEPT
AT(16,4)BEEP:OF$ :: OPEN #2:"DSK"&OF$,OUTPUT
140 DISPLAY AT(18,1):"Reform at to what length?" :: ACCEP
T AT(18,26)SIZE(2)VALIDATE(DIGIT):R
150 IF EOF(1)THEN 270 :: CALL CLEAR :: LINPUT #1:M$ :: M
$=P$&M$ :: P$=""
160 L=LEN(M$)+(POS(M$,CHR$(13),1)<>0):: IF L<=R AND POS(
M$,CHR$(13),1)<>0 THEN PRINT#2:M$ :: GOTO 150 ELSE IF L
<R THEN P$=M$&" " :: GOTO 150
170 C$=SEG$(M$,1,R):: CALL LASTPOS(C$," ",Y)
180 IF Y<>0 THEN 190 ELSE PRINT #2:C$ :: M$=SEG$(M$,R+1,
255):: GOTO 160
190 IF R-Y<3 THEN C$=SEG$(M$,1,Y):: M$=SEG$(M$,Y+1,255):
: PRINT #2:C$ :: GOTO 160
200 X=POS(M$," ",Y+1):: IF X=0 THEN X=LEN(M$)ELSE IF X=R
+1 THEN PRINT #2:C$ :: M$=SEG$(M$,Y+2,255):: GOTO 160
210 DISPLAY AT(2,1):M$ :: DISPLAY AT(8,1):SEG$(M$,1,R)
220 DISPLAY AT(12,1):SEG$(M$,Y+1,R-Y-1)&"-"&SEG$(M$,R,X-
R+1):: Z=R-Y
230 DISPLAY AT(15,1):"Hyphenate?" :: ACCEPT AT(15,12)SIZ
E(1)VALIDATE("YNyn"):Q$ :: IF Q$="N" OR Q$="n" THEN 260
240 ACCEPT AT(18,1)SIZE(Z):H$ :: IF POS(H$,"-",1)=0 THEN
 240
250 C$=SEG$(C$,1,Y)&H$ :: M$=SEG$(M$,Y+1+LEN(H$)-1,255):
: PRINT #2:C$ :: GOTO 160
260 PRINT #2:SEG$(C$,1,Y):: M$=SEG$(M$,Y+1,255):: GOTO 1
60
270 CLOSE #1 :: CLOSE #2 :: STOP
280 SUB LASTPOS(A$,B$,Y):: X,Y=0
290 X=POS(A$,B$,X+1):: IF X>0 THEN Y=X :: GOTO 290
300 SUBEND
```

I really think that all program listings should be published in 28-column format, as my Tips from the Tigercub have always been published, because that is how they appear on screen, making it much easier to key them in accurately. However, if you absolutely MUST reformat them, I think that this program will accurately reformat to/from any length up to 79 PROVIDING that you first put a carriage return at the end of every program line.

```
100 DISPLAY AT(3,6)ERASE ALL:"PROGRAM RELISTER":"":" Wi
ll reformat a LISTed XBasic program from any linelength
to any other length."
110 DISPLAY AT(8,1):" Each program line (not file line)
must end in a carriage return."
120 DISPLAY AT(12,1):"Input filename?":"DSK" :: ACCEPT A
T(13,4):IF$ :: DISPLAY AT(15,1):"Output filename?":"DSK"
 :: ACCEPT AT(16,4):OF$
130 DISPLAY AT(18,1):"Present line length?" :: ACCEPT AT
(18,22)SIZE(2)VALIDATE(DIGIT):A
140 DISPLAY AT(20,1):"Reformat to what length?" :: ACCEP
T AT(20,26)SIZE(2)VALIDATE(DIGIT):X :: IF X=A THEN 130
150 OPEN #1:"DSK"&IF$,INPUT :: OPEN #2:"DSK"&OF$,OUTPUT
:: IF X<A THEN 230
160 IF EOF(1)THEN 270 :: LINPUT #1:M$ :: L=LEN(M$):: IF
POS(M$,CHR$(13),1)=0 THEN 180
170 IF P+L<X+1 THEN PRINT #2:M$ :: P=0 :: GOTO 160 ELSE
PRINT #2:SEG$(M$,1,X-P):SEG$(M$,X-P+1,255):: P=0 :: GOTO
 160
180 IF L<A THEN M$=M$&RPT$("",A-L):: L=A
190 IF P=0 THEN PRINT #2:M$; :: P=L :: GOTO 160
200 IF P+L<X THEN PRINT #2:M$;:: P=P+L :: GOTO 160
210 IF P+L=X THEN PRINT #2:M$ :: P=0 :: GOTO 160
220 PRINT #2:SEG$(M$,1,X-P):SEG$(M$,X-P+1,255);:: P=LEN(
SEG$(M$,X-P+1,255)):: GOTO 160
```

continued on page 18

Right Justified Text

This comes from Bill Gaskill of Grand Junction, Colorado. The program that follows was saved to disk from a working Extended Basic program after I (Bob) made one or two slight alterations. The program does not allow for any more than 28 characters to be typed in as this is the width of the screen (hence, -28 in the Accept at statement). In fact, if you type in exactly 28 characters then it will already be right-justified, so if you want the program to work like the author intended then type in less than 28 characters. Fifteen to twenty would be a good number. As stated below, you must keep pressing Enter in order to bring the program to completion otherwise, the program will just 'sit' there. You will need to press Enter less than ten times. The rest of the article was written by Bill.

Routines to manipulate text strings have appeared in MICROpendium numerous times over the years, including routines to centre text on the screen, provide word wrap simulation and others. This routine demonstrates how a text string can be right justified. Most of the programming that is included is overhead that is used to let the user see what is occurring with the string while the program is working on it. The actual right-justify coding takes place in lines 140-190.

When the programs loads you are prompted to enter a string of text to justify. Once you do so and press the Enter key the string in then divided into two strings at the point where the first blank space is found. Among other things, line 140 sets B$ to equal a blank space and then T to equal the POSition of the first blank space in the sring. line 150 then sets S to equal the same position the first time around and then uses that position to SEGment the original string into two strings. The display on your screen will show what C$, the first half of the string, looks like and then just below it, D$, which is the second part of the original string.

Below both of the string segments E$ is displayed just above the numeric ruler that is used to show you what occurs with the string that will ultimately become the right justified text. The justification process takes place one action at a time so that you can observe it. Just press Enter each time you wish to see another part of the justification process take place.

When the string has been right-justified the program will jump to the end of the process in line 220 and then display the new string. You may press Enter again to do another string or simply end the program with FCTN Quit.

```
100 CALL CLEAR :: CALL SCREEN(5):: FOR A=1 TO 14 :: CALL
    COLOR(A,16,5):: NEXT A
110 DISPLAY AT(1,5):"RIGHT JUSTIFY TEXT" :: CALL CHAR(
    126,"00FF")
120 DISPLAY AT(3,1):"ENTER A STRING TO JUSTIFY:" :: DIS
    PLAY AT(6,1):"~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
130 DISPLAY AT(18,1):RPT$("|",28):"12345678901234567890
    12345678" :: DISPLAY AT(2 1,1):RPT$("~",28)
140 ACCEPT AT(5,1)SIZE(-28):A$ :: B$=" " :: I=LEN(A$)::
    IF I=28 THEN 220 ELSE T=POS(A$,B$,1):: G=T
    :: DISPLAY AT(22,1):" "
150 S=POS(A$,B$,T):: C$=SEG$(A$,1,S):: D$=SEG$(A$,S+1,(
    28-S))
 60 DISPLAY AT(9,1):C$ :: DISPLAY AT(11,1):D$
170 IF C$="" THEN E$=D$ :: T=G+1 :: GOTO 150
 180 E$=C$&B$&D$
190 DISPLAY AT(17,1):E$ :: A$=E$ :: IF LEN(A$)=28 THEN
    220 ELSE T=S+G
200 DISPLAY AT(23,1):"PRESS ANY KEY TO CONTINUE..."
210 CALL KEY(0,X,Y):: IF Y=0 THEN 210 ELSE 150
220 DISPLAY AT(5,1):A$ :: DISPLAY AT(22,1):"TEXT IS NOW
    RIGHT JUSTIFIED.":" ":" "
230 CALL KEY(0,Y,X):: IF X=0 THEN 230 :: CALL CLEAR ::
    GOTO 120
```

TI-Bits Number 19

by Jim Swedlow, CA USA

EA DISK ERROR CODES

Some programs (like Archiver) display a number when they encounter a disk error (something like "IO ERROR #7"). The numbers by themselves are of little use. Here is what they mean:

0 UNKNOWN DEVICE - Could not find the specified drive.

1 WRITE PROTECTED - The disk is write protected.

2 BAD OPEN ATTRIBUTE - One or more OPEN options were illegal or did not match the file characteristics.

3 ILLEGAL OPERATION - The book says that this code should not be generated!

4 OUT OF SPACE - The disk is full or you are trying to open more files than are allowed (127).

5 END OF FILE - You are trying to read beyond the end of the file.

6 DEVICE ERROR - The disk is not initialised, the disk is damaged, the disk drive is broken (oh no!), the drive door is open, etc.

7 FILE ERROR - The file does not exist or you are trying to read a BASIC file as if it were data.

These are the same as the second digit in the BASIC disk error codes.

MAGIC NINES

A fun program by Jim Peterson of the TIGER CLUB.

```
100 ! MAGIC NINES     By Jim Peterson
110 CALL CLEAR
120 PRINT ::PRINT "ENTER ANY 3 DIGIT NUMBER":
"WITH 3 DIFFERENT DIGITS"
130 INPUT N :: PRINT :: IF N<>INT(N) OR N>999 OR N<100
THEN 120 ELSE N$=STR$(N)
140 IF SEG$(N$,1,1)=SEG$(N$,2,1) OR SEG$(N$,1,1)=SEG$
(N$,3,1) OR SEG$(N$,2,1)=SEG$(N$,3,1) THEN
PRINT ">USE 3 DIFFERENT DIGITS<" :: GOTO 120
150 N2$,N4$="" :: FOR J=1 TO 3 :: N2$=SEG$(N$,J,1)&N2$
 :: NEXT J
160 N2=VAL(N2$) :: N3=ABS(N-N2)
170 PRINT N$;" BACKWARDS IS ";N2$ :: PRINT
180 N3$=STR$(N3) :: IF N3<100 THEN N3$="0"&N3$
190 IF N>N2 THEN PRINT N$;" MINUS ";N2$;" IS ";N3$
ELSE PRINT N2$;" MINUS ";N$;" IS ";N3$
200 PRINT :: FOR J=1 TO 3 :: N4$=SEG$(N3$,J,1)&N4$ ::
NEXT J
210 PRINT N3$;" BACKWARDS IS ";N4$: : N3$;" PLUS  ";N4$
;" IS 1089": :
220 PRINT :"I KNEW THAT WOULD BE": : "THE ANSWER.  LIST
THE": : "PROGRAM AND SEE!"
230 !!!!!!!!!!!!!!!!!!!!
240 !   THE ANSWER IS   !
250 !       1089        !
260 !!!!!!!!!!!!!!!!!!!!
```

TI LIVES

The December, 1988 issue of PC Computing (a magazine normally dedicated to MS DOS machines) has an article entitled "Gone But Not Forgotten". There is coverage of most of the orphans (TI, Osborne, Eagle, PCjr, etc). By the time you read this, you will not be able to buy that issue, but it might be worth checking out in your library. A couple of interesting quotes:

"The 99/4A and PCjr were early experiments in the home computing market. They were not nearly as fast or powerful as other computers of their time, yet in many homes they continue to fulfill the role that visionaries once predicted for them: they have evolved from somber, cold pieces of machinery to tools that are useful and fun."

"Fans of the TI 99/4A are legion, and they form a network of users that is as lively as FOG, if less structured. TI user groups number 300 with the Chicago group the largest at nearly 550 members. Other TI organisations are small but spirited. The 14 member north New Jersey group works with other New York area computer groups to sponsor the TI Computer Fest which each year draws about 300 people to attend workshops and hear speakers."

[FOG is an international User Group for owners of CPM and MS DOS machines - Ed]

TI WRITER MARGINS

For some of my editing, I use a device called a hanging indent. It looks like this:

```
.LM3;IN-3
```

The first line is set to the left margin but subsequent lines are indented. It could be called the opposite of normal paragraph indentation.

```
.LM-3;IN0
```

To do this, you must alter the left margin (.LM) and indentation (.IN) settings. For this column, the initial formatter settings were:

```
.FI;AD;LM0;RM55;IN5
```

This tells the formatter to FIll, ADjust, set the Left Margin at 0, the Right Margin at 55 and to INdent paragraphs by 5 spaces. An .IN0 would mean not to INdent at all.

When I want to start the hanging indent, I add this command:

```
.LM +3;IN -3
```

The "+3" after ".LM" tells the Formatter to move the left margin three spaces to the RIGHT. I could have said ".LM 4" but by using the plus sign I do not need to know the previous margin setting. In the same manner, the "-3" INdent tells the formatter to set paragraph indentation three spaces to the LEFT of the left margin.

When I want to revert to the original settings, I use this:

```
.LM -3;IN +0
```

The -3 after <.LM> moves the left margin three spaces to the LEFT or back where it was before the <.LM +3> command. The <.IN0> command cancels the hanging indent (i.e., it tells the Formatter to stop indenting). The + or - in the INdent command is relative to the current left margin. If the left margin is set to '0' then a + or - is not needed.

Notice that I combine a number of formatter commands on the same line. They are separated with semi-colons. This works for most (but not all) formatter commands. The formatter commands can be written with or without a space between command and number, that is, you could write .LM7 or .LM 7 and get the same result. Enjoy o

XB tips Number 20

by Jim Swedlow, CA USA

ERROR TRAPPING

We have talked here before about making your programs 'user proof'. No matter what the user does, your program should have a defense. A while back I covered one area of vulnerability - when the user inputs something from the keyboard. This month the subject is error trapping.

Say, for example, that the program must access a disk file to run. Fred Klutz, your program's user, puts the wrong disk in the drive (or does not) put any disk in). What happens? Well, your program opens a disk file and the Disk Controller goes to the specified drive to look for the file. When it does not find it, program execution stops, an error message appears on the screen and any data held in memory is virtually lost.

There is a way around this. Two Extended Basic commands can let you decide what happens when an error occurs: ON ERROR nnn and CALL ERR().

The default condition for ON ERROR is ON ERROR STOP. This means that if an error occurs, program execution stops and an error message is displayed. The alternative is ON ERROR nnn, where 'nnn' is a line number. With this, when an error occurs, program execution transfers to the specified line number.

I do not fully understand error trapping. I can use it but I do not understand it. Once you get the hang of error trapping, try intentionally causing an error with TRACE active. You will see that the computer does not exactly go to the error instructions even though it follows them.

Here is an example of how ON ERROR works:

```
200 INPUT "File Name: ":A$
210 ON ERROR 500 :: OPEN #1,"DSK1."&A$

        Program Continues

500 ! Error Instructions
510 PRINT "Could not find DSK1.";A$
520 ON ERROR 540 :: CLOSE #1
530 ON ERROR STOP :: RETURN 200
540 RETURN 530
```

Fearless Fred inputs a bogus file name in line 200. We set the error trap in line 210. Our 4A tries to open a file in line 210 but cannot find it. Control transfers to line 500.

First we tell Fleckless Fred that the file name was bad. Then we try and close the file. The code may seem odd, but it works. Sometimes, if you do not close the file an error will occur when you re-OPEN it but closing the file will also cause an error. So we put in an ON ERROR before closing the file just in case.

You have three options with RETURN in an ON ERROR routine. RETURN by itself will send you back to the instruction that caused the error. RETURN NEXT will return you to the very next instruction. And RETURN nnn will return you to line number nnn. These RETURN's do not work with GOSUB.

ON ERROR executes like a GOSUB. You could end the error language with a GOTO but you would create a pending RETURN that eats memory just as it does if GOSUB is not followed by RETURN. It could cause a problem if you use GOSUB later in the program!

Why the ON ERROR STOP in line 530, you ask. Well, once an ON ERROR nnn is triggered by an error, error control reverts back to ON ERROR STOP. However, I never know if the CLOSE #1 will cause an error condition and I do not want the ON ERROR 540 to be active after the file is closed, so I override it just to be safe.

Back to program flow. We had an error when opening the file, we told the user, we closed the file and we returned back to asking for a file name. The process starts over. If Fleabit Fred inputs a good file name, our program can continue. Anticipating a problem, we reset the error trap in line 210.

There is another tool you can use after ON ERROR has transferred control to error trapping language. It is CALL ERR(A,B,C,D). Look it up in your Extended Basic manual. It can tell you the error type, the line number in which the error occurred and the file number associated with the error if it is an I/O error. This information can be quite valuable in deciding what to do with an error.

A couple words of caution. First, do not add error trapping language to your program until you have completely debugged it. Otherwise, other errors in the program will be very difficult to locate.

Second, your TI executes ON ERROR STOP until you give it other instructions. In our sample program above, an error before line 210 would not trigger the error trapping language in line 500. Also, the ON ERROR 500 remains in effect until an error occurs or you execute another ON ERROR statement. This means that if an error occurs any where after line 210, the error message, "Could not find . . ." will appear even if it is not appropriate.

If you have more questions, just as when all else fails, read the manual -- it does give good information about Extended Basic.

ON PRE-SCAN AND USER SUBS

The Extended Basic book says that the first CALL for ALL subprograms must be within the active pre-scan area. It also states that all SUB and SUBEND statements must be pre- scanned. In debugging a program, I found that the first CALL of a user-defined sub does not need to be pre- scanned. Only CALLs of BUILT-IN subs must be pre- scanned.

This program will run without a hitch:

```
100 !@P-
110 CALL TEST :: END
120 !@P+
130 SUB TEST :: PRINT "OK" ::
    SUBEND
```

FREEWARE REVIEW: 99-CALC

99-CALC is a spreadsheet written for the 4A. What makes this program valuable is that it will run on a bare bones system (no memory expansion and no disk drive). It requires only a cassette player and Extended Basic. A printer, disk drive, and memory expansion are optional.

Given the small size of the requirements, 99-CALC compares well with full featured programs. You can do arithmetical functions (add, divide, subtract, multiply and per-cent) and you can total and average columns and rows.

You have full screen editing, can move from one cell to an adjacent one or jump to any active cell. The spreadsheet can be printed on a 80 column printer. You can opt to print formulas.

Phil Barnes, who wrote 99-CALC has truly come up with an ingenious program. He uses every bit of memory possible. For example, you can hide numbers under column and row titles. The program is filled with nice touches. 99-CALC comes with on disk documentation (that prints 7 pages) and a sample file.

RECOMMENDATION: If you do not have memory expansion and have need of a spread sheet program, 99-CALC would be a valuable addition to your library.

continued on page 10

Keeping Track of Petrol Costs

by Ben von Takach

Politicians at times will say things on the spur of the moment which happens to be something wise and the truth. This proves the point that they are almost human. I guess you may well remark that this is a strange way to introduce a computer program. The connection is the already famous remark by which one of our otherwise lack lustre leaders will be long remembered- "There is no such thing as a free lunch".

Now, is you run this program you will be able to prove conclusively that how true this statement is. Did you know that if you live - say - in Sutherland and accept my invitation to a free lunch in Wahroonga, then by the time you pack your car in the garage the free lunch has cost you more than ten dollars? This is exactly what the program will calculate for you. The printout example of a periodical report illustrates the data produced by the program. Naturally one does not have to print it to paper if it is not needed. The results are displayed on the screen and if so desired may be saved to disk as a DV/80 file for future use.

Analysing past reports and comparing these with current costs yields interesting results. Did you know that the average price of petrol in Sydney has risen from the beginning of 1989 to July, 1992 by 39.6%(from 48 cents/litre to 67 c/l)? The rise has caused an increase in my fuel costs per kilometre from 7 cents to 10 cents, which is a 43% increase. Considering that incomes have not increased during the same period, this is one example of our rapidly decreasing standard of living.

So, if you wish to nurture your growing ulcers with some more indisputable facts, here is a program for you. It is almost like 'star wars', except you will never win. (The last statement is copyrighted. Politicians may apply for details of the licence fee to the writer.)

How does it work? Just glance through the program listing. It is all there! Finally, if you are too tired to key in the program, give me a ring and I will upload it on the BBS.

```
100 !SAVE DSK.FINANCE.FUELCOST*
110 !************************
120 !* FUEL USED & COST C. *
130 !************************
140 !BY BEN TAKACH
150 !LAST REV. 2.July.89
160 ON WARNING NEXT
170 DIM L(25),C(25)
180 CALL CLEAR :: GOSUB 890
190 CALL CLEAR :: DISPLAY AT(8,3):"FUEL COSTS &
    CONSUMPTION." :: DISPLAY AT(14,1): "Wish to see the
    instructions (y/n) ?"
200 CALL KEY(3,RR,ST):: IF ST<>1 THEN 200
210 IF RR=78 THEN 220 ELSE IF RR<>89 THEN 200 ELSE GOSUB
    700
220 CALL CLEAR :: DISPLAY AT(6,3):"FUEL COST &
    CONSUMPTION." :: DISPLAY AT(10,1):"CALCULATES 1/100
    km, M/Gal & Fuel Costs..." :: DISPLAY AT(23,1):
    "Print out required ? (y/n)"
230 FL=0 :: IF P$="" THEN P$="PIO"
240 CALL KEY(3,RV,ST):: IF ST<>1 THEN 240
250 IF RV=89 THEN DISPLAY AT(23,1):"Print Device? ";P$
    :: ACCEPT AT(23,15)SIZE(-8):P$ :: FL=1 ELSE
    IF RV=78 THEN 270 ELSE 240
260 OPEN #1:P$,OUTPUT
270 CALL CLEAR :: DISPLAY AT(1,1):"Enter start & ending
    dates of the period analysed (Max. 21 Char.
    allowed): ":DD$
280 DISPLAY AT(6,1):"Initial Odometer Reading at Full
    Tank (km) ?";R1
290 DISPLAY AT(8,1):"Final Odometer Reading at Full
    Tank (km) ?";R2
300 DISPLAY AT(11,1):"How many fillups will be entered ?
    ";RF
310 ACCEPT AT(4,1)SIZE(-21):DD$
320 ACCEPT AT(7,18)SIZE(-6)VALIDATE(DIGIT):R1
330 DISPLAY AT(22,1):"Type 0 to correct an earlier
    entered incorrect answer"
340 ACCEPT AT(9,18)SIZE(-6)VALIDATE(DIGIT):R2 :: IF R2=0
    THEN 270
350 ACCEPT AT(12,18)SIZE(-3)VALIDATE(DIGIT):RF :: IF
    RF=0 THEN 290
360 IF RF>20 THEN DISPLAY AT(22,1)BEEP:"Only 20 Fillups
    are allowed per report.  please REENTER." :: GOTO
    350
370 RX=R2-R1 :: LX=0 :: CX=0 :: RO=7 :: CALL CLEAR
380 DISPLAY AT(1,3):RPT$("-",25):: DISPLAY AT(2,3):"|
    Fillups| Litres|Cost $|" : DISPLAY AT(3,3):
    "|";RPT$("-",23);"|"
390 FOR I=1 TO RF :: DISPLAY AT(I+3,3):"|";I;".";TAB(12)
    ;"|";TAB(20);"|";TAB(27) ;"|"
400 IF I=20 THEN 420
410 NEXT I
420 FOR J=1 TO RF
430 ACCEPT AT(J+3,14)SIZE(-6)VALIDATE(NUMERIC):L(J)::
    IF L(J)=0 THEN 430
440 ACCEPT AT(J+3,21)SIZE(-5)VALIDATE(NUMERIC):C(J)::
    IF C(J)=0 THEN 440
450 LX=LX+L(J):: CX=CX+C(J):: NEXT J :: GOSUB 900
460 CA=CX/LX :: FCK=(LX/RX)*CA :: LK=(LX/RX)*100 ::
    MG=RX/LX*2.82481 :: CALL CLEAR
470 DISPLAY AT(5,1):"TOTAL Km-s driven  ";RX
480 DISPLAY AT(6,1):"TOTAL FUEL USED (l)";INT(LX*100)
    /100
490 DISPLAY AT(7,1):"TOTAL COST ($)";INT(CX*100)/100
500 DISPLAY AT(8,1):"FUEL COST (c/km)";INT(FCK*10000
    )/100
510 DISPLAY AT(9,1):"AVER.COST/l (c/l)";INT(CA*10000)/
    100
520 DISPLAY AT(10,1):"litres/100km ";INT(LK*100)/100
530 DISPLAY AT(11,1):"miles/imp.gallon ";INT(MG*100)
    /100
540 DISPLAY AT(22,1):"PUSH ANY KEY TO CONTINUE" :: CALL
    KEY(3,RV,ST):: IF ST<>1 THEN 540
550 IF FL=0 THEN 980 ELSE DISPLAY AT(22,1):"WISH A
    PRINTOUT OF THE FULL REPORT OR THE SUMMARY (f/s)?"
    :: CALL KEY(3,RV,ST):: IF ST<>1 THEN 550
560 IF RV<>70 AND RV<>83 THEN 550 ELSE IF RV=70 THEN 650
570 PRINT #1:TAB(10);"FUEL COST SUMMARY":TAB(10);RPT$
    ("-",17):: PRINT #1 :: PRINT #1:TAB(5);"For
    period: ";DD$
580 PRINT #1:TAB(5);"TOTAL Km-s COVERED ";RX
590 PRINT #1:TAB(5);"TOTAL FUEL USED (l)";INT(LX*100
    )/100
600 PRINT #1:TAB(5);"TOTAL COST ($)";INT(CX*100)/100
610 PRINT #1:TAB(5);"FUEL COST (c/km)";INT(FCK*10000
    )/100
620 PRINT #1:TAB(5);"AVER.COST/l (c/l)";INT(CA*10000)
    /100
630 PRINT #1:TAB(5);"litres/100km ";INT(LK*100)/100
640 PRINT #1:TAB(5);"miles/imp. gallon  ";INT(MG*100)/
    100 :: PRINT #1 :: PRINT 1:TAB(10);RPT$("_",5)
    :: PRINT #1 :: CLOSE #1 :: GOTO 980
650 PRINT #1: : :TAB(5);"Fill Up and Cost Details:": :
660 PRINT #1:TAB(5);"ODOMETER AT START (km)";R1 :: PRINT
    #1:TAB(5);"ODOMETER AT END (km)";R2 :: PRINT #1
670 PRINT #1:TAB(5);" Fillups Litres Cost $ "
680 FOR J=1 TO RF
690 PRINT #1:TAB(7);J;TAB(16);L(J);TAB(23);C(J):: NEXT
    J :: PRINT #1 :: GOTO 570
700 GOSUB 870
710 CALL CLEAR :: PRINT "    FUEL COST CALCULATOR" ::
    PRINT :: PRINT "The program will calculate, then
    display and optionally print the following details:"
720 PRINT TAB(5);"Total fuel used (l)":TAB(5);"Total
    cost ($)":TAB(5);"Fuel cost (c/km)":TAB(5);"Av. cost
    (c/l)":TAB(5);"Consumption (l/100km)"
730 PRINT TAB(5);"Consumption (M/Imp.Gal.)" :: PRINT
    :"The program will accept 20 fillups per report.
    This limit will aid statistical evaluation of the
    results."
740 PRINT "The results may also saved  (printed) to disk
    in DV/80 format. The Program may be ended after
    a batch is completed, or it may be rerun."
750 PRINT "You may proceed then with  the next lot of
    entries."
760 PRINT : : :: GOSUB 880 :: GOSUB 850
770 CALL CLEAR :: GOSUB 870
780 PRINT :"HOW TO PREPARE THE DATA": :"First fill up
    your car and note the odometer reading.
    This is the starting km."
```

```
790 PRINT "Subsequently note the date, the odometer
    reading, the litres tanked and the amount paid each
    time the tank is filled."
800 PRINT "You do not have to completely fill the tank
    if you do not wish to do so ,except for the last
    entry of the period.  Just enter the data as "
810 PRINT "requested by the program."
820 PRINT "The program may be modified to display other
    data, e.g. the date of each refill or comments."
830 PRINT : : :: GOSUB 880 :: GOSUB 850
840 PRINT :: RETURN
850 DISPLAY AT(23,1):"PUSH ANY KEY TO CONTINUE" :: CALL
    KEY(0,RR,ST):: IF ST<>1 THEN 850
860 RETURN
870 FOR XX=0 TO 14 :: CALL COLOR(XX,1,5):: NEXT XX ::
    RETURN
880 FOR XX=0 TO 14 :: CALL COLOR(XX,16,1):: NEXT XX ::
    RETURN
890 CALL SCREEN(5):: FOR XX=0 TO 14 :: CALL COLOR(XX,
    16,1):: NEXT XX :: RETURN
900 DISPLAY AT(24,1):"ANY CORRECTIONS (y/n)?" :: CALL
    KEY(3,RV,ST):: IF ST<>1 THEN 900
910 IF RV=78 THEN 970 ELSE IF RV<>89 THEN 900
920 DISPLAY AT(24,1):"ENTER FILLUP No." :: ACCEPT AT(24
    ,18)VALIDATE(DIGIT)SIZE(2):RO
930 LX=LX-L(RO):: CX=CX-C(RO)
940 ACCEPT AT(RO+3,14)SIZE(-6)VALIDATE(NUMERIC):L(RO)::
    IF L(RO)=0 THEN 940
950 ACCEPT AT(RO+3,21)SIZE(-5)VALIDATE(NUMERIC):C(RO)::
    IF C(RO)=0 THEN 950
960 LX=LX+L(RO):: CX=CX+C(RO):: GOTO 900
970 RETURN
980 CALL CLEAR :: DISPLAY AT(12,1):"Wish to continue
    with a new lot of entries, print report to a disk
    file or exit the program? (c/d/e)"
990 CALL KEY(3,RR,ST):: IF ST<>1 THEN 990
1000 IF RR=67 THEN R1=R2 :: GOTO 220 ELSE IF RR=69 THEN
     END ELSE IF RR=68 THEN 010 ELSE 990
1010 REM FILE PRINT ROUTINE; END WITH GOTO 910 COMMAND
1020 CALL CLEAR :: IF DV$="" THEN DV$="DSK1"
1030 DISPLAY AT(6,1):"DEVICE NAME?  ";DV$
1040 DISPLAY AT(7,1):"FILE NAME?    ";FI$
1050 ACCEPT AT(6,15)SIZE(-12):DV$
1060 ACCEPT AT(7,15)SIZE(-10):FI$
1070 DF$=DV$."."&FI$ :: OPEN #2:DF$,DISPLAY,VARIABLE
     80,APPEND
1080 PRINT #2: : :TAB(5);"Fill Up and Cost Details:": :
1090 PRINT #2:TAB(5);"ODOMETER AT START (km)";R1 ::
     PRINT #2:TAB(5);"ODOMETER AT END (km)";R2 :: PRINT
     #2
1100 PRINT #2:TAB(5);"  Fillups  Litres Cost $ "
1110 FOR J=1 TO RF
1120 PRINT #2:TAB(7);J;TAB(16);L(J);TAB(23);C(J):: NEXT
     J :: PRINT #2
1130 PRINT #2:TAB(10);"FUEL COST SUMMARY":TAB(10);RPT$
     ("-",17):: PRINT #2 :: PRINT #2:TAB(5);"For
     period: ";DD$
1140 PRINT #2:TAB(5);"TOTAL Km-s COVERED ";RX
1150 PRINT #2:TAB(5);"TOTAL FUEL USED (1)";INT(LX*100)/
     100
1160 PRINT #2:TAB(5);"TOTAL COST ($)";INT(CX*100)/100
1170 PRINT #2:TAB(5);"FUEL COST (c/km)";INT(FCK*10000)/
     100
1180 PRINT #2:TAB(5);"AVER.COST/1 (c/1)";INT(CA*10000
     )/100
1190 PRINT #2:TAB(5);"litres/100km ";INT(LK*100/100
     1200 PRINT #2:TAB(5);"miles/imp. gallon
     ";INT(MG*100)/100 :: PRINT #2 :: PRINT#2:TAB(10)
     ;RPT$("_",5):: PRINT #2 :: CLOSE #2 :: GOTO 980
```

Here is a sample printout of a report that was mentioned in the earlier part of the article. The above program allows you to save your data to disk as well as to get a printout.

Fill Up and Cost Details:

ODOMETER AT START (km) 41376
ODOMETER AT END (km) 47735

Fillups	Litres	Cost $
1	63	40.89
2	63	41.9
3	61.69	39.42
4	56	35.78
5	20.26	13.51
6	50	31.95
7	61.02	39
8	63	40.9
9	60	38.95
10	56.8	41.8
11	59.23	37.6
12	59.97	40
13	62.04	42
14	30	20.97
15	63	42.78
16	60.6	40.48
17	61	41.79
18	62.41	43

FUEL COST SUMMARY

For period: 03/04/92 - 23/07/92
TOTAL Km-s COVERED 6359
TOTAL FUEL USED (1) 1013.02
TOTAL COST ($) 672.72
FUEL COST (c/km) 10.57
AVER.COST/1 (c/1) 66.4
litres/100km 15.93
miles/imp. gallon 17.73

o

Default Filing

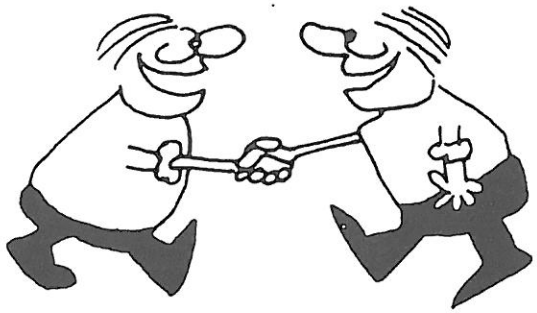
by Steve Burns, USA

As I sit here going through my disks of newsletter files wondering what this months "Editor's Desk" column will be about I am suddenly hit with a realisation. Although I had carefully planned how I was going to organize my disks for past and present newsletter articles it did not happen that way. My organisational method has again slipped into what is normally known in computer terminology as DEFAULT.

Default, as you probably know, is what happens during the course of using a program that requires you to make a decision at a certain point and you do not. This leaves the decision up to Fate (well actually, up to whatever the programmer has decided what is most likely to work, but quite often the two are close to equivalent). How does this compare to my disk organization ? Very well. Through careful planning I am able to look through at least 10 or 12 disks each time I need to find a particular article. This is roughly comparable to the method I use to organize my downloaded files for the new programs we introduce into the library.

The only difference is that I have to look through about 20-30 disks. I do have a method of sorting disks though... those with labels and those without. Nice neat method. I used to think that it was because I short on disks, or disk cases, but with those problems solved, I still find myself slipping in to the same old habits of sticking a file here and a disk there. Despite what others may think, I really DO know where everything is (uh, at least roughly). And despite all my excuses, I know that this mess is DEFAULT of de user.

o



TI-Base Tutorial #19

by Martin Smoley, USA

This months project is quite insignificant as far as innovative programming techniques are concerned, but it concerns a utility which I have wanted to create for a long time. This is the simple ability to type in the data for a mailing label and print the label and/or save the data for later use. I know this sounds untrue, but as many times as I have thought to myself that I should whip something up to do this task, I have not! So this month I am going to take care of my problem and drag you along for the ride. NOTE: There are some other reasons for this project which I will try to explain later.

I whipped up INPUT/C, which you see to the right. For those of you who are still struggling, it was not nearly as easy as I make it sound. It took forever to get those IF statement to work the way I wanted, and I kept using more LOCALs than I was allowed. I would also add that every CF I have, should be called a semi-finished product. Almost every time I use a CF I can see something I would like to change. Here are a few tricks I used in INPUT that you might find interesting. First I set up LOCALs to handle the keyboard input. This allowed me to print a label even if I did not save the data to the Db. Next, I used the data from RECord 0 as headings for the input screen. The use of RECord 0, coupled with the fact that the only direct connection between INPUT and your Db is made in the Sub-CF(INPSV) means that it should be quite easy to adapt INPUT to any Db you may be using.

```
  CREATED 04/06/90 CHANGED 04/06/90
FIELD  DESCRIPTOR  TYPE  WIDTH  DEC
  1    NM          N     004    00
  2    LN          C     015
  3    FN          C     015
  4    MI          C     002
  5    NL          N     004    00
  6    SA          C     025
  7    CT          C     020
  8    ST          C     002
  9    ZP          C     005
 10    PH          C     012
 11    XP          D     008    .
 12    ID          C     007
     000 1 TNAMES9000000/00002
```

I have modified our old standby database TNAMES to accommodate thoughts. for future projects. It is not necessary for you to make these changes. The CFs should work fine with the original Db. You can change the name of the Db used in INPUT and let it go at that. I will not place more than a few names in our DEMO Db so a conversion can be made at any time.

Database TNAMES90

REC	NM	LN	FN	MI	NL	SA	CT	ST	ZP	PH	XP	ID
0000	0	Last Name ..	First Name .	MI		Street Address	City	ST	Zip	Phone	/XP/	ID No.
0001	1	Smoley	Martin	A.		6149 Bryson Drive	Mentor	OH	44060	216-[illegible]-1661	/ /	M:44612

```
*                    INPUT/C
CLOSE ALL
 SET TALK OFF
 CLEAR
 WRITE 9,4,"           .. LOADING .. "
LOCAL IFN C 15
LOCAL IMI C 2
LOCAL ILN C 15
LOCAL ISA C 25
LOCAL ICT C 20
LOCAL IST C 2
LOCAL IZP C 5
LOCAL IPH C 12
  USE DSK1.TNAMES90
* you can use the original TNAMES
LOCAL INP C 2
  DO DSK1.INPSC
WHILE (INP<>"Q ").AND.(INP<>"q ")
 WRITE 5,1,">                <>  <";
          ">                 <"
 WRITE 9,1,">                          <"
 WRITE 13,1,">                       <";
           ">  <>      <"
 WRITE 17,1,">              <"
  READSTRING 5,2,IFN
  READSTRING 5,19,IMI
  READSTRING 5,23,ILN
  READSTRING 9,2,ISA
  READSTRING 13,2,ICT
  READSTRING 13,24,IST
  READSTRING 13,28,IZP
  READSTRING 17,2,IPH
 WRITE 22,2,"Selection ==> R          ";
"             "
 READSTRING 22,16,INP
 WRITE 22,16," ..... PROCESSING ..... "
  IF (INP="Q ").OR.(INP="q ")
    CLOSE ALL
    RETURN
  ENDIF
  IF (INP="S ").OR.(INP="s ")
    DO DSK1.INPSV
  ENDIF
  IF (INP="P ").OR.(INP="p ")
    DO DSK1.INPLBL
  ENDIF
  IF (INP="SP").OR.(INP="sp")
    DO DSK1.INPSV
    DO DSK1.INPLBL
  ENDIF
 WRITE 22,16,"                          "
 REPLACE INP WITH "R "
ENDWHILE
CLOSE ALL
SET HEADING ON
SET RECNUM ON
SET TALK ON
RETURN Copyright Martin A. Smoley 1990
```

Another possibility would be to use RECord 1 to blank the previous on-screen entries and set the length of each entry. That would allow you to create a universal input screen that would change its characteristics depending on the database that was opened. Changes could be handled by a Sub-CF that was called by a number stored in the 0 RECord and the use of a DOCASE. If this is a little too much for you, just use INPUT as a simple data entry CF and forget about the extra junk. Entering the data into the screen is more user friendly then using EDIT and I like to be able to use upper and lower case letters for commands. The CF will keep you posted as to what it is doing and it will automatically assemble an ID number for a name which is saved. The only automatic feature is the creation of NM by the INPSC CF. You will notice that it finds the last value for NM and makes the newest entry one number higher than that. This keeps the last entries at the end of the Db, no matter what their sort order should be.

At the end of an entry session you should run the CF named INP'RN. This CF will sort our Db on Last Name, First Name, renumber the NM field and then re-sort on the NM field. This is the way I like TNAMES90 sorted, you may have another idea. You should note that I have placed one blank space at the beginning of many of the RECord 0 fields. This will return that record to the head of the Db when sorting on those fields.

```
                ENTER INFORMATION                 INSTRUCTIONS
  First Name ... MI   Last Name ....
>             <> <>                 <  Enter the data between
                                         the greater than ">"
  Street Address .....                   and less than "<" signs.
>                     <                  Press ENTER after each
                                         item is entered, First
  City ..............  ST   Zip         Name <E>, Middle Initial
>                    <> <>     <        <E>, Last Name <E>, etc.
                                         When all the data is
  Phone .....      SELECTIONS            entered you can select
>           <      Q or q Quit           Save, Print, SavePrint,
                   S or s Save           Redo or Quit. When you
                   P or p Print          make a single character
                   SP or sp Save Print   entry, the character
                   R or r Redo           must be to the left and
                                         the blank space must be
 Selection ==> R                         to the right.
```

```
   renumber NM field   INP'RN/C
CLOSE ALL
LOCAL NUM N 4 0
USE DSK1.TNAMES90
SORT ON LN,FN
TOP
```

```
WHILE .NOT. (EOF)
 REPLACE NM WITH NUM
  MOVE
REPLACE NUM WITH NUM + 1
ENDWHILE
SORT ON NM
CLOSE ALL
RETURN Copyright Martin A. Smoley 1990


*                              INPSC/C
 SET HEADING OFF
 SET RECNUM OFF
 FIND 0
 WRITE 1,12,"ENTER INFORMATION"
 WRITE 3,2,FN
 WRITE 3,19,MI
 WRITE 3,23,LN
 WRITE 7,2,SA
 WRITE 11,2,CT
 WRITE 11,24,ST
 WRITE 11,28,ZP
 WRITE 15,2,PH
 WRITE 15,20,"SELECTIONS"
 WRITE 17,20,"Q or q Quit"
 WRITE 18,20,"S or s Save"
 WRITE 19,20,"P or p Print"
 WRITE 20,20,"SP or sp Save Print"
 WRITE 21,20,"R or r Redo"
RETURN Copyright Martin A. Smoley 1990

*                              INPSV/C
 WRITE 22,16," ..... SAVING DATA ..... "
LOCAL INM N 4 0
LOCAL FIN C 1
LOCAL LIN C 1
LOCAL PZ C 2
LOCAL AS C 2
BOTTOM
REPLACE INM WITH NM
APPEND BLANK
 REPLACE NM WITH INM+1
 REPLACE FN WITH IFN
 REPLACE MI WITH IMI
 REPLACE LN WITH ILN
 REPLACE SA WITH ISA
 REPLACE CT WITH ICT
 REPLACE ST WITH IST
 REPLACE ZP WITH IZP
 REPLACE PH WITH IPH
  REPLACE FIN WITH FN
  REPLACE LIN WITH LN
  REPLACE PZ WITH ZP
  REPLACE AS WITH SA
REPLACE ID WITH FIN | LIN | PZ | AS | PH
RETURN Copyright Martin A. Smoley 1990

*                              INPLBL/C
SET PAGE=000
LOCAL TEMP C 40
WRITE 22,16," ... Printing Label ..."
     PRINT (Drft),(E),(LF)
   REPLACE TEMP WITH TRIM(IFN) | " ";
           | IMI | " " | ILN
     PRINT TEMP
     PRINT ISA
   REPLACE TEMP WITH TRIM(ICT) | ", ";
           | IST | "  " | IZP
     PRINT TEMP,(LF),(LF)
RETURN Copyright Martin A. Smoley 1990
```

continued from page 12

```
230 IF EOF(1)THEN 270 :: LINPUT #1:M$
240 L=LEN(M$):: IF L+P>X THEN PRINT #2:SEG$(M$,1,X-P)::
M$=SEG$(M$,X-P+1,255):: P=0 :: GOTO 240
250 IF M$=CHR$(13)THEN 230
260 IF POS(M$,CHR$(13),1)<>0 THEN PRINT #2:M$ :: P=0 ::
GOTO 230 ELSE PRINT #2:M$;:: P=LEN(M$):: GOTO 230
270 CLOSE #1 :: CLOSE #2

      MEMORY FULL
```

Title Screen for TI-Base

by Bill Gaskill, USA

EDITOR'S NOTE: Here is an interesting article that I (Bob) have retyped from the May, 1992 issue of TIdbits (Memphis, Tennessee). I just got done doing the revision that he suggested and it worked well, first 'pop'. If you have never used a sector editor before than this might be a good chance to get started. I use Disk Patch on Funnelweb more often than not.

Here is a quick modification that I made to a BACKUP copy of the TI-Base to spruce up its title screen. It was done by copying the SCRN file from the TI-Base disk onto a freshly initialised disk and then sector editing SCRN to achieve the desired results. It is important to use a newly initialised disk and to have only the SCRN file on it.

This operation may appear to be really complex because there are so many bytes listed (in their HEX addresses), but it really is a piece of cake. Here is what it does. When TI-Base boots we all know that it displays a title screen and a time-out bar while it is loading. What this project does is modify the TI-Base name that is spelled with T's and I's and B's et cetera and changes all of them to the Texas state map outline. This same outline has shown up in custom cursor programs over the years, and I will admit that I did not care for the map outline of Texas as a cursor. However, here it is in inverse video and it really puts some life into the title screen. Honest!

Here is what you do:

1. Copy the SCRN file from the TI-Base disk onto a newly formatted one.
2. Load your favourite sector editor and go to sector >23.
3. Type in DC (the hex code for decimal 220) in the following bytes:

```
3E 3F 40 41 42 43 45 46 4C 4D 4E 54 5A 5B 5F 60 61 62 68
69 6D 6E 74 77 7B 7D 81 87 90 91 95 96 98 99 9A 9C 9D 9E
A2 A3 A4 A5 A6 AA AF B0 B8 B9 BD BE C4 C7 CA CE D3 D7 E0
E1 E5 E6 EC EF F2 F6 F9 FC
```

4. Write these changes to disk and move to sector >24. Now type in DC at bytes:

   ```
   00 09 0A 0E 0F 15 16 17 1B 1F 23 24 28 29 2A 2B
   ```

5. Save these changes to disk and you are done editing.
6. Now copy the SCRN file onto your BACKUP TI-Base disk, letting it overwrite the one already there if you wish as you do not need to erase the original one before the new one is copied. You are now done!

All that remains is for you to boot up TI-Base and see what a neat difference that subtle change makes. You can play around with the SCRN file in other ways too. For example, I have now changed mine so that it lists me as the owner of the program and it carries such things as my address, all done in inverse video characters. I also corrected the AUUGUST 28, 1990 date on the title screen so that August has only two U's instead of three. Hope you like the results but remember, do NOT do this with anything but a backup copy.

EDITOR'S NOTE: To accomplish the date modification mentioned above you simply go to sector >22 and change the screen to ASCII characters. If you are using Disk Patch this is done by pressing Fctn 2. Then move the cursor down to the 'A' of August and type right over the date the necessary correction. This means that when you are finished typing it in correctly, you are one character short of where the original date ended. Just erase the unneeded '0' and you are finished. Make sure you rewrite the sector by using Fctn 8 and pressing 'Y'. Have fun!

Sprite Tutorial

by Mack McCormick, USA

Definition: Any shape or colour. Can occupy screen positions independent of any character already present. Once set into motion, can move independently of direct program control. You can magnify or make double size.

How they can be used: Up to 32 sprites on the screen at any one time. Can be used in GRAPHICS and MULTICOLOR modes. Also can be used in BIT MAP mode but not the automatic motion feature (according to TI)
Sprites cannot be used in the TEXT mode.

There are three tables which contain all the information needed to use sprites:

1. SPRITE ATTRIBUTE TABLE
a. Sprite Position
b. Sprite Colour

2. SPRITE DESCRIPTOR TABLE
a. Sprite Pattern Identifier
b. Specify magnified or double sized sprites.

3. SPRITE MOTION TABLE
a. Define X and Y velocities of Sprites.

DEFAULT LOCATIONS OF SPRITE TABLES

Table	Table Begins at this VDP address
SPRITE ATTRIBUTE TABLE	>0300
SPRITE DESCRIPTOR TABLE	>0400
SPRITE MOTION TABLE	>0780

Sprites are numbered from 0 to 31. Here is how the screen is defined for Sprites:

Columns are labeled starting from the left from 0 to 255 (>00 to >FF). Rows are numbered from top left, the first row is numbered 256 (>100), followed by the numbers 0 to 190 (>0 to >BE). Each screen location defined in this manner is referred to as a pixel. A pixel is the smallest area of the screen you can turn on or off. Here is the way it looks:

Pixel 1 is in row >100 column >02.
Pixel 4 is in row >BE column >01.

Here are the formulas to convert row and column locations to pixel locations:

GRAPHIC TO PIXEL CONVERSIONS

GRAPHIC ROW TO PIXEL ROW	GR*8-7=PR
GRAPHIC COLUMN TO PIXEL COLUMN	GC*8-7=PC
PIXEL ROW TO GRAPHIC ROW	INT[(PR+7)/8]=GR
PIXEL COLUMN TO GRAPHIC COLUMN	INT[(PC+7)/8]=GC

SPRITE ATTRIBUTE TABLE

Begins at VDP >0300 by default. Contains the present position of sprites and their colours. Each sprite takes up four bytes in the table. The first byte is the row or Y position of the sprite. The second byte is the column or X position. The third byte references the pattern of the sprite as to where it is located in the Sprite Descriptor Table. The fourth byte is the early clock attribute and also codes for the colour of the sprite.

When the computer moves sprites it updates the information in the sprite attribute table. The more sprites it has to update the longer it takes to execute the program. To shorten this time place a value of >D0 as the Y location of the lowest numbered non-moving sprite. Always let the final unused sprite be undefined by specifying the Y location as >D0.

The third byte references a pattern in the Pattern Descriptor Table. Can range from >00 to >FF. For example if the third byte contained >80 it would point to >0400 through >0407 in the Sprite Descriptor Table.

The fourth byte controls the early clock and colour. The first four bits control the early clock. If the last bit (3) is reset to zero the early clock is off and the location of the sprite is said to be it's upper left hand corner. This means the sprite will fade in and out on the right hand side of the screen. If bit 3 is on the sprites location is shifted 32 pixels to the left. The sprite can then fade in and out from the left side of the screen.

Bits 4-7 of byte four contain the colour. Same as other VDP colours 0 to >F.

Here is an example Sprite Attribute:

```
                Sprite 0    Sprite 1

SAL   DATA   >3356,>8001,>A828,>810F,>D0
              / /   / /          (Third Sprite Undefined)
             Y X   / colour
                     pattern
```

SPRITE DESCRIPTOR TABLE

Just like the pattern descriptor table for characters. Usually begins at >0400. Addresses >0400 through >0407 are defined as sprite pattern >80.

You can also make sprites magnified or double sized by writing a value to the two least significant bits of VDP register 1.

SPRITE MOTION TABLE

Describes the X and Y velocities o each sprite. This table begins at >0780. Before a sprite can be placed into motion several conditions must be met. Your program must allow interrupts using LIMI 2 but before accessing VDP RAM you must disable interrupts with a LIMI 0. You must indicate how many sprites will be in motion by placing a value at CPU address >837A. For example if sprites 2, 5, and 7 are in motion you must place an >8 at address >837A which will allow motion of 0 through 7. A description of the motion must be placed in the Sprite Motion Table. Each sprite takes up four bytes in the table. The first byte is the Y velocity, the second byte is the X velocity. The third and fourth bytes are used by the interrupt routines, just be sure you leave space for them. The following are allowed as values for X and Y velocities:

A value of >01 will cause the sprite to move one pixel every 16 VDP interrupts. About once every 16/60 of a second.

A thought: Have you ever seen a screen dump program that would dump sprites? It could be done by obtaining their location and pattern and converting to printer bit graphics. Have fun!

```
*****************************************
*                                       *
*            CALL SPRITE                *
*     PROGRAM PLACES A HELICOPTER       *
*    SPRITE IN MOTION BY ENABLING       *
*       INTERRUPTS. PRESS ANY           *
*    KEY TO ALTER MAGNIFICATION         *
*     BY MACK MCCORMICK                 *
*****************************************
       DEF  START
       REF  VMBW,VWTR,KSCAN
*
HELI   DATA >007F,>0000,>0107,>0E0E   HELICOPTER PATTERN DESCRIPTION
       DATA >1EBE,>FFBF,>0F07,>020F   BLOCK 2
       DATA >00FF,>8080,>C0F8,>04C2          3
       DATA >DACA,>FEFC,>FBE0,>40F8          4
SDATA  DATA >7080,>8008  INITIAL SPRITE DATA
       DATA >D000        >D0 PREVENTS GHOST SPRITES
*
SPEED  DATA >0A0F,>0000  SPRITE SPEED FOR AUTO MOTION
STATUS EQU  >837C        GPL STATUS BYTE
VDP    DATA >01E0        INITIAL VALUE OF VDP REGISTER 1
MYREG  EQU  >8300        MYREG IN 16 BIT HIGH SPEED AREA OF MEMORY
*
```

continued on page 5

Beginning Forth - part 19

by Earl Raguse, UGOC, CA USA

Part of the credit for my writing this article belongs to Ron David, of fast trig functions, I trust you remember that, for arousing my curiosity to the point of investigating the SAY word on George Smythe's Forth disk which I call SuperForth because of all the lovely utilities thereon, and says "TI Forth is ready to start." This disk boots from Extended Basic, and it is in the library. I had not paid much attention to SAY because I thought it was a specialized one shot word that could only say the above sentence by making use of a lot of hex coding. Boy, was I wrong, as you will see.

Anyway, one night, Ron David showed me some Forth Assembly coding he had done to make Forth speak. He had nicely included the equivalent Assembly coding to explain each Forth Assembly code. I could easily read it but did not really comprehend how he was doing it. It encompassed three or more screens, but it worked. His word SAY could speak any of the 300 plus words in the vocabulary given in the Editor/Assembly Manual Appendix 24.6, page 422. This is the same list that is in the Extended Basic Manual without the location addresses.

This caused me to read the E/A Manual more thoroughly and to take a better look at the SAY word on SuperForth. Now that I knew the source of all the Hex numbers (vocabulary addresses), and having read the E/A Manual page 349, Section 22 SPEECH, I could now make sense of George Smyth(e)'s talking word. He uses only Forth elementary words and no Assembly Language; I will explain its rather simple operation in detail, hopefully even the most novice among you will understand it. Here is the word definition.

```
HEX
: TALK (addr---) 4 0 DO 4 SRC DUP F000
AND 4 SRL 4000 + 9400 ! LOOP 4000 9400 !
5000 9400 ! DROP ;
```

Despite the apparent simplicity of the word TALK, it really does work. I will summarise here what is said in the E/A Manual but I advise you to read at least pages 349 and 352 for yourself. If you are not Assembly oriented, the programs will not mean much, and you can ignore all the discussion on timing. Apparently Forth is slow enough that the extra microsecond delays required by Assembly are not required by Forth.

Essentially, what the E/A Manual says, is that you need to write an address to the Speech Synthesizer Module via SPCHWT (addr >9400) and then >40 to indicate the end of the address, then >50 to cause execution. Loading of the speech address (from Appendix 24.6) must be done digit-wise, least significant digit first. (they say nybble, but a nybble is half a byte, or 4 binary bits, which is what it takes to define one Hex digit). Now that I have impressed you with my knowledge of bytes and nybbles I will never bring it up again, a digit or a character is the same as a nybble. The manual also states that the digit "x" must first be prefixed by >4, as in >4x.

Unfortunately, it is necessary for me to emphasise that a Hex number(digit) is processed in the computer as four binary bits, so is a decimal digit for that matter.

Referring to the definition of TALK, the word SRC (Shift Right Circular) causes shifting of the number on the top of the stack in bitwise fashion. Four bit shifts is equivalent to shifting one digit. Shifting by amounts other than groups of four will give strange results if you are not thoroughly binary oriented, and I will not discuss that. Circular shifting means that bits shifted out at right are inserted at the left so no bits are lost. Hence 4 SRC just moves the least significant digit (LSD) to the left end of the number, that is 1234 becomes 4123. The DUP word simply saves this result for further shifting on the next pass through the loop. SRL (Shift Right Logical) does essentially the same thing except that bits shifted out on the right fall into the bitpit and are never heard from again. The vacated bits on the left are not replaced with zeros, so 4123 becomes 412 after SRL four times. The 4 bits of 3 are lost.

Note that in Forth it is not necessary to prefix Hex numbers with > (as in assembly) if one specifies HEX, as in the definition above, prior to using numbers, but in this discussion, it seems safer to do so, except when citing actual parts of the definition, to insure there is no confusion as to the type numbers being referred to.

The purpose of the 4 SRC is to move the LSD to the left end of the number to be picked off by the F000 AND instruction. Recall that the AND word compares two numbers on the top of the stack bit by bit, if both bits are "1" then a "1" is put on the stack, else a "0" is put on the stack. Remember also, that each 4 digit Hex number is 16 binary bits, and (Hex) >F is equivalent to "1111". Thus when >4123 is ANDed with >F000, the result is >4000. The 4 SRL simply moves this right one place to 400. The 4000 + adds 4000 to get 4400, and now we have the digit prefixed by the required >4. The properly prefixed digit is then loaded into SPCHWT by the 9400 !

LOOP repeats this process three more times to load the other three digits of the Speech Address into >9400. Next 4000 9400 ! stores the required >40 to end this address. Then 5000 9400 ! stores the execution instruction in SPCHWT and the phrase represented by the Hex address, is spoken by the Speech Synthesizer Module.

You may properly wonder why the two zeros are at the end of all the numbers when we are specifically only required to load two digits, that is, one byte, sorry, I said that I would not use that word again. See how unreliable I am. I forgot that I might think it necessary to answer the above question. The zeros are to fill out the rest of the sixteen bit number. When the computer is reading only one byte, it reads the most significant ones and if we did not pad the number with two zeros, it would read the most significant byte as zero or possibly garbage. This is as little beyond me, if you want details ask an Assembly Language expert.

Each execution of TALK will say only one phrase. I say phrase, because some of the addresses contain two or more words to be vocalised. To avoid writing TALK so many times, I have defined the simpler word T to represent TALK.

```
: T TALK ;
```

Thus the process of making the TI 99/4A talk is just to look up the word you want said then get the Hex address and follow that address with T. Simple enough, even I can do that; see the example given as Screen #60 FORTH TALK TEST. I will not tell you what it says. That is your incentive to type and run this screen.

If you have not previously incorporated the music words from BFORTH6 and 7, into your BSAVED fast loading Forth, you must load them first or delete the CH word at the end of line 13 which plays the bugle call CHARGE at the end of the message. Furthermore this screen uses my Useful Forth Word WAIT which is defined below for those of you who do not have a copy of ASF#2. The word WAIT must be loaded before attempting to load and use TALK.

```
: WAIT ( sec) 7200 * 0 DO LOOP ;
```

WAIT produces approximately a one second delay, in a loop to provide the required wait period in seconds.

For the curious, note that I have redefined TALK at the end of the screen, but it works anyway. That is because each definition has been separately compiled and each application of the word uses its own version. Redefining a word does not change the previous definition. To change it you must first FORGET it. Is not Forth wonderful?

continued on page 10

Writing in Machine Code

The Video Processor, part 1 by J.E. Banfield

The TI99/4A relies on the TMS9918A series of video processor chips, which differ mainly in their video output specifications, which need not concern us. This processor is a very complicated integrated circuit, probably much more complicated than the TMS9900 CPU chip, at least its data manual is three times as thick as the TMS9900 manual.

The video processor controls a 16K dynamic RAM which it uses for screen display data, the colour table and sprite definitions. There is sufficient dynamic RAM address space left over to be a very significant addition to other TI99/4A RAM; the BASIC interpreter uses it extensively. I use it as a buffer for disk data. For example, a full disk track can be set in VDP RAM prior to transfer to a DMA memory in my disk controller.

The control levels required to interface the video processor are defined in Table 6-1, extracted from the data manual. We can now face the job of screen control.

Screen Border Control

The border screen colour is controlled by the rightmost 4 bits of register 7 in the VDP chip. To change this colour code, we need to write to that register, 1 for black, F for white, etcetera.

This write to VDP register can be done in various ways. I chose here a long method which is less complicated than the alternative described later. The method of writing a program using the MiniMemory Easybug option was detailed in the last article and will not be repeated here. Enter the following program starting at M7FC0.

```
M7FC0   02 01    MOVEI 1,
        01 00    "black"
        02 02    MOVEI 2,
        87 00    VDP register 7
        D8 01    MOVB @.+2, 1
        8C 02    VDP write address
        10 00    SKIPA nil (delay)
        D8 02    MOVB @.+2, 2
        8C 02    vdp write address
        10 FF    SKIPA minus 1 (STOP)
```

Note that the first byte written contains the data, the second byte specifies a write to register 7 (see Table 6-1). Now press "." and type "E7FC0<enter>". The border colour goes black and we enter an infinite loop. Turn off the console and back on again (or reset any way you can).

Let us examine the program.

```
M7FC0   02 01
0 0 0 0   0 0 1 0   0 0 0 0   0 0 0 1
\_______020_______/ \_______/
     op-code           S address
     MOVEI               Ac 1
```

See Figure 1 in the last article. The instruction moves the immediate data, to be found in the next word (01 00) to Ac 1.

```
M7FC4   02 02
0 0 0 0   0 0 1 0   0 0 0 0   0 0 1 0
\_______020_______/ \_______/
     op-code           S address
     MOVEI               Ac 2
```

The data in the next word (87 00) is placed in Ac 2.

```
M7FC8   D8 01
1 1 0 1   1 0 0 0   0 0 0 0   0 0 0 1
\_____/ \_______/ \_______/
op-code   D address   S address
 MOVB       @.+2       Ac 1
```

Move the contents of the left hand byte in Ac 1 (that is 01) to the address given in the next word, which is 8C 02.

1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0

This has the address line of A14 high which enables the VDP chip for writing to as can be seen from Figure 6-1 and Table 6-1, that is U100 pin 14 (CSW(L)) will be low and pin 13 (MODE) will be high.

The next instruction is a delay which is required by the Video Processor chip. The following instruction is similar to 7FC8 but specifying the source as the left most byte of Ac 2, so the second byte written is 87, defining VDP register 7. The final instruction, SKIPA minus 1, as explained in the second last article, enters an infinite loop, in effect causing a stop.

OK, if all goes well we will make a change to avoid the inconvenience of the infinite loop. Change the instruction at 7FD2. In Easybug type:

```
M7FD2   04 60    JUMPA @.+2
        7F E0    address 7FE0
```

This instruction is made from:

```
M7FD2   04 60
0 0 0 0   0 1 0 0   0 1 1 0   0 0 0 1
\_______044_______/ \_______/
     op-code           S address
     JUMPA               @.+2
```

Then enter a short program at 7FE0.

```
M7FE0   02 06    MOVEI 6,
        70 E5    GROM address
        04 60    JUMPA @.+2
        00 60    ROM address
```

This loads Ac 6 with a GROM address (miniMemory GROM entry?) and the next jumps to the GPL interpreter in the console ROM. Check this by executing 7FC0.

There is no change to the border colour but "?" is displayed. Now change the colour data by:

```
M7FC2   09
```

and repeat the execution of 7FC0.

Now examine the contents of M83FE.

```
M83FE = 8C
M83FF = 02
```

Incredible, the VDP write data address is in console RAM and we did not put it there!

Sacred Sites

TI, in its wisdom (??), have sequestrated certain addresses for specific purposes and you may only change their contents AT YOUR PERIL. In particular:

```
83FA    (Ac D or R13 in GPL interpreter)
83FC    (Ac E or R14 in GPL interpreter)
83FE    (Ac F or R15 in GPL interpreter)
```

Although we must not change the contents of these registers, there is no bar to using them as data as described for Ac F later.

Multiple VDP Byte Write

Time and space is running out fast so I will leave detailed explanation of these routines to a later article. However, you might like to try them out and analyse the code. This routine writes a selected byte a number of times as defined in the data to incrementing VDP addresses. I use it to set up track data for formatting disks.

```
M7C60   C0 FB    MOVE 3, @ B+
        C1 03    MOVE 4, 3
        02 43    ANDI 3,
        FF 00
        02 44    ANDI 4,
        00 FF    count
        D8 03    MOVEB @.+2, 3
        8C 00    VDP write data
        06 04    SOS 4
        16 FC    SKIPNE minus 4
        04 5B    JUMPA @ B (return)
```

```
This is called by:
      06 A0     JSP @.+2
      7C 60
      WX YZ     data; WX is the ASCII byte to be
                written YZ times.
```

Before executing the 7FC0 subroutine, it is necessary to set the VDP address which can be done in the following subroutine.

```
M7A00   C0 7B     MOVE 1, @ B+
        D7 E0     MOVEB @ F, @.+2
        83 E3     Ac 1, right byte
        10 00     delay
        D7 C1     MOVEB @ F, 1
        04 5B     JUMPA @ B
```

In fact, we can use the 7A00 subroutin to change the border colour as as alternative to the 7FC0 routine.

Now enter the program:

```
M7A40   06 A0     JSP @.+2
        7A 00     set VDP address
        40 E3     address with write flag
        06 A0     JSP @.+2
        7C 60     multiple byte write
        41 10     write "A" 16 times
        04 60     JUMPA @.+2
        7F E0     to ? in Easybug
```

Now execute 7A40 several times. Try changes to address, data and count.

From Table 6-1, you will see that the only difference in VDP register write from VDP address write, is in the second byte written. So try:

```
M7A20   06 A0     JSP @.+2
        7A 00     set VDP address
        87 01     or 87 09 or 87 0X
        04 60     JUMPA @.+2
        7F E0     to ? in Easybug
```

The next article in this series will explain the above and show how to write the colour table in VDP RAM.

Correction to the article in Volume 11 Number 5, page 20. "Not valid for TMS9900" should read "Not valid for the TI99/4A". The TI99/4A lacks address decoding needed to implement these TMS9900 instructions.

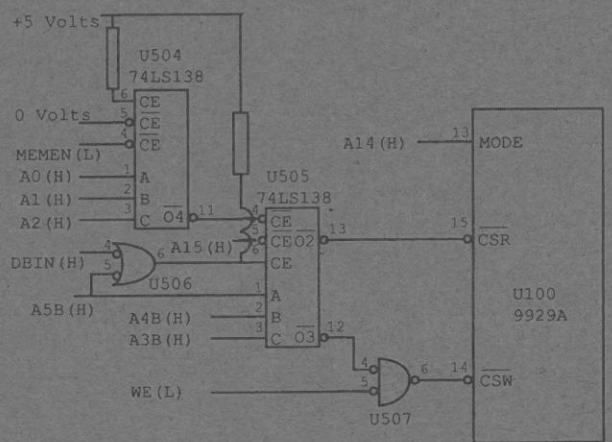


Figure 6-1
From TI99/4A Schematics Diagrams, sheets 1 and 4

```
O4[U504](L)=MEMEN•A0•A1•A2

O2[U505](L)=O4[U504]•A15•(DBIN+A5B)•A5B•A4B•A3B

     CSR(L)=MEMEN•A0•A1•A2•A3B•A4B•A5B•A15
x=don't care      1000  10xx  xxxx  xxx0
                    8     8           even

O3[U505](L)=O4[U504]•A15•(DBIN+A5B)•A5B•A4B•A3B

CSW(L)=WE•O3[U505]
      =WE•DBIN•MEMEN•A0•A1•A2•A3B•A4B•A5B•A15
                  1000  11xx  xxxx  xxx0
                    8     C           even
```

Newsletter Update

by Bob Relyea

In this update we have some newsletters from the users group in Perth so we know that they are alive and well. Newsletters are no longer available from the Melbourne group but we have made enquiries and word has come back saying that they are still meeting together and functioning.

TIBUG (Brisbane), June, 1992: Editorial; Word Processing #3; What's News (including a hopeful comment that we might be getting the 80 column boards soon!); Shop News; Tips to Remember (humorous); The MXB System; Tips #40; New Disk Drive Owner?; Bits and Pieces; Letter to the Editor; Tigercub Reformatter; Module Library; Trading Post; Word Shooter; Extended Basic Tips; In the P.O. Box; Cadet Console Expansion.

TI UP TIt BITS (Western Australia), December, 1991: Editorial; Rescuing a RAMdisk with a corrupted ROS; Mutant Mushrooms (game listing); PLUS V. 2.0; Another Joystick Conversion; Coming Events; Printers #3 by John Willforth; The TIsHUG 'AT' Multifunction Card; Competition advertisement.

February, 1992: From the Editor; John Birdwell's Disk Utilities; More On Printers; Archiving Revisited; Archiving - A Headache?; Extended Basic to Artist Graphics - Another Method; Beam Heading (game listing); Taking Control of Formatter.

March, 1992: Notice of 10th anniversary celebration of TI-UP; Assembler and Forth; C Language; What is Home Computer?; Playing With Matches; Technical Talk, The TI-99/4 Port; Crossword Solution; Another Gem from Frank Graham.

TOPICS, May, 1992: You're Looking At IT; XB Files by Earl Raguse; Comparing Health Insurance; The President's Message; Managing Your Money-2; Routine Interweaving.

June, 1992: This Is It; Tigercub Reformatter; President's Message; X.B. #12, More On Files; Managing Your Money #3; LA 99er Library; The Market Place.

THE BOSTON COMPUTER SOCIETY, May, 1992: Listen by Justin Dowling; Pin Diagrams; Remind Me; An Undocumented Phone Feature of Remind Me.

OTTAWA, June, 1992: Editor's Notes; PC Film for TI Joysticks; The President's Two Cents Worth; Fast Extended Basic by Lucie Dorais; Programming The Easy Way - Part 3 by Jim Peterson; Hotline Numbers.

SPIRIT OF 99, June, 1992: Calendar; The Annual Lima TI Conference; Tips #69; Barry's Corner; About the DOM; Assembly Language, Lesson 3 by Bob Webb; Speech (Part 2), Turbo Speech by Stephen Shaw; Lima Fair by Dick Beery; Files; Some New Updates; TI World News; Defragmenter.

TIdbits, June, 1992: President's Bit; Automatic Computer Music; Managing Your Money - 2; The Case For Extended Basic; Anyone Out There; A Southern Computer Fair; Debugging; TI-Base Tips; Date 890305; Special Thanks; Word Fun, Part 1, Answers; Program Bit. o

OPERATION	BIT 0	1	2	3	4	5	6	7	$\overline{CSW}$	$\overline{CSR}$	MODE
WRITE TO VDP REGISTER											
BYTE 1 DATA WRITE	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	0	1	1
BYTE 2 REGISTER SELECT	1	0	0	0	0	RS_0	RS_1	RS_2	0	1	1
WRITE TO VRAM											
BYTE 1 ADDRESS SETUP	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	0	1	1
BYTE 2 ADDRESS SETUP	0	1	A_0	A_1	A_2	A_3	A_4	A_5	0	1	1
BYTE 3 DATA WRITE	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	0	1	0
READ FROM VDP REGISTER											
BYTE 1 DATA READ	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	1	0	1
READ FROM VRAM											
BYTE 1 ADDRESS SETUP	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	0	1	1
BYTE 2 ADDRESS SETUP	0	0	A_0	A_1	A_2	A_3	A_4	A_5	0	1	1
BYTE 3 DATA READ	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	1	0	0

Table 6-1 - CPU/VDP Data Transfers. From the TMS9918A Manual o

Regional Group Reports

Meeting Summary For SEPTEMBER

Banana Coast	13/09/92 Sawtell
Central Coast	12/09/92 Saratoga
Glebe	10/09/92 Glebe
Hunter Valley	12/09/92
Illawarra	14/09/92 Keiraville
Liverpool	11/09/92
Northern Suburbs	24/09/92
Sutherland	18/09/92 Jannali

BANANA COAST Regional Group
(Coffs Harbour Environs)

We never miss meeting at Kerry Harrison's residence 15 Scarba St. Coffs Harbour, 2 pm second Sunday of the month. Visitors are most welcome. Contact Kerry 52 3736, Kevin 53 2649, Rex 51 2485 or John 54 1451.

CENTRAL COAST Regional Group

Regular meetings are normally held on the second Saturday of each month, 6.30pm at the home of John Goulton, 34 Mimosa Ave., Saratoga, (043) 69 3990. Contact Russell Welham (043)92 4000.

GLEBE Regional Group

Regular meetings are normally on the Thursday evening following the first Saturday of the month, at 8pm at 43 Boyce Street, Glebe. Contact Mike Slattery, (02) 692 8162.

HUNTER VALLEY Regional Group

The meetings are usually held on the second Saturday of each month at members homes starting at 3:15 pm. Check the location with Geoff Phillips on (049) 428 176. Note that after 9:00 pm this number is used for the ZZAP BBS which includes TI-99 information. Geoff.

ILLAWARRA Regional Group

Regular meetings are normally held on the second Monday of each month after the TIsHUG Sydney meeting, except January, at 7.30pm, at the home of Geoff & Heather Trott, 20 Robsons Road, Keiraville. A variety of activities accompany our meetings, including Word Processing, Spreadsheets and hardware repairs. At our last meeting we rebooted Bob's Ramdisk ROS, looked at features of the new Funnelweb V. 5.0 and did some disk copying. Contact Lou Amadio on (042) 28 4906 for more information.

LIVERPOOL Regional Group

Regular meeting date is the Friday following the TIsHUG Sydney meeting at 7.30 pm. Contact Larry Saunders (02) 6447377 (home) or (02) 7598441 (work) for more information.

NORTHERN SUBURBS Regional Group

Regular meetings are held on the fourth Thursday of the month. If you want any information please ring Dennis Norman on (02)452 3920, or Dick Warburton on (02) 918 8132. Come and join in our fun.

Dick Warburton.

SUTHERLAND Regional Group

Regular meetings are held on the third Friday of each month at the home of Peter Young, 51 Jannali Avenue, Jannali at 7.30pm.

Peter Young

TIsHUG in Sydney

Monthly meetings start promptly at 2pm (except for full day tutorials) on the first Saturday of the month that is not part of a long weekend. They are held at the RYDE INFANTS SCHOOL, Tucker Street (Post Office end), Ryde. Regular items include news from the directors, the publications library, the shop, and demonstrations of monthly software.

SEPTEMBER MEETING - 5th SEPTEMBER

This will be a tutorial day. If you have been wanting to get some help using a program (such as TI-Base) or in program writing or in using some utility then bring your problem to the meeting and I am sure there will be somebody there to help you.

****

The cut-off dates for submitting articles to the Editor for the TND via the BBS or otherwise are:

October	13 September
November	15 October

These dates are all Sundays and there is no guarantee that they will make the magazine unless they are uploaded by 6:00pm, at the latest. Longer articles should be to hand well before the aboves dates to ensure there is time to edit them.

****

TIsHUG Meetings for Sydney

September

The second all day tutorial session. Your chance to learn about using software, writing programs or understanding hardware. We can provide anything that you want but you must tell us what you would like and at what level you would like it.

October

The third buy, swap and sell day. This one is in the middle of the school holidays but plan to take the day off and see what is about.

November

The TI-Faire will be a few weeks after this meeting so it may be taken up with the organizational requirements of this big day. New software and hardware to be demonstrated. Watch this space for more details. Time to think about nominating for positions on the board. I am sure there will be some vacancies this year!

December

The Annual General Meeting followed by some festive eats and drinks, There will probably be a bit of celebration after the TI-Faire, if we are all still friendly after the event. Make sure that you attend and give your support to all the workers in the club. ○

continued from page 4

3 Grade Average Program
4 Grade Book
5 Grade Point Average
6 Grading Program
7 Teacher's Helper
8 Teacher's Pet
9 Quizwriter

DISK TC-990 is a collection of programs which relate in some way to sports, such as working out your handicap, or keeping statistics on different types of sports. The disk is in SSSD format and all programs require the Extended BASIC cartridge. The following programs can be selected from a menu.

1 Baseball Stats
2 Bowling Stats
3 Fumble
4 Golf Handicap
5 Handycapper
6 Magic Number
7 Ping Pong Scoreboard
8 Power Rater
9 League Scheduler
10 Tournament Fishing
11 Trifecta ○

A495, A505, TC09