The P-GRAM Card

Something else to spend your money on!

TIsHUG News Digest   **ISSN 0819-1984**

The Board

Co-ordinator
Dick Warburton   (02) 918 8132
Secretary
Terry Phillips   (02) 797 6313
Treasurer
Rolf Schreiber   (042) 84 2980
Directors
Robert Peverill   (02) 602 4168
Russell Welham   (043) 92 4000

Sub-committees

News Digest Editor
Geoff Trott   (042) 29 6629
BBS Sysop
Ross Mudie   (02) 456 2122
BBS telephone number   (02) 319 1009
Merchandising
Steven Carr   (02) 608 3564
Publications Library
Warren Welham   (043) 92 4000
Software library
Terry Phillips   (02) 797 6313
Technical co-ordinator
Lou Amadio   (042) 28 4906

Regional Group Contacts

Carlingford
Chris Buttner   (02) 871 7753
Central Coast
Russell Welham   (043) 92 4000
Coffs Harbour
Kevin Cox   (066) 53 2649
Glebe
Mike Slattery   (02) 692 0559
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 $25.00
Overseas Airmail Dues   AUS$50.00

TIsHUG Sydney Meeting

The next meeting will start at 2 pm on 7th of October at the Woodstock Community Centre, Church Street, Burwood.

Do not forget the Melbourne TI-Faire on Saturday 14th October

Printed by
The University of Wollongong
Printery

Index

TIsHUG Fairware Author of the Month

The Fairware Author for this month is Stuart Olsen for his Program Mass Transfer. All Donations collected at the meeting and sent in will be mailed to him this month

---

They're off   by Geoff Trott

A long time ago we bought a minicomputer at work. It was not a very expensive one but it cost us a lot of money. It was physically quite large from today's point of view as it lived in a 19" rack. Program entry and storage was paper tape although it had a magnetic tape system which eventually after considerable effort became usable for this purpose. After a number of years we took the plunge and bought a hard disk system, 10Mbyte for $10,000. It was also large, as large as the original computer. It made life much easier and the system continued to run for many years and is still there as it turns out but is not getting much use. My thoughts turned to this topic because I am currently looking at replacing its function with another computer and considering retiring it to my house where it would join the rest of the junk downstairs to be finally thrown out. The interesting thing is to consider the size and cost reduction for the same computing power that has occurred over the years. That minicomputer had a maximum memory capacity (all RAM) of 64Kbytes, unless you bought a memory management card. All the cards were large (.16 square metres) and you needed at least 3 cards for a minimum system. The power supply was large and heavy with a 20 ampere 5 volt supply. Memory (core) was expensive with 8Kbyte on one card and costing $2700. I am now using a TI99/4A system with more total memory and more facilities (like colour, graphics, speech, sound) and easier to use software which cost a fraction of the amount and takes up a fraction of the room of that ancient system. And yet that system is working just as fast as it ever did and still doing the job that it was bought to do. It is just starting to look like an anachronism and taking up too much space. In fact it is as fast if not faster than the PCs and Macintoshes in the things that it does well. Progress means different things at different times. I guess I am conservative in that regard: if something is still doing its job then there is no need to change it just because it is old. If the job changes and it is no longer able to keep up that is a different story. No prizes for guessing how old my car is!

Co-ordinator's Report

by Dick Warburton

Christmas and the festive season approaches. Our December meeting is usually social, and this year we would like to provide an outing for all the members of our long suffering families. What happens in the home of a "normal" TI99/4A user? Would a keen TI99/4A user go without eating or sleeping trying to fix a crashed RAMdisk, or even hard disk now? Do our members appear deaf at times, especially when the lawn needs mowing and we have not backed up that new disk? Have TI99/4A users been known to talk about things other than a TI99/4A? Well folks, we may find out the answers to these and other important questions at the end of year family picnic. Our social bloodhound, Les Andrews, is on the trail of a good deal for all.

The September issue of the TND was a beauty. I personally like the idea of a theme for each issue. This month's digest looks at hard disks. I found it particularly interesting. In fact, the technical smorgasboard being presented each month is so helpful, that I am developing a conflict trying to decide which activity to finish next. The problem of choice is usually resolved in my case, by the lack of sufficient cash. My congratulations again to the team in Wollongong. What has become clear to me is that there are many things I can learn to do to expand my system. We can reduce our costs if we do things for ourselves, as well as gain real personal satisfaction from the results. I do not know of any other group where the members do so much for themselves.

There are some interesting avenues for further expansion. The hard disk certainly seems worthwhile. The Geneve seems to have missed the boat. The P-GRAM card might be worth a second thought, and the club has bought one. Derek Wilkinson's IBM keyboard is another approach. Les Andrews has received cables from South Australia, which will allow connection to a CGA and EGA colour monitor. With an 80 column card, a hard disk, upgraded software and a good resolution monitor, TI99/4A users are well set up. Where will it end? What is the limit? Probably cash flow problems. An idea suggested to me by Gary Wilson at the last meeting seems worthy of further consideration. Gary suggests that as the cost of RAM chips is so dear, and RAMdisks are becoming very expensive to make, we could concentrate on producing a RAMdisk with EPROMs plugged in. The cost of 64K EPROMs is quite low compared with static memory chips. Three suggestions have been made.

1. We make a board for the PE box with 64K EPROMs, and put all our commonly used software in the EPROMs. We would add about 64K of RAM chips to load the programs into.
2. We make additions to existing RAMdisks by adding 2 64k EPROMs to the board.
3. We develop a real supercart with 128k of EPROM programs.

Craig Sheehan has already worked out a simple circuit, and claims that it can be done fairly easily and cheaply, using off the shelf components. I find the idea of this project quite exciting. Imagine most of your favourite programs available on EPROM. No more corruption of important programs. It would free up the available memory on RAMdisks for other uses. I think there would be many advantages.

I am quite excited about this as a project. I feel that there would be sufficient interest to set up a projects group, where we could develop and build our projects together. There are many things we can do, or learn to do. Tell me what you think of the idea at the next meeting.

See you there Dick Warburton   o

Fairware Author of the Month

We, the users of the TI99/4A, rely on many people for our enjoyment of our computer, none more so than those who have written software which we use and rely on every time we use our computer. Some of this will be commercial software which we should have paid for and received value for our money in the form of a working program with good documentation, but the majority of software will be Fairware, which may not have cost anything and yet still provides a working program and good documentation. Software authors who produce good useful programs and release them for us all to enjoy under the fairware concept are the ones who are keeping us all going. If you look at the price of commercial programs for other computers which do the jobs that we are able to do with our fairware programs, you will find that $100 will not buy more than 1 program and you may well need $1000 to get a state of the art program. Fairware software costs the price of a disk initially but if we use the program the onus is on us to send a contribution to the authors to repay them for their efforts and encourage them to continue development and perhaps write a new program as well. We can be sure that these authors are not relying on our contributions to live, as they do not ask enough and we do not send enough, if anything at all.

TIsHUG now offers us alternatives to sending the money direct to the author. Of course sending the money direct to the author is the best way to get on an author's mailing list and to ask some pertinant questions about the software or about improvements which might be made in the next release. TIsHUG is offering to collect money from us for fairware authors and to send it on in the correct currency. This allows us to contribute each month by mail or in person to the monthly fairware collection or to send in a contribution to be spread amongst several products and their authors. If you use this last method, be sure to send in a list of software and the amounts for each.

The Fairware software product for this month is Mass Transfer from Stuart Olson in Illinois. This is a terminal emulation program which I find easy to use with our BBS and with very few problems. It has Xmodem protocol available, which is not supported on the BBS but is an excellent way of sending files to another user directly. It also has a multiple file transfer Xmodem in which only one transfer is initiated and all the files are sent and appear on the receiver's disk as the separate files with their correct names. If you use this program then please send in a contribution. Do not forget the other authors of fairware software that you use and take up TIsHUG's offer of forwarding your contribution on to them.   o

SEND IN YOUR DONATIONS NOW!

$15 per year. As you can imagine $15 is little more than the cost of printing and mailing this great newsletter. If you would like to send your checks to me (Payable to the NorthCoast 99'ers UG), I will expedite your membership. Also any comments on the TI-Base column can be sent to Martin A. Smoley, 6149 Bryson Drive, Mentor, Ohio, 44060.

I am going to announce at this time that I will produce a TI-Base help disk. The disk will be a flippy and contain all of the tutorials and Command Files to date plus anything else I think may be helpful. I already have 390 sectors of tutorials. For this I would like a donation of $3 to cover the Disk, Mailer, Postage, Handling and wear on my disk drives. Please make these checks payable to Martin A. Smoley at the previous address, and make two checks if you want the help disk and a membership.   o

Secretary's Notebook
by Terry Phillips

The September meeting was reasonably well attended despite Geoff's problems in mailing the September issue of the newsdigest. Again we were relegated to the outbuildings for our meeting due to Woodstock having their drama society conducting a play in the main downstairs area. It happens from time to time that we will be moved around by Woodstock management and I guess we just have to fit in with their plans. The good news is that from the October meeting onwards we should be back upstairs in our normal meeting rooms. This next meeting will also see the kick-off of our Games Room activity in a small room upstairs. We hope to have a couple of systems setup to let those into games go for their lives.

Bulk subscriptions to Asgard News have been received and have been mailed to members who requested them. There are no spare subscription copies available but if you wish to have a perusal of this publication, it may be borrowed from the Library. It looks very good value with some excellent articles.

We only have two new members to welcome this month, and they are,
> Robert Keast of Dapto
> Mark Stuart-Street of Figtree.

It looks like the Illawarra boys are at it again and are having good success in recruiting new members. The TI99/4A is alive and well down Wollongong way.

There was no shop facility at the last meeting due to Stephen being on the sick list. Also on the sick list has been our Treasurer, Rolf, who has just been released from hospital. Get well soon all you guys and hope you can make it to the next meeting.

Coming up at the next meeting will be short talks by the various Special Interest Group leaders who will give timely information on where they are at. If you are interested in joining up with one of the SIG's and learning more about a particular piece of software or hardware then this will be your big chance to ask a few questions.

At this stage it is planned to hold a full day tutorial event at the November meeting. Full details of topics are still in the planning area but will be fully publicized in the November newsdigest.

Christmas is not all that far away, and as usual the December meeting will take the form of an informal Christmas party get together. Should be fun if we have a fine day and can get outside for a barbeque.   o

TIsHUG Software Column
by Terry Phillips

Inscebot Inc. have come out with a very nice new piece of software called TI-Sort. This is a perfect companion disk for owners of TI-Base and is subtitled "The Incredible Sortware". It comes on a single disk with a well written manual which gives an overview, getting started and detailed instructions on its use. It will just about sort anything, not only TI-Base files, so it should be a very handy item of software to have among your collection. Copies will be on sale at the shop at the October meeting for the cost of $15.

Inscebot have also released a maintenance update for TI-Base, numbered Version 2.02. According to Dennis Faherty this version is fully hard disk compatible and it also fixes some minor problems that existed in Version 2.01. Copies will be available at the October meeting, so either bring in your old disk or a blank disk to obtain the upgrade.

A disk has been received from Colin Christensen of the Brisbane Users group that contains a bunch of utilities including Hardmaster which is for use by members with hard disk drives. It also contains an excellent program called Tapemaster which I wish had been around a few years back. It makes backing up programs from disk to tape a dream. Boy would it have saved me some time. Also a Lotto/Pools number picker which really does a good job in checking your results. Unfortunately it did not pick the winning numbers for me when I tried it out but at least it is a lot of fun. This disk will also be available at the October meeting.

The Sutherland Regional Group have been trying out a new method of demonstrating software to their members. They give me 3 or 4 disks at each meeting and ask for copies of various items of software with the view being to show them at their meetings. All Regional Groups are welcome to do this, so just bring a few disks in with your list of software. If out of town and cannot make it to the meetings, post them down with the list.   o

Is the answer abc? That is the question!
by Ben Takach

I bet you have noticed the full page Benson and Hedges advertisement in the Daily Mirror, complete with a couple of tubes of 27C64 EPROMs spilled all over the page, and an abc equation on the top left hand corner, whether you are a smoker or a non-smoker.

The equation is written thus:

$$(b^2 + ab) \times (c + b) - b^2 \times c - b^3 + ac^2 + bc - SQR(a^3 \times c^2 / b) \times SQR(ab) - bc - (b^2 \times ac)/c = abc.$$

Well the person who wrote the equation is certainly no mathematician! Furthermore, he does not know how to use a TI99/4A. The equation of course is wrong. The left side cannot be equal to abc! Try to evaluate each term, if you feel like doing it, otherwise take my word for it. In fact, the left side of the equation is equal to $abc + ac^2 - a^2c$.

It is obvious the fellow was a sickly student. He missed many of his early mathematics lectures. Let us prove him wrong using the TI99/4A.

First we have to assign some values to a, b, and c. We can do all of our work in immediate mode, there is no need to write a program. It is easier to use Extended BASIC, it saves some typing. Here we go: first we assign some values to a, b, and c. Type in: >a=2::b=3::c=4 then press <ENTER>.

Next we enter the equation: PRINT (b^2+a*b)*(c+b)-b^2*c-b^3+a*c^2+b*c-SQR(a^3*c^2/b)*SQR(a*b)-b*c-(b^2*a*c)/c

Then press the <ENTER> key. The display will show the result is 40.

Now we enter: PRINT a*b*c.

Again the result will be displayed: 24.

Clearly the two sides are not equal.

Now do a FCTN[8] and add to the PRINT a*b*c line: +a*c^2-a^2*c, and press <ENTER>: The result of course is 40 (do you think I have written all this without trying it first?)

Postscript. I have several hundred WD & HO Wills shares for sale. I have decided to get out of this company. If their accountants calculate my dividends using similar calculating methods, then I am better off getting out while the sun is shining! And the moral to the story, you do not have to be an Einstein if you have a TI99/4A, just use common sense and try!   o

Expanding XBs Powers
Writing Assembly Routines, part 4
by David Caron, USA

The four Extended BASIC routines I am about to discuss are what make Assembly and Extended BASIC such a great team. Using these routines, you can transfer up to sixteen variables between Extended BASIC and Assembly. Any of these variables could be entire arrays if you wish. Just imagine! You could make up an unlimited number of new Extended BASIC procedures, like reading in a whole line from the VDP screen to a string all at once instead of using CALL GCHAR. Today, however, we will just use these routines to print something on the screen. The four routines are:

```
NUMREF (NUMber REFerence, BLWP @>200C)
NUMASG (NUMber ASsiGn,    BLWP @>2008)
STRREF (STRing REFerence, BLWP @>2014)
STRASG (STRing ASsiGn,    BLWP @>2010)
```

All of these routines are loaded from the Extended BASIC module into low memory along with VMBW, VMBR etc., when CALL INIT is executed.

NUMREF is a routine which copies a variable from Extended BASIC into Assembly. Let us take an example. Say you executed your assembly routine with: CALL LINK("START",X). X is an Extended BASIC variable which I will let be equal to 5. Just before you execute BLWP @NUMREF in Assembly, you load a value into R0 and R1. The value in R1 tells NUMREF where in the parameter list the variable is. In our case, the variable is the first parameter in the list, so R1=1. For simple non-array variables like X, always set R0=0.

If X had been an array like DIM X(10) and you wished to select any of the ten values from Extended BASIC, then the call link would look like CALL LINK("START",X()). In such a situation R0 is used to tell NUMREF which array element you want passed. If you wanted to pass X(4) then R0=4. If you wish to know how to access multi dimensional arrays, read section 17.2.1 in the Editor Assembler manual. For our purposes, knowing how to access single dimensional arrays will be sufficient.

Once R0 and R1 have been set, BLWP @NUMREF can be executed. The variable in the parameter list will be copied to CPU addresses >834A to >8352, yes 8 bytes, not two. I neglected to remind you that all of Extended BASIC numeric variables are in floating point notation and take 8 bytes to represent them. I will also add that they are always in a form of scientific notation, so it would be difficult to convert a single number like 5.000000000000E0 (X=5) to a Word (16 bit) number. Fortunately the TI99/4A home computer comes with a bunch of great little routines like CFI (Convert Floating point to Integer). This routine is accessed with:

```
BLWP @XMLLNK    *where XMLLNK=>2018
DATA >12B8
```

This very handy routine will take that awful 8 byte floating point number and convert it into the 16 bit number 5. It will then place this number at the address >834A. If you do something crazy like making X=1.465838734 then the CFI will simply round the floating point number and place a 2 at >834A. CFI even handles negative numbers! What more could one ask? Do not ask me what happens if X=1.0E99 or X=1.0E-99.

Remember that −32768<=X<=32767 for a possible conversion to 16 bit format. The only reason I am making such a big deal out of this is that I went to the trouble to make a routine similar to CFI and then learned of its existence in the console!

NUMASG is identical to NUMREF except that you place the 16 bit number you want sent to Extended BASIC, in

>834A. Notice however that before you can execute BLWP @NUMASG, the number must be in floating point notation. CFI will not work, but CIF will! (Makers of the TI99/4A evidently thought of everything.) CIF, as you may have guessed stands for Convert Integer Floating point. This console routine can be accessed using:

```
BLWP @XMLLNK    *where XMLLNK=(>2018)
DATA >20
```

and presto, there you are. All that is needed now is BLWP @NUMASG. If R1=1 then the Extended BASIC variable in the first parameter of CALL LINK will be set to the original integer at >834A. If the CALL LINK statement is something like CALL LINK("START",1) instead of CALL LINK("START",X) then NUMREF will return you to Extended BASIC and issue an error message.

STRREF is similar to NUMREF except that a string is passed from Extended BASIC to assembly instead of a number. R0 and R1 function the same way. R2 is used however to tell STRREF where you want the string. The usual procedure is to allot some memory before the actual start of your assembly routine. This memory is alloted in the same way as was done for the user workspace registers in the last article. You need only allot LEN(string)+1. The additional character indicates how long the actual string is. This is why strings cannot be any longer than 255 bytes: that is the largest number possible for a byte to represent.

Example: STRBUF BSS 25

This string can be no longer than 24 bytes. R2 is assigned the address STRBUF. Now only one more thing must be done. If, for example, the string in Extended BASIC was 255 characters long instead of 24, guess what would happen? STRREF would simply copy the string at STRBUF, then continue copying over your workspace register and much of your assembly routine. Such a situation would likely result in unpredictable results on the part of the TI99/4A computer. Fortunately there is a built in safeguard against this. When STRREF is called it will check the byte at the address indicated by R2 and check to make sure that the Extended BASIC string is no longer than the value of this byte. If it should be, STRREF will return execution to Extended BASIC and issue an error. When BLWP @STRREF is executed, the actual string starts at STRBUF+1 not STRBUF.

STRASG: Well, there is not much to say here. All you do is set R0=0, R1 to the string position in the CALL LINK parameter list, make up the string somewhere in CPU memory, set the byte immediately in front of that string equal to the length of the string, set R2 equal to the address of that byte, execute BLWP @STRASG and the string in CPU memory gets assigned into the Extended BASIC string variable.

Now I will rewrite the assembly routine from my last article using the above routines. This assembly routine can be accessed from Extended BASIC using CALL LINK("START",X,S$) where X is the starting address of the string S$ being written to screen, 0<= X <=767.

```
         DEF  START     *This places the word "START" in
                         the DEF table along with its
                         start address.
VSBW EQU  >2020          *The Extended BASIC
VMBW EQU  >2024          assembler
VSBR EQU  >2028          environment has no
VMBR EQU  >2030          REF table so the assembler
                         directive EQU (equate) must be
                         used to defined the constants
                         VSBW, VMBW, VSBR and VMBR. Take
                         a look at page 415 to 416 in the
                         Editor Assembler manual
NUMREF EQU  >200C
NUMASG EQU  >2008
STRREF EQU  >2014
STRASG EQU  >2010
XMLLNK EQU  >2018
```

Techo Time

with Lou Amadio

Unfortunately I was unable to attend the September meeting and I will be away on holidays for the October meeting. Still there are two meetings left this year which I should be able to attend (November and December). This year has gone very quickly for me. There has really not been enough time for me to participate in, let alone enjoy, my other hobby (building and flying radio controlled model aeroplanes).

Geoff brought back two very interesting items from the last meeting: a P-GRAM card kit and a small RGB monitor with interface. I will be assembling and playing with these over the next few weeks and will report my findings in this column. I also understand that Les Andrews has brought back a number of RGB monitor interfaces (designed and built by Colin Cartright of ATICC). Please contact Les if you are interested in one of these.

This month I will be presenting another way to interface Atari compatible joysticks and the final episode (I think) on the Direct I/O Interface).

Interfacing Joysticks to the TI99/4A

by Lou Amadio

Although interfacing joysticks has been covered more than once in this magazine, I am still asked either personally or through the mail how to adapt other brands to the task.

Virtually everyone who owns a computer will almost certainly own a pair of joysticks. If you bought a pair of original TI joysticks, chances are that they are probably worn out. If only the #2 stick is working, you could probably use the parts to keep #1 stick working. Eventually, however, you will be faced with the decision to buy one or more new joysticks.

Finding a joystick which is as easy to use (especially for children) as the TI stick will only solve one of your problems. The other is how to make it work with the console.
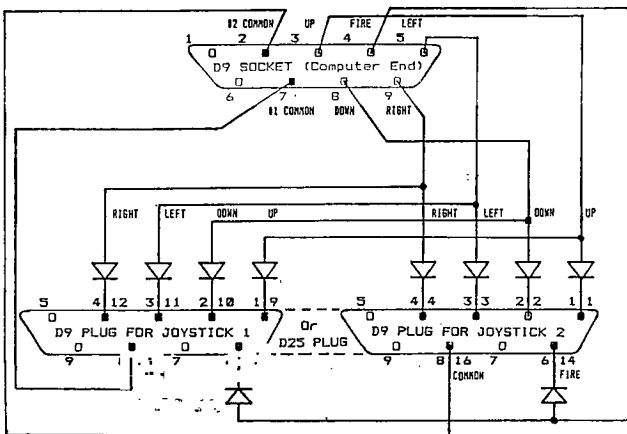
Joystick adapter circuit diagram

Courtesy of Paul Mulvaney, Hunter valley 99ers

Of all the interface articles that I have read, the one that I believe provides the best and most cost effective solution to the problem was described by Paul Mulvaney of the Hunter Valley 99ers. Paul used a 25 pin "D" connector (male) and associated plastic shell to form the joystick interface and house the isolation diodes. The diodes (1N4148) are supposed to prevent interaction between the two joysticks, which share a common return wire, as well as providing proper joystick operation in the diagonal directions. The D25 is wired to a 9 pin "D" connector (female) which provides the connection to the console.

To assist in plugging in the joysticks, the inner 7 pins of the D25 plug are removed. This can be achieved with a pair of needle nose pliers. The diodes are soldered directly to the pins of the D25 plug so as to fit inside the plastic shell. A short cable connects the seven wires to a 9 pin "D" connector (D9), which in turn plugs into the console joystick port.　　　　o

Direct I/O Interface
The final Episode?
by Lou Amadio

The Direct I/O (or Two-way Expansion) Interface project which first appeared in the July issue of the TND seems to have created a great deal of interest. First of all, errata on the subject was published in the August and September issues of the magazine. Apart from two errors in the wiring table, most of the confusion has been associated with the connections (or lack thereof) of the unallocated pins. For this reason, two tables are included this month, one referenced from the I/O port and the other referenced from the PEB connector. These tables supersede all other references to I/O to PEB connections. The new data shows a larger number of "pull-up" resistor connections when compared with the previous article. Although the interface will still work without the pull-up connections, it is recommended that you build it as per the attached tables.

The main purpose of this article, however, is to announce that a double sided printed circuit version of the Direct I/O Interface has been produced (courtesy of Geoff Trott). The PCB will obviate the need for most of the time consuming (and potential error producing) point to point wiring. Building the interface now will only involve making a number of through links, mounting 11 resistors, 4 capacitors, a voltage regulator and of course the I/O and PEB edge connectors.

Parts Required

11 x 1000 ohm 1/4 watt resistors
4 x 10 uF 25 volt tantalum capacitors
1 x 78L05 +5 volt regulator
Plus other parts as specified in the July '89 TND

Before soldering any components, drill five 3mm mounting holes through the PCB, one at each corner and one near the centre through the earth track as indicated. The central support is important if you decide to mount the I/O connector on a box. If the connector is to be mounted inside the box, the 44 way connector must be mounted proud of the PCB with sufficient clearance to form a proper connection with the console I/O port. (The wire-wrap edge connectors that I used had 13 mm long legs).

Construction

If you look at the two printed wiring diagrams presented with this article, you will see that one is labelled "component side" and the other is labelled "solder side". Note that the "solder side" diagram is a mirror image of what you would see by turning the PCB over.

Refer to the original article as a guide to the position and orientation of the edge connectors, and to the printed wiring diagrams below for the solder connections.

Solder all through links (approximately 46) prior to mounting any other components. Mount the four capacitors, regulator (watch polarity), and the eleven 1000 ohm resistors.

The I/O edge connector is mounted (6mm proud) from the "solder side" and soldered on the "component side" of the board. Please note the 6 through links associated with this connector and that pins 3,21 and 27 must be soldered on both sides of the PCB. For added strength, the I/O connector should also be soldered both sides on pins 1, 2, 22, 28, 43 and 44. One link from the I/O pin 44 track to PEB pin 10 track need only be made if Audio-In is required.

The PEB card edge connectors are mounted from the "component side" of the PCB and soldered on the "solder side" of the PCB. Note the 40 through links associated with these connectors and that six of these links must be made BEFORE the edge connectors are mounted onto the PCB.

Power Supply

Connections from the +18V, -18V, +9V and earth must be made to the separate power supply (see July '89 TND). As the 5 volt regulator is now part of the current design, the regulator specified in the original article is no longer required unless you intend to power an add on disk drive as well. In this case, all that is required is to substitute higher capacity transformers and to add a voltage regulator for the disk drive motor. For low power drives, I found that a 1A transformer was adequate for T2.

Referring to the circuit diagram on page 6 of the July '89 issue of the TND, substitute a 0/7.5/15 volt 2A transformer (Arlec PT6978) for T1 and a 0/15/30 volt 1A transformer (Arlec PT6672) for T2. Unregulated output voltages will increase to +11, +22 and -22. A 7812 (+12 volt) regulator is added to the +22 volt rail for the disk drive motor. (Refer to articles between July and September for hints on how to build power supplies).

I/O Port to PEB Connections

I/O #	Function	PEB #	I/O #	Function	PEB #
1	+5V SP	NC	23	GND	7,49
2	SBE(H)	NC	24	CLKOUT(L)	50
3	RESET(L)	6	25	GND	20,53
4	EXTIN(L)	17	26	WE(L)	54
5	A5(H)	40	27	GND	3,27
6	A10(H)	33	28	MBE(L)	NC
7	A4(H)	39	29	A6(H)	37
8	A11(H)	34	30	A1(H)	44
9	DBIN(H)	52	31	A0(H)	43
10	A3(H)	42	32	M(. . L)	56
11	A12(H)	31	33	C1(. .H)	55
12	READY(H)	4	34	D7(H)	19
13	LOAD(L)	18	35	D4(H)	24
14	A8(H)	35	36	D6(H)	22
15	A13(H)	32	37	D0(H)	28
16	A14(H)	29	38	D5(H)	21
17	A7(H)	38	39	D2(H)	26
18	A9(H)	36	40	D1(H)	25
19	A15/CRUOUT(H)	30	41	IAQ(H)	NC
20	A2(H)	41	42	D3(H)	23
21	GND	5,47	43	-5V	NC
22	CRUCLK(L)	51	44	AUDIOIN	10
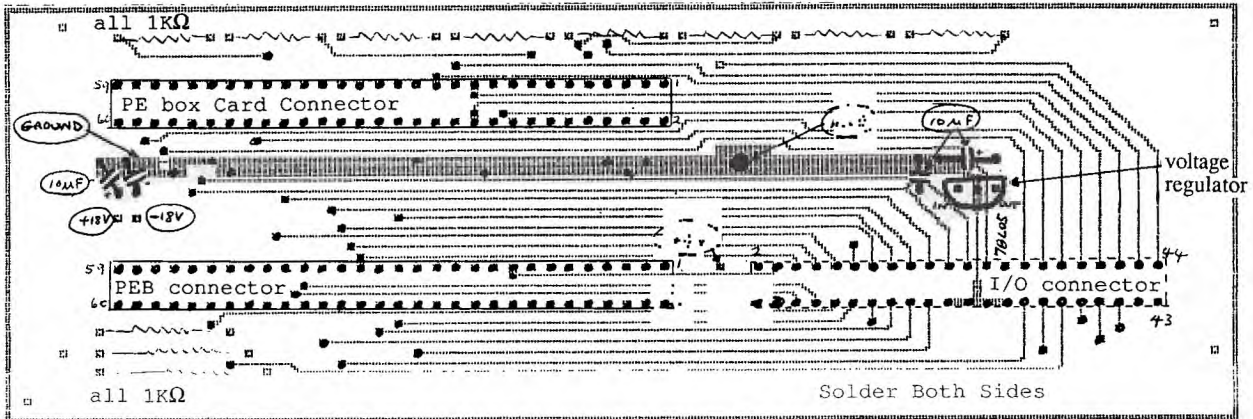
Direct I/O Interface - Component Side
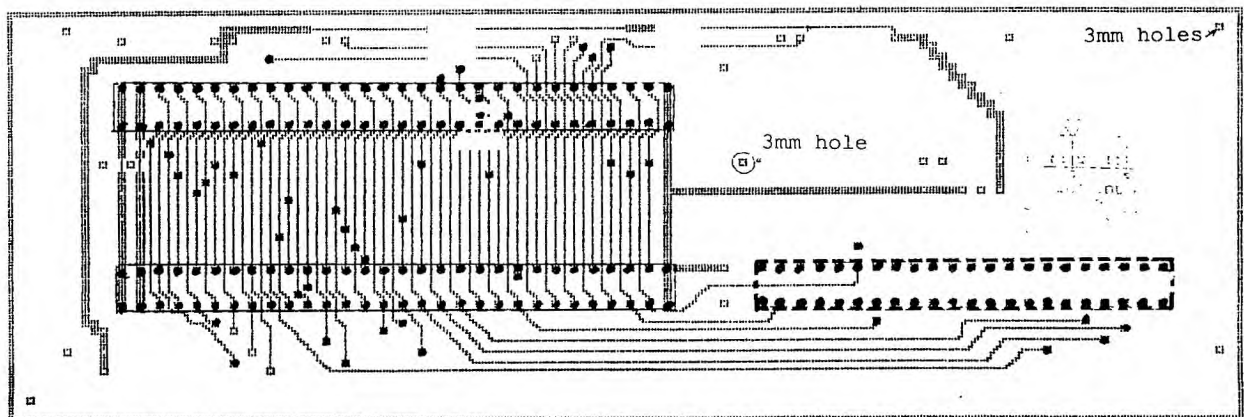
Direct I/O Interface - Solder Side

Extended BASIC Tutorial

by Tony McGovern, Funnelweb Farm

III. Sub-program Parameter Lists

In the last chapter we saw how sub-programs fitted into the overall workings of Extended BASIC. In this chapter we are going to go into the details of writing sub-programs. Most of the fiddly detail here concerns the construction of the parameter lists attached to CALL and SUB statements, and some of the little traps you can fall into.

Any information can be transmitted from the CALLing program to the CALLed sub-program via the parameter list, and anything not transmitted this way remains private for each program, with the exception of the DATA pool which is equally accessible to all. If something is mentioned in the parameter list then it is a two way channel unless special precautions, provided for in Extended BASIC, are taken. In this case the CALLing program can inform the sub-program of the value of a variable or entry in the parameter list, but not allow the CALLed program to change the value of the variable as it exists in the CALLing program. Arrays however, numeric or string, cannot be protected from the follies of sub-programs once their existence has been made known to the sub-program through the parameter list.

Let us for starters take a very simple but useful example, where a program needs to invoke a delay at various points. Now some BASICs (and TI Logo) have a built-in function called WAIT. Extended BASIC does not have this command (though a cynic might suggest that GPL gives it to you all the time whether you want it or not) so you have to program it. It can be done by a couple of CALL SOUNDs or with a FOR ,.., NEXT loop. Let us use an empty loop to generate the delay, about 4 milliseconds each time around the loop, and place the loop in a sub-program.

```
230 CALL DELAY(200)
670 CALL DELAY(200/D)
990 CALL DELAY(T)
3000 SUB DELAY(A):: FOR I=1 TO A :: NEXT I ::SUBEND
```

This is easier to follow when editing your program then using a GOSUB, and you would need to enter the subroutine in every sub-program since GOSUBbing or GOTOing out of a sub-program is verboten. Also it is less messy than writing the delay loop every time. The example shows several different CALLs to DELAY. The first supplies a number, and when DELAY is CALLed, the corresponding variable in the SUB list, A, is set to 200. This is a particular example of the kind of CALL from line 670 where the expression 200/D is first evaluated before being passed to DELAY to be assigned to A. Variable D might for instance represent the level of difficulty in a game. The CALL from line 990 invokes a numeric variable T, and A in the sub-program is set to the value of T in the CALLing program at the time when the CALL is executed.

Nothing untoward happens to T in this example, as the DELAY sub-program does nothing to change A. Now it may not matter in this instance if T did not retain its value after the sub-program CALL. Suppose instead the delay was to be called out in seconds. Then a sub-program on the same lines DELAYSEC might go

```
230 CALL DELAYSEC(2)
990 CALL DELAYSEC(T)
4000 SUB DELAYSEC(A):: A=250*A
4010 FOR I= 1 TO A :: NEXT I :: SUBEND
```

Now after DELAYSEC has been executed with the CALL from 990, T will have value 250 times its value before the CALL. This will not be a bother if you do not use T again for its previous value. If the CALLing program specifies a numeric constant as in line 230, or a numeric expression, the change in A in the sub-program has no effect on the main program. Suppose you cannot tolerate T being changed in line 990 (and this kind of thing can be a source of program bugs). You will find that Extended BASIC allows for forcing T to be treated as though it were an expression, thus isolating T from alteration by the sub-program, if T is enclosed in brackets in the CALL (not SUB) list. Suppose DELAYSEC is also called from line

```
970 CALL DELAYSEC((T))
```

If this CALL in line 970 is followed by the CALL from line 990, T not having been altered in the meanwhile, the same delay will be obtained, but if the order of CALLs were reversed the second delay would be 250 times the first. In the language of Extended BASIC this is known as "passing by value" as distinct from "passing by reference". This can only be done for single variables or particular array elements, which behave like simple variables in CALL lists. Whole arrays cannot be passed by value, but only by reference. Expressions and constants can only be passed by value, and its hard to see what else could be done with them. In the example as written, a different variable name was used in the SUB, but if you remember the little experiment in the last chapter you will see that it would not make any difference if T had been used in the SUB list instead of A.

Now let us complicate things a little by flashing up a message on the bottom line of the screen during the delay interval.

```
200 CALL MESSAGE(300," YOUR TURN NOW")
270 CALL MESSAGE(T,A$)
3000 SUB MESSAGE(A,A$):: DISPLAY AT(24,1):A$
3010 FOR I=1 TO A :: NEXT I :: DISPLAY AT(24,1):""
3020 SUBEND
```

The SUB parameter list now contains a numeric variable and a string variable in that order. Any CALL to this sub-program must supply a numeric value or numeric variable reference, and a string value or string variable reference, in precisely the same order as they occur in the SUB list. In the little program segment above, line 200 passes constants by value and line 270 passes variable references. There is no reason why one cannot be by value and one by reference if so desired.

This process can be extended to any number of entries in the parameter list, provided the corresponding entries in the SUB and CALL lists match up entry by entry, numeric for numeric, string for string. The Extended BASIC manual does not say so explicitly, but it appears that there is no limit apart from the usual line length problems, on the number of entries in the list. This is the only apparent difference between the parameter list in Extended BASIC sub-programs and the argument lists for CALL LINK("xxxxxx", ,...) to machine code routines in Extended BASIC, and MiniMemory and Editor Assembler BASICs.

One little freedom associated with built-in sub-programs is not available with user defined sub-programs. Some built-ins, such as CALL SPRITE permit a variable number of items in the CALLing list. Parameter lists in user defined sub-programs must match exactly the list established by the SUB list or an error "INCORRECT ARGUMENT LIST in ..." will be issued. User defined CALLs allow whole arrays, numeric or ·string, to be passed to a sub-program. Complete arrays may be passed by reference only. Individual array elements may be used as if they were simple variables and may be protected from alteration by bracketing in the CALL list. An array is indicated in the parameter list by the presence of brackets around the array index positions. Only the presence of each index need be indicated as in A(). MATCH(,,) indicates a three dimensional array MATCH previously dimensioned as such, explicitly or implicitly. Do not leave spaces in the list. If the sub-program needs to know the dimensions of the array these must be passed separately (or as predetermined elements of the array). TI BASICs are weaker than some others in that they do not permit implicit operations on an array as a whole, a very annoying deficiency.

Arrays may be DIMensioned within sub-programs. This will introduce a new array name to the program, and an array or variable name from the SUB parameter list cannot be used or an error message will result. In the following code the main program passes, among other things, an array SC to sub-program BOARD (perhaps a scoreboard writing routine in a game).

```
100 DIM SC(2,5) :: ....
450 CALL BOARD(P,A$(),SC(,))
4000 SUB BOARD(P,A$(),S(,)) :: DIM AY(5):: ......... ::
     CALL REF(P,AY(),S(,)))
4080 SUBEND
5000 SUB REF(V,A(),B(,)):: .... :: SUBEND
```

BOARD generates internally an array AY() which is passed to another sub-program REF (maybe this resolves ties) along with SC(,), which BOARD knows as S(,), and REF in its turn as B(,) -- the same name could have used in all places. There is however no way that the main program or any sub-program whose chain of CALLs does not come from BOARD can know about the array AY(). This would hold equally well for any variable or array, string or numeric, first defined within BOARD and whose value has not been communicated back to the CALLing program via some other variable mentioned in BOARD's parameter list.

By following this line of reasoning you can check out the conclusion that there is no way for a sub-program whose chain of CALLs does not come through BOARD to know about array AY(). The only way around this is for AY() to be DIMensioned in the main program (even if this is its only appearance there) and the message passed down all necessary CALL-SUB chains.

This idea of DIMensioning an array only within a sub-program is particularly useful if the array is to READ its values from DATA statements and to be used in the sub-program. This could be done again from any other sub-program needing the same data, without having to pass its name up and down CALL-SUB chains. Remember that DATA statements act as a common pool from which all sub-programs can READ. If the array values are the results of computations then these values must be passed through the CALL parameter lists.

For completeness note that although the Extended BASIC manual has nothing to say about it, IMAGE statements for formatting PRINT output are accessible from any part of a program in the same way as DATA statements and not confined to the sub-programs in which they occur as are DEF entries.

It is not necessary to have any parameters in the list at all. Sub-programs used this way can be very helpful in breaking up a long program into more manageable hunks for ease of editing. We shall also see in later chapters that there can be other benefits as well.

One more Extended BASIC statement for sub-programs remains, the SUBEXIT. This is not strictly necessary as it is always possible to write SUBEND on a separate line and to GOTO that line if a condition calling for an abrupt exit is satisfied. Like a lot of the little luxuries of life however, it is very nice to have and makes programs much easier to read and edit. It does not replace SUBEND which is a signal to the Extended BASIC pre-scan to mark the end of a sub-program. SUBEXIT merely provides a gracious and obvious exit from a sub-program (awkward in some Pascals for instance). The next chapter will demonstrate typical examples of its use.

IV. Useful Sub-program Examples

In the previous chapter we used as an example a DELAY sub-program which could, with a little refinement, be used to substitute for the WAIT command available in some other languages. You can extend this idea to build up for yourself a library of handy-dandy sub-programs which you can use in programs to provide your own extension of the collection of sub-programs that Extended BASIC offers. The MERGE facility with disk

based systems makes this particularly easy. See Jim Peterson's Tigercub Tips for many further examples.

For our first example let us take one of the more frustrating things that TI did in choosing the set of built-in sub-programs. If you have MiniMemory or Editor Assembler, you know that the system keyscan routine, SCAN, built into the console ROM returns keyboard and joystick information simultaneously, while Extended BASIC forces you to make separate sub-program CALLs, KEY and JOYST, to dig it out. Since these GPL routines are slow it is difficult to write a fast paced game in Extended BASIC that treats keyboard and joysticks on an equal footing, as is done by many cartridge games. On the other hand in games where planning and not arcade reaction is of the essence there is no reason why the player(s) should be forced to make a once and for all choice and not be able to use either at any stage of the game.

The sub-programmers approach to this problem, once it realised that it can be done (and we have seen commercial Extended BASIC games where the writers have not realised this), is to write the game using joysticks, but replacing JOYST by a user defined sub-program JOY which returns the same values as JOYST even when keys are used.

The first step in telling whether keys or joysticks are being used is to check the keys, and if none have been pressed then to check the joysticks. If a key has been pressed then its return, K, has to be processed so that the direction pads embedded in the keyboard split-scan return the corresponding JOYST value. A sub-program along the lines of the one used in TI Extended BASIC does just this.

```
900 SUB JOY(PL,X,Y):: CALL KEY(PL,K,ST):: IF ST=0 THEN
    CALL JOYST(PL,X,Y):: SUBEXIT
910 X=4*((K=4 OR K=2 OR K=15)-(K=6 OR K=3 OR K=14))
920 Y=4*((K=15 OR K=14 OR K=0)-(K=4 OR K=5 OR K=6))
930 SUBEND
```

PL is the player (left or right joystick or side of the split keyboard) number and is unaltered by the procedure. The simple minded approach for converting K to (X,Y) values by using the Extended BASIC logic operators (one of the more annoying omissions from console BASIC) seems to work as well as any. The sub-program as written checks the keys first but balances this out by putting the processing load on the key return.

This is as good a time as any to sharpen your own skills by working out alternative versions of this procedure, and also by writing one for mocking up a substitute CALL KEY routine to return direction pad values even if a joystick is used.   o

An example to clarify matters:
To change "12ABCD789" into "123456789":
where C$="12ABCD789":
CALL REPLACE(C$,3,4,"3456")

Form: CALL REPLACE(startstring$, start_pos, length, insert$)

Code:

```
27540 SUB REPLACE(T$,C,L,R$)
27550 IF S<1 OR S>LEN(T$) OR L<1 OR S+L-1>LEN(T$) OR
      T$="" THEN SUBEXIT
27560 IF LEN(R$)>L THEN R1$=SEG$(R$,1,L) :: GOTO 27580
27570 IF LEN(R$)<L THEN R1$=R1$&RPT$(" ",L-LEN(R$)) ELSE
      R1$=R$
27580 F$=SEG$(T$,1,S-1) ::
      L$=SEG$(T$,S+L,LEN(T$)-(S+L-1))
27590 T$=F$&R1$&L$
27600 SUBEND
```
                =======END======   o

TI-Base Tutorial #3

by Martin Smoley, NorthCoast 99'ers

Copyright 1988 by Martin A. Smoley

I am reserving the copyright on this material, but I will allow the copying of this material by anyone under the following conditions. (1) It must be copied in its entirety with no changes. (2) If it is retyped, credit must be given to myself and the NorthCoast 99ers, as above. (3) The last major condition is that there may not be any profit directly involved in the copying or transfer of this material. In other words, Clubs can use it in their newsletters and you can give a copy to your friend as long as it is free.

```
SET TALK OFF
*                      9/12/88  WHILE
* Command File WHTST3           ENDWHILE
*    Save as WHTST3/C           DOCASE
*                               ENDCASE
CLOSE ALL
 LOCAL ? N 2 0
 LOCAL SEL N 2 0
REPLACE ? WITH 0
  WHILE .NOT. (?)
    CLEAR
    WRITE 2,8,"** Make A Selection **"
    WRITE 4,10,"> 0 < To Quit CF"
    WRITE 6,10,"> 1 < DO WHTST4"
    WRITE 8,10,"> 2 < DO INITPR"
    WRITE 10,10,"> 3 < SEL. THREE"
    WRITE 12,10,"> 4 < SEL. FOUR"
    WRITE 22,4,"Enter 0-4"
    READ 22,15,SEL
    WRITE 22,3,"               "
     DOCASE
       CASE SEL = 0
        WRITE 18,10,"Have a nice day"
        REPLACE ? WITH 1
        BREAK
       CASE SEL = 1
         WRITE 18,15,"Number 1"
         DO WHTST4
         BREAK
       CASE SEL = 2
         WRITE 18,15,"Number 2"
         DO INITPR
         BREAK
       CASE SEL = 3
         WRITE 18,15,"Number 3"
         BREAK
       CASE SEL = 4
         WRITE 18,15,"Number 4"
         BREAK
      ENDCASE
   ENDWHILE
 CLEAR
CLOSE ALL
 SET TALK ON
RETURN
```

This month I will attack the DOCASE, ENDCASE and a couple of additional tidbits. This tutorial will finish off almost all of the major points in the TIB Manual. Hopefully at that point you will have some idea what is going on with this language. Future tutorials will be less wordy and contain more intricate programming. I will also try to touch on the items we did not cover in the manual so far.

The CF named WHTST3 is listed above. It is the beginning of TIB's menu capability and many other things which can be handled by combinations of WHILE, DOCASE and IF statements. Let us hit the high points.

LOCAL ? N 2 0, initializes a local variable named "?". I named it ? because I could not come up with a good name for it, as in SEL which stands for selection. ? is a Numeric Variable with a size of 2 and 0 decimal places. A Numeric Variable can also be used as a Boolean Operator (if you are careful). A Boolean Operator is just something that transmits a TRUE or FALSE to TIB. To TIB and to many many programs and computers, FALSE is represented by a Zero "0", and TRUE is represented by a one "1". When we REPLACE ? WITH 0, ? is both a Numeric Variable which contains the value 0 and a Boolean Operator which represents FALSE.

WHILE statements need Boolean Operators to decide whether to execute the lines following the WHILE statement or skip them all and go directly to the statement after the ENDWHILE. In this case WHILE .NOT. (?) means WHILE ? is NOT TRUE, do the statements following the WHILE. Because we placed a 0 in ? previously, it is FALSE (or not TRUE), so the WHILE will continue to loop until we change ? to a 1 or TRUE, which you can do in the CASE SELection number 0. If you grasp this logic, you can see why I named it ? and why I said be careful. If you do not grasp the idea, just type things in as you see them. There will be more chances to sort out program logic in the future.

When we enter the WHILE loop we CLEAR the screen and display a menu which can contain anything you wish TIB to do for you. At the bottom of the input screen TIB asks for your selection. Entering a number from 0 to 4 will set the variable SEL equal to that number. TIB then blanks out line 22 on your monitor and goes into the DOCASE routine.

In the DOCASE, TIB goes to the first CASE and compares the value in SEL to the value on the right side of the equal sign. Therefore, if you selected 0 when asked for your choice, TIB would find a TRUE match when it hit the first CASE comparison and would execute the lines between that CASE and the BREAK directly after it. In this case it would display the message "Have a nice day" and REPLACE ? WITH 1, which makes the variable ? TRUE. When TIB hits the BREAK after REPLACE ? WITH 1 it goes to the ENDCASE. In this instance it would then go to the ENDWHILE which sends TIB back to the beginning of the WHILE loop.

This time when we hit the WHILE .NOT. ? the ? equals 1 or TRUE so the WHILE loop does not execute and the program goes to the next directive after the ENDWHILE. "I know that is a roundabout way to get here, but the computer can do it a lot faster than I can explain it."

If you had selected 0, TIB would then finish and leave this CF which would return you to the DP. If, however, you had chosen any other number, TIB would have performed whatever tasks were present between the CASE that matched the SELection and the BREAK that followed it. For example, entering a 2 would DO the CF named WHTST4, or 3 would DO the CF named INITPR. I hope to eventually show you how to put a complete system together that will allow you to maintain and use a membership list for home, club, church or work, using menus and small CFs to do the work for you.

```
CLEAR
*                      9/15/88  WHILE
* Command File WHTST4           ENDWHILE
*    Save as WHTST4/C
WRITE 12,15,"*************"
WRITE 13,15,"*  WHTST4  *"
WRITE 14,15,"*************"
 LOCAL ANS N 3 0
   WRITE 22,1,"  Number of Cycles"
 READ 22,22,ANS
WRITE 22,1,"                    "
  WHILE (ANS > 0)
    WRITE 22,4,"  Cycles Left =",ANS
    REPLACE ANS WITH ANS - 1
  ENDWHILE
WRITE 22,1,"                    "
 CLEAR
RETURN
```

The CF above can be run by selecting number 1 from the menu screen of WHTST3. "Provided you type all this stuff in of course." WHTST4 does not really do a darn thing. When you run it, it asks you to enter a number. It will then start at that number and count down until it hits zero. You should enter a number like 4, 5 or 6 if you do not want to watch this thing counting down for a week. So you are saying to yourselves, why did this nut put this junk in the tutorial. Let us go through it and I will explain.

We initialize the LOCAL ANS as a number. "No big deal here." You enter a number of your choice and then we hit the WHILE loop. In this instance it is written, WHILE (ANS > 0). Take a look at it. It is different from the last one. In this case the (ANS > 0) forms the Boolean Operator. As long as ANS holds a number which is greater than zero (0) the result is a TRUE, and as long as the WHILE has a TRUE stamp on it ·everything inside the loop is executed.

Now inside the loop we find REPLACE ANS WITH ANS - 1. This is an accumulator. Each time the loop is executed you can add a quantity to your accumulator, or as in this case you can subtract a quantity from your accumulator. This is a lot like a FOR NEXT loop in Extended BASIC. You enter a quantity for · ANS. Each time the loop is executed 1 is subtracted from ANS. When ANS reaches 0 the loop is discontinued. I tried to show you this idea in its simplest form so you might have an easier time grasping the concept.

The CF below is a real application of this idea. It is slightly stripped down so it would not take up too much space, but it works and it is usable. It uses our old data base named TNAMES. When you run it (DO WHTST5), it opens TNAMES and displays the first record in the file. It then asks you how many labels you want. If you enter a zero (0), it MOVEs to the next record in the data base and puts that one on the screen for you with the same question. If you enter a quantity greater than 0, like 4, it will print out 4 labels and then go to the next record. "I hope you get the idea."

One thing about it that is slightly odd. The Emphasize command I placed in the first part of REPLACE TEMP1 WITH "<27>E   " comes up as an E on the screen at the top left of the display. (Note that a number in < > means that number as an ASCII code.) Just ignore it, the CF works fine.

I whipped WHTST5 up by loading WHTST4 into the FunnelWeb Assembler Editor just as it looks above. Then I pulled in the old CF named LBLS1 from Tutorial 1. With both CFs loaded I did a bunch of M)oving and C)opying. I typed over a few things and I had it. It took me about 45 minutes to smash it together and work out the bugs.

```
CLEAR
SET TALK OFF
SET RECNUM OFF
SET HEADING OFF
SET LINE=80
SET PAGE=000
* Command File WHTST5   10/07/88
*   Save as WHTST5/C
* USE TNAMES and Print Multiple Labels
WRITE 11,15,"* Multiple *"
WRITE 13,15,"*  Label    *"
WRITE 15,15,"* Program   *"
LOCAL TEMP1 C 40
LOCAL TEMP2 C 40
LOCAL TEMP3 C 40
LOCAL BLNK C 1
USE TNAMES
TOP
WHILE .NOT. (EOF)
    CLEAR
    REPLACE TEMP1 WITH "<27>E          ";
      | " Exp. Date " | XP
      WRITE 10,3,TEMP1
    REPLACE TEMP2 WITH TRIM(FN) | " ";
         | MI | " " | LN
      WRITE 12,3,TEMP2
      WRITE 14,3,SA
    REPLACE TEMP3 WITH TRIM(CT) | " ";
         | ST | ". " | ZP
      WRITE 16,3,TEMP3
  LOCAL ANS N 3 0
   WRITE 22,1,"  Number of Labels"
   READ 22,22,ANS
WRITE 22,1,"                      "
   WHILE (ANS > 0)
      PRINT TEMP1
      PRINT BLNK
      PRINT TEMP2
      PRINT SA
      PRINT TEMP3
```

```
   PRINT BLNK
   REPLACE ANS WITH ANS - 1
      WRITE 22,4,"   Cycles Left =",ANS
WRITE 22,4,"                      "
   ENDWHILE
   MOVE
  ENDWHILE
 CLEAR
CLOSE ALL
SET RECNUM ON
SET HEADING ON
SET TALK ON
RETURN
```

I am not telling you how, or how fast I created a CF to make you feel bad. I am doing it to demonstrate that you should develop a logical procedure, and maintain good programming habits. When you do not understand something about a language, create a small CF or program to test your ideas. Make your new test CF as complete as possible as far as house keeping is concerned. This will allow you to use your work as part of another larger program when the idea has been fully developed and you see it more clearly. Also, I had a lot less trouble finding bugs in WHTST4 than I would have in WHTST5 do to WHTST5's more complicated nature.

I have also noticed, as I get deeper into this, that TI-Base is slow. It is not always slow, but the more you ask it to do the slower it gets. This will not deter me from using or recommending that others use TI-Base, because what it can do outweighs this drawback. As I write CFs I will attempt to keep this in mind and attempt to minimize unneeded repetition. Let us take a quick look at WHTST5 for some ideas on this matter. "I would also like to add that some of this is theoretical and that any speed difference may vary greatly depending on your system."

We have two WHILE loops in WHTST5. The smaller WHILE which runs from WHILE (ANS > 0) to the first ENDWHILE (which is 10 lines down) is nested inside a larger WHILE loop. The larger WHILE loop runs from WHILE .NOT. (EOF) to the ENDWHILE directly following the MOVE statement. The number of times the small loop will run, and print out labels, is determined by your answer to how many labels you want. The larger WHILE loop will run until it reaches the E)nd O)f F)ile (or data base) that is in use at the time. "Here is the time saver."

Because the inner WHILE loop may run many times, depending on the number of labels you request, I have attempted to remove any non-essential code (program lines) from this loop. You will notice that I loaded all my variables (REPLACE TEMP1 WITH '''''), etc., before I got into this loop. Therefore, TI-Base did not have to perform that task 100 times if I said I wanted 100 labels. There is one other consideration you must make in this situation. In order to do this it was necessary to create the variables TEMP1, TEMP2 and TEMP3. You should not get carried away and use up all of TI-Base's variable space. You must balance the idea of speed with the lack of massive free memory space. In this case we have enough memory space to do the job and these variables will be thrown away and the space will be freed up when this CF terminates with the RETURN statement.

You will probably have your first real problem with memory space when you run a CF which runs another CF which in turn runs another CF. If each CF initializes some variables of its own, by the time you get to the last one the variable space will all be used up. I will point this out again later when it comes up in the natural scheme of things. We better move along to some items with more immediate use potential.

On this page you will find INITPR which was a selection from the menu of WHTST3. This CF is nothing new and spectacular, but I find it useful. It is the same as WHTST3, but it has been modified to send printer control codes. I think I have covered the ideas in INITPR previously. I would like to point out (??). If this CF is run from WHTST3 as selection 2 you must not re-use the (?) as a variable. If you do use the same name here and you change the value of ?, you may cause unexpected things to happen when TI-Base returns to WHTST3 and carries ? with it. Also, if you wish to use INITPR as a stand alone CF, you may want to turn the

```
RECNUM and stuff back on at the end of the CF.
SET TALK OFF
SET HEADING OFF
SET RECNUM OFF
*            09/12/88
* Command File INITPR
*    Save as INITPR/C
*
CLOSE ALL
 LOCAL ?? N 2 0
 LOCAL SEL N 2 0
 LOCAL CNTRL C 2
REPLACE ?? WITH 0
  WHILE .NOT. (??)
   CLEAR
     WRITE 1,6,"** Send Printer Controls **"
     WRITE 2,9,"** Make a selection **"
     WRITE 4,10,"> 0 < Leave this CF"
     WRITE 6,10,"> 1 < Emphasized on"
     WRITE 8,10,"> 2 < Italics on"
     WRITE 10,10,"> 3 < Condensed on"
     WRITE 12,10,"> 4 < Doublestrike"
     WRITE 14,10,"> 5 < RESET Printer"
     WRITE 22,4,"Enter 0-5"
     READ 22,15,SEL
     WRITE 22,3,"              "
     DOCASE
       CASE SEL = 0
        CLOSE ALL
        CLEAR
        WRITE 18,12,"Do Not Turn Your"
        WRITE 20,12," Printer Off."
        REPLACE ?? WITH 1
        BREAK
       CASE SEL = 1
        REPLACE CNTRL WITH "<27>E"
        PRINT CNTRL
        BREAK
       CASE SEL = 2
        REPLACE CNTRL WITH "<27>4"
        PRINT CNTRL
        BREAK
       CASE SEL = 3
        REPLACE CNTRL WITH "<15> "
        PRINT CNTRL
        BREAK
       CASE SEL = 4
        REPLACE CNTRL WITH "<27>G"
        PRINT CNTRL
        BREAK
       CASE SEL = 5
        REPLACE CNTRL WITH "<27>@"
        PRINT CNTRL
        BREAK
     ENDCASE
   ENDWHILE
  CLEAR
RETURN
```

INITPR is merely another demonstration of what you
can do with DOCASE and WHILE statements. Many of the
ideas I have presented in my tutorials can be done in
other ways. Some of the other ways may turn out to be
more efficient or more convenient to use. I still
consider myself to be a beginner at TI-Base so I
anticipate changing my ideas on how to optimize program
power and minimize program run time. As I stated in the
last tutorial, I would appreciate letters or notes from
TI-Base users with comments, tips or questions on this
subject. I do not have the time to write back to you.
In many cases I find myself rushing to the last minute
before the newsletter deadline to finish the months
tutorial.

```
100 ! ***** TIB->DV/80 **********
110 ! (C) 1988 Martin A. Smoley
120 !
130 ! Extended BASIC program to read TI-Base I/F40 files
140 ! and write D/V80 files for TI-Writer or FunnelWeb.
150 !
160 ! You must add one blank space to the beginning
170 ! of every line in the TI-Base I/F40 file.
180 ! After transfer, check all lines for any
190 ! missing characters, especially the end.
200 !
```

```
210 CALL CLEAR :: CALL SCREEN(6)
220 PRINT " Enter INPUT File ALL CAPS"
230 PRINT " Example: DSK1.OPERATOR/C"
240 INPUT "          ":IN$
250 IF LEN(IN$)>12 THEN OUT$=SEG$(I$,1,12)&"*DV"
260 IF LEN(IN$)<13 THEN OUT$=IN$&"*DV"
270 PRINT "OUT File= ";OUT$ :: PRINT
280 INPUT " Is that OK Y/N ":ANS$
290 IF ANS$="N" OR ANS$="n" THEN 210
300 OPEN #1:IN$,INTERNAL,FIXED 40,INPUT :: LN=40
309 ! OPEN #1:IN$,DISPLAY ,VARIABLE 80,INPUT :: LN=80
310 OPEN #2:OUT$,DISPLAY ,VARIABLE 80,OUTPUT
320 IF EOF(1)THEN CLOSE #1 :: CLOSE #2 :: GOTO 480
330 !
340 INPUT #1:A$
350 PRINT A$
360 FOR I=1 TO LN
370 T$=SEG$(A$,I,1)
380 ON ERROR 440
390 IF ASC(T$)>126 THEN T$=" "
400 IF ASC(T$)<32 THEN T$=" "
410 B$=P$
420 IF I<1 THEN P$=T$ ELSE P$=B$&T$
430 NEXT I
440 PRINT #2:P$
450 P$=""
460 !
470 GOTO 320
480 CALL CLEAR :: PRINT " *** FINISHED ***": : : : :
490 INPUT " Quit Program Y/N ":QT$
500 IF QT$="Y" OR QT$="y" THEN STOP ELSE GOTO 210
510 ! ***** TIB->DV/80 **********
520 END
```

I have also been recommending the use of FunnelWeb
in the non-word wrap mode. I have had some problems
with this procedure. I figured if I was having a
problem, someone else must be having the same problem.
The problem is hidden characters in the CF. In most
cases I am in a hurry to produce code (write programs or
CFs). Many times I jump into the wrong editor mode and
start typing. In many instances this will not be a
problem. In FunnelWeb pressing CTRL[0] will throw you
into non word wrap mode, which is the same as the
Assembler Editor. However, if you hit the CTRL key and
some other key at the same time while you are still in
word wrap mode, you can insert characters which are
invisible on the screen but do crazy things when the CF
is run. At one point I wasted more precious time than I
could afford trying to find one of these invisible
little land mines. I remembered a little Extended BASIC
program I had written for another task several weeks
earlier. At that time I wanted to convert several
Command Files (CFs) to DV/80 files so I could print them
out and study them more carefully. The program I wrote
was TIB->DV/80 which is listed on this page. I think
some of you may get some use out of it. As is is now,
it will read an I/F40 file (like a CF), and write it to
a D/V80 file for FunnelWeb. There is one thing you must
do first. A control code in CFs causes the loss of the
first character in every line. You can overcome this by
loading the CF into TI-Base using MODIFY COMMAND
(filename). Pressing FCTN[2] for insert mode, which
stays on until you press FCTN[2] again. Then add one
blank space to the beginning of every line. When you
run my program the blank space will be lost instead of
something you need. If you want to run the program on a
DV/80 file, remove the exclamation point from line 309
and place one in front of line 300. Adding the space is
only necessary with D/F40 files, not D/V80. The program
will check every character in the file and will kick out
all characters below 32 or above 126. That includes
those invisible land mines in your CF. Unfortunately
you will have to replace any printer controls.

Well, I am running out of space and my mind is
shot, so I would like to say a couple more things and
this one is finished. First I would like to thank the
people of the NorthCoast 99'ers for allowing me the
space in their newsletter to write this tutorial, and a
lot of miscellaneous articles in the past. The
NorthCoast members are a great group of people. I would
like to throw in the fact that any TI99/4A owner in the
continental US can join the NorthCoast 99'ers for only

Programming c99

part 3, by Craig Sheehan

Sub-programs are the most important element of 'C' that needs to be learnt. They make program writing easier and quicker since sections of a program are broken down into smaller and easier to manage sections. In addition, once a subprogram has been compiled, there is no need to re-compile it; the sub-programs required to run a program are simply loaded along with the main program. It is possible to build up a library of sub-programs so that commonly used functions need not be written every time you begin a new program.

Already we have seen one example of a sub-program, the PRINTF command. PRINTF is not part of the 'C' language. Indeed, 'C' does not have a print statement at all! It is up to the programmer to supply their own sub-program to carry out this task. Fortunately, somebody else has already written one, eliminating the need to write one each time you start a program. Other examples of sub-programs that are on the 'c99' disk include the file handling and graphics libraries.

As an example of a typical sub-program, Figure 6a contains two sub-programs that find the minimum and maximum values from a set a three integers. Figure 6b contains a sample program that makes use of these sub-programs .

```
/* Sub-programs to find minimum and maximum values */
/* from a list of three integers.                   */
/*                                                   */
/* MIN: returns the value of the smallest number    */
/*      in the list.                                 */
/* MAX: returns the largest number in the list.      */
/*                                                   */
/* Source: TND. October, 1989.                       */

entry min, max;

min(a, b, c)
  int a, b, c;

  { int minab;
    minab = ( (a < b) ? a : b );
    return( (minab < c) ? minab : c);
  }

max(a, b, c)
  int a, b, c;

  { int maxab;
    maxab = ( (a > b) ? a : b );
    return( (maxab > c) ? maxab : c);
  }
```
          Figure 6a - The ordering sub-programs.

```
/* Program to find the maximum and minimum values */
/* from the list 1, 7 and 3.                       */

extern printf();
extern min();
extern max();

main()
  { printf("Maximum: %d\n", max(1, 7, 3) );
    printf("Minimum: %d\n", min(1, 7, 3) );
    exit(0);
  }
```
  Figure 6b - A sample program using 'max' and 'min'.

Before describing how these programs operate, first a note on how to compile and run them. First type in Figures 6a and 6b separately, saving to different filenames, MIN_MAX/C and ORDER/C, say. Then compile each one with the 'c99' compiler, using output filenames MIN_MAX/S and ORDER/S for example. Finally, assemble these files to produce object files MIN_MAX/O and ORDER/O respectively. To run the program after carrying out this process, from the load and run option of the loaders, use the filenames: DSK1.CSUP, DSK1.PRINTF, DSK1.MIN_MAX/O and DSK1.ORDER/O. Press enter and then type in START as the program name. If you have carried the procedure out correctly, the numbers 7 and 1 should appear next the words "Maximum" and "Minimum" respectively.

We will now turn to a description of the program. Notice that the program has been deliberately broken into two distinct parts and is even compiled as separate parts. This done for a good reason, although it may not be apparent from a simple example like this one. The reason is as follows: next time you require to find the maximum or minimum of a list of three numbers in one of your programs, there is no need to write or copy a function to do this task. Simply load in the file containing these sub-programs when you load the main program and they will be available for use. As you program, it will become natural to do this as many sub-programs are commonly used in almost all programs.

The first few lines of the program in Figure 6a contains comments on what the program does, and its source. Do not underestimate the importance of this. Whilst it does not affect the running of the program, they make it clear in plain english what the sub-program does. More complex sub-programs should also contain some notes on the algorithm used. This makes debugging easier, especially for a third party. The source is included in case more information concerning the routine than is provided in the comment is required. Make this a habit!

The next program line contains an "entry" statement. This tells the compiler that the sub-programs that follow the "entry" are to be made available to outside routines. In this way the main program can access the sub-programs "min" and "max". It should be noted that "entry" is not part of standard 'C', and should be removed if you wish to compile these sub-programs on a computer other than a TI99/4A.

Each sub-program is defined by placing its name, followed by the list of parameters that are required in parentheses. Parameters are values that are passed to the sub-program in order for it to carry out its job. In this case the parameters are three integers, which will be referred to by the names 'a', 'b' and 'c' inside the sub-program. If no parameters are required, such as in "main", nothing is placed between the parentheses. Each parameter's type is defined immediately after this in the same manner as an ordinary variable declaration. In Figure 6a, this tells the compiler that each of the parameters is an integer.

The way in which the maximum and minimum number is determined involves a new command, which is buried within brackets and so may seem a little confusing at first. Consider the "min" sub-program. The statement:
          minab = ((a < b) ? a : b)
is evaluated by setting minab to the value of (a < b) ? a : b. First (a < b) is tested. If the value of this expression is non-zero (i.e it is true), then 'a' is value of the statement. If the condition is false, then 'b' is returned. Assuming 'a' is smaller than 'b', then the entire expression becomes:
          minab = (a)
which simply is equivalent to setting 'minab' to 'a'. The trick to understanding this statement is that the innermost brackets are calculated first.

The "return" statement sends an integer back to the main program. As an example, "return 1" would return the value 1 in place of the call to the sub-program. In the case of "min", the brackets are calculated first, which in similar way to 'minab', finds the smaller of 'minab' and 'c', and hence the smallest value of the entire set.

A final point to note is the "extern" statements in Figure 6b. Since the sub-programs are compiled separately to the main program, the compiler has no way of knowing that "min", "max" and even "printf" are to be

Tips from the Tigercub #35

by Jim Peterson, Tigercub Software, USA

Copyright 1986
156 Collingwood Ave., Columbus, OH 43213

Distributed by Tigercub Software to TI99/4A Users Groups for promotional purposes and in exchange for their newsletters. May be reprinted by non-profit users groups, with credit to Tigercub Software.

Over 130 original programs in BASIC and Extended BASIC, available on cassette or disk, only $3 each plus $1.50 per order for packing and mailing. Entertainment, education, programmer's utilities.

Descriptive catalog $1, deductable from your first order.

Tips from The Tigercub, a full disk containing the complete contents of this newsletter numbers 1 through 14, 50 original programs and files, just $15 postpaid.

Tips from the Tigercub volume 2, another disk full, complete contents of numbers 15 through 24, over 60 files and programs, also just $15 postpaid. Or, both for $27 postpaid.

Nuts & Bolts (No. 1), a full disk of 100 Extended BASIC utility subprograms in merge format, ready to merge into your own programs. Plus the Tigercub Menuloader, a tutorial on using subprograms, and 5 pages of documentation with an example of the use of each subprogram. All for just $19.95 postpaid.

Nuts & Bolts No. 2, another full disk of 108 utility subprograms in merge format, all new and fully compatible with the last, and with 10 pages of documentation and examples. Also $19.95 postpaid, or both Nuts & Bolts disks for $37 postpaid.

Tigercub Full Disk Collections, just $12 postpaid! Each of these contains either 5 or 6 of my regular $3 catalog programs, and the remaining disk space has been filled with some of the best public domain programs of the same category. I am NOT selling public domain programs - my own programs on these disks are greatly discounted from their usual price, and the public domain is a FREE bonus!

```
TIGERCUB'S BEST              PROGRAMMING TUTOR
PROGRAMMER'S UTILITIES  BRAIN GAMES  BRAIN TEASERS
BRAIN BUSTERS!   MANEUVERING GAMES    ACTION GAMES
REFLEX AND CONCENTRATION         TWO-PLAYER GAMES
KID'S GAMES MORE GAMES WORD GAMES  ELEMENTARY MATH
MIDDLE/HIGH SCHOOL MATH       VOCABULARY AND READING
MUSICAL EDUCATION         KALEIDOSCOPES AND DISPLAYS
```

For descriptions of these send a dollar for my catalog!

The April MICROpendium had a rather slow routine to count the number of words in a D/V text file. I think the following will be much faster. It ignores any lines beginning with a period (TI-Writer formatter commands), otherwise counts each cluster of characters followed by a space, plus the last cluster on the line.

```
10 !WORDCOUNT by Jim Peterson
100 DISPLAY AT(12,1)ERASE ALL:"INPUT FILENAME? DSK" ::
    ACCEPT AT(12,20):F$ :: OPEN #1:"DSK"&F$,INPUT
110 A=1 :: LINPUT #1:M$ :: IF ASC(M$)=46 THEN 130
120 X=POS(M$," ",A):: IF X=0 THEN 130 :: IF X=A THEN
    A=X+1 :: GOTO 120 ELSE F=1 :: C=C+1 :: A=X+1 :: GOTO
    120
130 C=C+F :: F=0 :: IF EOF(1)<>1 THEN 110 :: CLOSE #1 ::
    DISPLAY AT(12,1)ERASE ALL:"APPROXIMATELY "&STR$(C)&"
    WORDS"
```

Have you tried those black write-protect tabs, made of a material similar to electrical tape? They do not become dog-eared from bumping against the drive slot, and do not leave the disk sticky when you remove them.

```
100 !TIGERCUB GRAPHPRINT by Jim Peterson
110 !Will output to printer a line graph of 31 items of
    data, as for instance the temperature for each day
    of a month
120 !Values must be positive integers within a range of
    75 from minimum to maximum
```

```
130 M$=RPT$("* ",65):: DIM T$(31),D$(75):: MN=10000
140 DISPLAY AT(12,1)ERASE ALL:"Input data - maximum
    31":"items.  Enter to finish"
150 FOR X=1 TO 31 :: DISPLAY AT(14,1):X;TAB(4);CHR$(1)::
    ACCEPT AT(14,4)VALIDATE(DIGIT)SIZE(-5)BEEP:T$(X)::
    IF T$(X)=CHR$(1)THEN X=X-1 :: GOTO 170
160 T=VAL(T$(X)):: MX=MAX(MX,T):: MN=MIN(MN,T):: NEXT X
170 RN=MX-MN :: IF RN>75 THEN PRINT "EXCEEDS MAXIMUM
    RANGE OF 75" :: STOP
180 IF MX>75 THEN AD=MX-75
190 OPEN #1:"PIO",VARIABLE 132 ::
    PRINT #1:CHR$(15);CHR$(27);CHR$(51);CHR$(12)::
    PRINT #1:RPT$("_",132)
200 DISPLAY AT(12,1)ERASE ALL:"Wait, please...":
    :"......this takes time"
210 LM=LEN(STR$(MX)):: FOR J=1 TO 75 :: J$=STR$(76+AD-J)
220 IF J>66+AD THEN J$=J$&" "
230 IF J/2=INT(J/2)THEN D$(J)=RPT$(" ",LM)&
    SEG$(M$,1,132-LM) ELSE D$(J)=J$&SEG$(M$,1,132-LM)
240 NEXT J :: PRINT #1:RPT$(" ",LM)&SEG$(M$,1,132-LM)
250 J=1 :: T=VAL(T$(J))-AD :: T=76-T ::
    D$(T)=SEG$(D$(T),1,J*4+4)&CHR$(239)&
    SEG$(D$(T),J*4+6,255):: J=J+1
260 T2=T :: T=VAL(T$(J))-AD :: T=76-T :: FOR N=T2 TO T
    STEP (T2>T)+ABS(T>=T2):: D$(N)=SEG$(D$(N),1,J*4+2)&
    CHR$(253+(T<T2))&SEG$(D$(N),J*4+4,255):: NEXT N
270 J=J+1 :: D$(T)=SEG$(D$(T),1,J*4)&CHR$(239)&
    SEG$(D$(T),J*4+2,255):: IF J<=X THEN 260
280 FOR J=1 TO 75 :: PRINT #1:D$(J):: NEXT J :: PRINT #1
290 T=8 :: FOR J=1 TO 31 :: PRINT #1:TAB(T);STR$(J);::
    T=T+4 :: NEXT J
```

When you are analyzing an Extended BASIC program, or modifying it, it is often easier to work with single statement lines. This program will break all multi-statement lines into single statement lines, except when they are followed by IF or ELSE. When you are finished modifying, a Compactor or Smash program can be used to compact it again.

```
100 !DECOMPACTER by Jim Peterson
110 DISPLAY AT(3,5)ERASE ALL:"TIGERCUB DECOMPACTER": :"
    Program must first be -": :"RES 100,100": :"SAVE
    DSK(filename),MERGE"
120 DISPLAY AT(12,1):"INPUT FILENAME?":"DSK" ::
    ACCEPT AT(13,4):IF$
130 DISPLAY AT(12,1)ERASE ALL:"OUTPUT FILENAME?":"DSK"
    :: ACCEPT AT(13,4):OF$
140 OPEN #1:"DSK"&IF$,INPUT,VARIABLE 163 :: OPEN #2:
    "DSK"&OF$,OUTPUT,VARIABLE 163 :: LN=100
150 LINPUT #1:M$ :: P=POS(M$,CHR$(130),3):: IF P=0 THEN
    PRINT #2:M$ :: GOTO 270
160 A$=SEG$(M$,1,P-1):: IF POS(A$,CHR$(129),1)<>0 OR
    POS(A$,CHR$(132),1)<>0 THEN PRINT #2:M$ :: GOTO 270
170 PRINT #2:A$&CHR$(0)
180 AN=LN+1 :: GOSUB 280
190 M$=SEG$(M$,P+1,255)
200 P=POS(M$,CHR$(130),1)
210 IF P=0 THEN PRINT #2:LN$&M$ :: GOTO 270
220 A$=SEG$(M$,1,P-1)
230 IF POS(A$,CHR$(129),1)<>0 OR POS(A$,CHR$(132),1)<>0
    THEN PRINT #2:LN$&M$ :: GOTO 270
240 PRINT #2:LN$&A$&CHR$(0)
250 AN=AN+1 :: GOSUB 280
260 GOTO 190
270 LN=LN+100 :: IF EOF(1)<>1 THEN 150 ELSE CLOSE #1 ::
    CLOSE #2 :: END
280 LN$=CHR$(INT(AN/256))&CHR$(AN-256*INT(AN/256))::
    RETURN
```

I still think of the TI99/4A as a home computer, and I still think that the home computer is an invaluable educational tool, but I guess not many folks agree with me. I had thought of writing full disks of a progressive series of lessons on one subject, but my present two full disks of math education have sold a combined total of 7 copies in 7 months, so that would obviously be a waste of time.

I had written this next program for that purpose and I guess it is no use wasting it, so -

```
100 CALL CLEAR :: CALL TITLE(5,"TAKE AWAY")!by Jim
    Peterson
```

```
110 DISPLAY AT(3,10):"COPYRIGHT":TAB(10);"TIGERCUB
    SOFTWARE":TAB(10);"FOR FREE":TAB(12);".
    DISTRIBUTION":TAB(11);"SALE PROHIBITED"
120 CALL PEEK(-28672,A@):: IF A@=0 THEN 150
130 DATA FINE,NO,GOOD,UHOH,RIGHT,TRY AGAIN,YES,THAT IS
    NOT RIGHT
140 FOR J=1 TO 4 :: READ RIGHT$(J),WRONG$(J):: NEXT J
150 FOR D=1 TO 1000 :: NEXT D :: CALL DELSPRITE(ALL)
160 CALL CLEAR :: CALL CHAR(95,"FFFF"):: CALL MAGNIFY(2)
    :: RANDOMIZE :: CALL SCREEN(14):: FOR SET=5 TO 8 ::
    CALL COLOR(SET,16,1):: NEXT SET
170 CALL CHAR(120,"E700420018007E0000E700420099423CE
    700420099423C00E7004218003C4200")
180 CALL CHAR(124,"0E0004010007080070002080000E01000")
190 DISPLAY AT(3,10):"TAKE AWAY" :: CALL CHAMELEON
200 CALL COLOR(14,2,2):: CALL HCHAR(4,4,143,2)::
    CALL HCHAR(5,4,143,2)::
    CALL SPRITE(#25,120,11,25,25)
210 T=T+1 :: N=1-(T>5)-(T>15):: G=10-(T>5)*80-(T>15)*810
    :: H=0-(T>5)*10-(T>15)*90
220 X=INT(G*RND+H):: Y=INT(G*RND+H):: IF Y>X THEN TT=X
    :: X=Y :: Y=TT
230 IF X=X2 OR Y=Y2 THEN 220 :: X2=X :: Y2=Y :: Z=X-Y
240 GOSUB 250 :: GOTO 210
250 GOSUB 260 :: GOSUB 280 :: GOSUB 310 :: FOR D=1 TO
    200 :: NEXT D :: CALL DELSPRITE(ALL)::
    DISPLAY AT(18,1):: CALL CHAMELEON ::
    CALL SPRITE(#25,120,11,25,25):: RETURN
260 FOR J=1 TO LEN(STR$(X)):: :: FOR J=1 TO
    A(J)=VAL(SEG$(STR$(X),J,1)):: NEXT J :: FOR J=1 TO
    LEN(STR$(Y)):: B(J)=VAL(SEG$(STR$(Y),J,1)):: NEXT J
270 FOR J=1 TO LEN(STR$(Z))::
    C(J)=VAL(SEG$(STR$(Z),J,1)):: NEXT J ::
    W=LEN(STR$(Z))-LEN(STR$(X)):: RETURN
280 R=96 :: CC=96 :: FOR J=1 TO N :: CALL
    SPRITE(#J,48+A(J),11,R,CC):: CC=CC+16 :: NEXT J
290 R=116 :: CC=96 :: FOR J=1 TO N ::
    CALL SPRITE(#4+J,48+B(J),11,R,CC):: CC=CC+16 :: NEXT
    J
300 CALL HCHAR(18,12,95,N*3) :: CC=CC-16 :: RETURN
310 R=140 :: FOR J=LEN(STR$(Z))TO 1 STEP -1 ::
    IF LEN(STR$(X))=1 THEN M=CC :: GOTO 330
320 FOR M=CC TO CC+8 ::
    CALL LOCATE(#J-W,96,M,#J+4-W,116,M):: NEXT M
330 IF A(J-W)>=B(J-W)THEN 360 ::
    CALL SPRITE(#28,49,16,96,M-9)
340 IF F3=1 THEN 360 :: F1=1 :: A(J-W-1)=A(J-W-1)-1 ::
    IF A(J-W-1)<0 THEN A(J-W-1)=9 :: F2=1 ::
    A(J-W-2)=A(J-W-2)-1
350 CALL SPRITE(#22,48+A(J-W-1),16,80,M-24):: IF F2=1
    THEN CALL SPRITE(#21,48+A(J-W-2),16,80,M-40)
360 CALL SPRITE(#27,45,16,116,M-12)
370 CALL SPRITE(#20,63,11,R,M)
380 CALL KEY(3,K,ST):: IF ST<1 OR K<48 OR K>57 THEN
    CALL PATTERN(#20,32):: CALL PATTERN(#20,63):: GOTO
    380
390 CALL DELSPRITE(#20,#28)::
    CALL SPRITE(#12+J,K,11,R,M)
400 IF K-48<>C(J)THEN GOSUB 450 :: CALL DELSPRITE(#12+J)
    :: F3=1 :: GOTO 330
410 CALL DELSPRITE(#27):: IF F1=1 THEN 420 ELSE IF F2=1
    THEN 430 ELSE 440
420 F1=0 :: CALL DELSPRITE(#J-W-1):: FOR P=80 TO 96 ::
    CALL LOCATE(#22,P,M-24):: NEXT P ::
    CALL SPRITE(#J-W-1,48+A(J-W-1),16,96,M-24)::
    CALL DELSPRITE(#22):: GOTO 440
430 F2=0 :: CALL DELSPRITE(#J-1-W):: FOR P=80 TO 96 ::
    CALL LOCATE(#21,P,M-24):: NEXT P ::
    CALL SPRITE(#J-1-W,48+A(J-1-W),16,96,M-24)::
    CALL DELSPRITE(#21)
440 CC=CC-16 :: NEXT J :: GOSUB 480 :: F3=0 :: RETURN
450 DATA 123,124,125,123,124,125,123,120
460 IF A@=0 THEN 470 :: CALL SAY(WRONG$(INT(RND*4+1)))
470 RESTORE 450 :: FOR JJ=1 TO 8 :: READ P ::
    CALL PATTERN(#25,P):: XX=2^250 :: NEXT JJ :: RETURN
480 DATA 121,122,121,122,121,122
490 IF A@=0 THEN 500 :: CALL SAY(RIGHT$(INT(4*RND+1)))
500 RESTORE 480 :: FOR JJ=1 TO 6 :: READ P ::
    CALL PATTERN(#25,P):: XX=2^250 :: NEXT JJ :: RETURN
510 SUB CHAMELEON
520 M$="1800665AC342DB667E188100995AC3A5E78142BD24DB
    66008142992400 7E5AC3A53C241800FFDB5AFF7EFF00991881
    00660018"
530 RANDOMIZE ::
    CALL CHAR(128,SEG$(M$,INT(43*RND+1)*2-1,16))::
    X=INT(14*RND+3)
540 Y=INT(14*RND+3):: IF Y=X THEN 540 ::
    CALL COLOR(13,X,Y)
550 CALL HCHAR(1,2,128,30):: CALL HCHAR(24,2,128,30)::
    CALL VCHAR(1,31,128,96):: SUBEND
560 SUB TITLE(S,T$)
570 CALL SCREEN(S):: L=LEN(T$):: CALL MAGNIFY(2)
580 FOR J=1 TO L :: CALL SPRITE(#J,ASC(SEG$(T$,J,1)),
    J+1-(J+1=S)+(J+1=S+13)+(J>14)*13,J*(170/L),
    10+J*(200/L)):: NEXT J
590 SUBEND
```

When you give your printer instructions, it remembers them until you turn it off. That is why you may find that your letter to Aunt Sally is being printed in double width underlined italics. The solution is found in another gobbledegook paragraph in the Gemini manual; "when (ESC "@") is sent to the printer, the conditions of the printer are initialized."

In plain English, OPEN #1:"PIO" :: PRINT #1:CHR$(27);"@" in your program or CTRL[U], FCTN[R], CTRL[U], SHIFT[2] at the beginning of your TI-Writer text will cancel out any special orders the printer is still remembering and return it to its default conditions.

Here is a bright idea by Scott King in the AVTI UG newsletter. When you load a program in order to modify it, put a reminder of its filename in the first line, such as 1 ! SAVE DSK1.NAME . Then, when you are ready to save it, just list line 1, FCTN[8], use the space bar to erase the 1 !, and <ENTER>.

Memory Full!   Jim Peterson   o

The easiest way would be to .edit the existing DM5 files, and give these different file names (of course in the source code as well). This disk would then become the hospital disk. One could use this in an emergency situation to boot up DM5. Another possibility is to prepare a savable program which would redirect the program flow to another path. In an emergency, one could load this program into the memory, and save it to the DSK1 sub-directory of the hard disk, naming it MDM5. This would then over-write the corrupted file.

The long term solution is to modify and augment DM5 by extensive error trapping routines. If the pathway or a program file is not kosher, then a response by the user to an appropriate prompt, could redirect the program flow to an alternate path.

4. A few more lines on the previously published subject of backup.

30238 sectors of my hard disk are filled with data at present. This is about 7.75 Mbyte. Using the Myarc backup option would require approximately 45 diskettes. The same may be backed up, using manual file by file copying, on 22 DSDD disks. Archiving reduced the required space by approximately 50%, thus 11 DSDD diskettes were needed. The same is saved on 6 diskettes of 80 track, 720k capacity. This is better than 1/7 space utilization! The incredible saving was achieved by using the already supported 80 track option and available software. The moral: the backup routine of DM5 combined with the maximum disk density option and archiving would rationalize the backup routine. Each of the essential building blocks are already available, however, someone has to put it together.

To sum up, the present versions of DM5, sadly lacks error trapping as well as quad density support. DM5 is fair weather software only. Some of the features mentioned in the manual are not supported. Hard disk has advanced the old TI99/4A by light years, hard disk users would be well prepared to spend more on improved disk manager software if such software is clearly superior to that presently available.   o

Extended BASIC Sub-programs

by Stephen Shaw, England

Long before Acorn put sub-program capability into the BBC-Micro, and told anyone who would listen that it was the first BASIC with that capability, TI gave us this in EXTENDED BASIC. Not many users take advantage but here are some ideas for you to consider.

We are talking about the ability to write a program in such a way that you CALL up your own sub-programs, so that at the extreme a program could be:
CALL SETUP
CALL INSTRUCT
CALL PLAY
CALL HIGHSCORES

Get the idea? You are well used to using CALLs as you use them all the time even in TI BASIC, such as CALL CLEAR, CALL HCHAR, and so on! In Extended BASIC you can write your own.

For a very large selection you can do worse than buy copies of Jim Peterson's NUTS & BOLTS disks, but in the meantime here is an excellent selection of routines for you from our member Peter Hutchison. Many thanks Peter.

If you have a disk system, save each of these on disk in MERGE format and then just merge them into any program you wish to use them in.

CALL PUTAT:
    This command uses the whole screen (32x24) rather than the limited 28x24 used by DISPLAY AT.

Form: CALL PUTAT(row, column, string)
Sample: CALL PUTAT(5,1,"HELLO TI USERS EVERYWHERE")

    Code:

```
27000 SUB PUTAT(R,C,T$) 27010 IF R<1 OR C<1 OR T$=""
      THEN SUBEXIT
27020 R=MIN(R,24) :: P=1
27030 IF R>=24 AND C>32 THEN PRINT :: R=24 :: C=1
27040 IF C>32 THEN R=R+1 :: C=1
27050 CALL HCHAR(R,C,ASC(SEG$(T$,P,1)))
27060 IF P>=LEN(T$) THEN SUBEXIT
27070 C=C+1 :: P=P+1 :: GOTO 27030
27080 SUBEND
```

CALL LOWCASE:
    This command defines the TI99/4A small uppercase letters as real lowercase letters.

Form: CALL LOWCASE.
Example: CALL LOWCASE

    Code:

```
27100 SUB LOWCASE
27110 RESTORE 27120
27120 DATA 00000E020E120D0010101E1111111E0000000F10
      10100F0001010F1111110F00
27130 DATA 00000E111E100E0006090081C0808080000000E11
      110F010E10101E1111111100
27140 DATA 04000C04040404040E0200020202020A0410101214
      181412000C04040404040600
27150 DATA 00001A1515151100000001609090909000000E11
      11110E0000001E11111E1010
27160 DATA 00000F11110F0101000016191010100000000F10
      0E011E0008081E0808090600
27170 DATA 0000121212120D0000000011110A0A040000001111
      15150A000000110A040A1100
27180 DATA 00001111110F011E00001F0204081F0000000000
      0000000000000000000000000
27190 FOR C=97 TO 121 STEP 4 :: READ P$ :: CALL
      CHAR(C,P$) :: NEXT C
27200 SUBEND
```

CALL SAVECHAR / CALL LOADCHAR:
    These commands will save the definitions of the characters 96 to 127 into an array PAT$ and then use the array to restore the definitions.

Examples: CALL SAVECHAR(PAT$())   and
    CALL LOADCHAR(PAT$())

    Code:

```
27220 SUB SAVECHAR(PAT$())
27230 FOR C=96 TO 124 STEP 4 :: EL=C/4-23 :: CALL
      CHARPAT(C,P$(EL)) :: NEXT C
27240 SUBEND
27250 SUB LOADCHAR(PAT$())
27260 FOR C=96 TO 124 STEP 4 :: EL=C/4-23 :: CALL
      CHAR(C,P$(EL)) :: NEXT C
27270 SUBEND
```

CALL HSCROLL:
    This command can be used to scroll text along a line on the screen.

Form: CALL HSCROLL(row, column, length_displayed, text$)
Sample: CALL HSCROLL(5,9,10,TEXT$)
Note: Text$ is limited by the system and the code below to a length of 240 characters !

    Code:

```
27300 SUB HSCROLL(R,C,L,M$)
27310 M$="......."&M$&"......."
27320 IF LEN(M$)<L THEN 27310
27330 FOR A=1 TO LEN(M$)-L+1
27340 DISPLAY AT(R,C)SIZE(L):SEG$(M$,A,L)
27350 FOR B=1 TO 15 :: NEXT B :: NEXT A
27360 SUBEND
```

CALL GET:
    This command will return the character of the key pressed, or a null if no key is pressed.

Form: CALL GET(KEY$)

    Code:

```
27380 SUB GET(K$)
27390 CALL KEY(0,K,S)
27400 IF S=0 THEN K$="" ELSE K$=CHR$(K)
27410 SUBEND
```

CALL MOVE:
    Moving sprites with CALL MOTION and using CALL COINC can sometimes be a trifle imprecise. Here is an alternative approach! The command should be nested in a loop if smooth motion between positions is required.

Form: CALL MOVE(sprite, updown, leftright)
Sample: CALL MOVE(1,0,2)

    Code:

```
27440 SUB MOVE(S,R,C)
27450 IF S<1 OR S>28 THEN SUBEXIT
27460 CALL POSITION(#S,RW,CL) :: NR=RW+R :: NC=CL+C
27470 NR=MIN(NR,256) :: NR=MAX(NR,1)
27480 NC=MIN(NC,256) :: NC=MAX(NC,1)
27490 CALL LOCATE(#S,NR,NC)
27500 SUBEND
```

CALL REPLACE:
    This command replaces a part of a string with another.
    Start is the position in the string to be changed where the new TEXT$ is to be inserted and LENGTH is the number of characters to be replaced (eg removed):

The Game of Wit

by Chris Lang, MD USA

The WIT series contains 5 Educational word games which are Fairware. The author is:

Chris Lang,
1906 Jackson Rd,
Baltimore. MD. 21222 USA

The author's requested donation is $10 for User Group members and $15 for non—user group members.

Instructions

Object of the Game:

To be the player with the highest score when the game ends. From one to four players may play.

Equipment Required

1 floppy disk or cassette tape containing The Game of Wit program (included)
1 instruction booklet (included)
1 TI99/4A computer console (not included)
1 colour monitor (or 1 RF modulator and a colour TV set) (not included)
1 Exceltec (or TI) Extended BASIC command module (not included)
1 disk drive (for disk version only) (not included)
1 disk controller card (for disk version only) (not included)
1 32K memory expansion card (for cassette version with disk system attached and turned on; also for disk version) (not included)
1 peripheral expansion box with peripheral expansion card (for disk version only) (not included)
1 cassette recorder with interface cables (cassette version only) (not included)

Note: For disk version only, separate units can be used in lieu of the peripheral expansion box and all cards listed above.

Preparation:

Connect all equipment (not included with this package) as shown in each equipment's respective instruction manuals and insure that the equipment is working properly. Read and study this entire instruction booklet carefully before proceeding to play the game.

Game Description:

Each player, in turn, forms interlocking words, as in a crossword puzzle, on the playing grid using letters with various point values and placing them on the grid in locations that take best advantage of letter values and bonus squares to achieve the highest score by the end of the game.

Play:

The first player combines two or more of the letters in his holder to form a word and places them on the grid to read either across or down with one letter on the starting square. Diagonal words are not allowed.

A player's turn is completed after the computer tallies his score for any word(s) formed and adds that score to his score total. The computer will also automatically replace the letters used from the player's holder with letters selected at random from the letter pool.

The second player, and then each in turn, adds one or more letters to those already played so as to form new words. All letters played in any one turn must be placed in one straight line across or down the grid. They must form one complete word and if, at the same time, they touch letters in adjacent rows or columns, they must also form complete words, crossword fashion, with all such letters. The player gets full credit for all words formed or modified by his play.

New words may be formed by:

1. Adding one or more letters to a word or letters already on the grid.

2. Placing a word at right angles to a word already on the grid. The new word must use one of the letters of the word already on the grid or must add a letter to it.

3. Placing a complete word parallel to a word already played so that adjoining letters also form complete words.

Any words found in a standard dictionary are permitted except those capitalized, those designated as foreign words, abbreviations and words requiring apostrophes or hyphens. Consult a dictionary only when a word is challenged to check for spelling or usage.

Play continues until all letters in the letter pool have been drawn and one of the players has used all of the letters in his holder or until all letters in the letter pool have been drawn and each of every player, in succession, uses the pass option.

Examples of Play (two—player game):
1st player's letter holder: T A W R O N

Letters used in turn: W O R N	WORN

Points scored for forming: WORN

2nd player's letter holder: L L N A B E	B
	A
Letters used in turn: B A N	WORN
	N

Points scored for forming: BARN

1st player's letter holder: T A M E S T	B
	A
Letters used in turn: T A S T E	WORN
	N
Points scored for forming: TASTE, BARNS	TASTE

2nd player's letter holder: L L O R Z E	B
	A
Letters used in turn: O R	WORN
	NOR
Points scored for forming: NOR, NOT, RE	TASTE

1st player's letter holder: D E M T I B	B
	A
Letters used in turn: E D I T	WORN
	NOR
Points scored for forming: EDIT, NOTE, RED	TASTE
	EDIT

2nd player's letter holder: L L S K Z E	B
	A
Letters used in turn: S	SWORN
	NOR
Points scored for forming: SWORN	TASTE
	EDIT

Sequence of Statements

HOW MANY PLAYERS (1-4)?

Decide the amount of players (from one to four may compete), the order in which each player will play, and enter the total number.

PLAYER _: P=PASS; ENTER=PLAY

```
10 "Y'********************
20 '*** ***
30 KEM*** KINGDOM ***
40 REM*** ***
50 REM********************
60 GOSUB 110
70 GOSUB 310
80 GOSUB 520
90 GOSUB 1820
100 END
110 REM
120 REM***INSTRUCTIONS***
130 REM
140 GOSUB 270
150 PRINT "THIS IS A SIMULAT
ION OF THE COUNTRY OF SUMERI
A. YOU ARETHE SOVEREIGN RUL
ER, AND"
160 PRINT "YOUR REIGN WILL L
AST FOR TENYEARS."
170 PRINT
180 PRINT "THE DECISIONS YOU
 MAKE WILL AFFECT HUNDREDS O
F PEOPLE. YOUR DICTATORIAL
SKILLS WILL"
190 PRINT "BE RATED AT THE E
ND OF YOUR RULE."
200 PRINT
210 PRINT "YOU WILL BE ASKED
 TO MAKE   SEVERAL KEY DECIS
IONS EACH  YEAR."
220 FOR I=1 TO 5
230 PRINT
240 NEXT I
250 INPUT "PRESS ENTER WHEN
READY TO   CONTINUE: ":ANS$
260 RETURN
270 CALL CLEAR
280 PRINT TAB(8);"*** KINGDO
M ***"
290 PRINT
300 RETURN
310 REM
320 REM***SETUP***
330 REM
340 CALL CLEAR
350 RANDOMIZE
360 DIM NU$(11)
370 P=95
380 S=2800
390 H=3000
400 E=H-S
410 Y=3
420 A=H/Y
430 I=5
440 Q=1
450 D=0
460 Z=0
470 FOR X=1 TO 11
480 READ NU$(X)
490 NEXT X
500 DATA FIRST,SECOND,THIRD,
FOURTH,FIFTH,SIXTH,SEVENTH,E
IGHTH,NINTH,TENTH,ELEVENTH
510 RETURN
520 REM
530 REM***PLAY***
540 REM
550 GOSUB 270
560 FOR X=1 TO 3
570 PRINT
580 NEXT X
590 Z=Z+1
600 PRINT "HAMURABI, I BEG T
O REPORT TOYOU:"
610 PRINT
620 PRINT "IN THE ";NU$(Z);"
 YEAR,";D;"PEOPLE";
630 PRINT "STARVED;";I;"CAME
 TO THE ";"CITY."
640 P=P+I
650 IF Q THEN 670
660 GOSUB 1380

670 PRINT
680 PRINT "THE POPULATION IS
";P;"AND"
690 PRINT "THE CITY OWNS";A;
"ACRES."
700 PRINT "YOU HARVESTED";Y;
"BUSHELS PER"
710 PRINT "ACRE. RATS ATE";E
;"BUSHELS."
720 PRINT "YOU HAVE";S;
730 PRINT "BUSHELS IN ";"RES
ERVE."
740 PRINT
750 PRINT
760 IF Z=11 THEN 1810
770 Y=INT(RND*10)+17
780 PRINT "LAND IS TRADING A
T";Y
790 PRINT "BUSHELS PER ACRE.
 HOW MANY ACRES DO YOU WISH
TO BUY?"
800 INPUT Q
810 PRINT
820 IF Q<0 THEN 1460
830 IF Y*Q>S THEN 1490
840 IF Q>0 THEN 1520
850 INPUT "HOW MANY ACRES DO
 YOU WISH  TO SELL? ":Q
860 PRINT
870 IF Q<0 THEN 1560
880 IF Q=A THEN 1590
890 IF Q>A THEN 1610
900 A=A-Q
910 S=S+Y*Q
920 C=0
930 PRINT
940 PRINT "OF THE";S;"BUSHEL
S THAT"
950 INPUT "ARE LEFT, HOW MAN
Y DO YOU    WISH  TO FEED TO
YOUR          PEOPLE? ":Q
960 PRINT
970 IF Q<1 THEN 1650
980 IF Q=S THEN 1670
990 IF Q>S THEN 1690
1000 S=S-Q
1010 PRINT
1020 PRINT "OF THE";A;"ACRES
 YOU NOW"
1030 INPUT "OWN, HOW MANY DO
 YOU WISH TOPLANT WITH SEED?
":D
1040 PRINT
1050 IF D<1 THEN 1720
1060 IF D>A THEN 1740
1070 IF D/2>S THEN 1760
1080 IF D>10*P THEN 1780
1090 S=S-INT(D/2)
1100 Y=INT(RND*5)+1
1110 H=D*Y
1120 E=0
1130 IF INT(Y/2)*2=Y THEN 13
60
1140 S=S-E+H
1150 I=INT(Y*(20*A+S)/P/100+
1)
1160 C=INT(Q/20)
1170 Q=INT(10*(2*RND-.3))
1180 IF P<C THEN 1250
1190 D=P-C
1200 IF D>.5*P THEN 1270
1210 P1=((Z-1)*P1+D*100/P)/Z
1220 P=C
1230 D1=D1+D
1240 GOTO 550
1250 D=0
1260 GOTO 550
1270 GOSUB 270
1280 PRINT "YOU STARVED";D;"
PEOPLE IN"
1290 PRINT "ONE YEAR. YOU H
AVE DONE   SUCH A MISERABLE
JOB THAT"

1300 PRINT "YOU HAVE BEEN OV
ERTHROWN ANDEXILED TO A DESE
RTED ISLAND."
1310 FOR I=1 TO 13
1320 PRINT
1330 NEXT I
1340 WL=5
1350 GOTO 1810
1360 E=INT(S/Y)
1370 GOTO 1140
1380 CALL CLEAR
1390 PRINT " A HORRIBLE PLA
GUE STRUCK!!"
1400 FOR I=1 TO 5
1410 PRINT
1420 NEXT I
1430 PRINT "HALF OF YOUR PEO
PLE DIED...."
1440 P=INT(P/2)
1450 RETURN
1460 PRINT "HAMURABI, YOU CA
N'T DO THAT,TO SELL LAND, FI
RST BUY ZEROACRES."
1470 PRINT
1480 GOTO 780
1490 PRINT "HAMURABI, THINK
AGAIN!"
1500 PRINT
1510 GOTO 780
1520 A=A+Q
1530 S=S-Q*Y
1540 C=0
1550 GOTO 930
1560 PRINT "HAMURABI, YOU CA
N'T DO THAT.IF YOU DO NOT WI
SH TO SELL, THEN SELL ZERO A
CRES."
1570 PRINT
1580 GOTO 850
1590 PRINT "HAMURABI, YOU MU
ST KEEP AT  LEAST ONE ACRE O
F LAND!"
1600 GOTO 850
1610 PRINT "HAMURABI, YOU ON
LY OWN";A
1620 PRINT "ACRES."
1630 PRINT
1640 GOTO 850
1650 PRINT "HAMURABI, THE PE
OPLE WILL   STARVE! YOU MUST
 FEED THEM   SOMETHING."
1660 GOTO 930
1670 PRINT "HAMURABI, YOU MU
ST KEEP AT  LEAST ONE BUSHEL
 TO PLANT."
1680 GOTO 930
1690 PRINT "HAMURABI, YOU ON
LY OWN";S
1700 PRINT "BUSHELS."
1710 GOTO 930
1720 PRINT "HAMURABI, YOU MU
ST PLANT    SOMETHING SO THE
RE WILL BE  FOOD FOR NEXT YE
AR."
1730 GOTO 1010
1740 PRINT "YOU ONLY HAVE";A
;"ACRES!"
1750 GOTO 1010
1760 PRINT "HAMURABI, THAT I
S TOO MUCH  TO PLANT."
1770 GOTO 1010
1780 PRINT "YOU CAN ONLY FOR
CE ONE MAN  TO WORK 10 ACRES
 OF LAND.   YOUR POPULATION
OF";P
1790 PRINT "JUST ISN'T BIG E
NOUGH."
1800 GOTO 1010
1810 RETURN
1820 REM
1830 REM***END***
1840 REM
1850 IF WL=5 THEN 2090
```

```
1860 INPUT "PRESS ENTER TO S
EE YOUR       RATING.":A$
1870 GOSUB 270
1880 PRINT "IN YOUR 10 YEARS
OF RULE, ";P1;"PERCENT OF
THE"
1890 PRINT "POPULATION STARV
ED PER YEAR (ON THE AVERAGE)
; A TOTAL OF";D1;"PEOPLE DIE
D."
1900 L=A/P
1910 PRINT
1920 PRINT "YOU STARTED WITH
10 ACRES    PER PERSON AND Y
OU ENDED    WITH";L;"ACRES P
ER PERSON."
1930 IF P1>33 THEN 2010
1940 IF L<7 THEN 2010
1950 IF P1>10 THEN 2030
1960 IF L<9 THEN 2030
1970 IF P1>3 THEN 2050
1980 IF L<11 THEN 2050
1990 PRINT "A TRULY INSPIRED
JOB.  THE  PEOPLE LOVE AND
ADMIRE YOU."
2000 GOTO 2060
2010 PRINT "YOU ARE A DISGRA
CE!!!  THE  PEOPLE CHEERED W
HEN YOU LEFTOFFICE."
2020 GOTO 2060
2030 PRINT "YOU RULE LIKE AT
ILLA THE    HUN!!!  MOST OF
YOUR        SUBJECTS WOULD D
ANCE AT YOUR FUNERAL."
2040 GOTO 2060
2050 PRINT "YOU COULD HAVE D
ONE BETTER. FEW PEOPLE CARE
TO SEE YOU  RULE AGAIN."
2060 FOR I=1 TO 5
2070 PRINT
2080 NEXT I
2090 RETURN              o
```

```
100 CALL CLEAR
110 REM  MUSIC TUNER
120 REM  BY V. MAKER
130 PRINT "GUITAR STRINGS"
140 FOR T=0 TO 3
150 PRINT
160 NEXT T
170 PRINT "LOW  E:   165"
180 PRINT
190 PRINT "     A:   220"
200 PRINT
210 PRINT "     D:   294"
220 PRINT
230 PRINT "     G:   392"
240 PRINT
250 PRINT "     B:   494"
260 PRINT
270 PRINT "     E:   659"
280 PRINT
290 PRINT
300 PRINT
310  INPUT  "ENTER  REQUIRED
FREQUENCY:":FREQ
320 CALL SOUND(500,FREQ,0)
330 PRINT
340 PRINT "PRESS 'A' FOR  THE
SAME SOUND AGAIN.   PRESS 'B'
TO END.  PRESS ANY OTHER  KEY
TO       SELECT OTHER"
350 PRINT "NOTES."
360 CALL KEY(0,K,L)
370 IF L=0 THEN 360
380 IF K=65 THEN 320
390 IF K=66 THEN 410
400 GOTO 100
410 END                 o
```

```
10 REM********************
20 REM***            ***
30 REM***=ANIMAL LEARNER***
40 REM***            ***
50 REM********************
60 GOSUB 100
70 GOSUB 300
80 GOSUB 420
90 END
100 REM
110 REM***INSTRUCTIONS***
120 REM
130 GOSUB 260
140 PRINT "THIS IS A GAME TH
AT HAS THE ABILITY TO LEARN.
IT WILL   ATTEMPT TO GUESS
THE NAME"
150 PRINT "OF AN ANIMAL THAT
YOU PICK   AT RANDOM."
160 PRINT
170 PRINT "WHENEVER YOU STUM
P THE         COMPUTER, YOU ARE
ASKED       ABOUT THE ANIMAL
YOU CHOSE."
180 PRINT "BY COMPILING THIS
DATA, THE COMPUTER 'LEARNS'
."
190 PRINT
200 PRINT "ENTER 'STOP' WHEN
YOU ARE   DONE."
210 FOR I=1 TO 6
220 PRINT
230 NEXT I
240 INPUT "PRESS ENTER WHEN
READY TO   CONTINUE: ":ANS$
250 RETURN
260 CALL CLEAR
270 PRINT TAB(4);"*** ANIMAL
LEARNER ***"
280 PRINT
290 RETURN
300 REM
310 REM***SETUP***
320 REM
330 CALL CLEAR
340 DIM QU$(50),RI(50),WR(50
),RA$(50),WA$(50)
350 QU$(1)="DOES IT ROAR"
360 RI(1)=0
370 WR(1)=0
380 RA$(1)="LION"
390 WA$(1)="GRIZZLY BEAR"
400 FR=2
410 RETURN
420 REM
430 REM***PLAY***
440 REM
450 LI=1
460 GOSUB 260
470 FOR I=1 TO 10
480 PRINT
490 NEXT I
500 PRINT
510 PRINT "I KNOW";FR;"ANIMA
LS...."
520 PRINT
530 PRINT QU$(LI);
540 INPUT "?":ANS$
550 ANS$=SEG$(ANS$,1,1)
560 IF ANS$="Y" THEN 690
570 IF ANS$="N" THEN 740
580 IF ANS$="S" THEN 1340
590 A$="PLEASE ENTER 'YES' O
R 'NO'."
600 FOR I=1 TO LEN(A$)
610 CALL HCHAR(12,I+2,ASC(SE
G$(A$,I,1)))
620 NEXT I
630 FOR I=1 TO 300
640 NEXT I
650 FOR I=1 TO LEN(A$)
660 CALL HCHAR(16,I+2,32)
670 NEXT I
```

```
680 GOTO 540
690 IF RI(LI)<>0 THEN 720
700 GU$=RA$(LI)
710 GOTO 790
720 LI=RI(LI)
730 GOTO 520
740 IF WR(LI)<>0 THEN 770
750 GU$=WA$(LI)
760 GOTO 790
770 LI=WR(LI)
780 GOTO 520
790 PRINT
800 PRINT "IS IT A ";GU$;
810 INPUT "?":TA$
820 TA$=SEG$(TA$,1,1)
830 IF TA$="Y" THEN 1040
840 IF TA$="S" THEN 1340
850 PRINT
860 INPUT "WHAT WAS THE ANIM
AL? ":NA$
870 IF FR=51 THEN 1140
880 PRINT
890 PRINT "WHAT IS A QUESTIO
N I COULD  ASK TO TELL THE D
IFFERENCE  BETWEEN A ";GU$
900 PRINT "AND A ";NA$;
910 INPUT "? ":QU1$
920 PRINT
930 PRINT "FOR ";NA$;", THE"
940 PRINT "ANSWER IS WHAT";
950 INPUT "? ":YN$
960 YN$=SEG$(YN$,1,1)
970 IF (YN$<>"Y")*(YN$<>"N")
THEN 920
980 IF ANS$="Y" THEN 1230 EL
SE 1270
990 QU$(LI)=QU1$
1000 IF YN$="Y" THEN 1310
1010 RA$(LI)=GU$
1020 WA$(LI)=NA$
1030 GOTO 450
1040 FOR I=1 TO 200
1050 NEXT I
1060 CALL CLEAR
1070 A$="WOW!  I GOT IT!"
1080 FOR I=1 TO LEN(A$)
1090 CALL HCHAR(12,I+8,ASC(S
EG$(A$,I,1)))
1100 NEXT I
1110 FOR I=1 TO 300
1120 NEXT I
1130 GOTO 450
1140 CALL CLEAR
1150 PRINT "I CAN'T REMEMBER
THAT ONE,"
1160 PRINT "MY MEMORY IS FUL
L."
1170 FOR I=1 TO 12
1180 PRINT
1190 NEXT I
1200 FOR I=1 TO 400
1210 NEXT I
1220 GOTO 450
1230 RI(LI)=FR
1240 LI=FR
1250 FR=FR+1
1260 GOTO 990
1270 WR(LI)=FR
1280 LI=FR
1290 FR=FR+1
1300 GOTO 990
1310 RA$(LI)=NA$
1320 WA$(LI)=GU$
1330 GOTO 450
1340 RETURN            o
```

From the moment that the above statement appears at the bottom of the screen, the player has approximately 3 1/2 minutes to decide what letters from his holder he will use to form a new word or words and where the letters will be placed on the grid. He then presses the enter button when ready.

If the player cannot make a word, or does not wish to play any of the letters he has in his holder and desires a new set of letters he must then press the "P" key to use the pass option before time runs out.

This pass option will automatically place all of the letters in the player's holder back into the letter pool and then replace the empty holder with another set of randomly selected letters from the pool.

Using the pass option automatically forfeits the player's right to place any letters on the grid; thus the player, in effect, loses his turn. Play immediately continues on to the next player.

TOO MUCH TIME!!!

If, after 3 1/2 minutes, a player had not pressed either the enter button or the "P" key, the above statement will appear. The player retains all of the letters in his holder, he forfeits his chance to pass or play, and his turn is completed.

YOU LOSE YOUR TURN!!!

· This statement appears if:

1. Player runs out of time before deciding whether to pass or play.

2. A word being played is challenged and is found to be unacceptable.

3. A player tries to make an illegal word or attempts to place a word or illegal word in the incorrect row and/or column of the playing grid.

In all of the above cases, the player loses his chance to play any letters for this turn and play immediately moves to the next player.

SPELL WORD THEN PRESS ENTER.

After electing to play, by pressing the enter button, the above statement will appear briefly and then the line will go blank with only the flashing cursor showing at the bottom.

The player must now type in the ENTIRE word being formed, which includes any letters from the player's holder that is to be placed on the grid plus any letters already on the grid which, when connected to the newly-placed letters, form one complete word in a single direction.

```
EXAMPLE: before      B        after        B
         spelling    A        pressing     A
         word        WORN     enter        WORN
                     N                     NOR
                     TASTE                 TASTE
```

Letter holder contains: L L O R Z E

Player wishes to use the letters "O" and "R", from his letter holder, to place next to the "N" in the word "BARN", forming the principal word "NOR", along with two other words "NOT" and "RE".

When the player is asked to spell the word and press enter, he should type in the entire principal word "NOR" then press enter. The computer will automatically give him credit for the other two words.

ROW OF 1ST LETTER IN WORD:

The row numbers appear on the left side of the playing grid. After spelling out the principal new word being formed, enter the row number of the position where the first letter of that entire principal word will be placed on the grid.

COLUMN OF 1ST LETTER:

The column numbers appear at the top of the playing grid. Enter the column number of the position where the first letter of the entire principal word will be placed on the grid.

Both row number and column number, together, will determine the exact location on the playing grid where the first letter will go.

DIRECTION (A=ACROSS;D=DOWN)

Beginning at the position where the row and column numbers intersect on the playing grid, determine whether the entire word will be placed horizontally (across) or vertically (down) by entering an "A" or a "D".

ANY CHALLENGES? (Y/N) N

When the above statement appears, any of the other players, at this time, may elect to challenge the legality of a newly formed word, before it is placed on the grid, due to a suspected improper placement or spelling, by saying "YES". If no challenge is made, press enter and the new word or words will be placed on the grid. If there is a challenge, enter "Y".

WHO WON THE BATTLE ?

(P=PLAYER; C=CHALLENGER): P

After a word has been challenged, the first statement, above, will appear briefly; then the next statement, above, will appear. All players must now decide amongst themselves, consulting a dictionary, if necessary, if placement and spelling of the new word or words is proper. If placement and spelling is correct, press enter. If not, enter "C" and the current player will retain his letters in his holder and lose his turn.

Scoring

The computer automatically keeps a tally of each player's score and enters it in the appropriate scoring area.

The point value of each letter is located on the letter values chart to the right of the playing grid.

The score for each turn is the sum of the point values of all the letters in each word formed or modified in the play plus the bonus values resulting from placing letters on bonus squares.

Squares on the grid have different colours. The one square that is unique from all the others is the starting square; a letter placed on this square scores its normal letter value. All grey squares score the normal letter value. Bonus squares are either yellow, green, red or blue.

A letter placed on a yellow square scores two (2) times its normal value.

A letter placed on a green square scores three (3) times its normal value.

A letter placed on a red square scores four (4) times its normal value.

A letter placed on a blue square scores five (5) times its normal value.

Letter bonus values apply only in the turn in which they are first played. In subsequent turns, letters count at normal chart value.

When two or more words are formed in the same play, each is scored. Any letters which are common to both or all newly-formed words are counted (with full bonus value, if any) in the score for each word.

Any player who plays all six of his letters in his holder in a single turn scores a bonus of fifty (50) points in addition to his regular score for the play.

If the letter pool is depleted, and a player has just played the last letter in his holder, the game ends and gives an additional fifty (50) points to that player.

TIger's Tale

by Harry Brashear, Asgard News

"The user group is the life blood of the TI99/4A community." (period, end of sentence, absolute statement!) That emphatic judgement made by many, may not be sure anymore. In fact, in many cases, I could call some of the user groups today, the bleeding arteries of the TI99/4A community.

At some point, back a few years ago, when the TI99/4A was young, orphaned, and people had no place to turn, there was no question of the groups' value. Today because of bad management, bad judgement, and burned out leadership, there are many groups that are falling apart. When the group begins to fall, the remaining membership is in worse shape than ever.

I would like to share some thoughts with you based on my own experience and some stories I have been told.

First of all, and most importantly, not all groups are falling apart. Take the Rochester users group for instance. Two years ago, they were meeting in the president's basement because the entire group consisted of perhaps ten people. Today, they have thirty some members and they need to meet in a school.

How did this happen, you ask? Well, one of the primary ways they have found to get membership is to call up every newspaper advertisement they find for used TI99/4A equipment. This accomplishes two things: 1> they find lots more spare equipment; and, 2> they always tell the seller about the user group. Many of these people are surprised to find out that the TI99/4A is not dead and come to a meeting.

Another good example is West Penn 99ers. Not to long ago this group was on its way down hill but, the group leaders got involved in hardware projects. With the help of people like John Wilforth, getting more out of the existing systems became a major group project. The result was a terrific boost in the membership based on this single premise. Much of the TI99/4A community now looks to this gang for many of the hardware projects that update our computers.

They are only two examples of continuing success. They are successful because the leadership of the group cares and enjoys working with the TI99/4A, but when the caring fails, so does the group. A number of things can happen to begin a general erosion of TI morale. A BBS converts from TIBBS to FIDONET. A good programmer converts to IBM and starts talking about all the $$$ he is making. A little pack of converts starts meeting in the rear and talking IBM. (That I have seeming at some TIsHUG and Regional meeting lately.) A newsletter editor gets lazy and starts using Ventura at work. All of these things are bad news to the TI99/4A user group because it is generally made up of followers.

The worst I have heard about, is a group with a huge treasury and a few remaining members. They are hanging in so that they can split the money at the official fold-up. The last thing they want is new members. (Let this be a lesson. If you have not done so, commit your treasury to the local zoo or something, should a breakup occur.)

There seems to be this "thing" that says, "if I go out and buy a new computer, I have to make everybody else do the same, and since they are all my friends, so this makes it right". Wrong! If you are a leader, then act like a responsible leader. Stand down and move on to the IBM group. Why hang around and wreck everyone else's fun. It takes TI99/4A'ers to run a TI99/4A group, people that are looking forward to the next big breakthrough for our computer.

I have screamed time and time again, publicity, about multi-user groups. Computers of various kinds cannot coexist together any more than one computer's BASIC can be run into another. I have seen the results of letting the clones through the door and the guy with the little' computer gets trampled into the dust every time. (The above happened at the meeting at Han's place and to a degree at Cyril's place the last two meetings. It will not happen again if I can help it!!!)

We all take some interest in other computers, we have to, because that is often where some ideas come from. If it were not for Procomm for the IBM, we probably would not have TELCO. If it were not for WordPerfect, we would not have anything to model Press after. Of course, TI-Base is D-Base II through and through, so, thank God for other computers. But if the interest gets out of hand and a leader wants everyone to join the rest of the flag waving techies, look out! It takes a lot of guts to stand up to these individuals, but you have to, to preserve your sanity and more importantly, your group.

Another thing that will bring down a group is a lack of communication, both within the group and with the outside community. Without communications there is no excitement about what is to come. If I did not think there was anything new coming, would I stay with the TI99/4A? No! But I know there are great things coming because I read and write and I call people to find out what is new. People tell me what is new because they know that I generate some of the excitement that drives the TI99/4A engine. I write to people all over the world.. Australia, Germany, Italy, and all over Canada and the US. Sure, it costs me a few bucks to do this, but it is worth it. The money that you spend on movies, dinners out, Saturday afternoon at the pub, and petrol to do these things, I spend on long distance calls.

If your group is not supplying you with the information you need, get it yourself and pass it back. Do you take the time to read all those newsletters that your group trades back and forth, or do you ever see them at all? If you send me a big self addressed stamped envelope with a few comments of your own I will be happy to send you a list of groups, and mark off a few of what I feel have the best newsletters. Join these groups, buy MICROpendium, Asgard News, and any other periodical that might come along. If you are new to the community and feel a little left out, ask questions, or better still, write your questions down and send them to us.

Next time: The Computer versus The Family Unit. o

"*...Shit...*"

156 Collingwood Ave.
Columbus OH 43213

I am still offering over 120 original and unique entertainment, educational and utility programs at just $1 each, or on collection disks at $5 per disk.

The contents of the first 52 issues of this newsletter are available as ready to run programs on 5 Tips Disks at $10 each.

And my three Nuts & Bolts disks, $15 each, each contain over 100 subprograms for you to merge into your own programs to do all kinds of wonderful things.

My catalog is available for $1, deductable from your first order (specify Tigercub catalog).

****
TI-PD Library

I have selected public domain programs, by category, to fill over 230 disks, as full as possible if I had enough programs of the category, with all the Basic-only programs converted to Extended BASIC , with an E/A loader provided for assembly programs if possible, instructions added and any obvious bugs corrected, and with an autoloader by full program name on each disk. These are available as a copying service for · just $1.50 postpaid in U.S. and Canada. No fairware will be offered without the author's permission. Send a stamped self addressed envelo for list or $1, refundable for 9-page catalog listing all titles and authors. Be sure to specify TI-PD catalog.

****

In Tips #55 I published a CHARSUB routine to convert character patterns into assembly source code, and in Tips #55 and #56 I published several routines to manipulate hex codes into new character sets. Those patterns looked fine on my old TV, but when I demonstrated them on a high-resolution monitor I could see too many missing pixels.

So I wrote this CHARFIX program which, when MERGEd into a program and CALLed after any character redefinition is completed, will permit any normal or reidentified character to be viewed on screen and edited and will then write the hex codes of any range of printable characters into an assembly source file which can be assembled, loaded and linked to instantly change character sets.

This routine also reidentifies the common punctuation into the same character sets as the letters, as described in Tips #55. If you do not want this feature, delete lines 29001-29003.

```
29000 SUB CHARFIX
29001 DATA 32,33,34,44,46
29002 RESTORE 29001 :: FOR J=1 TO 5 :: READ CH :: CALL
     CHARPAT(CH,CH$):: CALL CHAR(J+90,CH$):: CALL
     CHAR(J+122,CH$):: NEXT J
29003 CALL CHARPAT(63,CH$):: CALL CHAR(64,CH$):: CALL
     CHAR(96,CH$)
29004 DISPLAY AT(1,1)ERASE ALL:"1 2 3 4 5 6 7 8 9 0 :
     ;":" ":"@ A B C D E F G H I J K L M":" ":"N O P Q R
     S T U V W X Y Z [":" ":"\ ] ^ _ a b c d e f g h i j"
29005 DISPLAY AT(9,1):"k l m n o p q r s t u v w x":"
     "
29006 CALL CHAR(128,"FF"&RPT$("81",6)&RPT$("FF",9)&
     "FFFF"&RPT$("C3",4)&"FFFF"):: CALL COLOR(13,2,16)
29007 CALL CHARVIEW
29008 SUBEND
29009 SUB CHARVIEW
29010 DISPLAY AT(13,14):"CTRL V TO VIEW" :: DISPLAY
     AT(14,14):" " :: DISPLAY AT(15,14):"CTRL E TO EDIT"
     :: DISPLAY AT(17,14):"CTRL S TO SAVE"
29011 DISPLAY AT(19,14):" " :: DISPLAY AT(20,14):" "
29012 CALL KEY(0,@,S):: IF S=0 THEN 29012 ELSE IF @=150
     THEN 29015 ELSE IF @=133 THEN 29014 ELSE IF @=147
     THEN 29013 ELSE 29012
29013 CALL DELSPRITE(#1):: CALL CHARSUB(HX$()):: DISPLAY
     BEEP :: STOP
29014 CALL EDIT(K):: GOTO 29010
29015 DISPLAY AT(24,1)BEEP:""
29016 DISPLAY AT(24,1):"PRESS A KEY" :: CALL
     KEY(0,K,S):: IF S<1 OR K<32 OR K>143 THEN 29016
29017 DISPLAY AT(24,1):"" :: CALL CHARPAT(K,CH$)
29018 R=13 :: FOR J=1 TO 15 STEP 2
29019 H$=SEG$(CH$,J,1):: CALL HEX_BIN(H$,B$)
29020 H$=SEG$(CH$,J+1,1):: CALL HEX_BIN(H$,BB$):: FOR
     L=1 TO 8 :: C$=C$&CHR$(ASC(SEG$(B$&BB$,L,1))+80)::
     NEXT L
29021 DISPLAY AT(R,1):C$;::: DISPLAY
     AT(R,10):SEG$(CH$,J,2);::: R=R+1 :: C$="" :: NEXT J
     :: DISPLAY AT(22,1):CH$;::: GOTO 29012
29022 SUBEND
29023 SUB HEX_BIN(H$,B$):: HX$="0123456789ABCDEF" ::
     BN$="0000X0001X0010X0011X0100X0101X0110X011
     1X1000X1001X1010X1011X1100X1101X1110X1111"
29024 FOR J=LEN(H$)TO 1 STEP -1 :: X$=SEG$(H$,J,1)
29025 X=POS(HX$,X$,1)-1 :: T$=SEG$(BN$,X*5+1,4)&T$ ::
     NEXT J :: B$=T$ :: T$="" :: SUBEND
29026 SUB CHARSUB(HX$())
29027 DISPLAY AT(12,1)ERASE ALL:"Source code
     filename?":"DSK" :: ACCEPT AT(13,4)SIZE(12)BEEP:F$
     :: OPEN #1:"DSK"&F$,OUTPUT
29028 DISPLAY AT(15,1):"LINKABLE program name?" ::
     ACCEPT AT(16,1)SIZE(6):P$
29029 DISPLAY AT(18,1):"Redefine characters from
     ASCII    to ASCII"
29030 ACCEPT AT(19,7)VALIDATE(DIGIT)SIZE(3):F
29031 ACCEPT AT(19,21)VALIDATE(DIGIT)SIZE(3):T
29032 PRINT #1:TAB(8);"DEF";TAB(13);P$ :: PRINT #1:"VMBW
     EQU >2024" :: PRINT #1:"STATUS EQU >837C"
29033 NB=(T-F)*8 :: CALL DEC_HEX(NB,H$):: A=768+F*8 ::
     CALL DEC_HEX(A,A$)
29034 FOR CH=F TO T :: IF CH<144 THEN CALL
     CHARPAT(CH,CH$)ELSE CH$=HX$(CH)
29035 IF FLAG=0 THEN PRINT #1:"FONT";::: FLAG=1
29036 FOR J=1 TO 13 STEP 4 :: M$=M$&">"&SEG$(CH$,J,4)&",
     " :: NEXT J :: M$=SEG$(M$,1,23)&" *"&CHR$(CH)
29037 PRINT #1:TAB(8);"DATA "&M$ :: M$="" :: NEXT CH
29038 PRINT #1:P$;TAB(8);"LI   R1,FONT" :: PRINT
     #1:TAB(8);"LI   R0,>"&A$ :: PRINT #1:TAB(8);"LI
     R2,>"&H$
29039 PRINT #1:TAB(8);"BLWP @VMBW":TAB(8);"CLR
     @STATUS":TAB(8);"RT":TAB(8);"END" :: CLOSE #1
29040 SUBEND
29041 SUB DEC_HEX(D,H$)
29042 X$="0123456789ABCDEF" :: A=D+65536*(D>32767)
29043 H$=SEG$(X$,(INT(A/4096)AND
     15)+1,1)&SEG$(X$,(INT(A/256)AND
     15)+1,1)&SEG$(X$,(INT(A/16)AND 15)+1,1)&SEG$(X$,(A
     AND 15)+1,1):: SUBEND
29044 SUB EDIT(CH)
29045 DISPLAY AT(13,14):"1 TO TOGGLE" :: DISPLAY
     AT(14,15):"CURSOR" :: DISPLAY AT(15,14):"E S D X TO
     MOVE" :: DISPLAY AT(17,14):"CTRL A TO ABORT"
29046 DISPLAY AT(19,14):"CTRL R TO" :: DISPLAY
     AT(20,15):"REIDENTIFY"
29047 R=13 :: C=3 :: X=128 :: CALL
     SPRITE(#1,130,11,R*8-7,C*8-7)::
     X$=CHR$(129)&CHR$(146)
29048 CALL KEY(0,K,S):: IF S<1 THEN 29048 ELSE ON
     POS("1EeSsDdXx"&X$,CHR$(K),1)+1 GOTO 29048,29049,
     29050,29050,29051,29051,29052,29052,29053,29053,
     29055,29056
29049 X=X+1+(X=129)*2 :: GOTO 29054
29050 R=R-1-(R=13):: GOTO 29054
29051 C=C-1-(C=3):: GOTO 29054
29052 C=C+1+(C=10):: GOTO 29054
29053 R=R+1+(R=20)
29054 CALL LOCATE(#1,R*8-7,C*8-7):: CALL HCHAR(R,C,X)::
     GOTO 29048
29055 CALL DELSPRITE(#1):: SUBEXIT
29056 FOR R=13 TO 20 :: FOR C=3 TO 10 :: CALL
     GCHAR(R,C,GH):: CALL LOCATE(#1,R*8-7,C*8-7)::
     B$=B$&CHR$(GH-80):: NEXT C
29057 CALL BIN_HEX(B$,H$):: DISPLAY AT(R,10):H$;::: B$=""
     :: HEX$=HEX$&H$ :: NEXT R :: DISPLAY
     AT(22,1):HEX$;::: CALL CHAR(CH,HEX$):: HEX$=""
29058 CALL DELSPRITE(#1):: FOR R=13 TO 20 :: DISPLAY
     AT(R,14):"" :: NEXT R :: SUBEND
29059 SUB BIN_HEX(B$,H$):: HX$="0123456789ABCDEF" ::
     BN$="0000X0001X0010X0011X0100X0101X0110X011
     1X1000X1001X1010X1011X1100X1101X1110X1111"
```

```
29060 L=LEN(B$):: IF L/4<>INT(L/4)THEN B$="0"&B$ :: GOTO
     29060
29061 FOR J=L-3 TO 1 STEP -4 :: X$=SEG$(B$,J,4)
29062 X=(POS(BN$,X$,1)-1)/5 :: T$=SEG$(HX$,X+1,1)&T$ ::
     NEXT J :: H$=T$ :: T$="" :: SUBEND
```

I think that programs, at least non-commercial ones, should be open for anyone to modify for their own use. For that reason, I would not normally publish the following routine. However, I recently received a large number of programs, originally in the IUG library, and found that the author's name had been erased from the title screen or REM of every one of them. I know, because I already had many of the original versions, including some that I wrote myself.

Now, that is inexcusable. If a programmer is willing to share his work, he does deserve credit for it. And if people are going to play that dirty, maybe there is good reason for protecting programs.

So here is how to do it. Ken Woodcock wrote this ingenious routine and published it in the Tidewater newsletter. I have modified it so that it can be deleted after it has done its work. It is to be MERGEd into any Extended BASIC program (32k required) and RUN, and will change the line length byte of each line to zero, so that the program cannot be LISTed, although it can be loaded and run.

```
1 CALL INIT :: CALL PEEK(-31952,A,B,C,D)::
     SL=C*256+D-65539 :: EL=A*256+B-65536 :: FOR X=SL TO
     EL STEP -4
2 CALL PEEK(X,E,F,G,H):: ADD=G*256+H-65536 :: J=J+1 ::
     IF J<4 THEN 3 :: CALL LOAD(ADD-1,0)
3 NEXT X :: STOP :: !@P-
```

Save that as FIX in MERGE format. Merge it into any program (RESequence first if it has line numbers less than 4) and RUN. Then type 1, FCTN X and FCTN 3 to delete line 1. Delete lines 2 and 3 in the same way. Then SAVE. Now try LISTing it and watch the fireworks.

Ken wrote an even more ingenious UNFIX routine to unprotect the program, but I am not passing that on!

Now, suppose you have a party game program that you do not want the kids playing with. So, RESequence it to some odd number, such as RES 797. Put in a line just before that 796 STOP. Then merge in FIX, run it, and delete those first 3 lines.

I hope you remember what line number you resequenced it to start from, because now you can only run it by RUN 797!

In Tips #57 I reported the discovery that printing to the disk from the TI- Writer Formatter, with the C option, really converted the carriage returns to trailing blank ASCII 32's, and I published a routine to strip them. I have found an easier way. First PF and C DSK... to convert the CRs to blanks. LF DSK... and SF DSK... to strip out those blanks, but that leaves the pestiferous tab line, so LF DSK... and PF DSK... again!

Here is a handy little "program that writes a program" which I often use to add instructions to programs.
     First key this in -

```
1 DISPLAY AT(24,5)ERASE ALL:"PRESS ANY KEY"
2 RESTORE 30721
3 REM
4 FOR J@=1 TO T@ :: READ @$ :: DISPLAY AT(J@,1):@$:" "
5 CALL KEY(0,K@,S@):: IF S@=0 THEN 5
6 NEXT J@
```

     Save it by -
     SAVE DSK1.MATRIX,MERGE
     Then key this in -

```
100 OPEN #1:"DSK1.MATRIX",VARIABLE 163,INPUT :: OPEN
     #2:"DSK1.MATRIX2",VARIABLE 163,OUTPUT :: L=179 ::
     FOR J=1 TO 6
```

```
110 LINPUT #1:M$ :: PRINT #2:CHR$(0)&CHR$(L+J)&
     CHR$(156)&CHR$(253)&CHR$(200)&CHR$(1)&"1"&
     CHR$(181)&CHR$(199)&CHR$(LEN(M$))&M$&CHR$(0)::
     NEXT J
120 CLOSE #1 :: PRINT #2:CHR$(255)&CHR$(255):: CLOSE #2
```

Run it to convert MATRIX into a merge format file MATRIX2 on DSK1. Then key this in. Do not change line numbers -

```
100 DISPLAY AT(3,1)ERASE ALL:"DATAWRITER by Jim
     Peterson":"":" To be used to add instruc-":"tions to
     programs."
110 DISPLAY AT(7,1):"Type and Enter the instruc-":"tions
     in single lines. They":"will be written to a
     D/V163":"file. When finished, enter"
120 DISPLAY AT(11,1):"END":"Then enter NEW, then
     MERGE":"DSK1.@DATA, then RUN.":"If everything is OK,
     load":"the program, be sure the"
130 DISPLAY AT(16,1):"lowest line number is
     higher":"than 6 and the highest is":"lower than
     30721, then merge":"in the @DATA file."
140 DISPLAY AT(24,5):"PRESS ANY KEY" :: DISPLAY
     AT(24,5):"press any key" :: CALL KEY(0,K,S):: IF S=0
     THEN 140
150 OPEN #1:"DSK1.@DATA",VARIABLE 163,OUTPUT :: CALL
     CLEAR :: DEF L$(X)=CHR$(120)&CHR$(X)
160 L=L+1 :: ACCEPT AT(12,0):M$
170 IF M$<>"END" THEN PRINT
     #1:L$(L)&CHR$(147)&CHR$(199)&CHR$(LEN(M$))&M$&CHR$(0)
     GOTO 160
180 REM KEEP THIS LINE OPEN
190 PRINT #1:CHR$(0)&CHR$(3)&"T@"&CHR$(190)&CHR$(200)&
     CHR$(LEN(STR$(L-1)))&STR$(L-1)&CHR$(0)
250 PRINT #1:CHR$(255)&CHR$(255):: CLOSE #1
```

Enter MERGE DSK1.MATRIX2, then SAVE it and try it out.
     Memory full!
     Please tell your friends about my TI-PD catalog. I put a lot of work into that, and am not getting many orders!
     Jim Peterson   o

Quantity of each letter in the letter pool

A = 11	G = 4	L = 5	Q = 1	V = 2
B = 2	H = 2	M = 2	R = 7	W = 2
C = 2	I = 11	N = 7	S = 6	X = 1
D = 5	J = 1	O = 9	T = 7	Y = 2
E = 14	K = 2	P = 2	U = 5	Z = 1
F = 2				

Total letters in letter pool: 115

Use the chart above as a reference during the game.

Playing the game

For disk version only: Make sure that the Extended BASIC command module is inserted into the command module port of your console. Turn on all peripherals, then turn on the console. When the master title screen appears, press any key to display the module's main menu. Now select Extended BASIC from the menu and wait until the word READY appears on your screen. Insert The Game of Wit program disk into disk drive #1. Type in the following statement: RUN "DSK1.GAMEOFWIT"

For cassette version only: make sure that the Extended BASIC command module is inserted into the command module port of your console. Turn on your monitor or TV set, then turn on your console. When the master title screen appears, press any key to display the module's main menu. Now select Extended BASIC from the menu and wait until the word READY appears on your screen. Insert The Game of Wit program cassette into your cassette recorder. Type in the following statement: RUN "CS1" and follow the loading instructions on your screen.   o

Forth to you too! Session 2

Author unknown

You have determined which of the editors suits you and found a display colour you like. They could be entered from the keyboard each time FORTH is booted. But there is a better method: let the disk do it for you! To begin with we will use the simple (and later on a more elegant) way. (If you have not made up an overlay yet, better do it now, else editing is not going to be easy. Programming in Forth is done by editing screens and the various editing functions are made a lot easier if you can refer to the overlay.)

So boot your Forth disk again and when the Menu shows up, enter either –EDITOR or –64SUPPORT. Now get out your manual and go to Appendix I (Contents of the Disk) and look at screen 3. This is the one that gives you the first inkling that something is going on by displaying "BOOTING". So you get an idea of the way Forth works, let us scan its contents before going on.

Line 0: The parentheses ( ) act like a REM in BASIC, so we see that it is called the Welcome Screen. GOTOXY is like DISPLAY AT, note the coordinates 0 0 preceding it.

Line 1: Forget the BASE->R for now, but let us do something with HEX. From your keyboard enter

HEX 83C2 DECIMAL .

Do not forget the period, actually a Forth word called Dot. (Look up each word in the Glossary!) What did you get? -31806 is correct. In plain English line 1 states: switch to Base 16, put >10 (16) on the stack, and C! (C-store, see page 17, Glossary) it at 83C2. This is how Forth does the CALL LOAD to disable FCTN[=] (Quit). (You have seen that one before!)

Line 2: DECIMAL returns us to Base 10, ignore the (84 LOAD), 20 LOAD loads SCREEN 20 (look at scr #20 and you will see that it is the menu which appears at boot time. 16 SYSTEM is CALL CLEAR (more about System Calls later) and finally MENU displays the menu. Take a moment to digest this, as it gives some idea as to how Forth works. The command 20 LOAD booted scr #20 at which time a new Forth word was compiled (see scr #20, line 1). MENU is now part of the Dictionary. Any time MENU is invoked, Forth looks it up and executes it. Try it, enter MENU. You get the menu and 'ok'. If you enter something Forth cannot find you will see a '?', sometimes followed by an error message (see Appendix H).

OK, back to the Welcome Screen. But now let us put it on display. Enter 3 EDIT and watch it come up. Skip to line 4 and note that here we have the menu words defined, i.e. : –EDITOR 34 LOAD ; etc. The first word after a ':' is the new word being added to the Forth's dictionary. Any words that follow must already be in the dictionary, otherwise the word being defined cannot be compiled. The definition ends with ';'. Move the cursor to line 12 and enter –EDITOR (or, if that is your choice, –64SUPPORT). Now, if you chose the regular editor, you can type the number from your SEE experiment followed by 7 VWTR. Now hit FCTN[9] (escape) to get out of the edit mode. Your additions to screen 3 are at this time only in a buffer, not on the disk. In order to record them on the disk you must enter FLUSH.

Remember: every time you edit a screen you must FLUSH, otherwise all your efforts will be for nought.

So let us check if your edit was successful. Enter COLD. (If you are using the 64 column editor enter TEXT COLD.) This word is like NEW in Extended BASIC, except you do not have to do anything else, Forth will re-boot. (It will take longer now because you are booting the editor also.)

Now let us recap: you have 'edited' screen 3 so it boots your editor and sets up the screen colour for you.

This was done while in the EDIT mode. You have also worked in the 'interactive' mode when you defined the word SEE to determine your colour choice. In this mode you can try out your definitions before you use them in a program. You will find this to be tremendously helpful because unlike BASIC there is no need to RUN the whole program to find out what happens.

Having worked my way into TI-Forth the hard way, I will leave you with a few suggestions which I feel will be helpful. As you encounter a new word look it up in the Glossary (Appendix D of the manual) to see what it is supposed to do. Mark the chapters and appendices in your TI-Forth manual for easier access to them. You will be using it frequently because, even though it may not seem so at first, it does contain a lot of information. Get a Forth book, preferably Leo Brodie's Starting Forth. It is sold in many bookstores and software houses. The manual (Appendix C) explains the differences between fig-Forth, which Brodie uses, and TI's implementation of it. Though it may read like Greek, scan through the manual. As we go along you might just remember having seen something that rings a bell. (Finding it again may be something else!)

Author's Note:
Since these tutorials were written, a new edition of Starting Forth has come out. In it, Brodie uses Forth-83 rather than fig-Forth. If at all possible try to get a copy of the old book, otherwise Appendix C of the TI-Forth manual will be meaningless and may lead to added confusion.

Forth Tidbits

by Lutz Winkler, USA

F-TIDBIT #1: True Lower Case for Forth

Your favorite CHARA1 file from TI-Writer (several versions exist) can be installed on your Forth disk for true lower case. Screen 19 of the system disk is only partially used by the Forth kernel, leaving sufficient space for the character definitions of ASCII 32 through 127. The parameters given below assume a two drive SS/SD system. For other configurations it will be necessary to adjust them accordingly. In case you prefer the 64 column editor, the following does not affect the display of its tiny characters. (There is no way to improve them.)

Step 1: Copy the CHARA1 file to a clean, initialized disk. Any disk manager can be used.

Step 2: Boot Forth and place the disk with the CHARA1 file in drive 2. The file will be found on screens 98 and 99. That is, the sectors which are needed are on these screens, the rest can be ignored.

Step 3: The file could be transferred now, but it is easier to combine it first onto one screen (100) before the transfer is made. The CHARA1 file starts on line 8 (address = 512) but the first 6 bytes (0 to 5) constitute the file header, so the address must be incremented by 6 (to 518). This is followed by 256 bytes (the character definitions for ASCII 0 to 31) which we do not need. Therefore, the starting address for the transfer is 98 BLOCK 774 +, destination is 100 BLOCK and we want to move the remaining 250 bytes of that screen:

98 BLOCK 774 + 100 BLOCK 250 CMOVE UPDATE FLUSH

The rest of the character definitions are found on screen 99 and 506 bytes have to be moved. They must follow what has already been put on screen 100:

99 BLOCK 100 BLOCK 250 + 506 CMOVE UPDATE FLUSH

Screen 100 now contains the entire set of definitions for displayable ASCII characters.

Step 4: Once again it is time to issue that old warning of "Do it on a backup disk!" With the Forth back-up disk in drive 1 and the character definitions on screen 100 in drive 2, the transfer is easily accomplished by:

100 BLOCK 19 BLOCK 256 + 768 CMOVE UPDATE FLUSH

Provided no errors were made and screen 33 (SYSTEM CALLS) is booted, the new character definitions are written to the PDT (pattern descriptor table) with

HEX 13 BLOCK 100 + 900 300 VMBW

For a quick check, it can be entered from the keyboard and some lower case characters typed. If everything works as expected, i.e., the display does not go haywire and lower case letters are properly shown, then the above statement should be placed on screen 3 (the welcome screen) to auto-boot the new character set along with whatever other auto-booting features may already have been installed there by the user.

While it is not necessary to put the entire character set into the PDT (the upper cases are there already) I use the whole range (32 to 127) because I have redefined the characters of my file. They are not only bigger but I have slashed the 0 and improved the lower cases. Also, I can easily put the same character set into the upper end of the PDT for conversion to inverse video. More about that in F-TIDBIT #2.

The procedure I have described makes use of space on the disk which is wasted otherwise. It does not require any modifications of other screens to accommodate the character definitions. A VMBW of >300 bytes does not add any noticeable delay when booting Forth.

F-TIDBIT #2: Adding Inverse Video

One feature which can enhance a program by making screen prompts and other information stand out, is 'inverse video' where text and background colours are reversed. With Forth this is easily done. Michael Jaegermann of Edmonton, Alberta, Canada, provided the routine which I adapted for use with a standard TI-Forth system.

```
0 ( INVERSE VIDEO )
1   33 CLOAD RANDOMIZE
2 BASE->R HEX
3     900 PAD 300 VMBR  PAD D00 300 VMBW
4   : INVERT  1000 D00 DO FF I VXOR LOOP ;   INVERT
5
6   : (^) COUNT OVER + SWAP DO I C@ 80 OR EMIT8 LOOP ;
7   : (IV) BEGIN KEY DUP 1F > WHILE 80 OR EMIT8 REPEAT
8         DROP DROP ;
9   : IVLIT   22 STATE @
10           IF COMPILE SLIT WORD HERE C@ 1+ =CELLS ALLOT
11           ELSE WORD HERE THEN ; IMMEDIATE
12  : IV"  [COMPILE] IVLIT STATE @
13           IF COMPILE (^) ELSE (IV) THEN ; IMMEDIATE
14
15 R->BASE
```

Line 1: Insures that the SYSTEM CALLS are booted which allows VMBR and VMBW to be used.
Line 2: VPD reads and writes are always easier in HEX.
Line 3: Read ASCII 32 to 127 to PAD and write them from PAD to the high order (ASCII 160 to 255) area in PDT. If you have put a character set on screen 19 (see F-TIDBIT #1) replace this line with : 13 BLOCK 100 + D00 300 VMBW.
Line 4: INVERT performs a VXOR on the high order character set to turn off-pixels on, and on-pixels off.
Line 6: (^) converts a string to inverse video.
Line 7: (IV) accepts input from keyboard for inverse video.
Line 9: IVLIT is WLITERAL adapted for IV purposes, delimiter is ASCII 34 (") instead of BL (ASCII 32).
Line 12: IV" (used in place of .") will display or compile text following it in inverse video until delimited by ".

Usage: IV" <ENTER>
       input from the keyboard is displayed in inverse video until <ENTER> is pressed again
    or you can use it in a word definition in the form of

: TEST IV" This is a test" ;
to display 'This is a test' when TEST is invoked.

F-TIDBIT #3: Improving the 40 Column Editor

In F-TIDBIT #1 I outlined how to install your preferred character set on the Forth disk and in F-TIDBIT #2 I provided information on how to get inverse video. Now we can put both of these features together to improve the 40 column editor. The procedure below not only corrects TI's omission of auto-repeat keys but also changes the character under the cursor to inverse video (unless it is a space). This merely requires that

1) there is a character set loaded into the PDT at >D00,
3) a few minor changes to screen 38, and
4) the addition of screen 41 (which is unused otherwise).

As to the first item, if you have installed your favorite character set on screen 19 (F-TIDBIT #1), all you need to do is a

HEX 13 BLOCK 100 + D00 300 VMBW

This puts a copy of it in the upper part of the PDT. To convert it to inverse video use INVERT from F-TIDBIT #2, then install the following routine on screen 41 of your system disk. (#41 should be a blank screen.)

```
0 ( BLINK AND DELAY FOR 40 COL EDITOR )
1   0 CLOAD DELAY
2 BASE->R HEX
3
4   : BLINK ( --- )
5         CURPOS @ DUP VSBR OVER OVER
6         DUP 21 < IF DROP 1E SWAP VSBW
7                ELSE 80 + VSBW
8                THEN
9         B0 0 DO LOOP ( blink rate )
10        SWAP VSBW ;
11
12  : DELAY ( --- )
13        800 0 DO LOOP ; ( repeat rate )
14
15  R->BASE
```

Now modify screen 38 as shown below:

```
0 ( SCREEN EDITOR 12JUL82 LCT) BASE->R HEX 29 CLOAD DELAY
1   : VED BOX SWAP CLS LISTL !CUR .CUR BEGIN ?KEY DUP IF CASE
```

(lines 2 through 9 remain unchanged)

```
10  7F OF -TAB ENDOF DUP 1F > OVER 7F < AND
12  BLINK DELAY ELSE DROP BLINK ENDIF AGAIN ; FORTH DEFINITIONS
```

(lines 13 through 15 remain unchanged)

Before you make this improved editor part of your auto-boot, you may want to try one more minor change. On line 3 of screen 34 modify the word BOX to read

: BOX 8F7 8F1 DO CC I VSBW LOOP ;

which makes the vertical sides of the box-shaped cursor 2 pixels wide and a bit easier to spot. This will provide you with a very satisfactory 40 column editor.

F-TIDBIT #4: In Between Disk Copier

From the time TI-Forth was first released a number of disk copying routines have been published. This was mainly in response to TI's implementation of FORTH-COPY which, being nothing more than a DO-LOOP of SCOPY, tediously copies one screen at a time while giving the disk drives a good workout. My complaint about those 3 pass copiers is that they necessitate re-booting and for the most part also disk swapping. In essence, not much time is saved. One might as well leave Forth and boot a disk manager. That, of course, is something a true fanatic is not going to do.

From my point of view, too much is made of speed anyway (I am in the enviable position of having plenty of time) and I am inclined to look for convenience. That is the reason my disk copier does not set any speed records, but it does away with disk swapping and re-booting. It copies from drive 1 to 2 (0 to 1, if you want to be finicky about it), takes only 720 bytes of memory, and 5 screens are read and written per pass. It will copy formats other than SS/SD, however, the disk formatting feature will only provide the format provided by your FORMAT-DISK word.

```
0 ( DISK-COPIER - 1 )   39 CLOAD AD  0 CLOAD COPY-DISK
1 BASE->R HEX      0 DISK_LO !   0 CONSTANT INC
2 : AT GOTOXY ;   ( skip this if already in your auto-boot )
3 : .SCR#  DUP 6 .R ; ( format for screen number display )
4 : READ5  5 0 DO I INC + .SCR#
5                   BLOCK I 400 * 1400 + 400 VMBW
6                 LOOP ;
7 : WRITE5  5 0 DO I 400 * 1400 + I INC +
8               DISK_SIZE @ + .SCR# BLOCK 400 VMBR UPDATE
9               LOOP  FLUSH  EMPTY-BUFFERS ;
10 : M1  F4 7 VWTR CLS 2 A AT ." Reading source screens" CR CR ;
11 : M2  4E 7 VWTR CLS 2 A AT ." Writing copy screens" CR CR ;
12 : M3  CLS CR ." FORGET INC to clear memory" CR ;
13 : MORE?  ( -- f ) CLS 0 ' INC ! 4 E AT ." Continue (Y/N) ? "
14          KEY DUP 59 = SWAP 79 = OR 0= ;
15 -->
```

```
0 (DISK-COPIER - 2 )
1 : TITLE  CLS 4 5 AT ." 0>------ COPY-DISK ------>1"
2             4 8 AT ." Insert source disk in drive 0,"
3             4 A AT ."        copy disk in drive 1,"
4             4 D AT ." Press 1 to format copy disk or"
5             4 F AT ." any other key when ready  "
6           KEY 31 = IF 1 FORMAT-DISK THEN ;
7 : XFER  DISK_SIZE @ 5 / 0
8           DO M1 READ5 M2 WRITE5  5 ' INC +!
9           LOOP ;
10 : COPY-DISK  EMPTY-BUFFERS
11               BEGIN TITLE XFER  F4 7 VWTR MORE?
12               UNTIL  M3 ABORT ;
13
14 R->BASE  COPY-DISK  ;S
15
```

The first parameters in M1 and M2 (as well as the one preceding 7 VWTR in DISK-COPY) change the text and background colours. You may substitute them to suit, just make sure to do it in hexadecimal. READ5 and WRITE5 are DO-LOOPs which read/write five screens at a time. XFER combines them into one DO-LOOP which derives its limit from DISK_SIZE. The top level word COPY-DISK is an indefinite loop which allows repeated execution by way of MORE?. About the >1400 in READ5 and WRITE5: this is the address of an unused area of VDP memory which is used as a buffer. As I said convenience, not speed, was my objective. So please excuse me while I fetch that second cup of coffee.

F-TIDBIT #5: Sorting Numbers

Inevitably a programmer encounters the need to sort some information into some logical sequence, either numerically or alphabetically. Quite a few sorting algorithms have been devised over the years and here we will deal with a Forth version of the Quicksort. This particular interpretation was written by Gary Nemeth. Based on it I have implemented a demonstration to sort twenty numbers.

```
0 ( QUICKSORT DEMO - 1  BASED ON GARY NEMETH'S QUICKSORT )
1 : NOT 0= ;              : 2/  1 SRA ;
2 : 2DUP  OVER OVER ;    : 2SWAP  ROT >R ROT R> ;
3 : 2OVER  SP@ 6 + @ SP@ 6 + @ ;
4
5 0 VARIABLE XX 40 ALLOT
6
7 : NO.INP  ( ---n )
8             QUERY INTERPRET ;
9 : ENTER  CR 40 0 DO I 2 / 1+ 14 .R ." INPUT : "
10                   NO.INP  XX I + ! CR 2
11                   +LOOP ;
12 : SHOW  CR 40 0 DO CR I 2 / 1 + 18 .R ." : "
13               XX I + @ 3 .R  2
14               +LOOP ;
15 -->
```

This screen sets up a number of operators (from NOT to 2OVER) which are needed for the sorting operation. VARIABLE XX is established to receive the numbers in random order by way of ENTER. After they have been sorted SHOW displays them. The words which perform the sort are shown on below:

```
1 ( QUICKSORT DEMO - 2 )
2 0 VARIABLE MIDDLE
3 : K@  2 * XX + @ ;
4 : K!  2 * XX + ! ;
5 : MID@  OVER - 2/ + K@ MIDDLE ! ;
6 : COMP  K@ MIDDLE @ - ;
7 : EXCH  2DUP K@ SWAP K@ ROT K! SWAP K! ;
8 : SORT  ( n1 n2 --- )
9         2DUP > IF DROP DROP
10              ELSE 2DUP 2DUP MID@
11      BEGIN SWAP BEGIN DUP COMP 0< WHILE 1+ REPEAT
12            SWAP BEGIN DUP COMP 0 > WHILE 1- REPEAT
13            2DUP > NOT IF 2DUP EXCH 1 -1 D+ THEN 2DUP >
14      UNTIL SWAP ROT 2OVER 2OVER - ROT ROT -
15         < IF 2SWAP THEN MYSELF MYSELF THEN ;
```

SORT obviously is the top level word. K@ and K! retrieve and store the integers from variable XX, while MID@, COMP and EXCH are needed to enable SORT to perform its function. (It is not the purpose of this article to explain the workings of a typical quicksort. This information should be obtained from other sources.)

The twenty number limit for the sort is arbitrary. It was chosen so the entire result of the sort can be displayed without scrolling off screen. If XX is modified to allot more (or fewer) cells and the loop counters in ENTER and SHOW are changed accordingly a larger (or smaller) number can be sorted.

Usage is as follows:
    ENTER - allows input of 20 positive or negative numbers
0 19 SORT - sorts them in ascending order (loop parameters must be put on the stack!)
    SHOW - displays the result of the sort

In spite of the length of SORT, you will be amazed at the speed of the sorting operation. In the next installment we shall explore the use of this routine for string sorts.

F-TIDBIT #6: Sorting Strings

In the preceding article I demonstrated how numbers can be sorted. Now we will see how the same routine can be adapted for sorting strings, i.e., putting them into alphabetical order. Most obvious changes are the use of an additional variable (ORDER) and a few added and/or modified words. Instead of moving strings into alphabetical order, pointers are stored in ORDER, similar to sector 1 of a disk containing pointers to the files in their alphabetical order while the files themselves remain in random order. There is also the -TEXT word from Brodie to compare strings, except that it is called =TEXT to avoid confusion with TI-Forth's -TEXT. Variable STRGS serves the same purpose as XX which stored numbers in the previous example. ENTER has become ENTER$, and similarly SHOW is SHOW$. SET is used to initialize ORDER.

```
0 ( STRING SORT DEMO - 1  BASED ON GARY NEMETH'S QUICKSORT )
1 : NOT 0= ;              : 2/  1 SRA ;
2 : 2DUP  OVER OVER ;    : 2SWAP  ROT >R ROT R> ;
3 : 2OVER  SP@ 6 + @ SP@ 6 + @ ;
4
5 : =TEXT  ( address1 u address2 --- f)
6           2DUP + SWAP
7           DO DROP 2+ DUP 2- @ I @ - DUP
8             IF DUP ABS / LEAVE THEN 2
9           +LOOP SWAP DROP ;
10
11 0 VARIABLE ORDER 40 ALLOT
12 0 VARIABLE STRGS 40 ALLOT
13
14
15 -->
```

Trials and Headaches with Myarc's DM5

by Ben Takach

The first part of this report published in the September issue dealt with the back-up problems, and the need of a streaming tape backup support. This part will deal with the problems encountered by using DM5, the Disk manager program for the hard and floppy disk controller.

1. The 80 track drive support.

The manual clearly states that 80 track drives are supported by the Hard disk control card and DM5. This option may be set by the dip switches on the HFDC card. Myarc provided 4 settings: 360k 40 track 16ms step rate, 360k 40 track 8ms step rate, 720k 80 track 2ms step rate and 1.44M 80 track 2 ms step rate for future expansion.

If the appropriate dip switch pair is set to the 80 track 1.44M (quad density) mode, then the particular drive will be treated as a 720k 40 track drive 2ms step rate. I have tried to use an IBM 1.4M HD drive on this setting, the disks however were only formatted to 1440 sectors. The same drive will format disks to 2880 sectors if the dip switches are set to 80 track 720k.

2. Compatibility with other cards in the PE-box.

The Hard disk card co-exists happily with the Mechatronic GRAM card, the TI-32K card, the Myarc 512K RAMdisk card, or the Rave RAMdisk. All my module files have been saved to the hard disk. The GRAM card loads the selected files at the appropriate command. Thus I am able to boot up in any format, eg., Extended BASIC, MiniMemory, Editor Assembler, TEII, TI-Writer, etc., without a cartridge in the module port or a diskette in the drives. The Hard disk control card will access any program or data file in any sub-directory of the hard disk.

DM5 however, is not so user friendly. It will refuse to boot up in the presence of the Mechatronic GRAM card unless the 512K Myarc RAMdisk is in the PE BOX. If the Rave EXP.MEM or the TI-32K card is fitted, then I have to pull out the Mechatronic card to enable DM5. This is a serious short coming, and so far I have been unable to come up with any remedy or get an explanation.

3. The Tricky DM5.

DM5 version 1.26 and version 1.29 are the two most recently released Myarc hard and floppy disk managers. It is a delight to use it if and when it runs. Getting it booted can at times be most difficult or even impossible. I had to shed all the surplus hardware from the PE box at first, to get it up and running. The main problem stems from the entirely different concept of a hard disk environment, compared to the floppy system. If you have a corrupted diskette, you can remove it from the drive and try its back up. The same option is not available to hard disk users. If a program file is corrupted on the hard disk, then the only option left is to run it from a diskette out of a floppy drive. If the program happens to be a multiple file chained assembler object code, which is configured to run out of drive 1, then all of the files with the exception of the load file have to be present in drive 1, otherwise the files will not be found.

The root of DM5's problem stems from this explicit rule of the TI99/4A disk memory organization. Naturally one has to install the essential DM5 program files on the hard disk in order to use the instant access option through the "CALL MDM" or "call mdm" from BASIC. If the program file MDM5 gets corrupted on the hard disk, then one can no longer load, run or access the disk manager DM5. Myarc did not provide an emergency back door! It does not matter how much back up copy one has on floppy disks, and from which drive one tries to run them (including the ram disk) the hard disk control card will try to read MDM5 off the hard disk.

Any mass storage media is likely to get corrupted. In fact they have a habit of getting corrupted at the most inopportune times. This of course is not such a big drama with floppies, as I have said before.

So the catch 22 situation with DM5 is that its virtue, to be ever present and only a CALL away, once it is installed on the Winchester drive, is also its downfall. The program logic does not prompt for a path name if the file MDM5 is corrupted or missing, it simply drops the bundle and locks up the system. It is easy to fall into an iron-tight inescapable trap without trying too hard.

The ever present Mr Murphy decided to test my patience and ability to get out of a straight-jacket a few weeks ago. Originally, the MD5 files were on the hard disk in two places.

```
WDS1.Root
  |
  |
  |
  |--DSK1.DM5 FILES
  |
  |
  |--DSK.DM5.DM5 FILES
  |    |
  |    |
  v    v
```
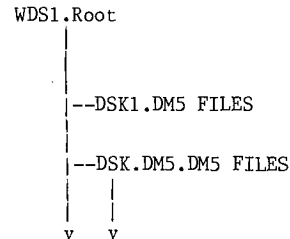
Thus the path names were: WDS1.DSK1.MDM5 and WDS1.DSK.DM5.MDM5.

Why did I stored these twice? The Myarc manual is not very clear regarding the residential status of the MDM program files. I understood it may be anywhere on the hard disk, thus I duly created a sub-directory for it, and saved the path together with the set up details as directed by the prompts in the SETUP screen of DM5. Subsequent CALL MDM's went unanswered, so I decided to copy the essential files on to the DSK1 sub-directory. This is where I committed the fatal error. I lost access to MDM forever. As it happened, I had to transfer the hard drive on to another system, thus it did not matter. What was my error? I should have repeated the set up procedure after the files were copied to the DSK1 sub-directory before leaving the DM5 environment. Once one quits, there is no way to get back ever again! Every time I called MDM, it found an unacceptably torturous path and hung up!

Things were back to normal after this episode, with the new hard disk drive, until the recent visit of Murphy. Here is the result of the post-mortem after one week of keyboard and card swapping gymnastics.

Once I managed to boot up DM5 again, I had copied the MDM files from the hard drive to floppy disk, then using the disk sector editor, printed out the MDM5 program file, which is the main part of the program. Almost half of the codes were missing! The immediate result was that I could not boot up the disk manager, no way, no how! It proceeded just so far to give me a brief glance at the disk manager's title screen, then took me back to the TI99/4A title screen.

I could have solved the problem by deleting the MDM5 file from the DSK1, or rename the sub-directory to DSK2 etc., or by copying a new MDM5 file to WDS1.DSK1. Alas, DM5 had to be up and running to do any of these.

Finally, after many frustrating hours, I pulled the 20 way IDC plug from the hard disk card after the system was booted, ran the DM5 program from the floppy, then re-connected the 20 way ribbon to the Hard-card after the program was up and running. I made a copy of MDM residing in the DSK sub-directory, to a diskette for further investigation, as I already mentioned above, then copied the necessary DM5 files onto the hard disk.

There are many possible ways to recover from such a predicament. All of these do need special software.

Disk Controls

by Michael A. Ballman, USA

This series of articles will explain some of the secrets of the TI disk controller. To start off you need to know several memory locations and commands. At all of these addresses the data is inverted.

```
>5FF0  status read address
>5FF2  track address read
>5FF4  sector address read
>5FF6  data from disk
>5FF8  command write address
>5FFA  track address write
>5FFC  sector address write
>5FFE  data to write
```

TYPE	COMMAND	BITS 7 6 5 4 3 2 1 0
I	Restore	0 0 0 0 h v R r
I	Seek	0 0 0 1 h v R r
I	Step	0 0 1 u h v R r
I	Step in	0 1 0 u h v R r
I	Step out	0 1 1 u h v R r
II	Read command	1 0 0 m b e 0 0
II	Write command	1 0 1 m b e A a
III	Read address	1 1 0 0 0 e 0 0
III	Read track	1 1 1 0 0 1 0 s
III	Write track	1 1 1 1 0 1 0 0
IV	Force interrupt	1 1 0 1 j k l n

```
h=1 load head at beginning
v=1 verify track register
u=1 update track register
m=1 multiple records
b=1 IBM format (TI uses IBM format)
e=1 enable 10ms delay for head settling
Aa binary count data marks (FB,FA,F9,F8)
Rr binary count for step speed (6,6,10,20ms)
jkln various interrupts (use '0')
```

Do not be too worried if you do not understand how any of these instructions are used. All will be explained by the end of this series.

The CRU base address for the disk controller is >1100 and must be in register twelve. With R12 loaded these bits can be used for various control functions with the SBZ and SBO assembly language instructions.

```
SBO +0  turn on card
    +1  motor on by toggling this bit
    +2  activates the ready line
    +3  sets head load line
    +4  selects drive called DSK1
    +5  selects drive called DSK2
    +6  selects drive called DSK3
    +7  selects side two of drive
```

Now for the first program segment. This part will read a track on a disk in drive one. The track can be protected and does not have to be formatted for this program to read it. (No you can not read a track then write it to copy a disk some of the control information will change.)

```
**************************************************
*                                                *
* READ TRACK                                     *
*                                                *
**************************************************
        DEF  TREAD          define start
TREAD   LWPI MYREG          load work registers
        LI   R12,>1100      set CRU address
        SBO  +0             turn on card
        SBZ  +5             not drive two
        SBZ  +6             not drive three
        SBO  +4             select drive one
        SBZ  +7             select side one
* zero track program goes here later
*AGAIN  BLWP @GET#          for later program
```

```
*       BL   @SETTRK        for later program
        LI   R2,>1000       set byte count
        LI   R10,TBUFF      buffer pointer
        BL   @SENDC         send command
        DATA >1B00          read track cmd inverted
        SBO  +2             enable ready
TREAD1  MOVB @>5FF6,R0      read byte
        INV  R0             make normal
        MOVB R0,*R10+       save byte
        DEC  R2             adjust byte count
        JNE  TREAD1         last byte? no
        SBZ  +2             disable ready line
        NOP                 display routine
        NOP                 goes here
**************************************************
*                                                *
* USE DEBUG TO VIEW DATA PUT BREAKPOINT          *
* WHERE THE TWO NOP'S ARE AND USE 'M'            *
* TO LOOK AT THE BUFFER 'TBUFF'                  *
*                                                *
**************************************************
STOP    JMP  STOP           wait here forever
*       JMP  AGAIN          for later program
SENDC   MOV  *R11+,R0       get command
        MOVB @>5FF0,R6      read status
        SLA  R6,1           get ready bit
        SBZ  +1             toggle drive on
        SBO  +1             *
        JOC  WRTCD          ready set? yes
        LI   R6,>7530       set-up delay
WAITL   SRC  R5,4           * so motor can
        SRC  R5,4           * get up to
        DEC  R6             * speed
        JNE  WAITL          delay end? no
WRTCD   MOVB R0,@>5FF8      send command
        SBO  +3             load head
        SRC  R5,8           kill time
        SRC  R5,8           kill some more
        B    *R11           return
MYREG   BSS  >20            work registers
        TEXT 'BSTART'       so start can be found
TBUFF   BSS  >1000          buffer
        TEXT 'BEND'         so end can be found
```

Here is the control information on the disk and a program to accept a track number from the keyboard. The FDC recognizes some data bytes as special when a write track command or a read command is given. These bytes and their meaning are:

```
F7-Write CRC characters used in error checking
F8-Data address mark (deleted data) Aa=11
F9-Data address mark (user defined) Aa=10
FA-Data address mark (user defined) Aa=01
FB-Data address mark (user defined) Aa=00
FC-Index address mark (hole in disk)
FD-Spare (no special meaning)
FE-ID address mark
```

When data is written it is intermixed with a clock bit between each data bit. In order to easily identify address marks they are written with some clock bits missing. The special coding is why data can not be written to the disk with the write track command.

```
x=data bit c=clock bit
1 1 0 0 0 1 1 1 =C7      1 1 0 1 0 1 1 1 =D7
cxcxcxcxcxcxcxcx         cxcxcxcxcxcxcxcx
 1 1 1 1 1 ? ? ?=F8,9,A,B,E  1 1 1 1 1 1 0 0=FC
```

When the FDC sees an F7 byte on write track, two bytes of data to be used for error checking are written. Remember two bytes.

GAP	AM	TRACK#	SIDE	SECTOR#	SECTOR-LEN	CRC
000000	FE	28	00	09	01	??

The ID field tells the FDC where it is. This information indicates track 40 and sector 9 on a single sided disk with a sector size of 256 bytes. By changing this information or putting it someplace else on the disk, the data in the sector will not be found with the TI DOS (disk operating system). This a type protection used prevent backup of their software.

Under the 'IBM' disk format (TI's also 01) the sector length is used as a shift value of 0 to 3 giving a sector length of 128 to 1024 bytes. If the 'b' bit in the read and write commands is a zero, the length byte is used as a multiplier. That is 16*(sector length) = bytes in sector, giving a sector length of 16 to 4096 bytes. (Do not use the reserved bytes F7 to FE.)

```
VREG1   BSS  >20            set-up registers
H30     DATA >0030          data for conversion
H07     DATA >0007          data for conversion
TRACK#  DATA >0000          track no. to read
SCREN0  DATA PRESS,>0000    text pointers
PRESS   DATA >02E2,22       address and length
        TEXT 'SELECT TRACK TO READ >'
GET#    DATA VREG1,BEGIN    BLWP pointers
BEGIN   LI   R0,>2000       byte to write
        LI   R1,>0300       bytes to clear
        LI   R2,>0000       address to start
        BL   @CLEAR         clear screen
        LI   R10,SCREN0     get pointer for screen
        BL   @TEXT0         write screen
```

```
***********************************************
* KEY SCAN                                    *
***********************************************
        LI   R1,>02F8       load write location
KEY     LWPI >83E0          use GPL registers
        BL   @>000E         scan keyboard
        LWPI VREG1          go to my registers
        CLR  R0             clear R0
        MOVB @>8375,R0      get key
        CI   R0,>FF00       check for no key
        JNE  KEY            yes. key pressed
KEY1    LWPI >83E0          use GPL registers
        BL   @>000E         go to scan routine
        LWPI VREG1          use my registers
        MOVB @>8375,R0      store key
        CI   R0,>0D00       was key ENTER?
        JEQ  ENTER          yes.
        CI   R0,>3000       was key zero or higher?
        JLT  KEY1           yes.
        CI   R0,>3A00       is key greater than 9?
        JLT  DECODE         yes. decode and display
        CI   R0,>4100       is key less than 'A'?
        JLT  KEY1           yes.
        CI   R0,>4600       is key greater than 'F'?
        JGT  KEY1           yes.
DECODE  MOV  R0,R5          move key to R5
        MOV  R1,R2          get write location
        INC  R1             adjust write location
        BL   @ADDST1        set-up write address
        MOVB R5,@>8C00      write key to screen
        SWPB R5
        S    @H30,R5
        CI   R5,>000A       is key a hex digit?
        JLT  SAVE           no.
        S    @H07,R5        make a hex digit
SAVE    SLA  R8,4
        SOC  R5,R8          store number in R8
        JMP  KEY            get next key
ENTER   ANDI R8,>00FF       mask off mistakes
        MOV  R8,@TRACK#     save track to read
        RTWP               return to read program
TEXT0   MOV  R11,R6         save return
TEXT1   MOV  *R10+,R0       get pointer to text
        MOV  R0,R0          was it last text?
        JEQ  RETN6          yes.
        MOV  *R0+,R2        get write address
        MOV  *R0+,R1        get length
        BL   @ADDST1        set up VDP address
WTEXT1  MOVB *R0+,@>8C00    write data
        DEC  R1             adjust count
        JNE  WTEXT1         done? no
        JMP  TEXT1          go for more text
***********************************************
* CLEAR SCREEN                                *
***********************************************
CLEAR   MOV  R11,R6         save return
        BL   @ADDST1        set-up VDP write address
WLOOPA  MOVB R0,@>8C00      write byte
        DEC  R1             adjust count
        JNE  WLOOPA         last byte? no.
RETN6   B    *R6            return
```

```
***********************************************
* SET-UP VDP WRITE ADDRESS                    *
***********************************************
ADDST1  AI   R2,>4000       make a write address
        SWPB R2
        MOVB R2,@>8C02
        SWPB R2
        MOVB R2,@>8C02
        B    *R11           return
```

Now put these two segments together.

A while ago someone ask me the size the files in the disk directory. I have now come across the answer. The files are (decimal) 38 bytes long. Any other size will give an error.

I once asked what all the names and uses of subprograms in the disk service routine. I now have the answer. If you want to know just ask.

Now this program needs:

1) Error checking so you cannot try to read past the last track.
2) Some way to quit the program.
3) A display routine.
4) A way to switch to the second side.

I feel there is some merit in typing in programs so the only way this program is available from me will be on paper. It is less than 200 lines long so it should be easy to input it in with one sitting.

I have not used some of the commands available on the FDC (WD-1771). Here is how they work:

SEEK: The track register at >5FF2 must have the current track number, load the data register at >5FFE with the desired track, then send the SEEK command. The FDC will then issue the necessary step signals to arrive at the requested track. If the 'v' bit is set in the command the FDC will then check of the track number on the disk.

STEP: This command just steps the head the same direction as the last head move.

READ: Position the head on the correct track, load the sector register at >5FF4 with the desired sector, then send the read command. The track register and the track data on the disk must agree! Data will be at >5FF6. If there is a CRC error in the ID field this command will abort.

WRITE: This command is like the READ command except data is put at >5FFE to be written to the disk.

READ ADDRESS: When sent this command the FDC will read the next ID data field. The data will be at >5FF6 and will be six bytes long. The bytes will be:
1) Track number as it appears on the disk
2) Side number. Not use by the FDC.
3) Sector number.
4) Sector length.
5) First byte of the CRC error checking data.
6) Second byte of the CRC data.
   If there is a CRC error, this command, or any which check the CRC, sets the status bit number three. Remember these bits are not numbered backwards like the bits on the TI.

WRITE TRACK: Just like the READ TRACK except it writes data instead of reads it. Do not forget the different addresses for read and write.

FORCE INTERRUPT: When this command is received the FDC stops whatever it is doing and goes not busy.

Now the status bits at >5FF0:

7) All commands- NOT READY
6) Write and type I commands- WRITE PROTECTED
   Read- 1st bit for data ID byte decode

```
    5) Type I- Head engaged, Write- Write fault.
       Read- 2nd bit for data ID byte decode
    4) Type I- Seek error
       Others- Requested data not found
    3) All commands- CRC error
    2) Type I- Track zero, Others- Lost data
    1) Type I- Index (beginning of the track)
       Others- Requesting service of data register
    0) All commands- BUSY

***************************************************
* SET THE DRIVE TO TRACK ZERO                     *
***************************************************
        LIMI  >0000      disable interrupts
        BL    @SENDC     send instruction
        DATA  >F700      seek track zero
        BL    @CBUSY     wait until done
        CLR   R7         pointer to current track
***************************************************
* MOVE THE ABOVE SECTION OF CODE TO THE PLACE     *
* RESERVED FOR THE ZERO TRACK ROUTINE AND DELETE* *
* THE '*' AT THE BEGINNING OF THE THREE LINES     *
* MARKED FOR FUTURE USE.  DELETE THE STOP LINE    *
***************************************************
SETTRK  MOV   R11,R13    save return
        MOV   @TRACK#,R1  get desired track numb.
        C     R1,R7      compare desired / actual
        JLT   SOUT       if less then step out
        JEQ   RET13      equal then return
SIN     BL    @CBUSY     check busy bit
        BL    @SENDC     send command
        DATA  >BD00      select step in cmd
        INC   R7         adjust track pointer
        C     R1,R7      is drive at wanted track
        JNE   SIN        no.
        JMP   RET13      return to main program
SOUT    BL    @CBUSY     check busy bit
        BL    @SENDC     send command
        DATA  >9D00      select step out cmd
       .DEC   R7         adjust track pointer
        C     R1,R7      is drive at wanted track
        JNE   SOUT       no.
RET13   B    *R13        return
***************************************************
* CHECK FOR DRIVES BUSY                           *
***************************************************
CBUSY   MOVB  @>5FF0,R0  get status
        SLA   R0,8       find busy flag
        JNC   CBUSY      is it set? n
        B    *R11        return
        END  ·
```

Merge these three segments together in the order presented, then make the move and changes noted in this last segment, then assemble the result with the 'R' option. If when you assemble this code you want a printed copy; type in the correct print device. If you have PIO include a '.' after the name.

This is the last of this series. If you have any questions please write (I repeat write) to me at the following address.

If you have any Assembly Language or Hardware questions you would like answered, please send them to:
    Miami Users Group
    PO Box 650955
    Miami, FL 33265-0955   o

For Sale

3 x Consoles (1 black) UHF, $90 each
Speech Synthesiser $50
Terminal Emulator $20
BMC BX80 printer $220
6 x modules $10 each
Assembly Language and Systems Handbook $20

Phone Homero on 042-286585

RAMdisk or hard disk which way to go?

by Lou Amadio

If you have owned a fully expanded system for some time and are thinking of expanding further, you may be wondering what options are available to you with regard to mass storage. Now that the cost of static RAM chips has slowly increased to a point where we are paying approximately $1 per kilobyte, the question of which way to expand, RAMdisk or Hard Disk, becomes very relevant.

Up until recently I had been of the opinion that RAMdisks were the only way to go, so let us have a look at the relevant statistics, or what you get for your money:

RD Size	Basic	SRAM	Total
96Kb	$80	$96	$176
192Kb	$80	$192	$272
384Kb	$80	$384	$464
512Kb	$85	$512	$597
1024Kb	$90	$1024	$1114

    "RD" = RAMdisk
    "SRAM" = Static RAM chips
    "Basic" refers to the cost of a RAMdisk kit with
everything except the SRAM chips.

As you can see, a small RAMdisk (96Kb) will cost you almost $180 while a DSDD RAMdisk will set you back over $460.

On the other hand, a hard disk system has the following costs:

Myarc HFDCC ---------------- $380
10Mb Hard Disk (used) ------ $100

          Subtotal $480

The above costs assume that you will be mounting the Hard Disk inside the TI Peripheral Expansion Box (PEB) and powered by a modified PEB power supply.

A stand-alone power supply and box for a hard disk costs approx $85 on a do it yourself basis. (See TND Sept '89 for more information on how to power a Hard Disk).

Given the above costs, which way would I go?

My first choice would be a minimum specification RAMdisk. By minimum specification I mean enough SRAM to run the John Johnson Menu programme as well as a few important utilities such as a disk manager and a word processor. For this you could probably get away with 64K or even 32K of SRAM. So for little more than the cost of a floppy drive you can have a very powerful menu driven system which will change the way you use your computer, and make you wonder how you ever got along without it.

After that, you either add extra memory chips to your basic RAMdisk, if and when you can afford them, or wait until the price of SRAMs comes down to a reasonable level.

The second choice is not as easy: Given the above costs, and the uncertainty of the prices of SRAMs, you can bite the bullet and go for a Hard Disk system. After the shock to your wallet has subsided somewhat, you will find that not only is the Hard Disk a very useful addition to your system, but it nicely compliments the RAMdisk.

Perhaps then, when taken in the right order, it is not such a hard choice after all.   o

From the Trenches

by Werner Kanitz

It did it again!!!

Half way through a page of typing the RAMdisk died. Oh woe is me!

"Well there y'go complain enuff an' some one'l come t' the rescue (or the devil will come to get his dues)."

Remember that most wonderful of devices, that circuit board with all those fancy chips stuck onto it gingerly inserted into a vacant slot in that most useful of add-ons the PEBox, backed up as it were with a number of batteries in case there is a lack of power. I referred to it above, that devil's own device, the RAMdisk.

For years I managed to avoid constructing one due in no small way to the fact that I am more into spitting chips than soldering them. I could however not elude that silver tongued fox Herr Amadio. I was persuaded to buy a ready made model. Once installed the boy thought she was a ripper, and so it was, when it was running. Blessed are those without for they shall not suffer the aggravations of blips and surges, and require to reinstall their ROS every few weeks.

"It is the price of technology they reckonned yer tryen t' make the thing do tings it wozn't meant t' do." That is not much consolation.

Never mind, Lou to the rescue. "Ve can fix det midt ein chip preprogrammed wif der operating zystem burnt into it for ever (EPROM)." Great news, no more worries, get me one. Done. Thinks I, get it home pop out the old, slip in the new and you are cooking with gas so to speak.

"B!**sh%t!!"

You need different software to drive it, no problem, I will install it. You need to make hardware changes to your RAMdisk. (*@!*, is nothing simple!)

I do not know anything about oxy-cutting and welding even on this minute scale. It is getting to the point where I am tempted to cut my losses and go PC. At least everything is plug in. G*%, I hate electronics!

The Rhyme to an Ancient Computer.
    by Werner "Wordsmith" Kanitz

I don't know why I bought the thing
Some said it was a bargain
The cost was cheap free games rolled in
But then we reassessed again
For nothing there was serious

Chips and cards were everywhere
And the number of free slots did shrink
Chips and cards were everywhere
Still not enough I think.

Its chips were there its slots were free
Its keys cold and inviting
Its skin was white as leprosy
My nightmare TI PC was she
Who drains man's purse expanding

For joy for joy this glorious toy
The men would play and fiddle
Expanding up was all the go.
First box and then the extra memory
The RAMdisk followed close behind
To my everlasting misery.

I can't believe I am here today
Dad I but said, Mazz go away,
When first my mind was tempted
But then again as you can see
My soul could once again be free
If this association would be ended.

(my apologies to Mr Colleridge)   o

```
STRBUF BSS   256        *This is where strings will be
                         transferred between assembly and
                         Extended BASIC.
NEWREG BSS   32         *This is where our workspace will
                         be.  32 bytes are reserved since
                         16 registers X 2=32
START  LWPI  NEWREG     *This instructs the computer to
                         use the memory locations
                         indicated by NEWREG as the
                         workspace registers.
       CLR   R0         *Loads R0 with 0
       LI    R1,1       *Loads R1 with 1 (first
                         parameter)
       BLWP  @NUMREF    *Gets the value of X from CALL
                         LINK
       BLWP  @XMLLNK    *Converts X from a
       DATA  >12B8       floating point number to an
                         integer which is in >834A
       MOV   @>834A,R3  *Saves the integer X into R3
                         since the the word at >834A may
                         be altered by STRREF.
       LI    R1,2       *Loads R1 with 2 (second
                         parameter)
       LI    R2,STRBUF  *Address of where S$ should be
                         copied to.
       LI    R0,255     *Loads R0 with 255 (largest
                         possible string)
       MOV   R0,*R2     *Makes @STRBUF=255 this allows S$
                         to be up to 255 characters or
                         less.
       CLR   R0         *Restores R0 to 0 since S$ is not
                         an array.
       BLWP  @STRREF    *Copies S$ into STRBUF+1
       LI    R2,STRBUF+1 *Tells the routine below where
                         to find S$.
       MOV   R3,R0      *Tells the routine below where to
                         start writing the message to the
                         VDP screen.
                         (The following LOOP routine is
                         identical to that in Part 3.
                         Please Refer for explanation on
                         entries.)
LOOP   MOVB  *R2+,R1
       AI    R1,>6000
       BLWP  @VSBW
       CI    R2,STRING+4
       JLE   LOOP
       CLR   R0
       MOVB  R0,@>837C
       LWPI  >83E0
       RT
```

With this assembly routine you can control where and what your message is written to the VDP screen. Keep in mind that unlike DISPLAY AT, if you let X be 767 and S$ is 256 bytes long , screen wrap will not occur with S$ being printed at the top of the screen, but rather, S$ will be written to the VDP being used for the first 31 character definitions, ASCII 0 to 30. Since the memory for these characters is being used to hold some of Extended BASIC's system variable, queer things could occur.   o

For Sale

2 x 35 track floppies in box with power supply $100
Speech Synthesiser $50
Peter Schubert Mini PE System (no box) $240

Phone Kevin on 042-616301

Making your own Dictionaries #3, by Geoff Trott

This is the third article on writing programs in c99 using the example of reconstructing dictionary files for the Dragonslayer Spell Check program. Of course the main part of the whole task is the editing of the dictionary files, deleting and adding words and this remains to be completed. If anyone is interested in this project, either in the dictionries as they are, or in progress reports, I suggest they contact me.

This month I will explain how the string procedures work. These have been modified so if you want to compare these with the originals, you will have to do the comparison yourself. I have tested each of these procedures and am confident that they do work as they are supposed to. I have changed them so that they do not need to be compiled with your program but they are loaded at the same time as the main program and the support procedures. To this end you must include the statement "#include DSK.H.STRINGI" near the start of your program. This looks in disk called DSK.H. (or sub-directory DSK.H.) for the file which contains the following:

```
extern strlen(),strcmp(),stncmp(),index(),rindex();
extern strcat(),stncat(),strcpy(),stncpy();
```

These lines define the procedure names as being external to this program and so not to give an error when one of these names is used. Then you may use any of these procedures. They perform the following functions:

strlen(s): returns the length of the string s
strcmp(sl, s2): returns a value which shows the result of the comparison of two strings, sl and s2
stncmp(sl, s2, n): returnsa value which shows the result of the comparison of the first n characters of sl and s2
index(s, c): returns the position of the first occurence from the left of the character in the string s
rindex(s, c): returns the position of the first occurence from the right of the character in the string
strcpy(sl, s2): copy s2 into sl
stncpy(sl, s2, n): copy the first n characters of s2 into sl
strcat(sl, s2): copy s2 onto the end of sl
stncat(sl, s2, n): copy the first n characters of s2 onto the end of sl

What is a string? Well, a string is a number of characters which travel together. In C, a string is an array of characters terminated by a NULL or 0 character. The name of an array is a pointer to the first element in the array. So we are going to have to understand pointers if we are going to see what these procedures do. To look now at the code, the first few lines are comments followed a statement to give a value to a name (NULL = 0) and then two lines of assembler code enclosed in the "#asm" and "#endasm" statements, which ensures that the procedure names are available to other programs at load time. This could also be done with the "#entry" statement (see Craig Sheehan's tutorial). Then on to the first procedure.

```
/*
** string function library
**
** contributed by :
** Tom Wible
** 203 Cardinal Glen Circle
** Sterling, VA
** USA 22170
**
** Modified by Geoff Trott, 29 June 1989
** 20 Robsons Road, Keiraville, NSW 2500, Australia
**
*/
#define NULL 0
```

```
#asm
  DEF STRLEN,STRCMP,STNCMP,INDEX,RINDEX
  DEF STRCPY,STRCAT,STNCPY,STNCAT
#endasm

/*   returns string length   */
strlen(s)
char *s;
{
  int n;
  n = 0;
  while (*s++) n++;
  return (n);
}
```

The string name passed to the procedure must be declared before the procedure starts. In this case it is an array of characters so the name is a pointer to a character. "*s" means the contents of the memory cells pointed to by "s" and in this case is a character. Inside the procedure a local variable is declared which is used to count the number of characters in the string. This is first initialized to zero and then as long as the next character in the string is not NULL, it is incremented. When the NULL character of the string is reached, n has the number of characters scanned and this is returned as the result of the procedure, like a function.

Note the use of the ++ operator to increment both the pointer to the string and the counter. This procedure could have been done using other operators as we will see with the other procedures.

```
/*   compares sl to s2,
return: number<0 if sl before s2;
        0 if sl same as s2;
        number>0 if sl after s2   */
strcmp(sl, s2)
char *sl, *s2;
{
  int rl2;
  for ( ; (*sl) != NULL; sl++, s2++) {
    rl2 = (*sl) - (*s2);
    if (rl2) return (rl2);
  }
  rl2 = (*sl) - (*s2);
  return (rl2);
}
```

Comparison of strings is trickier but all that happens is that each character of the two strings are compared, one at a time, until they differ or one is a NULL. If a NULL appears before they differ, the strings are identical, whereas if they differ the difference is returned as the result. The work is done by a "for" loop. This one has no initialization statement with both pointers to the strings being incremented each time round the loop which terminates if the character from sl is NULL. Note the "!=" meaning not equal to. If the characters are not the same then the "if" statement returns that difference as the result of the procedure. When the "for" loop terminates, the result is the difference between the last characters.

```
/* compares n chars of sl to s2,
return: number<0 if sl before s2;
        0 if sl same as s2;
        number>0 if sl after s2   */
stncmp(sl, s2, n)
char *sl, *s2;
int n;
{
  int rl2,i;
  if (n < 1) return(0);
  for ( ; n > 1; --n, sl++, s2++) {
    rl2 = (*sl) - (*s2);
    if (rl2) return (rl2);
    if (*sl == NULL) return (rl2);
  }
  rl2 = (*sl) - (*s2);
  return (rl2);
}
```

This comparison has the added complication of only looking at the first n characters. The "for" loop is modified to make sure that only n characters are examined. There are then two exits from the function within the loop and one check at the end of the loop. Note that the "==" in the "if" statement means is equal to.

```
/*   returns location of c in s   */
index(s, c)
char *s, c;
{
  int n;
  for (n = 1; *s != NULL; n++, s++) {
    if (*s == c) return (n);
  }
  return (0);
}
```

The "for" loop now initializes n to 1 and increments n and the pointer to the characters in the string while the loop finishes when the NULL character is reached. In that case the character has not been found and zero is returned. If the character is found the current value of n is returned.

```
/*   returns location of c in s from right   */
rindex(s, c)
char *s, c;
{
  int n;
  for (n = 0; *s != NULL; n++, s++) ;
  s--;
  for ( ; n > 0; s--, n--) {
    if (*s == c) return (n);
  }
  return (0);
}
```

The first "for" loop finds the length of the string and sets the pointer to the last character. The second "for" loop searches for a character match from the end of the string to the start. The use of "--" decrements the pointer and the counter.

```
/*   copy s2 into s1   */
strcpy(s1, s2)
char *s1, *s2;
{
  while ((*s1++) = (*s2++)) ;
  return;
}
```

This procedure returns a string which can only be done through a pointer if there are more than one item, in this case a number of characters. This is done by the test of the "while" loop (which is empty). Here the power of C can be seen as each pointer is incremented after one character is stored in the other string. The single "=" means to assign the value, not to test the result. The "while" loop continues until the character value is NULL, the end of the string.

```
/*   copy at most n chars of s2 into s1   */
stncpy(s1, s2, n)
char *s1, *s2;
int n;
{
  for ( ; n > 0; n--, s1++, s2++) {
    *s1 = *s2;
    if (*s1 == NULL) return;
  }
  *s1 = NULL;
  return;
}
```

For this case there is too much to be done to use a simple "while" loop as before, so a "for" loop is used with one exit if the string is less than the length n. If not then the new string must have a NULL as the last character.

```
/*   concatenate s2 to end of s1   */
strcat(s1, s2)
char *s1, *s2;
{
  while (*s1++) ;
  s1--;
  while ((*s1++) = (*s2++)) ;
  return;
}
```

To concatenate two strings, first the pointer must be moved to the end of the first string and then the second string copied to the end of that string. Very simple and elegant!

```
/*   concatenate at most n chars of s2 to end of s1   */
stncat(s1, s2, n)
char *s1, *s2;
int n;
{
  while (*s1++) ;
  s1--;
  for ( ; n > 0; n--, s1++, s2++) {
    *s1 = *s2;
    if (*s1 == NULL) return;
  }
  *s1 = NULL;
  return;
}
```

Concatinating at most a number of characters is more complicated and requires the "for" loop, but is just a combination of other procedures.

I hope that you find some value in looking at these procedures but I suggest that you have a book on C close at hand while you do it. There are other procedures to look at and try out with the c99 package from Clint Pulley of Canada. I commend the language to you if you have a program you wish to write and have execute at a faster speed than it would in Extended BASIC. C is one of the languages of the future for just such uses and is very powerful in the assembler language sense as well as having many of the features of high level languages. o

PEB to I/O Port Connections

PEB #	Function	I/O #	PEB #	Function	I/O #
1	+9V	NC	31	A12(H)	11
2	+9V UNREG	NC	32	A13(H)	15
3	GND	21	33	A10(H)	6
4	READY(H)	12	34	A11(H)	8
5	GND	23	35	A8(H)	14
6	RESET(L)	3	36	A9(H)	18
7	GND	25	37	A6(H)	29
8	SCLK(H)PULLUP		38	A7(H)	17
9	LCP(L) PULLUP		39	A4(H)	7
10	AUDIO	44	40	A5(H)	5
11	RDBEN PULLUP		41	A2(H)	20
12	PCBEN PULLUP		42	A3(H)	10
13	HOLD(L)PULLUP		43	A0(H)	31
14	IAQ(H) PULLUP		44	A1(H)	30
15	SENILA PULLUP		45	AMB(H) PULLUP	
16	SENILB PULLUP		46	AMA(H) PULLUP	
17	INTA(L)	4	47	GND	23
18	LOAD(L)	13	48	AMC(H) PULLUP	
19	D7(H)	34	49	GND	25
20	GND	27	50	CLKOUT(L)	24
21	D5(H)	38	51	CRUCLK(L)	22
22	D6(H)	36	52	DBIN(H)	9
23	D3(H)	42	53	GND	27
24	D4(H)	35	54	WE(L)	26
25	D1(H)	40	55	CRUIN(H)	33
26	D2(H)	39	56	MEMEN(L)	32
27	GND	21	57	-18V UNREG	NC
28	D0(H)	37	58	-18V UNREG	NC
29	A14(H)	16	59	+18V UNREG	NC
30	A15/CRUOUT(H)	19	60	+18V UNREG	NC o

Telco setup for Smart Modems

by Larry Saunders

```
Select : AUTO DIALER
Press : M
Press : 1
Type : TEXPAC BBS 2400 Baud rate
Phone : 3191009
Terminal : ANSI
Baud rate : 2400
Parity : 8N1
Width : 40
Select : Option Setup
Select : Initialization Setup
Type : ATSO=0
Select : Dial String
Type : ATDP
Select : Hangup String
Type : ATH!
Goto : Setup Option Screen
Press C for save Changes
```

Now Telco is setup for smart modem. If you want a different Baud rate for BBS, setup AUTO DIALER the same as at start but put in a different baud rate.

```
Ideal setup should read
1> TEXPAC BBS 2400 Baud rate
2> TEXPAC BBS 1200 Baud rate
3> TEXPAC BBS 300 Baud rate
4> AUSTPAC's 2400 Baud rate
```

Just PRESS number and Telco will do the rest. If you Select 1, you will connect at 2400, If you select 2 or 3 you will connect at 1200 or 300 Baud rate .

Intelligent modem setup

```
    From RS232

    2  3  6      7   20
    |  |  |      |   |
    3  2  20-4   7   6

    To Modem
```

For those with TELCO, setup the AUTO DIALER three times for BBS, one at 300, one at 1200 and one at 2400. When you use the auto-Dialer it will change TELCO to the rating that the AUTODIALER is set to for the program number.
For eacmple:

```
1 TIsHUG BBS 300
2 TIsHUG BBS 1200
3 TIsHUG BBS 2400
```
If you select 1 the smart modem will connect you at 300
If you select 2 it will connect at 1200
If you select 3 it will connect at 2400

One other thing with Telco, if you have any problems with dialing put the phone number as ATDP3191009.

I also found with MENU 7.35 a quick way of turning it off. Just press Q.   ✺

Jenny's Younger Set

Here is a program from my old friend Vincent Maker. He also sent me a letter but I am as yet not able to get it off the cassette. It is a good idea when saving things on cassette to record it twice, one after the other, just in case, especially when sending it to someone else who is not very cluey like me. It does not take much longer as you do not need to rewind or find the end of the previous one.

From the Bulletin Board

```
MAIL TO : ALL
MAIL FROM : GOWFAR
```

Hi all!! A bit of update news for you from Gary Christensen (Qld) re: Geneve and other Myarc gear. It appears that Gary received a letter from Myarc (after numerous overseas calls not answered by them) stating that the Geneve's ordered by anyone here (through Gary) will be arriving in 4 to 6 weeks. Also, anyone having purchased a HFDCC through either Gary or (sorry) "whoever-it-is" in Sydney, who has not received them, the HFDCCs will be arriving shortly (1 week). Also soon to be released (1 to 2 weeks) in USA, are runtime Pascal, Myart and Advanced Basic Compiler (yes Compiler!!). So, for any updates you may need, either contact Gary Christensen in Qld or watch out for updates here. SCI-FI BBS will (at last) soon be on the Geneve!!

```
MAIL TO : ALL
MAIL FROM : LOU
```

I have some cartridges for sale, surplus to requirements. Addition and Subtraction 1+2, Alpiner, Disk Manager 2, Early Learning Fun, Early Reading, MiniMemory, Multiplication 1, Parsec, Reading Fun, Scholastic Spelling 4-5-6, Terminal Emulator, Extended BASIC, TI-Invaders.
For information phone Lou (02)626 8855

```
MAIL TO : ALL
MAIL FROM : SCI-FIBBS
```

Hi all. Pardon my ignorance in case this has been answered before but I have recently downloaded ARC111XB from the BBS only to find an anomaly in it. It will not pack files where the file being packed INTO (end result ARC file) is larger than (I think) 349 sectors. Now, having DS/DD drives, my BBS uses files which nearly fill the whole of both drives. Therefore, the problem was: how could I ARC my BBS (including all files current) as a backup? The answer was that I archived into 2 different files, each of which had less than the number of sectors at which space ARCIIIXB would not proceed. Then, I archived both of those already archived files into yet another ARC file. When wanting a single normal file, I have to un-archive the particular ARC file including the already archived file, un-archive that file and then extract the file wanted. Why do all this? The result of archiving twice gave me: A) My total BBS (remember it nearly fills 2 DS/DD drives) on one disk and reduced the total space considerably. B) By reducing as such, if i need to upload to another BBS, the file is shorter than the multiple files that went into the 2 files and the 1 file is shorter than the 2 files. It may be a round about way of doing it but then, when you are next on your favourite BBS and wish to upload multiple files, archived, longer than the bug in ARCIIIXB allows, you may thank me instead of watching your daily time limit go to nothing. Hope this helps someone. If anybody knows how to fix the bug, I would like to know, too.
Ta!! Regards , Greg.
--   ○

For Sale

We have had a rather difficult time here in the Leisure Coast these last few weeks (or months). My right hand man, Rolf, has had a real bout of illness with rather major surgery. This all happened just after he had done the layout but still had to stick in the headings and tidy up. We also lost communications in our regional group and were locked out of our meeting place the next evening but recovered from that by invading Phil's place at short notice. My car in the meantime was at the panel beaters for reconstructive surgery for two weeks and then Lou managed to drive his wife's car into the side of someone else's car which had no right to be there at that time. He had just left me for a few minutes to transport one of his children to somewhere while we were in the middle of laying out the PCB for the direct I/O interface. Then I have had some tight deadlines to meet at work and my car still has not emerged from the panel beaters after 3 weeks. I did manage to get the last TND out, with help from Sue, but it was a few days late as you probably noticed. I hope that this month's issue is back to normal, although I will be away for the last week of the school holidays, which is the week before the next meeting. This may mean a slight delay as someone else will have to do the envelope stuffing and posting. That means that I will not be at the next meeting and Lou will also be away so only Rolf will be able to attend.

****

Now for the newsletters. First there are those from Australia. Bug Bytes (Brisbane) August has an article from Col Christensen in which he talks about his Tapemaster program. It looks good and it is good to see Col back on deck after his heart attack. He also mentioned using my transistor circuit for the PIO cable to enable any printer to work with the TI RS232 card. Nice to have confirmation that it does help. Garry Christensen has an article on tips for the hard disk controller card with the TI99/4A and the Geneve. Mike Wright has an interesting article on using a system for file names and disk numbering to help keep track of all your files.

We have 3 from the Hunter Valley, June, July and August. They had a change in executive in June and also a change in cover. Bob Carmany has a useful article on how to write better programs. There is an interesting interview with the editor Brian Woods which I can relate to. Ron Pratt suggests that if you carefully make a plastic bag for your console you will keep it nice and clean. Bob Carmany also talks about Forth and programs that write programs. In July, Ron Klienschafer talks about the DSR in the Quest RAMdisk and a problem with a discharging battery in a Superspace cartridge. Take out the pullup resister on the WE line. Richard Terry continues with his Forth column, very interesting. There are: a beginner's BASIC article, a glossary of terms and some keyboard strips. August brings a challenge to assembly language programmers from Tony McGovern, more Forth from Richard Terry and also from Joe Wright and some musings from Bob Carmany.

From overseas comes the June issue of The Pug Peripheral of Pittsburgh. Anne Dhein writes about high resolution graphics and compares all the graphics programs, John Wilforth continues with disk drives and gives an adapter between 3 1/2" and 5 1/4" floppies, anote that Barry Boone has released V3.03 of Archiver and is working on a hard disk version and a review of Jim Peterson's program "homework helper".

July's TIC talk from the Rocky Mountain 99ers has the TI-Base tutorial from Martin Smoley and Tips from the Tigercub from Jim Peterson. In August there are more of the same plus a short tutorial on interrupts by Jim Ness. Note that the times we will get will be at 1/50 second rather than 1/60 seconds.

ROM in May from Orange County, California, has articles by Earl Raguse continuing with his program FILO, N. Armstrong shows a tinygram for two column printing and how to do it all, Adrian Robinson shows how to add boot tracking to your programs and Earl Raguse introduces the theory of dark. In July, Adrian Robinson looks at more assembly language to write a "RUN" statement to the end of an Extended BASIC loader program, more from Earl Raguse on FILO, some programs from Jim Swedlow and hardware corrections for the Horizon RAMdisk.

August TIdbits from Tennessee has reviews and news items from Beery Miller, Gary Cox, Bill Gaskill and Michael Dorman.

TopIcs of July from Los Angeles has a list of PEEKs and LOADs, lots of news from Bill Gaskill and beginning Forth from Earl Raguse. August has more news from Bill Gaskill, more Forth from Earl Raguse and a TI-Base quick reference chart.

****

As I sit here in front of my monitor wondering what I should rabbit on about this month, I often wonder if there is anyone out there who cares. I mean that it is very hard to know what you would like us to put into your newsletter if you do not tell us some how. I guess it goes with computer people, they do not like writing letters. I know I do not like writing letters. For some reason I do not mind typing in all this rubbish or trying to explain something in an article or two, but to write a personal letter requires me to get my mind into another gear. This always seems to be something to be done later rather than sooner. It would be nice if you all would overcome your inhibitions and bombard us with letters. It would make us think that someone really cares about what we do each month.

****

I received a phone call from someone in the Hunter Valley group about problems with a MiniPE RAMdisk. Unfortunately I have lost his name and so have not been able to send him a disk with a 32K version of the memory test program. I have not had time to look at many repairs this month although I have repaired two PE box cards. These were a RS232 card and a Myarc disk controller card which ceased working when the PE box was jammed back against a wall or something. This obviously caused these two cards to be misplaced in their sockets as the -18 volts had shorted to the CRUIN line on the board and blown two 9902 chips on the RS232 and a 74LS251 on the disk controller board. The CRUIN line is next to the -18 volt line on the PE box motherboard so be very careful with cards in the PE box.   o

loaded later, in comparison to a variable that you simply forgot to define. By placing the "extern" followed by the function's name with open and closed parentheses, you announce to the compiler that these are functions that will be loaded later. "extern" is the complementary function of "entry".

As with the last two months' articles, a lot of very subtle information has been crammed in. In order to gain a better insight it is necessary to experiment. For example, write a sub-program to find the middle number in a list or a set of sub-programs to make imperial to metric conversions. If you are having any trouble in programming 'c99' or using the compiler, do not hesitate to ask me at the monthly meetings, or send a letter to the TND.

NEXT MONTH
We will take our first look at 'char' variables and arrays. There will also be some more examples on sub-programs.
---
     o

This routine can easily be adapted for inclusion in a program where an alphabetical listing is desired. Parameters for allotting sufficient space in ORDER and STRGS would have to be derived from the number of records (actual or allowed). The same is true for any loop to read the pointers in ORDER and look up the corresponding records. Storing the contents of ORDER in the program (i.e., on disk) would eliminate the need for sorting until a new record is added to the file. Sorting takes time and unless =TEXT can be re-written in assembly language to speed it up one should remember to keep string length to a minimum. The longer the strings, the longer it takes to sort them.   o

Regional Group Reports

Banana Coast	8/10/89	Sawtell
Carlingford	18/10/89	Carlingford
Central Coast	14/10/89	Toukley
Glebe	12/10/89	Glebe
Illawarra	16/10/89	Keiraville
Liverpool	13/10/89	
Northern Suburbs	26/10/89	
Sutherland	20/10/89	Jannali

BANANA COAST Regional Group
(Coffs Harbour area)

Regular meetings are held in the Sawtell Tennis Club on the second Sunday of the month at 2 pm sharp. For information on meetings of the Banana Coast group, contact Kevin Cox at 7 Dewing Close, Bayldon, telephone (066)53 2649, or John Ryan of Mullaway via the BBS, user name SARA, or telephone (066)54 1451.

CARLINGFORD Regional Group.

Regular meetings are normally on the third Wednesday of each month at 7.30pm. Contact Chris Buttner, 79 Jenkins Rd, Carlingford, (02)871 7753, for more information.

CENTRAL COAST Regional Group.

Regular meetings are normally held on the second Saturday of each month, 6.30pm at the Toukley Tennis Club hall, Header St, Toukley. 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 St, Glebe. Contact Mike Slattery, (02)692 0559.

ILLAWARRA Regional Group.

Regular meetings are normally on the third Monday of each month, except January, at 7.30pm, Keiraville Public School, Gipps Rd, Keiraville, opposite the Keiraville shopping centre. 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)644 7377 (home) or (02)708 5916 (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. Group co-ordinator is Peter Young, (02) 528 8775. BBS Contact is Gary Wilson, user name VK2YGW on this BBS.

The regular monthly meetings at Sutherland Group always seem to produce the unexpected and the August meeting was no exception. Garry Wilson produced some backup disks for copying which refused to be initialised. The floppys turned out to be old MS-DOS disks but Garry was still left scratching his head, as virtually all attempts to solve the problem failed.

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 Woodstock Community Centre, Church street, Burwood. Regular items include news from the directors, the publications library, the shop, and demonstrations of monthly software.

At last month's meeting, Ross Mudie demonstrated how to transfer files and programs between computers using modems. Les Andrews showed a small monitor as well as holding the TI-Artist SIG meeting. Problems relating to the EPROM ROS in the RAMdisk was discussed and a tutorial on the use of TI-Writer's transliterate command was held. If you missed all that, do not worry, there is more to come in next couple of months.

October 7 - What have the various special interest groups achieved over the last six months? When do they meet? How can you become involved? All of these questions will be answered with presentations from each of the SIGs. The TI-Artist SIG will meet and the final tutorial in the transliterate series will be held.

November 4 - Full day workshop. Starts at 10am. A collection of tutorials, program workshops and demonstrations that will be of interest to any member, whether a programmer, a program user or a hardware hacker. Mark this date on your calendar.

December 2 - Christmas Party

Craig Sheehan (Meeting coordinator).   o

```
0 ( STRING SORT DEMO - 2 )
1
2 : SET   40 0 DO I 2 / ORDER I + ! 2
3              +LOOP ;
4
5 : ENTER$  SET
6           40 0 DO CR ." ENTER STRING : "
7                   I 2 / 2 .R ." : " PAD 2 EXPECT
8                   PAD STRGS I + 2 CMOVE 2
9              +LOOP ;
10
11 : SHOW$  40 0 DO CR ORDER I + @ DUP 12 .R SPACE
12              2 * STRGS + 2 TYPE 2
13              +LOOP ;
14 -->
15
```

The basic sort itself (now on screen 3) remains virtually unchanged with the exception of COMP (now $COMP) which has to furnish parameters for =TEXT. String length is purposely limited to 2 letters in the interest of speed. Unless it is absolutely imperative that, for example, the names RICHARD, RICHARDS, RICHARDSEN and RICHARDSON be sorted in proper order, the sort should be performed with a limited string length. This is because =TEXT is a high-level definition and is by no means a speed demon.

```
1 ( STRING SORT DEMO - 3 )
2 0 VARIABLE MIDDLE
3 : K@  2 * ORDER + @ ;
4 : K!  2 * ORDER + ! ;
5 : MID@  OVER - 2/ + K@ MIDDLE ! ;
6 : $COMP  K@ 2 * STRGS + 2 MIDDLE @ 2 * STRGS + =TEXT ;
7 : EXCH  2DUP K@ SWAP K@ ROT K! SWAP K! ;
8 : SORT  ( n1 n2 --- )
9       2DUP > IF DROP DROP
10           ELSE 2DUP 2DUP MID@
11        BEGIN SWAP BEGIN DUP $COMP 0< WHILE 1+ REPEAT
12            SWAP BEGIN DUP $COMP 0 > WHILE 1- REPEAT
13            2DUP > NOT IF 2DUP EXCH 1 -1 D+ THEN 2DUP >
14        UNTIL SWAP ROT 2OVER 2OVER - ROT ROT -
15            < IF 2SWAP THEN MYSELF MYSELF THEN ;
```

Usage is as follows:
   ENTER$ - allows input of twenty 2-letter strings
0 19 SORT - generates the alphabetical list in the form of pointers which are stored in ORDER
   SHOW$ - displays the result of the sort