MILLERS GRAPHICS

APRIL 1984

$1.50

THE SMART PROGRAMMER

To start off the April newsletter I would like to apologize for the tardiness in getting it out. My recent involvement with CorComp's Board of Directors has required a fair amount of my time. We are deeply involved with the software and hardware design of new products for the 99/4A. I'm in hopes that in a few issues from now we will be back on our original schedule of mailing out the newsletter in the early part of the month. Until then please bear with me, I believe that you will be pleased with both the newsletter and with the new CorComp products.

On the question of LONG or SHORT programs it looks like its a toss up. So our current plans are to keep going as we have and to place an occasional long program in some of the future issues. Now that we are working with CorComp, I hope to be able to bring you some articles of a technical nature. So gather up your TECH questions and send them in and I'll talk them over with the electrical engineers at CorComp.

While we are on the subject of CorComp I would like to talk about a few new things. First off they are starting up a program to support 99/4A User's Groups. They have recently mailed out a package to all the User's Groups they knew of at the time. One of the items in this package was a card to be filled out and mailed back to CorComp. The card asked for the name and address of the group and for a phone number and person to contact at the group. What they plan to do is to compile a list of ALL the 99/4A User's Groups and send it out with each new product that is shipped from CorComp Inc. If your User's Group has NOT received this package then please have your group write to CorComp and let them know. Their address is:

CorComp Inc.
1255 North Tustin Ave.
Anaheim, CA 92807

I talked about CorComp's three new products last month but I would like to bring you more up to date with them. The 99000 EXPANSION SYSTEM is approx 4 inches high by 12.5 inches long and 12 inches deep, a little bit bigger than two full height disk drives sitting side by side. The graphics display panel on the right hand side of the face returns the following information about the 99000 EXPANSION SYSTEM, your computer and your disk drives.

DRIVE SELECT	- DSK1 through DSK4
DRIVE STATUS	- Side 1, Side 2, Index Mark, Track 0, Step In and Step Out.
SYSTEM STATUS	- CRU Clock, Interrupt Request and WAIT.
MOTHERBOARD SYSTEMS	- Flex Cable, RS232, 32K RAM, Disk Controller, RAM Disk (optional) and AUX 1 through AUX 4 (optional).

There are also 7 switches on this panel which control the following items. POWER, RS232, 32K RAM, Disk Controller, System PAUSE, ECHO and RESET. The ECHO switch is for modem users in that it will allow you to Echo your modem input directly out to your printer! The RESET switch serves 2 functions, first it will RESET your computer back to the Title screen. Secondly, for the advanced programmer, you can use it to activate the LOAD interrupt. By this I mean that you can load your workspace pointer into >FFFC and your PC into >FFFE and then when you press RESET the computer will leave what it is doing and hop into say a Debugger or whatever assembly program you want it to! This EXPANSION SYSTEM is not just an add-on its a TRUE UPGRADE for the 99/4A as well as being part of the 99000 Computer System.

The Double-sided Double-density disk controller that is in the 99000 EXPANSION SYSTEM, the 9900 Disk Controller card and the 9900 Micro-Expansion System will add the following new commands and programming statements to Basic and Extended Basic:

CALL POKE & CALL PEEK for rapid reading and writing to CPU Memory.
CALL POKEV, CALL PEEKV for rapid reading and writing to VDP RAM.
CALL MOVEM for moving blocks of memory from VDP RAM to VDP RAM or CPU Memory to VDP RAM or VDP RAM to CPU RAM or CPU Memory to CPU RAM. This new CALL can move approx 30-40 screens full of data per second!!!
CALL EXEC for Executing ROM or Expansion Memory routines.
CALL MGR for loading and running the NEW CorComp Disk Manager program.

The NEW CorComp Disk Manager program is supplied on disk. Not only is this program considerably faster than TI's, it is also a pleasure to use! (Not that I'm biased or anything). One of the many enhancements of this program is that it allows you to CONFIGURE the manager to <u>your system</u>. You can select your own text and screen colors, set up each of the different drive types attached to the controller for number of sides, density and number of tracks and set up your printer type for catalog and disk test print outs. This CONFIGURE is saved on the disk and is used for the defaults when you load the disk manager! I think you'll like it, its truly a nice utility.

The 9900 Micro-Expansion System is what I called a stand alone RS232 Interface in the last issue. It is actually a small (about the size of 2 speech synthesizers put together) Expansion System that plugs directly into the side of your 99/4A. You can buy it as a RS232 Interface only and then at a later date add the 32K RAM and Disk Controller option board or buy it with everything installed. The suggested retail price for the 9900 Micro-Expansion System with a RS232 Interface is 149.95. The suggested retail price for the 9900 Micro-Expansion System with the RS232 Interface, 32K RAM and Disk Controller is 399.95

For further information and pricing PLEASE contact your local dealer.

OOPS!

Boy did we goof on the printing of more than 80 columns in the last issue. Quite a number of you wrote to us with the following info. To print more than 80 columns just OPEN your output device using the VARIABLE statement, such as:

OPEN #1: "PIO",VARIABLE 132

This will allow a 132 column printout without having to worry about the carriage returns and line feeds. Thanks to everyone that sent us this information.

Q & A

This is VDP month. In the past we have received a number of questions regarding VDP memory, so this month we are devoting the Q & A section to this area. Instead of answering individual questions we will start our series on the VDP memory with a general overview. Next month we will get a little more specific on the Extended Basic use of VDP. To get started lets briefly look at each of the main areas of VDP memory as they are used by Extended Basic.

SCREEN IMAGE TABLE >000 - >02FF

This area of VDP RAM holds the items that you see on your screen. Hex >0000 is the first column of the first row (upper left hand corner of the screen). The values in this table have an offset (bias) of Hex >60 or Decimal 96. So to place an "A" on the screen at >0000 you must POKEV a value of 161 at address >0000, (65+96=161). Using the POKEV routine that is in this issue the statement that would accomplish this would look like; CALL LINK("POKEV",0,161). To place a "MG" on the screen on the 12th row at the 15th column you would use; CALL LINK("POKEV",367,173,167). The formula to find the row and column is: (ROW-1)*32+COL, (12-1)*32+15=367, M=77+96=173, G=71+96=167. In the next issue I'll try to explain why there is an offset of 96.

SPRITE ATTRIBUTE TABLE >0300 - >036F

This table holds the information on the sprites that is placed there by CALL SPRITE or you can directly POKEV the information into this table. We will go into more info on this table in a latter issue but for now this is what it holds for each of the 28 sprites. The dot row and dot column position of the sprite. The character number of the sprite and its color.

PATTERN DESCRIPTOR TABLE >0370 - >077F

When you use CALL CHAR(..... the character's pattern that you defined is placed in this table. Also when the computer powers up, the standard character set is loaded into this table. Each character uses up eight bytes. To find the starting address in decimal for a character use the following formula. 768+8*char num. So for "A" the address would be; 768+8*65=1288. The address for the cursor is 1008 (768+8*30=1008).

WARNING--- changing the pattern for character numbers 0 through 29 could cause your system to lock up. This area is used for temporary storage of system information, disk information, screen information and the sound table by Basic and Extended Basic. We will explain this area better in a latter issue.

SPRITE PATTERN TABLE >0370 - >077F

In Extended Basic this table and the Pattern Descriptor table reside in the same place. That is why when you CALL CHAR to a character that is on the screen and that is also used for a sprite, they both will change. In assembly language these two tables are in separate parts of VDP RAM so the characters used for sprites are different than the ones printed on the screen. In X-Basic the formula for the sprite character's address in decimal is the same as the one for the Pattern Descriptor Table.

SPRITE MOTION TABLE >0780 - >07FF

This table holds the sprite row and column velocity and it is used by the Interrupt routine in console ROM. This ROM routine is executed 60 times a second since it is Interrupt driven and it uses the values in this table to update the row and column values in the Sprite Attribute Table. Unlike the other VDP tables this table's location is fixed at >0780 by this ROM routine. The other tables can be moved around by changing the different VDP Register values, that is why we can have different VDP memory maps for the different languages. Each sprite uses 4 bytes in this table. One for row velocity, one for column velocity and two that are reserved for system use. If you compare the BASIC and Extended BASIC VDP maps for this area you'll notice that this table is sitting where character sets 15 & 16 used to be and that is why these characters are not available to us in Extended BASIC.

COLOR TABLE >0800 - >081F

This table holds the foreground and background color information for each of the character sets. Each character set uses one byte for its color definition. The first four bits, 0-3, set the foreground color and the last four bits, 4-7, set the background color. Here are the character numbers for each of the 32 bytes in this table:

Byte	Char's	Byte	Char's	Byte	Char's
1	0- 7	12	88- 95	23	176-183
2	8- 15	13	96-103	24	184-191
3	16- 23	14	104-111	25	192-199
4	24- 31	15	112-119	26	200-207
5	32- 39	16	120-127	27	208-215
6	40- 47	17	128-135	28	216-223
7	48- 55	18	136-143	29	224-231
8	56- 63	19	144-151	30	232-239
9	64- 71	20	152-159	31	240-247
10	72- 79	21	160-167	32	248-255
11	80- 87	22	168-175		

DYNAMIC MEMORY SPACE >0820 - >35D7

This area of memory holds the PABS, STRINGS, SYMBOL TABLES, NUMERIC VALUE TABLE, LINE NUMBER TABLE and your PROGRAM if you do not have memory expansion or if you are programming in BASIC. In Extended BASIC with memory expansion this area holds the PABS, STRINGS and SYMBOL TABLES, the other items are moved out to high memory expansion. In BASIC your program is loaded from the bottom (>35D7) up. So if the first line that you typed in is line number 100 it will be located around >35C7 (depending on its length). The next line that you type in might be around >35B0, etc. So if you typed in the program lines in order, ie: 100, 110, 120 etc., they are placed in memory like this:

```
>0800 +--------------------------------+
      |                                |
      |       LINE NUMBER 150 LINE NUMBER|
      | 140  LINE NUMBER 130 LINE NUMBER 1|
      |20  LINE NUMBER 110 LINE NUMBER 100|
      |                                |
>3FFF +--------------------------------+
```

FILE BUFFERS >35D8 - >3FFF

With CALL FILES(3) this area starts at >35D8, any other CALL FILES will move its starting address higher or lower. On POWER UP, with a disk controller, the computer automatically reserves this space for Drive control information, File allocation information and Data buffering. Do not change any bytes in this area our you may wipe out your diskettes.

OVERALL VDP MAPS WITH BASIC AND EXTENDED BASIC

```
Hex     Dec   BASIC                          Hex     Dec   ENTENDED BASIC
             +-----------------------------+                +-----------------------------+
>0000     0  | SCREEN IMAGE TABLE          |  >0000     0  | SCREEN IMAGE TABLE          |
             | Start PATTERN DESC TABLE    |                | Start PATTERN DESC TABLE    |
             |                             |                | Start SPRITE PATTERN TABLE  |
             | +96 Offset (>60 Bias)       |                | +96 Offset (>60 Bias)       |
>02FF   767  | END SCREEN IMAGE            |  >02FF   767  | End Screen Image            |
             +-----------------------------+                +-----------------------------+
>0300   768  | COLOR TABLE                 |  >0300   768  | SPRITE ATTRIBUTE TABLE      |
>031F   799  |                             |                |                             |
             +-----------------------------+                |                             |
             |                             |  >036F   879  |                             |
             +-----------------------------+                +-----------------------------+
>0370   880  | PATTERN DESC TABLE          |  >0370   880  | PATTERN DESC TABLE          |
             | +96 Offset (>60 Bias)       |                | +96 Offset (>60 Bias)       |
             |                             |                |                             |
             |                             |                |                             |
             | 768+8*character number=     |                | 768+8*character number=     |
             |  address in decimal         |                |  address in decimal         |
             |                             |                |                             |
>03F0  1008  | Character number 30         |                |                             |
>03F8  1016  | Character number 31         |                |                             |
>0400  1024  | Character number 32         |                |                             |
>0408  1032  | Character number 33         |                |                             |
             |     etc.                    |                |                             |
             |                             |                |                             |
             |                             |                |                             |
             |                             |  >077F  1919  |                             |
             |                             |                +-----------------------------+
             |                             |  >0780  1920  | SPRITE MOTION TABLE         |
>07FF  2047  |                             |  >07FF  2047  |                             |
             +-----------------------------+                +-----------------------------+
>0800  2048  | PABS                        |  >0800  2048  | COLOR TABLE                 |
             | STRINGS                     |  >081F  2079  |                             |
             | SYMBOL TABLES               |                +-----------------------------+
             | NUMERIC VALUES              |  >0820  2080  |                             |
             | LINE NUMBER TABLE           |                | Without MEM-EXPANSION       |
             | PROGRAM SPACE               |                | SAME USE AS BASIC           |
             |                             |                |                             |
             |                             |                |  With MEM-EXPANSION         |
             |                             |                |     PABS                    |
             |                             |                |     STRINGS                 |
             |                             |                |     SYMBOL TABLES           |
             |                             |                |  (numeric values,           |
             |                             |                |   line number table,        |
             |                             |                |   and program are           |
             |                             |                |   in High Memory            |
             |                             |                |   Expansion)                |
             |                             |                |                             |
>35D7 13783  |                             |  >35D7 13783  |                             |
             +-----------------------------+                +-----------------------------+
>35D8 13784  |                             |  >35D8 13784  |                             |
             | DISK FILE BUFFERS           |                | DISK FILE BUFFERS           |
>3FFF 16383  |                             |  >3FFF 16383  |                             |
             +-----------------------------+                +-----------------------------+
```

TMS9918A VDP REGISTERS

REG #	BIT #	DESCRIPTION
0	0-5	Reserved for future use must be 0's
	6	M3 (mode bit 3) 1=Graphics II mode
	7	External VDP enable 0=disable 1=enable
1	0	4/16 K selection 0=4027 RAM 4K 1=4108/4116 RAM 16K
	1	Blank enable/disable 0=blank 1=display
	2	Interrupt enable 0=disable 1=enable
	3	M1 (mode bit 1) 1=Text mode
	4	M2 (mode bit 2) 1=Multicolor mode
	5	Reserved
	6	Size (sprite size select) 0=8x8 bit 1=16x16 bit
	7	Magnification 0=1x 1=2x

M1	M2	M3	Mode	
0	0	0	Graphics I mode	32 column
0	0	1	Graphics II mode	Bit-Mapped
0	1	0	Multicolor mode	64 column
1	0	0	Text mode	40 column

REG #	BIT #	DESCRIPTION
2	0-3	Reserved
	4-7	Screen table base address = R2 * >400
3	0-7	Color table base address = R3 * >40
4	0-4	Reserved
	5-7	Pattern (character) descriptor table base = R4 * >800
5	0	Reserved
	1-7	Sprite attribute table base address = R5 * >80
6	0-4	Reserved
	5-7	Sprite pattern table base address = R6 * >800
7	0-3	Foreground color (character color) in TEXT mode.
	4-7	Background color (screen color) all modes.
Status	0	Interrupt Flag bit.
	1	5 or more sprites on the same row.
	2	Sprite coincidence.
	3-7	Number of the 5th sprite when bit 1 is on.

EIGHT WRITE ONLY VDP REGISTERS

AND THE VDP STATUS REGISTER

```
Bit #  0         1         2         3         4         5         6         7

   >80 128   >40   64  >20   32  >10   16  >8     8  >4     4  >2     2  >1     1
  +---------+---------+---------+---------+---------+---------+---------+---------+
0 |    0    |    0    |    0    |    0    |    0    |    0    |   M3    |Ext VDP  |
  |         |         |         |         |         |         | bit-map |         |
  +---------+---------+---------+---------+---------+---------+---------+---------+
  B=>00 (0)                    XB=>00 (0)                    EA=>00 (0)

  +---------+---------+---------+---------+---------+---------+---------+---------+
1 | 4/16K   | BLANK   |Interrup |   M1    |   M2    |    0    |  SIZE   |   MAG   |
  |  RAM    | SCREEN  | Enable  |  text   |  multi  |         |         |         |
  +---1-----+---1-----+---1-----+---0-----+---0-----+---0-----+---0-----+---0-----+
  B=>E0 (224)                  XB=>E0 (224)                  EA=>E0 (224)

  +---------+---------+---------+---------+---------+---------+---------+---------+
2 |    0    |    0    |    0    |    0    | SCREEN TABLE BASE ADDRESS   x >400    |
  |         |         |         |         |                                       |
  +---------+---------+---------+---------+---------+---------+---------+---------+
  B=>00 (0) {>0000}            XB=>00 (0) {>0000}            EA=>00 (0) {>0000}

  +---------+---------+---------+---------+---------+---------+---------+---------+
3 |                     COLOR   TABLE   BASE   ADDRESS   x >40                    |
  |                                                                               |
  +---------+---------+---------+---------+---------+---------+---------+---------+
  B=>0C (12) {>0300}           XB=>20 (32) {>0800}           EA=>0E (15) {>0380}

  +---------+---------+---------+---------+---------+---------+---------+---------+
4 |    0    |    0    |    0    |    0    |    0    |CHARACTER BASE ADD x >800    |
  |         |         |         |         |         |                             |
  +---------+---------+---------+---------+---------+---------+---------+---------+
  B=>00 (0) {>0000}            XB=>00 (0) {>0000}            EA=>01 (1) {>0800}

  +---------+---------+---------+---------+---------+---------+---------+---------+
5 |    0    |           SPRITE  ATTRIBUTE  TABLE  BASE  ADDRESS   x >80           |
  |         |                                                                     |
  +---------+---------+---------+---------+---------+---------+---------+---------+
  B=>06 (6) {>0300}            XB=>06 (6) {>0300}            EA=>06 (6) {>0300}

  +---------+---------+---------+---------+---------+---------+---------+---------+
6 |    0    |    0    |    0    |    0    |    0    |SPRITE PAT BASE ADD x >800|
  |         |         |         |         |         |                             |
  +---------+---------+---------+---------+---------+---------+---------+---------+
  B=>00 (0) {>0000}            XB=>00 (0) {>0000}            EA=>00 (0) {>0000}

  +---------+---------+---------+---------+---------+---------+---------+---------+
7 |     TEXT MODE FOREGROUND COLOR          |      SCREEN COLOR  (all modes)        |
  |                                         |                                       |
  +---------+---------+---------+---------+---------+---------+---------+---------+
  B=>07 (7) Cyan               XB=>07 (7) Cyan        EA=>F5 (245) White/Lt Blue

  +---------+---------+---------+---------+---------+---------+---------+---------+
S |Interrup |5 Sprite | COINC   |           FIFTH SPRITE NUMBER                   |
  |  Flag   |on a row |         |            when bit 1 is set                    |
  +---------+---------+---------+---------+---------+---------+---------+---------+
          *-----------------------------------*
          *                                   *
          *    VDP PEEKV, POKEV & POKER       *
          *              by                   *
          *          John Brown               *
          *                                   *
          *                                   *
          * Syntax Info                       *
          * PEEKV & POKEV can peek or poke    *
          * up to 15 values.                  *
          *                                   *
          * add = Address to PEEK or POKE     *
          *   v = Numeric Variable            *
          *   n = Number or numeric           *
          *                    expression     *
          *-----------------------------------*

                        DEF  PEEKV,POKEV,POKER
     200C     NUMREF    EQU  >200C
     2008     NUMASG    EQU  >2008
     2018     XMLLNK    EQU  >2018
     2024     VMBW      EQU  >2024
     202C     VMBR      EQU  >202C
     2030     VWTR      EQU  >2030
     834A     FAC       EQU  >834A
0000          BUFF      BSS  18
0012 0064     STORE     DATA 100
0014          MYWS      BSS  10
          *-----------------------------------*
          * PEEKV Reads VDP RAM Value(s)      *
          *                                   *
          * CALL LINK("PEEKV",add,v,v....)    *
          *-----------------------------------*
001E 02E0     PEEKV     LWPI MYWS
0020 0014'
0022 0300               LIMI 0
0024 0000
0026 0200               LI   R0,100
0028 0064
002A C800               MOV  R0,@STORE
002C 0012'
002E 04C0               CLR  R0
0030 0201               LI   R1,1
0032 0001
0034 0420               BLWP @NUMREF
0036 200C
0038 0420               BLWP @XMLLNK
003A 2018
003C 12B8               DATA >12B8
003E C020               MOV  @FAC,R0
0040 834A
0042 0201               LI   R1,BUFF
0044 0000'
0046 D0A0               MOVB @>8312,R2
0048 8312
004A 0982               SRL  R2,8
004C 0222               AI   R2,-1
004E FFFF
0050 0420               BLWP @VMBR
0052 202C
0054 04C5     LOOP1     CLR  R5
0056 D122               MOVB @BUFF-1(R2),R4
0058 FFFF'
005A 0984               SRL  R4,8
005C 1315               JEQ  NEXT
005E 04C3               CLR  R3
0060 3CE0               DIV  @STORE,R3
0062 0012'
0064 C805               MOV  R5,@FAC+2
0066 834C
0068 C805               MOV  R5,@FAC+4
006A 834E
006C C805               MOV  R5,@FAC+6
006E 8350
0070 0205               LI   R5,>4000
0072 4000
0074 A144               A    R4,R5
0076 0283               CI   R3,1
0078 0001
007A 1106               JLT  NEXT
007C 0205               LI   R5,>4100
007E 4100
0080 A143               A    R3,R5
0082 06C4               SWPB R4
0084 C804               MOV  R4,@FAC+2
0086 834C
0088 C805     NEXT      MOV  R5,@FAC
008A 834A
008C 04C0               CLR  R0
008E C042               MOV  R2,R1
0090 0581               INC  R1
0092 0420               BLWP @NUMASG
0094 2008
0096 0602               DEC  R2
0098 16DD               JNE  LOOP1
009A 0460               B    @RETURN
009C 00F8'
          *-----------------------------------*
          * POKEV Writes VDP RAM value(s)     *
          *                                   *
          * CALL LINK("POKEV",add,nv,nv...)   *
          *-----------------------------------*
009E 02E0     POKEV     LWPI MYWS
00A0 0014'
00A2 0300               LIMI 0
00A4 0000
00A6 04C0               CLR  R0
00A8 0201               LI   R1,1
00AA 0001
00AC C801               MOV  R1,@STORE
00AE 0012'
00B0 0420               BLWP @NUMREF
00B2 200C
00B4 0420               BLWP @XMLLNK
00B6 2018
00B8 12B8               DATA >12B8
00BA C820               MOV  @FAC,@BUFF
00BC 834A
```

```
     00BE 0000'
     0000 B820          AB   @>8312,@STORE+1
     00C2 8312
     00C4 0013'
     00C6 0203          LI   R3,2
     00C8 0002
     00CA 04C0   LOOP2  CLR  R0
     00CC C043          MOV  R3,R1
     00CE 0420          BLWP @NUMREF
     00D0 200C
     00D2 0420          BLWP @XMLLNK
     00D4 2018
     00D6 12B8          DATA >12B8
     00D8 D8E0          MOVB @FAC+1,@BUFF(R3)
     00DA 834B
     00DC 0000'
     00DE 0583          INC  R3
     00E0 8803          C    R3,@STORE
     00E2 0012'
     00E4 16F2          JNE  LOOP2
     00E6 C020          MOV  @BUFF,R0
     00E8 0000'
     00EA 0201          LI   R1,BUFF+2
     00EC 0002'
     00EE C083          MOV  R3,R2
     00F0 0222          AI   R2,-2
     00F2 FFFE
     00F4 0420          BLWP @VMBW
     00F6 2024
     00F8 04C0   RETURN CLR  R0
     00FA D800          MOVB R0,@>837C
     00FC 837C
     00FE 02E0          LWPI >83E0
     0100 83E0
     0102 0460          B    @>0070
     0104 0070
                 *-----------------------------*
                 * POKER Writes to a VDP Register *
                 *                             *
                 * CALL LINK("POKER",reg no.,value)*
                 *-----------------------------*
     0106 0300   POKER  LIMI 0
     0108 0000
     010A 04C0          CLR  R0
     010C 0201          LI   R1,1
     010E 0001
     0110 0420          BLWP @NUMREF
     0112 200C
     0114 C820          MOV  @FAC,@STORE
     0116 834A
     0118 0012'
     011A 0201          LI   R1,2
     011C 0002
     011E 0420          BLWP @NUMREF
     0120 200C
     0122 0420          BLWP @XMLLNK
     0124 2018
     0126 12B8          DATA >12B8
     0128 C020          MOV  @FAC,R0
     012A 834A
     012C D020          MOVB @STORE+1,R0
     012E 0013'
     0130 0420          BLWP @VWTR
     0132 2030
     0134 045B          RT
                        END
0000 ERRORS
```

PEEKV, POKEV & POKER   CALL LOADS

The following program contains the CALL LOADS to install the CALL LINK("PEEKV",.. CALL LINK("POKEV",.. & CALL LINK("POKER",.. VDP routines into Low Memory Expansion.

```
100 CALL CLEAR :: CALL INIT
:: CALL LOAD(8196,63,232)!
set up Last Free Address LFA

110 CALL LOAD(16360,80,79,75
,69,82,32,38,12,80,79,75,69,
86,32,37,164,80,69,69,75,86,
32,37,36)! set up REF/DEF

120 CALL LOAD(9491,100)! STO
RE data 100

125 ! Start PEEKV routine

130 CALL LOAD(9508,2,224,37,
20,3,0,0,0,2,0,0,100,200,0,3
7,18,4,192,2,1,0,1,4,32,32,1
2,4,32)

140 CALL LOAD(9536,32,24,18,
184,192,32,131,74,2,1,37,0,2
08,160,131,18,9,130,2,34,255
,255,4,32,32,44)

145 ! Start LOOP1 routine

150 CALL LOAD(9562,4,197,209
,34,36,255,9,132,19,21,4,195
,60,224,37,18,200,5,131,76,2
00,5,131,78,200,5)

160 CALL LOAD(9588,131,80,2,
5,64,0,161,68,2,131,0,1,17,6
,2,5,65,0,161,67,6,196,200,4
,131,76)

165 ! Start NEXT routine

170 CALL LOAD(9614,200,5,131
,74,4,192,192,66,5,129,4,32,
32,8,6,2,22,221,4,96,37,

254)

175 ! Start POKEV routine
```

```
180 CALL LOAD(9636,2,224,37,
20,3,0,0,0,4,192,2,1,0,1,200
,1,37,18,4,32,32,12,4,32,32,
24,18,184)

190 CALL LOAD(9664,200,32,13
1,74,37,0,184,32,131,18,37,1
9,2,3,0,2)

195 ! Start LOOP2 routine

200 CALL LOAD(9680,4,192,192
,67,4,32,32,12,4,32,32,24,18
,184,216,224,131,75,37,0,5,1
31,136,3)

210 CALL LOAD(9704,37,18,22,
242,192,32,37,0,2,1,37,2,192
,131,2,34,255,254,4,32,32,36
)

215 ! Start RETURN routine

220 CALL LOAD(9726,4,192,216
,0,131,124,2,224,131,224,4,9
6,0,112)

225 ! Start POKER routine

230 CALL LOAD(9740,3,0,0,0,4
,192,2,1,0,1,4,32,32,12,200,
32,131,74,37,18,2,1,0,2,4,32
,32,12,4,32)

240 CALL LOAD(9770,32,24,18,
184,192,32,131,74,208,32,37,
19,4,32,32,48,4,91)

250 CALL LOAD(8194,39,04)!Ch
ange First Free Address to e
nd of new routines
```

TEST VDP ROUTINES

The following program uses the VDP routines that you installed with the previous program or the assembly program.

```
100 CALL CLEAR :: FOR I=0 TO
 764 STEP 8 :: CALL LINK("PO
KEV",I,161,162,163,164,165,1
66,167,168):: NEXT I

110 CALL CLEAR :: FOR I=0 TO
 758 STEP 16 :: CALL LINK("P
OKEV",I,161,162,163,164,165,
166,167,168,169,170,172,173,
174,175,176):: NEXT I

120 CALL CLEAR :: FOR I=126
TO 223 :: CALL LINK("POKEV",
396,I,I+1,I+2,I+3,I+4,I+5,I+
6,I+7):: NEXT I

130 DATA 0,0,0,0,0,0,0,255,2
54,130,130,130,130,130,130,2
54,0,124,124,124,124,124,124
,124
140 FOR I=1 TO 3 :: INPUT "P
RESS ENTER TO CHANGE CURSOR"
:A$ :: READ A,B,C,D,E,F,G,H
:: CALL LINK("POKEV",1008,A,
B,C,D,E,F,G,H):: NEXT I

150 DATA 0,0,0,0,0,0,0,255,2
54,130,130,130,130,130,130,2
54,0,0,0,0,0,0,0,0

160 FOR I=1 TO 3 :: INPUT "P
RESS ENTER TO CHANGE EDGE  C
HAR":A$ :: READ A,B,C,D,E,F,
G,H :: CALL LINK("POKEV",101
6,A,B,C,D,E,F,G,H):: NEXT I

170 PRINT : :"SCREEN COLOR C
HANGES" :: FOR T=1 TO 5 :: F
OR I=0 TO 15 :: CALL LINK("P
OKER",7,I):: NEXT I :: NEXT
T :: CALL SCREEN(8)

180 A$="!@#$%^&*()+123456789
0=-:><,.;/~_?'|{}\`ABCDEFGHI
JKLMNOPQRSTUVWXYZabcdefghijk
lmnopqrstuvwxyz" :: FOR I=1
TO 5 :: PRINT A$ :: NEXT I

190 PRINT : :"CHARACTER COLO
R CHANGES" :: FOR I=30 TO 70
 :: FOR T=2063 TO 2076 :: CA
LL LINK("POKEV",T,I):: NEXT
T :: NEXT I :: CALL CHARSET

200 INPUT "PRESS ENTER FOR T
EXT MODE":A$ :: CALL LINK("P
OKER",7,244):: CALL LINK("PO
KER",1,240)
210 INPUT "PRESS ENTER FOR M
ULTI COLOR MODE ":A$ :: CALL
 LINK("POKER",1,232):: FOR I
=1 TO 300 :: NEXT I :: CALL
LINK("POKER",1,224)

220 INPUT "PRESS ENTER FOR B
IT MAP MODE ":A$ :: CALL LIN
K("POKER",0,2):: CALL LINK("
POKER",1,224):: FOR I=1 TO 3
00 :: NEXT I

230 CALL LINK("POKER",0,0)
```

EXTENDED BASIC SYMBOL TABLE STRUCTURE

The starting point of the Symbol table in VDP RAM is pointed to by >833E
The structure for the Symbol table is as follows:

Byte No.	
1	Variable type >00->07 (0-7) = Numeric & Numeric arrays >40->47 (64-71) = Numeric DEF & DEF arrays >80->87 (128-135) = String & String arrays ie: >00 = numeric variable like A or COUNT (not an array) >03 = three element numeric array A(x,x,x) >84 = four element string array S$(x,x,x,x)
2	Length of the variable name. ie: COUNT would equal 5
3-4	Points to next entry in the symbol table. >0000 for last entry.
5-6	Points to the name of this variable in the Symbol table.
7-8	Points to the location in the Numeric value table for type >00 or to the location of the string in VDP RAM for type >80.
9 & up	Start of the variable's name for types >00, >40 and >80

If it is not type >00, >40 or >80 then the following applies for the different types:

Byte No.	
7-8	1st element's DIM if type >01, >41 or >81 variable.
9-10	2nd element's DIM if type >02, >42 or >82 variable.
11-12	3rd element's DIM if type >03, >43 or >83 variable.
13-14	4th element's DIM if type >04, >44 or >84 variable.
15-16	5th element's DIM if type >05, >45 or >85 variable.
17-18	6th element's DIM if type >06, >46 or >86 variable.
19-20	7th element's DIM if type >07, >47 or >87 variable.

After all of the DIM's follows the location in memory for each of the elements of a String array. First comes the locations for each of the first elements, ie: S$(x̂,x,x) then the second elements, S$(x,x̂,x) etc.

If it is a numeric array the word following the DIM's points into the Numeric value table where the array starts. Since each numeric variable is 8 bytes long in Radix 100 notation the first 8 bytes in the Numeric Value table are for A(0,0,0) the next 8 bytes are for A(1,0,0) etc. (With OPTION BASE 0) After all of the location(s) pointers follows the name of the variable, this does not include the brackets and numerals for arrays. S$(1,2,3) would only have S$ in this location and A(1,2,3) would only have A in this loacation.

EXAMPLES with OPTION BASE 0

DIM S$(2,2) {type >82)

Byte No.	
7-8	= >02
9-10	= >02
11-12	= VDP address of S$(0,0)
13-14	= VDP address of S$(1,0)
15-16	= VDP address of S$(2,0)
17-18	= VDP address of S$(0,1)
19-20	= VDP address of S$(1,1)
21-22	= VDP address of S$(2,1)
23-24	= VDP address of S$(0,2)
25-26	= VDP address of S$(1,2)
27-28	= VDP address of S$(2,2)
29-30	= >5324 = S$

DIM A(12,3) {type >02}

Byte No.	
7-8	= >0C
9-10	= >03
11-12	= Start address in Numeric Value table of entire array.
13	= >41 = A

The following program uses CALL LINK ("PEEKV"....) and CALL PEEK to look into the Extended BASIC symbol table and subprogram symbol table. The symbol table contains information on the variables in your program (see previous page). The subprogram symbol table contains information on the variables used in user written subprograms and the built in subprograms. This program will print out to a printer or to the screen the following information: The address in the VDP symbol table that references the variable. The type of variable, how many elements if it is an array type variable, what the length of the variables name is and its name and what the DIM is for each of the elements if it is an array. The subprogram symbol table returns the following info: What the start location is in GROM for the built in subprograms (ie: CALL CLEAR). Which variables are used for parameter passing and which are used in the subprogram for user written subprograms, as well as the the regular info on the type, elements etc.

After you type this program in save it out to disk using the MERGE option, then you can MERGE it back into your own program. Don't forget that you must have the PEEKV and POKEV routines loaded to utilize this program. One last thing, when this info is printed out remember that the variable names that start with @ or [are probably from this program. Have fun.

```
1 CALL CLEAR :: PRINT "OUTPU
T TO PRINTER Y/N N" :: ACCEP
T AT(23,23)SIZE(-1)VALIDATE(
"YN"):@@$ :: IF @@$="Y" THEN
 OPEN #1:"PIO" :: @@=1

2 DIM @$(5),[$(1):: @$(0)="N
umeric" :: @$(1)="Num Array"
 :: @$(3)="Num  DEF" :: @$(4
)="String" :: @$(5)="Str Arr
ay"

3 @@$="---------------------
----------------------------
-------" :: GOSUB 12

4 CALL PEEK(-31942,[1,[2)::
[1=[1*256+[2 :: PRINT #@@:@@
$:"SUBPROGRAM INFORMATION":
:"NAME";TAB(17);"TYPE"

5 CALL LINK("PEEKV",[1,[3,[4
,[5,[6,[7,[8,[9,[10):: FOR @
=0 TO [4-1 :: CALL LINK("PEE
KV",[7*256+[8+@,[2):: @1$=@1
$&CHR$([2):: NEXT @

6 IF [3 THEN PRINT #@@:@@$ :
: PRINT #@@,USING "#########
###### GROM Address #####":@
1$,[9*256+[10 :: @1$="" :: G
OTO 11

7 PRINT #@@:@@$ :: PRINT #@@
,USING "############### User
 Written":@1$ :: @1$="" :: I
F [9=0 THEN 10

8 FOR @=[1+2 TO [1+90 STEP 2
 :: CALL LINK("PEEKV",@+4,[9
,[10,[11,[12):: [1=@ :: @=@-
91*([11=0):: NEXT @

9 PRINT #@@: :"Parameter pas
sing variables": : :: @1=[9*
256+[10 :: GOSUB 13

10 CALL LINK("PEEKV",[1+12,[
11,[12):: @1,[11=[11*256+[12
 :: IF [11 THEN PRINT #@@: :
"Subprogram variables": : ::
 GOSUB 13

11 IF [5+[6 THEN [1=[5*256+[
6 :: GOTO 5 ELSE END

12 CALL PEEK(-31938,@1,@2)::
 @1=@1*256+@2 :: PRINT #@@:"
 ADD   TYPE           EL LEN
VARIABLE NAME   DIM":@@$

13 CALL LINK("PEEKV",@1,@3,@
4,@5,@6,@7,@8):: FOR @=0 TO
@4-1 :: CALL LINK("PEEKV",@7
*256+@8+@,@2):: @1$=@1$&CHR$
(@2):: NEXT @

14 @2=@3+128*(@3>127):: IF [
1 THEN 16

15 FOR @=1 TO @2*-(@2<8 OR @
2>127):: CALL LINK("PEEKV",@
1+4+@*2,@7,@8):: @2$=@2$&STR
$(@7*256+@8)&" " :: NEXT @

16 PRINT #@@,USING "##### ##
# ##########  # ##  ########
####### ####################
#########":@1,@3,@$(INT(@3/3
2)-(@2>0)),@2,@4,@1$,@2$

17 @1$,@2$="" :: IF @5+@6 TH
EN @1=@5*256+@6 :: GOTO 13 E
LSE RETURN
```

5TH 1- =FORTH

We have two different Forth programs for you this month. The first program is to be placed on your Forth System diskette on SCREEN 6. The DISK-INIT is used to initialize, clear and install the error messages on a blank disk that will be used to store your Forth programs on. This program assumes that you have set up your Forth System disk according to last months newsletter (with the BSAVE).

Boot in your TI Forth disk (the BSAVEd version) and type in 6 CLEAR FLUSH. Now type in 6 EDIT and then on SCREEN 6 type in the following program. After the program has been typed in press FCTN 9 (BACK) and type in FLUSH. Now that you have saved this on SCREEN 6 you can easily load it into memory by typing in 6 LOAD. After the program is loaded you can run it by typing in DISK-INIT. When you run this program follow the instructions that come up on the screen. When the program is finished your screen will clear and TI FORTH will appear at the top.

```
SCR #6
  0 ( PROGRAM DISK INITIALIZATION ) BASE->R DECIMAL
  1
  2 : KEY? CR ." and Press any key." KEY DROP ;
  3 : CLEAR-IT DISK_SIZE @ 0 DO I CLEAR LOOP FLUSH ;
  4
  5 : INIT PAGE
  6   ." Insert blank disk in drive 1"
  7   KEY? 0 FORMAT-DISK CLEAR-IT DISK-HEAD ;
  8
  9 : INSTALL-ERRS PAGE
 10   ." Place FORTH System diskette in drive 1" KEY? CR CR
 11   4 BLOCK UPDATE 5 BLOCK UPDATE
 12   ." Place Initialized blank disk in drive 1" KEY? FLUSH ;
 13
 14 : DISK-INIT INIT INSTALL-ERRS PAGE ABORT ;
 15 R->BASE
```

Documentation

line 0 Screen name in brackets, move the old number base to the return stack and change the number base to DECIMAL.

KEY? Generate a carriage return and then print the "and Press any key... " message on the screen. Next look for any key press, DROP its value off the top of the stack and end this word.

CLEAR-IT Take the value stored in DISK_SIZE as the end of the loop and 0 as the start of the loop. Next start a loop (similar to FOR I=0 TO DISK_SIZE) and put the loop counter, I, on the stack. CLEAR the screen number that I left on the stack until the loop is finished. Then FLUSH the buffers out to the disk to CLEAR the last few screens and end this word.

INIT Clear the screen and home the cursor (see the March newsletter for PAGE) and then print the "Insert blank...." message. Next use our KEY? word for the rest of the prompt and input. Afterwards initialize the diskette in drive 1, then do the CLEAR-IT routine and finally write out a disk header that the disk manager can recognize and end this word.

INSTALL-ERRS Clear the screen and home the cursor and then print the "Place FORTH..." message. Next use the KEY? word and after a key is pressed generate 2 carriage returns. Next read in screen 4, mark it as updated, read in screen 5, mark it as updated and then print the "Place Initialized..." message. Next use the KEY? word and after a key is pressed FLUSH (write) the error messages (SCREENs 4 & 5) out to the new disk and end this word.

DISK-INIT This word starts and directs the program flow. First we execute INIT then we execute INSTALL-ERRS next we clear the screen and home the cursor and finally we leave the program and print TI FORTH on the screen.

line 15 This takes the old number base from the return stack and stores it back into BASE.

After you are done using this program you can type in FORGET KEY? and it will be erased from memory.

WARNING...... Make sure you follow the screen prompts and that you have covered the write protect notch back up after this program is installed on your Forth System diskette!!!

Now that you have this program installed on the System diskette lets go ahead and use it to initialize a new diskette and then we will store our next program on it. In the next column you will find an Extended BASIC program listed. This program allows you to type in a name and address and then it will print it back out on the screen in its proper format. After which it changes the screen color a few times and then starts over again. To make the Forth program easier to compare to the XB version we used the line numbers with an L in front as the Forth words. So L100 does in Forth what line number 100 does in Extended BASIC.

```
100 CALL CLEAR :: CALL SCREE
N(6):: FOR I=0 TO 14 :: CALL
 COLOR(I,2,15):: NEXT I

110 CALL CLEAR :: DISPLAY AT
(4,1):"Name    :":"Address:":
"City    :":"State   :":"Zip
  :"

120 ACCEPT AT(4,9)SIZE(20):N
$ :: ACCEPT AT(5,9)SIZE(20):
A$ :: ACCEPT AT(6,9)SIZE(20)
:C$ :: ACCEPT AT(7,9)SIZE(2)
:S$ :: ACCEPT AT(8,9)SIZE(5)
:Z$

130 DISPLAY AT(12,1):N$:A$:C
$&", "&S$&" "&Z$

140 FOR I=1 TO 16 :: CALL SC
REEN(I):: FOR J=1 TO 100 ::
NEXT J :: NEXT I :: CALL SCR
EEN(6):: GOTO 110
```

```
SCR #6
( XB TO FORTH SAMPLE ) BASE->R DECIMAL

: MG ; ( "FORGET MG" WIPES OUT ALL OF THE FOLLOWING NEW WORDS )

: L100 CLS 5 SCREEN 17 3 DO 1 14 I COLOR LOOP EMPTY-BUFFERS ;

: L110 CLS 0 3 GOTOXY ."   Name    :" CR ."   Address:" CR
       ."   City    :" CR ."   State   :" CR ."   Zip     :" ;

: CLRBUF 8210 90 32 FILL ;

: L120 CLRBUF 10 3 GOTOXY 8210 20 EXPECT
  10 4 GOTOXY 8232 20 EXPECT 10 5 GOTOXY 8254 20 EXPECT
  10 6 GOTOXY 8276  2 EXPECT 10 7 GOTOXY 8280  5 EXPECT ;

--> ( = LOAD NEXT SCREEN ALSO )

SCR #7
( XB TO FORTH SAMPLE CONT. )

: L130 2 12 GOTOXY 8210 22 -TRAILING 2- TYPE
       2 13 GOTOXY 8232 22 -TRAILING 2- TYPE
       2 14 GOTOXY 8254 22 -TRAILING 2- TYPE ." , "
                   8276  4 -TRAILING 2- TYPE ."  "
                   8280  7 -TRAILING 2- TYPE CR CR ;

: QUIT? ?TERMINAL IF TEXT ABORT ENDIF ; ( FCTN 4 STOPS PROGRAM )

: L140 15 0 DO I SCREEN 100 0 DO QUIT? LOOP LOOP 5 SCREEN ;

: LOOPIT L110 L120 L130 L140 MYSELF ;

: RUN GRAPHICS L100 LOOPIT ;
R->BASE                RUN ( START PROGRAM AFTER LOADING )
```

This is just one example of how an INPUT or ACCEPT AT statement is handled in Forth. Since Forth is so versitile there are many ohter ways to accomplish the same effect. This program is not truly useful since it does not store the names and addresses out to disk, but with a few modifications you could easily accomplish this. Lets move on to the documentation for this program and how to save and load it for future referance.

After you have used DISK-INIT to format a blank disk for Forth program storage remove your Forth System diskette from drive one and place this initialized diskette in drive one. Next type in 6 EDIT and then while you are in the edit mode type in SCREEN 6 as it appears on the left. After you have completed SCREEN 6 press FCTN 4 (ROLL UP) to edit SCREEN 7. Then type in SCREEN 7 as it appears on the left. After both screens are typed in press FCTN 9 (BACK) and type in FLUSH. To test it out type in 6 LOAD and your program should load, compile and run. If you run into a problem or you just want to reload the screens just type in FORGET MG and the programs words will be erased from the vocabulary so you won't get the "isn't unique" message.

You can stop the running program by pressing FCTN 4 (CLEAR) when the program is changing the screen colors after the input prompts. After you have stopped the program you can restart it by typing in RUN since that is the word we defined in the program as the start word.

The L100, L110, L120, L130 and L140 words accomplish the same thing as the corresponding line numbers in the Extended Basic version.

The EMPTY-BUFFERS at the end of L100 clears out the space we will be using for our string storage. It also clears the UPDATEd flag if it was set so that we don't accidentally write anything out to the disk. The EMPTY-BUFFERS word, however, places the null character >00 throughout the disk buffer area in low memory. The problem with this occurs when you want to print your strings out on the screen they ended with a bunch of white squares (char 0) so we created another word. The CLRBUF word writes char 32, the space character, out to the area of memory that we are using for our temporary string storage.

The QUIT? word uses ?TERMINAL to check for the FCTN 4 key press. If it was pressed we will leave the GRAPHICS mode and go into the TEXT mode. Then the ABORT word exits our program and places the TI FORTH message on the screen.

Now lets follow this program from the RUN word. First we enter the GRAPHICS mode (32 column display like XB).

Next we execute the L100 word. This word clears the screen, changes the screen color to Lt Blue, sets the colors for the characters as Black on Grey and empties the disk buffer space. In Forth you have the entire 255 ASCII character set to your avail and as such character set 0 is for chars 0 through 7, set 1 is for 8 through 15, set 2 is for 16 through 23, set 3 is for 24 through 31, etc. That is why we have our DO LOOP set up for sets 3 through 17. Next we execute the word LOOPIT.

LOOPIT is the main word for executing this program in that it directs the flow for the other words. First it executes L110 which clears the screen, places the cursor at row 4 column 1 and prints " Name :". The CR word moves the cursor down one line and then we print the next prompt etc. After all the prompts are printed we leave this word and continue execution with L120. If you think of Forth words as subprograms in that after the word is executed it returns to the next statement following it, you might find it easier to trace through a Forth program. for example think of LOOPIT as being the following line in Extended BASIC- 50 GOSUB 110 :: GOSUB 120 :: GOSUB 130 :: GOSUB 140 :: GOTO 50 and imagine that each of these lines has a RETURN at the end of them.

L120 is our ACCEPT AT line but first it will use CLRBUF to clear out our buffer area and then it starts looking for inputs. The 10 3 GOTOXY positions the cursor at row 4 column 11 for the NAME : input prompt. Adress 8210 is the first section in our buffer area and 20 is the maximum number of characters to save. After 20 characters have been typed in, or the ENTER key is pressed, the 20 or less characters that you typed in will be stored at address 8210 (decimal). After the first input is complete the program continues with the next GOTOXY until it has reached the end of the L120 word (;)

After all of the inputs are filled in the program continues execution with L130. This is the word that prints out all of the information that you typed in. The word EXPECT from the previous line places two nulls (>00 >00) at the end of the input line as an end of line marker so we must remove them before we print the line out on the screen. So if we typed in 20 characters our line will be 22 characters long, if we typed in 14 characters our line will be 16 characters long etc. The -TRAILING word strips the extra space characters off the end of the line, if we did not fill it out, and 2- strips the two nulls off the end of our input. TYPE will then print out on the screen just or input without the nulls and extra space characters. This may not have been the most efficient way to input, store and retrieve and print out these strings but we had to start somewhere. We could have used user variables or stored them right in the dictionary, but we will get to that in a future issue.

When L130 is complete the program will continue execution with L140. This word sets up a DO LOOP to change the screen color and the uses a nested DO LOOP with the word QUIT? in it as a time delay between color changes. This is why you can stop the program while the screen colors are being changed. After the LOOPs are completed we will change the screen color back to Lt. Blue and leave this word.

The last word that LOOPIT executes is the word MYSELF. This word allows Forth to be recursive, in that any word can execute itself. So when Forth comes to MYSELF it will start LOOPIT all over again.

As you can see its not to difficult to translate programs from BASIC or Extended BASIC into Forth. A lot of the words are similar except that Forth usually has the column before the row such as COL ROW GOTOXY instead of DISPLAY AT(row,col). Also the loops are slightly different in that the starting number and ending number are swapped and the ending number is 1 unit greater such as: FOR I=1 to 10 :: NEXT I in XB is like this in Forth: 11 1 DO LOOP. The colors, and sprite numbers are also off by 1 digit. Gray in XB is 15 and in Forth its 14 so the colors in Forth run from 0 through 15 and the sprite numbers run from 0 through 31. In Forth you also have to add 3 to the XB character set value to arrive at the same char numbers. And lastly don't forget that the screen in Forth starts with row 0 and column 0.

I hope that this short program example and documentation will help you in writing and running your own Forth programs. Next month we will have more examples for you and until then keep experimenting, that's how I learned.

PC NOTES

This month in the PC column I would like to talk about the machine a little bit. First off I'm glad to see that TI is putting approx 9 million dollars into an ad campaign that compares the TI PC to the IBM PC. The TI is a little faster than the IBM, it has MUCH better resolution on the monitor than the IBM and the keyboard is a joy to use!!! Why buy an IBM? 1. Because you like Charlie Chaplin. 2. Because you like the initials IBM. 3. Because you like to pay more for less. 4. Because you're mad at TI for the home computer pull out.

Seriously, don't rush out and buy an IBM because of item 4 above. That is a fact of life in the computer industry. If you are in the market for a machine of the PC caliber first you should compare keyboards AND resolution on the monitor (including number of colors in hi-res mode). Next look at the software availability. Just recently TI released MS-DOS 2.1 for the PC and on this disk is a program entitled EMULATE. When you type in this command the system installs the proper values into RAM that will allow the TI PC to run many of the IBM programs. Now I'm not going to say that you can run all of the IBM programs, but you can certainly run a lot more of them then before. There is a saying out there that states "The ONLY computer that is truly IBM compatible is an IBM" (Peanut excluded of course). That is because the only computer that uses IBM PC DOS is IBM and they hold the rights to it and their ROM code that makes it compatible. However the TI PC with MS-DOS 2.1 can run many of the IBM programs provided they DO NOT contain graphics. TI graphics are MUCH better than IBM's and as such they are not compatible. Most of the better programs such as LOTUS 1-2-3 that use graphics have been written for the TI PC. If you are planning on buying a PC type computer I'd recommend taking a look at the TI PC, I've been pleased with it.

This months PC program is a random pattern generator that uses the box and box filled commands of MS DOS Basic. You must have the three plans graphics board in order to run this program. While the program is running you can press the RETURN key to pause. Then if you want to continue with the same pattern just press C, if you want to start a new pattern press any other key. Also while the program is running pressing any key other than the RETURN will start a new pattern. After a random number of boxes have been generated, as determined by line 120, the program will automatically clear the screen and start a new pattern. I hope you enjoy it.

One last thing, We've received a few letters requesting that we use this space for more 99/4A info and drop the PC column. So, this may be the last PC column unless we hear differently from you, please let us know, We have a lot more info and programming that we would like to pass on about the TI PC.

```
 10 'BOXES1 BY CRAIG MILLER 1-1-84
 20 CLS: PALETTE:KEY OFF : DEFINT A-Z :P=2 :COLOR 4,0:LOCATE ,,0:PRINT
 30 RANDOMIZE VAL(RIGHT$(TIME$,2))*VAL(MID$(TIME$,4,2)):GOTO 170
 40 LINE (X+X+360,Y+150)-(C+C+360,R+150),P+4,BF
 50 LINE (X+X+360,Y+150)-(C+C+360,R+150),P,B
 60 LINE (Y+Y+360,X+150)-(R+R+360,C+150),P,B: RETURN
 70 X=(X+E) MOD 149: Y=(Y+F) MOD 149
 80 C=(C+G) MOD 149: R=(R+H) MOD 149
 90 Y=-Y:R=-R:GOSUB 40 : X=-X:C=-C:GOSUB 40
100 Y=-Y:R=-R:GOSUB 40 : X=-X:C=-C:GOSUB 40
110 IF RND*A<1 THEN GOSUB 140
120 IF RND*150<2 THEN 170
130 I$=INKEY$:IF I$="" THEN 70 ELSE IF I$=CHR$(13) THEN 200 ELSE 30
140 E=RND*12-6: F=RND*12-6
150 G=RND*12-6: H=RND*12-6
160 P=RND*7:A=8 : RETURN
170 X=RND*129-65 : Y=RND*129-65
180 C=RND*129-65 : R=RND*129-65
190 A=4:CLS :GOTO 90
200 I$=INKEY$:IF I$="" OR I$=CHR$(13) THEN 200 ELSE IF I$="C" THEN 70 ELSE 30
```

S U B S C R I P T I O N   I N F O R M A T I O N

THE SMART PROGRAMMER - a monthly 16+ page newsletter published by **MILLERS GRAPHICS**
U.S. 12.50 year - Foreign Surface Mail 16.00 year - Foreign Air Mail 26.00 year

To subscribe send a Check, Money Order or Cashiers Check, payable in U.S. currency

TO: **MILLERS GRAPHICS**
1475 W. Cypress Ave.
San Dimas, CA 91773

MG

MILLERS GRAPHICS
1475 W. Cypress Ave.
San Dimas, CA 91773

BULK RATE
U.S. POSTAGE
PAID
San Dimas, CA 91773
PERMIT NO. 191

THE SMART PROGRAMMER