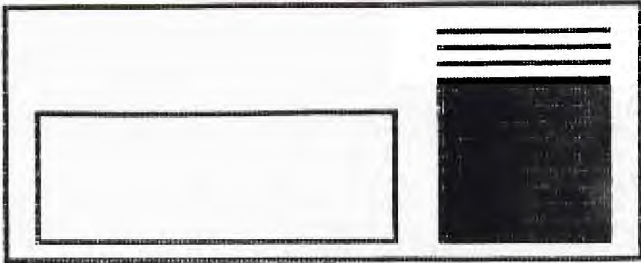


(179)
8706



THE
DATA
BUS

ISSUE :
VOL. 5
NO. 5
- - - -
J U N E
1 9 8 7

THE DELAWARE VALLEY USERS GROUP
DEDICATED TO THE TI AND COMPATIBLE HOME COMPUTER FAMILY

P.O. BOX 6240 STANTON BRANCH, WILMINGTON DE 19804

Minutes of the Delmarva Chapter - 8 June 1987

The Treasurer's Report was read by Chapter President, Chuck Bower.

Jim England reported on his brief experience with the AVATEX 1200 Modem. The unit is almost totally compatible with Hayes.

The Chapter is still in need of a Printer and an RS232 Card to complete rigging of the equipment package maintained at the Courthouse for meetings.

The dues for the Delmarva Chapter IIBBS are now due.

Jim England advised those present that it is necessary that the Chapter reimburse Chuck Bower for the Equipment he provided to get the Bulletin Board operating. Comment was made by Chuck that the PE Box would remain on loan since the Chapter Treasury could not afford same, at this time.

Since the SYSOPS for our BBS provides the manpower, expertise and utilities to keep it operational, it does not seem logical that he be asked to pay dues for access to same. Motion was made, seconded and carried to exclude the SYSOPS from IIBBS dues.

Dem Dawson reported that the Chairman of the South Jersey Chapter is reportedly a highly skilled TI Technician who can do extensive mechanical repairs of member's equipment.

Kay and Gil Quillen donated a MULTIPLAN to the Chapter for a possible future raffle.

Jim England demonstrated the Paragon Computing's Enhanced II Extended Basic.

Afterwards, Jim then continued instructions on II BASIC Programming, concentrating on ASCII Codes.

The next meeting will be held on Monday, 13 July 1987.

Design Your Own Cursor
by J. F. Willforth - West Penn 99ers

I can't give proper credit to whoever originated this program to create a TEXAS cursor, but my goal is not for you to be the proud user of a TEXAS type cursor, but rather you know how to create your own CUSTOM CURSOR!

Whatever program that you use, assembly, or extended basic, you will have to encode the design for your CUSTOM CURSOR. The program will be the vehicle for your own cursor.

```

1 !TEXAS CURSOR from GOTO Ne
wsletter of Columbus, Ga. Us
ers Group; unattributed, but
Jim Peterson's and Dr. Ron A
lbright's names came up.
2 CALL CLEAR :: CALL INIT
3 CALL LOAD(8196,63,248) ! R
EF table pointer at >2004 (3
F,FB)
4 CALL LOAD(16376,67,85,82,8
3,79,82,48,8) ! Indicates th
at a program named "CURSOR"
begins at >3008
5 CALL LOAD(12288,48,48,63,2
55,254,124,24,12) ! THIS IS
WHERE WE START THE CUSTOM CU
RSOR DESIGN
6 CALL LOAD(12296,2,0,3,240,
2,1,48,0,2,2,0,8,4,32,32,36,
4,91)
7 CALL LINK("CURSOR") ! Link
s to the cursor program.
```

If you are interested in creating your own cursor, please read the rest of this page, and I'll show you how to chart out this TEXAS CURSOR, and how to create your very own, let's say one with your initials, or a square box. The creation is very much the same as charting a sprite in extended basic, but instead of using HEX, you will be using straight BINARY.

```

BINARY WEIGHT !128! 64! 32! 16! 8 ! 4 ! 2 ! 1 !
-----
ROW # 1 - 48 | | | X | X | | | | |
-----
ROW # 2 - 48 | | | X | X | | | | |
-----
ROW # 3 - 63 | | | X | X | X | X | X | X |
-----
ROW # 4 - 255 | X | X | X | X | X | X | X | X |
-----
ROW # 5 - 254 | X | X | X | X | X | X | X | |
-----
ROW # 6 - 124 | | X | X | X | X | X | | |
-----
ROW # 7 - 24 | | | | X | X | | | |
-----
ROW # 8 - 12 | | | | | X | X | | |
```

! Look at line "5" in the above program

CONTINUED ON PG 10

PAGE 2 - DELAWARE VALLEY USERS GROUP

DUUG EXECUTIVE COMMITTEE MEMBERS IN 1987

PRESIDENTTOM AUGUST
VICE PRESIDENTJIM DAVIS
SECRETARYTIM EVERS
TREASURERTOM KLEIN
SGT. AT ARMSJIM FOLZ
DELMARVA CHAPTER CHRCHARLES BOWER
SO. JERSEY CHAPTER CHRTONY DIFEBBO
SHORE CHAPTER CHRHARVEY ADAMS

NORMAL MEETING SCHEDULE

CHRISTIANA GROUP 4th Thursday 6:30-9:30
DELMARVA CHAPTER 2nd Monday 7:00-9:00
SOUTH JERSEY CHAPTER 3rd Monday 6:45-9:00
SHORE CHAPTER 1st Thursday 7:30-9:00

MEETING PLACES

CHRISTIANA GROUP: Delaware's Christiana Mall on Rte. 7, at I-95 Exit 4-S. We meet in the Community Room. Enter between J. C. Penney and Liberty Travel inside the Mall.

DELMARVA CHAPTER: Kent County Courthouse, Basement Conference Rm #25, Green & State Sts, Dover, De. Use the Green St. side entrance.

SOUTH JERSEY CHAPTER: Deptford Municipal Bldg, Cooper Ave. and Delsea Drive, (Rtes. 534 & 47), in Gloucester County. Enter and park in rear of the building.

SHORE CHAPTER: Scullville Firehouse #1, County Rte. 559 (on left, between mile markers 4 and 3), in Atlantic County. Ignore Station #2 on right enroute.

DUUG BULLETIN BOARDS

(302)322-3999 Anytime
(609)429-7792 Monday-Thursday 3:00 PM-7:00 AM
Friday 3:00 PM-Monday 7:00 AM
(302)674-1449 6:00 PM-6:00 AM

For general information, you may contact

TOM KLEIN Pa. (215)494-1372
JIM FOLZ Del. (302)995-6848
BUTCH FISHER N.J. (609)783-8276
GUS LEWIS N.J. (609)927-5601

Delaware Valley Users Group membership includes: library and software privileges, monthly DATABUS newsletter, plus other special benefits. Annual membership rates are: Family or Individual \$15; Student \$10; Newsletter only (beyond 75 mi) \$10.

TRANSMIT YOUR NEWSLETTER COPY TO: The Data Bus Editor --- Jim Folz, Telephone (302)995-6848, or use the DUUG mailing address shown on Page One. PLEASE SUBMIT NEWSLETTER ARTICLES FOR AN ISSUE BEFORE THE 2ND THURSDAY OF EACH MONTH.

An article appearing in The Data Bus may be reproduced for publication by another II Users Group as long as acknowledgement is given to the sources as indicated. We encourage exchange newsletters; mail to DUUG business address shown on Page One.

DUUG ADVERTISING RATES FOR THE DATA BUS:

1/4 page - \$ 5/issue, or \$ 45/12 issues
1/2 page - \$ 8/issue, or \$ 75/12 issues
Full page - \$15/issue, or \$125/12 issues

NOISE on The Data Bus
by Jim Folz

The Executive Board is considering a picnic/computerfest. No site has been chosen yet, but a place central to the four chapters is being sought. The intent is to find a place with indoor facilities (to protect the computers) and some outdoor facilities for those who may not be interested in computers (whoever that is). Flea markets, demos, etc. are planned. Each family would be responsible for their food arrangements. A representative from each group will make up the planning committee. If you would like to help plan the event, contact your chapter officers. A date in mid-September has been suggested.

Don't forget our contest for the best 5-line program. The July deadline will be here before you know it. Talk it up! This is a great way to learn/sharpen programming skills.

Get your raffle tickets for the Rave keyboard. The winner will probably be chosen at the June meeting.

At the June Christiana meeting, Bill McLean will discuss Multiplan. A demo of Fortran 99 is also planned.

At the July South Jersey meeting, a Super Multicart demo is planned. Modifications to the TI console (faster crystal/faster RAM) will be discussed. Barry Traver is scheduled to appear to show his diskazine. A Turbo Pascal demo is also possible. The Super Multicart demo is also planned for the July Christiana meeting.

COMPUTER FAIR SCHEDULE:

07/19/87 10 A.M.-4 P.M. Holiday Inn, Cherry Hill Rt. 70 and Sayre Avenue
08/02/87 10 A.M.-4 P.M. Holiday Inn, Suffern Exit 14B-New York Thruway
08/09/87 10 A.M.-4 P.M. Armory Place, Silver Spring 925 Wayne Avenue

CONTENTS OF THE JUNE ISSUE OF THE DATA BUS:

Minutes - Delmarva Chapter Page 1
Design Your Own Cursor Page 1
NOISE on The Data Bus Page 2
BASIC/XBASIC Programming Techniques Page 3
Progs That Write Progs - Parts 3,4,5 Pages 5,7,8
Sprites/Worksheet Pages 5-6
Sprites - Part 2 Page 7
Bell Compatibility Page 8
Debugging Pages 8-9
Slashed Zero Page 10

DELAWARE VALLEY USERS GROUP - PAGE 3

BASIC/XBASIC PROGRAMMING TECHNIQUES 72 H FF88888883888888 99 c FFFFFFFC3BF8F8FC3
by Jack Shattuck

REVERSE VIDEO FOR HIGHLIGHTS ON SCREEN

One technique used with another well-known computer to emphasize certain text on screen is the visual effect of reverse video, whereby your normal dark printing on a light background (that is, normal as printed on paper - unlike DM 1000, TI-Writer, Multiplan, etc.) is reversed to give a varied enhanced graphic display.

One practical use is during debugging, when you'll want to check typing: zero vs. letter "o" for instance.

I think Patrick Parrish once may have given the reverse char codes in a Compute! article in the distant past, but if so, I can't find it and it wasn't reprinted in any of the bound Compute! collections. Those of you using Tom Freeman's fairware Easy Sprite could try entering (one by one) the standard char codes and then modify for a negative output. You'll see the effect during Easy Sprite instantly. Listings below have been obtained from that source, so I'm sending Tom a fairware check for this column - it demonstrates the clear value of his Easy Sprite to me.

Using XBASIC, the normative character code listing can be obtained using this statement:

```
FOR N=32 TO 126::CALL CHARPAT(N,CS)::PRINT N;CHR S(N);" ";CS::NEXT N
```

CALL CHAR, PRINT or DISPLAY bring those to the screen. To reassign values, a complete list of reverse video char codes for the TI-99/4A:

32	FFFFFFFFFFFFFFFF	52	4	FFF7E7D7B7B3F7F7
33	FFEFEFEFEFEFEFEF	53	5	FF83BF87BF8B88C7
34	FFD7D7D7FFFFFFFF	54	6	FFE7DF87B7B888C7
35	FFD7D783D783D7D7	55	7	FF83BF87EFD0F0DF
36	FFC7ABAFC7E8ABC7	56	8	FFC7B888C78888C7
37	FF9F9BF7EFD83F3	57	9	FFC7B888C73BF87CF
38	FFDFAFAFDFA887CB	58	:	FFFFCFCFFCFCFF
39	FFF7F7EFFFFFFFFF	59	;	FFFFCFCFFCFCFF
40	FFF7EFD0F0FEFF7	60	<	FFF7EFD0F0FEFF7
41	FFD0FEFF7F7FEFDF	61	=	FFFFFFB3FB3FFFF
42	FFFFD7EFB3EFD7FF	62	>	FFD0FEFF7F87EFD0F
43	FFF7E7E7E7E7E7E7	63	?	FFC7B888C7E7E7E7
44	FFFFFFFFFCFEFDF	64	@	FFC7B888C7A8A3BFC7
45	FFFFFFFFB3FFFFFF	65	A	FFC7B88888888888
46	FFFFFFFFFCFCF	66	B	FFB7D8D8C7D8D8B7
47	FFFFFBF7EFD8BFFF	67	C	FFC7B888C7B8B8C7
48	FFC7B888888888C7	68	D	FFB7D8D8D8D8D8B7
49	FFEFCFEFEFEFEFC7	69	E	FFB3BF87B87B8F83
50	FFC7B888C7EFD8B3	70	F	FF83BF87B87B8F83
51	FFC7B888C7E7B88C7	71	G	FFC3BF87A3B888C7

73	I	FFC7E7E7E7E7E7C7	100	d	FFFFFF87D8D8D8B7
74	J	FFB8FB87B87B88C7	101	e	FFFFFFB387B87B83
75	K	FF88B7AF9FAF87B8	102	f	FFFFFFB387B87B8F
76	L	FF8FB87B87B87B83	103	g	FFFFFFC387A3B8C7
77	M	FF8893A8A8888888	104	h	FFFFFF8888838888
78	N	FF88988A8838388	105	i	FFFFFFC7E7E7E7C7
79	O	FF83888888888883	106	j	FFFFFF7F7F7B7CF
80	P	FFB788887B87B8F	107	k	FFFFFFD8D7CFD7DB
81	Q	FFC788888A887CB	108	l	FFFFFF87B87B8F83
82	R	FFB788887AF87B8	109	m	FFFFFF8893A88888
83	S	FFC7888C7F888C7	110	n	FFFFFF8898A88388
84	T	FF83E7E7E7E7E7E7	111	o	FFFFFF8388888883
85	U	FF888888888888C7	112	p	FFFFFF878887B8F
86	V	FF88888D7D7E7E7	113	q	FFFFFFC788A887CB
87	W	FF88888A8A8A8D7	114	r	FFFFFF878887B8B8
88	X	FF8888D7EFD78888	115	s	FFFFFFC387C7F8B7
89	Y	FF8888D7E7E7E7E7	116	t	FFFFFF83E7E7E7E7
90	Z	FF8387E7E7E7E7E7	117	u	FFFFFF88888888C7
91	[FFC7D0F0F0F0F0C7	118	v	FFFFFF8888D7D7E7
92	\	FFFFBFD0FEFF7F8FF	119	w	FFFFFF8888A8A8D7
93]	FFC7F7F7F7F7F7C7	120	x	FFFFFF88D7EFD7B8
94	^	FFFFEFD788FFFFFF	121	y	FFFFFF88D7E7E7E7
95	_	FFFFFFFFFFFFFFF83	122	z	FFFFFF83F7EFD83
96	`	FFFFD0FEFF7FFFFFF	123	{	FFE7D0F0F0F0FE7
97	a	FFFFFFC788838888	124		FFEFEFEFFFEFEFEF
98	b	FFFFFF87D8C7D887	125	}	FFCFF7F7B87F7CF
		and last but not least:	126	~	FFFD0F87FFFFFF

In BASIC, assign the reverse video alphabet to characters 129-154 which can print out by use of CTRL A - CTRL Z. In XBASIC, chars 144-159 are reserved for Sprites, so be selective. Try this:

```
1 CALL TURNPAGE
1000 SUB TURNPAGE
1010 DATA FFFFFFFFFFFFFFFF,FF87B88887B87B8F,FFB7
BB8887AF87B8,FFB387B87B8F83,FFC7888C7F888C7
1020 DATA FFC7888883888888,FF88988A8838388,FFB8
88D7E7E7E7E7,FF88B7AF9FAF87B8,FF83E7E7E7E7E7E7
1030 DATA FF83888888888883,FFC78888C7B8B8C7,FFC7
E7E7E7E7E7C7,FF888888888888C7
1040 CALL COLOR(13,2,16)::CALL COLOR(14,2,16)::R
ESTORE 1010::FOR N=128 TO 141::READ R$
1050 CALL CHAR(N,R$)::NEXT N
1070 !CALL KEY(O,K,S)::IF S=0 THEN 1070
1080 SUBEND
```

RUN it; delete the ! in line 1070 and add: 1060 DISPLAY AT(24,1):",ABCDD,EFG,HCG,IJ,KJ FILMC," holding CTRL for items in quotes. RUN!

EXPLORE YOUR COMPUTER'S DEPTHS LINK UP TO

HOME NETWORK

In the past if you wanted a library of educational software you had but one choice - spend hundreds of dollars.

But now there's a NEW ALTERNATIVE -
HOME NETWORK.

Home Network is your gateway to thousands of hours of educational courseware, as well as games, electronic mail and bulletin boards.

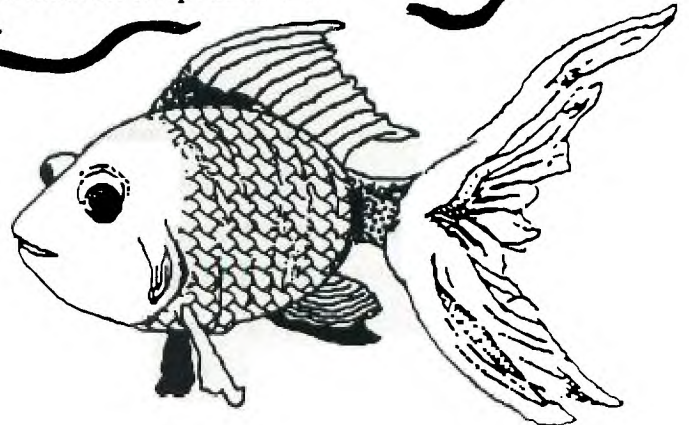
With Home Network educational courseware, you can study astronomy tonight and French tomorrow.

Via Home Network's electronic communication features you can send messages to fellow members or engage in lively bulletin board conversations on topics ranging from microcomputers to current events.

With Home Network games, you can match wits with the computer or play against other subscribers.

The Home Network is part of the University of Delaware's PLATO system which was previously available only to students. Now through the availability of microcomputers this system can be yours for a fraction of the cost of similar services.

Interested? Call (302) 451-8161 and ask to speak to the Home Network Representative.



DELAWARE VALLEY USERS GROUP - PAGE 5

Programs That Write Programs - Part 3
by Jim Peterson

Let's start learning how to actually write a program that writes a program.

A MERGED program is a D/V 163 file, so -

```
OPEN #1:"DSK1.(filename),VARIABLE 163,OUTPUT
```

Every program line begins with a line number, of course. In MERGE format the line number, whether 1 or 32767, is squished into two characters. We don't need to get into how this is done, but you can accomplish it with CHR\$(INT(LN/256))&CHR\$(LN-256*INT(LN/256)), where LN has been predefined as the line number.

To print a statement or command, anything that is represented by a token in the token list, just print the CHR\$(of its token ASCII. For instance, the token for DATA is 147, so you would print CHR\$(147).

To print a variable name, either numeric or string, just enclose it in quotes, "A" or "AS".

To print a value, or a string which is not in quotation marks (such as in a DATA statement), or the word which follows a CALL, you must print CHR\$(200) followed by a token giving the number of characters to follow, such as CHR\$(5) for a 5-letter word such as CLEAR, then the value in quotes. For instance, the token for CALL is 157, so CALL CLEAR is CHR\$(157)&CHR\$(200)&CHR\$(5)&"CLEAR".

Similarly, tokens for parentheses are 183 and 182, so the variable name A(1) is "A"&CHR\$(183)&CHR\$(200)&CHR\$(1)&"1"&CHR\$(182).

A quoted string is handled in the same way except that it is preceded by token 159, so PRINT "HELLO" is CHR\$(156)&CHR\$(199)&CHR\$(5)&"HELLO". Don't worry about the quotation marks, the computer will handle that.

If you need to refer to a line number, as in GOTO 500, use token 201 followed by the line number formula, thus CHR\$(134)&CHR\$(201)&CHR\$(INT(500/256))&CHR\$(500-256*INT(500/256)).

Don't print more than 163 characters in a record. You can print multiple-statement XBasic lines, but be sure to use the double-colon token 130 as the separator, not two of the 181 colon tokens.

Each program line must end with CHR\$(0) as the end-of-line indicator, and the last record you print must be CHR\$(255)&CHR\$(255) as the end-of-file indicator.

If you get an I/O ERROR 25 when you try to merge your program, it means that you left off the final double-255. If the program merges, but crashes when you run it, you will probably be able to spot an obvious error in the line when you LIST it. If the line looks OK but gives you a DATA ERROR or SYNTAX ERROR, you left off a CHR\$(0) or gave the wrong count of characters after token 199 or 200. The program published in Part 2 will help you to track down these bugs.

Now let's write a program. What is the longest possible one-liner program?

Well, RANDOMIZE is the longest statement that can stand alone. It is represented by the single token 149, and to repeat it must be followed by the double-colon token 130. Since any line number will take two bytes, let's use a 5-digit line number. And don't forget that final CHR\$(0). That still leaves us 160 of the 163 bytes, so we can repeat tokens 149 and 130 for 79 times, followed by a final 149.

```
100 OPEN #1:"DSK1.LONG",VARIABLE 163,OUTPUT
110 FOR J=1 TO 79 :: MS=MS&CHR$(149)&CHR$(130):: NEXT J
:: MS=CHR$(254)&CHR$(254)&MS&CHR$(149)&CHR$(0):: PRINT #1:MS :: PRINT #1:CHR$(255)&CHR$(255)
120 CLOSE #1
```

RUN, NEW, MERGE DSK1.LONG and LIST - over 34 lines long! But that one-liner doesn't do anything, so try this one.

```
100 OPEN #1:"DSK1.LONG",VARIABLE 163,OUTPUT
110 FOR J=1 TO 52 :: MS=MS&CHR$(162)&"X"&CHR$(130):: NEXT J
:: MS=CHR$(254)&CHR$(254)&MS&CHR$(162)&"X"&CHR$(0):: PRINT #1:MS
120 PRINT #1:CHR$(255)&CHR$(255):: CLOSE #1
```

Again RUN, enter NEW, then MERGE DSK1.LONG, then RUN. You'll get a message BREAKPOINT IN 32510 (don't ask me why!) but just enter RUN again.

Next month - using DEF to make it all easier.

USING SPRITES
by Jim Davis

Last month, the newsletter reprinted some notes on sprites by Jim Peterson. This is a supplement, in fact a simple "HOW TO", so you can start making your own sprites. I've made a worksheet that has a reminder for the commands that are used for sprites. Below is a simple XBasic program to display a dog.

```
100 CALL CLEAR
200 CALL CHAR(96,"0507B7FAFCFCA00")
300 CALL SPRITE(#1,96,2,100,100)
400 CALL MAGNIFY(2)
500 INPUT X
600 CALL MOTION(#1,5,5)
700 INPUT X
```

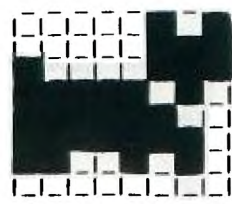


SPRITE WORKSHEET

PATTERN CODE

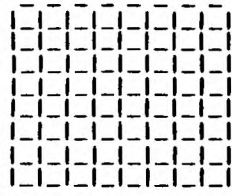
8 4 2 1	HEX	#
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	8
1 0 0 1	9	9
1 0 1 0	A	10
1 0 1 1	B	11
1 1 0 0	C	12
1 1 0 1	D	13
1 1 1 0	E	14
1 1 1 1	F	15

8 4 2 1 8 4 2 1 "DOG"



0	5	1
0	7	2
8	7	3
F	A	4
F	C	5
F	E	6
C	A	7
D	D	8

8 4 2 1 8 4 2 1



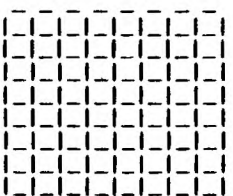
CHARACTER SETS

SET	CODES	CHARACTERS
0	30-31	(CURSOR) (EDGE)
1	32-39	(SPACE) ! " # \$ % & ' () * + , - . /
2	40-47	() * + , - . /
3	48-55	0 1 2 3 4 5 6 7
4	56-63	8 9 : ; < - > ?
5	64-71	@ A B C D E F G
6	72-79	H I J K L M N O
7	80-87	P Q R S T U V W
8	88-95	X Y Z [\] ^ _
9	96-103	` a b c d e f g
10	104-111	h i j k l m n o
11	112-119	p q r s t u v w
12	120-127	x y z () ~
13	128-135	
14	136-143	

CALLS:

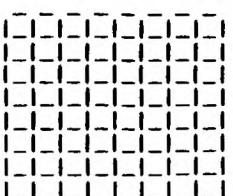
- SPRITE #SPR, CHR#, COLR#, ROW, COL, RV, CV
- CHAR CHR#, PATTERN-ID
- CLEAR
- COINC #SPR, #SPR, TOLER, VAR
- #SPR, ROW, COL, VAR
- ALL, VAR
- COLOR #SPR, FCOLR
- DELSprite #SPR
- #SPR, #SPR...
- DISTANCE #SPR, #SPR, VAR
- #SPR, ROW, COL, VAR
- MAGNIFY FACTOR (1X, 2X, 4SPR, 2X4SPR)
- MOTION #SPR, RV, CV
- PATTERN #SPR, CHAR#
- POSITION #SPR, ROW, COL
- SCREEN COLR#

8 4 2 1 8 4 2 1



---	1
---	2
---	3
---	4
---	5
---	6
---	7
---	8

8 4 2 1 8 4 2 1



---	1
---	2
---	3
---	4
---	5
---	6
---	7
---	8

COLOR COMBINATIONS

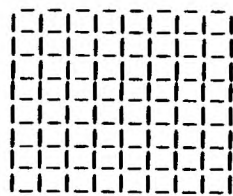
BACKGROUND

	T	B	M	L	D	L	D	C	M	L	D	L	D	M	G	W
	R	L	E	T	A	T	A	Y	E	A	R	T	A	A	R	H
	A	A	D	R	R	R	R	A	D	R	R	R	R	R	A	I
	N	C	G	E	K	B	K	N	R	K	Y	E	K	E	A	T
	S	K	R	E	E	L	E	R	E	R	E	L	E	N	A	E
	P	E	E	E	B	L	R	E	D	D	Y	E	L	G	A	E
	A	E	E	E	L	E	E	D	D	Y	E	L	L	E	A	E
	R	E	E	E	L	E	E	D	D	Y	E	L	L	E	A	E
	E	N	N	E	L	E	E	D	D	Y	E	L	L	E	A	E
	N	T														
1																
2			++	+	+	+++	++	++	+	+	+	+++	++	++	++	
3												+++				++
4																+
5				++		++		++						++	++	++
6				+											+	+
7									++		++			++	+	
8																
9																
10																
11																
12																
13			++	++				++		+++	++			++	+	
14										++						
15																
16																

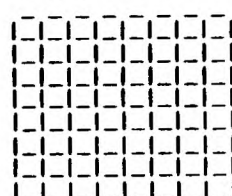
FOREGROUND

- 1 TRANSPARENT
- 2 BLACK
- 3 MEDIUM GREEN
- 4 LIGHT GREEN
- 5 DARK BLUE
- 6 LIGHT BLUE
- 7 DARK RED
- 8 CYAN
- 9 MEDIUM RED
- 10 LIGHT RED
- 11 DARK YELLOW
- 12 LIGHT YELLOW
- 13 DARK GREEN
- 14 MAGENTA
- 15 GRAY
- 16 WHITE

8 4 2 1 8 4 2 1



8 4 2 1 8 4 2 1



---	1
---	2
---	3
---	4
---	5
---	6
---	7
---	8

DELAWARE VALLEY USERS GROUP - PAGE 7

Sprites - Part 2
by Jim Peterson

Several sprites can be created by one statement, such as CALL SPRITE(#1,42,16,10,10,#2,65,2,20,20).

The pattern of several sprites can be changed at once by CALL PATTERN(#1,CHAR,#2,CHAR) - this is very useful when changing the pattern of a character which has been created from two or more sprites.

Several sprites can be set in motion simultaneously, or have their motion changed simultaneously, by CALL MOTION(#1,RV,CU,#2,RV,CU,#3,RV,CU) etc. This is also very useful when moving a character formed of two or more sprites.

Several sprites can be recolored simultaneously with CALL COLOR(#1,C,#2,C) etc.

Several sprites can be relocated together by CALL LOCATE(#1,DOTROW,DOTCOL,#2,DOTROW,DDTCOL) etc.

The position of more than one sprite can be found at one time by CALL POSITION(#1,DOTROW1,DOTCOL1,#2,DOTROW2,DOTCOL2) etc.

A sprite can have only one color, unlike a screen character which can have a foreground and background color. Any dots which are not "turned on" in the character being used for the sprite will be transparent. However, a sprite with a higher number, using a redefined character with all dots turned on and of a different color, can be created at the same dotrow and dotcolumn, giving the illusion of a sprite with foreground and background color. Up to 4 sprites can be stacked in this way to create a multicolored sprite effect. If the sprite is stationary, colored graphics behind all 4 sprites can give the illusion of even more colors.

Sprites always appear to be in front of screen graphics, and lower-numbered sprites always appear in front of higher numbered sprites. However, by skillful swapping of sprites, remarkable 3-D effects can be created, seeming to show a sprite passing before and then behind another, or before and then behind a graphics object.

Another way to simulate 3D is to place a second higher-numbered sprite behind the first, of the same pattern but of a darker color, and offset by a few dotrows downward and to the side, so that when both are set in motion the one appears to be flying above the surface with the second following as its shadow.

Sprites can also be used to add an apparent third color to screen graphics, which can have only two colors in one character.

It is difficult to create the impression of curved lines with redefined characters because they are composed of dots rather than lines. This becomes even more obvious in sprite magnifications 2 and 4, when each dot is magnified into 4 dots. A circle will appear more round, and of the same size, if it is composed of 4 redefined characters in magnification 3 than of one character in magnification 2.

Larger figures can be created using several sprites placed next to each other, providing that not more than four are in a row horizontally. These can be of several colors, and can be set in motion simultaneously.

Although it is stated that sprites, once set in motion, will continue to move regardless of what the program is doing, this is not quite true. If the program is doing a lot of calculating, the sprite motion will be jerky and irregular.

By setting a sprite in motion, and using a loop to change it through a series of patterns, remarkable animated graphics can be created, in much the same way that cartoon movies are made.

It is difficult to control motion exactly with CALL MOTION. For more precise control, sprites can be moved from one point to another, dot by dot, by using CALL LOCATE within a loop, such as FOR DC=1 to 100 :: CALL LOCATE(#1,50,DC) :: NEXT DC. This movement will be very smooth but slow; adding a STEP 2 or STEP 3 will make it faster but less smooth.

If you have Memory Expansion, CALL LOAD (-31806,96) will freeze all sprite motion and CALL LOAD(-31806,0) will release all sprites to their normal motion. By first freezing the motion and then creating up to 28 sprites with predefined motion, all can be set into motion at once, creating some very remarkable effects.

Programs That Write Programs - Part 4
by Jim Peterson

Well, if you have tried your hand at any MERGE format program writing, you have already discovered that it is slow work, and you need to cram more onto a line than will fit. When a little CALL HCHAR(24,12,32,5) turned into CHR\$(157)&CHR\$(200)&CHR\$(5)&"HCHAR"&CHR\$(183)&CHR\$(200)&CHR\$(2)&"24"&CHR\$(179)&CHR\$(200)&CHR\$(2)&"12"&CHR\$(179)&CHR\$(200)&CHR\$(2)&"32"&CHR\$(179)&CHR\$(200)&CHR\$(1)&"5"&CHR\$(182) you gave up? There is an easier way! Using DEF can make the job so simple that you might decide to do all your programming in MERGE format? Well no, it's not quite that easy.

The DEF does slow up program execution time considerably, especially when DEFs call each other, but we can tolerate that here.

For instance, that complicated mess of parentheses to squish a line number can be written just once as DEF LINES\$(X)=CHR\$(INT(X/256))&CHR\$(X-256*INT(X-256)) and then, whenever you need a line number, just write LINES\$(100) or whatever.

The flag token and counting of characters and all for an unquoted string can be DEF'd as US\$(X)=CHR\$(200)&CHR\$(LEN(X))&X\$. Then, to write "HELLO" just write US\$("HELLO") and let the computer do the work. For a numeric value in the unquoted string, use UNS\$(X)=CHR\$(200)&CHR\$(LEN(STR\$(X)))&STR\$(X), and then 999 becomes UNS\$(999).

CALL HCHAR can be DEF HCHAR\$=CHR\$(157) for CALL and, since one DEF can call another, US\$("HCHAR") and, since it is always followed by an opening parentheses, CHR\$(183) - but wait, let's define that open parentheses as OPS=CHR\$(183).

Now DEF HCHAR\$=CHR\$(157)&US\$("HCHAR")&OPS, and you can use HCHAR\$ for CALL HCHAR(.

Let's also DEF the comma with DEF CS=CHR\$(179) and the closing parentheses with DEF CP\$=CHR\$(182). Now that long HCHAR that had you discouraged can be abbreviated to CHAR\$&UNS\$(24)&CS&UNS\$(12)&CS&UNS\$(32)&CS&UNS\$(5)&CP\$.

I have written a program of 162 of these DEFs, and another program to print out a handy look-up chart of them. It would take 4 pages to print them, so if you want them just ask me for a copy.



PAGE 8 - DELAWARE VALLEY USERS GROUP

Programs That Write Programs - Part 5
by Jim Peterson

In addition to writing programs in MERGE format, the same techniques can be used to analyze or modify programs which have been SAVED in MERGE format. The D/U 163 file editor in Part 2 of this series was an example.

Here is a simple program to remove REM statements -

```
100 DISPLAY AT(3,5)ERASE ALL
: "REM REMOVER": : "Program
must be SAVED in": "MERGE for
mat by": "SAVE DSKX.(filename
),MERGE"
110 DISPLAY AT(12,1): "FILENA
ME? DSK" :: ACCEPT AT(12,14)
: FS :: DISPLAY AT(14,1): "NEW
FILENAME? DSK" :: ACCEPT AT
(14,18): NFS
120 OPEN #1: "DSK"&FS,VARIABL
E 163,INPUT :: OPEN #2: "DSK"
&NFS,VARIABLE 163,OUTPUT
130 LINPUT #1:MS :: A=POS(MS
,CHRS(131),1):: B=POS(MS,CHR
$(154),1):: A=MAX(A,B):: IF
A=3 THEN 150 :: IF A=0 THEN
PRINT #2:MS :: GOTO 150
140 PRINT #2:SEGS(MS,1,A-1)&
CHRS(0)
150 IF EOF(1)<>1 THEN 130 ::
CLOSE #1 :: PRINT #2:CHRS(25
5)&CHRS(255):: CLOSE #2
```

The REM statement will begin with either a !, which is CHRS(131), or REM which is CHRS(154). So, line 130 reads in the lines one at a time. A finds the position in the line of ! and B finds the position of REM; one or the other, or both, will not be present and will equal 0. Then MAX finds the larger of A and B, which will be whichever one is present, or 0 if neither.

If ! or REM is in the 3rd position, immediately after the 2-byte line number, we want to delete the line entirely, so we do not reprint it. If A=0 then neither ! nor REM is present, so we reprint the entire line in the new file.

Otherwise, the REM statement is obviously a tail remark, so we reprint to the new file the segment of it starting with the first character and consisting of the number of characters one less than the position of the ! or REM. And, since we have lopped off the end of the line, we do not forget to replace the end-of-line marker CHRS(0).

If we have not reached the end of the file, we go back for the next line. Otherwise, we close the old file, but we remember to add the end-of-file marker to the new file before we close that too.

Bell Compatible?

by Jim Swedlow - ROM Newsletter

Ever noticed that modem ads include a statement about Bell compatibility? This will give you an idea of what that means.

BELL 103A is the standard format for transmitting data by telephone at speeds of 300 baud or less.

BELL 202 is a standard format for transmitting data by telephone at 1,200 baud. Bell 202 format is half duplex only and has now largely been replaced by Bell 212A.

BELL 212A is the standard format for transmitting data by telephone at 1,200 baud.

DEBUGGING

by Jim Peterson

When you have finished writing a program, the next thing you should do is to run it. And, very probably, it will crash!

Don't be discouraged. It happens to the very best of programmers, very often.

So, the next thing to do is to debug it. And you are lucky that you are using a computer that helps you to debug better than some that cost ten times as much.

There are really three types of bugs. The first type will prevent the program from running at all - it will crash with an error message. The second type will allow the program to run, but will give the wrong results.

And the third type, which is not really a bug but might be mistaken for one, results from trying to run a perfectly good program with the wrong hardware, or with faulty hardware. As for instance, trying to run a Basic program, which uses character sets 15 and 16, in Extended Basic.

First, let's consider the first type. The smart little TI computer makes three separate checks to be sure your program is correct. First, when you key in a program line and hit the Enter key, it looks to see if there is anything it can't understand - such as a misspelled command or an unmatched quotation mark. If so, it will tell you so, most likely by SYNTAX ERROR, and refuse to accept the line.

Next, when you tell it to RUN the program, it first takes a quick look through the entire program, to find any combination of commands that it will not be able to perform. This is when it may crash with an error message telling you, for instance, that you have a NEXT without a matching FOR, or vice versa.

And finally, while it is actually running and comes to something that it just can't do, it will crash and give you an error message - probably because a variable has been given a value that cannot be used, such as a CALL MCHAR(R,C,32) when R happens to equal 0.

The TI has a wide variety of error messages to tell you when you did something wrong, what you did wrong, and where you did it wrong. But, it can be fooled! For instance, try to enter this program line (note the missing quotation mark).

```
100 PRINT "Program must be s
aved in:"merge format."
```

And, sometimes you may be told that you have a STRING-NUMBER MISMATCH when there is no string involved, because the computer has tried to read a garbled statement as a string.

Also, the line number given in the error message is the line where the computer found it impossible to run the program; that line may actually be correct but the variables at that point may contain bad values due to an error in some previous line.

If the error occurs in a program line which consists of several statements, and you cannot spot the error, you may have to break the line into individual single-statement lines. This is the easiest way to do that - Be sure the line numbers are sequenced far enough apart. Bring the problem line to the screen, put a ! just before the first ::, and enter it. Bring it back to the screen with FCIN 8, retype the line number 1 higher, use FCIN 1 to delete the first statement and the ! and ::, put a ! before the first ::, and continue. Then, when you have

DELAWARE VALLEY USERS GROUP - PAGE 9

solved the bug, just delete the ! from the original line and delete all the temporary lines.

Pages 212-215 of your Extended Basic manual list almost all the error codes, and almost all the causes of each one - it will pay you to consult these pages rather than guessing what is wrong.

You may create some really bad bugs when you try to modify a program that was written by someone else - especially if you add any new variable names or CALLS to the program. Your new variable might be one that is already being used in the program for something else, perhaps in a subscripted array. I have noticed that programmers rarely use @ in a variable name, so I always tack it onto the end of any variable that I add to a program.

Also, the program that you are modifying may have ON ERROR routines, or a prescan, already built in. The ON ERROR routine was intended to take care of a different problem than the one you create, so it could lead you far astray - you had better delete that ON ERROR statement until you are through modifying.

The prescan had better be the subject of another lesson, but if the program has an odd-looking command !@P- up near the front somewhere, it has a prescan built in. And if so, if you add a new variable name or use a CALL that isn't in the program, you will get a SYNTAX ERROR even though there is no error. One way to solve this is to insert a line with !@P+ just before the problem line, and another with !@P- right after it.

When a program runs, even though it crashes or is stopped by FCTN 4 or a BREAK, the values assigned by the program to variables up to that point will remain in memory until you RUN again, or make a change to the program, or clear the memory with NEW. This can be very useful. For instance, if the program crashes with BAD VALUE IN 680 and you bring line 680 to the screen and find it reads CALL HCHAR(R,C,CH), just type PRINT R;C;CH and you will get the values of R, C and CH at the time of the crash. You will find that R is less than 1 or more than 24, or C is less than 1 or more than 32, or CH is out of range.

In Extended Basic, you can even enter and run a multi-statement line in immediate mode (that is, without a line number), if no reference is made to a line number. So, you can dump the current contents of an array to the screen by FOR J=1 TO 100::PRINT A(J)::NEXT J or you can even open a disk file or a printer to dump it to.

You can also test a program by assigning a value to a variable from the immediate mode. If you BREAK a program, enter A=100 and then enter CON, the program will continue from where it stopped but A will have a value of 100.

You can temporarily stop a program at any time with FCTN 4, of course (the manual says SHIFT C, but it was written for the old 99/4), and restart it from that point with CON. Or you can insert a temporary line at any point, such as 971 BREAK if you want a break after line 970. Or, you can put a line at the beginning of the program listing the line numbers before which you want breaks to occur, such as 1 BREAK 960,970,980. Note that in this case the program breaks just BEFORE those listed line numbers. You can also use BREAK followed by one or more line numbers as a command in the immediate mode.

The problem with using BREAK and CON is that BREAK upsets your screen display format,

resets redefined characters and colors to the default, and deletes sprites. So, it is sometimes better to trace the assignment of values to your variables by adding a temporary line to DISPLAY AT their values on some unused part of the screen. If you want to trace them through several statements, it will be better to GOSUB to a DISPLAY AT. And if you need to slow up the resulting display, just add a CALL KEY routine to the subroutine.

Sometimes, your program will appear to be not flowing through the sequence of lines you intended (perhaps because it dropped out of an IF statement to the next line!) and you will want to trace the line number flow. This can be done with TRACE, either as a command from the immediate mode or as a program statement, which will cause each line number to print to the screen as it is executed. If used as a command, it will trace everything from the beginning of the program, so it is usually better to insert a temporary line with TRACE at the point where you really want to start. Once you have implemented TRACE, the only way to get rid of it is with UNTRACE.

TRACE has its limitations because it can't tell you what is going on within a multi-statement line, and it will certainly mess up any screen display. Sometimes it is better to insert temporary program lines to display line numbers. I use CALL TRACE() with the line number between the parentheses, and a subprogram after everything else 30000 SUB TRACE(X) :: DISPLAY AT(24,1):X :: SUBEND.

Some programmers use ON ERROR combined with CALL ERR as a debugging tool, but I can't tell you much about that because I have never used it. ON ERROR can give more trouble than help if not used very carefully, and I cannot see that CALL ERR gives any information not available by other means.

Sometimes you can debug a line by simply retyping it. It is only very rarely that the computer is actually interpreting a line differently than it appears on the screen, but retyping may result in correcting a typo error that you just could not see. In fact, most bugs turn out to be very simple errors.

When you are debugging a string-handling routine, don't take it for granted that a string is really as it appears on the screen - it may have invisible characters at one or both ends. Try PRINT LEN(M\$) to see if it contains more characters than are showing; or PRINT "*"&M\$*" to see if any blanks appear between the asterisks and the string.

There is no standard way to debug a program. Each problem presents a challenge to figure out what is going wrong, to devise a test to find out what is really happening.

Don't debug by experimenting, by changing variable values just to see what will happen, etc. Even if you succeed, you will not have learned what was wrong so you will not have learned anything - and if your program contains lines that you didn't understand when you wrote them, you will have real problems if you ever try to modify the program. (Believe me, I speak from experience!)

FOR SALE

P-Box w/32K, BRAND NEW (never used) RS232, TI disk controller, and one SS disk drive. \$300 or best offer. Contact John Kelley, (302)328-6059, 5 Holly Drive, Oak Run, New Castle, DE 19720 or TIBBS.

