50¢

VALLEY OF THE SUN TI 99 USERS GROUP

newsletter

VOL. 2 NOVEMBER 8, 1986 NO. 9

CONTENTS

VAST 99 BBS 437-4335

----------NOVEMBER, 1986----------

VAST 99 INFORMATION

The **VAST 99 USERS' GROUP** is a support group for TI 99 Home Computer users. We meet on the second Saturday of the month at the Los Olivos Resort Motel in the "Phoenix" room at 202 E. McDowell Road (about a block East of the Library). The meetings start at 10:00 AM and continue until 11:00 AM with socializing starting at 9:00 AM. The yearly membership fee is $6.00.

All meetings are open and anyone may attend. Only dues paying members may vote in elections and obtain programs from the Users' Group library.

The current officers are:

President
Gerry Kennedy.....992-7668
Vice-President
Doug Otten.....973-7768
Secretary
Mike Marfisi.....897-8280
Treasurer
Ike Van Kampen.....934-5164
User Group Librarian
Earl Bonneau.....269-3802
Newsletter Editor/BBS SysOp
Jim Ely.....437-1796

****

A FORTH Tutorial is being conducted by Rene' LeBlanc in this newsletter. It consists of a continuing series of articles relating to his version of FORTH which is available from the User Group Library. For more information, please contact him at (602) 991-1403.

The Users' Group's BBS is now in operation 24 hours a day. Contact it at (602) 437-4335. There is a lot of interesting conversation and information available here so give it a try.

Deadline for submission of articles or advertising for the Newsletter is the last Saturday of every month. Articles may be submitted in any form, however, the preferred method is by phone transfer directly to the Editor.

****

Advertising rates are as follows:

Commercial:

Full Page $10.00
Half Page $ 7.00
Quarter Page $4.00

Personal:

Four lines,
30 Characters/line
$1.00
$.20 per line
over four.

All rates are for **ONE** issue only!

****

Programs are available from the USERS' GROUP LIBRARY at the following rates:

SS/SD Disk $2.00
SS/DD Disk $4.00
DS/SD Disk $4.00
DS/DD Disk $8.00

If copying of documentation is required, it will be at the rate of $.10 per page. If the User Group supplies the disk, please add $1.00 to the above charges. An exchange program for free programs is also in effect. Please contact the librarian for further information.

****
* *
* VAlley of the Sun TI99 Users Group *
* *
****

----------NOVEMBER, 1986----------

From the Editors Desk

Gosh! The months are sure going fast. Next month is Christmas already. Let's get right to it!

I normally publish last month's meeting minutes on this page. There are no minutes for the October meeting, however, because of the SWAP MEET which took up most of the meeting and, by the way, was a great success. We will do that again some time. Now...

THE STORIES UNFOLD...

In this issue..... Page 9 has part 3 of Using T.I. Multiplan, this time taking a look at the "VALUE" function. Rene' explains how to enter an array in WHEREFORTHS OF FORTH on page 4. Computer Tutor takes a look at a couple of different functions (commands) of Extended Basic with some short sample programs, which, as usual, will be available on the BBS download section late this afternoon. The article starts on page 6. Gerry Kennedy has written a timely article on how to change a "FLOPPY" disk into a "FLIPPY" disk and this article is on page 8.

I hope to start in next month's newsletter a series of articles explaining T.I. Assembly language and to also have an article on using the Upload/Download section of the BBS. Stay tuned!

SOME OLD NEWS...

It was a real struggle last month to get the newsletter out and in the process, a few things got a little messed up. The screen that appeared as part of the Whereforths column was incorrect! In reformatting the article, the lines sort of got "squished" together and a number of characters in each line were dropped. If you got the newsletter by mail, you got the corrected version of the FORTH screen. Those of you who got the newsletter at the meeting and want the corrected screen are in luck! I felt the best way to do the correction was to make the whole page available. You can take the old page out of last month's and replace it with the new page and no one will ever know. Also available is a new page 3-4. On this one, the margin was on the wrong side and if you hole punched it, you lost some of the information. Again, those that got it by mail, got the correct copy. Those Halloween ghosts and gremlins struck a little early in October....

That's enough OLD NEWS. Now...

SOME NEW NEWS...

I got a note in the mail from Home Computing Journal (WHO??). It seems that the next issue of their quarterly (now about 2 months overdue) is moving along quite well (their gossip, not mine). No date as to when it will be mailed but, in the mean time, they have a lot of old back issues of HCM (remember them?) and software items they are trying to sell. Any takers?? Contact them directly.

Jim Peterson of Tigercub Software out of Columbus, OH, has announced that he is ceasing publication of his "Tips from the Tigercub" newsletter. It's hard to run a one man User Group and apparently some of the groups he has communicated with have not been exactly friendly. Let me assure Jim that none of the material he has sent to this user group in the way of software has made it to our library for distribution and some that has been uploaded to the BBS by other persons has been removed before

Continued on Page 10 ==>

----------NOVEMBER, 1986----------

WHEREFORTHS OF FORTH

In WHEREFORTHS #9 we ·efined the ACCEPT_AT and)ISPLAY_AT words and added a ,ype conversion word stoi 'string to integer). I am rying to gradually build up \ set of I/O primitives that ,ill form a nice base for a /ariety of application pro- ,ramming projects. You see, 'n Forth we usually start)ut by making a set of lan- guage extensions that will)ake our particular applica- ·ion program much easier.

In this issue we are going to plunge into some more advanced concepts of Forth to create a new "de- fining word". This means that we are going to extend the compiler itself so that we have the ability to de- fine arrays of strings stored under names of our choice for reference within \ program.

"What's that???" you say with fear in your voices!!! Well, perhaps an example would make this seem less threatening.

I want the ability to define an array of text strings, each with its own preceding count ("dimension- ed strings"), and I want to give the whole array a name so I can reference it with an index and its name. For example, I'd like to be able to define the array some- thing like:

"6 $ARRAY MYSTRINGS The first string is the zeroth string,String #1,String #2, Yet a third string,A forth string,And finally a 5th string,"

Given this defin- ition for MYSTRINGS, any part i c u l a r string could be ref- erenced as: "n MYSTRINGS" where n is the index. For example: "3 MYSTRINGS" would return the address of the string: "Yet a third string"

Another array with a different name could be declared:

"3 $ARRAY $meal Have breakfast ,Have lunch ,Have dinner ,"

Then, "0 $meal" would re- turn the address for "Have breakfast ", "1 $meal" would return the address for "Have lunch " and so forth.

Well how do we do this? We need to define the new "defining word" $ARRAY which will allow us to use it to define unique array names that reference user-speci- fied strings.

I went on to use screen #11 on my work disk and created the following words. Now we can follow through how they work. This screen gets us into some of the real power of Forth! (See Figure 1 below.)

I'll admit that we are getting into some "heavier" Forth words here, but if you dig in with me I think you will find that you can un- derstand what is happening. Furthermore, you will learn some things that will enable you to do some really power- ful things with the Forth language.

The first definition is for the word: !$ELEM (store string element). This is just a component word that I factored out of the $ARRAY word to keep its definition simpler.

"ASCII ," returns the ascii value for "," (comma). I have chosen this to be the delimiter for each string element. If you would rather use "/" or some other character as a delimiter, then substitute "ASCII /" or whatever other character you want as a delimiter.

WORD will parse the input stream using the specified character as a delimiter. I used "," so our strings could contain blanks. When this word is executed the input stream will either be from a disk buffer if the definition is being loaded from disk, or it will be the Terminal Input Buffer (TIB). It will collect the char- acters it finds in the input stream and move them to the top of the dictionary, be- ginning at HERE as a dimen- sioned string. (How handy; just what we wanted!)

However, WORD does not allot the space, so we have the code "HERE C@" to fetch the count of characters in the string, "1+" to add one character to accommodate the count character itself, then "ALLOT" which moves the Dic- tionary Pointer (DP) up to the next byte after the string found by WORD.

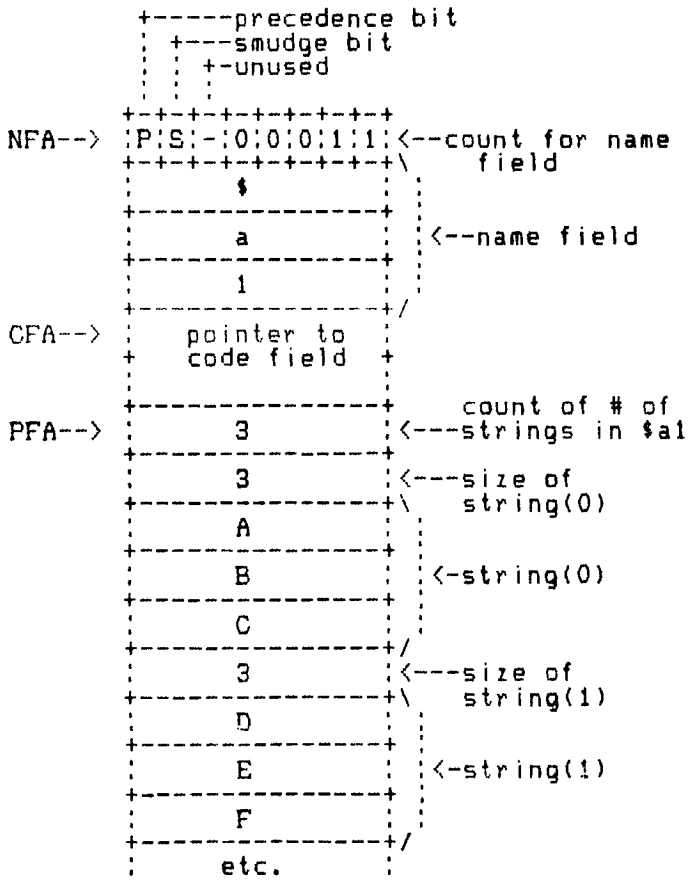
The next word defined is "<ER1>". I defined this word as a simple way to create an error message string that I need to use if someone uses an improper in- dex value as an index to an array defined by the $ARRAY

```
\ I/O experiments3
: !$ELEM ASCII , WORD HERE C@ 1+ ALLOT ;

: <ER1> ." Index too large" ;

: $ARRAY \ compile time ( n -- ) execution time ( n -- addr )
  <BUILDS DUP C, 0 DO !$ELEM LOOP DOES>
  DUP C@ ROT DUP >R >
  IF   1+ R> -DUP IF 0 DO COUNT + LOOP THEN
  ELSE R> DROP DROP [ ' <ER1> 2+ ] LITERAL THEN ;

\ Example Usage:
6 $ARRAY $A1 ABC,DEFG,HIJKL,MN OP QR,S T U V, 8 till 11 ,

: TST ( n -- ) $A1 COUNT TYPE ;
```

(Figure 1)

NOVEMBER, 1986

WHEREFORTHS OF FORTH CONTINUES



(Figure 2)

word. I'll explain this in more detail later.

Now we come to the main workhorse: the $ARRAY word. Remember, this word will be used to COMPILE other words that are defined by the user to create string arrays. This word uses the

"<BUILDS <name> <words1...> DOES> <words2...>"

construct where a new word called whatever you use for <name> will be created in the dictionary. When you are executing the <BUILDS word you are COMPILING the <name> word and will execute the <words1...> sequence. This is important! The <words1...> sequence is executed while COMPILING <name>.

Later, when you execute the word <name>, IT will execute the <words2...> sequence following the DOES>. These are called the "execution time words".

So, when we execute the word $ARRAY, we will be COMPILING an array definition with our choice for <name1> and executing the <words1...> sequence which happens to be:

DUP C, 0 DO
!$ELEM LOOP

Assume we are executing the definition: "3 $ARRAY $a1 ABC,DEF,HIJKL,"

In this case <name> = $a1, and it will be used to reference three strings: "ABC", "DEF" and "HIJKL".

When we execute $ARRAY, the <BUILDS word will create the header for $a1 in the dictionary. However, we still have the "3" on the stack (from "3 $ARRAY") that specifies how many elements we want to define for this array. The "DUP" will execute after the <BUILDS word and create another "3" on the stack. Then "C," (C-comma) will execute and allot one byte in the parameter field for $a1 and stuff the extra copy of "3" into it. We want to store this here so we can range check the index parameter that will be supplied to the $a1 word during ITS execution.

Next we have the literal "0" which will push on top of the remaining "3" that was on the stack. This "3 0" on the stack becomes the loop indices for a DO loop that will execute the !$ELEM word three times, once for each string that we must supply. The !$ELEM word will execute WORD and parse the input stream to find each of the strings and put them at HERE which will be moved up by the ALLOT word each time !$ELEM is executed. In this way, the three strings are stored as dimensioned strings in the dictionary. This concludes the <words1...> "compile time" execution of the word $ARRAY.

Each array (such as $a1) that is defined using this word will have its parameter field loaded first with a count of the number of strings in its array followed by each dimensioned string in the array. However, the CODE field pointer for each of these array words will point to the common "execution time" words in the DOES> portion of the $ARRAY defining word used to create the individual string array words.

Figure 2 at left shows the form of the dictionary entry for the $a1 string word.

When the $a1 word executes, the DOES> word portion of the defining $ARRAY word pointed to by the CFA of the $a1 word will be executed. Just keep on thinking about this until you understand it, and you will be greatly rewarded!

Execution of the $a1 word will begin with its PFA being placed on the stack by the DOES> word. Then we see the sequence:

"DUP C@ ROT DUP >R >"

DUP duplicates the PFA, C@ reads the contents of the PFA which contains a count of the number of strings in the array $a1. When $a1 is executed, the array index i is on the stack. The DOES> puts the PFA on top of it. DUP C@ puts the stored count of strings (#s) in $a1 on top of that so we have

Continued on PAGE 7 --=>

----------NOVEMBER, 1986----------

Hope you had more treats than tricks last month! We have three treats for you in this months Computer Tutor section...and the good news is that all three programs are of the Q&D variety. Q&D stands for quick and dirty which means you won't wear your fingers down to the knuckles keying in the program lines. This month we have a program which helps you find all possible key/ASCII combinations; a program that helps you understand the POS statement in basic and, my favorite, a Word Generator.

WORD GENERATOR: Browsing through PC Magazine last month I saw a review of a commercially available program that is used to generate new words. Many companies, like Proctor & Gamble, Colgate / Palmolive, etc., pay hundreds of thousands of dollars to persons to create new product names. The commercial program was priced at about $200. Now you can create the same results with some creative programming.

The concept is pretty simple. The alphabet is made up of vowels and consonants. There are typical alphabet patterns in many words. For instance, 5 letter words are frequently made up of two vowels and three consonants. My word generator program designates the first, third and fifth characters as consonants and the 2nd and 4th as vowels. We assign all the vowels to V$ and all consonants to C$. Random numbers will be used to pick the vowels from the strings. The program will generate a block of 36 words per screen page.

If you want to save these word gems, insert a program line that will also print the words to your line printer as well as the screen.

```
100 RANDOMIZE :: CALL CLEAR
:: CALL SCREEN(15)
110 V$="AEIOUY" :: C$="BCDFG
HJKLMNPQRSTVWXZ"
120 PRINT "WORD GENERATOR: b
y tom moran":RPT$("-",28): :
130 FOR I=1 TO 36
140 FOR J=1 TO 3 :: C(J)=INT
(20*RND+1):: NEXT J
150 FOR J=1 TO 2 :: V(J)=INT
(6*RND+1):: NEXT J
160 W$=SEG$(C$,C(1),1)&SEG$(
V$,V(1),1)&SEG$(C$,C(2),1)&S
EG$(V$,V(2),1)&SEG$(C$,C(3),
1)
170 PRINT W$,:: NEXT I
180 PRINT :: PRINT "PRESS AN
Y KEY FOR MORE WORDS"
190 CALL KEY(0,K,S):: IF S=0
 THEN X=RND :: GOTO 190 ELSE
 CALL CLEAR :: GOTO 120
```

POS STATEMENT: The POS statement in TI X-Basic is a handy little command. You can use it to find a string within a string or search for a special character within a string. The demo program below uses POS to find and separately define a first and last name. When asking someone to input their name they will, obviously, insert a space between their first and last names. So we use the POS statement to search for a space character (ASCII 32). When it finds a space we record the position in the string where it is located. Then, by using the SEG$ statement we create two strings from one. The full name is N$. The first name is put in FN$ and the last name in LN$.

```
100 CALL CLEAR :: CALL SCREE
N(4):: DISPLAY AT(12,1):"ENT
ER YOUR FIRST & LAST NAME>"
110 ACCEPT AT(13,3):N$ :: X=
POS(N$," ",1):: FN$=SEG$(N$,
```

COMPUTER TUTOR Continues.....

```
1,X):: LN$=SEG$(N$,X+1,LEN(N
$))
120 CALL CLEAR
130 PRINT "HI ";FN$;" ";
140 PRINT LN$;" IS SURE A FU
NNY";"LAST NAME"
150 STOP
```

KEYBOARD / ASCII READ: Practically all 255 ASCII characters are available to be included in your program from the keyboard. The only problem is that TI did not print a list to tell you how to access characters lower than 32 or greater than 126. This little program will record the character and the ASCII number when you press just about any key. I say just about because if you press FCTN+ you'll return to the title screen. For instance if you press FCTN V (that means the FCTN key and the V key simultaneously) you will see that it is ASCII character 127.

```
100 CALL CLEAR :: CALL SCREE
N(11):: DISPLAY AT(12,9):"PR
ESS ANY KEY"
110 ON BREAK NEXT
120 DISPLAY AT(1,1):"    KEY
BOARD/ASCII SEARCH" :: DISPL
AY AT(2,1):RPT$("-",28)
130 CALL KEY(0,K,S):: IF S=0
 THEN 130
140 IF K>128 OR K<32 THEN 15
0 ELSE 160
150 PRINT "Character:   - No
t Defined -" :: GOTO 170
160 PRINT "Character:   ";CH
R$(K)
170 PRINT "ASCII #:    ";K ;
: PRINT :: GOTO 120
```

Next month we'll have some Christmas gems for you. Until then....don't eat too much Turkey this month.

T.M

+++++++++++++++++

WHEREFORTHS..... Continued from PAGE 5

(i pfa #s) on the stack. ROT DUP leaves us with (pfa #s i i). Then >R pushes the top "i" onto the return stack for temporary storage. Then ">" tests if #s > i and leaves a boolean flag above the pfa value on the stack. If this is true, then the index "i" supplied is within the valid range.

The IF test removes the flag, leaving only the pfa on the stack. Assuming the index was valid we want to step the pfa to the first string. 1+ does this. Now we yank the saved value of i off the return stack with R>. -DUP conditionally duplicates it only if it is greater than 0. If you want to access the "zeroth" element we are already pointed to the right place so we want to skip the rest of the code and just return.

If i is not 0, then -DUP will make two copies and then the IF test will consume one of them. Then the 0 puts the initial loop index on top of the stack and the stack now contains (pfa+1 i 0). The (i 0) provide the loop indices for a DO loop to step to the desired element of the array of strings. "COUNT +" steps the address to the end of the current string, and this is repeated i times to finally leave the address of the selected string on the stack.

Finally we will look at the error case (the ELSE clause) where i was larger than the number of string elements in the array. If the ELSE clause is being executed, the index i is still on the return stack, so the R> drop gets it off and dumps it into the bit bucket. The next DROP eliminates the pfa+1 from the stack since we don't know what to do with it. Instead, we want to return the address of an error message string that I created with the <ER1> word up above.

The "[" word says "Lets stop compiling for a minute and just execute the following code." The "following code" is "' <ER1> 2+" which calculates the address of the error message string in the parameter field of the <ER1> word (well, yes, it is a bit "tricky"). This address is left on the stack and then the "]" word says, "Let's resume compiling again." "LITERAL" is a word that expects to find something on the stack and compile it into the dictionary. You see, it is this address that we want to compile into the dictionary so that when the ELSE clause executes it will encounter the literal and put it back on the stack, leaving it as the output for the $a1 word (or any other string array word compiled by $ARRAY) when it is given an invalid index.

The rest of the code on the screen is an example array and a test word to make it easy to try out this thing. Just execute "n TST" and it should print out the desired string element or the error message if n is too big.

I hope you have been able to dig through this session. It does cover some of the more interesting capabilities of Forth.

Rene' LeBlanc

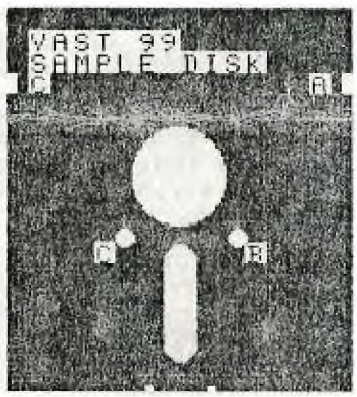
----------NOVEMBER, 1986----------

THIS AND THAT

DOUBLE YOUR DISK CAPACITY

This article may be old news to many of you, but I have encountered quite a few people lately who haven't discovered "flippy" disks. If you are still using the original equipment single sided disk drive that came with your expansion box, you have the option of modifying your disks so you can use both sides of them. In order to read the backside you must flip the disk over, thus the "flippy" name.

The only tool necessary for this project is a hole punch. It would be advisable to use one that has a hinged holder to catch the pieces as you punch the holes, otherwise they could fall into the disk jacket.



In the illustration, we have a disk that is modified to be a flippy. At point A is a square notch. This is the read/write protect notch. At point B is a hole with a matching one on the back side. If you turn the disk carefully in the jacket, you will see a small hole in the disk here. This is an index hole for the disk drive. To make use of the back side of your disk, you simply make holes on the left side of the jacket to match those on the right. Those additional holes are shown by points C in the illustration. I know, the read/write notch is square. The disk drive doesn't care if it's round or square. If you insist on having a square hole, I've seen tools advertised that will cut a square hole and they cost about 15 bucks. The round punch is about $2. Don't worry about making that hole too deep either. You would have to punch a hole almost a half inch deep to cut the disk inside (you wouldn't cut it that deep would you?).

The best way I have found to mark the disk for punching is to lay another disk upside down over the intended victim. If you turn the disk so the index hole is showing, you can mark the jacket of the bottom disk through it. Point of caution here! You only want a hole in the jacket, not the disk. You'll have to mark both sides of the jacket at the index hole and slip your hole punch into the jacket. I use a piece of label backing to protect the disk from scratches when I do this part. If you have a disk that's no good you can use the jacket for a template by removing the disk. It's a lot easier to mark the index hole this way. Otherwise you can only make a small mark on the jacket though the index hole in the upper disk.

One other thing I might mention here just so you can be forewarned. Theoretically, what you are doing is a no-no. When you turn the disk over, you're turning it backwards in the drive. The lining in the jacket is there to catch dust etc. and keep it off the disk. When you turn it backwards, you are putting dust on the disk. In actual practice, however, I've never heard of anyone having a problem. Many people do this, including software manufacturers. It isn't necessary to purchase double sided disks either. Although single sided disks are only guaranteed on one side, I've yet to see one that was bad on the back. If you should have a bad sector, it will be flaged when you initialize the disk if you verify the sectors and then when the computer writes to that disk, those flagged sectors are just skipped.

Gerry Kennedy

----------NOVEMBER, 1986----------

HINTS AND TIPS

USING T.I. MULTIPLAN (PART 3)

One of the main uses of the T.I. MultiPlan (TIMP) for a home computer owner would be to keep track of investments. Each person would want to set up his own spreadsheet, so below is an example.

The amount in column 4 would be automatically calculated by causing col 3 to be multiplied by col 2 for each row and the amount in col 4 designated as total would be kept up to date by having column 4 add itself and show the total at R8C4, which is the way a cell is designated.

In practice, a column could be inserted so that the Stock Exchange Symbols would be in column 1, cost and total cost columns could be inserted between cols 2 and 3 and even expected returns and yield columns are feasible. And in all cases the amounts would figure themselves out at each recalculation.

As an example of how these formulae are entered, to get the value in R8C4, the cursor would be placed over that cell: pressing "V" for "value" would command that a formula be entered and then the UP arrow would be used until the cursor is over the R3C4 cell. The add(+) is punched and the cursor immediately drops to the home cell (R8C4). Use the UP arrow again to the next lower amount, use + and again the cursor comes home. Repeat this for each value to be added except the last when ENTER is pressed completing the operation and the Total sought will enter itself in the proper (home) cell. Upon RECALC any changes in any of the figures will reflect in the TOTAL cell (R8C4 in this case). In using VALUE the formula can contain + - * / as needed. The method using the arrow keys works very well for a few lines, but if our list of stocks were longer we might just type out the formula for each line, thus:(assuming row 3)

R[-C2]*R[-C1] (Press ENTER)

This translates into: on the same row take the cell 2 columns to the left and multiply it by the cell one column to the left and enter the result in the home cell (R8C4).

A series of columns with an additional total column could be added to show the dividends received on these stocks or a separate file (XTERn) could be set up as apppended to the file. If the latter is done the external files are updated and entered on the main spreadsheet each time the TRANS - Load: mainfile (or other name) is loaded.

Notice that the difference in widths of the columns in the example, and the difference in the type of information in each of them is set using the FORmat command.

Next time we'll look at some of the other aspects of TIMP.

By Herbert Schlesinger

Sample Worksheet

	1	2	3	4
1	Name of Company	#shares	Pr.Val	Total Value
2				
3	Aetna Life	100	$45.25	$4525.00
4	Burroughs Corp	150	$55.25	$8287.50
5	Detroit Edison	100	$17.25	$1725.00
6				
7				
8	TOTAL			$14537.50

----------NOVEMBER, 1986----------

Editors Desk Continues

making it to the download section. Sorry to see the Tigercub Newsletter go, but Jim is going to continue in the software business, at least for a while, while taking on other projects. Good Luck, Jim.

MICROpendium announced in the last issue that you could get a FREE issue of their magazine (one copy only) by just sending them your address. Sounds like a good deal to me for a really good T.I. specific periodical. Their address is P.O. Box 1343, Round Rock, TX 78680.

Our BBS will pass the 1800 caller mark this weekend. I've made a few changes, I hope for the better, in the Upload and Download sections. They are very apparent in the download section in that now you can see when a file was uploaded and how many times a file has been successfully downloaded. One of the main reasons for this change is an effort on my part to write a "REMOTE SYSOP" program so that an assistant SysOp can maintain the board as well as the main SysOp who is running it. Bulletins can be changed and updated, uploads can be checked, deleted or moved to the download section and much more. Most of this program is finished and working just great, just ask Gerry Kennedy.

That's about enough of this rambling for now. I hope you all have a very nice Thanksgiving and I will see you all at the next meeting on Dec. 13, 1986.

Jim Ely,
Newsletter Editor/
BBS SysOp

NEWSLETTER
VAST 99 USERS' GROUP
c/o 4144 E. Nancy Ln.
Phoenix, AZ 85040

THIRD CLASS MAIL

TO:

EDMONTON TI99/4A COMPUTER
P.O.BOX 11983
EDMONTON, ALBERTA
CANADA T5J 3L1

THIRD CLASS MAIL