Apple's Evolutionary New IIc

COMPUTE!

$2.95
July
1984
Issue 50
Vol. 6, No. 7
£2.25 UK $3.75 Canada
02193
ISSN0194-347X

The Leading Magazine Of Home, Educational, And Recreational Computing

Blueberries And Bunny Hop: Two Fascinating Games For Children

Statistics For Nonstatisticians For The Apple, VIC-20, Commodore 64, Atari, IBM PC And PCjr, TI, And Radio Shack Color Computer

Commodore 64 And 1521 Disk Drive ROM Generations: All The Changes

Applesoft Lister: Format Programs For Fast Debugging

Atari Artist: Create Graphics The Easy Way

July 1984 Vol. 6, No. 7

NOTE: See page 152 before typing in programs.

AP Apple **AT** Atari, **P** PET/CBM, **V** VIC-20, **C** Radio Shack Color Computer, **64** Commodore 64, **TS** Timex/Sinclair, **TI** Texas Instruments, **PCjr** IBM PCjr, **PC** IBM PC, **AD** Coleco Adam, *All or several of the above.

TOLL FREE Subscription Order Line
800-334-0868 (In NC 919-275-9809)

COMPUTE! Publications, Inc.
One of the ABC Publishing Companies

One of the ABC Publishing Companies:
ABC Publishing, President, Robert G. Burton
1330 Avenue of the Americas, New York, New York 10019

COMPUTE! The Journal for Progressive Computing (USPS: 537250) is published monthly by COMPUTE! Publications, Inc., P.O. Box 5406, Greensboro, NC 27403 USA. Phone: (919) 275-9809. Editorial Offices are located at 324 West Wendover Avenue, Greensboro, NC 27408. Domestic Subscriptions: 12 issues, $24. Send subscription orders or change of address (P.O. form 3579) to **COMPUTE!** Magazine, P.O. Box 914, Farmingdale, NY 11737. Second class postage paid at Greensboro, NC 27403 and additional mailing offices. Entire contents copyright © 1984 by COMPUTE! Publications, Inc. All rights reserved. ISSN 0194-357X.

Remember, though, that unless you're handy with a soldering iron, changing the device number via hardware modifications is permanent.

Also remember that—although not specifically stated in the user's manual—opening the disk drive and performing these modifications yourself may void the warranty. Consult your local Commodore representative and thoroughly read the user's manual before attempting any of these modifications.

If you feel uncomfortable changing the device number using the hardware method, the same thing can be achieved with a software (program) modification. Here is the procedure:

1. *Turn off all disk drives.*
2. *Turn on the disk drive whose device number you want to change.*
3. *Type and enter the following commands:*

```
CLOSE 15: OPEN 15,8,15
PRINT#15,"M-W",CHR$(119)CHR$(0)CHR$(2)CHR$
  (n+32)CHR$(n+64)
CLOSE 15
```

Change the n *in the PRINT#15 command above to the device number you want to assign to the disk drive. It might be best to limit the device numbers to the range between 9 and 14.*

Now you can turn on the other drive(s), and start processing.

Note the syntax of the PRINT#15,"M-"... *command. Contrary to the instructions in the 1541 user's manual, do not include the colon after the memory-write (M-W) command. If the colon is included, the device number change will not be successful.*

The 1541 demo disk that was included with your drive also contains a program to change device numbers. LOAD the program DISK ADDR CHANGE *then RUN. The user prompts will tell you what to do.*

VIC Animation

I am 11 years old and I own a VIC-20. My friend owns a Commodore 64. We both make graphics on them. He has the advantage of sprites, but I have figured out a way for the VIC-20 to have a form of sprites. The VIC-20's graphics are made from top to bottom, which allows vertical sprites. Here's a demonstration program.

Bryan D. Stanton

```
5 DIM A(8):K=7167
1Ø PRINT"{CLR}"
15 POKE36869,255
2Ø FORM=7168TO7311:POKEM,Ø:NEXTM
3Ø FORX=1TO18:PRINTTAB(9);CHR$(63+X):NEXT
   X
4Ø FORN=1TO8:READA(N):POKEK+N,A(N):NEXTN
5Ø FORR=1TO137
55 FORL=1TO8:POKEK+L,A(L):NEXTL:POKEK+1,Ø
   :K=K+1
6Ø NEXTR
65 PRINT"{CLR}":POKE36869,24Ø:END
7Ø DATA 6Ø,24,24,24,255,126,24,24
```

Joystick To Keyboard Control On The TI

Many of your TI-99/4A games require a joystick. Unfortunately, I don't own one. Could you provide a routine that would enable me to convert these programs to keyboard control?

Mike Burgin

Several approaches can be taken to convert a program from joystick to keyboard control on the TI. Probably the simplest approach, in console BASIC, is to GOSUB to a keyboard subroutine whenever the JOYST subprogram is CALLed.

You should locate this keyboard subroutine at the beginning of the program, to speed execution. Let's put such a subroutine at line 10. The entire routine will occupy four lines beginning at line 10, so RESequence your program to begin at line 50.

*Next, find where the subprogram JOYST is CALLed within the program. The general form for this statement is CALL JOYST (*n*,X,Y). Here,* n *refers to the joystick number (either 1 or 2) while X and Y are values returned based on the joystick position.*

X and Y may be represented by any legitimate numerical variable name. Note the variable names used for X and Y in the CALL JOYST statement and then replace this statement with GOSUB 10.

Then, type in the following lines:

```
5 GOTO 5Ø
1Ø CALL KEY(Ø,K,SS)
2Ø X=((K=67)+(K=68)+(K=82))*-4+((K=83)+(K
   =87)+(K=9Ø))*4
3Ø Y=((K=69)+(K=82)+(K=87))*-4+((K=67)+(K
   =88)+(K=9Ø))*4
4Ø RETURN
```

Now, substitute the variable names from the CALL JOYST statement into the above subroutine for X and Y. Also, if K and SS are used in the main program, you may need to name them differently here.

Just as with the CALL JOYST statement, X and Y will be returned as –4, 0, or +4 in lines 20 and 30. The standard arrow keys (E, S, D, and X) are tested for in this routine along with W, R, Z, and C for diagonal movement.

Providing a routine for keyboard control in Extended BASIC is even easier. Since we can write our own subprogram (using SUB), we no longer need worry about the variable names for X and Y in the main program. Variables used in a subprogram are local to that subprogram.

Our subprogram, which we'll call JOY, must be placed at the end of the program. Assuming there's room above line 999, type in the following:

```
1ØØØ SUB JOY(Z,X,Y)
1Ø2Ø X=((K=67)+(K=68)+(K=82))*-4+((K=83)+
     (K=87)+(K=9Ø))*4
1Ø3Ø Y=((K=69)+(K=82)+(K=87))*-4+((K=67)+
     (K=88)+(K=9Ø))*4
1Ø4Ø SUBEND
```

Next, in the main program, change CALL JOYST(n,X,Y) to CALL JOY(n,X,Y) so that our keyboard subprogram will be CALLed rather than the system joystick subprogram (n is 1 or 2).

Last, for either console or Extended BASIC, check to see if the fire button is used. You should find a statement of the form CALL KEY(n,K,S) in the program (n is 1 or 2). Shortly thereafter in the program, a check for the value of K will be made. If K is equal to 18, then the fire button has been pressed.

With keyboard control, we can use the space bar rather than the fire button. Change n (which is 1 or 2) to 0 in the appropriate CALL KEY(n,K,S) statement. Also, change 18 to 32 in the subsequent check for the value of K.

Hex-To-Decimal Conversions

As a faithful reader of your magazine, I'd like to say that I'm surprised at how many computer hobbyists still have not found a simple decimal-to-hexadecimal conversion program. And I haven't noticed one in any issue of your magazine, so I've written this short BASIC program to do the conversions. It will work on most computers with little or no modification.

Frank Sgabellone

```
10 A$="Ø123456789ABCDEF":INPUT"DEC/HEX";A
   :B=1:C=9:D=16↑C:PRINTA;" = $";:A=A+1
                                  :rem 1Ø7
2Ø IFA-D>ØTHENA=A-D:B=B+1:GOTO2Ø :rem 156
3Ø PRINTMID$(A$,B,1);:B=1:C=C-1:D=16↑C:IF
   C>-1THEN2Ø                     :rem 235
4Ø PRINT"{5 SPACES}":GOTO1Ø          :rem 9
```

Hexadecimal numbers are widely used in machine language since they are more convenient for that kind of programming than the normal decimal numbers.

Compilers For The 64

I would appreciate some clarification on compilers. I have seen advertisements for several compilers (DTL-BASIC, and Metacompiler for Forth) and would like to know if they actually produce ML code that will run on any 64.

In other words, can I write a program in BASIC or Forth, run it through the compiler, and have ML code that will run on another 64 that doesn't have access to the compiler?

Paul Filiant

There are two types of compilers: those that produce native code *(machine language), and those that generate* pseudocode *(P-code). P-code compilers translate the source program into another, smaller, faster language. This pseudocode must still be interpreted, like BASIC, but it's interpreted much more quickly. Also, P-code interpreters can run the same P-code program on many machines, whatever the microprocessor used. But to run a P-code compiled program, you must have a copy of the P-code interpreter.*

Other compilers generate true machine language. This has the advantage of speed, if not portability. The object code produced by the compiler needs a set of general-purpose subroutines. Otherwise, the compilation of PRINT would expand into a large chunk of machine language each time it is used. Instead, it is more memory-efficient to compile PRINT into a subroutine call to the general PRINT routine. The set of subroutines required is called the runtime package, and is included in the compiled program.

A compiler generally produces a complete program that will run on any machine, without the compiler itself. However, we now run into the problem of copyright. You have written and therefore own the rights to the original, uncompiled program, but who owns the compiled program? You might think you retain the copyright, since compiling is something like translating a book into a different language.

However, you don't own the runtime package. Some companies require you to pay a royalty for selling the compiled program. Other companies require a special security key to run the compiled program. (A security key prevents a program from running without it. It is usually a ROM chip or some device that plugs into a joystick or cassette port.) This is like a royalty; you must buy security keys for every copy of the compiled program you distribute. Still other companies give you the freedom to distribute your compiled code, as long as you include a notice specifying that it was compiled with their product. Be sure you understand what copyright rules are enforced by the compiler company. If in doubt, write them.

Reading The Atari 800 PIA Registers

I am 16 years old and own an Atari 800. Currently, I am trying to use the joystick ports for certain I/O applications. So far, the only problem I've encountered is speed. The registers that store input information from the joyports are updated only every sixtieth of a second. This is too slow for me. Is there any way to read the joystick ports at a faster rate?

Christopher Terpin

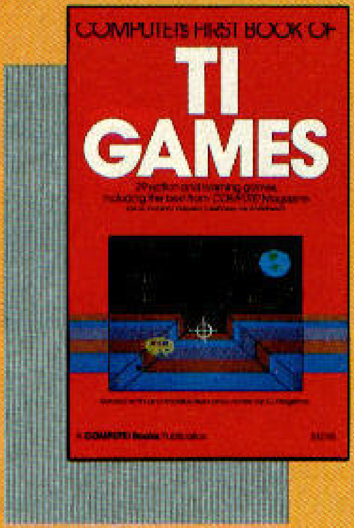
Instead of using the shadow locations for the joysticks, you can read the joystick ports directly from the data direction registers in the 6520 PIA chip. These are truly general-purpose input/output ports, with one byte used for two joysticks. Each bit can be programmed independently for input and output. Complete information on this can be found in the Atari Hardware Reference Manual. *Some information is also found in* Mapping the Atari, *available from COMPUTE! Books. In the meantime, examine the information found at $D300.*

COMPUTE!'s First Book Of Commodore 64 Games

Packed full of games: "Snake Escape," "Oil Tycoon," "Laser Gunner," "Zuider Zee," and many more. Machine language games requiring fast hands and a good eye, as well as strategy games which will exercise your mind. Introductory chapters and annotated listings provide ideas and techniques for writing games. An excellent ntroduction for 64 owners who want to begin writing games.

[illegible] pages, paperback.
piral bound for easy access to programs.

12.95
BN 0-942386-34-5

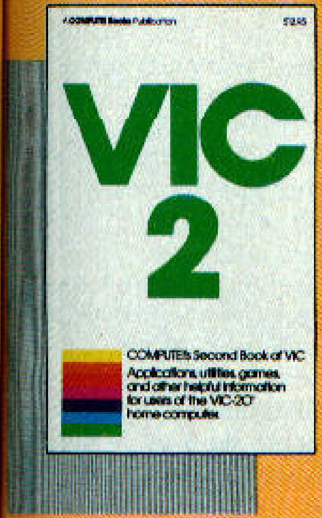


COMPUTE!'s First Book Of TI Games

Although this book is packed with ready-to-type-in games (29 in all), it is more than just a book of games. It is designed to teach game programming techniques. Introductory chapters explain the special features of the TI-99/4 and 99/4A, giving advice on coding techniques. Most games include an explanation of how the program works. Contains mazes, chase games, old favorites, thinking games, creative challenges, and more.

211 pages, paperback.
Spiral bound for easy access to programs.

$12.95
ISBN 0-942386-17-5

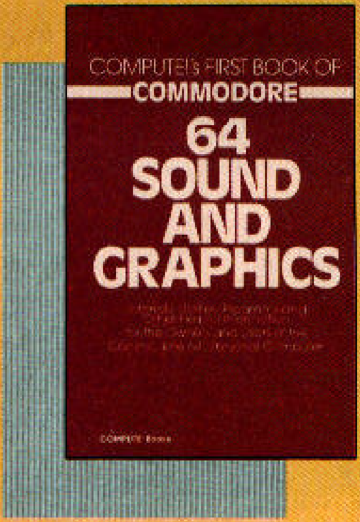

VIC
2
COMPUTE!'s Second Book of VIC
Applications, utilities, games, and other helpful information for users of the VIC-20 home computer

COMPUTE!'s Second Book Of VIC

This is just the book to follow the best-selling *First Book of VIC*: clear explanations of programming techniques, an extensive memory map, a mini word processor, a system for creating sound effects, a custom character maker, a machine language assembler, and "Gumball," an extraordinary all-machine-language game.

274 pages, paperback.
Spiral bound for easy access to programs.

$12.95
ISBN 0-942386-16-7



COMPUTE!'s First Book Of 64 Sound And Graphics

Clear explanations of the 64's sound and graphics capabilities. Includes many tutorials and example programs: "MusicMaster," a complete music synthesizer; "High-Resolution Sketchpad," an all-machine-language program for making computer art; and "Ultrafont Character Editor," one of the best character editors available. The appendices feature useful reference charts and conversion tables.

275 pages, paperback.
Spiral bound for easy access to programs.

$12.95
ISBN 0-942386-21-3

COMPUTE!™ Publications, Inc. abc
One of the ABC Publishing Companies
Post Office Box 5406, Greensboro, North Carolina 27403

Statistics For Nonstatisticians

A. Burke Luitich

Basic statistical methods can help you make logical decisions in everyday situations.

For the most part, elementary statistical methods measure a group of similar things to see how these measurements vary when compared to some standard. Another use for statistics is to see how creating a group of objects can cause variations in these objects.

This program, "Statistics," takes your raw data and returns figures which you can use to make everyday decisions, for example, about the best way to build a wall or how much cash you'll need when you go shopping.

As a first example, let's look at two ways to cut a 2 x 4, by using a power table saw and a handsaw. We set the table saw guide to one foot and cut five pieces. We cut five more pieces using a handsaw, then measure the actual lengths of all ten pieces to see how accurately we made the cuts.

If nothing unusual is allowed to affect the cutting, we can expect the length of the pieces to vary depending on the process used. Statisticians call this an *unbiased random sample.*

Assume the measurements are as follows:

Table saw lengths (feet)	Handsaw lengths (feet)
1.05	1.22
.98	.91
1.03	.80
1.07	1.28
.96	.88

The Same Mean

A look at the values alone suggests that cutting with the handsaw is a far less consistent method than using the table saw. However, if you add up the lengths for each method and divide by 5 (the total cuts for each) you will find that both methods give the same *mean* (average) length of 1.018 feet.

Just finding an average length doesn't tell us much. What we need to know is how widespread the values are likely to be, and which method gave us the most lengths that were nearer our standard of one foot. In statistical terms, we need to calculate the *range* and the *standard deviation.*

We find the range by subtracting the shortest length from the longest, for each cutting method. For the handsaw the range is .48 feet (1.28 –.80), and for the table saw the range is .11 feet (1.07 – .96). Immediately, we can see that the table saw cut more consistently, because the range, or variation, is smaller.

We can use the standard deviation and the mean length to predict how often a given length is likely to occur. You don't have to worry about how to calculate a standard deviation; the program does this for you. If you type in the above lengths for the handsaw, the program will return a standard deviation of .217 feet. The standard deviation for the table saw is .047 feet.

Degree Of Accuracy

If we made a large number of cuts, then measured and graphed the lengths, the graph would form a bell curve, or normal distribution. By combining the standard deviation and the mean length, we get a range of lengths that includes 68.3 percent of all lengths (again, you don't have to know the theory; just use the number). To illustrate, first take the mean length, 1.018 feet, and subtract from it the standard deviation for the handsaw, .217 feet, to get .801 feet. Then add the standard deviation to the mean length to get 1.235 feet. This means that 68.3 percent of our lengths fall in the range between .801 and 1.235 feet.

By adding and subtracting the standard deviation (.047 feet) with the mean length of the table saw cuts (1.018 feet), we find that 68.3 percent (roughly two-thirds) of these lengths fall in the range from .971 to 1.065 feet.

If you want a wider sample, you must increase the number of standard deviations. To include 95.4 percent of all lengths, use two standard deviations. For the handsaw, we now have .434 feet, two standard deviations. Combining it with the mean length, we get a range of .584 to 1.452 feet. Our table saw range becomes .924 to 1.102 feet (1.018 plus and minus .094).

Food For Thought

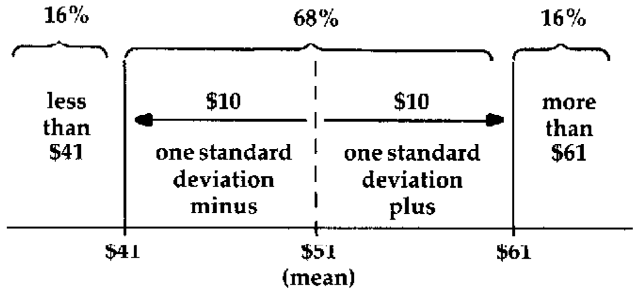
You can use the same methods to calculate a food budget. In this case, your data consists of the amounts you spent on groceries over a 13-week period (one-fourth of a year):

Week	Amount	Week	Amount
1	$42	8	47
2	50	9	65
3	75	10	49
4	37	11	43
5	51	12	52
6	45	13	54
7	56		

If you type this data into the Statistics program, you will find that your mean amount spent was about $51; that your spending varied from $37 to $75, for a range of $38; that you spent more than $50 (your median amount) as often as you spent less than that; and your standard deviation is about $10.

Applying The Statistics

Combining one standard deviation and the mean (or average) amount spent, we find that two-thirds of the weeks you spend between $41 and $61 at the grocery store. One-sixth of the time you spend less than $41; one-sixth of your bills are more than $61. So, if you budget $61 for groceries, you'll have enough 84 percent of the time.



If you want to be sure you'll have enough in case prices rise, you might want to use two standard deviations. By adding two standard deviations ($20) to the mean amount ($51), you will find that, to be about 98 percent sure, you should budget $71 each week.

There are other factors to be considered, of course, such as vacations, birthday parties, or visiting relatives, that can affect your food budget. The Statistics program does not take these kinds of things into account. But it does give you a tool which takes some of the guesswork out of everyday decision-making.

The Statistics program requests input of the size of the sample, or number of items to be entered (line 410), then requests the values of the sample measurements (lines 500–550). All the statistics referred to in this article are then calculated, that is, mean, standard deviation, median, and range.

Lines 325–350 and 4900–5610 give the user a thumbnail sketch of the information to be calculated and a description of each of the statistics. While the sample size is limited to 100 for the VIC version (other versions allow up to 300), this should be more than adequate in most cases.

Error Correction

An error correction routine is included in lines 555–580 and 5900–6190. This provides for the change of any entry before the calculation. While the program is running, a delay of up to two minutes will be experienced while the program performs several sorts on the data. This is normal for BASIC and may be longer for sample sizes in the 80 to 100 range or greater.

Program 1 requires at least 3K of expansion memory in the VIC computer. If the instructions, error correction routine, and headings are eliminated, the program will run on an unexpanded VIC. Specifically, the following lines should be deleted if the program is to run without memory expansion: 95–180, 325–350, 555–580, 4900–5610, and 5900–6190.

Further reductions can be made by reducing the sample size, redimensioning the array in line 90 to the new sample size (SA), and changing the value of 100 in line 420 to the new maximum sample size.

Statistics for a sample of 100 readings requires about 30–45 minutes to calculate by hand. This program requires about 8–10 minutes, including input.

Program 1: VIC Statistics

Refer to the "Automatic Proofreader" article before typing this program in.

```
90 DIM SA(100)                         :rem 185
95 REM GENERAL INTRODUCTION            :rem 242
100 PRINT "{CLR}":POKE 36879,126:PRINT "
    {BLK}"                             :rem 207
110 FOR K=1 TO 3:PRINT:NEXT K          :rem 147
120 PRINT TAB(4);"{4 DOWN}{RVS}VIC STATIS
    TICS{OFF}"                         :rem 208
130 PRINT TAB(9);"{DOWN}FOR"           :rem 249
140 PRINT TAB(2);"{DOWN}{RVS}NON-STATISTI
    CIANS{OFF}"                        :rem 171
180 FORK=1TO2000:NEXTK                  :rem 98
190 PRINT"{CLR}"                       :rem 254
200 PRINT TAB(4);"{2 DOWN}THIS PROGRAM"
                                       :rem 108
210 PRINT TAB(3);"CALCULATES THE":rem 188
220 PRINT "{2 SPACES}FOLLOWING VALUES"
                                       :rem 230
```

```
IK 1160 IF EN>N OR EN<1 OR EN<>INT(EN)
         THEN 1150
PO 1170 POSITION 7,11:? "Sample ";EN:P
        OSITION 22,11:? "Value ";SA(EN
        )
KN 1180 POSITION 7,13:? "Enter your ne
        w value":POSITION 7,14:INPUT C
        :SA(EN)=C
CE 1190 POSITION 7,19:? "Any more chan
        ges (y/n)?"
AM 1200 GOSUB 1350
DF 1210 IF A=89 THEN 1120
JC 1220 GOTO 400
LH 1230 GOSUB 1060:POSITION 5,2:? "The
        se are the first ten values:"
PM 1240 POSITION 11,5:? "ENTRY":POSITI
        ON 22,5:? "VALUE"
GM 1250 FOR K=1 TO 10
JJ 1260 POSITION 12,K+7:? K:POSITION 2
        4,K+7:? SA(K):NEXT K
ML 1270 GOTO 1070
GH 1280 POSITION 5,2:? "These are the
        next ten values:{DELETE}":IF K
        <=300 THEN GOSUB 1340
HU 1290 CT=8:FOR K=K TO K+9
FA 1300 IF K>300 THEN K=K+9:NEXT K:GOT
        O 400
KF 1310 POSITION 12,CT:? K:POSITION 24
        ,CT:? SA(K)
FB 1320 CT=CT+1:NEXT K
MI 1330 GOTO 1070
JJ 1340 FOR J=1 TO 10:POSITION 12,J+7:
        ? "{3 SPACES}":POSITION 24,J+7
        :? "{15 SPACES}":NEXT J:RETURN
FE 1350 GET #1,A:IF A<>89 AND A<>78 TH
        EN 1350
KL 1360 RETURN
```

Program 4: TI-99/4A Statistics

```
100 DIM SA(300)
110 CALL CLEAR
120 PRINT TAB(10);"STATISTICS"
130 PRINT : : :
140 PRINT TAB(13);"FOR"
150 PRINT : : :
160 PRINT TAB(7);"NON-STATISTICIANS
    "
170 PRINT : : : : : :
180 FOR K=1 TO 400
190 NEXT K
200 CALL CLEAR
210 PRINT "THIS PROGRAM CALCULATES
    THE": :
220 PRINT "FOLLOWING VALUES FROM DA
    TA": :
230 PRINT "YOU INPUT:"
240 PRINT : :
250 PRINT TAB(4);"1. MEAN"
260 PRINT : :
270 PRINT TAB(4);"2. STANDARD DEVIA
    TION"
280 PRINT : :
290 PRINT TAB(4);"3. MEDIAN"
300 PRINT : :
310 PRINT TAB(4);"4. RANGE"
320 PRINT : : :
330 PRINT TAB(2);"PRESS ANY KEY TO
    CONTINUE"
340 PRINT :
350 GOSUB 2170
360 SUM=0
370 MEAN=0
380 DFF=0
390 SDDEV=0
400 RG=0
410 REM INSTRUCTIONS REQUEST
420 PRINT TAB(6);"INSTRUCTIONS (Y/N
    )?"
430 PRINT : : : : : : : : : :
440 GOSUB 2170
450 IF (K<>89)*(K<>78)THEN 440
460 IF K=78 THEN 490
470 GOSUB 1330
480 REM DATA ENTRY
490 CALL CLEAR
500 PRINT TAB(3);"ENTER SAMPLE SIZE
     ";
510 INPUT N
520 IF (N>300)+(N<=1)THEN 490
530 CALL CLEAR
540 PRINT TAB(3);"ENTER YOUR DATA O
    NE VALUE": :
550 PRINT "AT A TIME, THEN PRESS":
    :
560 PRINT "RETURN.": : : :
570 PRINT TAB(3);"IF YOU MAKE AN ER
    ROR,": :
580 PRINT "CONTINUE WITH DATA ENTRY
    .": :
590 PRINT "YOU WILL BE ABLE TO MAKE
    ": :
600 PRINT "CORRECTIONS LATER.": : :
     : :
610 PRINT TAB(2);"PRESS ANY KEY TO
    CONTINUE": :
620 GOSUB 2170
630 FOR I=1 TO N
640 CALL CLEAR
650 PRINT "DATA ENTRY #";I;
660 INPUT R$
670 SA(I)=VAL(R$)
680 NEXT I
690 REM ERROR CORRECTION REQUEST
700 CALL CLEAR
710 PRINT TAB(3);"ANY CORRECTIONS (
    Y/N) ?"
720 PRINT : : : : : : : : : :
730 GOSUB 2170
740 IF K<>89 THEN 770
750 GOSUB 1800
760 REM CALCULATION OF MEAN AND STD
    . DEVIATION
770 PRINT TAB(9);"PLEASE WAIT": : :
780 PRINT "STATISTICS BEING CALCULA
    TED"
790 PRINT : : : : : : : : :
800 FOR I=1 TO N
810 SUM=SUM+SA(I)
820 NEXT I
830 MEAN=SUM/N
840 FOR I=1 TO N
850 DFF=DFF+(SA(I)-MEAN)^2
860 NEXT I
870 SDDEV=SQR(DFF/(N-1))
880 REM SORT OF DATA INTO NUMERIC O
    RDER
890 FL=0
900 FOR I=1 TO N-1
910 IF SA(I)<=SA(I+1)THEN 960
920 Q=SA(I)
930 SA(I)=SA(I+1)
940 SA(I+1)=Q
950 FL=1
```

```
960 NEXT I
970 IF FL=1 THEN 890
980 REM CALCULATION OF RANGE
990 RG=SA(N)-SA(1)
1000 LR=SA(1)
1010 HR=SA(N)
1020 REM CALCULATION OF MEDIAN
1030 IF N/2<>INT(N/2)THEN 1090
1040 IF SA(N/2)<>SA(N/2+1)THEN 1060
1050 MDD=SA(N/2)
1060 IF SA(N/2)=SA(N/2+1)THEN 1080
1070 MDD=(SA(N/2)+SA(N/2+1))/2
1080 GOTO 1110
1090 MDD=SA(INT(N/2+1))
1100 REM PRINT RESULTS TO SCREEN
1110 CALL CLEAR
1120 PRINT TAB(5);"CALCULATION RESU
     LTS": :
1130 PRINT "***********************
     *****": : :
1140 PRINT "SAMPLE SIZE";TAB(19);N:
     :
1150 PRINT "MEAN (X BAR)";TAB(19);I
     NT(MEAN*10000+.5)/10000: :
1160 PRINT "STD. DEVIATION";TAB(19)
     ;INT(SDDEV*10000+.5)/10000: :
1170 PRINT "MEDIAN";TAB(19);INT(MDD
     *10000+.5)/10000: :
1180 PRINT "RANGE";TAB(19);INT(RG*1
     0000+.5)/10000: :
1190 PRINT "LOWEST VALUE";TAB(19);L
     R: :
1200 PRINT "HIGHEST VALUE";TAB(19);
     HR: : : :
1210 PRINT TAB(8);"PRESS ANY KEY"
1220 GOSUB 2170
1230 REM REQUEST TO CONTINUE OR END
1240 PRINT " WISH TO PROCESS MORE D
     ATA": :
1250 PRINT TAB(12);"(Y/N)?": : : :
     : : : : : :
1260 GOSUB 2170
1270 IF K=78 THEN 1320
1280 FOR I=1 TO N
1290 SA(I)=0
1300 NEXT I
1310 GOTO 360
1320 END
1330 PRINT TAB(3);"THE MAXIMUM NUMB
     ER OF EN-": :
1340 PRINT "TRIES YOU CAN MAKE IS 3
     00.": :
1350 PRINT "THE MINIMUM NUMBER IS 2
     ": : :
1360 PRINT TAB(3);"THE MEAN IS THE
     ARITH ": :
1370 PRINT "METIC AVERAGE OF THE NU
     MBERS": :
1380 PRINT "YOU ENTER.": : :
1390 PRINT TAB(3);"STANDARD DEVIATI
     ON IS A": :
1400 PRINT "MEASURE OF HOW WIDELY Y
     OUR": :
1410 PRINT "NUMBERS SPREAD FROM THE
     ": :
1420 PRINT "AVERAGE.": : :
1430 GOSUB 2160
1440 CALL CLEAR
1450 PRINT TAB(3);"SINCE THE VALUES
      YOU ENTER": :
1460 PRINT "TEND TO FORM A BELL CUR
     VE": :
1470 PRINT "(NORMAL DISTRIBUTION)
     THE": :
1480 PRINT "STD. DEVIATION IS A MEA
     SURE": :
1490 PRINT "OF THE AREA UNDER THE B
     ELL": :
1500 PRINT "CURVE.": : :
1510 PRINT TAB(4);"NO. OF STD.
     {4 SPACES}% AREA"
1520 PRINT TAB(5);"DEV.(+/-)"
1530 PRINT TAB(4);"-----------
     {4 SPACES}------": :
1540 PRINT TAB(8);"1{11 SPACES}68.3"
1550 PRINT TAB(8);"2{11 SPACES}95.5"
1560 PRINT TAB(8);"3{11 SPACES}99.7"
1570 PRINT TAB(8);"4{11 SPACES}99.9"
     : : :
1580 GOSUB 2160
1590 PRINT TAB(3);"THE MEDIAN IS TH
     E VALUE AT": :
1600 PRINT "THE MID-POINT OF YOUR D
     ATA.": : :
1610 PRINT TAB(3);"THE RANGE IS THE
      DIF-": :
1620 PRINT "FERENCE BETWEEN YOUR LO
     WEST": :
1630 PRINT "DATA VALUE AND THE HIGH
     EST.": :
1640 PRINT "IT IS A QUICK AND-DIRTY
     ": :
1650 PRINT "ESTIMATE OF THE SPREAD.
     ": :
1660 PRINT "STANDARD DEVIATION IS M
     ORE": :
1670 PRINT "RELIABLE, HOWEVER.": :
     : :
1680 PRINT TAB(3);"PRESS ANY KEY TO
      START"
1690 GOSUB 2170
1700 RETURN
1710 REM DISPLAY CORRECTION OPTION
1720 GOSUB 2170
1730 IF (K<>67)*(K<>78)*(K<>81)THEN
      1720
1740 FL=0
1750 IF K<>78 THEN 1780
1760 FL=1
1770 GOTO 1980
1780 IF K=81 THEN 770
1790 REM ERROR CORRECTION SUBR
1800 PRINT "REMEMBER INCORRECT SAMP
     LE #": :
1810 PRINT TAB(11);"(Y/N) ?": : : :
      : : : : : :
1820 GOSUB 2170
1830 IF K=78 THEN 1980
1840 INPUT "WHAT IS THE SAMPLE # ?
     ":EN$
1850 EN=VAL(EN$)
1860 IF (EN>N)+(EN<1)+(EN<>INT(EN))
     THEN 1840
1870 PRINT : :
1880 PRINT "SAMPLE";EN;"{3 SPACES}"
     ;"VALUE=";SA(EN)
1890 PRINT : :
1900 PRINT "ENTER YOUR NEW VALUE :
     "
1910 INPUT SA(EN)
```

Richard Mansfield, Senior Editor

Trapping Bugs

It was a moth, according to legend, that caused a program to crash in the early days when computers were built of vacuum tubes and tons of copper wire. The critter had flown into the machine. From this we get the term *bug*, meaning that there is an error, a problem in a computer program. And tracking down bugs is called *debugging*.

As all programmers soon learn, there is no permanent cure for bugs—they are always hiding inside a freshly written program of any complexity. Some bugs are obvious and will show up the first time a program is tried out. Some are hidden away and permit most of the program to run without error. A complex program might run well for weeks or months and then a particular sequence of events will trigger a well-hidden bug.

Program Sketches

For many, programming is similar to painting or sculpting. First you jump in and roughly create the outlines, the main ideas. At this point you've essentially made a sketch of the final program. Then you start testing the program by RUNning it, refining it until it performs as it should.

What are the best ways to look for bugs? Luckily, the most common bugs, typos, are reported to you by BASIC itself. On the Atari, if you try to enter a line like this: PRINF X, you will get an immediate SYNTAX ERROR report. Other versions of BASIC wait to report typos until after you RUN the program, but the effect is the same. Your computer tells you what's wrong and which line to fix.

Many other bugs show up quickly when you first try out the program: Nothing appears onscreen; things appear, but in the wrong places; or the numbers are all wrong. In other words, the program isn't even coming close to your expectations. These are often easy bugs to work with because they aren't usually caused by the interaction of two parts of your program. There's some gross failure somewhere. You've simply got to look at your formatting routine or your mathematical definitions to see where the problem is.

Between The Cracks

Some of the hardest bugs to find are hidden in the cracks. They are usually the result of a clash between two otherwise perfectly functional subroutines. For example, if your program uses the variable *T* to stand for the total of an addition problem and then you use a subroutine with a loop that also uses T:

```
10 T = BOLTS + WASHERS
.
.
.
800 FOR T = 1 TO 500
```

As you can see, no matter what your total of bolts and washers is, it will be left at 500 anytime you use the subroutine at line 800.

A similar interaction between variables can be even more subtle. In many versions of BASIC, only the first two letters of a variable name have any significance. So, if you name one thing BOLTS and another thing BOWLING, these two things will appear to the computer as a single variable called *BO*. And, as in the example above, the most recent number assigned to BO will be the *only* value that variable can have.

The Worst Bugs

But the worst bugs are not in the computer at all. They're in the programmer's mind. And since you must use your brain to ferret out the errors *caused* by that brain—you can see the paradox. These errors tend to be of two types: incorrect setups and bad logic.

An example of an incorrect setup would be thinking you've defined a variable when, in fact, you haven't, or using > when you mean <. The variations on this theme are endless and you can look at > dozens of times and not even stop to think about it as a possible source of error.

Bad logic would include such things as subroutines which exit via GOTO instead of RETURN; INPUT at the wrong time; or forgetting about the first or last item in a sequence like a DATA list.

Sometimes there's only one way to find a deeply hidden bug: stepping through the program. There are two levels of step testing. You can insert STOP in various places, then check to see that the variables are what they should be at these stopping points. Then CONT to the next STOP and ask to have the variables printed again (type: ? X,Y,Z$). This rough test is often enough to pinpoint the place where the program has gone wrong.

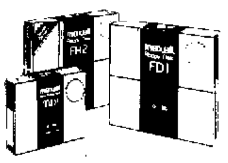
Alternatively, you can use the single-stepping TRACE function found in many programmer's aid programs. These aids add commands to BASIC like RENUMBER, DELETE, and usually have a single-stepping function as well.

When you activate a TRACE command, your program executes step by step, one command at a time. After each command, the status of all active variables is displayed on screen along with the program line so you can locate where things begin to come unglued. Often, a TRACE function permits you to define how fast it will execute and even allows you to turn it on or off from within the program. TRACEing is a slow, but nearly always successful way to trap the most devious bugs.

If all else fails, it's sometimes advisable to ask for help from a friend. His brain won't have been implicated in the original error, and he can therefore often spot the > you keep ignoring. ©

Maxell Floppy Disks

The Mini-Disks with maximum quality.



Dealer inquiries invited. C.O.D's accepted.
Call FREE (800) 235-4137.

PACIFIC EXCHANGES
100 Foothill Blvd.,
San Luis Obispo, CA 93401
In Cal. call (800) 592-5935 or
(805) 543-1037

MasterCard VISA

Software City®

SCHOLARSHIP AWARDS

$10,000 Scholarship plus four $1,000 Scholarships

Entry Deadline: December 31, 1984

Your original computer program could win you a $10,000 Scholarship and substantial royalties!

CATEGORIES

- Business Applications
- Education
- Systems Software
- Recreation
- Home Applications

For complete Scholarship Award instructions and application, visit your local Software City store.

SOFTWARE ALWAYS DISCOUNTED

Programs, disks, peripherals, accessories and custom forms, plus a complete selection of books and magazines.

Software City®

ALABAMA: Birmingham • Huntsville **ARIZONA:** Phoenix • Tucson
CALIFORNIA: San Francisco **COLORADO:** Westminster
CONNECTICUT: Danbury • W. Hartford • Orange • Stamford
DELAWARE: Wilmington **FLORIDA:** Orange Park • St. Petersburg
Sarasota • Tampa **GEORGIA:** Atlanta • Augusta • Columbus **IDAHO:**
Idaho Falls **ILLINOIS:** Arlington Heights • Chicago **INDIANA:**
Indianapolis **IOWA:** Davenport **KENTUCKY:** Louisville **LOUISIANA:**
Baton Rouge **MARYLAND:** Bethesda **MASSACHUSETTS:** Lexington
W. Springfield **MICHIGAN:** Ann Arbor • Okemos • Southfield
Sterling Hts. **MISSOURI:** Creve Coeur • Independence **NEBRASKA:**
Omaha **NEW JERSEY:** Bergenfield • Cherry Hill • Englishtown
Fair Lawn • Green Brook • Hamilton • Linwood • Little Falls
Midland Park • Montvale • Morristown • Pine Brook • Pompton Lakes
Red Bank • Ridgefield • Summit • Teaneck **NEW YORK:**
Albany • Brooklyn • Buffalo • Fairport • Forest Hills • Great Neck
Long Island • Manhattan • Mt. Kisco • North White Plains
Poughkeepsie • Scarsdale • Staten Island • Syracuse
NORTH CAROLINA: Charlotte • Fayetteville • Greensboro **OHIO:**
Centerville • Columbus • Mayfield Hts. **OKLAHOMA:** Tulsa
PENNSYLVANIA: Bethel Park • Exton • Fairless Hills • King of Prussia
Pittsburgh (3 locations) • Whitehall **SOUTH CAROLINA:** Spartanburg
TEXAS: Austin • Dallas • Houston **VERMONT:** Burlington
VIRGINIA: Fairfax • Falls Church • Richmond (2 locations)
Virginia Beach **WASHINGTON:** Bellevue **CANADA:**
Halifax, Nova Scotia • Montreal, Quebec (look for Logiville)
ENGLAND: London **PUERTO RICO:** San Juan

Business Software Catalog at all stores.

RETAIL STORE FRANCHISES
Offering by prospectus only. Direct inquiries to
SOFTWARE CITY FRANCHISE DEPARTMENT C
1415 Queen Anne Road, Teaneck, NJ 07666

C. Regena

Programming Techniques In TI BASIC

This month, by answering some of the common questions I have received from readers, I'm going to give you a variety of programming techniques that you can use in your own programs.

How do you clear part of a screen?

Let's say you have onscreen a nice picture with a description underneath. CALL CLEAR will clear the whole screen; but you want to clear the printing, not the picture. Use CALL HCHAR with the row and column parameters under the picture, and use the number of repetitions that will clear the section you want. For example, to clear the lower half of the screen, CALL HCHAR(13,1,32,32*12). We're starting with row 13, column 1, and clearing with the space (character code 32) for 32*12 squares—32 columns times 12 more rows.

To clear with a different color, redefine a character (in a color set you are not using) as a colored square, then use CALL HCHAR to put that character on the screen:

```
300 CALL COLOR(13,16,16)
310 CALL HCHAR(13,1,128,32*12)
```

To clear a vertical section of the screen, use CALL VCHAR:

```
CALL VCHAR(1,17,32,24*16)
```

To try out this technique, try this sample program:

```
100 CALL HCHAR(1,1,42,32*24)
110 CALL HCHAR(13,1,32,32*12)
900 GOTO 900
```

Change line 110 to the CALL VCHAR statement above and try the program. Next take out line 110 and put in lines 300 and 310 listed above. Experiment with different numbers of repetitions.

How do you get a border around the screen?

CALL SCREEN(*c*), where *c* is a number from 1 to 16, defines the screen color. When you use this statement in a program, the whole screen instantly changes color. CALL COLOR(*s*,*f*,*b*) defines the character colors. The characters are divided into sets of eight characters each. The *s* in the parentheses is the set number and can be from 1 to 16. The *f* is the foreground color of the character, *b* the background color, and they can be one of the 16 color numbers, from 1 to 16.

Now take a look at the characters in set 1. The space is code 32 in set 1. The screen is filled with spaces wherever there isn't any printing or graphics. If you change the color of set 1 to something other than the screen color (background color 1), you'll get color where all the spaces are.

```
100 CALL CLEAR
110 CALL SCREEN(14)
120 CALL COLOR(1,2,16)
900 GOTO 900
```

Press FCTN 4 (CLEAR) to stop the program. You've got a border on the top and on the bottom, but you would like the sides also. When we PRINT messages we have a 28-column line, but when we do graphics we actually have 32 columns—there are two columns on each side of the regular printing section. They currently have spaces in them. To get the screen color in those columns, add

```
115 PRINT ::::::::::::::::::::::::
```

Or, as you print messages, those extra columns fill with the screen color. (As you PRINT, columns 1, 2, 31, and 32 will contain character 31.) A quicker way to get rid of the spaces in those columns is to fill the columns with a character in the screen color. You may add these lines instead:

```
115 CALL CHAR(152,"")
116 CALL VCHAR(1,1,152,48)
117 CALL VCHAR(1,31,152,48)
```

Now try a few PRINT messages, such as

```
150 PRINT "HELLO"
```

Notice that the letters have little squares of the screen color around them. All the color sets are automatically defined as CALL COLOR(S,2,1), which is black with a transparent background. The color number 1, transparent, will be the screen color. If you want the printing to be black on your inner screen color (the color of the spaces), you need to define the sets with the background color that you used in set 1. Change line 120 above to

```
120 FOR S=1 TO 12
130 CALL COLOR(S,2,16)
140 NEXT S
```

This defines a white background for the first 12 character sets, those sets which have letters and symbols. Now run the program and you will see that the message no longer has the screen color background.

How do you make a simple math drill with graphics?

I have had quite a few requests for an arithmetic drill program. Many readers would like to develop such programs on their own and want to know how to draw a certain number of pictures for the numbers chosen randomly in a simple math problem.

Here is a short program to give you the general idea of using the graphics. I defined character 128 to be the picture. The variables A and B can be numbers from zero to four. Lines 170–200 print the problem on the screen—a simple addition problem. Lines 210 and 220 draw the right number of characters for A and B.

Program 1: Simple Math Drill

```
100 REM  SIMPLE MATH
110 CALL CLEAR
120 CALL CHAR(128,"0024002418")
130 CALL COLOR(13,2,11)
140 RANDOMIZE
150 A=INT(5*RND)
160 B=INT(5*RND)
170 CALL HCHAR(8,10,A+48)
180 CALL HCHAR(10,8,43)
190 CALL HCHAR(10,10,B+48)
200 CALL HCHAR(11,8,95,3)
210 CALL HCHAR(8,12,128,A)
220 CALL HCHAR(10,12,128,B)
230 CALL SOUND(150,1497,4)
240 CALL KEY(0,K,S)
250 IF S<1 THEN 240
260 IF K=32 THEN 400
270 IF K=A+B+48 THEN 310
280 CALL SOUND(100,330,2)
290 CALL SOUND(100,262,2)
300 GOTO 240
310 CALL HCHAR(13,10,K)
320 PRINT "CORRECT!"
330 CALL SOUND(100,262,2)
340 CALL SOUND(100,330,2)
350 CALL SOUND(100,392,2)
360 CALL SOUND(200,532,2)
370 CALL SOUND(1,9999,30)
380 CALL CLEAR
390 GOTO 140
400 CALL CLEAR
410 END
```

If you prefer to have a space between graphics characters, place a character in every other space. You can do this by changing lines 210 and 220 above to the following:

```
210 FOR C=12 TO 12+2*(A-1) STEP2
212 CALL HCHAR(8,C,128)
214 NEXT C
220 FOR C=12 TO 12+2*(B-1) STEP2
222 CALL HCHAR(10,C,128)
224 NEXT C
```

In this sample program, an addition problem is presented and the student answers by pressing a number. If it is incorrect, there is an "uh-oh" sound. If it is correct, an arpeggio is played and the computer goes to the next problem. To stop, press the space bar.

How can you draw a bar graph?

This procedure is similar to the previous sample program. The easiest way to draw a bar graph is to use HCHAR with the appropriate number of repetitions (or VCHAR). You may need to scale the numbers. Take the highest number you'll need to graph, relate it to the greatest number of repetitions you can have in your HCHAR statement, and stay on that row.

Another method is to use PRINT and print the right number of characters for the bar. The following sample program segment demonstrates this method. Character 128 will be a red square. For purposes of illustration, I will use random numbers N up to 90 for the amounts to be graphed. You would probably have specific numbers that have been calculated or read in from DATA.

A is the scaled value (rounded) for N—for every four units one square can be drawn. Line 170 prints the number N then says to start the next printing in the fifth print column. Lines 180–200 print the appropriate number of red squares.

Program 2: Bar Graph Generator

```
100 REM  BAR  GRAPH
110 CALL CLEAR
120 CALL COLOR(13,7,7)
130 FOR I=1 TO 10
140 RANDOMIZE
150 N=INT(90*RND)
160 A=INT(N/4+.5)
170 PRINT N;TAB(5);
180 FOR B=1 TO A
190 PRINT CHR$(128);
```

```
200 NEXT B
210 PRINT ::
220 NEXT I
230 GOTO 230
240 END
```

How do you print a list of items in more than two columns?

As you know, the comma in PRINT statements prints items in two columns—items start either in the first print position or the center position. To get three columns or more, use the TAB function. TAB works like the tab key on a typewriter. You may specify which column to start printing. TAB(7) would start the next print item in the seventh print column. Here's a sample that types three columns of names.

```
100 CALL CLEAR
110 READ L$,M$,N$
120 IF L$="@" THEN 180
130 PRINT L$;TAB(10);M$;TAB(19);N$
140 GOTO 110
150 DATA MIKE,BOB,DICK,RICH
160 DATA JIM,JERRY,MARY,PAULA
170 DATA CHRIS,KEVIN,KATHY,KIRK,@,@
,@
180 END
```

How can you print a screen without seeing the scrolling?

Some people don't like to see scrolling as they print. Messages on the TI are always printed on the twenty-fourth row then moved upward. To block this motion, change the screen to black first (because the printing is black), print the messages, then change the screen back to a different color so you can read the printing.

```
100 CALL CLEAR
110 CALL SCREEN(2)
120 PRINT "THIS IS AN EXAMPLE"
130 PRINT ::"TO SEE A SCREEN"
140 PRINT ::"ALL AT ONCE.":::::
150 CALL SCREEN(4)
160 GOTO 160
```

How can you print what is on the screen to the printer?

I'm sorry, but I don't know how to do a *screen dump* of graphics because none of the printers I have right now has the graphics capabilities. You will need to look at your own brand printer manual to see how to use the dot-addressable graphics. If you have a screen of printing, however, with regular printed symbols, you can use the following procedure. The character in each row and column is determined, then that character is printed on the printer. You may need to change the OPEN statement in line 100 to suit your particular printer configuration.

```
100 OPEN #1:"RS232.BA=600"
110 FOR ROW=1 TO 24
120 FOR COL=3 TO 30
130 CALL GCHAR(ROW,COL,G)
140 PRINT #1:CHR$(G);
150 NEXT COL
160 PRINT #1
170 NEXT ROW
180 CLOSE #1
190 END
```

If you want everything you are printing to go both to the screen and to the printer, use both a PRINT statement and a PRINT #1 statement for items printed.

```
100 OPEN #1:"RS232.BA=600"
110 CALL CLEAR
120 PRINT #1:CHR$(12)
130 PRINT "HELLO"
140 PRINT #1:"HELLO"
150 PRINT "ANY MESSAGE"
160 PRINT #1:"ANY MESSAGE"
170 CLOSE #1
180 END
```

Line 120 above goes to the top of a page.

How can you simulate time on the TI?

If you need an exact time, use the CALL SOUND statement in which you can specify an exact duration in milliseconds. If you don't want to hear the sound, use a high frequency and the softest volume.

```
100 PRINT "START"
110 CALL SOUND(1000,9999,30)
120 CALL SOUND(1,9999,30)
130 PRINT "END"
140 END
```

Line 120 is necessary to end the first sound.

If you want to time someone as they are pressing keys to move or are answering a question, use a counter in your CALL KEY loop. You can't relate this counter to an exact time because in each program it will be different—depending on how you do the programming, how long your program is, and how full the memory is. However, once you have your program working, you can print the counter value and use a stopwatch to figure out a formula that relates the actual time to the counter value. ("Type-ette Timer" in my *Programmer's Reference Guide to the TI-99/4A* from COMPUTE! Books uses this technique to time how fast you can type sentences.) Here is a sample:

```
100 T=0
110 CALL KEY(0,K,S)
120 T=T+1
130 IF S<1 THEN 110
140 PRINT T
150 GOTO 100
160 END
```

The faster you press a key, the lower the value for T will be. The longer you wait, the more times the computer will go through the loop and increment T.

Other computers use PRINT AT; how can we do it?

In TI Extended BASIC you can specify a row and column to begin printing an item. However, we don't have that feature in regular console BASIC on the TI. There are several ways to accomplish this, though they're slower than regular printing. First, you can use the regular PRINT statement, perhaps with the TAB function, and then use colons to move the message up to the proper row.

PRINT TAB(9);"START PRINTING"::::::

The main problem with this method is that it scrolls the screen. If I am labeling graphics, I do all the printing first, then use CALL HCHAR and CALL VCHAR to put up the graphics.

Another method is to treat the letters in the printed message as graphics characters, and use CALL HCHAR to specify the row and column to place the letters on the screen. Here's a general-purpose subroutine that you can use. M$ is the message you want printed, R is the row, and C is the column you want the message to start in.

```
300 FOR L=1 TO LEN(M$)
310 CALL HCHAR(R,C-1+L,ASC(SEG$(M$,
    L,1)))
320 NEXT L
330 RETURN
```

Before you call the subroutine with a GOSUB, specify a row R and a column C and the message M$:

```
900 M$="TEST PRINTING"
910 R=6
920 C=12
930 GOSUB 300
```

How can I put a code in my program?

I have had lots of young people ask me how they can write a program so that whoever runs it must enter a code before the program continues—they don't want their brothers and sisters using their program. The general idea is that you put a code name in the program as a string variable. Next, use INPUT for the user who is running the program to type in the code. Now compare the INPUT value with the code to see whether to continue or not.

```
100 CALL CLEAR
110 CODE$="RANDY"
120 INPUT "ENTER CODE NAME: ":A$
130 IF A$=CODE$ THEN 160
140 PRINT ::"SORRY, INVALID CODE."
150 STOP
160 REM PROGRAM CONTINUES
```

The only problem with specifying the code in line 110 is that anyone can load the program, then LIST it to find out what the code name is. One method I use so people can't read the code name is to hold down the CTRL key (key with the red dot) while you type your code message. Line 110 will now look like this:

110 CODE$="   "

or you may get some funny-looking graphics characters between the quotes. Now when someone lists your program, they can't tell what the code name is. When *you* run the program, be sure to hold the CTRL key down when you INPUT the code name, and it will match the code in the program.

A Couple Of Warnings

Always use the SHIFT key on the left side of the keyboard to type the plus sign. You don't want to go for the right SHIFT key and accidentally hit the FCTN key—and *quit*!

Do *not* use TI Extended BASIC to run regular TI BASIC programs because they may not run properly. One reason is the double colon used in PRINT statements, and another reason is that I often use graphics in character sets 15 and 16, which are not available in Extended BASIC.

If you have a disk drive attached to your computer, the disk uses up some memory. For any of my published programs, type in CALL FILES(1) and press ENTER, then type NEW and press ENTER, then proceed normally (load a program or start typing a program). This procedure clears about 1000 bytes of memory so a program can fit.

Until Next Time ...

I hope these ideas help you in your programming. Your computer can be a lot of fun. Part of the joy of programming is getting that machine to do what you want it to do. As I continue these columns I hope to present a variety of programs so you can see that this computer is really quite versatile. Your suggestions and letters are always welcome. ©

Program Conversion With Sinclair BASIC And TI BASIC

Julie Knott and Dave Prochnow

Program conversion between BASIC dialects is often easier than imagined. This tutorial demonstrates the compatibility of TI BASIC and Sinclair BASIC and includes helpful tables and sample conversion programs.

Program conversion can be an easy and convenient operation. Virtually every home computer uses BASIC, which, because it's easy to learn and to manipulate, is ideal for ready-made language conversion. However, no two BASICs are created equal. For many years the industry's standard was Microsoft BASIC, then different dialects began to emerge. Manufacturers would use the Microsoft format and introduce nuances and subtleties in the structuring, labeling each of these alterations an "improvement" of BASIC. But many were only changes in the protocol—the manner in which a command is expressed. And it becomes relatively easy to convert BASIC dialects if the major differences are in protocol or syntax.

Two versions of BASIC which lend themselves to such a program conversion are Sinclair BASIC and TI BASIC. Sinclair BASIC, used in the Timex/Sinclair-1000, is unique in that all keywords are single-stroke entries. For example, the P key stands for the PRINT command. (The use of a touch-membrane keyboard dictates this procedural necessity.)

Texas Instruments TI-99/4 and 4A use TI BASIC, which is more conventional in that each individual letter has to be typed—PRINT would require five keystrokes.

There are only slight variations between Sinclair BASIC and TI BASIC, but their similarities allow for easy program conversion. By studying which statements and commands are equivalent for both BASICs, and what substitutions are necessary, program conversion can be relatively simple. Also, you can virtually double your software by translating programs published for the other machines.

For the sake of brevity, the following glossary does not contain all of the keywords in Sinclair BASIC and TI BASIC—only those words which are confusing, complicated, or not directly translatable have been listed. For a more complete listing, consult the appropriate user's manual.

Sinclair BASIC

AND — a logical operator, often used in IF-THEN statements

ACS — function that gives the arc cosine of an angle in radians

ASN — function that gives the arc sine of an angle in radians

AT — used in a PRINT statement to give a location at which to PRINT

BREAK — stops program execution, key activated and may not be included as a command in a program

CLEAR — deletes all variables from memory

CLS — clears the screen

CODE — a string function used to obtain the numeric value of a given character

CONT — resumes execution of a program following a report code

COPY — copies the contents of the screen to printer

DELETE — erases keywords and characters while programming

FAST — fast mode, a time-saving mode for increased RUN speed

FUNCTION — function mode

GRAPHICS — graphics mode

INKEY$ —used in IF-THEN statements as a conditional statement, executes exclusive of ENTER

LLIST — lists the contents of a program listing to a printer

LOAD — loads a prerecorded program from cassette tape to the computer's memory

LPRINT — PRINTs to printer

NOT — inverts the truth value of an expression

OR — a logical operator, used in conditional statements

PAUSE — creates a time delay while the program is RUNning

PEEK gives the value of the byte at a specific address in memory

PI — gives the value of PI

PLOT — draws a pixel at a given location

POKE — puts a numeric value into memory at a specific address, erasing the previous one

SCROLL — scrolls the screen up one line, eliminating the top line

SLOW — slow mode, the standard operating mode

UNPLOT — erases a pixel at a given location

USR — calls a machine language routine at a specific memory address

TI BASIC

APPEND — an open mode, allows data to be added at the end of the existing file

ASC — ASCII value or character code

BASE — option base

BREAK — sets breakpoints in a program, used for error checking

BYE — erases memory, returns to title screen

CALL — special subprogram to obtain color and sound

CLOSE — closes the association between a file and a program

CONTINUE (CON) — continues a program after a breakpoint

DATA — stores data

DEF — defines user-established functions in a program

DELETE — removes a program or data file from a filing system

DISPLAY — prints on screen only

ELSE — conditional part of IF-THEN/ELSE statement

END — terminates program, similar to STOP

EOF — End-Of-File, determines if the end of a file has been reached on an accessory device

FIXED — files with a specified length, used with RELATIVE or SEQUENTIAL

INTERNAL — file type recorded in machine language

NUMBER (NUM) — automatic line number generator

OLD — loads a previously SAVEd program

ON — a conditional numeric expression, used with ON-GOTO or ON-GOSUB

OPEN — prepares to use data files stored in accessory device

OPTION — option base, sets lower limit of array subscripts to 1 instead of 0

OUTPUT — transfers data out of a program

PERMANENT — file life

POS — position

READ — reads data in DATA statements

REC — points to a specific record in a RELATIVE file

RELATIVE — defines a file with FIXED

RESEQUENCE (RES) — reassigns line numbers

RESTORE — identifies which DATA to use with the next READ

SEG$ — string segment, substring

SEQUENTIAL — defines a file, used with FIXED or VARIABLE

SUB — part of GO SUB

TRACE — outlines the order that statements will be performed when the program is RUN

UNBREAK — removes breakpoints

UNTRACE — cancels TRACE

UPDATE — an open mode, for reading and writing into files

VARIABLE — defines a varying length file, used with SEQUENTIAL

Special Subprograms Used With Graphics And Sound In TI BASIC

Each subprogram is preceded by CALL (for example, CALL CLEAR)

CLEAR — erases the entire screen

COLOR — specifies screen character colors

SCREEN — changes screen color

CHAR — defines user-special graphic characters

HCHAR — places a character and repeats it horizontally

VCHAR — similar to HCHAR except repetition is vertical

SOUND — produces tones and noises of different duration, frequency, and volume

GCHAR — reads a character anywhere on the screen

KEY — transfers character directly from keyboard to program without ENTER

JOYST — inputs data with remote controllers

Easy Conversions

Many of the commands and statements of these two BASICs are directly translatable. Table 1 shows the direct BASIC equivalents for Sinclair BASIC and TI BASIC. The only major differences between these two dialects are in their nomenclature.

Several dialects of BASIC have an ON-GOTO statement expressed as:

```
ON x GOTO w,y,z
```

where x is the value of a numerical expression and w, y, and z are line numbers. This statement is available in TI BASIC, but not in Sinclair BASIC. Through the use of conditional expressions, the

Sinclair BASIC substitution is:

GOTO (w AND x=1)+(y AND x=2)+(z AND x=3)

The operators AND and OR would make this possible.

The translation of many program lines requires only the replacement or substitution of a word unique to that particular BASIC. Several of the more common functions and statements are evaluated in this manner in Table 2. The following Sinclair BASIC line will await the pressing of the Y key, exclusive of ENTER:

```
100 IF INKEY$ <> "Y" THEN GOTO 100
```

To perform the same statement in TI BASIC, replace INKEY$ with the KEY subprogram, as follows:

```
100 CALL KEY(0,K,Z)
110 IF K<>89 THEN 100
```

The main difference is in the structuring. The KEY subprogram (subprograms are obtained with CALL) uses three variables to establish where the key is originating, its ASCII code, and its status. In this example the ASCII code of 89 represents the Y character.

TI BASIC has the ability to store expressions and assign values to these variables with the statements DATA, READ, and RESTORE (see the glossary). Vast arrays can be developed and initialized with this method. Sinclair BASIC is not directly convertible with DATA, READ, and RESTORE. A large battery of LET statements *could* crudely handle the data. Alternatively, a properly DIMensioned INPUT statement allows the creation of such an array. Upon completion, the INPUT statements are removed and a GOTO command is used for program starting (RUN erases the variable array).

String Handling

Strings can be equally bothersome. Slicing will supply usable substrings in Sinclair BASIC. A string expression's parameters govern the start and finish of the slice. No special statement is necessary:

A$(x TO z)

with x representing the starting number and z the finish. For example:

"COMPUTE" (4 TO 7) = "PUTE"

The statement SEG$ (A$,x,y) in TI BASIC has the same result, but, again, with different nomenclature. X is the number of the start for the substring and Y is the length of the substring. For example:

A$ = "COMPUTE"
SEG$ (A$,4,4) = "PUTE"

While string slicing is easily translated, the TI BASIC user-defined function is not. DEF allows the definition of functions within a program.

DEF X$ = "Y"

The string function's name is X and the string expression is Y. VAL and string variables can be user-defined in Sinclair BASIC.

LET X$ = "Y"
VAL X$ = Y

This is a very limited and a "sometimes-maybe" proposition. DEF has the ability to also handle numeric functions. This ability, as well as using parameters in argument evaluation, is beyond VAL's means.

When attempting a program conversion you may run across a few Sinclair BASIC terms that are completely unfamiliar to you. The terms USR, PEEK, and POKE are not procedures for the examination of some strange alien creature. They are primarily associated with direct access to memory. To call a machine language routine that begins with a specific address, USR is used. This will start a machine language program running. POKE is used by the T/S-1000 to store a numeric value at a specific address in the computer's memory. For example:

POKE 17529, 38

POKEs the value 38 into address 17529. Conversely, the PEEK command is used to read certain addresses to see what is stored there. The PEEK command is followed by the address to be PEEKed.

PRINT PEEK 17529

would PRINT the number 38. When you are translating a program from Sinclair BASIC which contains USR, PEEK, and POKE statements, you must find out what they accomplish and then interpret that into TI BASIC.

PRINTing on the screen is accomplished by a blending of line and row markers. Memory conservation techniques notwithstanding, PRINT can be used to move the PRINT line. For example:

PRINT
PRINT
PRINT "COMPUTE"

Sinclair BASIC also allows the movement of PRINT with AT and TAB.

PRINT AT x,y

and

PRINT TAB y

TAB moves the PRINT position a prescribed number of spaces to the right. Even though TAB is present in TI BASIC, the vocabulary is different. Line changes are accomplished with colons (:). Duplicating the above examples,

PRINT::...(x) TAB (y)

and

PRINT TAB (y)

X is the number of colons necessary to equal the value of the line number (x) in the Sinclair BASIC example.

The Timex/Sinclair lacks color and sound features, but these features are of importance on the TI-99/4. TI BASIC's color and sound statements are subprograms that begin with CALL. Clever usage of Sinclair BASIC's character set can duplicate some of these color combinations. As a rule, however, TI BASIC CALL subprograms should be removed and not directly substituted in a program conversion to Sinclair BASIC. This allows concentration on the program's more important graphics. Consultation with Texas Instruments' *User's Reference Guide* will provide the proper protocol for development and inclusion of color and sound subprograms in a Sinclair BASIC converted to TI BASIC program.

To illustrate the principles of program conversion, examine these sample programs. While each program is unique in its results, the approach is similar and convertible. The purpose of this program is to display the entire character set along with the character codes.

T/S-1000 Version

```
10 FORA=Ø TO 255
20 LET A$ = CHR$ A
30 PRINT AT 10,13; A
40 PRINT AT 7,10; A$;"{6 SPACES}"
50 PRINT AT 7,17; A$;"{6 SPACES}"
60 PRINT AT 13,10; A$;"{6 SPACES}"
70 PRINT AT 13,17; A$;"{6 SPACES}"
80 NEXT A
```

TI-99/4 Version

```
100 FOR A=32 TO 127
110 B$=CHR$(A)
120 CALL CLEAR
130 CALL SCREEN(2
140 PRINT TAB(13);B$;TAB(18);B$
150 PRINT
160 PRINT TAB(14);A
170 PRINT
180 PRINT TAB(13);B$;TAB(18);B$
190 PRINT ::::::::
200 FOR S=3 TO 16
210 CALL SCREEN(S)
220 CALL SOUND(400,110+20*(S-3),1)
230 NEXT S
240 NEXT A
```

In line 10 of the Timex/Sinclair example, a loop establishes the number of character codes to be examined (the entire character set is 0 to 255). Note that the characters with codes 67–127 cannot be printed and will show on the screen as question marks. Lines 20 and 30 PRINT the code or numeric value for each character. The arrangement of the printed characters is defined in lines 30–40. In this way, you can easily interpret the delay, and read the code value and the character almost simultaneously. This program will RUN until BREAK is pressed.

Table 1:
Reference Chart Of BASIC Equivalencies

Sinclair BASIC	= TI BASIC
ABS	ABS
ATN	ATN
CHR$	CHR$
CODE	ASC
COS	COS
DIM	DIM
EXP	EXP
FOR	FOR
GOSUB	GOSUB or GO SUB
GOTO	GOTO or GO TO
IF	IF
INPUT	INPUT
INT	INT
LEN	LEN
LET	LET
LN	LOG
LOAD	OLD
NEW	NEW
NEXT	NEXT
PRINT	PRINT
RAND	RANDOMIZE
REM	REM
RETURN	RETURN
RND	RND
RUN	RUN
SAVE	SAVE
SGN	SGN
SIN	SIN
SQR	SQR
STEP	STEP
STOP	STOP
STR$	STR$
TAB	TAB
TAN	TAN
THEN	THEN
TO	TO
VAL	VAL
CLS	CALL CLEAR

Table 2:
Substitution Chart For BASIC Nonequivalents

Sinclair BASIC	= TI BASIC
NEW	BYE
PRINT	DISPLAY
GOTO (W AND X=1) +(Y AND X=2)+(Z AND X=3)	ON X GOTO W,Y,Z
IF X THEN GOTO Y	IF X THEN (Y)
LET X=Y+Z	LET X=Y+Z or X=Y+Z
PAUSE or FOR X=Z TO Y NEXT X	FOR X=Z TO Y NEXT X
INKEY$	CALL KEY
A$(X TO Z)	SEG$(A$,X,Y)
PI	4*ATN (1)
LET X$="Y" VAL X$=Y	DEF X=Y
STOP	END or STOP
PRINT AT X,Y	PRINT::...(X) TAB(Y)
PRINT TAB Y	PRINT TAB(Y)
ASN 1	π /2 or 4*ATN(1)/2 or 2*ATN(1)
IF X=Y THEN GOTO A GOTO Z	IF X=Y THEN A ELSE Z ©

signed for the Apple II and IIe, with software for the Atari and IBM PC and PCjr to be available in the future.

One disk drive is required to run the program.

All programs teach 1000 of the most common words in the target language. Where words have more than one meaning, the program allows for those other meanings, along with English translation.

The package retails for $56.95. Languages currently available include Spanish, French, German, Italian, Biblical Hebrew, modern Hebrew, and Arabic. Latin, Russian, Polish, Swedish, and classical Greek will be available in the near future.

Each language program is menu-driven with sequential review, random review, and quiz options.

Soflight Software
2223 Encinal Station
Sunnyvale, CA 94087
(408) 735-0871

Personal Finances Software

A software product designed to help consumers make personal financial decisions has been announced by Electronic Arts. Called *Financial Cookbook*, the program contains "recipes," or formulas, that produce answers about money matters.

Through the program's 32 different recipes, users can figure such data as returns on investments, effective tax shelters and IRAs, effects of inflation, mortgage calculations, and tax rates.

Each recipe asks the user to enter variables, such as interest or inflation rates, and then makes calculations based on those numbers.

Calculations for 11 basic tax shelters available to most consumers are found in the recipes. The instruction manual includes a tutorial, recipe instructions, and index.

Financial Cookbook is available for the entire Apple II line, the IBM PC and PCjr, Commodore 64, and Atari 800.

Suggested retail price is $50.

Electronic Arts
2755 Campus Drive
San Mateo, CA 94403
(415) 571-7171

Text Adventure For Youngsters

Infocom has announced *Seastalker*, an interactive text adventure game for ages 9 and up.

In it, players aboard the specially equipped submarine Scimitar must save the Aquadome, earth's first undersea research station.

Unfortunately, the Scimitar hasn't been tested in deep water, and the crew of the Aquadome may have a traitor in its ranks. If the right course isn't charted, players might end up as shark bait.

Solving hints are included in the game package.

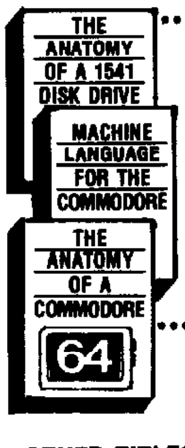
Seastalker is available for the Apple II, Atari, Commodore 64, IBM PC and PCjr, and TI-99/4A at a cost of $39.95.

Infocom, Inc.
55 Wheeler St.
Cambridge, MA 02138
(617) 492-1031

New Product releases are selected from submissions for reasons of timeliness, available space, and general interest to our readers. We regret that we are unable to select all new product submissions for publication. Readers should be aware that we present here some edited version of material submitted by vendors and are unable to vouch for its accuracy at time of publication.

COMPUTE! *welcomes notices of upcoming events and requests that the sponsors send a short description, their name and phone number, and an address to which interested readers may write for further information. Please send notices at least three months before the date of the event, to: Calendar, P.O. Box 5406, Greensboro, NC 27403.* ©

COMMODORE 64
HACKERS ONLY
BOOKWARE FROM ABACUS


THE ANATOMY OF A 1541 DISK DRIVE
MACHINE LANGUAGE FOR THE COMMODORE
THE ANATOMY OF A COMMODORE
64

•••• Unravel the mysteries of the misunderstood floppy disk. 300+ pages of in-depth information. Sequential, relative random files. Many useful utilities. 1541 ROM listing fully commented. **$19.95**

•••• A machine language reference guide specifically to the Commodore 64. All instructions fully explained. With these complete program listings for an ASSEMBLER, DISASSEMBLER & 6510 simulator. 200+ pages. **$19.95**

•••• 300+ page detailed guide to the internals of your favorite computer. Covers graphics, synthesizer, kernal, BASIC. Includes full commented ROM listings. **$19.95**

OTHER TITLES COMING SOON!
ASK FOR FREE CATALOG
FOR QUICK SERVICE
PHONE 616-241-5510

Abacus Software

P.O. Box 7211 Grand Rapids, MI 49510 Add $2.00 postage and handling. Foreign add $4.00. Michigan residence add 4%. MC, VISA, AMEX accepted.

BASF DISKETTES
$1.75 WITH LIBRARY CASE!!!!
DIGITAL CASSETTES
C-10 for 35c (100 Lot)!!!!
FOR YOUR COMPUTER
ALL DISKS AND CASSETTES ARE......
—100% ERROR FREE (Diskettes Fully Certified)—
—LIFETIME GUARANTEED—
****
—CUSTOM LENGTHS AVAILABLE—
COMPUTER TAPE PRICES

	25 LOT	100 LOT	1000 LOT
C-10	.50/12.50	.35/35.00	.30/300.00
C-20	.50/13.75	.40/40.00	.35/350.00
C-30	.60/15.00	.45/45.00	.40/400.00
--------------BASF DPS GRADE TAPES--------------			
C-10	.55/13.75	.40/40.00	.35/350.00
C-20	.60/15.00	.45/45.00	.40/400.00
C-30	.65/16.25	.50/50.00	.45/450.00

****
QUALITY NORELCO CASSETTE CASE...
AND LABEL PRICES
(WITH CASSETTE ORDERS ONLY)

25-249 Cases/.20 Ea.		250/.13 Ea.	1000/.11 Ea.
Labels - Sheet .20	12/.20	120/1.70	1200/14.50
Tractor Feed Cassette Labels (1 up)			1000/14.50

—INTRODUCTORY OFFER—
BASF 5¼ Single Side Double Density Diskettes
with...Hub Ring, Label, Jacket, W/Protect Stickers
—Free Hard Library Case with Every 10 Disks—

10 LOT	20 LOT	50 LOT	100 LOT
2.00/20.00	1.95/39.00	1.80/90.00	1.75/175.00

All Prices Include U.S. Shipping & Handling
—Write For Volume Prices—
—Phone Orders Add $2.50 C.O.D. Fee—
(Canadian Customers May Call or Write for Shipping Costs)
Send Cashier's Checks, Money Orders, & Checks to:

CASS-A-TAPES
BOX 8123-C
KANSAS CITY, MO. 64112
816-444-4651