VOLUME 2 ISSUE 5 $1.50 JANUARY 1986

SUPER 99 MONTHLY

The third installment of Warren Agee's FORTH tutorials will appear in next month's issue.

SEE YOU AT 99' FEST WEST '86!

EXTENDED BASIC

Databases: Some New Directions For 99/4A Programmers

STANDARD: 1A 2XB TW 4B 5A 6B 7B 9B 10B (o)

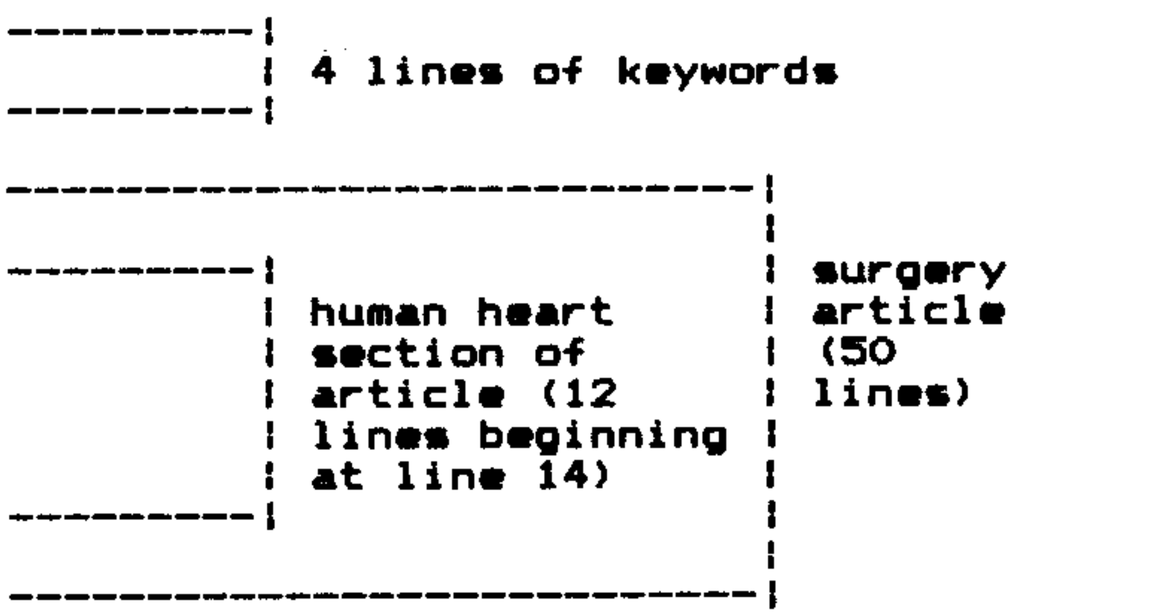
While many so-called database programs have been written for the 99/4A, the overwhelming majority of such programs have taken on such a narrow scope as to best be termed "mail lists". Though a mail list is a form of database, to constrain ourselves forever to only those functions akin to mail lists is rather ludicrous in light of the power of our 99/4A. There are (at least) two database possibilities that are generally not available through a mail list. The first is a strong system of cross-referencing of data, with databases encompassing such a system often being referred to as "relational databases". The second possibility is access to information of varying lengths (the mail list always formats a specific number of lines, usually about 3 to 8 lines, into one record). This article will address the second possibility, that of accessing information of varying lengths. This article assumes at least some knowledge of TI Extended BASIC, but will be directed to the database novice and will therefore avoid coverage of the technical aspects of databases. We will attempt to avoid confusion by not using advanced terminology such as KSAM, ISAM, Image, etc.

Accessing information of varying lengths can be useful in a number of situations, though it is certainly not always necessary. One example of a situation that would dictate use of variable length data would be if a surgeon would have a collection of articles that needed to be accessible at various points in the articles. For instance, if one article was on modern surgical techniques and the surgeon was interested in only a portion of the article that pertained to the human heart, it would be quite convenient for him to be able to simply key in "human heart" and receive a printout of only the section of the article that pertained to the heart, regardless of whether the passage was one line or many lines long.

-->

So, let's take a look at how a database program could be written to meet the requirements of the surgeon mentioned above. We'll assume that the articles that the doctor has are in typical 99/4A text format and can be loaded into TI-Writer.

As mentioned above, our objective is to access information based upon certain words, known to database programmers as "keywords". Obviously, the keywords must somehow be linked to a corresponding text passage. This is done with a pointer. To keep matters as simple as possible, we will set up our keywords at the beginning of our TI-Writer file, along with the pointers for those keywords. Our keyword list will be placed at the beginning of the file, so that the file might have the following structure:



To further simplify matters, we will assume that keywords can be up to 14 characters long and must occupy 14 character positions, with the beginning pointer occupying 3 character positions and the ending pointer also occupying 3 character positions. We will use the line occupied in TI-Writer (after inserting 4 lines for keywords) as the pointer index. Here is how the keyword list for the example above can be set up:

```
MODERN SURGERY005054HUMAN HEART   019030
```

Once the keywords are entered, the file should be saved in Fixed format, so that Extended BASIC can access the file as Relative records. To save the file, use PF (Print File), then F DSKx.FILENAME, where "F" indicates a Display Fixed 80 file format and "x" designates the disk drive number. Once the file has been saved in this manner, an Extended BASIC program can be written to access the information via the keywords.

Here is an Extended BASIC program listing that will access the information based on the file structure shown above:

```
30 ! DATABASE TEXT READER
     COPYRIGHT 1986
     BYTEMASTER COMPUTER
        SERVICES
40 OPTION BASE 1
50 DIM L$(21)
60 DIM F$(127)
100 GOSUB 1000 !
    CATALOG DRIVE
110 GOSUB 2000 !
    SELECT FILE
120 GOSUB 3000 !
    SELECT FROM KEYWORDS
130 GOSUB 4000 !
    SELECT OUTPUT
140 GOTO 5000 !
   AGAIN?
```

-->

```
999 END
1000 DISPLAY AT(1,1)ERASE AL
L:"DATABASE TEXT READER":"CO
PYRIGHT 1986":"BYTEMASTER CO
MPUTER SERVICES"
1010 DISPLAY AT(8,1):"SELECT
 DATA FILE":"DISK DRIVE (1-5
)  1"
1020 ACCEPT AT(9,19)BEEP VAL
IDATE("12345")SIZE(-1):D$
1030 OPEN #1:"DSK"&D$&".",IN
PUT ,RELATIVE,INTERNAL
1040 I=1 :: ON ERROR 1900
1050 INPUT #1:A$,X,Y,Z
1060 INPUT #1:A$,X,Y,Z
1070 IF X=1 AND Z=80 THEN F$
(I)=A$ :: I=I+1
1080 IF A$<>"" THEN 1060
1090 I=I-1 :: RETURN
1900 RETURN 1090
2000 DISPLAY AT(1,1)ERASE AL
L:"SELECT A FILE:" :: ON ERR
OR 2040
2010 IMAGE ### ##########
2020 J=19
2030 J=J-19
2040 X=0
2050 J=J+1 :: X=X+1
2060 IF F$(J)<>"" THEN DISPL
AY AT(X+1,1):USING 2010:J,F$
(J)
2070 IF X>18 OR F$(J)="" THE
N DISPLAY AT(21,1):"ENTER SE
LECTION (#,N=NEXT,  P=PREVIO
US,E=END):":"1" ELSE 2050
2080 ACCEPT AT(23,1)BEEP VAL
IDATE(DIGIT,"NnPpE@")SIZE(-3
):S$
2090 IF POS("Nn",S$,1)<>0 TH
EN 2040
2100 IF POS("Pp",S$,1)<>0 AN
D J>18 THEN 2030
2110 IF POS("E@",S$,1)<>0 TH
EN 999
2120 ON ERROR 2900
2130 IF VAL(S$)>I THEN 2080
2140 S=VAL(S$)
2150 ON ERROR 2800
2160 CLOSE #1
2170 RETURN
2800 RETURN 2170
2900 RETURN 2080
3000 DIM K$(16),B(16),E(16)
3010 ON ERROR 3900 :: OPEN #
2:"DSK"&D$&"."&F$(S),RELATIV
E,FIXED
3030 FOR J=1 TO 4
3040 LINPUT #2:A$
3050 FOR K=1 TO 4
3060 C=J*4-3+K-1 :: K$(C)=SE
G$(A$,K*20-19,14):: IF K$(C)
=RPT$(" ",14)THEN 3080
3070 B(C)=VAL(SEG$(A$,K*20-5
,3)):: E(C)=VAL(SEG$(A$,K*20
-2,3))
3080 NEXT K
3090 NEXT J
3100 DISPLAY AT(1,1)ERASE AL
L:"SELECT A KEYWORD:":"":"#
 KEYWORD            RECORDS"
3110 IMAGE ## ##############
   ###
3120 IMAGE ##   ############
## ###
3130 FOR J=1 TO 16 :: Z=E(J)
-B(J)+1 :: Z$=STR$(Z):: IF B
(J)=0 THEN Z$=""
3140 IF (J+3)/4=INT((J+3)/4)
THEN DISPLAY AT(J+4,1):USING
 3110:J,K$(J),Z$ ELSE DISPLA
Y AT(J+4,1):USING 3120:J,K$(
J),Z$
3150 NEXT J
3160 DISPLAY AT(22,1):"SELEC
TION: 1"
3170 ACCEPT AT(22,12)BEEP VA
LIDATE(DIGIT)SIZE(-2):C
3180 IF C>16 OR K$(C)=RPT$("
 ",14)THEN 3160
3190 CLOSE #2 :: RETURN
3900 DISPLAY AT(20,1):"FILE
IS NOT A PROPERLY FORMATTED
DATABASE FILE" :: F$(S)="" :
: RETURN 110
4000 DISPLAY AT(1,1)ERASE AL
L:"OUTPUT TO:":"":"1. SCREEN
 (WRAP TEXT)":"2. SCREEN (28
 COLUMNS)":"3. PRINTER, DISK
, ETC.":"":"SELECTION: 1"
4010 ACCEPT AT(7,12)BEEP VAL
IDATE("123")SIZE(-1):Z
4020 IF Z=3 THEN DISPLAY AT(
12,1):"OUTPUT DEVICE:":"PIO"
 :: ACCEPT AT(13,1)BEEP SIZE
(-28):P$ :: OPEN #3:P$ :: Y=
0
4030 CALL CLEAR :: DISPLAY A
T(23,1):"ONE MOMENT..." :: O
PEN #2:"DSK"&D$&"."&F$(S),RE
LATIVE,FIXED
4040 FOR J=B(C)TO E(C)
4050 LINPUT #2,REC J-1:A$
4060 ON Z GOSUB 4400,4600,48
00
4070 NEXT J
4080 IF Z<3 THEN GOSUB 4950
4090 ON ERROR 4900 :: CLOSE
#2 :: CLOSE #3
4100 GOSUB 4500 :: RETURN
4400 ON ERROR 4450
4410 IF Y=0 THEN Y=-2
4420 Y=Y+3 :: L$(Y)=SEG$(A$,
1,28):: L$(Y+1)=SEG$(A$,29,2
8):: L$(Y+2)=SEG$(A$,59,28)
4430 IF Y=19 THEN GOSUB 4950
 :: Y=0
4440 RETURN
4450 RETURN 4420
4500 PRINT "":"PRESS ANY KEY
 TO CONTINUE"
```

-->

```
4510 CALL KEY(5,KY,ST):: IF
ST<1 THEN 4510
4520 CALL CLEAR :: RETURN
4600 B$=SEG$(A$,1,28):: X=0
:: H=0 :: G=0
4610 B$=SEG$(A$,1,28):: X=0
:: H=0 :: G=0
4620 IF POS(B$," ",1)=0 THEN
 4640
4630 IF SEG$(B$,LEN(B$),1)<>
" " THEN C$=SEG$(B$,LEN(B$),
1)&C$ :: B$=SEG$(B$,1,LEN(B$
)-1):: X=X+1 :: IF LEN(B$)>1
 THEN 4620
4640 IF B$=RPT$(" ",LEN(B$))
THEN 4660 ELSE Y=Y+1 :: L$(Y
)=B$ :: IF Y=21 THEN GOSUB 4
950 :: Y=0
4650 G=G+1 :: H=H+LEN(C$)::
B$=SEG$(A$,G*28-H+1,28-LEN(C
$)):: C$="" :: IF LEN(B$)>1
THEN 4620
4660 RETURN
4700 RETURN 4660
4800 PRINT #3:A$
4810 RETURN
4900 RETURN 4100
4950 CALL CLEAR :: FOR M=1 T
O 21 :: DISPLAY AT(M,1):L$(M
):: NEXT M :: DISPLAY AT(23,
1):" PRESS ANY KEY TO CONTIN
UE"
4960 CALL KEY(5,KY,ST):: IF
ST<1 THEN 4960
4970 DISPLAY AT(23,1):"ONE M
OMENT..." :: FOR M=1 TO 21 :
: L$(M)="" :: NEXT M :: RETU
RN
5000 DISPLAY AT(1,1):"PRESS"
:" 1. TO ACCESS ANOTHER DISK
":" 2. TO ACCESS THE SAME DI
SK":" 3. TO END"
5010 CALL KEY(5,KY,ST):: IF
ST<1 THEN 5010 ELSE ON KY-48
 GOTO 100,110,999
```

The program listed above will allow establishing 2032 keywords on one disk! When the program is run, it automatically finds all DISPLAY FIXED 80 files on disk and allows selection from any of those files. Once a file is selected, the program automatically allows selection from any keyword in that file. Files without a properly formatted keyword index are no longer recognized in that session as a possible database file and the list of available files is offered for a new selection. Output can go to printer, disk or screen. Screen output can be unformatted or with words wrapped to the next line.

So, there you have it -- a multi-record database program. Now, if we could just find time to put an encyclopedia on disk....

Extended BASIC Assembly Language 1-2-3

by Barry A. Traver

STANDARD: 1A 2XB EA 4B 5A 6B 7B 9B

This is the second in a series of articles on using assembly language programs with Extended BASIC. (See the September 1985 issue of *Super 99 Monthly* for the previous article, "DISPLAY AT in Text Mode from Extended BASIC.") Chronologically second, the present article is logically first, but it has taken months to think through and test out an optimum approach for combining Extended BASIC with assembly language.

Jim Peterson of Tigercub Software has put together two excellent disks full of "Nuts 'n' Bolts" Extended BASIC subprograms. What I propose is that the Extended BASIC programmer add to that a similar library of assembly language programs that may be accessed from Extended BASIC. The idea in both cases, of course, would be that the Extended BASIC programmer would choose from these libraries the particular XB subprograms or a/l programs he would need for a particular Extended BASIC program.

It is not difficult to do at all; in fact, it's as simple as 1-2-3. All you need (after you have chosen which a/l programs you want to use) is to include them in an AGENDA. The AGENDA is composed of three parts: (1) the SETUP (which itself has three parts), (2) the "COPYLIST" (the individual a/l routines you have chosen), and (3) the END.

First, the SETUP (which ordinarily need never be changed) is composed of three parts, by which the following tasks are accomplished: (1) SET UP the Extended BASIC EQUates, (2) SET UP some required space (general WorkSpace plus perhaps a place for STRING storage), and (3) SET UP a safe return to Extended BASIC.

Second, the "COPYLIST" is the only part that changes from one occasion to another. It is composed

-->

- JANUARY 1986 4 -
SUPER 99 MONTHLY

of COPY directives for the source code files for the a/l programs you have decided to make use of.

Third, the END is simply the END. That's all there is to it: it's as simple as 1-2-3!

Since it's easier to show than it is to explain, careful study of the examples given should make everything clear. (For the sake of clarity, I like to repeat as comments in the code for the individual a/l programs information that also appears in SETUP and END, and I recommend that you do the same, but it is not necessary.)

The first example is a CALL LINK ("KEYR",R$). This is useful for two purposes. For one thing, it is a good substitute for CALL KEY, which usually causes a screen "glitch" in text mode. For another, it is a useful "shorthand" for the following frequently used code:

```
100 CALL KEY(0,K,S)
110 IF S=0 THEN 100
120 R$=CHR$(K)
```

The second example is a very simple CALL LINK("NEW"), which is similar to NEW in command mode, except that it -- unlike the CALL LOAD(-31962,100,124) noted by Craig Miller -- does not clear the screen (I've found it useful, but if you don't like my CALL LINK("NEW"), you can easily modify the code to do it his alternative way!).

One final recommendation. If you put a/l programs within XB subprograms with passed parameters, you don't have to write CALL LINK every time you want to access the a/l routine. For example, if you use CALL LINK("KEYR",R$) frequently and want to save on your typing, add the following XB subprogram to your XB program:

```
30000 SUB KEYR(R$)!! CALL LINK
("KEYR",R$)!! SUBEND
```

Then all you have to do in your main program to access your a/l program is simply say CALL KEYR(R$)!

I have discovered that with this approach, using assembly language programs with Extended BASIC is as simple as 1-2-3. (I only wish I could have discovered an equally easy way to work out the approach to begin with, and then you wouldn't have had to wait so long for this article!) I am sure, however, that many further refinements are possible, so I will be happy to receive any suggestions you might like to send, and we may be able to share some of them in future issues of *Super 99 Monthly*.

```
* AGENDA: An Extended BASIC - assembly language utility
*           by Barry Traver, 835 Green Valley Drive,
*             Philadelphia, PA 19128 (215/483-1379).

* Purpose:  to put together the complete agenda needed to
*  combine Extended BASIC with assembly language routines.
*  It's as simple as one-two-three!

* (1) SETUP

    COPY "DSK2.SETUP"

* (2) "COPYLIST" (This changes from one occasion to another.)

    COPY "DSK2.KEYR/S"
    COPY "DSK2.NEW/S"

* (3) END

    END
```

-->

```
* KEYR/S: An Extended BASIC-assembly language utility
*            by Barry Traver, 835 Green Valley Drive,
*               Philadelphia, PA 19128 (215/483-1379)

* Purpose:  to provide a CALL KEY alternative that
*  will not cause a screen "glitch" in text mode.
*  Also, to provide an alternative for the following
*  frequently-used Extended BASIC code:
*       100 CALL KEY(0,K,S)
*       110 IF S=0 THEN 100
*       120 R$=CHR$(K)
*  A CALL LINK("KEYR",R$) will accomplish the same
*  result.  In addition, a CALL KEYR(R$) will do the
*  same thing, if the following XB subprogram is added:
*       30000 SUB KEYR(R$)!! CALL LINK("KEYR",R$)!! SUBEND

       DEF  KEYR

* The following are included in SETUP:

* KEYDEV EQU  >8374
* KSCAN  EQU  >201C
* STATUS EQU  >837C
* STRASG EQU  >2010
*
* WS     BSS  32
* STRING BSS  256     * String Storage Space
*                       (actually, only 2 bytes
*                        really required)

KEYR   LWPI WS

       CLR  @KEYDEV           * Similar to
*                               CALL KEY(0,K,S)

K1     BLWP @KSCAN
       MOVB @STATUS,R1
       COC  @PRSTST,R1        * Does S=0?
       JNE  K1                * If so, stay in loop

       MOV  @KEYDEV,@STRING   * Move K to buffer
       LI   R5,>0100
       MOVB R5,@STRING        * Set string length to 1

       CLR  R0                * Not an array
       LI   R1,1              * Use 1st parameter
       LI   R2,STRING         * Location of string

       BLWP @STRASG           * Pass CHR$(K) to XB

       B    @RETURN           * Return to XB program

PRSTST DATA >2000             * "Press test" (i.e., S<>0)

* NEW/S:  An Extended BASIC-assembly language utility
*            by Barry Traver, 835 Green Valley Drive,
*               Philadelphia, PA 19128 (215/483-1379)

* Purpose:  to provide an XB statement (i.e., CALL
*  LINK("NEW") that is the equivalent of NEW in
*  immediate mode (useful at end of programs).

* This program provides the equivalent of a CALL
*  LOAD(-31952,255,231,255,231), although Craig Miller
*  in his excellent list of CALL PEEKs and CALL LOADS
*  in his fine Night Mission book mentions CALL
*  LOAD(-31962,100,124) an equivalent for NEW which
*  will also clear the screen, if such is desired.
```


-->

```
        DEF NEW

* The following are included in SETUP:

* GPLWS  EQU >83E0
* STATUS EQU >837C
*
* WS     BSS 32

NEW     LWPI WS
        LI   R1,>FFE7     >FFE7 = decimal 255,231
        MOV  R1,@>8330    Move 255,231 to memory location
*                          -31952
        MOV  R1,@>8332    Move 255,231 to memory location
*                          -31950
        B    @RETURN

* The following is included in AGENDA:

*       END

* SETUP:  An Extended BASIC - assembly language utility
*           by Barry Traver, 835 Green Valley Drive,
*             Philadelphia, PA 19128 (215/483-1379).

* Purpose:  to provide for general housekeeping needed to
*  combine Extended BASIC with assembly language routines.
*  It's as simple as one-two-three!

* (1) Set up Extended BASIC EQUates (See E/A manual, pp. 415-416)

ERR     EQU  >2034        error reporting utility
FAC     EQU  >834A        floating point accumulator
GPLWS   EQU  >83E0        gpl workspace
KEYDEV  EQU  >8374        key device
KEYVAL  EQU  >8375        key value
KSCAN   EQU  >201C        key scan routine
NUMASG  EQU  >2008        numeric assignment
NUMREF  EQU  >200C        get numeric parameter
STATUS  EQU  >837C        status register
STRASG  EQU  >2010        string assignment
STRREF  EQU  >2014        get string parameter
VMBR    EQU  >202C        vdp ram multiple byte read
VMBW    EQU  >2024        vdp ram multiple byte write
VSBR    EQU  >2028        vdp ram single byte read
VSBW    EQU  >2020        vdp ram single byte write
VWTR    EQU  >2030        vdp ram write to register
XMLLNK  EQU  >2018        link to rom utilities
XRTN    EQU  >8377        x return
YRTN    EQU  >8376        y return

* (2) Set up general Work Space and string storage space

WS      BSS  >20          designate workspace registers
STRING  BSS  256          string storage

* (3) Set up safe return to Extended BASIC

        DEF  RETURN

RETURN  LWPI GPLWS        load gpl workspace
        B    @>006A       return to extended basic, with
*                          status byte cleared (thanks to
*                          Paul Charlton for this one!)
```

The Terminal II

by Steven Szymkiewicz, MD

STANDARD: 1A 2EA 3B 4B 5A 6B 7B 9BC

Last summer, a friend gave me an ancient but repairable terminal and told me to quit griping about not having an 80 column display. Well, fixing the terminal was easy enough but using it as an alternate screen required reworking the BASIC editor, not to mention customizing any program I wanted to use on it. I put it aside and went onto other projects. Then I began learning FORTH and again longed for an 80 column display to see the entire program screen at one time. Fortunately, the creators of TI FORTH built into the language provisions for using alternate input and output devices.

Using alternate I/O devices is really quite simple but to build routines to use them requires an understanding of PABs (Peripheral Access Blocks). This information is found in both the FORTH manual and the Editor Assembler manual and should be read before attempting your own alternate I/O routines. This article follows step by step the screen for making an RS232 port both the input and output device of the TI.

First, be sure the file I/O routines have been loaded for use. This can be done by -FILE but if the routines are already loaded time will be wasted. A better way is to use CLOAD which begins loading a screen only if the word following it is not currently in the dictionary. Make certain the file I/O routines are located starting at screen 68.

```
DECIMAL 68 CLOAD STAT
```

The input and output files are created by the defining word FILE. Note that the PAB address (PAB-ADDR) for the file TERMIN is 16 bytes into the area for PABs (using the variable PABS) and that the PAB VDP buffer (PAB-VBUF) is the byte preceding it. PABs may reside anywhere in VDP RAM as long as they do not overlap onto an area already being used. However, the byte preceding the PAB must be the PAB-VBUF as a requirement for alterate I/O devices. Since the PABs will be used instead of the display and the keyboard, a buffer in RAM (PAB-BUF) does not need to be defined, thus the 0 is used.

```
PABS  16 + 0 OVER 1- FILE TERMIN

PABS  48 + 0 OVER 1- FILE TERMOUT
```

Now the alternate I/O procedure (named TERMINAL) can be defined. The file word must be used first to make it the referenced file. A PAB is created by SET-PAB for an input file from the second RS232 port and is opened. Default values open the file as sequential, fixed record length, and displayable data.

```
: TERMINAL

  TERMIN SET-PAB INPT F-D" RS232/2.BA=9600.CR.EC" OPN
```

The first byte of the PAB is the I/O opcode of the PAB. By changing it to a 2 the file is to be read. With alternate I/O devices the usual RD procedure is unnecessary.

```
  2 PAB-ADDR  VSBW
```

The fifth byte of the PAB is the record length, which will always be 1 for the input routine. This may also be changed using the REC-LEN routine.

```
  1 PAB-ADDR  4 + VSBW
```

The sixth byte of the PAB is the character count, which again is always 1

-->

for the input routine.

```
1 PAB-ADDR  5 + VSBW
```

Lastly, the PAB address is stored in ALTIN so that FORTH can use the PAB. Ordinarily, ALTIN has a value of 0, indicating that no alterate device is to be used for input.

```
PAB-ADDR  ALTIN !
```

Next, the terminal must also be the output device of the computer. Fortunately, the TI and COR-COMP I/O cards allow one serial port to be opened in two different PABs provided one PAB is input and the other output. Thus, another PAB is created exactly like the first except the PAB is to be written to only. This is done by using a 3 instead of a 2 in the first PAB byte.

```
TERMOUT SET-PAB OUTPT F-D" RS232/2 .BA=9600.CR.EC" OPN

3 PAB-ADDR  VSBW 1 PAB-ADDR  4 + VSBW 1 PAB-ADDR  5 + VSBW

PAB-ADDR  ALTOUT ! ;
```

Finally, there must be a way to get back to using the console as the I/O device. As mentioned previously, a 0 must be written into ALTIN and ALTOUT, and the alternate I/O files should be closed.

```
: CONSOLE

  0 ALTIN ! 0 ALTOUT ! TERMIN CLSE TERMOUT CLSE ;
```

That's it. Now, to use an 80 column terminal in FORTH just LOAD the screen and type TERMINAL. To revert back to the console keyboard and screen type CONSOLE. Alternate I/O devices may be useful for a number projects besides allowing the use of a terminal. It may be helpful in data collection, multiple displays or even terminal emulation. A limitation does crop up at higher speeds, however. The FORTH main program outputs and inputs data at the speed of one byte every sixieth of a second. Thus, the maximum speed for serial devices is 480 (8) BAUD.

Unfortunately, the editor that comes with TI FORTH does not use the I/O routines but directly manipulates VDP memory, so it cannot make use of the extra display area. Also, beware of the SWCH and UNSWCH routines for printing; these words change ALTOUT and should be modified to accommodate using TERMINAL. However, all other functions which do not manipulate VDP memory to create a display or change ALTIN or ALTOUT will work normally.

Since the editor from TI will not work with the terminal display I sumbit the following editor for terminal use. I will not dwell on its program since only a few people will wish to write their own editor. I admit my twisted mind makes for convoluted code and since I am not by nature a programmer, I welcome improvements and/or additions from those who are.

```
SCREEN 90
0     ( RS232 TERMINAL SUPPORT )
1     BASE->R DECIMAL 68 CLOAD STAT
2
3     PABS  16 + 0 OVER 1- FILE TERMIN
4     PABS  48 + 0 OVER 1- FILE TERMOUT
5
6     : TERMINAL
7       TERMIN  SET-PAB INPT  F-D" RS232/2.BA=9600.CR.EC" OPN
8       2 PAB-ADDR  VSBW 1 PAB-ADDR  5 + VSBW PAB-ADDR  ALTIN !
9       1 PAB-ADDR  4 + VSBW
10      TERMOUT SET-PAB OUTPT F-D" RS232/2.BA=9600.CR.EC" OPN
11      3 PAB-ADDR  VSBW 1 PAB-ADDR  5 + VSBW PAB-ADDR  ALTOUT !
12      1 PAB-ADDR  4 + VSBW ;
13
14    : CONSOLE 0 ALTIN ! 0 ALTOUT ! TERMIN CLSE TERMOUT CLSE ;
15    R->BASE -->
```

-->

```
SCREEN 91
0     ( TERMINAL EDITOR )
1     BASE->R DECIMAL
2     0 VARIABLE T-BUFF 0 VARIABLE CURSE 0 VARIABLE Z1 0 VARIABLE Z2
3     0 VARIABLE BUFF 80 ALLOT
4     : ID# ." SCREEN " SCR  . 25 EMIT ;  : CHAR3 EMIT EMIT EMIT ;
5     : TOP 9 9 10 CHAR3 5 1 DO 9 I 48 + 9 CHAR3 LOOP 9 13 10 CHAR3
6       32 EMIT ." ...............0...............0...............0"
7       ." ...............0" 13 10 EMIT EMIT ;
8     : SIDE 10 1 DO 7 1 DO 32 EMIT LOOP 13 10 I 48 + CHAR3 LOOP
9       6 0 DO 7 1 DO 32 EMIT LOOP 13 10 I 65 + CHAR3 LOOP 6 1 DO 32
10      EMIT LOOP 13 48 49 CHAR3 ;
11    : MARK T-BUFF  CURSE  + ;   : CHNG MARK C! ;
12    : BELL 7 EMIT ;   : CLR 12 EMIT ;
13    : HOME 25 EMIT 4 1 DO 10 EMIT LOOP 9 EMIT 32 EMIT 0 CURSE ! ;
14    : NEWLINE DUP 64 MOD 0= ;   : OLDLINE DUP 1+ 64 MOD 0= ;
15    R->BASE -->
SCREEN 92
0     ( TERMINAL EDITOR )
1     BASE->R DECIMAL
2     : UP CURSE  64 - DUP 0< IF BELL DROP ELSE CURSE ! 31 EMIT
3       ENDIF ;
4     : DOWN CURSE  64 + DUP 1023 > IF BELL DROP ELSE CURSE !
5       10 EMIT ENDIF
6     : LEFT CURSE  DUP 0= IF BELL DROP ELSE 1- OLDLINE IF 32 31 13
7       CHAR3 10 1 DO 9 EMIT LOOP ELSE 8 EMIT ENDIF CURSE ! ENDIF ;
8     : RIGHT CURSE  1+ DUP 1023 > IF BELL DROP ELSE NEWLINE IF
9       9 10 13 CHAR3 32 EMIT ELSE 28 EMIT ENDIF CURSE ! ENDIF ;
10    : TAB BEGIN RIGHT MARK C 32 - 0= CURSE  1023 = OR UNTIL
11      BEGIN RIGHT MARK C 32 - CURSE  1023 = OR UNTIL ;
12    : STED CURSE  16 OVER 64 / DO 64 0 DO DUP T-BUFF  + C EMIT 1+
13      LOOP 9 10 13 CHAR3 32 EMIT LOOP DROP HOME ;
14
15    R->BASE -->
SCREEN 93
0     ( TERMINAL EDITOR )
1     BASE->R DECIMAL
2     : DET 0 Z1 ! CURSE  BEGIN DUP 1+ 64 MOD WHILE 1 Z1 +! DUP DUP
3       1+ T-BUFF  + C DUP EMIT SWAP T-BUFF  + C! 1+ REPEAT 32 SWAP
4       T-BUFF  + C! 32 EMIT 8 EMIT Z1  0 DO 8 EMIT LOOP ;
5     : IST 0 Z1 ! 32 Z2 ! CURSE  BEGIN DUP Z2  DUP EMIT SWAP T-BUFF
6       + DUP C Z ! C! 1+ DUP 64 MOD WHILE 1 Z1 +! REPEAT
7       8 EMIT Z1  0 DO 8 EMIT LOOP ;
8     : START DUP SCR ! CLR TOP SIDE HOME BLOCK T-BUFF !
9       0 CURSE ! STED ;
10    : NEXT SCR  1+ DUP DISK_HI  < IF DUP START THEN DROP ;
11    : LAST SCR  1- DUP DISK_LO  > IF DUP START THEN DROP ;
12
13    : RETURN CURSE  DUP 64 / 1+ 64 * SWAP - 0 DO RIGHT LOOP ;
14
15    R->BASE -->
SCREEN 94
0     ( TERMINAL EDITOR )
1     BASE->R DECIMAL
2     : ROW CURSE  64 / ;                 : COL CURSE  64 MOD ;
3     : ROWAD DUP 64 * T-BUFF  + ;    : BLANKROW 80 32 FILL ;
4     : MROW 32 0 DO OVER  OVER ! 2+ SWAP 2+ SWAP LOOP DROP DROP ;
5     : POSITION BEGIN 1- DUP WHILE OVER EMIT REPEAT DROP DROP ;
6
7     : DELL ROW ROWAD BUFF MROW ROW CURSE  Z1 ! BEGIN DUP 15 - WHILE
8       DUP ROWAD DUP 64 + SWAP MROW 1+ REPEAT ROWAD BLANKROW HOME Z1
9        64 / 64 * CURSE ! ROW 0 DO 10 EMIT LOOP STED Z1  CURSE !
10      28 COL 1+ POSITION 10 ROW 1+ POSITION ;
11    : INSL 15 BEGIN DUP ROW - WHILE DUP ROWAD DUP 46 - SWAP MROW 1-
12      REPEAT ROWAD BUFF SWAP MROW CURSE ! DUP Z1 ! HOME 64 / 64 *
13      CURSE ! ROW 0 DO 10 EMIT LOOP STED Z!  CURSE ! 28 COL 1+
14      POSITION 10 ROW 1+ POSITION ;
15    R->BASE -->
10      TERMOUT SET-PAB OUTPT F-D" RS232/2.BA=9600.CR.EC" OPN
11      3 PAB-ADDR  VSBW 1 PAB-ADDR  5 + VSBW PAB-ADDR  ALTOUT !
12      1 PAB-ADDR  4 + VSBW ;
13
14    : CONSOLE 0 ALTIN ! 0 ALTOUT ! TERMIN CLSE TERMOUT CLSE ;
15    R->BASE -->
```

-->

```
SCREEN 95
0     ( TERMINAL EDITOR )
1     BASE->R DECIMAL
2     : EDIT START BEGIN KEY DUP 27 - WHILE CASE
3        8 OF LEFT                  ENDOF    28 OF RIGHT       ENDOF
4       10 OF DOWN                  ENDOF    31 OF UP          ENDOF
5        7 OF BELL                  ENDOF     9 OF TAB         ENDOF
6       12 OF IST                   ENDOF   127 OF DET         ENDOF
7       29 OF DELL                  ENDOF    11 OF INSL        ENDOF
8       20 OF NEXT                  ENDOF    22 OF LAST        ENDOF
9       25 OF HOME                  ENDOF    13 OF RETURN      ENDOF
10      DUP 31 SWAP < IF DUP EMIT DUP CHNG 1 CURSE +! CURSE  DUP
11      1024 = IF HOME THEN NEWLINE IF 13 EMIT 10 EMIT 9 EMIT THEN
12      UPDATE THEN ENDCASE REPEAT DROP 12 EMIT ;
13    : ED SCR  EDIT ;
14
15    R->BASE
```

CORRECTIONS:
December 1985: The reference to the number of sectors required for saving the Extended BASIC module to disk via the GRAM Kracker™ was incorrect. The correct sector count is 204. Also, "console" was misspelled as "council".

In the Extended BASIC Link article, the example listing on page 6 should have read "DSK1.SCREEN/O" instead of "DSK1.SCREEN/S".

In the article on chaining programs, no mention was made of sprite data being chained. Perhaps a full tutorial on that subject may be appropriate, but in the meantime a minimal approach would be to CALL DELSPRITE(ALL).

SHOW NEWS

99' Fest West '86

One of the major spring shows for 99'ers will be 99' Fest West '86 in Los Angeles, now only days away. Vendors currently scheduled to appear include (a few booths remained available at press time, but will surely be occupied by the show date):

Super 99 Monthly
Millers Graphics
Genial TRAVeleR
Kent Thompson
Holmes and Company
MYARC
Compuserve Information Services
The Source
Asgard Software
Stewart Company
Dijit Systems
99'er User Group Association
Texaments
Computer Shopper
MICROpendium
Byte Data
T.A.P.E., Ltd.
Data Systems
DataBioTics
Irish Input

Several new products will be shown, including a new hardware device from Millers Graphics. Details of the device are not being released, but rumor has it the device will be a peripheral expansion card that will include a Z80 microprocessor on board. The only clue released by the firm is "the decimal number 15", a rather vague hint that has users eager to see what Millers Graphics has in store for 99'ers for 1986 after their 1985 release of the highly acclaimed GRAM Kracker™.

Our own publishing firm, Bytemaster Computer Services, will debut two disk software packages. One will be a hodge-podge of useful programs and the other will be a specialized database. In addition, *Super 99 Monthly* will have a major announcement at the show.

Asgard Software will be introducing at least three new software offerings.

Approximately 2000 users are expected to attend the 99' Fest West '86 show at the Shrine Auditorium in Los Angeles on March 1 and 2. Next month, we'll have complete coverage of the show.

GRAM KRACKER™

Manual Released

STANDARD: 1A 9B 15A

The completed manual and disk for GK have now been released. Overall, the manual and disk represent an excellent piece of work.

There are a few problems with the programs on the disk. First, the NEWCHARS program accesses CHARA1 files from TI and, while Millers Graphics knew of some user modifications, it was not discovered until after release of the disk that TI had apparently released more than one version of their TI-Writer upgrade disk that included the CHARA1 file. Versions that use 8 pixel high characters will appear without the top pixels showing because BASIC only loads data for 7 pixels. A fix was not available by press time, but one will likely be available for next month's issue.

Additionally, when the GROM portion of TI-Writer is moved with a utility on the GK disk, the Formatter is not accessed properly. The fix is very simple. Using a sector editor, find the first sector of the FORMA1 file (using Advanced Diagnostics use FF FORMA1). Beginning at byte 33 (the 34th byte), in hex, the values are 9C 02 10 00D0 60. Change byte 35 to 11 (from NOP to JMP past module check) leaving the values 9C 02 10 11 D0 60.

If any other problems arise, we will, of course, give full coverage as quickly as possible so that everyone can make full use of the fantastic GK device. Additionally, we have delayed getting into any advanced projects until after release of the GK disk, in order to attempt to avoid conflict with the location of code from Millers Graphics. We'll try to bring you some advanced projects soon.

-->

SUBSCRIPTIONS (PER YEAR):

U.S. AND POSSESSIONS	
FIRST CLASS	$16.00
THIRD CLASS	$12.00
OTHER COUNTRIES	
AIR MAIL	$26.50
SURFACE MAIL	$16.00
INDIVIDUAL COPIES:	
U.S. SUBSCRIBERS	
FIRST CLASS	$ 1.35
THIRD CLASS	$ 1.00
CANADA SUBSCRIBERS	$ 1.35
OTHER	$ 1.50

Check or Money Order in U.S. funds, coded for processing through the U.S. Federal Reserve Bank System.
No billings or credit sales.
(all issues available at press time)

SUPER 99 MONTHLY ORDER FORM

NAME_____

ADDRESS_____

CITY_____STATE_____

ZIP_____COUNTRY_____

For back issues, specify which:

_____

READER FEEDBACK: (Attach comments)

SUPER 99 MONTHLY is published monthly by Bytemaster Computer Services, 171 Mustang Street, Sulphur, LA 70663.

STANDARD KEY

1	Computer	A	TI-99/4A
2	Module	XB	Extended BASIC
		TW	TI-Writer
		EA	Editor/Assembler
3	RS-232	B	TI
4	Disk Drive	B	TEAC 55-B
5	Expansion Box	A	TI
6	Disk Controller	B	CorComp
7	Memory Card	B	MYARC MEXP-1 (128K)
9	Monitor or TV	B	TI Color Monitor
10	Printer	B	Gemini 15-X
15	GRAM device	A	GRAM Kracker (tm)

EDITOR
Richard M. Mitchell (CIS 70337,1011)

CORRESPONDING STAFF WRITERS
Barry A. Traver
Charles M. Robertson
Steven J. Szymkiewicz, MD

Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 70663

Bulk Rate
U.S. Postage
PAID
Sulphur, LA 70663
Permit No. 141

POSTMASTER: ADDRESS CORRECTION REQUESTED.
RUSH -- TIME DATED MATERIAL.