TANDY ELECTRONICS
The biggest name in little computers

# MICROCOMPUTER NEWSLETTER

Volume No. 4 — July 1979

## MONITORS:

Those of you who have purchased a TRS-80 system recently will have received a different looking monitor than was previously supplied. This is only a styling change as both units are electrically the same. The new monitor will mount on the expansion interface in the same manner as the old one.

## TRS-DOS VER 2.2:

TRS-DOS Version 2.2 has been released. It has several features not found on Ver 2.1 including —

1. Renumbering as a DISKBASIC command
2. Self test on memory and disk drives
3. Re enter BASIC programs from D.O.S.
4. APPEND and VERIFY changes

As well as the above changes there are several new commands plus improvements to existing routines.

This new version for D.O.S. will fix most unexplained problems with version 2.1; Disk 1/0 errors, lost data during read, and back up problems. Version 2.2 also has a modified format routine which lets you format a non-blank disk and has a key-bounce routine which loads automatically on switch on (Booting D.O.S.).

As with future versions of D.O.S., version 2.2. will be sent free of charge to all owners of 26-1160 Disk Drives.

## SOFTWARE:

The following additional TRS-80 software has now arrived and is available from your local Tandy Store. If your store does not have an item in stock, they will be happy to order it for you.

| | |
|---|---|
| 26-2005 Level II Course Part I | $24.95 (Level II) |
| 26-2006 Level II Course Part II | $29.95 (Level II 16K) |
| 26-1705 Advanced Statistical Analysis | $69.95 (Level II) |

The Level II Course is a two part tape course which will guide you through the Level II TRS-80.

Part I requires at least 4K or Memory and Level II. It assumes no previous computer experience and takes the user through the fundamental commands required to write a BASIC program using most of Level II's commands.

Part II is for the more advanced user and deals with the more advanced commands and concepts such as arrays, strings and the use of machine language subroutines. Requires Level II and 16K memory.

**********

## HINTS:

Due to the large number of TRS-80's in use and growing number of users groups, the question of program interchangeability has been raised. As you will know Level I programs can be converted to Level II and Level II tape programs will run on disk systems.

The only problem that could arise is if a friend with a TRS-80 has a line printer and you do not. There are three ways to enable you to use a program written for a TRS-80 line printer.

1) Edit all lines that contain LPRINT to become PRINT statements — tedious and time consuming if the program is very long.

2) Change the address of the line printer driver so that all printer output is sent to the video screen. The following line will accomplish this.

```
10 POKE 16422, (PEEK(16414)):POKE 16423, (PEEK(16415))
```

3) Execute the following program which will search memory for LPRINT commands and change them to PRINT commands.

```
30000 C = 15572:B = MEM:A = C – B
30010 FOR X = 17129 TO (17129 + A)
30020 IF PEEK (X) = 175 THEN POKE X, 178
30030 NEXT X
30040 END
```

## NOTES:

C = Memory size (available memory); 17129 represents the start address of BASIC programs; 175 is the code for LPRINT commands and 178 is the code for PRINT commands. If you have more (or less) memory the 16K use the following value for C.

| MEMORY CAPACITY | "C = " |
|---|---|
| 4K | 3284 |
| 16K | 15572 |
| 32K | 31956 |
| 48K | 48340 |

## APPENDING BASIC PROGRAMS:

Those of you who make extensive use of common subroutines must have said to yourself "Why isn't there some way that I can just load these routines from tape and add them to my program, instead of typing them each time I need them?"
Well it can be done! What you have to do is to fool the CPU into thinking it is loading a program into low memory when in fact it is loading into high memory. This is done by moving the start of memory pointer to point to the end of a resident program and then performing a 'CLOAD'. This can be done in the following manner.

```
1. PRINT PEEK (16549);" ";PEEK(16548):E = 17129
2. S = E: E = PEEK(S + 1)*256 + PEEK (S): IF E > Ø THEN 2
3. POKE 16549, INT(S/256): POKE 16548, S-INT(S/256)*256
4. END
```

2) Run it and note the two values printed on the screen (usually 66 and 233).

3) CLOAD the program that you want to add.

4) From the keyboard POKE the two values from (2) into locations 16549 and 16548.

The preceding steps can be repeated as often as need be to load in subroutines. The following points should be observed:

a) Line numbers of added subroutines must be higher than the resident program (if necessary use the RENUMBER program 26-2004 to renumber the finished program.)

b) Delete lines 1 - 4 before you RUN your appended program.

c) This routine will not work with D.O.S. systems.

## WRONG!

In last month's Newsletter there was a mistake in the Hint section dealing with the line printer.

To check if the line printer is ready, use the following:

```
100 IF PEEK(14312) > 127 THE PRINT "PRINTER NOT READY"
```

# Tandy's Unique "Live Keyboard" Input Routine

### Dress up your programs with this unusual coding

The program reproduced on this page will give you a considerable amount of power and control over the process of input to your programs.

This program, called INKEY, can recognize the difference between numeric and alphanumeric input. (By the way, don't confuse subroutine INKEY with the BASIC command INKEY$, which it uses.)

For numeric input, the program will not allow you to type a "bad" number. For example, it keeps you from putting a letter into numeric field, or two decimal points, etc.

This kind of control can be used to great advantage in computer-assisted instruction programs and other when the person doing the input tends to make errors — i.e., people like you and me!

For quickest execution, this subroutine should be near the beginning of your program. Here's how to use it.

```
100  IN$ = "":W$ = INKEY$:W = 14:WD = 0:WS = WD:WL% = WD:IFFL = WDTHENFL = 1
105  PRINTSTRING$(ABS(FL),136);STRING$(ABS(FL),24);
110  PRINTCHR$(W);:FORW% = 1TO25:W$ = INKEY$:IFW$ <> ""THEN115ELSENEXT:
PRINTCHR$(15);:FORW% = 1TO25:W$INKEY$:IFW$ <> ""THEN115ELSENEXT:GOTO110
115  PRINTCHR$(W);:IFABS(FL) = WL%THEN125ELSEIFFL >0ANDW$) = "
     "ANDW$ < = "Z"THEN170ELSEIFFL(0ANDW$)"/"ANDW$(":"THEN170
117  IFW$ = ","THENPRINTW$;:WL% = WL% + 1:GOTO175
120  IFW$ = "."ANDWD = OTHENWD = 1:GOTO170
123  IF(W$ = " – "ORW$ = " + ")ANDWS = 0ANDWL% = OTHENWS = 1:GOTO170
125  IFW$ <>CHR$(8)THEN150ELSEIFWL% = 0THEN110ELSEPRINTCHR$(24);:IFFL)
     OTHEN135ELSEIFPEEK(16418) = 44THEN140
130  IFPEEK(16418) = 46THENWD = 0:GOTO135ELSEIFPEEK(16418) = 430RPEEK(16418) =
     45THENWS = 0
135  IN$ = LEFT$(IN$,LEN(IN$) – 1)
140  WL% = WL% – 1:POKE16418,136:GOTO110
150  IFW$ = CHR$(24)THENPRINTSTRING$(WL%CHR$(24));:GOTO100
155  IFW$<> CHR$(13)THEN110ELSEPRINTSTRING$(ABS(FL) – WL%,32);
160  PRINTCHR$(15);:W% = 25:NEXT:RETURN
170  PRINTW$;:IN$ = IN$ + W$:WL% = WL% + 1
175  IFABS(FL) = 1THEN160ELSE110
```

First, place the cursor at the beginning of the input field on the screen. Next, give a value to the flag variable FL. Finally, execute a GOSUB 100 which sets INKEY into operation.

You assign a value to the variable FL according to the following rules.

The numeric (or absolute) value of FL is the maximum number of characters INKEY will accept for input. INKEY will take fewer than FL characters, but not more.

If FL is positive, INKEY will expect alphanumeric input. These characters include everything from the "space" (20 hexadecimal) through the letter "Z" (5A hexadecimal).

If, however, you make FL negative, INKEY will expect numeric input. The acceptable numeric characters are **0 through 9, +, –,.** and **comma.** INKEY will only accept one +, – and.(decimal point). Any number of commas may be typed, but they are ignored by INKEY. The keyboard will not respond to any other characters during numeric input.

Finally, if you make the numeric (absolute) value of FL equal to 1, it will not be necessary to press the ENTER key after typing the acceptable character. INKEY will just go ahead and resume execution.

The information you type will return to the calling program in the string variable IN$. (By the way, if the input was numeric, the contents of IN$ may be converted to the internal computational form by using the function VAL...e.g., Y = VAL (IN$.)

Another nice feature of INKEY is that it "formats" the line for input. It prints as many graphics block characters as the maximum number of characters it will accept, and then places the cursor at the beginning of the field.

This way, you get visual "feedback" on how close you are to maximum input as you type. You don't have to count characters.

Of course our old friends backspace ( ← )— to erase the last character — and shift/backspace—to erase the entire line—still work as usual.

One last thing—INKEY is a copyrighted program of Tandy and is included here for the benefit of our customers. This program should not be reproduced or sold under any circumstances. Good luck and happy computing!