Ø94
8705

50¢

VALLEY OF THE SUN TI 99 USERS GROUP newsletter

VOL. 3 MAY 9, 1987 NO. 5

****

CONTENTS

VAST 99 BBS 437-4335

----------------------------------MAY,1987----------------------------------

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	
Mike Marfisi	897-8280
Vice-President	
Stu Olson	846-7624
Secretary	
Bob Nixon	838-4088
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
****

-----------------------------------MAY,1987-----------------------------------

From the Editors Desk

MINUTES for APRIL 11,1987

The April meeting of VAST 99 was held on Saturday, April 10, 1987, at the Olivos Hotel on East McDowell in Phoenix. Ike Van Kampen was unable to be present so Mike Marfisi acted as Treasurer. The meeting was called to order by President Gerry Kennedy at 10:00AM.

The first order of business was the election of officers for the 1987/88 year. During the March meeting the following members were nominated to hold office:

President	Mike Marfisi
Vice President	Stu Olson
Secretary	Bob Nixon
Treasurer	Ike Van Kampen

Gerry opened the floor for more nominees. There being none, the nominations were declared closed and the nominiated candidates declared elected. Next month rather than hold a formal meeting, we are going to hold a computer swap meet. Anyone having things to sell is welcome to bring them along. Gerry said that as far as he knows the hotel has no objection to the meet overflowing into the rear parking lot. He will check with management. Also we will arrange for some advertisement in local newspapers. It was pointed out that last years event brought in several new members.

Gerry pointed out that the Orlando, FL User Group has bought the entire stock of the Scott Adams adventure games. The module is selling for $3.00 and the disks are $4.00 each. Mike said that we had recieved notice from a user group in Oklahoma that has purchased the entire library of the National 99 User Group and is offering to sell programs for about $2.00 each. The address for the Scott Adams games is:

Greater Orlando User Group
P.O. Box 1391
Maitland, FL 32751

For the library info write:

C.O. 99ers Users Group
P.O. Box 30703
Midwest City, OK 73110

We have recieved information that KL Systems is marketing a keyboard interface that will allow a user to replace the TI keyboard with any IBM style keyboard. The interface is a printed circuit board that fits into the console just like the RAVE interface that Mike demo'd a few months ago. Other new items included an interface from Mack McCormick that will allow use of an RGB monitor with the TI. We have no real info on this. Gerry saw a blurb about it in MICROpendium.

Stu read an advertisement that described the new hard disk controller card to be marketed by Myarc. He said that it is floppy compatable and is selling for around $250. It will be available from TENEX.

Stu announced that the user group to which Gary Sweers now belongs has started a project building Super Carts for sale to individuals for a very reasonable price. Contact Stu for more info.

CONTINUED NEXT PAGE -=>

Editors Desk Continues

Gerry asked if any member would be interested in taking the club system home and using it every once in a while to keep it in peak condition. It seems that when it is not being used on a regular basis, it objects to being used at the meetings. There didn't seem to be anyone interested at this time.

At that point, the meeting was closed.

Mike Marfisi,
Secretary

* I/O ERROR

Let's start this month's issue with a correction to an article that appeared last month. I published Jim Peterson's BXB program from "Tips from Tigercub." There is an error in the second program. I copied the programs as they were published in "Tips" but there is a bug. The bug is in the line number that the second program creates. The line number created is 32000, but the line number that should be created is 30002. When you merge it as is, the program doesn't work. The change for the second program is in line 110 and here is the corrected line:

```
110 OPEN #1:"DSK1.BXBDATA",V
ARIABLE 163, OUTPUT :: PRINT
 #1:CHR$(117)&CHR$(50)&"][\[
]$"&CHR$(190)&CHR$(199)&CHR$
(136)&M$&CHR$(0)
```

The rest of both programs is correct. I had a chance to play with this program a little this past month (after correcting above) and here are a few comments...

When typing in and creating the programs, make sure there are no assembly routines in expansion memory. Start with a fresh (just turned on) system, otherwise the data created will be somewhat scrambled and the results less that impressive.

I merged this into 2 BASIC only programs to see what it would do. The programs where "CAMELOT" and "COUNTING FUN" (You are probably familiar with the first). It works "like a Million Bucks". The programs run perfectly now in EX-BASIC, only FASTER. If you haven't tried this program, do so. It really works well and you will be impressed! Now on to other things...

IN THIS ISSUE...

I'm so overwhelmed by the articles you folks have submitted that this month I have 2 articles on FORTH. I could have had 3 but one was submitted a little late. You'll get that one next month. It looks like you people only want articles on FORTH. It's OK with me if that is what you want. Page 5 is GOING FORTH by Bill Wedmore about creating a FORTH utility, M-Stack. Rene' begins his discussion of his DISK-COPY utility on page 6. Part 3 of our tutor on SORT Routines by Tom Moran is on page 8 and page 9 is part 4 of our tutor on TMS9900 Assembly Language. You only get out of it what you put into it!

A PROBLEM...

Our new President, Mike Marfisi, is going to be unable to continue as President. He has had a change in his work schedule

CONTINUED ON PAGE 12-=>

-----------------------------MAY,1987-----------------------------------

- - GOING FORTH - -

- - GOING FORTH - -

***** GOING FORTH *******

M-STACK, A_FORTH_Extension

If you have all been reading Rene's WHEREFORTHS, you will know by now that FORTH maintains two stacks, a PARAMETER stack with which it does most of its operations and a RETURN stack "R" which is used for DO loop housekeeping and auxiliary functions.

Frequently, we come upon operations where we want to roll strings of things into and out of a stack. For this, one needs two stacks. "R" can be used for this but if you are also inside a DO loop or two at the time, life can get very complicated. On the way to building my FORTHOPS system, I created a third stack "M" as an extension to support these operations. I think you will find M very handy as an auxiliary stack in your FORTH system.

The screen below defines a series of words creating stack M and the following is a line-by-line description of what is going on:

Line 2 places the M stack at low memory location 3D00. This is the same area occupied by FORTH's Return stack. They are over 300 words apart and growing in opposite directions. They cannot collide unless their sum exceeds that value, which is not likely in normal usage.

Line 3 initializes MPTR to point to the base location of M.

Line 5-7 defines PUSH, which moves the top of the parameter stack onto M, updating MPTR.

Line 8-13 defines POP which moves the top of M onto the parameter stack and decrements MPTR.

Line 14 defines M@ which gets a copy of the top of M onto the parameter stack. It does not disturb M.

One of the advantages of having M in the low memory space is that it is outside of FORTH's dictionary space and numbers placed on M will not be disturbed if the dictionary contents are changed. Thus M can be used to pass parameters and values between FORTH applications that are spooled in as binary load modules from disk.

Next month I will give an example or two of the use of M to do some fun things in TI-FORTH.

Bill Wedmore

```
SCR 247
0  ( Definition of M stack)
1  BASE->R HEX
2  3D00 CONSTANT M-STACK
3  M-STACK VARIABLE MPTR
4   ( Set up pointer to stack location)
5  : PUSH ( n -- )
6      MPTR @ 2+ SWAP OVER MPTR ! ;
7   ( Push n onto M-STACK, update MPTR)
8  : POP  ( -- n )
9      MPTR DUP M-STACK 2+ <
10     IF ." STACK EMPTY" M-STACK MPTR !
11     ELSE @ DUP @ SWAP 2- MPTR !
12     THEN ;
13  ( Pop n to parameter stack )
14 : M@ ( -- N ) MPTR @ @ ;
15  ( Gets a copy of M )   R->BASE
```

MAY,1987

WHEREFORTHS OF FORTH

In WHEREFORTHS #14 I provided a nine screen Forth program designed to do a sector-by-sector copy of disks using only a single disk drive. I intend this program to be a reasonable example of a real and useful program written in Forth, and at the same time, I have tried to show how previous utility words we have developed in the WHEREFORTHS articles can be used as components in an application program.

How does one get started to write a program in Forth? In many respects, it is no different than getting started writing a program in any other computer language. We must decide what we want the program to do (WHAT?), decide or a method by which the program can do it (HOW?), and then make a number of implementation choices of how to encode that method into an operational program (DETAILS).

Let's review these steps for the COPY-DISK program:

WHAT?:

Copy any formatted TI diskette to another diskette on a sector-by-sector basis. This is different than simply copying all the files from somewhere on the source disk to somewhere on the destination disk. Instead, this means "cloning" the source disk so that not only does the destination disk contain the same information as the source disk; but doing it so the information is located exactly the same on the copy disk as it is on the source disk.

An additional requirement is that only a single disk drive is needed to do this copy function. I also require that it work with all valid formats (SSSD, SSDD, DSSD DSDD) for TI disks.

HOW?:

The main strategy is to read as much as possible from the source disk into memory, then prompt the user to change diskettes and then write the stored data back onto the destination disk and prompt the user to change back to the source disk and keep repeating this until the entire disk has been copied.

The first subproblem is to figure out how much spare memory is available for storing the intermediate disk data being copied, and where the spare memory is located. This will be the subject of the rest of this article.

There are three main areas of RAM (Random Access Memory) available to store the data from the source disk before prompting the user to change disks and writing it back out to the destination disk. The TI memory address space is partitioned into "low RAM" from addresses 8192 to 16767; "high RAM" from 40960 to 65535; and VDP RAM. In Whereforths #4 I provided a memory map of the RAM address space. Also this and the VDP RAM space is summarized in your TI Forth manual.

Of course, not all of this address space is available to store disk data because the Forth operating system and the disk copy program itself occupy the RAM space, and the VDP RAM is used for display of the screen data, graphic characters for screen display of ASCII characters, disk buffers and other things. I had to dig around a little bit to determine what memory is available within each of these address spaces to store the disk data.

It turns out that in low RAM the available space consists of the five Forth system disk buffers. The address of the first disk buffer is stored in the Forth user variable FIRST and the first address above the Forth disk buffers is stored in the Forth user variable LIMIT.

In high RAM the space consists roughly of the space between the end of the scratch pad buffer PAD which floats above the top of the Forth dictionary and the parameter stack whose base is pointed to by the Forth user variable S0. The parameter stack starts at high memory and grows downward toward low memory.

The available VDP RAM can be determined from Page 3 of Chapter 4 of your TI Forth manual as the space between >1400 (hex 1400) and the highest available address of VDP RAM (This is stored in the CPU RAM address >8370 (See Chapter 4, Page 5 of your TI Forth manual)).

By knowing these things, we can compute the available space within each of these three regions of memory. Most of these calculations are made on screen #4. Then a final calculation is made at the end of screen #9.

WHEREFORTHS OF FORTH CONTINUES...

```
\ Disk Copy Program SCR#4
 BASE->R DECIMAL
 0 VARIABLE #B   \ Number of Blocks
 0 VARIABLE BP   \ Block Pointer
 S0 @ PAD 40 + -   B/BUF /  CONSTANT #HB \ # Hi  Bufs
 LIMIT FIRST -     B/BUF /  CONSTANT #LB \ # Low Bufs
 HEX 8370 @ 1400 - B/BUF /  CONSTANT #VB \ # VDP Bufs
 1400 CONSTANT VDPBUF \ Address of first VDP Buffer

 R->BASE

  -->
```

```
\ Disk Copy Program SCR#9
 : COPY-DISK TEXT DR0 .SD GET_#B 0 B!
   BEGIN B@ #B@ <
   WHILE CR .RB B@ >R
         GET-LO-BUFS GET-HI-BUFS GET-VDP-BUFS .TD
         CR .WB R> B!
         PUT-LO-BUFS PUT-HI-BUFS PUT-VDP-BUFS .SD
   REPEAT
   CR .COMPL ;

 S0 @ PAD 40 + - B/BUF / ' #HB ! \  Set #HB to final value
```

Now, what did we do here? On screen #4, I first set the BASE to DECIMAL (just to be sure) then defined the variables #B (Number of Blocks) and BP (Block pointer). Next, I defined three constants: #HB (Number of hi blocks), #LB (Number of low blocks) and #VB (Number of VDP blocks). Remember, Forth partitions disks into blocks of 1K each, so we are setting up how many 1K blocks we can put in each of the three memory areas.

The first constant definition for #HB is temporarily calculated as

S0 PAD 40 + - B/BUF /

Since PAD will move up as the dictionary grows, I recalculate this on screen #9 where the dictionary has grown to the maximum after loading the program. I put it on screen #4 because it will be referenced by subsequent screens. We will just overwrite it with a new value on screen #9. S0 @ returns the address of the stack base (the effective high address in memory), then PAD returns the beginning of the scratchpad buffer. I add 40 to it, because during output of text to the screen, this buffer will be used. I chose 40 because no user messages are longer than this. So PAD 40 + computes 40+PAD which is the low byte address of this range. Then the - sign subtracts PAD+40 from S0 yielding the number of bytes of available space. Actually, there is a slight risk here because I didn't allow for any values on the stack. It is unlikely that more than 3 or 4 values would be used, and I probably should have used a value of S0 8 - to be safer. You can add this refinement if you wish.

B/Buf is a system constant set at 1024 (bytes per buffer). We divide by this to see how many buffers will fit into that range of addresses. The forth word "/" discards any remainder which is what we want. We only want the integer number of buffers that will fit.

For #LB the calculation

LIMIT FIRST - B/BUF /

performs a similar calculation.

Then HEX 8370 @ returns the highest VDP RAM address available, and 1400 - gives us the number of VDP RAM bytes available. B/BUF again gives us the number of blocks that will fit into that space and this defines the value for the constant #VB.

After the program is done loading and PAD will return its final value, we want to patch the proper value into the constant #HB. On screen #9, we recalculate the num-

CONTINUED PAGE 9 ==>

-----------------------------------MAY,1987-----------------------------------

SORTING_OUT_THE_SORTS

PART III

SORTING POINTERS

This is part 3 of our three part tutor series on sorting algorithms in basic. In part 1 & 2 we examined sort routines that organize your data whether it be numbers or strings. Now let's see how to keep everything together when you have multiple fields.

PROBLEM: We have written a phone-book type database that includes first name, last name, address, city, state, zip-code and phone number. Each record has a separate designated array. We want to sort the database in alphabetical order by the last name. How do we make sure that all the accompanying data is sorted properly.

SOLUTION: This is accomplished by using pointers which keep track of where all that information is located. When sorting in RAM (vs. the slooow disk-sort) here is the easiest way to accomplish the task.

To keep this routine as comprehensible as possible we will use a simple select sort routine. While this is the least efficient, it is easiest to follow.

```
10 REM - First we need to cr
eate a sample data base
20 DATA JONES,SMITH,ELY,COOP
ER,KENNEDY,BROWN
30 DATA JACK,SAM,ED,CARL,KOR
Y,BILL
40 FOR I=1 TO 6 :: READ LAST
$(I):: NEXT I
50 FOR I=1 TO 6 :: READ FIRS
T$(I):: NEXT I
60 CALL CLEAR :: PRINT "UNSO
RTED LIST:"
70 FOR I=1 TO 6 :: PRINT FIR
ST$(I);" ";LAST$(I):: NEXT I
80 REM - Now let's sort thes
e names
100 FOR I=1 TO (6-1)
110 IF LAST$(I+1)>=LAST$(I)T
HEN 200
120 B$=LAST$(I+1)
130 A$=FIRST$(I+1)
140 LAST$(I+1)=LAST$(I)
150 FIRST$(I+1)=FIRST$(I)
160 LAST$(I)=B$
170 FIRST$(I)=A$
180 GOTO 100
200 NEXT I
210 PRINT :: PRINT "SORTED L
IST:"
220 FOR I=1 TO 6 :: PRINT FI
RST$(I);" ";LAST$(I):: NEXT
I
```

Line 110 is the actual sorting algorithm. This line compares all of the last names. If a shuffle of the names is needed then the program continues on to line 120, else if proceeds to the next loop.

Lines 120-180 is the exchange routine. A$ and B$ are variables to hold the string being replaced so we don't lose it during the exchange. B$ holds the last name and A$ holds the first name. If you had other fields to sort, like adress, zip-code, etc., you would need to create separate variables to hold the exchange data as well. When the exchange is completed the program returns to line 100 to make more comparisons.

Now if you wanted to sort this same list by first name, you would simply change the variable in line 110 from LAST$ to FIRST$.

This is a permanent sort. That means that your original list is changed forever. There may be times when you want to leave the original list intact while sorting by another field. This is a bit more complicated.

COMPUTER TUTOR CONTINUES....

Here you would use pointers. For instance if you were sorting this list by last name you would amend the LAST$ variable with its record (or position) number. Therefore the first record, LAST$(1) would now change from JONES to JONES1. The way you would do this is to include a line of code that says:

```
    FOR I=1 TO 6 :: LAST$(I)=LAS
    T$(I)&STR$(I):: NEXT I
```

This appends the position number onto the end of the last name string. Now you always know where that string belongs. Such as JONES always should be kept in the first position, or better stated, is the first record. That is known as a pointer. That number appended to the string points to the position of the data.

Pointers are best used with hugh data files and disk sorts. Your RAM sorts work best with the exchange method.

I hope this series of sorting information has been helpful to you. If you have any questions, please see me at a UG meeting or drop a line to the VAST UG Newsletter editor, Jim Ely.

T.M.

)O()O()O()O()O(

WHEREFORTHS.... CONTINUED FROM PAGE 7

ber of high buffers and then ' #HB returns the parameter field address of the constant #HB. We then store the new value into that address and all previous references to it will return the new value at run time.

The number of blocks that can be stored in each of the three buffer areas have now been calculated and assigned to the constants #HB, #LB and #VB. Note that to optimize the amount of space in high RAM you should strip the starting Forth configuration down to just the kernal plus just those extensions needed to support the COPY-DISK program.

Next month we'll discuss other aspects of the COPY-DISK program.

Rene' LeBlanc

HINTS AND TIPS

TMS9900 ASSEMBLY LANGUAGE TUTORIAL

Part IV

THE_BEAUTY_OF_BASIC_(CONTINUED)

by Steve Royce - WNY 99'ERS

Last month, I went on at great length about the value of creating your own subroutines for use in your assembly programs. This month, I present two routines: one rather simple (CLEAR) and one not so simple (SPRITE). I won't go into any great explanation, but will let the routines do the work. Type them in yourself as you will see how the structure works if you do the work.

ASSEMBLY LISTING STARTS ON NEXT PAGE -=>

HINTS AND TIPS CONTINUES....

```
       DEF TEST
       REF VSBW,VMBW,VWTR
TEST   BL @CLEAR
       LI R2,>0100
       MOVB R1,@>837A
       BL @SPRITE
       DATA 0,95,9,1,5,5,7,33
       LIMI 2
       JMP $
       TITL '* CLEAR SUBROUTINE  V 2.0  9-4-84  SJR *'
       PAGE
*
* SUB CLEAR  COMMON WORKSPACE, USES R0,R1,R11
*
CLEAR  LI R0,767             LOWER RIGHT SCREEN LOCATION
       LI R1,>2000           BLANK CHAR (ASCII 32)
       BLWP @VSBW            WRITE
       DEC R0                NEXT SCREEN LOC
       JLT $+4               GOTO RT IF R0 IS NEG
       JMP $-8               BACK TO BLWP
       RT
*
*
       TITL '* SPRITE SUBROUTINE V 2.0  9-16-84  SJR *'
       PAGE
*      SUB SPRITE
*      USES R0,R1,R2, R11
*      REF VSBW, VMBW REQUIRED
*      START WITH SPRITE 0 WHEN DEFINING
*      PUT # OF SPRITES TO MOVE IN >837A
*      CLR @SPRITE-6 UPON REDO OF PROGRAM
       BSS 2                 **INITIALIZE FLAG
       DATA >D000,>0000      Y,X,CHAR,COLOR INIT DATA
SPRITE MOV @SPRITE-6,R0      MOV FROM BSS 2, ABOVE
       CI R0,>0000           CHECK IF INITIALIZED YET
       JNE $+64              IF YES, JMP TO MAIN
*
*  INITIALIZE TABLES
*
       LI R0,>0300           SPRITE ATT LIST
       LI R1,SPRITE-4        DATA TO LOAD
       LI R2,4               4 BYTES
       BLWP @VMBW            WRITE TO VDP
       INCT R0               BUMP POINTER
       INCT R0               BUMP POINTER
       CI R0,>0380           AT END OF TABLE?
       JNE $-12
*
       LI R0,>0780           MOTION TABLE
       LI R1,SPRITE-2        ZERO DATA
       LI R2,2               2 BYTES
       BLWP @VMBW            VDP WRITE
       INCT R0               BUMP POINTER
       CI R0,>0800           AT END OF TABLE?
       JNE $-10
*
       LI R0,>0601           WRITE TO VDP R6
       BLWP @VWTR            TO START PAT LIST AT >0800
*
*  SET FLAG
*
       SETO @SPRITE-6
*
*      MAIN SPRITE SUBROUTINE
       LI R2,4
       MOV *11+,R0           GET SPRITE #
```

CONTINUED NEXT PAGE -=>

HINTS AND TIPS CONTINUES....

```
        CI R0,>2000
        JL $+4
        MOV *R0,R0
        CI R0,33
        JNE $+4
        RT
        SLA R0,2              MPY BY 4
        AI R0,>0300           ADJUST TO S.A.L.
        MOV *11+,R1           GET DATA
        CI R1,>2000
        JL $+4
        MOV *R1,R1
        SWPB R1               PUT IN MSB
        BLWP @VSBW            VDP WRITE
        INC R0                BUMP POINTER
        DEC R2                DECREASE COUNTER
        JNE $-20              BACK TO GET NEXT DATA
*
        AI R0,>047C           ADJUST TO MOTION TABLE
        MOV *11+,R1           GET Y MOTION
        CI R1,>2000
        JL $+4
        MOV *R1,R1
        SWPB R1               PUT IN MSB
        BLWP @VSBW
        INC R0                BUMP POINTER
        MOV *11+,R1           GET X MOTION
        CI R1,>2000
        JL $+4
        MOV *R1,R1
        SWPB R1               PUT IN MSB
        BLWP @VSBW
        JMP $-88
*
        PAGE
        END
```

Assemble this routine and RUN it. You should get a silly stick wriggling down your screen from upper left to lower right. That's character 95. Too bad we don't have a routine to define characters, isn't it????

THINK ABOUT THAT ONE!

Editors Desk Continues

and will most likely miss most of our meetings. We need a person to lead our group, or do we just disband because nobody is interested? Again, THE CHOICE IS YOURS!!

By the way, in reading through the Newsletters from the other groups around the country, this seems to be a very common problem. Everyone wants to be a member of the group, but nobody wants to lead. Too bad we can't find this fella NOBODY!

IN OTHER NEWS...

There isn't much. Suppliers of the new Myarc 9640 Computer are beginning to receive their orders and shipments to consumers are slow, but going well. Bet you thought you would never see the day, HUH? Me too.

OUR BBS...

Has received over 4500 calls and is going well. It's been a slower month than last month as Brenda hasn't been on. Come to think of it, neither has Don Shores. I wonder..... They (he?) must have lost the phone number to the BBS. Hope you like the recent changes.

That's it. See you next month.

Jim Ely,
Newsletter Editor,
BBS SysOp

NEWSLETTER
VAST 99 USERS' GROUP
c/o 1425 E. Del Rio Dr.
Tempe, AZ 85282

FIRST CLASS MAIL

TO:

FIRST CLASS MAIL
May, 1987