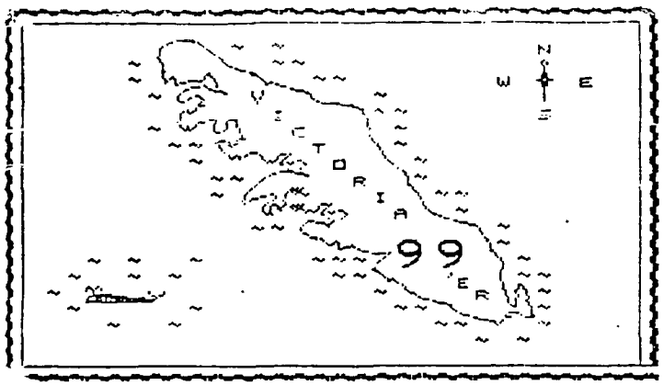

AUGUST 1984

PROGRAMING WITH BASIC AND X-BASIC :

In order to write programs which you can modify or adapt easily months or years after they are written you need to make a deliberate effort to make them readable.

Most of these techniques take a little more memory than is needed to do the program without this documentation. If space becomes a problem later in the program the documentation can easily be reduced of eliminated.

When nameing varaibles use names which explain what the variable is or is used for. For example names like "XP","YP","OXP" or "RC" may be clear when you are writing the program, but will probably be unclear a few months down the road , or to someone else trying to understand your program. Names like "XPOSITION", "YPOSITION","OLDXPOSITION" and "ROCKETCOLOR" are self explanatory.

Be sure to use comments in your program either a REM statement or with X-BASIC the ! comment. Using lower case letters in the comments make them stand out when the program is listed.

Try to never use GOTO statements. If you must use one try and make it to a line that can be listed at the same time and try to make the purpose of the GOTO self explanatory. For example :

```
100 CALL KEY(0,KEY,STATUS)
110 IF STATUS <> 0 THEN 130
120 GOTO 100
130 STOP
```

That was just an example. You would really use something like :

```
100 CALL KEY(0,KEY,STATUS)
110 IF STATUS = 0 THEN 100
120 STOP
```

A good technique to use is a list of commented subroutines.

```
100 REM Chess program
110 GOSUB 200 ! Instructions.
120 GOUSB 300 ! Game type selection.
130 GOSUB 400 ! Player data entry.
140 GOSUB 500 ! Make screen.
150 GOSUB 600 ! Start of play.
160 STOP ! Game over.
199 REM Instructions.
200 CALL CLEAR
210 PRINT "DO YOU WANT INSTRUCTIONS Y or N "
220 INPUT : REPLY$
230 IF REPLY$="N" THEN RETURN
240 PRINT "INSTRUCTIONS ARE IN DEVELOPMENT AND UNAVAIBLE AT THIS TIME."
250 RETURN
```

If you have X-BASIC and need a routine with no or only a few parameters, and the routine is to be called from many places in the program, try using a SUB program.

For example in a program that uses thebottom two lines of the screen for comments and prompts, with the remainder of the screen holding graphics; it becomes necessary to clar the bottom two lines of the the screen frequently. You could put in a line entry everytime with CALL HCHAR(23,1,32,64) or use the sub program CALL CLRCOMENT .

```
10000 SUB CLRCOMENT :: CALL HCHAR(23,1,32,64) :: SUBEND
```

Remember when using sub programs that all variables in the main program that are needed must be passed in to the sub program and take up additional space in memory as they become new variables in the sub program. So if variables are shared by a number of program segments it is better to use commented sub routines than sub programs.

Jim Bisakowski has TI 99/4A,s with 3 months of use for $65.00
Phone 479-6500

CODE BREAKER PROGRAM :

```
100 REM      MASTERMIND          X-BASIC AUG 10,84          BY TONY BIGRAS
110 RANDOMIZE
120 FOR I=1 TO 5 :: NUMB(I)=INT(RND*6+1):: NEXT I
130 CALL CLEAR
140 FOR I=1 TO 20 ! Start of guessing loop
150 ACCEPT AT(I,4)SIZE(5)VALIDATE("123456"):GUESS$
160 IF LEN(GUESS$)<5 THEN 150
170 FOR J=1 TO 5 :: GUESS(J)=VAL(SEG$(GUESS$,J,1)):: NEXT J
180 FOR J=1 TO 5
190 IF GUESS(J)=NUMB(J)THEN CORPOS=CORPOS+1
200 NEXT J
210 FOR J=1 TO CORPOS :: DISPLAY AT(I,15+J):"*" :: NEXT J
220 IF CORPOS=5 THEN I=21 ELSE 240 ! If match is complete make some noise else c
ontinue up to twenty trys
230 FOR J=1 TO 4 :: FOR II=1 TO 30 :: CALL SOUND(10,110*J,II):: NEXT II :: NEXT
J :: CALL SOUND(1000,1000,1)
240 CORPOS=0
250 NEXT I
260 RUN !Game over start again.
```

****

DIMENSIONED ARRAYS :

An array is a group of variavles all with a name and a number.
EXAMPLE WEEKDAY$(1) WEEKDAY$(2) WEEKDAY$(3)
The number in brackets is called the element number and can also be a variable. For example if DAY=3 you could go WEEKDAY$(DAY) which is the same as WEEKDAY$(3) In this example WEEKDAY$(1)="SUNDAY" WEEKDAY$(2)="MONDAY" WEEKDAY$(3)="TUESDAY" etc.

Lets assume you need to keep track of the number of eggs that are layed every hour on your chicken farm. We will use a 2 dimenson array.
DIM EGGPRODUCTION(7,24) for 7 days with 24 hours in each day. On manday between 7 and 8 am 100 eggs are layed therefore EGGPRODUCTION(2,8)=100 .

If you needed to know what your production was every minute you would use a 3 dimension array DIM EGGPRODUCTION(7,24,60) for days hours minutes. This array has 10080 elements 7*24*60=10080 and would need 86,400 that is 85K of memory. If you owned a chicken ranch that had a per minute egg output worth keeping track of you would probably own another computer with enough memory capicity> The TI/994A uses 8 bytes to hold each number normally. Even with data compression techniques it would be difficult to have one weeks production in the memory.

Arrays can take up lots of memory and cannot be removed by the program when running , so use use them with care.

****

DATA STATEMENTS :

Data statements are often used when lots of information needs to be used in the same fashion. To define custom characters for example. In that case the data often cantains alternating numbers and strings.

```
100 DATA 32,"0000000000000000",33,"FFFFFFFFFFFFFFFF"
```

This would be used with a loop to read the data and to define the characters.

```
200 FOR I=1 TO 2
210 READ CHARACTER,CHARACTER$
220 CALL CHAR(CHARACTER,CHARACTER$)
230 NEXT I
```

Data statements are read in sequence of line numbers. For instance if there was data at lines 100 110 120 3000 3010 it would start at 100 read all the data on that line then go onto 110 , then 120 , then 3000 , then 3010 .
You can restart the sequence from any line number with the RESTORE statement. You could read all the data at line 100 go RESTORE 100 and start reading data from the begining of line 100 again.

Say you wanted to display 3 different screens at different times in your program. You would write one routine for displaying the strings that make up the screen, write 3 seperate groups of DATA statements and when you wanted to display a screen you would restore its data then GOSUB to your routine to read the data and make the screen.

****

FORTH BUGS :

Another bug in TIFORTH. When you are in a loop using the joysticks you cannot get out of the loop with ?TERMINAL . Normally ?TERMINAL will check for a press of the break key and produce a true flag if pressed else a false flag. When the JOYST procedure is in use ?TERMINAL will not work. You need to write your own high level version of ?TERMINAL . Here is a version that will work with JOYST .

```
: ?TERMINAL   KEY 2 = IF 1 ELSE 0 THEN ;
```

****

TO SUBMIT MATERIAL FOR THIS NEWS LETTER CALL TONY @ 383-3946