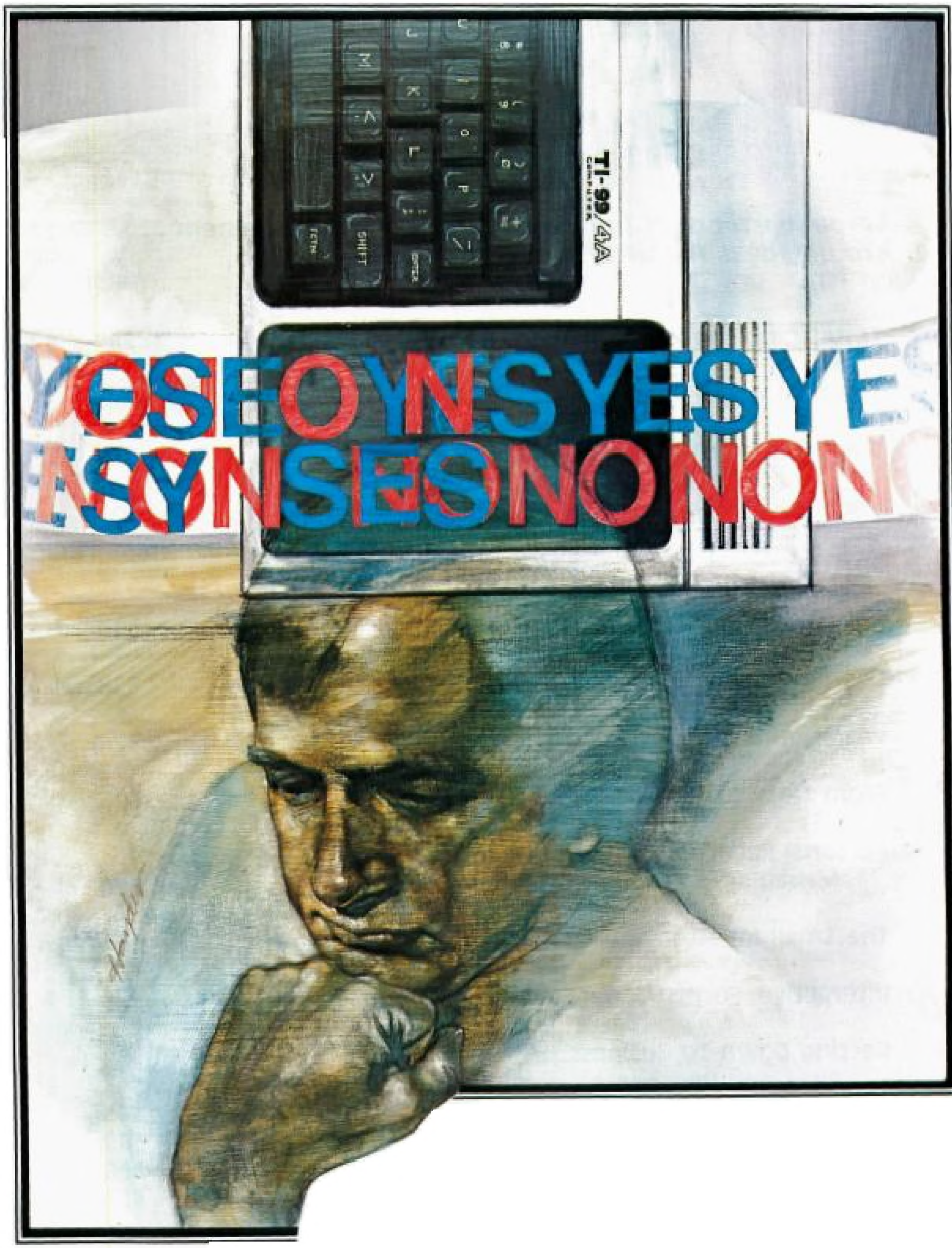
8
Applications and Utilities



8

Applications and Utilities

From bartending, banking, and budget management . . . to big ideas for small businesses.

A MICRO BARTENDER

TI BASIC ON THE ROCKS

Entertaining guests can indeed be a chore—especially when you have to help them decide on the choice of drinks, remember how to correctly mix the selected drinks, and simultaneously explain to your curious visitors exactly how you use the exotic computer in your livingroom. Now, this three-part task can be handled much more enjoyably with *Micro Bartender*—a TI BASIC program.

The next time guests arrive just sit them in front of your home computer and let them choose their own mixed drinks. The program will not only provide easy-to-follow recipes, but will also show your guests how the finished drinks should appear—in full color, with proper glass and garnish!

But what's the use of choosing drinks that are impossible to make because you're missing one or more ingrediants? It's definitely slow and frustrating when the only way to find "possible" drinks is by scanning all the ingredients on page after page of recipes. But happily, this tedious process is now a thing of the past. With *Bartender*'s built-in search routine, you can tell the computer what ingredients are actually on hand, and it will tell you what drinks you can, in fact, make. Then, you can look up the details of each recipe and see a graphic representation of the finished drink's appearance.

Cramming nearly a score of drink recipes (plus the associated graphics) into the TI-99/4's 16K of RAM memory was no easy feat. Observant programmers will notice our extensive use of data reconstruction techniques. For those programmers who happen to be non-drinkers—and debugging alone could drive a man to drink—the program logic and control structure is suitable with many other types of reconstructed "recipes." [Only kidding, of course, about "driving a man to drink. . . ."—Ed.]

EXPLANATION OF THE PROGRAM
Micro Bartender

Line Nos.	
200-240	Prints title screen.
250-290	Subroutine to determine color for graphics.
300-1350	Subroutine for graphics.
1360-1650	Defines special characters.
1660-1750	Reads data while title screen is displayed.
1760-1860	Prints screen of two major options.
1870-2220	First option. Prints two menu screens of the list of drinks, receives user's choice.
2230-2250	Clears screen, sets colors of graphics for drink chosen.
2260-2540	Prints name of drink and type of glass.
2550-2580	Prints amounts and ingredients in recipe.
2590-2650	Prints mixing instructions.
2660-2710	Prints cocktail or whiskey sour glass.
2720-2810	Prints garnish and sets colors for garnish.
2820-2850	Prints instructions for stir rod or straws.
2860-3000	Draws the drink.
3010-3020	User may press any key to continue program.
3030-3090	Second option. Prints instructions for ingredient inventory.
3100-3200	Receives user's input Y or N for each ingredient in INV$ array.
3220-3260	Prints message for no drinks possible.
3270-3370	Compares each drink's ingredients with inventory list and prints possible drinks to make.
3380-3400	User presses any key to go back to option screen.
3410-3780	Data for DRINK$ array of attributes for each drink.
3790-3810	Names of ingredients for inventory list.

Copyright © 1983 Emerald Valley Publishing Co.

```
100 REM ** BARTENDER **
110 REM
120 REM
170 CALL SCREEN(8)
180 CALL CLEAR
190 DIM DRINK$(19,23),INV$(15,1)
200 PRINT "** BARTENDER **"::::::::::
210 CALL COLOR(2,7,16)
220 C=7
230 GOSUB 250
240 GOTO 1360
250 CALL COLOR(14,13,C)
260 CALL COLOR(11,2,C)
270 CALL COLOR(16,11,C)
280 CALL COLOR(12,C,C)
290 RETURN
300 REM NORMAL GLASS
310 CALL HCHAR(4,23,105,5)
320 CALL HCHAR(13,23,104,5)
330 CALL VCHAR(5,22,96,8)
340 CALL VCHAR(5,28,103,8)
350 CALL HCHAR(13,22,106)
360 CALL HCHAR(13,28,107)
370 CALL HCHAR(4,28,108)
380 CALL HCHAR(4,22,109)
390 FOR I=12 TO 7 STEP -1
400 FOR J=27 TO 23 STEP -1
410 CALL HCHAR(I,J,120)
420 NEXT J
430 NEXT I
440 RETURN
450 REM COCKTAIL GLASS
460 CALL HCHAR(6,21,129)
470 FOR I=1 TO 3
480 CALL HCHAR(6+I,21+I,113)
490 NEXT I
500 FOR I=1 TO 3
510 CALL HCHAR(6+I,20+I,102)
520 NEXT I
530 CALL HCHAR(6,28,128)
540 FOR I=1 TO 3
550 CALL HCHAR(6+I,28-I,112)
560 NEXT I
570 FOR I=1 TO 3
580 CALL HCHAR(6+I,29-I,97)
590 NEXT I
600 CALL HCHAR(10,24,100)
610 CALL HCHAR(10,25,101)
620 CALL VCHAR(11,24,96,3)
630 CALL VCHAR(11,25,103,3)
640 CALL HCHAR(14,23,104,4)
650 CALL HCHAR(5,21,105,8)
660 CALL HCHAR(7,23,120,4)
670 CALL HCHAR(8,24,120,2)
680 RETURN
690 REM TALL GLASS
700 CALL HCHAR(4,23,105,4)
710 CALL HCHAR(15,23,104,4)
720 CALL VCHAR(5,22,96,10)
730 CALL VCHAR(5,27,103,10)
740 CALL HCHAR(4,22,109)
750 CALL HCHAR(4,27,108)
760 CALL HCHAR(15,22,106)
770 CALL HCHAR(15,27,107)
780 FOR I=14 TO 7 STEP -1
790 FOR J=26 TO 23 STEP -1
800 CALL HCHAR(I,J,120)
810 NEXT J
820 NEXT I
830 RETURN
840 REM OLD-FASH GLASS
850 CALL HCHAR(4,23,105,7)
860 CALL HCHAR(11,23,104,7)
870 CALL VCHAR(5,22,96,6)
880 CALL VCHAR(5,30,103,6)
890 CALL HCHAR(11,22,106)
900 CALL HCHAR(11,30,107)
910 CALL HCHAR(4,30,108)
920 CALL HCHAR(4,22,109)
930 FOR I=10 TO 7 STEP -1
940 FOR J=29 TO 23 STEP -1
950 CALL HCHAR(I,J,120)
960 NEXT J
970 NEXT I
980 RETURN
990 REM PONY GLASS
1000 CALL VCHAR(5,22,96,5)
1010 CALL VCHAR(5,27,103,5)
1020 CALL HCHAR(9,26,112)
1030 CALL HCHAR(10,25,112)
1040 CALL HCHAR(10,26,97)
1050 CALL HCHAR(9,23,113)
1060 CALL HCHAR(10,24,113)
1070 CALL HCHAR(10,23,102)
1080 CALL HCHAR(11,24,100)
1090 CALL HCHAR(11,25,101)
1100 CALL VCHAR(12,24,96,3)
1110 CALL VCHAR(12,25,103,3)
1120 CALL HCHAR(15,23,104,4)
1130 CALL HCHAR(4,23,105,4)
1140 CALL HCHAR(4,22,109)
1150 CALL HCHAR(4,27,108)
1160 CALL HCHAR(9,24,120,2)
1170 CALL HCHAR(8,23,120,4)
1180 CALL HCHAR(7,23,120,4)
1190 RETURN
1200 REM SPK OLIVE OR CHERRY
1210 CALL HCHAR(7,24,150)
1220 CALL HCHAR(6,23,98)
1230 RETURN
1240 REM LEMON TWIST
1250 CALL HCHAR(7,24,152)
1260 CALL HCHAR(7,25,153)
1270 RETURN
1280 REM ORANGE
1290 CALL COLOR(16,12,8)
1300 CALL HCHAR(5,22,157)
1310 CALL HCHAR(5,23,154)
1320 CALL HCHAR(6,22,155)
1330 CALL HCHAR(6,23,156)
1340 CALL HCHAR(4,21,98)
1350 RETURN
1360 CALL CHAR(152,"7F7F7F7F")
1370 CALL CHAR(110,"030303030303FFFF")
1380 CALL CHAR(158,"0000000C8CDFFFF")
1390 CALL CHAR(153,"FEFEFEFE")
1400 CALL CHAR(112,"0103070F1F3F7FFF")
1410 CALL CHAR(128,"0103070F1F3F7FFF")
1420 CALL CHAR(97,"FFFEFCF8F0E0C080")
1430 CALL CHAR(150,"3C7EFFFFFFFF7E3C")
1440 CALL CHAR(99,"2424242424242424")
1450 CALL CHAR(100,"FF7F3F1F0F070303")
1460 CALL CHAR(101,"FFFEFCF8F0E0C0C")
1470 CALL CHAR(102,"FF7F3F1F0F070301")
1480 CALL CHAR(113,"80C0E0F0F8FCFEFF")
1490 CALL CHAR(129,"80C0E0F0F8FCFEFF")
1500 CALL CHAR(106,"030303")
1510 CALL CHAR(107,"C0C0C0")
1520 CALL CHAR(108,"0000000000C0C")
1530 CALL CHAR(109,"0000000000000303")
1540 CALL CHAR(104,"FFFFFFFFFF")
1550 CALL CHAR(96,"0303030303030303")
1560 CALL CHAR(103,"C0C0C0C0C0C0C0C")
1570 CALL CHAR(105,"000000000000FFFF")
1580 GOSUB 460
1590 CALL CHAR(136,"3F7FFFFCFCFF7F3F")
1600 CALL CHAR(137,"FCFEFF3F3FFFFEFC")
1610 CALL CHAR(98,"8040201008040201")
1620 CALL CHAR(157,"0F0F3030C8C4C2C1")
1630 CALL CHAR(154,"F0F00C0C13234383")
1640 CALL CHAR(155,"C1C2C4C83030000F0F")
1650 CALL CHAR(156,"83432313 0C0CF0F0")
1660 FOR I=1 TO 19
1670 CALL COLOR(2,16,7)
1680 FOR J=0 TO 23
1690 READ DRINK$(I,J)
1700 NEXT J
1710 CALL COLOR(2,7,16)
```


Copyright © 1983 Emerald Valley Publishing Co.

```
1720 NEXT I
1730 FOR I=0 TO 15
1740 READ INV$(I,0)
1750 NEXT I
1760 CALL CLEAR
1770 CALL COLOR(2,2,1)
1780 PRINT "DO YOU:":::"(1) WANT TO SEE
THE RECIPE"
1790 PRINT "    FOR A SPECIFIC DRINK?"
1800 PRINT ::"(2) WANT TO KNOW WHAT YOU
"
1810 PRINT "    CAN MAKE WITH THE"
1820 PRINT "    INGREDIENTS YOU HAVE?":
::::::::
1830 CALL SOUND(150,1397,2)
1840 CALL KEY(0,K,S)
1850 IF (K<49)+(K>50)THEN 1840
1860 ON K-48 GOTO 1870,3030
1870 CALL CLEAR
1880 PRINT "DRINK:"::
1890 PRINT :"    (1) MARTINI"
1900 PRINT :"    (2) DRY MARTINI"
1910 PRINT :"    (3) EXTRA DRY MARTINI"
1920 PRINT :"    (4) VODKA MARTINI"
1930 PRINT :"    (5) MANHATTAN"
1940 PRINT :"    (6) DRY MANHATTAN"
1950 PRINT :"    (7) SWEET MANHATTAN"
1960 PRINT :"    (8) PERFECT MANHATTAN"
1970 PRINT :"    (9) WHISKEY SOUR"
1980 PRINT :"    (C) CONTINUE"
1990 CALL SOUND(150,1397,2)
2000 CALL KEY(0,K,S)
2010 IF K=67 THEN 2050
2020 IF (K<49)+(K>57)THEN 2000
2030 II=K-48
2040 GOTO 2230
2050 CALL CLEAR
2060 PRINT "DRINK:"
2070 PRINT :"    (0) WARD EIGHT"
2080 PRINT :"    (1) DAIQUIRI"
2090 PRINT :"    (2) BACARDI"
2100 PRINT :"    (3) SCREWDRIVER"
2110 PRINT :"    (4) PINK LADY"
2120 PRINT :"    (5) SALTY DOG"
2130 PRINT :"    (6) GIN COOLER"
2140 PRINT :"    (7) TOM COLLINS"
2150 PRINT :"    (8) BLACK RUSSIAN"
2160 PRINT :"    (9) OLD-FASHIONED"
2170 PRINT :"    (R) REPEAT FIRST SCREEN"
2180 CALL SOUND(150,1397,2)
2190 CALL KEY(0,K,S)
2200 IF K=82 THEN 1870
2210 IF (K<48)+(K>57)THEN 2190
2220 II=K-38
2230 CALL CLEAR
2240 C=VAL(DRINK$(II,1))
2250 GOSUB 250
2260 PRINT "**";DRINK$(II,0)::::
2270 ON VAL(DRINK$(II,2))GOTO 2280,2310
,2340,2370,2420,2450
2280 PRINT "FILL MIXING GLASS"
2290 PRINT "2/3 FULL WITH ICE CUBES"
2300 GOTO 2550
2310 PRINT "FILL HIGHBALL GLASS"
2320 PRINT "FULL WITH ICE CUBES"
2330 GOTO 2550
2340 PRINT "SALT RIM OF HIGHBALL GLASS"
2350 PRINT "FILL WITH ICE"
2360 GOTO 2550
2370 PRINT "FILL TALL FROSTED"
2380 PRINT "GLASS WITH ICE"
2390 PRINT "SQUEEZE 1/4 LIME"
2400 PRINT "OVER ICE-DROP IN"
2410 GOTO 2550
2420 PRINT "FILL OLD-FASHIONED GLASS"
2430 PRINT "WITH ICE"
2440 GOTO 2550
2450 PRINT "1 CUBE OF SUGAR IN"
2460 PRINT "OLD-FASHIONED GLASS"
2470 PRINT "2-3 DROPS ANGOSTURA BITTERS
"
2480 PRINT "OVER CUBE OF SUGAR"
2490 PRINT "RIM GLASS WITH LEMON TWIST"
2500 PRINT "DROP IN":"1/2 OZ. SODA OR W
ATER"
2510 PRINT "MUDDLE THOROUGHLY"
2520 PRINT "FILL WITH ICE"
2530 PRINT "1 OZ. BOURBON":"STIR"
2540 GOTO 2720
2550 FOR JJ=8 TO 23
2560 IF DRINK$(II,JJ)="" THEN 2580
2570 PRINT DRINK$(II,JJ);INV$(JJ-8,0)
2580 NEXT JJ
2590 ON VAL(DRINK$(II,3))GOTO 2720,2600
,2620,2640
2600 PRINT "STIR AND STRAIN INTO"
2610 GOTO 2660
2620 PRINT "SHAKE AND STRAIN INTO"
2630 GOTO 2660
2640 PRINT "STIR LIGHTLY"
2650 GOTO 2720
2660 ON VAL(DRINK$(II,4))GOTO 2720,2670
,2690,2710
2670 PRINT "3 OZ. CHILLED COCKTAIL GLAS
S"
2680 GOTO 2720
2690 PRINT "5 OZ. CHILLED COCKTAIL GLAS
S"
2700 GOTO 2720
2710 PRINT "WHISKEY SOUR GLASS"
2720 ON VAL(DRINK$(II,5))GOTO 2800,2730
,2760,2790
2730 PRINT "GARNISH WITH SPIKED OLIVE"
2740 CALL COLOR(15,13,C)
2750 GOTO 2800
2760 PRINT "GARNISH WITH SPIKED CHERRY"
2770 CALL COLOR(15,10,C)
2780 GOTO 2800
2790 PRINT "GARNISH WITH LEMON TWIST"
2800 IF DRINK$(II,6)="1" THEN 2820
2810 PRINT "AND ORANGE SLICE"
2820 ON VAL(DRINK$(II,7))GOTO 2860,2830
,2850
2830 PRINT "SERVE WITH STIR ROD"
2840 GOTO 2860
2850 PRINT "SERVE WITH TWO STRAWS"
2860 IF DRINK$(II,2)="1" THEN 2890
2870 ON VAL(DRINK$(II,2))-1 GOSUB 310,3
10,700,850,850
2880 GOTO 2900
2890 ON VAL(DRINK$(II,4))-1 GOSUB 460,4
60,1000
2900 ON VAL(DRINK$(II,5))GOSUB 290,1210
,1210,1250
2910 IF DRINK$(II,6)="1" THEN 2930
2920 GOSUB 1290
2930 ON VAL(DRINK$(II,7))GOTO 2980,2940
,2970
2940 CALL VCHAR(3,26,96,4)
2950 CALL HCHAR(4,26,110)
2960 GOTO 2980
2970 CALL VCHAR(3,26,99,4)
2980 IF DRINK$(II,2)<>"3" THEN 3010
2990 CALL COLOR(16,16,8)
3000 CALL HCHAR(4,23,158,5)
3010 CALL KEY(0,K,S)
3020 IF S=0 THEN 3010 ELSE 1760
3030 CALL CLEAR
3040 PRINT "IN THE FOLLOWING LIST,"
3050 PRINT "PRESS ""Y"" IF YOU HAVE"
3060 PRINT "THE INGREDIENT."
3070 PRINT "PRESS ""N"" IF YOU DO NOT."
3080 PRINT :"PRESS ""B"" TO BACK UP":::
:::
3090 CALL SOUND(150,1397,2)
3100 YS=0
3110 FOR KK=0 TO 15
3120 PRINT "_ ";INV$(KK,0)
```

Copyright © 1983 Emerald Valley Publishing Co.

```
3130 CALL KEY(0,KEY,S)
3140 IF KEY=66 THEN 3030
3150 IF KEY=78 THEN 3180
3160 IF KEY<>89 THEN 3130
3170 YS=YS+1
3180 CALL HCHAR(23,3,KEY)
3190 INV$(KK,1)=CHR$(KEY)
3200 NEXT KK
3210 DR=0
3220 PRINT ::"YOU CAN MAKE:"::
3230 IF YS>1 THEN 3270
3240 PRINT "NOTHING; SORRY.":::"YOU NEED TO GO TO THE LIQUOR"
3250 PRINT "STORE IF YOU'RE THIRSTY."
3260 GOTO 3380
3270 FOR I=1 TO 19
3280 FOR J=8 TO 23
3290 IF DRINK$(I,J)="" THEN 3310
3300 IF INV$(J-8,1)="N" THEN 3350
3310 NEXT J
3320 PRINT DRINK$(I,0)
3330 CALL SOUND(150,1397,2)
3340 DR=DR+1
3350 NEXT I
3360 IF DR=0 THEN 3240
3370 PRINT :"THAT'S ALL."
3380 PRINT :"PRESS ANY KEY TO CONTINUE"
3390 CALL KEY(0,K,S)
3400 IF S=0 THEN 3390 ELSE 1760
3410 DATA MARTINI,16,1,2,2,2,1,1
3420 DATA "1 OZ.",,,"1/3 OZ.",,,,,,,,,,,,""
3430 DATA "DRY MARTINI",16,1,2,2,2,1,1
3440 DATA "1 1/4 OZ.",,,"1/4 OZ.",,,,,,,,,,,""
3450 DATA "EXTRA DRY MARTINI",16,1,2,2,2,1,1
3460 DATA "1 1/2 OZ.",,,"2-3 DROPS",,,,,,,,,,,,""
3470 DATA "VODKA MARTINI",16,1,2,2,2,1,1
3480 DATA "","1 OZ.",,"1/3 OZ.",,,,,,,,,,,,""
3490 DATA MANHATTAN,7,1,2,2,3,1,1
3500 DATA "",,"1 OZ.",,,"1/2 OZ.",,,,"2-3 DROPS",,,,,,""
3510 DATA "DRY MANHATTAN",7,1,2,2,2,1,1
3520 DATA "",,"1 OZ.","1/2 OZ.",,,,,,,,,,,,""
3530 DATA "SWEET MANHATTAN",7,1,2,2,3,1,1
3540 DATA "",,"1 OZ.",,,"1/2 OZ.",,,"1/4 OZ.","2-3 DROPS",,,,,,""
3550 DATA "PERFECT MANHATTAN",7,1,2,2,4,1,1
3560 DATA "",,"1 OZ.","1/4 OZ.",,"1/4 OZ.",,,,"2-3 DROPS",,,,,,""
3570 DATA "WHISKEY SOUR",12,1,3,4,3,2,1
3580 DATA "",,"1 OZ.",,,,,"1 OZ.","1/2 OZ.",,,,,,,""
3590 DATA "WARD EIGHT",7,1,3,4,3,2,1
3600 DATA "",,"1 OZ.",,,,,"1 OZ.",,,"1/2 OZ.",,,,,""
3610 DATA "DAIQUIRI",16,1,3,3,1,1,1
3620 DATA "",,,,"1 OZ.",,,"1 OZ.","1/2 OZ.",,,,,,,""
3630 DATA BACARDI,7,1,3,3,1,1,1
3640 DATA "",,,,"1 OZ.",,,"1 OZ.",,,"1/2 OZ.",,,,,""
3650 DATA SCREWDRIVER,11,2,1,1,1,1,2
3660 DATA "",,"1 OZ.",,,,,,,,,,,"FILL WITH",,,""
3670 DATA "PINK LADY",10,1,3,3,1,1,3
3680 DATA "1 OZ.",,,,,,,,,,"1/2 OZ.",,,,,"1 1/2 OZ."
3690 DATA "SALTY DOG",16,3,4,1,1,1,2
3700 DATA "1 OZ.",,,,,,,,,,,,"FILL WITH",,,,""
3710 DATA "GIN COOLER",7,4,4,1,3,2,3
3720 DATA "1 OZ.",,,,,,,"1 OZ.",,,"1/2 OZ.",,,"FILL WITH",,""
3730 DATA "TOM COLLINS",16,4,4,1,3,2,3
3740 DATA "1 OZ.",,,,,,,"1 OZ.","1/2 OZ.",,,,,,"FILL WITH",""
3750 DATA "BLACK RUSSIAN",2,5,4,1,1,1,1
3760 DATA "",,"1 OZ.",,,,,"1/2 OZ.",,,,,,,,""
3770 DATA "OLD-FASHIONED",7,6,1,1,3,2,1
3780 DATA "",,"1 OZ.",,,,,,,"2-3 DROPS",,,,,,"1/2 OZ.",""
3790 DATA GIN,VODKA,BOURBON,"DRY VERMOUTH","LIGHT RUM","SWEET VERMOUTH",KAHLUA
3800 DATA "LEMON JUICE","SIMPLE SYRUP","ANGOSTURA BITTERS",GRENADINE,"GRAPEFRUIT JUICE"
3810 DATA "ORANGE JUICE","GINGER ALE","SODA","MILK OR CREAM"
```


Copyright © 1983 Emerald Valley Publishing Co.

THE RULE OF

"Why Mr. Templeton, you can't figure that!" said the lady at the finance company. I had merely asked her the formula for computing the payoff amount on the installment contract on my 1978 Datsun.

This emphatic "can't do" sent me racing off to the library in my soon-to-be-liberated Datsun. And it was there that I discovered the existence of the *Rule of 78.* So, armed with this knowledge, I decided to write a program that applied the Rule to installment contracts and let my TI-99/4A do the figuring for me.

From the name of this article you might have expected some sort of game, but the Rule of 78 is no game. It determines the amount of money required to pay off an installment contract at any given time, or the amount to be re-financed when you trade in before making all the payments. Should you be so unfortunate and have to default, the Rule of 78 determines the balance that becomes due and payable—the amount the finance company would be entitled to recover by repossessing the car. This Rule also is the method recognized by the Internal Revenue Service for computing the portion of the finance charge deductible each year during the life of the contract.

The Rule of 78 defines the fraction of the total finance charge that is on the unused portion. The numerator of the fraction is the sum of the numbers of the remaining payments; the denominator is the sum of the numbers of all payments. The number of the first payment is equal to the number of payments in the contracts—e.g., 48 payments for a four-year contract. The number of each succeeding payment is one less; the last payment is number 1. At the time the Rule got its name, 12-payment contracts were the usual type. The sum of 12, 11,. . ., and 1 is 78, the denominator of the fraction. A more appropriate name in our day would be rule of 1176, which is the sum of 48 through 1.

Many installment contracts allow an *acquisition charge* to be deducted from the finance charge before multiplying it by the fraction. This is almost a prepayment penalty, but not quite—because you usually pay *only a portion* of the acquisition charge. When applicable, the acquisition charge affects the payoff amount of the contract.

The Rule of 78 is also known as the Sum of the Monthly Balances Method and the Sum of the Months Digits Method. According to the *Consumer and Commercial Credit Installment Sales,* a subscription service published by Prentice-Hall, it is widely used in installment contracts. From these volumes, which contain federal and state law on the subject, I discovered that the Rule is required by law in some states and allowed by law in all states. It applies to installment contracts on automobiles, furniture, and appliances, and to some types of loans. Internal Revenue Service Publication 545, *Interest Expense,* explains the Rule and its application to income tax deductions.

Running the Program

The program is shown in the listing at the end of this article. It is written in TI BASIC, but will also run in TI Extended BASIC. Copy the program into your computer and enter the RUN command.

Consult a copy of the contract. First, be sure it mentions the Rule of 78 or one of its aliases in the section on prepayment. Then locate the amounts requested in the initial display. All of the amounts are usually typed in except the acquisition charge; it is printed in the contract. The display is as follows:

```
          INSTALLMENT PAYMENTS
AMOUNT FINANCED: $
FINANCE CHARGE: $
ACQUISITION CHARGE: $
AMOUNT OF PAYMENT: $
NUMBER OF PAYMENTS:
FIRST PAYMENT DATE:
```

The prompts of the displays are typical of the names used in contracts. The amount financed is the sum of the price of the merchandise, sales taxes, insurance, etc., less the down payment. The finance charge is the amount added to the amount financed to compute the total of payments. The acquisition charge is printed in the section on prepayment. (It is $25 in many contracts.) It is easy to come up with the amount of payment: That's the amount you pay each month. Typically, you make 12 payments on appliances and 48 on new cars. Enter the date of the first payment expressed as three numbers separated by slashes. The first number represents the month, 1 through 12. The second is the day of the month, 1 through 31. The last number is the year, represented by the last two digits. For example, if the first payment were due December 23, 1984, you would enter 12/23/84.

After you enter the figures, the program lists the options as follows:

```
CHOOSE ONE
  1. CONTRACT SCHEDULE
  2. CONTRACT STATUS
  3. TAX DEDUCTION
  4. NEW CONTRACT
ENTER NUMBER:
```

The Contract Schedule option provides the date, total paid, balance prepay amount, and amount saved by prepaying for the first payment. By pressing ENTER you request the next payment. By repeatedly pressing ENTER you can display these five items for each payment of the contract. On the display for December of each year, the program also displays the tax deduction for the year.

When you specify the Contract Status option, the program requests a date. The program then displays the status of the contract on that date. If the date is during the period of the contract, the status display includes the date, total paid, balance, prepay amount, amount saved

Copyright © 1983 Emerald Valley Publishing Co.

by prepayment, and the tax deduction for the year if the contract is prepaid on that date. The status figures, of course, apply only if all payments have been made up to the requested date.

The Tax Deduction option shows you the allowable income tax deduction for each year of the contract. This same information is provided in the contract schedule displays; because this option gives you *only* the tax deduction, it is much faster. In many cases, prepayment is not possible, but deducting the proper portion of the finance charge is important.

The New Contract option returns to the beginning of the program and requests the inputs previously described. If you were really into installment contracts, you could compute the figures for the contract on your car, then on your TV, etc. Option 4 would enable you to enter figures for each additional contract.

If you select option 1, 2, or 3, the program lists the values you entered at the top of the screen, as follows:

```
AMOUNT FINANCED: $2,545.73
FINANCE CHARGE: $    781.03
ACQUISITION CHARGE:$ 25.00
AMOUNT OF PAYMENT:$ 92.41
NUMBER OF PAYMENT:     36
FIRST PAYMENT:       12/23/80
```

Below this display, the specific display for the selected option appears. For the Contract Schedule option, the following display is repeated for each payment:

```
          CONTRACT SCHEDULE
AFTER PAYMENT ON 12/23/80
TOTAL PAID        $      92.41
BALANCE           $   3,234.35
PREPAY AMOUNT     $   2,558.92
SAVE BY PREPAY    $     675.43
DEDUCTION FOR 1980 $42.22
```

For the Contract Status option, the initial display requests the date, as follows:

```
CONTRACT STATUS
ENTER DATE:
```

Enter a date in the format previously described. If you enter a date before the month of the first payment, the following is displayed:

```
STATUS ON 11/30/80
TOO EARLY
```

On the other hand, if you enter a date later than the last day of the month in which you will make the last payment, the following is displayed:

```
STATUS ON 12/1/83
PAID UP
```

When you enter a date during the period of the contract, the following is displayed:

```
STATUS ON 12/31/81:
TOTAL PAID        $   1,201.33
BALANCE           $   2,125.43
PREPAY AMOUNT     $   1,838.23
SAVE BY PREPAY    $     287.20
DEDUCTIBLE IN 81  $     451.61
IF PAID OFF ON 12/31/81
```

For the tax deduction option, the display is as follows:

```
IF YOU PAY ALL PAYMENTS
AS SCHEDULED, YOU MAY
DEDUCT FINANCE CHARGE
AS FOLLOWS:

   YEAR                  AMOUNT
   1980         $         42.22
   1981         $        415.14
   1982         $        246.27
   1983         $         77.40
```

At the bottom of each screen, the program displays the following message;

```
PRESS ENTER TO CONTINUE
OR 9 TO QUIT
```

For the Contract Schedule option, you get the figures for the next payment when you press ENTER. When all payments have been displayed, pressing ENTER displays the list of options previously described. For options 2 and 3, which have one screen each, pressing ENTER displays the option list.

The accuracy of the figures depends on the accuracy of the computer. Texas Instruments claims ten digits of accuracy for the TI-99/4A. In the case of the contract on my 1978 Datsun, the finance company's figures were not exactly the same as mine. The differences were a penny or two, most likely due to differences in computer accuracy. Of course, I paid the amount *their* computer wanted.

Changing the Program

If you have a printer, you will want to change the program to print the data displayed on the screen and you will probably want to change the format as well. The contract schedule can be printed in tabular form, one line per payment, on an 80-column printer.

The program has a subroutine for each option, but you may not want all the options; if not, you can leave one or two out. The contract schedule subroutine begins on line 680, and ends on line 1540. The contract status subroutine begins on line 1560 and continues through line 2280. The tax deduction subroutine occupies lines 2300 through 2650. Each subroutine is independent of the other two; however, the driver (lines 170 through 660) and the miscellaneous subroutines from line 2670 to the end of the program are required for all subroutines.

Streamlining for TI Extended BASIC

You can run the program in Extended BASIC as it is, or you can streamline it, exploiting some of the features of the more powerful language. The power of the DISPLAY statement of Extended BASIC is particularly valuable in this program.

Line 170 is a DEF statement that defines a rounding function. A format defined by an IMAGE statement automatically rounds fractions, and this function is used to align decimal points in the displays. The function is not needed if you use a specified format.

The subroutine beginning at line 2880 displays a string at a defined point on the screen. When you use a

Copyright © 1983 Emerald Valley Publishing Co.

DISPLAY statement with the AT option, this subroutine is not required. Similarly, the subroutine at line 2940 adds zeros to the right of the decimal point, where required. It also inserts a comma between the hundreds and thousands digit of numbers greater than 999.99. A defined format adds least significant zeros but does not insert the comma. If you want to use a format and give up the comma, omit this subroutine.

To incorporate these changes, modify the program shown in Listing 1 by performing the following steps:

1. Omit line 170 and modify line 180 as follows:
   180 IMAGE" #####.##"
2. Omit lines 490 and 500; modify line 510 as follows:
   510 PRINT USING "AMOUNT FINANCED : :$#####.##":UB
3. Omit lines 520 and 530; modify line 540 as follows:
   540 PRINT USING "FINANCE CHARGE :$ #####.##":FC
4. Omit lines 550 and 560; modify line 570 as follows:
   570 PRINT USING "ACQUISITION CHARGE: $####.##":AC
5. Omit lines 580 and 590; modify line 600 as follows:
   600 PRINT USING "AMOUNT OF PAYMENT: $####.##":PMNT
6. Omit line 630 and modify line 640 as follows:
   640 PRINT USING "FIRST PAYMENT: ##/##/##" ;MO,DA,YR
7. Omit lines 810, 830, and 840; modify line 850 as follows:
   850 DISPLAY AT (14, 17):USING "##/##/##": CMO, DA,CYR
8. Omit lines 880-910; modify line 930 as follows:
   930 DISPLAY AT(15, 17):USING 180:TOTPD
9. Omit lines 950-990; modify line 1000 as follows:
   1000 DISPLAYS AT(16, 17):USING 180:BAL
10. Omit lines 1080-1110; modify line 1130 as follows:
    1130 DISPLAY AT(17, 17):USING 180:PREPAY
11. Omit lines 1140-1170; modify line 1190 as follows:
    1190 DISPLAY AT(18, 17):USING 180:SAV
12. Omit lines 1280-1310 and 1330; modify line 1340 as follows:
    1340 DISPLAY AT(20,1):USING "DEDUCTION FOR 19## $###.##":CYR,ADED
13. Omit lines 1430-1460 and 1480; modify line 1490 as follows:
    1490 DISPLAY AT(20,1):USING "DEDUCTION FOR 19## $####.##":CYR,ADED
14. Omit lines 1830 and 1840; modify line 1850 as follows:
    1850 PRINT USING "TOTAL PAID $ #####.## ":TOTPD
15. Omit lines 1880 and 1890; modify line 1900 as follows:
    1900 PRINT USING "BALANCE $#####.##" :BAL
16. Omit lines 1970 and 1980; modify line 1990 as follows:
    1990 PRINT USING "PREPAY AMOUNT $ #####.## ":PREPAY
17. Omit lines 2000 and 2010; modify line 2020 as follows:
    2020 PRINT USING "SAVE BY PREPAY $ #####.## ":SAV
18. Omit lines 2140 and 2150; modify line 2160 as follows:
    2160 PRINT USING "DEDUCTIBLE IN ## $ #####.##" :SYR,DEDUCT
19. Omit lines 2440 and 2450; modify line 2430 as follows;
    2430 PRINT USING "19## $#####.##" :DYR,DED
20. Omit lines 2590 and 2600; modify line 2580 as follows:
    2580 PRINT USING "19## $#####.##" :DYR,DED
21. Omit lines 2690-2710 and modify line 2720 as follows:
    2720 DISPLAY AT(23,1):"PRESS ENTER TO CONTINUE"
22. Omit lines 2730 and 2740; modify line 2750 as follows:
    2750 DISPLAY AT(24,1):"OR 9 TO QUIT"
23. Remove references to function RND2 in the following lines:

```
1050 SAV=RUL 78 (AFC)
1270 ADED=DEDUCT/DEN*FC
1400 SAV=X
1420 ADED=DEDUCT/DEN*FC
1940 SAV=RUL 78(AFC)
2130 DEDUCT=DEDUCT-RUL78(FC)
2200 SAV=1/DEN*AFC
2420 DED=RUL78(FC)
2510 DED=NP/DEN*FC
2570 DED=RUL78(FC)
2640 DED=1/DEN*FC
```

The subroutine beginning at line 2810 is not required if an ACCEPT statement is used to input the character. Omit line 2770 and modify lines 2760 and 2780 as follows:

```
2760 ACCEPT AT(24,14):SEL$
2780 IF SEL$="9" THEN 2800
```

And don't forget to omit the subroutines (lines 2810-3070). Extended BASIC allows further compression by putting several statements on the same line and by using statements in IF-THEN-ELSE statements. However, the changes I have suggested provide a significant reduction in the size of the program.

With this program in your computer, you have all the secrets of the Rule of 78 at your disposal. Your computer will tell you everything you ever wanted to know about an installment contract, but didn't ask because you would not have been told.

Copyright © 1983 Emerald Valley Publishing Co.

```
100 REM    ********************
110 REM    *        PAYMENTS:   *
120 REM    *        RULE OF 78  *
130 REM    ********************
140 REM
150 REM
160 REM    DEFINE FUNCTIONS
170 DEF RND2(X)=INT(X*100+.5)/100
180 DEF INC=28-LEN(X$)
190 DEF RUL78(D)=P*Q/2/DEN*D
200 REM   INPUT DATA ********
210 CALL CLEAR
220 PRINT "      INSTALLMENT PAYMENTS"::
230 INPUT "AMOUNT FINANCED: $":UB
240 INPUT "FINANCE CHARGE: $":FC
250 INPUT "ACQUISITION CHARGE: $":AC
260 AFC=FC-AC
270 INPUT "AMOUNT OF PAYMENT: $":PMNT
280 INPUT "NUMBER OF PAYMENTS: ":NP
290 INPUT "FIRST PAYMENT DATE: ":DATE$
300 N=POS(DATE$,"/",1)
310 MO$=SEG$(DATE$,1,N-1)
320 M=POS(DATE$,"/",N+1)
330 L=M-N-1
340 DA$=SEG$(DATE$,N+1,L)
350 YR$=SEG$(DATE$,M+1,2)
360 MO=VAL(MO$)
370 DA=VAL(DA$)
380 YR=VAL(YR$)
390 DEN=NP*(NP+1)/2
400 PRINT ::"CHOOSE ONE:"::
410 PRINT " 1 CONTRACT SCHEDULE"
420 PRINT " 2 CONTRACT STATUS"
430 PRINT " 3 TAX DEDUCTION"
440 PRINT " 4 NEW CONTRACT"::
450 INPUT "ENTER NUMBER: ":SEL
460 IF SEL=4 THEN 210
470 IF (SEL<1)+(SEL>3)<0 THEN 400
480 CALL CLEAR
490 X$=STR$(UB)
500 GOSUB 2940
510 PRINT "AMOUNT FINANCED: $";TAB(INC);X$
520 X$=STR$(FC)
530 GOSUB 2940
540 PRINT "FINANCE CHARGE: $";TAB(INC);X$
550 X$=STR$(AC)
560 GOSUB 2940
570 PRINT "ACQUISITION CHARGE:";TAB(20);"$";TAB(INC);X$
580 X$=STR$(PMNT)
590 GOSUB 2940
600 PRINT "AMOUNT OF PAYMENT:";TAB(20);"$";TAB(INC);X$
610 X$=STR$(NP)
620 PRINT "NUMBER OF PAYMENTS:";TAB(26);X$
630 X$=DATE$
640 PRINT "FIRST PAYMENT:";TAB(INC);DATE$
650 ON SEL GOSUB 680,1560,2300
660 GOTO 400
670 REM  DISPLAY SCHEDULE **
680 PRINT ::"        CONTRACT SCHEDULE"::
690 PRINT "AFTER PAYMENT ON"
700 PRINT "TOTAL PAID          $"
710 PRINT "BALANCE             $"
720 PRINT "PREPAY AMOUNT       $"
730 PRINT "SAVE BY PREPAY      $"::::::
740 CYR=YR
750 CMO=MO
760 TOTPD=0
770 DEDUCT=0
780 BAL=UB+FC
790 N=0
800 FOR I=NP TO 1 STEP -1
810 X$=STR$(CMO)&"/"&DA$&"/"&STR$(CYR)
820 CALL HCHAR(14,21,32,9)
830 C=INC+1
840 R=14
850 GOSUB 2880
860 CMO=CMO+1
870 TOTPD=TOTPD+PMNT
880 X$=STR$(TOTPD)
890 GOSUB 2940
900 C=INC+1
910 R=15
920 CALL HCHAR(15,19,32,11)
930 GOSUB 2880
940 BAL=BAL-PMNT
950 X$=STR$(BAL)
960 GOSUB 2940
970 C=INC+1
980 R=16
990 CALL HCHAR(16,19,32,11)
1000 GOSUB 2880
1010 P=I-2
1020 IF P=1 THEN 1390
1030 IF P<0 THEN 1370
1040 Q=P+1
1050 SAV=RND2(RUL78(AFC))
1060 IF SAV<1 THEN 1370
1070 PREPAY=BAL-SAV
1080 R=17
1090 X$=STR$(PREPAY)
1100 GOSUB 2940
1110 C=INC+1
1120 CALL HCHAR(17,19,32,11)
1130 GOSUB 2880
1140 R=18
1150 X$=STR$(SAV)
1160 GOSUB 2940
1170 C=INC+1
1180 CALL HCHAR(18,19,32,11)
1190 GOSUB 2880
1200 N=N+1
1210 DEDUCT=DEDUCT+I
1220 IF CMO>12 THEN 1420
1230 GOSUB 2670
1240 NEXT I
1250 IF DEDUCT=0 THEN 1360
1260 Q=DEDUCT
1270 ADED=RND2(DEDUCT/DEN*FC)
1280 C=2
1290 R=20
1300 X$=STR$(ADED)
1310 GOSUB 2940
1320 CALL HCHAR(20,23,32,9)
1330 X$="DEDUCTION FOR 19"&STR$(CYR)&"$"&X$
1340 GOSUB 2880
1350 GOSUB 2670
1360 RETURN
1370 SAV=0
1380 GOTO 1070
1390 X=1/DEN*AFC
1400 SAV=RND2(X)
1410 GOTO 1060
1420 ADED=RND2(DEDUCT/DEN*FC)
1430 X$=STR$(ADED)
1440 GOSUB 2940
1450 C=2
1460 R=20
1470 CALL HCHAR(20,23,32,9)
1480 X$="DEDUCTION FOR 19"&STR$(CYR)&"$"&X$
1490 GOSUB 2880
1500 DEDUCT=0
1510 N=0
1520 CMO=1
1530 CYR=CYR+1
1540 GOTO 1230
1550 REM  DISPLAY STATUS ***
1560 PRINT ::"CONTRACT STATUS"::
1570 INPUT "ENTER DATE: ":SDATE$
1580 CALL HCHAR(23,1,32,32)
1590 PRINT "STATUS ON ";SDATE$;":"::
```


Copyright © 1983 Emerald Valley Publishing Co.

```
1600 N=POS(SDATE$,"/",1)
1610 SMO$=SEG$(SDATE$,1,N-1)
1620 M=POS(SDATE$,"/",N+1)
1630 L=M-N-1
1640 SDA$=SEG$(SDATE$,N+1,L)
1650 SYR$=SEG$(SDATE$,M+1,2)
1660 SMO=VAL(SMO$)
1670 SDA=VAL(SDA$)
1680 SYR=VAL(SYR$)
1690 TMO=MO+NP-INT((MO+NP)/12)*12
1700 TMO=TMO+(TMO=0)*-12
1710 TYR=YR+INT(NP/12)-(TMO<MO)
1720 IF SYR>TYR THEN 2240
1730 IF SYR<YR THEN 2270
1740 IF (SYR=TYR)+(SMO>=TMO)=-2 THEN 22
40
1750 IF (SYR=YR)+(SMO<MO)=-2 THEN 2270
1760 N=SYR-YR
1770 N=N*12
1780 X=SMO-MO
1790 N=N+X
1800 IF SDA<DA THEN 1820
1810 N=N+1
1820 TOTPD=N*PMNT
1830 X$=STR$(TOTPD)
1840 GOSUB 2940
1850 PRINT "TOTAL PAID        $";TAB(INC);
X$
1860 BAL=UB+FC
1870 BAL=BAL-TOTPD
1880 X$=STR$(BAL)
1890 GOSUB 2940
1900 PRINT "BALANCE           $";TAB(INC);
X$
1910 P=NP-N-1
1920 IF P=1 THEN 2200
1930 Q=P+1
1940 SAV=RND2(RUL78(AFC))
1950 IF SAV<1 THEN 2220
1960 PREPAY=BAL-SAV
1970 X$=STR$(PREPAY)
1980 GOSUB 2940
1990 PRINT "PREPAY AMOUNT    $";TAB(INC);
X$
2000 X$=STR$(SAV)
2010 GOSUB 2940
2020 PRINT "SAVE BY PREPAY $  ";TAB(INC)
;X$
2030 DEDUCT=FC-SAV
2040 IF SYR=YR THEN 2100
2050 F=13-MO
2060 AYR=YR+1
2070 IF SYR=AYR THEN 2110
2080 F=F+12
2090 AYR=AYR+1
2100 GOTO 2070
2110 P=F
2120 Q=NP-(NP-F+1)
2130 DEDUCT=DEDUCT-RND2(RUL78(FC))
2140 X$=STR$(DEDUCT)
2150 GOSUB 2940
2160 PRINT "DEDUCTIBLE IN ";SYR$;" $";T
AB(INC);X$
2170 PRINT "IF PAID OFF ON ";SDATE$:::
2180 GOSUB 2670
2190 RETURN
2200 SAV=RND2(1/DEN*AFC)
2210 GOTO 1960
2220 SAV=0
2230 GOTO 1960
2240 PRINT "PAID UP"::
2250 GOSUB 2670
2260 RETURN
2270 PRINT "TOO EARLY":::
2280 GOTO 2250
2290 REM TAX DEDUCTION ****
2300 PRINT :::"IF YOU PAY ALL PAYMENTS"
2310 PRINT "AS SCHEDULED, YOU MAY"
2320 PRINT "DEDUCT FINANCE CHARGE"
2330 PRINT "AS FOLLOWS:"::
2340 PRINT "YEAR                      AMOUNT
":::
2350 ND=NP
2360 DYR=YR
2370 P=13-MO
2380 IF P=1 THEN 2510
2390 P=(P<=ND)*-P+(P>ND)*-ND
2400 N=ND-P+1
2410 Q=ND+N
2420 DED=RND2(RUL78(FC))
2430 X$=STR$(DED)
2440 GOSUB 2940
2450 PRINT "19";STR$(DYR);"          $";
TAB(INC);X$
2460 ND=ND-P
2470 DYR=DYR+1
2480 IF ND<12 THEN 2530
2490 P=12
2500 GOTO 2400
2510 DED=RND2(NP/DEN*FC)
2520 GOTO 2430
2530 IF ND=0 THEN 2610
2540 IF ND=1 THEN 2640
2550 P=ND
2560 Q=ND+1
2570 DED=RND2(RUL78(FC))
2580 X$=STR$(DED)
2590 GOSUB 2940
2600 PRINT "19";STR$(DYR);"          $";
TAB(INC);X$
2610 PRINT ::
2620 GOSUB 2670
2630 RETURN
2640 DED=RND2(1/DEN*FC)
2650 GOTO 2580
2660 REM NEW SCREEN *******
2670 CALL GCHAR(23,25,X)
2680 IF X<>32 THEN 2760
2690 C=2
2700 R=23
2710 X$="PRESS ENTER TO CONTINUE"
2720 GOSUB 2880
2730 R=24
2740 X$="OR 9 TO QUIT"
2750 GOSUB 2880
2760 C=16
2770 GOSUB 2820
2780 IF SEL=57 THEN 2800
2790 RETURN
2800 STOP
2810 REM INPUT SUBROUTINE *
2820 CALL HCHAR(24,C,30)
2830 CALL KEY(0,SEL,STAT)
2840 IF STAT=0 THEN 2830
2850 CALL HCHAR(24,C,32)
2860 RETURN
2870 REM PRINT SUBROUTINE *
2880 FOR I=1 TO LEN(X$)
2890 X=ASC(SEG$(X$,I,1))
2900 CALL HCHAR(R,C+I,X)
2910 NEXT I
2920 RETURN
2930 REM EDIT ROUTINE *****
2940 Q=POS(X$,".",1)
2950 IF Q=0 THEN 2990
2960 IF Q=LEN(X$)-1 THEN 3010
2970 IF LEN(X$)>6 THEN 3030
2980 RETURN
2990 X$=X$&".00"
3000 GOTO 2970
3010 X$=X$&"0"
3020 GOTO 2970
3030 A=LEN(X$)-6
3040 Y$=SEG$(X$,1,A)
3050 Z$=SEG$(X$,A+1,6)
3060 X$=Y$&","&Z$
3070 RETURN
3080 END
```

Copyright © 1983 Emerald Valley Publishing Co.

The Electronic Home Secretary

TI BASIC

Now that you have a personal computer, you've probably been looking for ways to use it around the house. When writing software for home applications, it's often possible to create a *general* program that functions in a variety of household situations. The program accompanying this article follows this design philosophy. With it, you can create a personal phone and address directory, time events (such as elapsed telephone connect time), have your computer dial or redial any number in your directory, and set up an inventory of household possessions for insurance and maintenance purposes. All this in standard 16K TI BASIC—with some room to spare for customizing the program according to your preference.

GENERAL DESCRIPTION OF THE PROGRAM

Data Entry

When the program is first RUN, the screen options give the user a choice of updating or using a previous data file saved on cassette or disk, or creating an entirely new data file for one of two options: (1) the phone and address directory, or (2) the household inventory. Both of these options also provide sub-options: For example, the program can draw on the data files to dial (by the dual-tone method) an appropriate phone number, or sum the total cost in the inventory, and then print hardcopy listings of either. The category names for the file organization are provided in the DATA statements 220 and 230.

The input data is stored in the arrays A1$, A2$, A3$, A4$, and A5$. A dimension of 60 is assigned to each of the arrays, and a maximum string length of 190 characters is allowed for each complete entry. Line 710 checks the validity of each data set. At this stage, the program also checks for dimension overflow and memory overflow (lines 480 and 810), and appropriate warning messages are displayed. These features prevent you from accidentally keying in excess data—a situation that would result in an error and program termination. Additionally, the cost category (A2$) in option 2 is designed to accept only numerical input so that you can conveniently carry out numerical operations on the data—for example, total the cost of possessions. And keep in mind that you can, of course, change the categories by altering the data in lines 220 and 230.

Sort Routine

An efficient sort subroutine is presented in the program at line 2410. The routine employs a tree sort procedure which needs approximately $2*N*(\text{Log}_2\ N-1)$ comparisons to sort N entries. Since various versions of sorting routines have been previously published and are readily available, I won't discuss the mathematical details of the sorting procedure. [See reference 2, for example, or any elementary book on numerical analysis.—Ed.] Here, the sorting is based on the entries in the arrays A1$ (i.e., names or items in the default categories). The remaining arrays are appropriately rearranged to be consistent with the original data. The procedure is carried out without the use of any intermediate arrays, thereby saving on the core usage. Completely sorting and rearranging 50 entries takes about 4 minutes.

Data Deletion and Alteration

The subroutine at line 1010 updates any existing data set. You can access any particular entry by its serial number or by its name (or a segment of its name). A search routine (line 1790) retrieves the data set with the specified name, or the next higher one if the name match is not exact. As previously described, the program validates the altered data for allowable string length and memory overflow. At this stage, you have the option of moving up or down in the list, searching for a different entry, or finishing the editing session. Any alteration of the entry title (i.e., A1$) causes the variable FLAG2 to be set equal to unity. Before the directory can be displayed, the data set is resorted.

Display of the Directory

The program allows you to display the data directory in two formats. The first format (at line 1420) provides a concise, quick-reference listing of the complete directory. This includes name and phone number for the Phonebook option, and item and cost for the Inventory option.

In the second format, you can display all the data contained in any single entry. Access to individual entries is either by its serial number in the directory, or by a string search as discussed in the previous section.

Additionally, you can get a hardcopy listing of the entire directory (line 4280) through an RS232-compatible printer, or the TI thermal printer. The screen printing

Copyright © 1983 Emerald Valley Publishing Co.

routine at line 4150 was used to get a hardcopy print-out of screen displays for this article. This portion (lines 4150-4260) can be deleted without affecting the operation of the program.

Computerized Phone Dialing

Now let's look at Touch-Tone dialing with the TI-99/4A. Since the telephone company prohibits direct connections to the phone line of any user equipment not approved by the FCC, the method we will have to use involves simple proximity: Placing the microphone from the phone handset in the front of the monitor speaker dials the phone without any direct connection to the phone lines.

Briefly, the Touch-Tone system of telephone dialing operates by sending a specific pair of audio frequency tones over the voice channel of the phone line for each digit. The switching circuits at the telephone facility decode the tones and actuate the appropriate circuits to make the connection. The tone pairs consist of a low frequency group (697-941 Hz) and a high frequency group (1209-1477 Hz) as shown in Figure 1. For example, to dial the number 5, we have to send the audio tones at 770Hz and 1336 Hz simultaneously for a sufficiently long time to be recognized by the switching circuits. There should also be a sufficient gap between digits for each digit to register individually. Although a 40 millisecond signal duration followed by a 40 millisecond silence should theoretically be adequate, a 150-200 millisecond signal duration and a gap of about 100-150 milliseconds is required for reliable operation with this system.

With the CALL SOUND (*duration, frequency 1, volume 1, frequency 2, volume 2*) command of TI BASIC, the TI-99/4A can generate the dual tones of Figure 1. In doing this, however, an interesting problem arises: If we examine the monitor's output on an oscilloscope, we can observe that the so called "pure tone" from the computer is, in fact, a *square wave* and not a *sine wave*. By Fourier analysis, the square wave can be decomposed into its constituent sine waves. (Interested readers can refer to any elementary book of calculus for the details of the analysis.) To be specific, the output from CALL SOUND (100,500,1) is a square wave of 500 Hz for a 100 millisecond duration at the volume level 1. This is a combination of sine waves at 500 Hz, 1500 Hz, 2500 Hz, and so on. This can pose a problem when we try to dial the first two members (i.e., 698 Hz and 770 Hz) of the low frequency group. The third harmonics of these frequencies, namely, 2091 Hz and 2310 Hz, are also recognized by the switching circuits, resulting in the rejection of the signal. The third harmonics of 852 Hz and 941 Hz seem to be outside the frequency response of the switching circuits and pose no problem.

There are several ways we can overcome this problem when dialing the digits 1 thru 6. One very simple and inexpensive way is to use a passive low-pass filter with a cut-off frequency of about 1.5 KHz in the audio line to the monitor, thereby attenuating the higher frequencies. Figure 2 shows a block diagram for the installation. The circuit for the filter (which I built for less than five dollars) is shown in Figure 3.

HOW TO USE THE PROGRAM

Initial Set-Up

With the choice of N (for NO) for the Load Data option in Display 1, the program has you select either the Phone Directory or Household Inventory option. (If your choice was Y, and you loaded a file, one of the data elements on the file tells the program which option to branch to.) You then key in the data file, guided by the input prompts. The phone number can be entered with spaces and parentheses, if desired. The most recent entry can be re-entered by pressing R for the name (or item). You can terminate by pressing E for EXIT; this causes the data to be sorted and returns you to the master selection list (Display 3).

Load Previous Data File

To load a previously stored data file, we select Y for the Load Data option and follow the screen displays to operate the cassette player or disk. When loaded, the name of the data file, its size and the date of the previous revision will be displayed (Display 2); the program will then return you to the master selection list (Display 3).

Low Frequency Group \ High Frequency Group	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Figure 1. Basic Frequencies for the Two-Tone System of Telephone Dialing

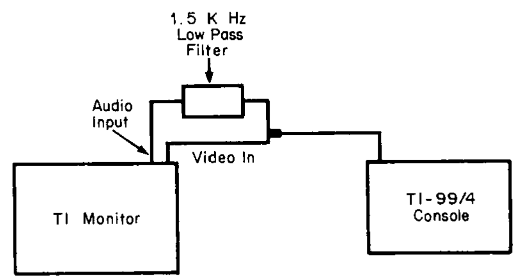


Figure 2. Schematic Layout of Filter Location

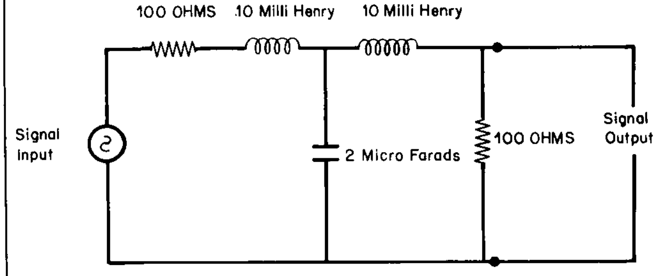


Figure 3. Circuit Diagram of the Filter

Note: On many touch-tone phone systems this filter will not be needed for correct dialling. We suggest you first try without it—Ed.

Copyright © 1983 Emerald Valley Publishing Co.

```
   LOAD DATA? (Y/N)        N
PHONE BOOK?  (Y/N)         Y
   ENTER
      E  TO  EXIT
      R  TO  REENTER
NAME:DOE ↵
PHONE: 987 6543 ↵
STREET:4321 NORTH SOUTH ST ↵
CITY & ZIP:HOLLYWOOD; CA99888 ↵
MISC:JOHN; DATE OF BIRTH JAN ↵
   1 1921; WIFE MARY; CHILDREN
   JOE : SUSAN; WEDD ANNIV FEB
   25; ↵
```

DISPLAY 1   INITIAL SET-UP

Note:

↵ = pressing ENTER,
after the user's response

```
  LOAD DATA (Y/N)          Y
  ENTER
  1.  CS1
  2.  DISK 1
  3.  OTHER

  *  REWIND CASSETTE TAPE      CS1
     THEN PRESS ENTER            ↵

  *  PRESS CASSETTE PLAY       CS1
     THEN PRESS ENTER            ↵
INVENTORY – 1
LSIZE(3800)=1628
LAST UPDATE: MARCH 26 81
```

DISPLAY 2   LOAD PREVIOUS DATA FILE*
(FOR OPTION 1)

* OPTION 2: ENTER FILE NAME
OPTION 3: ENTER DEVICE NAME:

```
PRESS
1 – TO ADD MORE DATA
2 – TO ALTER THE DATA
3 – TO DISPLAY THE DIRECTORY
4 – TO DISPLAY ONE ENTRY
5 – TO USE THE DATA
6 – TO STORE DATA FILE
7 – FOR PRINTER LISTING
8 – TO END PROGRAM

*** UPDATE DIRECTORY ***
```

DISPLAY 3   MASTER SELECTION LIST

```
WHICH ONE; DOE ↵
ENTER
   NEW DATA AT CURSOR
   'D' TO DELETE THE ITEM
   'ENTER' FOR NO CHANGES

DOE?
987 6543;  (424) 987 6543 ↵
4321 NORTH SOUTH ST? ↵
HOLLYWOOD; CA99888? ↵
JOHN; DATE OF BIRTH JAN 1 19
21;  WIFE MARY; CHILDREN JOE
$ SUSAN: WEDD ANNIV FEB 25;? ↵
```

DISPLAY 4   DATA ALTERATION

```
1. ARPACI JOE              321 1234
2. DOE               (424) 987 6543
3. DOE MARY          (424) 789 3456
4. MOORE N.            578 657 8901
5. NORTON P.               356 4473
6.  OHSHIMA                368 8714
7. SASTRY M.               765 2345
8. SHIELD B.           654–789 4532
9. SHYAMALA                206  6808
0. SUBBAIAH          (213) 356 4473
1. WONG V.                 256 3902
PRESS ANY KEY TO CONTINUE
```

DISPLAY 5   SHORT FORM DIRECTORY

```
   WHICH ONE?   DOE ↵
DOE
(424) 987 6543
4321 NORTH SOUTH ST
HOLLYWOOD; CA99888
JOHN; DATE OF BIRTH JAN 1  19
21; WIFE MARY; CHILDREN JOE
& SUSAN; WEDD ANNIV FEB 25;
PRESS
   E  TO LIST UP
   X  TO LIST DOWN
   S  TO SEARCH MORE
PRESS ANY KEY TO CONTINUE
```

DISPLAY 6   SINGLE ITEM DISPLAY

```
   DOE
(424) 987 6543
4321 NORTH SOUTH ST
HOLLYWOOD; CA99888'
JOHN; DATE OF BIRTH JAN 1  19
21;  WIFE MARY; CHILDREN JOE
$ SUSAN; WEDD ANNIV FEB 25;
           1(424) 987 6543
PRESS
   R  TO REDIAL
   S  TO START STOPWATCH
   N  FOR NEW NUMBER
PRESS ANY KEY TO CONTINUE ↵
PRESS
   R  TO REDIAL
   S  TO START STOPWATCH
   N  FOR NEW NUMBER
PRESS ANY KEY TO CONTINUE     S
HOLD DOWN
R  TO DIAL AGAIN
ANY KEY TO CONTINUE
                         00:55
```

DISPLAY 7   PHONE DIALING AND STOPWATCH

```
TOTAL COST OF ALL THE ITEMS

      $$$$$$$$$$$$$$$$$$$
      $                 $
      $                 $
      $                 $
      $      7450.6     $
      $                 $
      $                 $
      $                 $
      $$$$$$$$$$$$$$$$$$$

PRESS ANY KEY TO CONTINUE
```

DISPLAY 8   TOTAL COST OF INVENTORY

```
  ENTER    1. CS1
           2. DSK1
           3. OTHER

           YOUR CHOICE? ↵
TODAY'S DATE:  MARCH 7  1981 ↵
DIR. NAME: PHONE BOOK – 1 ↵
  *  REWIND CASSETTE TAPE      CS1
     THEN PRESS ENTER            ↵
  *  PRESS CASSETTE RECORD     CS1
     THEN PRESS ENTER            ↵
  *  PRESS CASSETTE STOP       CS1
     THEN PRESS ENTER            ↵
```

DISPLAY 9   SAVE DATA FILE

Copyright © 1983 Emerald Valley Publishing Co.

Master Selection List and Its Functions

The master selection list (Display 3) provides access to the program's various options. A banner (***UPDATE DIRECTORY***) will be displayed if there has been any alteration of the data file since the last update. This should act as a constant reminder to save the revised version of the data on a cassette or disk. The different options of the master selection list are as follows:

Option 1: Select this to add any new entry to the data file. This leads to the data entry of Display 1.

Option 2: This leads to Display 4. You can access any individual entry by its serial number in the directory (from display 5) or by a string search. Here, entering a null string (i.e., just pressing the ENTER key) for any category will leave it unaltered.

Option 3: This displays a short form of the directory as in Display 5. The display stops when the screen is filled. Pressing any key causes the remaining data to be displayed, or returns you to the master selection list if no more data is to be displayed.

Option 4: This produces a complete listing of a single entry (Display 6), selected by its serial number in the directory, or by a string search as in Display 4.

Option 5: This allows the program to use the data files when dialing/redialing in the Phonebook option, or to obtain the total purchase cost of the inventory in the Household Inventory option. If you are in the Phonebook option, the program will advance to Display 6. If you approve the display by pressing any key other than E, X, and S, the computer dials the displayed phone number. In the beginning, you may have to adjust the volume control of your TV set or monitor for proper operation. The digits will be displayed one by one as they are dialed. If the total number of characters in the phone number is greater than or equal to 10, the routine recognizes it as a long distance call, and dials 1 at the beginning (Display 7). After getting familiar with the operation, you may want to reduce the time periods assigned in the CALL SOUND statements in lines 3540, 3580, 3590. You can redial the number by pressing R, start the stopwatch by pressing S (and quickly releasing the key), or select a new number using the choice N. Any other key (including a prolonged pressing of S) terminates the dialing session and the master selection screen will be displayed.

With the selection of S, the stopwatch routine on line 3700 is activated. The elapsed time is displayed at the lower right-hand corner (Display 7). You can control the accuracy of the stopwatch by adjusting the time delay constants of the DATA statement in line 3320. Holding down R starts the dialing procedure all over again; pressing any other key returns you to the master selection list (Display 3).

In the Household Inventory option, choice 5 of the master selection list will cause the program to calculate the total purchase cost (Display 8) for all the items in the data file. There's no adjustment here for inflation. This, however, could easily be done. For example, you could key in the consumer price index into the data file at the time of an item's purchase and scale the purchase cost with the current index when evaluating the SUM (in the routine on line 3150). I felt, however, that this procedure would be rather involved for day-to-day use.

Option 6: This permits storing the data file on either cassette or disk. The computer asks (Display 9) for the title of the data file and the date of revision for future reference. This information will be displayed when you re-load the data for another session.

Option 7: This produces a hardcopy listing (with nine complete entries per page) on either the TI thermal printer, or a printer connected to the RS232 interface. The computer first asks you to verify that either the thermal printer or the RS232 interface is connected in order to avoid the File-Error termination. As a precaution, always SAVE the updated file on cassette or disk (option 6) *prior* to printing.

SUMMARY AND FINAL REMARKS

This program is capable of performing a wide variety of functions. We have seen how to use it to maintain a computerized phone directory and dial your phone automatically, as well as to maintain very flexible data files for day-to-day use in the home. Typical applications include an inventory of household valuables, a record of credit cards and bank accounts, lists of author/subject references for research, recipe files, etc. Some of the individual subroutines (in particular, the sorting routine and the stopwatch routine) should also be useful in many other applications. The program, as presented here, is contained within the standard 16K TI BASIC. A version in Extended BASIC to access the additional 32K RAM should give the program an even broader scope.

References

Floyd, R. W. "Algorithm 245, TREESORT 3." *Communications of the A. C. M.*, December 1964, p. 701.

Blinchikoff, H. J. and Zverev, A. I. *Filtering in the Time and Frequency Domain.* John Wiley and Sons, 1976, Chapter 4.

Luff, P. P. "The Electronic Telephone." *Scientific American*, March 1978, pp. 58-64.

Renbarger, J. "A Telephone-Dialing Microcomputer." *BYTE*, June 1980, pp. 140-170.


```
100 REM   *********************
110 REM   * THE HOME SECRETARY *
120 REM   *********************
130 REM
140 REM
170 DIM A1$(60),A2$(60),A3$(60),A4$(60
    ),A5$(60)
180 DIM CAT$(6)
190 DIM P1(3),P2(3)
200 DATA 697,770,852,1210,1340,1481
210 READ P1(1),P1(2),P1(3),P2(1),P2(2)
    ,P2(3)
220 DATA NAME:,PHONE:,STREET:,CITY&ZIP
    :,MISC:
230 DATA "ITEM:      ",COST:,SHOP:,WHEN
    :,MISC:
240 CALL CLEAR
250 LSIZE=0
260 OPT=1
270 READ CAT$(1),CAT$(2),CAT$(3),CAT$(
    4),CAT$(5)
280 PRINT "LOAD DATA? (Y/N)"
290 GOSUB 3120
300 IF KEY<>89 THEN 330
```

Copyright © 1983 Emerald Valley Publishing Co.

```
310 GOSUB 1900
320 GOTO 410
330 REM  NEW SET UP
340 PRINT "PHONE BOOK?  (Y/N)"
350 GOSUB 3120
360 IF KEY=89 THEN 390
370 OPT=2
380 READ CAT$(1),CAT$(2),CAT$(3),CAT$(4),CAT$(5)
390 N=0
400 GOSUB 430
410 GOSUB 850
420 GOTO 410
430 REM  KEY INPUT FOR DATA SET UP
440 PRINT :"    ENTER":"           E TO EXIT":"           R TO REENTER"
450 FLAG1=1
460 FLAG2=1
470 I=N+1
480 IF I<=60 THEN 520
490 PRINT ::"   *** ARRAY FULL (N=60)***"::
500 GOSUB 3100
510 RETURN
520 PRINT
530 INPUT CAT$(1):A1$(I)
540 IF A1$(I)="E" THEN 750
550 IF A1$(I)="END" THEN 750
560 IF A1$(I)="" THEN 530
570 IF A1$(I)<>"R" THEN 620
580 I=I-1
590 N=I-1
600 PRINT :"   *** REENTER LAST SET ***"
610 GOTO 520
620 IF OPT<>1 THEN 650
630 INPUT CAT$(2):A2$(I)
640 GOTO 670
650 INPUT CAT$(2):T
660 A2$(I)=STR$(T)
670 INPUT CAT$(3):A3$(I)
680 INPUT CAT$(4):A4$(I)
690 INPUT CAT$(5):A5$(I)
700 GOSUB 770
710 IF T>190 THEN 600
720 GOSUB 800
730 N=I
740 GOTO 470
750 GOSUB 2410
760 RETURN
770 REM  MEMORY CHECK
780 T=LEN(A1$(I)&A2$(I)&A3$(I)&A4$(I)&A5$(I))
790 RETURN
800 LSIZE=LSIZE+T
810 IF LSIZE<3300 THEN 840
820 CALL SOUND(200,800,4)
830 PRINT :"** WARNING ** MEMORY GETTING   FULL":"  (LSIZE=";STR$(LSIZE)&"/3800)":
840 RETURN
850 SC=4
860 GOSUB 3060
870 PRINT "PRESS":::"1 - TO ADD MORE DATA"::"2 - TO ALTER THE DATA"
880 PRINT :"3 - TO DISPLAY THE DIRECTORY"::"4 - TO DISPLAY ONE ENTRY":
890 PRINT :"5 - TO USE THE DATA"::"6 -  TO STORE DATA FILE"::"7 - FOR PRINTER LISTING"::
900 PRINT "8 - TO END PROGRAM"
910 IF FLAG1=0 THEN 930
920 PRINT "  *** UPDATE DIRECTORY ***"
930 GOSUB 3120
940 IF KEY<49 THEN 850
950 IF KEY>56 THEN 850
960 GOSUB 3060
970 ON (KEY-48)GOSUB 990,1010,1430,1560,3150,2140,4280,4450
980 RETURN
990 GOSUB 430
1000 RETURN
1010 REM  DATA ALTERATION
1020 INPUT "WHICH ONE?  ":M$
1030 IF M$="" THEN 1410
1040 PRINT ::"   ENTER"::"              NEW DATA AT CURSOR"::"          'D' TO DELETE THE ITEM"::"       'ENTER' FOR NO CHANGES":
1050 GOSUB 1800
1060 PRINT ::
1070 FLAG1=1
1080 T1$="?    "
1090 I=M
1100 GOSUB 770
1110 T=-T
1120 GOSUB 800
1130 INPUT A1$(M)&T1$:TMP$
1140 IF TMP$="" THEN 1220
1150 IF TMP$<>"D" THEN 1200
1160 A1$(M)="__"
1170 GOSUB 2410
1180 N=N-1
1190 RETURN
1200 A1$(M)=TMP$
1210 FLAG2=1
1220 INPUT A2$(M)&TI$:TMP$
1230 IF TMP$="" THEN 1250
1240 A2$(M)=TMP$
1250 INPUT A3$(M)&TI$:TMP$
1260 IF TMP$="" THEN 1280
1270 A3$(M)=TMP$
1280 INPUT A4$(M)&TI$:TMP$
1290 IF TMP$="" THEN 1310
1300 A4$(M)=TMP$
1310 INPUT A5$(M)&TI$:TMP$
1320 IF TMP$="" THEN 1340
1330 A5$(M)=TMP$
1340 GOSUB 770
1350 IF T<192 THEN 1380
1360 PRINT :"** REENTER LAST SET **"
1370 GOSUB 1130
1380 GOSUB 800
1390 GOSUB 1650
1400 ON T GOTO 1130,1020,1410
1410 RETURN
1420 REM  DISPLAY ENTIRE DIRECTORY
1430 IF FLAG2=0 THEN 1450
1440 GOSUB 2410
1450 FOR I=1 TO N
1460 M=29-LEN(A2$(I))
1470 T$=STR$(I)&"."
1480 PRINT TAB(4-LEN(T$));T$;A1$(I);TAB(M);A2$(I)
1490 IF I=20 THEN 1520
1500 IF I=40 THEN 1520
1510 GOTO 1530
1520 GOSUB 3100
1530 NEXT I
1540 GOSUB 3100
1550 RETURN
1560 REM  SINGLE ITEM LISTING
1570 INPUT "WHICH ONE?  ":M$
1580 IF M$="" THEN 1640
1590 GOSUB 1800
1600 GOSUB 3060
1610 PRINT A1$(M)::A2$(M)::A3$(M)::A4$(M)::A5$(M)::
1620 GOSUB 1650
1630 ON T GOTO 1600,1570,1640
1640 RETURN
1650 PRINT :"PRESS":"    E TO LIST UP":"    X TO LIST DOWN":"    S TO SEARCH MORE"
1660 GOSUB 3100
1670 T=3
1680 IF KEY<>69 THEN 1720
1690 T=1
1700 IF M=1 THEN 1780
```

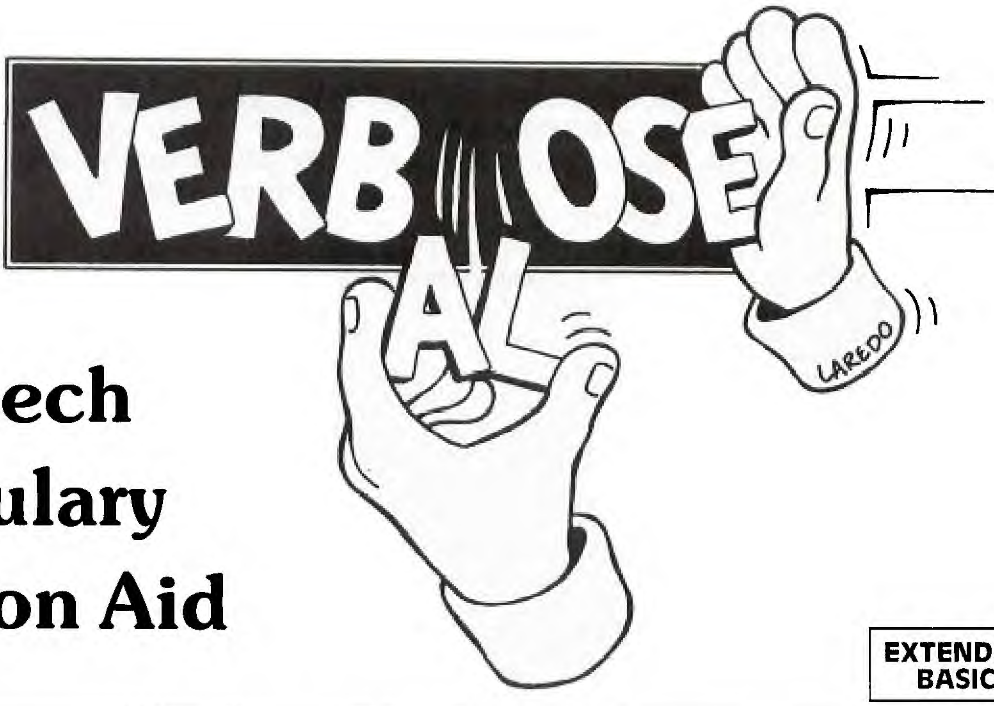

Copyright © 1983 Emerald Valley Publishing Co.

```
1710 M=M-1
1720 IF KEY<>88 THEN 1760
1730 T=1
1740 IF M=N THEN 1780
1750 M=M+1
1760 IF KEY<>83 THEN 1780
1770 T=2
1780 RETURN
1790 REM SEARCH ROUTINE FOR SINGLE ITEM LISTING
1800 IF ABS(ASC(M$)-53)>4 THEN 1850
1810 M=VAL(M$)
1820 IF M<=N THEN 1840
1830 M=N
1840 RETURN
1850 FOR I=1 TO N
1860 M=I
1870 IF M$<=A1$(I)THEN 1890
1880 NEXT I
1890 RETURN
1900 REM LOAD DATA
1910 PRINT "ENTER"::"1. CS1"::"2. DISK1"::"3. OTHER"
1920 INPUT DEV
1930 IF DEV<>1 THEN 1960
1940 DEV$="CS1"
1950 GOTO 2010
1960 IF DEV<>2 THEN 2000
1970 INPUT "ENTER FILE NAME:":FIL$
1980 DEV$="DSK1."&FIL$
1990 GOTO 2010
2000 INPUT "ENTER DEVICE NAME:":DEV$
2010 OPEN #2:DEV$,INTERNAL,INPUT ,FIXED 192
2020 INPUT #2:OPT,N,FIL$,DATE$,LSIZE
2030 IF OPT=1 THEN 2050
2040 READ CAT$(1),CAT$(2),CAT$(3),CAT$(4),CAT$(5)
2050 PRINT ::FIL$::"LSIZE(3800)=";LSIZE::"LAST UPDATE: ";DATE$::
2060 FOR I=1 TO N
2070 INPUT #2:A1$(I),A2$(I),A3$(I),A4$(I),A5$(I)
2080 NEXT I
2090 IF DEV=1 THEN 2120
2100 FOR TD=1 TO 1000
2110 NEXT TD
2120 CLOSE #2
2130 RETURN
2140 REM SAVE DIRECTORY
2150 IF FLAG2=0 THEN 2170
2160 GOSUB 2410
2170 PRINT "ENTER 1. CS1"
2180 PRINT "       2. DSK1"
2190 PRINT "       3. OTHER"::::
2200 INPUT "YOUR CHOICE?":ANS
2210 IF (ANS<1)+(ANS>3)THEN 2170
2220 ON ANS GOTO 2230,2250,2310
2230 DEV$="CS1"
2240 GOTO 2320
2250 INPUT "ENTER FILE NAME.":NAM$
2260 IF LEN(NAM$)<11 THEN 2290
2270 PRINT "ENTER NO MORE THAN TEN LETTERS PLEASE."
2280 GOTO 2170
2290 DEV$="DSK1."&NAM$
2300 GOTO 2320
2310 INPUT "ENTER DEVICE NAME.":DEV$
2320 INPUT "ENTER DATE.":DATE$
2330 OPEN #3:DEV$,INTERNAL,OUTPUT,FIXED 192
2340 PRINT #3:OPT,N,FIL$,DATE$,LSIZE
2350 FOR I=1 TO N
2360 PRINT #3:A1$(I),A2$(I),A3$(I),A4$(I),A5$(I)
2370 NEXT I
2380 CLOSE #3
2390 FLAG1=0
2400 RETURN
2410 REM SORTING ROUTINE
2420 FLAG2=0
2430 CALL SOUND(100,800,6)
2440 PRINT :::" ***** SORTING DATA *****"::::
2450 IF (N-1)THEN 2470
2460 RETURN
2470 FOR I=1 TO N
2480 NEXT I
2490 N2=INT(N/2)
2500 N21=N2+2
2510 ICT=1
2520 I=2
2530 N1=N21-I
2540 NN=N
2550 IK=N1
2560 GOSUB 2880
2570 JK=2*IK
2580 IF JK>NN THEN 2660
2590 IF JK=NN THEN 2620
2600 IF A1$(JK+1)<=A1$(JK)THEN 2620
2610 JK=JK+1
2620 IF A1$(JK)<=C$ THEN 2660
2630 GOSUB 2940
2640 IK=JK
2650 GOTO 2570
2660 GOSUB 3000
2670 ON ICT GOTO 2680,2780
2680 IF I>=N2 THEN 2710
2690 I=I+1
2700 GOTO 2530
2710 ICT=2
2720 NP2=N+2
2730 I=2
2740 N1=NP2-I
2750 NN=N1
2760 IK=1
2770 GOTO 2560
2780 IK=1
2790 GOSUB 2880
2800 JK=N1
2810 GOSUB 2940
2820 IK=N1
2830 GOSUB 3000
2840 IF I>=N THEN 2870
2850 I=I+1
2860 GOTO 2740
2870 RETURN
2880 C$=A1$(IK)
2890 M$=A2$(IK)
2900 T$=A3$(IK)
2910 TMP$=A4$(IK)
2920 TI$=A5$(IK)
2930 RETURN
2940 A1$(IK)=A1$(JK)
2950 A2$(IK)=A2$(JK)
2960 A3$(IK)=A3$(JK)
2970 A4$(IK)=A4$(JK)
2980 A5$(IK)=A5$(JK)
2990 RETURN
3000 A1$(IK)=C$
3010 A2$(IK)=M$
3020 A3$(IK)=T$
3030 A4$(IK)=TMP$
3040 A5$(IK)=TI$
3050 RETURN
3060 REM CLEAR & SET SCREEN
3070 CALL CLEAR
3080 CALL SCREEN(SC)
3090 RETURN
3100 REM KEY RETURN
3110 PRINT ::"PRESS ANY KEY TO CONTINUE"
3120 CALL KEY(0,KEY,STATUS)
3130 IF STATUS=0 THEN 3120
3140 RETURN
3150 REM UTILITY PROGRAMS
3160 IF OPT=1 THEN 3290
3170 SUM=0
```

Copyright © 1983 Emerald Valley Publishing Co.

```
3180 FOR I=1 TO N
3190 SUM=SUM+VAL(A2$(I))
3200 NEXT I
3210 PRINT "TOTAL COST OF ALL THE ITEMS":::
3220 PRINT ::::TAB(10);SUM:::::::::
3230 CALL HCHAR(11,7,36,18)
3240 CALL HCHAR(19,7,36,18)
3250 CALL VCHAR(12,7,36,7)
3260 CALL VCHAR(12,24,36,7)
3270 GOSUB 3100
3280 RETURN
3290 REM DIAL PHONE
3300 REM CLOCK TIME DELAYS FOLLOW
3310 RESTORE 3320
3320 DATA 2,57,55,53,51
3330 READ D1,D2,D3,D4,D5
3340 PRINT ::"YOU WANT ME TO DIAL"::
3350 GOSUB 1570
3360 IF M$="" THEN 3610
3370 CALL CLEAR
3380 H=23
3390 V=8
3400 T$=A2$(M)
3410 L=LEN(T$)
3420 PRINT A1$(M):A2$(M):A3$(M):A4$(M):A5$(M)::::
3430 IF L<10 THEN 3460
3440 L=L+1
3450 T$="1"&T$
3460 FOR J=1 TO L
3470 TMP$=SEG$(T$,J,1)
3480 CALL HCHAR(H,V,ASC(TMP$),1)
3490 V=V+1
3500 IF ASC(TMP$)<48 THEN 3600
3510 IF ASC(TMP$)>57 THEN 3600
3520 T=VAL(TMP$)
3530 IF T<>0 THEN 3560
3540 CALL SOUND(300,941,0,1336,2)
3550 GOTO 3590
3560 I=INT((T-1)/3)+1
3570 IJ=T-3*(I-1)
3580 CALL SOUND(300,P1(I),0,P2(IJ),2)
3590 CALL SOUND(250,44000,29)
3600 NEXT J
3610 PRINT ::"PRESS ":"         R TO REDIAL":"         S TO START STOPWATCH":"         N FOR NEW NUMBER"
3620 GOSUB 3100
3630 IF KEY=82 THEN 3370
3640 IF KEY=78 THEN 3350
3650 IF KEY<>83 THEN 3690
3660 PRINT :"HOLD DOWN ":"R TO DIAL AGAIN":"ANY KEY TO CONTINUE":::
3670 GOSUB 3700
3680 IF T=82 THEN 3290
3690 RETURN
3700 REM STOP WATCH
3710 H=23
3720 V=23
3730 JS=1
3740 S=4
3750 DELAY=D1
3760 FOR J1=0 TO 16
3770 A1=48+J1
3780 FOR J2=0 TO 9
3790 A2=48+J2
3800 FOR J3=0 TO 5
3810 A3=48+J3
3820 FOR J4=JS TO 9
3830 A4=48+J4
3840 GOSUB 4020
3850 IF STATUS<>0 THEN 4010
3860 S=4
3870 DELAY=D2
3880 NEXT J4
3890 JS=0
3900 DELAY=D3
3910 NEXT J3
3920 DELAY=D4
3930 S=11
3940 NEXT J2
3950 A2=48
3960 DELAY=D5
3970 S=10
3980 GOSUB 4020
3990 NEXT J1
4000 GOTO 3760
4010 RETURN
4020 REM COMPENSATED CLOCK
4030 CALL HCHAR(H,V+2,32,1)
4040 CALL KEY(0,T,STATUS)
4050 FOR J=1 TO DELAY
4060 IF STATUS<>0 THEN 4010
4070 NEXT J
4080 CALL SCREEN(S)
4090 CALL HCHAR(H,V,A1,1)
4100 CALL HCHAR(H,(V+1),A2,1)
4110 CALL HCHAR(H,V+2,58,1)
4120 CALL HCHAR(H,V+3,A3,1)
4130 CALL HCHAR(H,V+4,A4,1)
4140 RETURN
4150 REM SCREEN PRINTING
4160 IF RP=0 THEN 4270
4170 OPEN #1:"RS232"
4180 FOR IJ=1 TO 20
4190 PRINT #1:T$
4200 T$=""
4210 FOR J=1 TO 32
4220 CALL GCHAR(IJ,J,CH)
4230 T$=T$&CHR$(CH)
4240 NEXT J
4250 NEXT IJ
4260 CLOSE #1
4270 RETURN
4280 REM COMPLETE LISTING ON PRINTER
4290 INPUT "ENTER DEVICE NAME.":DEV$
4300 PRINT "IS ";DEV$;" READY?(Y/N)"
4310 GOSUB 3120
4320 IF KEY<>89 THEN 4440
4330 OPEN #3:DEV$,OUTPUT
4340 T=INT(N/9)
4350 FOR J=0 TO T
4360 FOR M=1 TO 9
4370 I=9*J+M
4380 PRINT #3:TAB(5);I;")";TAB(10);A1$(I):TAB(10);A2$(I):TAB(10);A3$(I):TAB(10);A4$(I):TAB(10);A5$(I)::
4390 IF I=N THEN 4430
4400 NEXT M
4410 GOSUB 3100
4420 NEXT J
4430 CLOSE #3
4440 RETURN
4450 PRINT "DO YOU WISH TO HALT THE  PROGRAM AND LOSE ALL DATA INMEMORY? (Y/N)"
4460 CALL KEY(0,K,S)
4470 IF S=0 THEN 4460
4480 IF K=ASC("N")THEN 850
4490 IF K<>ASC("Y")THEN 4440
4500 END
```


Copyright © 1983 Emerald Valley Publishing Co.



A Speech Vocabulary Expansion Aid

EXTENDED BASIC

Verbose is a program that was written in an evolutionary manner. One thing just lead to another. The story goes something like this:

One day I decided to make a program speak a simple sentence. After all, the TI Speech Synthesizer must have something to say. Well, anyway, I came up with a simple sentence—don't remember what it was now—in a program which I entered and ran.

Wow—almost half of the words in the sentence were not in the resident vocabulary! It was clearly time for me to read the manual that came with the unit. Surprise. I found it had a vocabulary limited to three or four hundred words. That was not enough for me. Further research was definitely called for.

Reading the TI Extended BASIC manual, I found a program on page 206 that allowed adding standard suffixes to resident vocabulary words (e.g.,-ed, -ing, -s). After playing with this suffix program awhile, I realized it would be possible to *concatenate* two resident vocabulary words to produce a totally new word: therefore, meanwhile, or update. I wrote a routine to do this. Once this concatenation routine was working, it seemed like a speech tool starting to evolve.

It would be nice, I thought, to have the results of the concatenation routine printed in the form of DATA statements. I could then write these DATA statements containing the new word's speech data into other programs that needed to speak the new word. So, I generated a routine to do this, and added it to the concatenation routine.

All of these routines, including a method of building a vocabulary file on disk, were combined into a nice, neat, simple-to-use program. The result is Listing 4. As you can see from the listing, I originally called the program *Word Builder*. When I decided to write an article on it, however, the name seemed too mundane. So in a fit of cleverness, I renamed the program *Verbose*. My wife and my friends just shook their heads and groaned. . .

A TV picture is worth a thousand words, right? Well, perhaps not quite, so I have combined some text with screen images to guide you through the operation of *Verbose*. Before you start the *Verbose* program, make sure you have either the *TI Extended BASIC* or *TI Speech Editor* Command Cartridge plugged in. *Verbose* uses the SPGET and SAY subroutines that are available in these modules. OK, now you're ready to load *Verbose* and type RUN.

```
          + + +WORD BUILDER+ + +
       ENTER NUMBER OF YOUR CHOICE
             1-JOIN TWO WORDS
           2-PRINT SPEECH DATA
        3-STORE NEW WORD ON DISK
                 4-EXIT
?
```

Here we are at the main menu screen. Let's create a new word by joining two words. The new word that we will generate will be *rewrite* and will be made from vocabulary words *read* and *right*. Type 1 and press the ENTER key.

```
         ENTER FIRST WORD JOIN
?
```

We are asked for the first word that will be used in the joining. Type READ and press ENTER.

```
       ENTER SECOND WORD TO JOIN
?
```

Now type the word RIGHT and press ENTER.

```
   ENTER THE SPELLING OF THE NEW WORD
?
```

Type in REWRITE and then press ENTER.

```
       TRUNCATE HOW MANY BYTES?
```

OK, don't panic here! *Verbose* just wants to know how much of the first word (READ) to truncate before it com-

Copyright © 1983 Emerald Valley Publishing Co.

bines it with the second word (RIGHT). We don't know how much, so we make a wild guess of, say, 34. What we want is to truncate the AD from READ and combine that sound with RIGHT. As soon as you press ENTER this time, the TI-99/4A will say the new word for you.

```
SAY AGAIN? (Y OR N)
```

Here you can answer the question with Y as many times as you like to check the sound of the new word. After hearing enough of it, enter N.

```
        SAY AGAIN? (Y OR N) N
        1 - CHANGE SOME MORE
        2 - BACK TO MAIN MENU
?
```

If you don't think the new word sounded quite right, type 1 and press ENTER.

```
TRUNCATE HOW MANY BYTES?
```

This time type 55 and press ENTER.

Listen to the word as many times as you like. With 55 bytes truncated, it sounds to me close enough to use. When you are satisfied, return to the main menu.

```
        + + +WORD BUILDER+ + +
    ENTER NUMBER OF YOUR CHOICE
        1 - JOIN TWO WORDS
        2 - PRINT SPEECH DATA
      3 - STORE NEW WORD ON DISK
              4 - EXIT
?
```

Here we are back at the ranch. Let's print the data for our new word by selecting option 2. Don't forget to press ENTER. (I'm not going to remind you about that anymore 'cause you've got the ENTER key down pat.)

```
ENTER THE WORD WHOSE DATA YOU WANT TO
              PRINT - -
?
```

After you enter REWRITE and press the you-know-what, the printer will output what you see in Listing 1. It didn't work? Well your printer must be set up differently from mine. Go to Listing 4 and modify line 870 of *Verbose* (the OPEN statement for the printer) to match your setup. If you don't have a printer, delete lines 870 and 1070. Also modify lines 940, 950, 990, and 1060 by deleting the "#1:" of each print statement. Now, instead of going to the printer, everything will go to the screen of the TI-99/4A. The last change is to enter this line:

```
1070 INPUT F$
```

Now it will stay on the screen (so you can copy it on paper) until you press ENTER.

Look over Listing 2. This is a sample TI BASIC program that shows how the DATA statements for *Verbose* can be used. You will note the DATA statements for the word REWRITE are entered in lines 360-490 of Listing 2. Lines 280-330 build the string E$ which will cause REWRITE to be spoken. The FOR-NEXT loop here is terminated when the last byte is read. The loop counter limit (133) was the number of bytes printed out for REWRITE by *Verbose*. The subroutines SAY and SPGET are explained in the speech synthesizer manual.

```
        + + +WORD BUILDER+ + +
    ENTER NUMBER OF YOUR CHOICE
        1 - JOIN TWO WORDS
        2 - PRINT SPEECH DATA
      3 - STORE NEW WORD ON DISK
              4 - EXIT
?
```

It is very tiring to enter all those DATA statements of the previous sample program. For those of you with a disk system, an easier method of saving and using words from *Verbose* is available with option 3. Go ahead and select it now.

```
    PUT THE DISK WITH "WORDS"
        FILE IN DRIVE ONE.
    PRESS ENTER WHEN READY
```

The disk on which you wish to keep your new vocabulary words should now be placed in disk drive 1. The words that will be saved will be appended to a file called WORDS on this diskette. See line 1160 of Listing 4 for the OPEN statement for this file.

```
    PUT THE DISK WITH "WORDS"
        FILE IN DRIVE ONE.
    PRESS ENTER WHEN READY

ENTER THE WORD WHOSE DATA YOU WANT TO
              SAVE - -
?
```

Enter the word REWRITE to save. The disk drive will run and then *Verbose* will return to its main menu. Use this option to save a few more words that you choose. Then run the *Spelling Test Game* in Listing 3 using the resultant WORDS file.

The *Spelling Test Game* program will accept up to 20 words for the WORDS file. It then speaks each word, checks the spelling that is input, and keeps score. Any children in your home should find it useful for spelling drill.

Study Listing 3 and notice lines 230-270. The WORDS file has a pair of strings for each word saved. The first string contains the spelling of the word. The second string contains the actual speech data.

As mentioned earlier, the program listing for *Verbose* is Listing 4.

A final note of caution: Once you start that TI-99/4A talking, BEWARE—you may have trouble getting a word in edgewise. . .

Copyright © 1983 Emerald Valley Publishing Co.

Listing 1

```
THE WORD IS ** REWRITE **
LENGTH = 133 BYTES

DATA 96,0,42,161,19,49,92,60,149,149
DATA 78,86,51,117,147,223,26,61,196,197
DATA 69,253,170,93,103,231,176,108,167,10
DATA 158,83,211,151,156,188,40,21,157,106
DATA 180,178,42,89,125,96,0,85,162,101
DATA 33,221,57,28,139,154,142,144,176,116
DATA 172,106,58,92,162,67,137,105,248,82
DATA 142,49,39,169,209,7,179,84,220,175
DATA 218,196,26,103,157,119,235,83,133,156
DATA 233,220,113,110,117,170,88,51,77,58
DATA 238,169,211,240,100,207,186,167,201,69
DATA 196,162,42,205,46,245,41,179,68,87
DATA 97,51,24,105,146,233,22,0,64,1
DATA 93,121,60
```

Listing 2

```
100 REM  +++++++++++++++++++++
110 REM  +   SPEAK--  "THIS  +
120 REM  +   PROGRAM HAD A   +
130 REM  +     REWRITE."     +
140 REM  +                   +
150 REM  +                   +
160 REM  +THIS SAMPLE SHOWS+
170 REM  +HOW THE OUTPUT OF+
180 REM  +"VERBOSE" IS USED+
190 REM  +IN TI BASIC....   +
200 REM  +                   +
210 REM  +++++++++++++++++++++
220 REM
230 REM
240 CALL SPGET("THIS",A$)
250 CALL SPGET("PROGRAM",B$)
260 CALL SPGET("HAD",C$)
270 CALL SPGET("A",D$)
280 RESTORE 360
290 E$=""
300 FOR I=1 TO 133
310 READ X
320 E$=E$&CHR$(X)
330 NEXT I
340 CALL SAY("",A$,"",B$,"",C$,"",D$,"
   ",E$)
350 REM ** REWRITE **
360 DATA 96,0,42,161,19,49,92,60,149,1
   49
370 DATA 78,86,51,117,147,223,26,61,19
   6,197
380 DATA 69,253,170,93,103,231,176,108
   ,167,10
390 DATA 158,83,211,151,156,188,40,21,
   157,106
400 DATA 180,178,42,89,125,96,0,85,162
   ,101
410 DATA 33,221,57,28,139,154,142,144,
   176,116
420 DATA 172,106,58,92,162,67,137,105,
   248,82
430 DATA 142,49,39,169,209,7,179,84,22
   0,175
440 DATA 218,196,26,103,157,119,235,83
   ,133,156
450 DATA 233,220,113,110,117,170,88,51
   ,77,58
460 DATA 238,169,211,240,100,207,186,1
   67,201,69
470 DATA 196,162,42,205,46,245,41,179,
   68,87
480 DATA 97,51,24,105,146,233,22,0,64,
   1
490 DATA 93,121,60
```

Listing 3

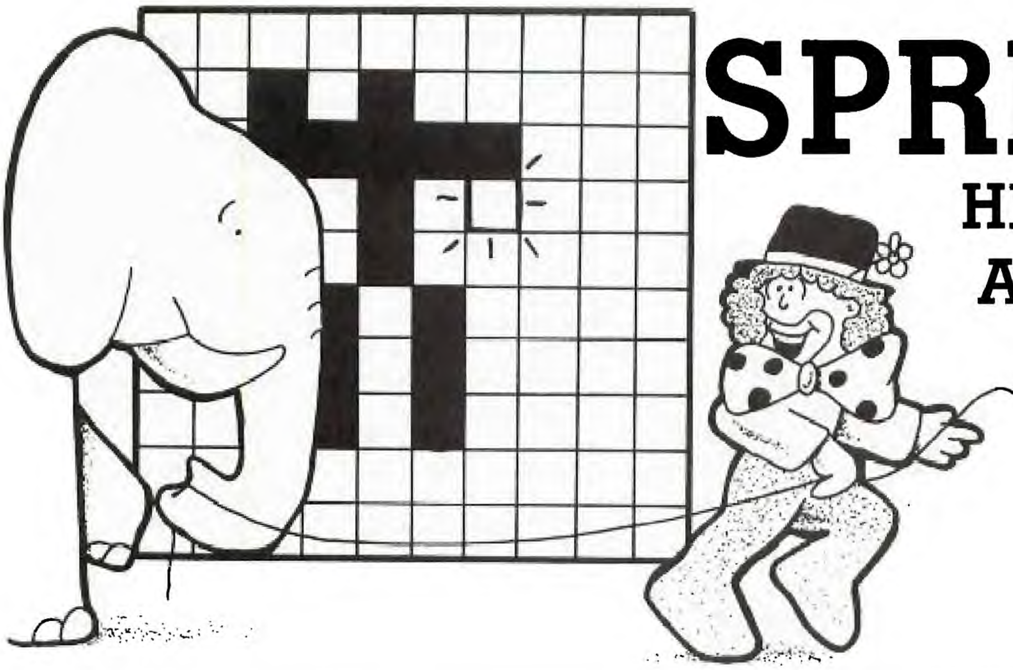
```
100 REM  +++++++++++++++++++++++++
110 REM  +   SPELLING  TEST  GAME +
120 REM  +++++++++++++++++++++++++
130 REM
140 REM
150 REM  +++++++++++++++++++++++++
160 REM    USES "DSK1.WORDS" FILE AS SO
   URCE OF WORDS TO
170 REM        GUESS IN GAME.
180 DIM WORD$(20),F$(20)
190 CALL CLEAR
200 PRINT "PUT DISK WITH ""WORDS"" FIL
   E  IN DRIVE ONE"
210 INPUT "PRESS ENTER WHEN READY ":X$
220 OPEN #1:"DSK1.WORDS",INTERNAL,INPU
   T ,VARIABLE 254
230 FOR I=1 TO 20
240 IF EOF(1)<>0 THEN 300
250 INPUT #1:WORD$(I)
260 INPUT #1:F$(I)
270 NEXT I
280 LAST=I
290 GOTO 310
300 LAST=I-1
310 CLOSE #1
320 REM
330 CALL CLEAR
340 SCORE=0
350 PRINT "THERE ARE ";LAST;"WORDS":::
360 PRINT "SEE IF YOU CAN SPELL THEM
    ALL CORRECTLY.  GOOD LUCK!"
370 FOR M=1 TO 700
380 NEXT M
390 FOR J=1 TO LAST
400 CALL CLEAR
410 PRINT "WORD #";J:::
420 CALL SAY("",F$(J))
430 INPUT "SPELL IT- ":X$
440 IF X$=WORD$(J)THEN 470
450 CALL SAY("UHOH")
460 SCORE=SCORE-1
470 SCORE=SCORE+1
480 PRINT ::"CORRECT SPELLING IS
          >>";WORD$(J);"<<"::
490 PRINT "YOUR SCORE IS ";SCORE;" OUT
    OF";J
500 FOR Y=1 TO 500
510 NEXT Y
520 NEXT J
530 CALL CLEAR
540 PRINT "YOU GOT ";INT((SCORE/LAST)*
   100);"% CORRECT":::::
550 INPUT "ENTER ""Y"" TO TRY AGAIN ":
   Z$
560 IF Z$="Y" THEN 330
570 CALL CLEAR
580 END
```

Copyright © 1983 Emerald Valley Publishing Co.

Listing 4

```
100 REM  ++++++++++++++++++++++++++
110 REM  +    V  E  R  B  O  S  E    +
120 REM  +                          +
130 REM  ++++++++++++++++++++++++++
140 REM
150 REM
160 REM
170 REM
180 REM
190 REM  SPEECH MAKER  ROUTINE
200 REM  WILL COMBINE WORDS INTO NEW STRINGS.
210 REM
220 CALL CLEAR
230 PRINT "   +++ WORD BUILDER +++":::::
240 PRINT "ENTER NUMBER OF YOUR CHOICE"::
250 PRINT :"    1 - JOIN TWO WORDS"::
260 PRINT "    2 - PRINT SPEECH DATA"::
270 PRINT "    3 - STORE NEW WORD ON DISK"::
280 PRINT "    4 - EXIT":::
290 INPUT CHOICE
300 IF (CHOICE<1)+(CHOICE>4)=-1 THEN 220
310 ON CHOICE GOSUB 550,870,1100,330
320 GOTO 220
330 CALL CLEAR
340 END
350 REM  +++ TRUNCATE FIRSTWORD +++
360 CALL CLEAR
370 INPUT "TRUNCATE HOW MANY BYTES?":BYTES
380 MAXBYTES=LEN(B$)-3
390 IF BYTES<MAXBYTES THEN 420
400 PRINT "TOO MANY BYTES....."
410 GOTO 370
420 IF BYTES>-1 THEN 450
430 PRINT "NO NEGATIVE NUMBERS"
440 GOTO 370
450 L=MAXBYTES-BYTES
460 C$=SEG$(B$,1,2)&CHR$(L)&SEG$(B$,4,L)
470 RETURN
480 REM  +++ SPEAK NEW WORD +++
490 CALL CLEAR
500 CALL SAY("",NEWDATA$)
510 INPUT "SAY AGAIN?  (Y OR N)":CHOICE$
520 IF CHOICE$="Y" THEN 500
530 RETURN
540 REM   +++ JOIN TWO WORDS SUBROUTINE +++
550 CALL CLEAR
560 PRINT "ENTER FIRST WORD TO JOIN"
570 INPUT FIRSTWORD$
580 IF FIRSTWORD$=LASTMADE$ THEN 610
590 CALL SPGET(FIRSTWORD$,B$)
600 GOTO 620
610 B$=LASTDATA$
620 CALL CLEAR
630 PRINT "ENTER SECOND WORD TO JOIN"
640 INPUT SECONDWORD$
650 IF SECONDWORD$=LASTMADE$ THEN 680
660 CALL SPGET(SECONDWORD$,D$)
670 GOTO 690
680 D$=LASTDATA$
690 CALL CLEAR
700 PRINT "ENTER THE SPELLING OF THE   NEW WORD"
710 INPUT NEWWORD$
720 REM
730 GOSUB 350
740 NEWDATA$=C$&D$
750 GOSUB 480
760 PRINT "     1 - CHANGE SOME MORE":"       2 - BACK TO MAIN MENU"
770 INPUT CHOICE
780 IF (CHOICE<1)+(CHOICE>2)=-1 THEN 760
790 IF CHOICE=1 THEN 730
800 LASTMADE$=NEWWORD$
810 LASTDATA$=NEWDATA$
820 RETURN
830 REM   +++ PRINT SPEECH DATA SUBROUTINE +++
840 REM  ------------------------------------------------
850 REM   THIS OPEN STATEMENT MAY NEED TO BE MODIFIED
860 REM   FOR YOUR PRINTER......
870 OPEN #1:"RS232.DA=8.BA=9600"
880 REM  ------------------------------------------------
890 CALL CLEAR
900 PRINT "ENTER THE WORD WHOSE DATA     YOU WANT TO PRINT--      ":::
910 GOSUB 1230
920 IF L=0 THEN 1070
930 VALUES$=""
940 PRINT #1::::"THE WORD IS **   ";WORD$;"  **":
950 PRINT #1:"LENGTH =";L;"BYTES"::
960 FOR I=1 TO L
970 VALUES$=VALUES$&STR$(ASC(SEG$(F$,I,1)))
980 IF I/10<>INT(I/10)THEN 1020
990 PRINT #1:"DATA ";VALUES$
1000 VALUES$=""
1010 GOTO 1030
1020 VALUES$=VALUES$&","
1030 NEXT I
1040 IF VALUES$="" THEN 1070
1050 VALUES$=SEG$(VALUES$,1,LEN(VALUES$)-1)
1060 PRINT #1:"DATA ";VALUES$
1070 CLOSE #1
1080 RETURN
1090 REM   +++ ADD NEW WORD TO VOCABULARY FILE +++
1100 CALL CLEAR
1110 PRINT "PUT THE DISK WITH ""WORDS""      FILE IN DRIVE ONE."
1120 INPUT "PRESS ENTER WHEN READY  ":X$
1130 PRINT :::"ENTER THE WORD WHOSE DATA         YOU WANT TO SAVE--     ":::
1140 GOSUB 1230
1150 IF L=0 THEN 1200
1160 OPEN #1:"DSK1.WORDS",INTERNAL,APPEND,VARIABLE 254
1170 PRINT #1:WORD$
1180 PRINT #1:F$
1190 CLOSE #1
1200 RETURN
1210 REM
1220 REM   +++ FIND WORD SUBROUTINE
1230 INPUT WORD$
1240 F$=""
1250 IF WORD$="" THEN 1300
1260 IF WORD$=LASTMADE$ THEN 1290
1270 CALL SPGET(WORD$,F$)
1280 GOTO 1300
1290 F$=LASTDATA$
1300 L=LEN(F$)
1310 RETURN
```


Copyright © 1983 Emerald Valley Publishing Co.



SPRITER

HIGH-SPEED ANIMATION WITH SPRITES

EXTENDED BASIC

TI Extended BASIC lets you fill the screen with rapidly moving sprites of many colors. See for example *Sprite Chase* in "Computer Gaming." Although the smooth and rapid motion possible with sprites is indeed quite impressive and arcade-like, think how much more spectacular these screen displays would be if we animated the moving sprites: After all, why just move a man-shaped sprite when you can also move his arms and legs? Picture the visual impact of a bird-sprite flying across the screen flapping its wings. How about a circus parade with clowns tumbling, animals walking, and elephants moving their trunks? All of this—and more—is possible with sprite animation.

The technique of animation is old and well-known. First we draw a series of figures with each figure in a slightly different position and posture. Then, we rapidly flash the figures one after the other on the screen, and *persistence of vision* goes to work—fooling our eyes and causing us to see figures move as if alive. Now with sprites we can duplicate this movie animation technique on the TI-99/4 and 99/4A through simple commands in Extended BASIC. [See "3-D Animation with the TMS9918A Video Chip."—Ed.]

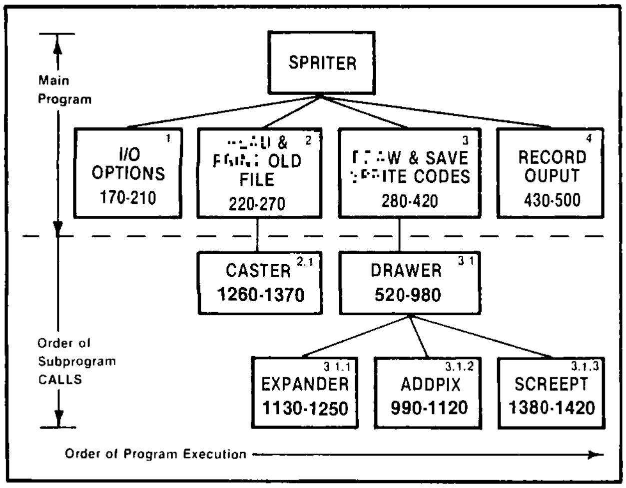
The usual trouble with computer animation is the tedious series of tasks that you have to do after drawing the figures: You have to figure out, keep track of, and key in those long pattern identifiers. If you have chosen to work with sprites that are four characters large, these codes then become 64 hexadecimal characters long! This situation prompted me to write *Spriter* (Listing 1), a program that does much of the work for you, and leaves you free to concentrate on the fun—drawing the figures for the animation sequence. *Spriter* automatically computes, files, and saves an array of four-character pattern identifiers that define sprites of magnification 3 or 4 (figurines). After you draw each figurine you can output a model of it to the thermal printer (optional) and when you are finished, you can save the whole file on cassette tape or disk.

When you run *Spriter*, it presents you with a 16 × 16 character work area in the screen's character display field. Under your directon, the computer generates an enlarged model of the figurine within the work area. The image is made up of dark and bright character squares, each of which has a counterpart in the figurine. Changes in the display field are automatically converted into changes of the figurine's pattern identifier. The figurines in the computer's memory (RAM) can be stored permanently on tape or disk and later accessed by either *Spriter* or any other program with animation recall capability. See, for example, the animation demonstration program in Listing 2. *Spriter* thus allows you to generate new figurines, transfer any figurine that you have stored on tape into RAM, and rework any figurine that is present in RAM.

How to Run *Spriter*

Instructions are almost self-contained: A series of prompts guides you through much of the program. First, you are asked if you have a thermal printer and if you want to input a file of characters from tape or disk. If so, you are asked the corresponding file name. Then the work area is framed on the screen. If you have chosen to show an existing figurine by reading in an old file, *Spriter* copies that figurine to the work area. When the cursor appears in the upper lefthand corner of the work area, you are ready to draw a new figurine or redraw an existing one. You can move the cursor anywhere within the work area by using the arrow keys for horizontal and vertical motion, and the W, R, C, and Z keys for diagonal motions. When the cursor is moved, it automatically leaves a trace as determined by the polarity keys: bright if the A key was pressed and dark if the F key was previously pressed. When the cursor first appears, the polarity of the trace is dark. Afterwards, by using the motion and polarity keys you can draw and erase portions of the model until you are satisfied with the results. Then press the Q key to exit the drawing mode. A new series of prompts will guide you through the rest of the program.

Copyright © 1983 Emerald Valley Publishing Co.



How *Spriter* Works

Space does not allow a line-by-line description of *Spriter* (see Listing 1). But for those interested in exploring the intricacies of the program, I have provided a road map in the form of a structure diagram (Figure 1). Functions identified within the main program are depicted as boxes above the dashed line; those identified with subprograms are below the dashed line. The order of program execution in this figure is from left to right, and the order of subprogram calls is from top to bottom. The program line numbers to which these various functions refer are listed at the bottom of each box.

The main task of drawing a series of sprite figurines is under the direction of Function 3 (Draw & Save Sprite Codes). The task of initializing the work area and handling individual figurines and their models is directed by subprogram Expander, which constructs this model when given the pattern identifier for the figurine. After this, Drawer directs changes in the model display according to the user's keyboard inputs. Then it calls upon subprogram Addpix to make the corresponding changes in the pattern identifier for the figurine that is being drawn. When the figurine is complete, Drawer will call subprogram Screept to output the model on the thermal printer (if this option is chosen).

A Demonstration Program

After you generate cassette or disk data files of figurine pattern-identifiers with *Spriter*, you are ready to incorporate these into an animation sequence within a program. The short demonstration program (Listing 2) is a very simple example. This program is in effect a continuous loop projector that sequences through a series of sprite figurines to produce animation of the sprite that is moved across the screen. After the program reads the pattern identifiers from cassette tape or disk, it goes into the animation loop. You can stop the looping by pressing SHIFT C (on the 99/4) or FCTN 4 (on the 99/4A).

Keep in mind that this program is just a very simple demonstration of the sprite animation technique. You can use it to study the figurines files created by *Spriter*, and perhaps as a starting point for writing more elaborate sprite animation programs that are more apt to your specific applications [We've also included an additional program (Listing 3) that incorporates DATA statements for those wishing to get a feel for the animation process before working with *Spriter*.—Ed.]

Listing 1

```
100 REM ***********************
110 REM *       SPRITER       *
120 REM ***********************
130 REM
140 REM
150 REM
160 CALL CHARSET :: FOR I=96 TO 143 :: CALL CHAR(I,"") :: NEXT I
170 INPUT " DO YOU HAVE A THERMAL PRINTER(Y/N)?":TP$
180 DIM CHA$(50),ID$(50)
190 INPUT "DO YOU WANT TO INPUT A FILE OF CHARACTERS FROM TAPE OR DISK(Y/N)?":AN$ :: IF AN$<>"Y" THEN 240
200 DISPLAY AT(24,1):"FILE NAME" :: ACCEPT AT(24,11)SIZE(10)VALIDATE(UALPHA,DIGIT):NAM$ :: IF POS(NAM$," ",1)<>0 THEN 200
210 PRINT "ENTER '1' FOR TAPE       '2' FOR DISK" :: INPUT "(1/2)?":AN$
220 IF AN$="1" THEN @FILE$="CS1" ELSE IF AN$="2" THEN @FILE$="DSK1."&NAM$ ELSE GOTO 210
230 GOSUB 1260 :: GOTO 250
240 IF AN$<>"N" THEN 190 ELSE NS=0 :: GOTO 290
250 IF TP$="N" THEN 280
260 OPEN #1:"TP.U.E.S",OUTPUT :: FOR J=0 TO NS
270 PRINT #1:J,ID$(J) :: NEXT J :: CLOSE #1
280 NS=NS+1 :: C$=CHA$(0)
290 FOR I=NS TO 1000
300 GOSUB 520
310 DISPLAY AT(2,1):ID$(NS) :: DISPLAY AT(22,1):C$
320 DISPLAY AT(3,1):"PRESS ANY KEY TO CONTINUE."
330 CALL KEY(0,K,S) :: IF S=0 THEN 330
340 CALL CLEAR :: INPUT "ENTER COLOR CODE FOR SPRITE. ":COL
350 CALL CHAR(96,C$) :: CALL SPRITE(#1,96,COL,30,30,0,-30) :: CALL MAGNIFY(4)
360 DISPLAY AT(10,3):"PRESS ANY KEY TO CONTINUE."
370 CALL KEY(0,K,S) :: IF S=0 THEN 370 ELSE CALL DELSPRITE(ALL)
380 INPUT "DO YOU WANT TO SAVE THE CHARACTER CODE OF THIS SPRITE(Y/N)?":AN$
390 IF AN$="Y" THEN CHA$(NS)=C$
400 INPUT "DO YOU WANT TO CONTINUE(Y/N)?":AN$ :: IF AN$="N" THEN 430 ELSE IF AN$<>"Y" THEN 400
410 NS=NS+1
420 NEXT I :: END
430 INPUT "DO YOU WISH TO SAVE RESULTS ON TAPE OR DISK(Y/N)?":AN$
440 IF AN$="N" THEN 510 ELSE IF AN$<>"Y" THEN 430
450 DISPLAY AT(24,1):"ENTER FILE NAME" :: ACCEPT AT(24,16)SIZE(10)VALIDATE(UALPHA,DIGIT):NAM$ :: IF POS(NAM$," ",1)<>0 THEN 450
460 PRINT "ENTER '1' FOR TAPE       '2' FOR DISK" :: INPUT "(1/2)?":AN$
470 IF AN$="1" THEN @FILE$="CS1" ELSE IF AN$="2" THEN @FILE$="DSK1."&NAM$ ELSE GOTO 460
480 OPEN #1:@FILE$,INTERNAL,OUTPUT,FIXED 128
490 PRINT #1:NAM$,NS
```


Copyright © 1983 Emerald Valley Publishing Co.

```
500 FOR K=0 TO NS :: PRINT #1:ID$(K),CHA$(K):: NEXT K :: CLOSE #1
510 END
520 REM SUB DRAWER(TP$,C$,NS,AN$,CHA$(),ID$())
530 CALL CHAR(33,RPT$("F",16))
540 IF C$="" THEN 590
550 INPUT "DO YOU WANT TO INITIALIZE WITH A PREVIOUSLY DEFINED CHARACTER (Y/N)?":AN$
560 IF AN$="N" THEN C$="" :: GOTO 590 ELSE IF AN$<>"Y" THEN 550
570 INPUT "ENTER INDEX OF CHARACTER DESIRED,ANY '-' VALUE FOR MOST RECENTLY DEFINED":NOS
580 IF NOS<0 THEN 590 ELSE C$=CHA$(NOS):: NXX=NOS :: GOTO 600
590 NXX=NS-1
600 M=16 :: IF LEN(C$)=0 THEN C$=RPT$("0",64):: F=0 ELSE F=1
610 IF LEN(C$)=16 THEN C$=C$&RPT$("0",48)
620 N=1 :: C1$=SEG$(C$,1,16):: C2$=SEG$(C$,17,16):: C3$=SEG$(C$,33,16):: C4$=SEG$(C$,49,16)
630 PRINT "USE ARROW KEYS AND 'W,R,C,Z' TO MOVE CURSOR,OR TO CHANGE POLARITY USE'F'FOR DARK AND 'A'FOR LIGHT."
640 CALL KEY(0,K,S):: IF S=0 THEN 640
650 CALL CLEAR :: CALL HCHAR(4,4,30,M+2):: CALL HCHAR(M+5,4,30,M+2)
660 CALL VCHAR(5,4,30,M):: CALL VCHAR(5,M+5,30,M):: X,Y=5
670 IF AN$="Y" THEN GOSUB 1130
680 IF NXX>=0 THEN DISPLAY AT(2,1):ID$(NXX):: DISPLAY AT(22,1):C$
690 CALL HCHAR(X,Y,30,1):: CT$=C$ :: GOSUB 990 :: C$=CT$
700 CALL KEY(1,K,S)
710 IF S=0 THEN 700 ELSE IF N=1 THEN CALL HCHAR(X,Y,33,1)ELSE CALL HCHAR(X,Y,32,1)
720 IF K=1 THEN N=0
730 IF K=12 THEN N=1
740 IF K=5 AND X>5 THEN X=X-1
750 IF K=0 AND X<M+4 THEN X=X+1
760 IF K=2 AND Y>5 THEN Y=Y-1
770 IF K=3 AND Y<M+4 THEN Y=Y+1
780 IF K=4 AND X>5 THEN IF Y>5 THEN X=X-1 :: Y=Y-1
790 IF K=6 AND X>5 THEN IF Y<M+4 THEN X=X-1 :: Y=Y+1
800 IF K=15 AND X<M+4 THEN IF Y>5 THEN X=X+1 :: Y=Y-1
810 IF K=14 AND X<M+4 THEN IF Y<M+4 THEN X=X+1 :: Y=Y+1
820 IF K=18 THEN 910
830 IF X>4 AND X<13 THEN IF Y>4 AND Y<13 THEN P=1 ELSE P=3 ELSE IF Y>4 AND Y<13 THEN P=2 ELSE P=4
840 IF P=1 THEN X0=X-5 :: Y0=Y-5 :: CH$=SEG$(C$,1,16)
850 IF P=2 THEN X0=X-13 :: Y0=Y-5 :: CH$=SEG$(C$,17,16)
860 IF P=3 THEN X0=X-5 :: Y0=Y-13 :: CH$=SEG$(C$,33,16)
870 IF P=4 THEN X0=X-13 :: Y0=Y-13 :: CH$=SEG$(C$,49,16)
880 CT$=CH$ :: GOSUB 990 :: CH$=CT$
890 IF P=1 THEN C1$=CH$ ELSE IF P=2 THEN C2$=CH$ ELSE IF P=3 THEN C3$=CH$ ELSE C4$=CH$
900 CALL HCHAR(X,Y,30,1):: C$=C1$&C2$&C3$&C4$ :: GOTO 700
910 DISPLAY AT(22,1):"ENTER SPRITE NAME." :: DISPLAY AT(23,1):"" :: DISPLAY AT(24,1):""
920 ACCEPT AT(23,1):ID$(NS)
930 IF TP$="N" THEN GOTO 980
940 DISPLAY AT(22,1):"WANT TO COPY ON T.P.(Y/N)?" :: ACCEPT AT(23,1):AN$
950 IF AN$="N" THEN GOTO 980 ELSE IF AN$<>"Y" THEN 940
960 DISPLAY AT(2,1):ID$(NS):: DISPLAY AT(22,1):C$
970 CALL SCREEPT
980 RETURN
990 REM SUB ADDPIX(X,Y,N,C$)
1000 IF Y0<4 THEN ZT=2*X0+1 :: YT0=3-Y0 ELSE ZT=2*X0+2 :: YT0=7-Y0
1010 A2$=SEG$(CT$,ZT,1)
1020 IF ZT>1 THEN A1$=SEG$(CT$,1,ZT-1)
1030 IF ZT<16 THEN A3$=SEG$(CT$,ZT+1,16-ZT)
1040 NH=ASC(A2$):: IF NH<=57 THEN NH=NH-48 ELSE NH=NH-55
1050 ZZ=INT(NH/(2^YT0))-2*INT(NH/(2^(YT0+1)))
1060 IF ZZ=0 AND N=1 THEN NH=NH+2^YT0
1070 IF ZZ=1 AND N=0 THEN NH=NH-2^YT0
1080 IF NH<=9 THEN A2$=STR$(NH)ELSE A2$=CHR$(NH+55)
1090 IF ZT=16 THEN CT$=A1$&A2$
1100 IF ZT=1 THEN CT$=A2$&A3$
1110 IF ZT<>16 AND ZT<>1 THEN CT$=A1$&A2$&A3$
1120 RETURN
1130 REM SUB EXPANDER(C$,X0,Y0)
1140 DEF B(A)=INT(NHF/(2^A))-2*INT(NHF/(2^(A+1)))
1150 FOR IW=0 TO 15 :: FOR JW=0 TO 15
1160 IF JW>7 THEN JW0=JW-8 ELSE JW0=JW
1170 IF IW>7 THEN IW0=IW-8 ELSE IW0=IW
1180 IF IW<8 THEN IF JW<8 THEN LW=1 ELSE LW=3 ELSE IF JW<8 THEN LW=2 ELSE LW=4
1190 IF JW0<4 THEN ZW=2*IW0+1 :: YW=3-JW0 ELSE ZW=2*IW0+2 :: YW=7-JW0
1200 SA2$=SEG$(S$,ZW,1)
1210 SA2$=SEG$(C$,ZW+16*(LW-1),1)
1220 NHF=ASC(SA2$):: IF NHF<=57 THEN NHF=NHF-48 ELSE NHF=NHF-55
1230 IF B(YW)=1 THEN CALL HCHAR(X+IW,Y+JW,33,1)
1240 NEXT JW :: NEXT IW
1250 RETURN
1260 REM SUB CASTER(@FILE$,N,I$(),C$())
1270 OPEN #2:@FILE$,INTERNAL,INPUT ,FIXED 128 :: GOTO 1280
1280 INPUT #2:NAM$,NS
1290 FOR I=0 TO NS
1300 INPUT #2:ID$(I),CHA$(I):: NEXT I :: CLOSE #2
1310 N3=23 :: N1=0 :: IF NS<=24 THEN N2=NS ELSE N2=23
1320 FOR I=N1 TO N2 :: IF I>NS THEN 1370
1330 PRINT I;ID$(I):: NEXT I
1340 PRINT "PRESS ANY KEY TO CONTINUE."
1350 CALL KEY(0,K,S):: IF S=0 THEN 1350
1360 IF NS>N3 THEN N1=N1+24 :: N2=N2+24 :: N3=N3+24 :: GOTO 1320
1370 RETURN
1380 SUB SCREEPT
1390 OPEN #255:"TP.U.E.S",OUTPUT :: FOR X=1 TO 24 :: S$=""
1400 FOR Y=1 TO 32 :: CALL GCHAR(X,Y,Z):: S$=S$&CHR$(Z)
1410 NEXT Y :: PRINT #255:S$ :: NEXT X :: CLOSE #255
1420 SUBEND
```

Copyright © 1983 Emerald Valley Publishing Co.

Listing 2

```
100 REM ************************
110 REM *     SPRITE DEMO       *
120 REM ************************
130 REM
140 REM
150 REM
160 REM
170 REM DEMONSTRATION OF SPRITE ANIMAT
    ION USING CASSETTE OR DISK DATA FI
    LE.
180 CALL CLEAR
190 DIM CHA$(17),ID$(17)
200 CALL CASTER(NS,ID$(),CHA$())
210 FOR I=0 TO NS :: CALL CHAR(136-4*I
    ,CHA$(I))
220 NEXT I
230 CALL CLEAR
240 CALL SPRITE(#1,136,2,30,30,0,-10):
    : CALL MAGNIFY(4)
250 FOR I=0 TO NS :: CALL PATTERN(#1,1
    36-4*I):: CALL DELAY :: NEXT I ::
    GOTO 250
260 END
270 SUB CASTER(N,I$(),C$())
280 REM
290 REM ********************
300 REM SET FILE$="CS1" FOR
310 REM TAPE FILES; USE YOUR
320 REM FILE NAME:
330 REM      "DSK1.FILENAME"
340 REM FOR DISK FILES.
350 REM ********************
360 REM
370 OPEN #2:FILE$,INTERNAL,INPUT ,FIXE
    D 128
380 INPUT #2:NAM$,N
390 FOR I=0 TO N
400 INPUT #2:I$(I),C$(I):: NEXT I :: C
    LOSE #2
410 SUBEND
420 SUB DELAY
430 FOR I=0 TO 15 :: NEXT I
440 SUBEND
```

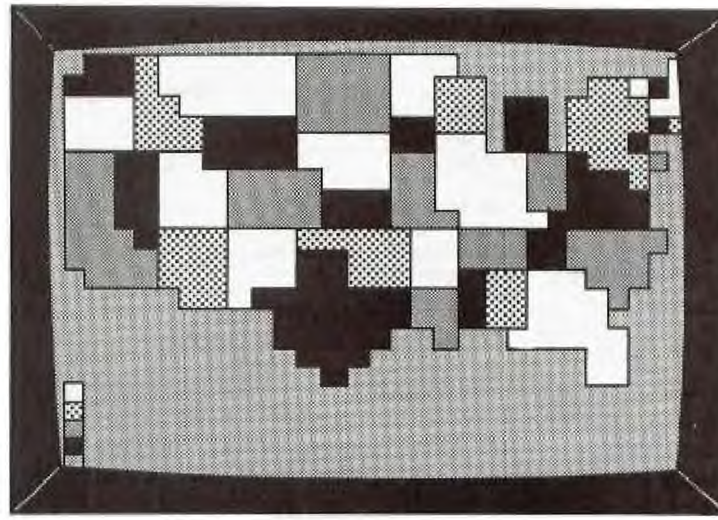
Listing 3

```
100 REM *******************
110 REM *   SPRITE DEMO 2   *
120 REM *******************
130 REM
140 REM
150 REM DEMONSTRATION OF SPRITE ANIMAT
    ION USING DATA STATEMENTS
160 CALL CLEAR
170 DIM I$(17),C$(17)
180 GOSUB 250 !CASTER
190 FOR I=0 TO N :: CALL CHAR(136-4*I,
    C$(I))
200 NEXT I
210 CALL CLEAR
220 CALL SPRITE(#1,136,2,30,30,0,-10):
    : CALL MAGNIFY(4)
230 FOR I=0 TO N :: CALL PATTERN(#1,13
    6-4*I):: GOSUB 300 :: NEXT I :: GO
    TO 230
240 END
250 REM SUBROUTINE CASTER
260 READ NAM$,N
270 FOR I=0 TO N
280 READ I$(I),C$(I):: NEXT I
290 RETURN
300 REM SUBROUTINE DELAY
310 FOR I=0 TO 15 :: NEXT I
320 RETURN
330 DATA MAN6NB1,12
340 DATA MAN#1,0006090906 0F0F0F1E060F0
    F190804080000000000000000000020508
    0000000
350 DATA MAN#2,0304040307072F130303070
    6060207000080800008090D0A080808080 8
    0808000
360 DATA MAN#2.5,0609090 60F0F172606060
    E1EA24201030000000000000804080000000
    000000000
370 DATA MAN#3,0007090306 0F0F172F06060
    60F09081800000000000000000000000000
    0804080
380 DATA MAN#4,00001824 1C0C1C2C4E16060
    706020206 000000000000000000040A0000
    0000000
390 DATA MAN#5,0000000000000000387FDE966
    242810001000000000000205080000000 0
    0008000
400 DATA MAN#6,00061424140C0C0C0C1C1E1
    E1E1D0C100000000000000000000000000
    0008080
410 DATA MAN#6.5,00000020201 84C7C0C0C0
    E0606090E0400000000000000000000000
    000008080
420 DATA MAN#7,000000000000000040804 02F1
    E37664 0C0000000000000000000408000000
    0804020
430 DATA MAN#8,000000110A0603010101030
    3062A120600000008448 50A0C0C0080000 0
    0000000
440 DATA MAN#1,0006090906 0F0F0F1E060F0
    F190804080000000000000000000020508
    0000000
450 DATA MAN#3,0007090306 0F0F172F06060
    60F09081800000000000000000000000000
    0804080
460 DATA MAN#2.5,0609090 60F0F172606060
    E1EA24201030000000000000804080000000
    000000000
```


Copyright © 1983 Emerald Valley Publishing Co.

COLOR MAPPING and the TI-99/4A

TI BASIC



One of the principal features of the new technology exhibited by low-cost home computers is their graphic capabilities. But these small computers' graphic capabilities in the area of mapping is often overlooked. Statistical mapping is not new; cartographers have used the methods described in this article for decades, and sophisticated mapping programs that run on large mainframe computers have been available (from Harvard University and elsewhere) for a number of years. Their application for the small computer field, however, especially in the classroom setting, should be further explored and documented.

The program described in this article, *United States Choropleth Map,* was written for the TI-99/4A. No peripherals are needed, except for a cassette recorder to store the program. Therefore, anyone with the console can get started immediately and experience the excitement of computer mapping. The program should benefit a large number of users: For example, classroom teachers, from the upper elementary grades through college levels in geography, can utilize it; sales and marketing managers, and others interested in the spatial distribution of goods and services may also find it especially useful; political scientists can easily see the national election results displayed almost instantaneously.

Choropleth Mapping

Simply defined, choropleth mapping has been likened to a spatial table. Enumeration units—which can be census tracts, counties, states, or other small area geography—are symbolized by different area patterns, depending on the values they represent. Typically, the original data are divided into a number of data classes (map classes). The individual enumeration units will be symbolized according to the map class into which their data value falls. Enumeration units are put into classes because it is usually impractical, or not feasible, to apply an area pattern for *each* data value.

Classing, of course, is similar to a sieve; individual values "fall" into each group depending on the class limits. This results in a generalization, and the final map is a *simplification* of the original data. Nonetheless, choropleth mapping has a number of advantages over a simple table of values. It provides a third, or spatial dimension to a rather dull list of values in tabular format. In the bibliography, I've listed several good books that discuss the methods and rationale of this form of mapping.

Symbolization on choropleth maps takes on several different forms. In the case of *black and white* mapping, the enumeration units are symbolized by area *patterns* to differentiate each class from all others. Different shades of grey, ranging from near-black to near-white, are often used. *Color* symbolization includes two forms: (1) different hues (such as green, red, blue, etc.) for the various classes, or (2) different values (shades) of the *same* color. The present program uses the second method.

Main Features of the Program

Figure 1 illustrates the main components of the program's logic, and Figure 2 lists the most important variables. I wrote the program with flexibility in mind: New subroutines can be incorporated as different versions are developed. Lines 170 to 260 of the program are used for an opening screen, which displays the program name.

The first section, Program Instructions, provides the option list and incorporates directions for data input. The present version accommodates only data from the keyboard. (You may wish to add program statements to read data from a file system). The data is input by entering the values to be mapped for each state, by the alphabetical order of the states.

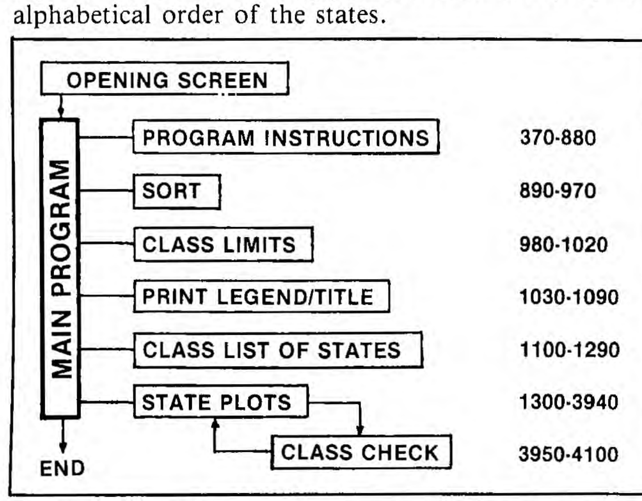


Figure 1 -Main program logic, showing subroutines, or United States Choropleth Map.

Copyright © 1983 Emerald Valley Publishing Co.

SC	·Map background color
MC	·Map color (blue or green)
C(1-5)	·Map class colors
V(1-50)	·Values of each state to be mapped
TT$	·Map title
X1	·Limit for class 1
X2	·Limit for class 2
X3	·Limit for class 3
X4	·Limit for class 4
K(1-5)	·Character sets
S(1-5)	·ASCII character identifiers
SN$(1-50)	·States' names
NN	·State's number

Figure 2 ·Principal variables used in mapping program.

After the data are entered, the main program directs the flow to a simple bubble sort subroutine, where the data values are sorted into ascending order. The data values are then classed, and the class limits are selected in the Class Limits section. There are a number of ways in which data may be classed. This program will class the data values into *quintiles*—that is, into *five* classes each having the same *number* of values. As the data set has been arrayed in ascending order, the values of the class limits are computed rather easily.

Program flow is next directed to printing. With the TI-99/4A and the BASIC language supplied with the standard computer, printed ASCII characters must be displayed *before* the color graphic blocks are called on the screen. Otherwise, scrolling will move the color graphics off the screen. The Print Legend and Title subroutine displays the classed values and user-chosen title on the lower portion of the screen.

State Plots is next. Each state is assigned an ordinal number based on its alphabetical rank (1-50). As each state is encountered, flow is directed to a subroutine, Class Check, in which the state's ordinal number is used to determine which color the state should be.

Outlines of the states are not variable, but the color (symbolization) varies, of course, depending on the class in which each state falls. Flow continues until each state has been displayed on the screen. A color graphic block is displayed adjacent to the printed legend values at the bottom of the screen. The program ends with a GO TO statement (line 3940); the screen will display the map until the user presses SHIFT C or FCTN = to BREAK program.

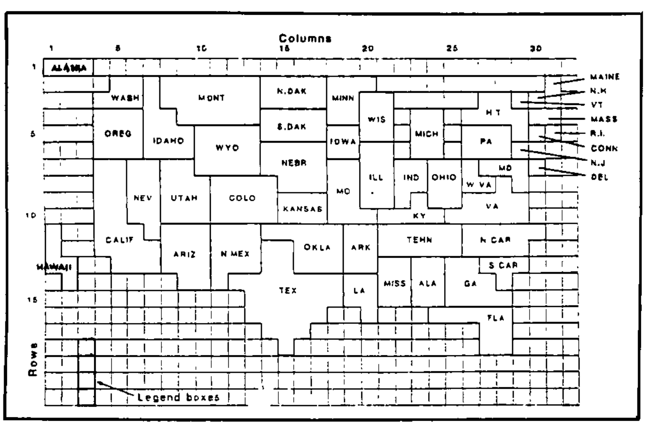


Figure 3 - The graphic blocks used to identify the shapes of the states in the choropleth map program. Each block's color is generated with the CALL COLOR command.

Mapping on the TI-99/4A

The color graphic capabilities of the TI-99/4A include a screen which is divided into 32 columns and 24 rows, each block of which is addressable by a row and column identifier in the CALL HCHAR and VCHAR commands. Any of 16 colors (including transparent) can be specified. Further resolution is possible by using the CALL CHAR command, with which the user can specify the "on" and "off" condition of 64 dots in each graphic block, through the use of hexadecimal codes. The present program utilizes only the 32 × 24 resolution screen, and does not develop the refinements of the shapes of the states that are possible with the CALL CHAR command. The blocks used to identify the states are illustrated in Figure 3. Although only an approximation is achieved with this resolution, the shapes resemble fairly well the individual states, and relative area is proportional to real geographical areas. Other users may wish to modify these (although I suspect that the 16K RAM will be taxed if they do).

The Choice of Color Symbolization

As mentioned previously one standard, acceptable way to symbolize the areas on choropleth maps is to vary the lightness or darkness of one color, in accordance with the values represented. Classes having higher values are rendered darker, and the lower-valued classes lighter. For this program, the highest class is black, the lowest class white, and the three intermediate classes are in three shades of green or three shades of blue. The TI-99/4A can display 15 different colors, and fortunately there are three different greens and blues, each ranging from light to dark. Symbolizing the color classes in this manner better shows the *total form* of the distribution over the map. The map reader gets a better idea of the continuously changing nature of the *spatial* attributes of the data.

Program Enhancements

You can make any number of useful changes to this program. You may wish to provide alternate ways of classing the data (e.g., quartiles, equal steps, standard deviations, or others), to add new subroutines, or to enter your own classes. A *different* color for each class could be used in the color symbolization. The variable C(1-5) need only be changed to conform to the other color code options used by the TI BASIC. With small changes, data sets could be input from external files rather than from the keyboard. This would be especially useful in classroom settings, where census or other data from previous years (and other geographical data) can be compared with present patterns.

Computer-aided instruction (CAI) which uses inquiry questions generated by the spatial distribution seen on the screen could also be added to this program. Geographical concepts could be brought out in this manner, and students could easily test hypotheses.

One most intriguing enhancement would be to introduce animation (dynamic cartography) to the program. Various data sets could be read (from files) and displayed in fairly fast sequence to produce a dynamic, *changing* image of the geographical distribution. For example, different population densities from 1850 to 1980 would show the steady drift of our population from east

Copyright © 1983 Emerald Valley Publishing Co.

to west; a temporal set of sales (or income) performance data would be of interest to marketing analysts. One advantage of all computer mapping is its ability to show the dynamic qualities of geographical data. This capability is possible on the versatile TI-99/4A.

Bibliography

Robinson, Arthur; Sale, Randall; and Morrison, Joel. *Elements of Cartography.* 4th ed. New York: John Wiley and Sons, 1978.

Lawrence, G.R.P. *Cartographic Methods.* 2nd ed. New York: Methuen and Co., Ltd., 1979.

Harvard Graduate School of Design. Various publications, Laboratory for Computer Graphics and Spatial Analysis. Cambridge, Mass.


```
100 REM  ******************
110 REM  * CHOROPLETH MAP *
120 REM  ******************
130 REM
140 REM
150 REM
160 DIM SN$(50),V(50),VV(50)
170 CALL CLEAR
180 CALL SCREEN(12)
190 PRINT TAB(12);"UNITED"
200 PRINT TAB(12);"STATES"
210 PRINT ::::TAB(8);"CHOROPLETH MAP":::::::::
220 CALL COLOR(9,5,5)
230 CALL VCHAR(3,4,96,17)
240 CALL HCHAR(3,5,96,25)
250 CALL HCHAR(19,5,96,25)
260 CALL VCHAR(4,29,96,15)
270 RESTORE 280
280 DATA ALABAMA,ALASKA,ARIZONA,ARKANSAS,CALIFORNIA,COLORADO,CONNECTICUT,DELAWARE,FLORIDA,GEORGIA,HAWAII,IDAHO
290 DATA ILLINOIS,INDIANA,IOWA,KANSAS,KENTUCKY,LOUISIANA,MAINE,MARYLAND,MASSACHUSETTS,MICHIGAN,MINNESOTA
300 DATA MISSISSIPPI,MISSOURI,MONTANA,NEBRASKA,NEVADA,NEW HAMPSHIRE,NEW JERSEY,NEW MEXICO,NEW YORK
310 DATA NORTH CAROLINA,NORTH DAKOTA,OHIO,OKLAHOMA,OREGON,PENNSYLVANIA,RHODE ISLAND,SOUTH CAROLINA,SOUTH DAKOTA
320 DATA TENNESSEE,TEXAS,UTAH,VERMONT,VIRGINIA,WASHINGTON,WEST VIRGINIA,WISCONSIN,WYOMING
330 FOR I=1 TO 50
340 READ SN$(I)
350 NEXT I
360 CALL CLEAR
370 PRINT TAB(6);"PROGRAM INSTRUCTIONS"
380 PRINT TAB(14);"***"
390 PRINT "CHOOSE MAP BACKGROUND COLOR"::
400 PRINT TAB(8);"1-MEDIUM RED"
410 PRINT TAB(8);"2-LIGHT RED"
420 PRINT TAB(8);"3-DARK YELLOW"
430 PRINT TAB(8);"4-LIGHT YELLOW"
440 PRINT TAB(8);"5-GREY"::::
450 CALL KEY(0,KEY,ST)
460 IF (KEY<49)+(KEY>53)=-1 THEN 450
470 IF KEY<>53 THEN 500
480 SC=15
490 GOTO 510
500 SC=KEY-40
510 CALL CLEAR
520 PRINT "CHOOSE MAP COLOR"
530 PRINT ::"1-BLUE"::"2-GREEN"::::
540 CALL KEY(0,KEY,ST)
550 IF KEY=49 THEN 610
560 IF KEY<>50 THEN 540
570 C(2)=4
580 C(3)=3
590 C(4)=13
600 GOTO 640
610 C(2)=8
620 C(3)=6
630 C(4)=5
640 C(1)=16
650 C(5)=2
660 CALL CLEAR
670 PRINT "DATA ENTRY INSTRUCTIONS"
680 PRINT :"PLEASE ENTER THE VALUES"
690 PRINT "FOR EACH STATE."
700 PRINT :"YOU MAY ENTER A VALUE":"UP TO 8 DIGITS."
710 PRINT "DO NOT USE COMMAS."::::
720 FOR I=1 TO 50
730 INPUT SN$(I)&" = ":V(I)
740 VV(I)=V(I)
750 PRINT
760 NEXT I
770 CALL CLEAR
780 PRINT TAB(6);"WHAT IS THE TITLE"
790 PRINT TAB(6);"OF YOUR MAP?"
800 PRINT
810 PRINT TAB(6);"BECAUSE OF SPACE"
820 PRINT TAB(6);"LIMITATIONS, KEEP"
830 PRINT TAB(6);"YOUR TITLE TO LESS"
840 PRINT TAB(6);"THAN 14 SPACES."::
850 INPUT "TITLE ":TT$
860 CALL CLEAR
870 PRINT TAB(6):"ONE MOMENT PLEASE"
880 PRINT :TAB(9);"* SORTING *"::
890 REM SORT SUBROUTINE
900 FOR N=1 TO 49
910 FOR LT=N+1 TO 50
920 IF VV(N)<=VV(LT)THEN 960
930 LET W=VV(N)
940 LET VV(N)=VV(LT)
950 LET VV(LT)=W
960 NEXT LT
970 NEXT N
980 REM CLASS LIMITS SUBROUTINE
990 X1=VV(10)+(VV(11)-VV(10))/2
1000 X2=VV(20)+(VV(21)-VV(20))/2
1010 X3=VV(30)+(VV(31)-VV(30))/2
1020 X4=VV(40)+(VV(41)-VV(40))/2
1030 REM PRINT LEGEND, TITLE
1040 CALL CLEAR
1050 PRINT TAB(2);VV(1);"-";X1
1060 PRINT TAB(2);X1;"-";X2
1070 PRINT TAB(2);X2;"-";X3;TT$
1080 PRINT TAB(2);X3;"-";X4
1090 PRINT TAB(2);X4;"-";VV(50)
1100 REM ALL STATE PLOTS
1110 K(1)=9
1120 K(2)=10
1130 K(3)=11
1140 K(4)=12
1150 K(5)=13
1160 S(1)=96
1170 S(2)=104
1180 S(3)=112
1190 S(4)=120
1200 S(5)=128
1210 FOR T=1 TO 5
1220 CALL COLOR(K(T),C(T),C(T))
1230 NEXT T
1240 CALL SCREEN(SC)
1250 CALL HCHAR(19,4,96)
```

Copyright © 1983 Emerald Valley Publishing Co.

```
1260 CALL HCHAR(20,4,104)
1270 CALL HCHAR(21,4,112)
1280 CALL HCHAR(22,4,120)
1290 CALL HCHAR(23,4,128)
1300 REM  ALABAMA
1310 NN=1
1320 GOSUB 3960
1330 CALL VCHAR(13,23,S(T),3)
1340 CALL VCHAR(13,24,S(T),3)
1350 REM  ALASKA
1360 NN=2
1370 GOSUB 3960
1380 CALL HCHAR(1,1,S(T),3)
1390 REM  ARIZ
1400 NN=3
1410 GOSUB 3960
1420 CALL VCHAR(11,8,S(T),3)
1430 CALL VCHAR(11,9,S(T),4)
1440 CALL VCHAR(11,10,S(T),4)
1450 REM  ARK
1460 NN=4
1470 GOSUB 3960
1480 CALL VCHAR(11,19,S(T),3)
1490 CALL VCHAR(11,20,S(T),3)
1500 REM  CALIF
1510 NN=5
1520 GOSUB 3960
1530 CALL VCHAR(7,4,S(T),6)
1540 CALL VCHAR(7,5,S(T),7)
1550 CALL VCHAR(11,6,S(T),3)
1560 CALL VCHAR(12,7,S(T),2)
1570 REM  COLO
1580 NN=6
1590 GOSUB 3960
1600 CALL HCHAR(8,11,S(T),4)
1610 CALL HCHAR(9,11,S(T),4)
1620 CALL HCHAR(10,11,S(T),4)
1630 REM  CONN
1640 NN=7
1650 GOSUB 3960
1660 CALL HCHAR(5,30,S(T))
1670 REM  DEL
1680 NN=8
1690 GOSUB 3960
1700 CALL HCHAR(7,30,S(T))
1710 REM  FLA
1720 NN=9
1730 GOSUB 3960
1740 CALL HCHAR(16,24,S(T),5)
1750 CALL HCHAR(17,27,S(T),2)
1760 CALL HCHAR(18,27,S(T),2)
1770 REM  GA
1780 NN=10
1790 GOSUB 3960
1800 CALL HCHAR(13,25,S(T),2)
1810 CALL HCHAR(14,25,S(T),3)
1820 CALL HCHAR(15,25,S(T),3)
1830 REM  HAWAII
1840 NN=11
1850 GOSUB 3960
1860 CALL VCHAR(11,1,ST,3)
1870 CALL VCHAR(13,2,S(T),3)
1880 REM  IDAHO
1890 NN=12
1900 GOSUB 3960
1910 CALL VCHAR(2,7,S(T),5)
1920 CALL VCHAR(4,8,S(T),3)
1930 CALL VCHAR(5,9,S(T),2)
1940 REM  ILL
1950 NN=13
1960 GOSUB 3960
1970 CALL VCHAR(6,20,S(T),4)
1980 CALL VCHAR(6,21,S(T),4)
1990 REM  IND
2000 NN=14
2010 GOSUB 3960
2020 CALL VCHAR(7,22,S(T),3)
2030 CALL VCHAR(7,23,S(T),2)
2040 REM  IOWA
2050 NN=15
2060 GOSUB 3960
2070 CALL HCHAR(5,18,S(T),2)
2080 CALL HCHAR(6,18,S(T),2)
2090 REM  KAN
2100 NN=16
2110 GOSUB 3960
2120 CALL HCHAR(9,15,S(T),3)
2130 CALL HCHAR(10,15,S(T),3)
2140 REM  KY
2150 NN=17
2160 GOSUB 3960
2170 CALL HCHAR(9,23,S(T))
2180 CALL HCHAR(10,21,S(T),4)
2190 REM  LA
2200 NN=18
2210 GOSUB 3960
2220 CALL VCHAR(14,19,S(T),2)
2230 CALL VCHAR(14,20,S(T),3)
2240 REM  MAINE
2250 NN=19
2255 GOSUB 3960
2260 CALL VCHAR(2,31,S(T),2)
2270 REM  MD
2280 NN=20
2290 GOSUB 3960
2300 CALL HCHAR(7,27,S(T),3)
2310 CALL HCHAR(8,29,S(T))
2320 REM  MASS
2330 NN=21
2340 GOSUB 3960
2350 CALL HCHAR(4,30,S(T),2)
2360 REM  MICH
2370 NN=22
2390 GOSUB 3960
2400 CALL VCHAR(4,23,S(T),3)
2410 CALL VCHAR(4,24,S(T),3)
2420 REM  MINN
2430 NN=23
2440 GOSUB 3960
2450 CALL HCHAR(2,18,S(T),3)
2460 CALL HCHAR(3,18,S(T),2)
2470 CALL HCHAR(4,18,S(T),2)
2480 REM  MISS
2490 NN=24
2500 GOSUB 3960
2510 CALL VCHAR(13,21,S(T),3)
2520 CALL VCHAR(13,22,S(T),3)
2530 REM  MO
2540 NN=25
2550 GOSUB 3960
2560 CALL VCHAR(7,18,S(T),4)
2570 CALL VCHAR(7,19,S(T),4)
2580 CALL VCHAR(10,20,S(T))
2590 REM  MONT
2600 NN=26
2610 GOSUB 3960
2620 CALL HCHAR(2,8,S(T),6)
2630 CALL HCHAR(3,8,S(T),6)
2640 CALL HCHAR(4,9,S(T),6)
2650 REM  NEBR
2660 NN=27
2670 GOSUB 3960
2680 CALL HCHAR(6,14,S(T),4)
2690 CALL HCHAR(7,14,S(T),4)
2700 CALL HCHAR(8,15,S(T),3)
2710 REM  NEV
2720 NN=28
2730 GOSUB 3960
2740 CALL VCHAR(7,6,S(T),4)
2750 CALL VCHAR(7,7,S(T),5)
2760 REM  NH
2770 NN=29
2780 GOSUB 3960
2790 CALL HCHAR(3,29,S(T))
2800 REM  NJ
2810 NN=30
```


Copyright © 1983 Emerald Valley Publishing Co.

```
2820 GOSUB 3960
2830 CALL HCHAR(6,29,S(T))
2840 REM N MEX
2850 NN=31
2860 GOSUB 3960
2870 CALL VCHAR(11,11,S(T),4)
2880 CALL VCHAR(11,12,S(T),3)
2890 CALL VCHAR(11,13,S(T),3)
2900 REM N YORK
2910 NN=32
2920 GOSUB 3960
2930 CALL HCHAR(3,27,S(T),2)
2940 CALL HCHAR(4,26,S(T),4)
2950 CALL HCHAR(5,29,S(T))
2960 REM NC
2970 NN=33
2980 GOSUB 3960
2990 CALL HCHAR(11,26,S(T),5)
3000 CALL HCHAR(12,26,S(T),4)
3010 REM N DAK
3020 NN=34
3030 GOSUB 3960
3040 CALL HCHAR(2,14,S(T),4)
3050 CALL HCHAR(3,14,S(T),4)
3060 REM OHIO
3070 NN=35
3080 GOSUB 3960
3090 CALL VCHAR(7,24,S(T),3)
3100 CALL VCHAR(7,25,S(T),3)
3110 REM OKLA
3120 NN=36
3130 GOSUB 3960
3140 CALL HCHAR(11,14,S(T),5)
3150 CALL HCHAR(12,16,S(T),3)
3160 CALL HCHAR(13,16,S(T),3)
3170 REM ORE
3180 NN=37
3190 GOSUB 3960
3200 CALL HCHAR(4,4,S(T),3)
3210 CALL HCHAR(5,4,S(T),3)
3220 CALL HCHAR(6,4,S(T),3)
3230 REM PA
3240 NN=38
3250 GOSUB 3960
3260 CALL HCHAR(5,26,S(T),3)
3270 CALL HCHAR(6,26,S(T),3)
3280 REM RI
3290 NN=39
3300 GOSUB 3960
3310 CALL HCHAR(5,31,S(T))
3320 REM S CAR
3330 NN=40
3340 GOSUB 3960
3350 CALL HCHAR(13,27,S(T),3)
3360 CALL HCHAR(14,28,S(T))
3370 REM S DAK
3380 NN=41
3390 GOSUB 3960
3400 CALL HCHAR(4,14,S(T),4)
3410 CALL HCHAR(5,14,S(T),4)
3420 REM TENN
3430 NN=42
3440 GOSUB 3960
3450 CALL HCHAR(11,21,S(T),5)
3460 CALL HCHAR(12,21,S(T),5)
3470 REM TEX
3480 NN=43
3490 GOSUB 3960
3500 CALL HCHAR(12,14,S(T),2)
3510 CALL HCHAR(13,14,S(T),2)
3520 CALL HCHAR(14,12,S(T),7)
3530 CALL HCHAR(15,13,S(T),6)
3540 CALL HCHAR(16,13,S(T),5)
3550 CALL HCHAR(17,14,S(T),3)
3560 CALL HCHAR(18,15,S(T))
3570 REM UTAH
3580 NN=44
3590 GOSUB 3960
3600 CALL VCHAR(7,8,S(T),4)
3610 CALL VCHAR(7,9,S(T),4)
3620 CALL VCHAR(8,10,S(T),3)
3630 REM VERMONT
3640 NN=45
3650 GOSUB 3960
3660 CALL HCHAR(3,30,S(T))
3670 REM VA
3680 NN=46
3690 GOSUB 3960
3700 CALL HCHAR(8,28,S(T))
3710 CALL HCHAR(9,26,S(T),4)
3720 CALL HCHAR(10,25,S(T),5)
3730 REM WASH
3740 NN=47
3750 GOSUB 3960
3760 CALL HCHAR(2,5,S(T),2)
3770 CALL HCHAR(3,4,S(T),3)
3780 REM W VA
3790 NN=48
3800 GOSUB 3960
3810 CALL HCHAR(7,26,S(T))
3820 CALL HCHAR(8,26,S(T),2)
3830 REM WISC
3840 NN=49
3850 GOSUB 3960
3860 CALL VCHAR(3,20,S(T),3)
3870 CALL VCHAR(3,21,S(T),3)
3880 REM WYO
3890 NN=50
3900 GOSUB 3960
3910 CALL HCHAR(5,10,S(T),4)
3920 CALL HCHAR(6,10,S(T),4)
3930 CALL HCHAR(7,10,S(T),4)
3940 GOTO 3940
3950 REM CLASS CHECK
3960 IF V(NN)<=X1 THEN 4020
3970 IF V(NN)<=X2 THEN 4040
3980 IF V(NN)<=X3 THEN 4060
3990 IF V(NN)<=X4 THEN 4080
4000 T=5
4010 RETURN
4020 T=1
4030 RETURN
4040 T=2
4050 RETURN
4060 T=3
4070 RETURN
4080 T=4
4090 RETURN
4100 END
```

Copyright © 1983 Emerald Valley Publishing Co.

OVERLAND FLOW

TI BASIC

When rain falls to earth, part of it passes into the soil (unless the surface is impervious, such as concrete or asphalt) and the remainder disappears over a period of time either by evaporation or by runoff *(overland flow)* or by both. In most engineering drainage systems, the amount of water lost by evaporation is negligible; thus, drainage must be provided for all rainfall that does not infiltrate the soil or is not stored temporarily in surface depressions (lakes, swamps, etc.) within the drainage area. Until recently, the Rational Method was used for calculating design discharges for storm drains. This method has various drawbacks and is of limited applicability. For that reason, the method used in this program is the Izzard dimensionless hydrograph. This method has been verified in laboratory tests and gives computed overland flow hydrographs agreeing closely with the measured hydrographs. The result of Izzard's method can be used by engineers in the design of drainage facilities for parking lots, airports and highways, etc. (See sample problem.)

Program Description

Input to the *Overland Flow* Program consists of two elements. The first consists of rainfall data and the second consists of physical characteristics.

Standard curves (see Fig.1) may be developed to express rainfall intensity-duration relationships with an accuracy sufficient for drainage problems. Rainfall intensity-duration data have been published by the National Weather Service.

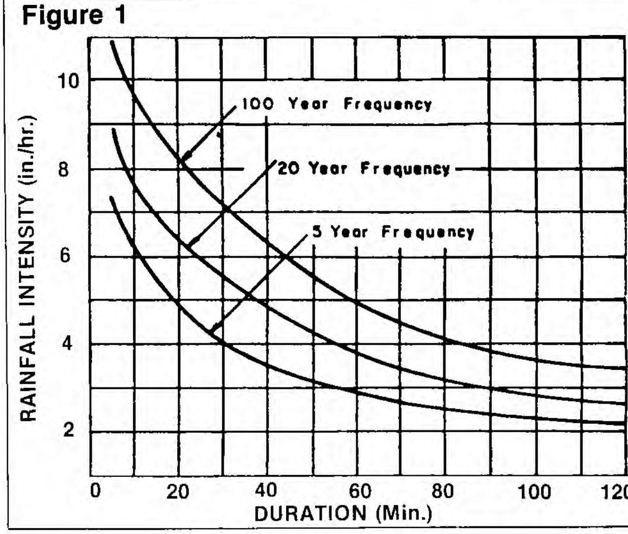
The following physical characteristics are needed: length, width and slope of the area of interest, and a coefficient to describe the surface. The computer program contains a table from which the surface coefficient can be determined.

Output from the program can be displayed on the monitor screen or TI thermal printer. The program displays input data and the overland flow hydrograph in tabular and/or graphic format. The program can calculate and display two hydrographs at any one time. Thus, it is possible to vary the input data and compare the results.

Definition of Terms.

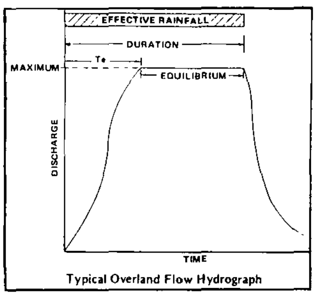
Depression Storage:	Rainwater retained in puddles, ditches and other depressions in soil surface.
Equilibrium:	Occurs when the intensity of effective rainfall is equal to the outflow discharge. See Figure 2.
Equilibrium Time:	Time in minutes to reach the equilibrium condition. See Figure 2.
Infiltration:	Passage of water through the soil surface into the soil.
Intensity:	Effective rainfall intensity in inches per hour. Effective rainfall is that which occurs after depression storage and infiltration capacities are met. See Figure 2.

Figure 1



Copyright © 1983 Emerald Valley Publishing Co.

Maximum Discharge:	The discharge, in cubic feet per second, when equilibrium is reached. See Figure 2.
Roughness Factor:	A coefficient that characterizes the resistance to flow of a particular surface type.
Length:	Distance, in feet, in the direction of slope, on which overland flow occurs. See Figure 3.
Slope:	See Figure 3.
Width:	Distance, in feet, perpendicular to the length, on which overland flow occurs. See Figure 3.



Typical Overland Flow Hydrograph

Figure 2

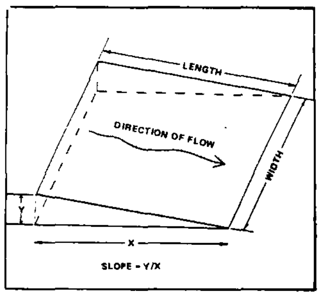


Figure 3

Operating Instructions

Step 1: Insert the cassette into a recorder, type: OLD CS1 and press ENTER. The computer then displays directions for loading the tape.

Step 2: When the cursor appears, type RUN, and press ENTER. When the title screen appears, press any key. Then select the screen or thermal printer as the device for output from the program.

Step 3: After choosing the output device, the computer asks for the input data needed to compute the overland flow hydrograph. Type in the data requested and press ENTER.

Step 4: After all data is entered, the computer will generate a hydrograph and display the menu. Select one of the following options:

1. DISPLAY DATA (GIVEN AND CALCULATED).
2. DISPLAY HYDROGRAPH.
3. COMPUTE ANOTHER HYDROGRAPH AND COMPARE.
4. REDIRECT OUTPUT.
5. ENTER NEW PROBLEM.
6. END PROGRAM.

After completing any of options 1 through 5 the computer returns to the menu.

OPTION 1: DISPLAY DATA (GIVEN AND CALCULATED)—If you select option 1, the computer will display the input data that you entered and the calculated values for equilibrium time and maximum discharge.

OPTION 2: DISPLAY HYDROGRAPH—If you select option 2, the computer asks you if you want the hydrograph displayed in tabular or graphic form or both. The graphic form plots the hydrograph points as percent of maximum discharge versus time. When two hydrographs are plotted, the maximum discharge is the greater of the two hydrograph maximums.

OPTION 3: COMPUTE ANOTHER HYDROGRAPH AND COMPARE—If you select option 3, the computer asks you to enter another set of data in order to calculate another hydrograph. Since the first hydrograph is retained by the computer, this option can be used to vary any of the input data and examine the result (see sample problems). The option can be used as many times as the user wishes. The computer always compares to the original hydrograph computed when the program was initially run. If a subsequent hydrograph is preferred to the original, select option 5 and enter the new hydrograph as the original. Thus, all other hydrographs computed via option 3 will be compared to the new hydrograph.

OPTION 4: REDIRECT OUTPUT—If you select option 4, you change the device to which the output is displayed.

OPTION 5: ENTER NEW PROBLEM—If you select option 5, the program begins again. This option is used to rerun the program without having to type RUN. Also, use this option in conjunction with option 3 to compare several hydrographs and select one that is best suited to the problem.

OPTION 6: END PROGRAM—This option returns the computer to TI BASIC.

EXPLANATION OF THE PROGRAM

Overland Flow

Line Nos.	
160-530	Program initialization: Character assignments and array dimensioning.
540-1080	Data entry.
1090-1490	Calculation of Overland Flow Hydrograph.
1500-1800	Display hydrograph, in tabular form, on video monitor or TI thermal printer.
1810-1990	Display menu and go to portion of program according to option selected.
2000-3200	Display hydrograph, in graphic form, on video monitor or TI thermal printer.
3210-3460	Subroutine to align numbers on display.
3470-3590	Display given and calculated data.
3600-3710	Prepare program to accept and calculate a second hydrograph.
3720-3790	Subroutine to blank and restore screen when displaying information on video monitor.
3930-4220	Scale and label axes of graph.
4230-4250	Common subroutine to check keyboard entry.
4260-4510	Select drive to program output.

Sample Problem 1

A parking lot 300 ft. long in the direction of the slope and 900 ft. wide has a tar and gravel pavement on a slope of .0025. Assuming a uniform rainfall intensity of 2.75 in/hr for 30 minutes, what is the maximum discharge that a gutter should be designed for?

Type RUN, then press ENTER.

Copyright © 1983 Emerald Valley Publishing Co

Sample Problem 1-cont.

Select the thermal printer as the output device.

Enter the following data:

Intensity	2.75
Duration	30
Length	300
Width	900
Slope	.0025
Roughness Factor	.017

Select option 1.

The gutter should be designed for a maximum discharge of 17.2 cfs.

Sample Problem 2

If the parking lot described in problem 1 is resurfaced with asphalt, what effect will this have?

After problem 1 is complete select option 3.

Enter the same data as for problem 1 with the exception of the roughness factor. Enter .007.

Select options 1 and 2.

```
100 REM   ****************
110 REM   * OVERLAND FLOW *
120 REM   ****************
130 REM
140 REM
150 REM
160 CALL CLEAR
170 CALL SCREEN(15)
180 GOSUB 3720
190 PRINT TAB(8);"OVERLAND FLOW"::::::::::::::::
200 PRINT TAB(4);"PRESS ANY KEY TO BEGIN";
210 CALL SOUND(150,600,1)
220 GOSUB 3760
230 GOSUB 4230
240 CALL CLEAR
250 CALL SCREEN(8)
260 PRINT TAB(9);"INITIALIZING"::::::::::::
270 OPTION BASE 1
280 DIM H(2,2,22),IN(2),DU(2),LE(2),WI(2),SL(2),CR(2),TE(2),QW(2),QE(2),TPP(2,2)
290 DEF RD(X)=INT(100*X+.5)/100
300 DEF RD2(X)=INT(1E3*X+.5)/1E3
310 DEF RD4(X)=INT(1E4*X+.5)/1E4
320 DEF RD6(X)=INT(1E6*X+.5)/1E6
330 RESTORE 380
340 FOR I=1 TO 42
350 READ CODE,CH$
360 CALL CHAR(CODE,CH$)
370 NEXT I
380 DATA 99,E0808080818181FF,100,E08080808080808080,101,F880808080808080,102,00000000010101FF
390 DATA 103,000001010101 01FF,104,60909060,105,06090906,106,0000000006090 9060,97,0000241818 24
400 DATA 107,00000000006090906,111,90606090,112,09060609,113,00000000090606090,96,0018242418
410 DATA 114,0000000009060609,121,F0F0F0F0,122,96696996,123,906060906 0909060,64,FFFFFFFFFFFFFFFF
420 DATA 124,906060900609 0906,131,69969669,132,0F0F0F0F,133,09060609609 09060
430 DATA 134,0906060906090906,141,60909060606090,142,06090906906060 90,143,00000000F0F0F0F0
440 DATA 144,0000000096696996,151,609090609060609,152,06090906090606 09,153,0000000069969669
450 DATA 154,00000000F0F0F0F,155,007840404444FF,156,00F040404040404 0,157,00F840404040404 0,93,0906
460 DATA 158,000000000404FF,159,00000404 0404FF,145,,98,00003C3C3C3C,91,00000000000 06090,92,9162040810204689F
470 GOSUB 4260
480 HY=1
490 FOR I=1 TO 2
500 FOR J=1 TO 22
510 H(1,I,J)=0.0
520 NEXT J
530 NEXT I
540 CALL CLEAR
550 CALL SCREEN(8)
560 PRINT "  HYDROGRAPH #";STR$(HY)::
570 PRINT "  ENTER DATA:"::
580 INPUT "   INTENSITY(IN/HR)  ":IN(HY)
590 IN(HY)=RD2(IN(HY))
600 IF IN(HY)<1E4 THEN 630
610 GOSUB 4470
620 GOTO 580
630 PRINT
640 INPUT "   DURATION(MIN)     ":DU(HY)
650 DU(HY)=RD(DU(HY))
660 IF DU(HY)<1E5 THEN 690
670 GOSUB 4470
680 GOTO 640
690 PRINT
700 INPUT "   LENGTH(FT)        ":LE(HY)
710 LE(HY)=RD(LE(HY))
720 IF LE(HY)<1E6 THEN 750
730 GOSUB 4470
740 GOTO 700
750 PRINT
760 INPUT "   WIDTH(FT)         ":WI(HY)
770 WI(HY)=RD(WI(HY))
780 IF WI(HY)<1E6 THEN 810
790 GOSUB 4470
800 GOTO 760
810 PRINT
820 INPUT "   SLOPE(FT/FT)      ":SL(HY)
830 SL(HY)=RD6(SL(HY))
840 IF SL(HY)<=.04 THEN 910
850 CALL SOUND(200,120,1)
860 PRINT :" FOR ACCURATE RESULTS THE":" VALUE FOR SLOPE SHOULD":" BE LESS THAN 0.04.":
870 PRINT " DO YOU WISH TO RE-ENTER":" A VALUE FOR SLOPE?(Y/N)"::
880 GOSUB 4230
890 IF (KEY=89)+(KEY=78)=0 THEN 880
900 IF KEY=89 THEN 820
910 PRINT
920 FOR I=1 TO 350
930 NEXT I
940 CALL CLEAR
950 PRINT "   ENTER ROUGHNESS FACTOR":
960 PRINT "   USING TABLE AS A GUIDE"::
970 PRINT "SURFACE TYPE        FACTOR":
980 PRINT "------------        ------"
990 PRINT "SMOOTH ASPHALT PAVE..0.0070"::"TAR AND SAND PAVE....0.0075"::"CRUSHED-SLATE PAPER..0.0082"::
1000 PRINT "CONCRETE PAVEMENT....0.0120"::"TAR & GRAVEL PAVE....0.0170"::
1010 PRINT "CLOSELY CLIPPED SOD..0.0260"::"DENSE BLUEGRASS TURF.0.0600":::
1020 INPUT "    ROUGHNESS FACTOR= ":CR(HY)
1030 CR(HY)=RD4(CR(HY))
```


Copyright © 1983 Emerald Valley Publishing Co.

```
1040 IF CR(HY)<1 THEN 1070
1050 GOSUB 4470
1060 GOTO 1020
1070 FOR I=1 TO 250
1080 NEXT I
1090 CALL CLEAR
1100 CALL SCREEN(3)
1110 PRINT "          COMPUTING":::::::
:::::
1120 QE(HY)=IN(HY)*LE(HY)/43200
1130 KX=((0.0007*IN(HY))+CR(HY))/SL(HY)
^(1/3)
1140 DE=KX*LE(HY)*QE(HY)^(1/3)
1150 TE(HY)=DE/(30*QE(HY))
1160 KNT=0
1170 FOR J=1 TO 10
1180 IF (J/10*TE(HY))>DU(HY)THEN 1190 E
LSE 1210
1190 KNT=J
1200 GOTO 1250
1210 H(HY,2,J)=J/10*TE(HY)
1220 NEXT J
1230 H(HY,2,11)=((DU(HY)-TE(HY))*0.5)+T
E(HY)
1240 H(HY,2,12)=DU(HY)
1250 QW(HY)=QE(HY)*WI(HY)
1260 RESTORE 1270
1270 DATA .01,.06,.18,.38,.55,.7,.83,.9
,.94,1,1,1
1280 FOR J=1 TO 12
1290 IF (J>KNT)+(KNT<>0)=-2 THEN 1340
1300 READ MULT
1310 H(HY,1,J)=MULT*QW(HY)
1320 NEXT J
1330 H(HY,1,12)=QW(HY)
1340 IF KNT=0 THEN 1360
1350 QW(HY)=H(HY,1,KNT-1)
1360 DO=(CR(HY)/SL(HY)^(1/3))*LE(HY)*QE
(HY)^(1/3)
1370 FOR I=1 TO 9
1380 H(HY,2,I+12)=(DO/(120*QW(HY)/WI(HY
)))*(((1-I/10)^(-2/3))-1)
1390 H(HY,2,I+12)=H(HY,2,I+12)+DU(HY)
1400 H(HY,1,I+12)=(1-I/10)*QW(HY)
1410 NEXT I
1420 H(HY,2,22)=(DO/(120*QW(HY)/WI(HY))
)*6.368064+DU(HY)
1430 TPP(HY,1)=0
1440 TPP(HY,2)=0
1450 H(HY,1,22)=.05*QW(HY)
1460 IF KNT<>0 THEN 1810
1470 TPP(HY,1)=.25*(DU(HY)-TE(HY))+TE(H
Y)
1480 TPP(HY,2)=.75*(DU(HY)-TE(HY))+TE(H
Y)
1490 GO TO 1810
1500 CALL CLEAR
1510 IF FILE=0 THEN 1530
1520 PRINT "        HYDROGRAPH PRINTING"::
:::::::::
1530 FOR I=1 TO HY
1540 PRINT #FILE:TAB(9);"HYDROGRAPH #";
STR$(I):
1550 IF FILE=0 THEN 1570
1560 PRINT #FILE:
1570 PRINT #FILE:"   TIME(MIN)","DISCHAR
GE(CFS)":
1580 IF FILE=0 THEN 1600
1590 PRINT #FILE:"   ---------","-------
-------":
1600 FOR J=1 TO 22
1610 IF J=11 THEN 1710
1620 IF H(I,2,J)=0 THEN 1710
1630 IL=4
1640 N=H(I,2,J)
1650 FL=1
1660 GOSUB 3210
1670 N=H(I,1,J)
1680 FL=3
1690 GOSUB 3210
1700 PRINT #FILE:TAB(4);E$;TAB(16);D$
1710 NEXT J
1720 IF FILE<>0 THEN 1770
1730 PRINT "  PRESS ANY KEY TO CONTINUE"
;
1740 GOSUB 4230
1750 CALL CLEAR
1760 GO TO 1780
1770 PRINT #FILE::::
1780 NEXT I
1790 IF TST=1 THEN 1810
1800 GO TO 2030
1810 CALL CLEAR
1820 GOSUB 3720
1830 CALL SCREEN(15)
1840 PRINT "  PRESS";TAB(9);"TO":"  -----
";TAB(9);"--":"    1     DISPLAY DAT
A(GIVEN":TAB(10);"AND CALCULATED)"
::
1850 PRINT "   2     DISPLAY HYDROGRAPH"
::
1860 PRINT "   3     COMPUTE ANOTHER":TA
B(10);"HYDROGRAPH AND":TAB(10);"CO
MPARE"::
1870 PRINT "   4     REDIRECT OUTPUT"::"
    5     ENTER NEW PROBLEM"::"    6
     END PROGRAM"::::
1880 CALL SOUND(200,600,1)
1890 GOSUB 3760
1900 GOSUB 4230
1910 IF (KEY<49)+(KEY>54)=-1 THEN 1920
ELSE 1940
1920 CALL SOUND(250,110,1)
1930 GO TO 1900
1940 CALL SOUND(100,666,1)
1950 CALL CLEAR
1960 CALL SCREEN(8)
1970 ON (KEY-48)GO TO 3470,3800,3650,20
00,480,1980
1980 CALL CLEAR
1990 END
2000 GOSUB 4260
2010 GO TO 1810
2020 CALL SOUND(150,600,1)
2030 CALL CLEAR
2040 F2=0
2050 QMAX=QW(1)
2060 IF HY=1 THEN 2090
2070 IF QMAX>=QW(2)THEN 2090
2080 QMAX=QW(2)
2090 TMAX=H(1,2,22)
2100 IF HY=1 THEN 2230
2110 IF TMAX>=H(2,2,22)THEN 2130
2120 TMAX=H(2,2,22)
2130 CALL CLEAR
2140 IF QMAX>=10 THEN 2150 ELSE 2160
2150 QMAX=RD(QMAX)
2160 PRINT #F2:TAB(12);"LEGEND":TAB(12)
;"------":"  X = HYDROGRAPH #1"::"
0 = HYDROGRAPH #2"::
2170 PRINT #F2:TAB(2);CHR$(64);" = COIN
CIDENCE OF 1 & 2"::"  MAX DISCHARGE
(CFS)=";RD2(QMAX):::::::
2180 IF (FILE=0)+(F2>0)<0 THEN 2210
2190 F2=FILE
2200 GOTO 2160
2210 FOR I=1 TO 700
2220 NEXT I
2230 CALL CLEAR
2240 CALL SCREEN(8)
2250 FOR I=9 TO 16
2260 CALL COLOR(I-8,2,1)
2270 CALL COLOR(I,2,16)
2280 NEXT I
2290 CALL SOUND(150,600,1)
2300 FOR I=1 TO 20
2310 CALL HCHAR(I,9,145,20)
2320 NEXT I
```

Copyright © 1983 Emerald Valley Publishing Co.

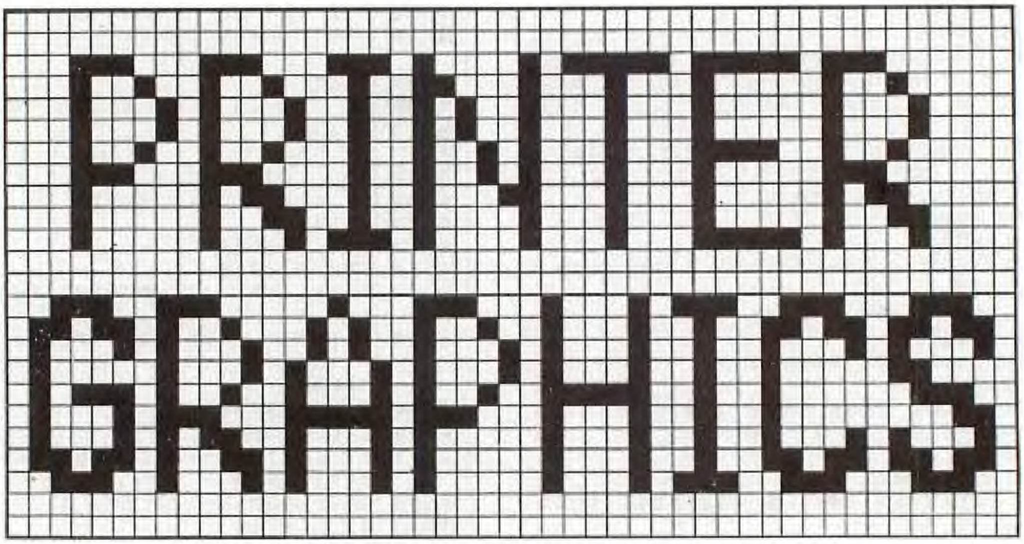
```
2330 CALL VCHAR(1,9,101)
2340 CALL VCHAR(2,9,100,4)
2350 CALL VCHAR(6,9,101)
2360 CALL VCHAR(7,9,100,4)
2370 CALL VCHAR(11,9,101)
2380 CALL VCHAR(12,9,100,4)
2390 CALL VCHAR(16,9,101)
2400 CALL VCHAR(17,9,100,4)
2410 CALL VCHAR(20,9,99)
2420 CALL HCHAR(20,10,102,3)
2430 CALL HCHAR(20,13,103)
2440 CALL HCHAR(20,14,102,4)
2450 CALL HCHAR(20,18,103)
2460 CALL HCHAR(20,19,102,4)
2470 CALL HCHAR(20,23,103)
2480 CALL HCHAR(20,24,102,4)
2490 CALL HCHAR(20,28,103)
2500 GOSUB 3930
2510 FOR J=1 TO HY
2520 FOR I=1 TO 22
2530 IF (I=11)+(I=12)+(KNT=0)<>-2 THEN
      2570
2540 P1=TPP(J,I-10)/TMAX*19.99999
2550 P2=QW(J)/QMAX*19.99999
2560 GOSUB 2930
2570 P1=H(J,2,I)/TMAX*19.99999
2580 P2=H(J,1,I)/QMAX*19.99999
2590 GOSUB 2930
2600 NEXT I
2610 NEXT J
2620 IF FILE=0 THEN 2870
2630 NUS$="                    ***GRAPH PRINTING***
     "
2640 FOR I=1 TO LEN(NUS$)
2650 CALL VCHAR(24,I,ASC(SEG$(NUS$,I,1)
     ))
2660 NEXT I
2670 FOR I=1 TO 23
2680 B$=""
2690 FOR J=1 TO 32
2700 CALL GCHAR(I,J,A)
2710 IF (J=9)+(I>20)=-2 THEN 2820
2720 IF (I=20)+(J<9)+(J>28)=-2 THEN 282
     0
2730 GOSUB 4200
2740 IF (A=91)+(A=93)<>-1 THEN 2760
2750 A=145
2760 IF A<>92 THEN 2780
2770 A=37
2780 IF (A<104)+(A>154)+(A=145)=-1 THEN
      2820
2790 A=INT(A/10)+86
2800 IF A<99 THEN 2820
2810 A=98
2820 B$=B$&CHR$(A)
2830 NEXT J
2840 PRINT #FILE:B$
2850 NEXT I
2860 PRINT #FILE:::::
2870 ZZ$="    PRESS ANY KEY TO CONTINUE"
2880 FOR I=1 TO LEN(ZZ$)
2890 CALL VCHAR(24,I,ASC(SEG$(ZZ$,I,1))
     )
2900 NEXT I
2910 GOSUB 4230
2920 GOTO 1810
2930 T1=P1-INT(P1)
2940 T2=P2-INT(P2)
2950 Y=20-INT(P2)
2960 X=9+INT(P1)
2970 IF Y=20 THEN 3200
2980 IF X=9 THEN 3200
2990 IF T1<.5 THEN 3030
3000 IF T2<.5 THEN 3060
3010 PS=2
3020 GOTO 3090
3030 IF T2<.5 THEN 3080
3040 PS=1
3050 GOTO 3090
3060 PS=4
3070 GOTO 3090
3080 PS=3
3090 IF J=2 THEN 3120
3100 CHAR=PS+110
3110 GOTO 3180
3120 CALL GCHAR(Y,X,CHAR2)
3130 IF CHAR2=145 THEN 3170
3140 IF (CHAR2<108)+(CHAR2>103)=-2 THEN
      3200
3150 CHAR=CHAR2*10-990+PS
3160 GOTO 3180
3170 CHAR=103+PS
3180 CALL VCHAR(Y,X,CHAR)
3190 CALL SOUND(100,220*J,3)
3200 RETURN
3210 N2=10^FL
3220 N1=ABS(N)+.5/N2
3230 IP=INT(N1)
3240 FP=INT(N2*(N1-IP))+N2
3250 D$=STR$(IP)
3260 L=LEN(D$)
3270 IF (IL=0)+(IP=0)<>-2 THEN 3300
3280 D$=""
3290 L=0
3300 IF IL<L THEN 3400
3310 IF IL<=L THEN 3350
3320 D$=" "&D$
3330 L=L+1
3340 GOTO 3310
3350 IF FL<=0 THEN 3390
3360 D$=D$&"."&SEG$(STR$(FP),2,FL)
3370 IF FL=3 THEN 3390
3380 E$=D$
3390 RETURN
3400 D$="*"
3410 FOR I=1 TO (IL+FL)
3420 D$=D$&"*"
3430 NEXT I
3440 IF FL=3 THEN 3460
3450 E$=D$
3460 RETURN
3470 CALL CLEAR
3480 IF FILE=0 THEN 3500
3490 PRINT "          DATA PRINTING":::::
     ::::::::
3500 FOR I=1 TO HY
3510 PRINT #FILE:TAB(7);"DATA--HYDROGRA
     PH #";STR$(I)::
3520 PRINT #FILE:"GIVEN:":"-------":" IN
     TENSITY(IN/HR)=  ";IN(I)::" DURATIO
     N(MIN)=     ";DU(I)::
3530 PRINT #FILE:" LENGTH(FT)=      ";
     LE(I)::" WIDTH(FT)=       ";WI(I)
     ::" SLOPE(FT/FT)=      ";SL(I)::
3540 PRINT #FILE:" ROUGHNESS FACTOR= ";
     CR(I)::
3550 PRINT #FILE:"CALCULATED:":"-------
     ----":" EQUILIB. TIME(MIN)= ";RD(T
     E(I))::
3560 PRINT #FILE:" MAX DISCHARGE(CFS)=
     ";RD2(QW(I))::
3570 IF FILE<>0 THEN 3620
3580 PRINT " PRESS ANY KEY TO CONTINUE"
     :
3590 GOSUB 4230
3600 CALL CLEAR
3610 GO TO 3630
3620 PRINT #FILE::::
3630 NEXT I
3640 GO TO 1810
3650 HY=2
3660 FOR I=1 TO 2
3670 FOR J=1 TO 22
3680 H(2,I,J)=0.0
3690 NEXT J
3700 NEXT I
3710 GO TO 540
3720 FOR I=1 TO 8
```


Copyright © 1983 Emerald Valley Publishing Co.

```
3730 CALL COLOR(I,1,1)
3740 NEXT I
3750 RETURN
3760 FOR I=1 TO 8
3770 CALL COLOR(I,2,1)
3780 NEXT I
3790 RETURN
3800 CALL CLEAR
3810 GOSUB 3720
3820 PRINT TAB(6);"HYDROGRAPH DISPLAY":
     TAB(6);"-----------------"::TAB(7
     );"PRESS   FOR"::TAB(9);"1     TAB
     LE":
3830 PRINT :TAB(9);"2     GRAPH"::TAB(9
     );"3      BOTH"::::::
3840 CALL SOUND(200,600,1)
3850 GOSUB 3760
3860 GOSUB 4230
3870 IF (KEY<49)+(KEY>51)=-1 THEN 3880
     ELSE 3900
3880 CALL SOUND(250,110,1)
3890 GOTO 3860
3900 CALL SOUND(150,666,1)
3910 TST=KEY-48
3920 ON TST GOTO 1500,2030,1500
3930 TI3$="100 75 50 25  0"
3940 FOR I=1 TO 5
3950 FOR J=1 TO 3
3960 CALL VCHAR(5*I-4,J+5,ASC(SEG$(TI3$
     ,3*I+J-3,1)))
3970 NEXT J
3980 NEXT I
3990 SC=1
4000 IF TMAX<=132.666 THEN 4020
4010 SC=10
4020 FOR I=1 TO 4
4030 TM$=STR$(I*TMAX/(4*SC)+.5)
4040 CALL HCHAR(21,I*5+8,ASC(SEG$(TM$,1
     ,1)))
4050 IF SEG$(TM$,2,1)="." THEN 4070
4060 CALL HCHAR(21,I*5+9,ASC(SEG$(TM$,2
     ,1)))
4070 IF (SC=1)+(I=4)+(TMAX>99.4999999)
     <>-3 THEN 4090
4080 CALL HCHAR(21,30,ASC(SEG$(TM$,3,1)
     ))
4090 NEXT I
4100 TI2$=CHR$(91)&CHR$(92)&CHR$(93)&"O
     F MAX DISCHARGE"
4110 FOR I=1 TO LEN(TI2$)
4120 CALL VCHAR(I+1,5,ASC(SEG$(TI2$,I,1
     )))
4130 NEXT I
4140 TI4$="   TIME(MIN)"
4150 IF SC=1 THEN 4170
4160 TI4$="TIME(MIN) X 10"
4170 FOR I=1 TO LEN(TI4$)
4180 CALL VCHAR(23,I+11,ASC(SEG$(TI4$,I
     ,1)))
4190 NEXT I
4200 IF (I<>20)+(J<>9)=-2 THEN 4220
4210 A=A+56
4220 RETURN
4230 CALL KEY(0,KEY,ST)
4240 IF ST<=0 THEN 4230
4250 RETURN
4260 CALL CLEAR
4270 GOSUB 3720
4280 PRINT TAB(4);"OUTPUT DESTINATION":
     TAB(4);"------------------"::
4290 PRINT "   PRESS   FOR"::"      1
       SCREEN"::"     2      THERMAL PRI
     NTER"::::::
4300 GOSUB 3760
4310 GOSUB 4230
4320 IF (KEY<49)+(KEY>50)=-1 THEN 4330
     ELSE 4350
4330 CALL SOUND(250,110,1)
4340 GOTO 4310
4350 CALL SOUND(100,666,1)
4360 FILE=0
4370 IF KEY=49 THEN 4460
4380 FILE=1
4390 DVC$="TP.U.S"
4400 IF KEY=50 THEN 4420
4410 DVC$="RS232"
4420 IF DFG$="" THEN 4440
4430 CLOSE #1
4440 OPEN #1:DVC$,OUTPUT
4450 DFG$="1"
4460 RETURN
4470 PRINT :
4480 CALL SOUND(300,110,1)
4490 PRINT " NUMBER OUTSIDE NORMAL RANG
     E":
4500 PRINT "  ENTER":
4510 RETURN
```

Copyright © 1983 Emerald Valley Publishing Co.

Programming



TI BASIC

The special block graphics character sets that are built into some printers can be extremely useful. In the business world, for example, applications might include the production of charts and graphs, the printing of business forms, or even the design of a letterhead.

The following short program demonstrates how DATA statements are used to format selected graphics characters to produce a letterhead. The DATA statements here are for use with the Epson MX-80 printer (without the GRAFTRAX option) but can be easily modified to accommodate any printer with similar block graphics capabilities. Keep in mind that this is a *shell* program; you can plan the DATA statements to direct the printer to produce virtually any design or pattern (within the limits of the resident block graphics set). The actual graphic design (the letterhead) in this example is unimportant; understanding how to plan and implement it is crucial.

DATA statements are read sequentially from left to right, using the READ statement. The Epson MX-80 printer uses numerical codes 160 to 223 (ASCII 32 to 95 with a 1 for the 8th bit) to generate graphics characters already defined within the printer. Each graphics character is made up of one to six squares within a 2 × 3 matrix as indicated below.

1	2
4	8
16	32

(The numbers within the squares are not important if you have a coding table in front of you. They represent a particular manufacturer's coding of the matrix print head. For example, numerical code 165 would produce the fifth character in the set, and would cause wires 1 and 4 to fire; if we want the 21st character, wires 1, 4, and 16 would be fired.)

The key part of the program lies in lines 430-470. This controls what will happen when a DATA statement is read. If the first DATA cell is a number greater than 100, that character will be printed. If the first DATA cell is a number greater than 0 but less than 100, the program will read the second DATA cell, and then print it the number of times specified in the first DATA cell. For example, if the first DATA element is 8, and the next is 160 (a blank space), the computer will print a blank space 8 times. This lessens the amount of required DATA when it is necessary to repeat the same character several times. If the first DATA cell read is equal to 0, a Carriage Return will be executed.

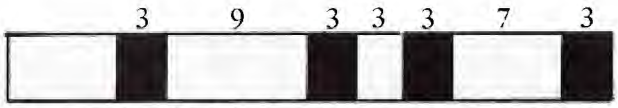
Regular text can be printed on the same line with the graphics. In this program a negative number inserted in the DATA statement signals the program to print the text. The value of the negative number designates which message is to be printed out. This can also be used to change the printer's type style, if your printer has that capability. For example, if you want to print a graphics pattern on the left, and a printed message on the right, you would place the negative number just before the zero that causes the Carriage Return. In that case, a message will be printed—according to the directions in line 470—on the same line *before* the Carriage Return.

Every time a character is printed—whether graphics, text, or a control character—it should always be followed by a semicolon. This will insure that the next printed character will be on the same line, until the Carriage Return is executed. The only exception to this is in line 610 where I wanted the Carriage Return to be executed.

The following is an example of the DATA and the graphics line it creates. This line can be found in the 8th print line of the letterhead.

```
270   DATA 7,160,3,223,9,160,3,223,3,160,3,223,7
      160,3,223,−4,0
```


Copyright © 1983 Emerald Valley Publishing Co.



In this graphics line, I first needed to put seven blank spaces between the first character position and the first graphics character. This was done with the first two DATA cells. When the program first READs A, its value is 7. Because this value is less than 100 and greater than 0, the program will READ B, the next DATA cell. The value of B is 160. ASCII (160) is a blank character on the MX-80, so the program will now PRINT B (blank), A (7) times. On the next cycle the value stored in A will be 3, and the value stored in B will be 223. This will cause B (whose value is the ASCII code for a solid 2 × 3 block) to be printed 3 times. This process is continued until a value less than or equal to zero is encountered.

If the value in A is a negative number, the program will branch off to a subroutine which will PRINT a text message. In the above example, the value −4 caused the message "FOR USERS of TI-99/4" to be printed. These subroutines are extremely versatile; you can change the type style, print a message as I have done, continue the program to do calculations, or run program lines.

Note: All the data necessary for one entire print line is contained in a *single* DATA statement (except for lines 310-320 and lines 330-340). This makes the program a little easier to debug, because you don't have the confusion of counting across statement boundaries to find character positions and their corresponding codes.

EXPLANATION OF THE PROGRAM *LETTERHEAD*

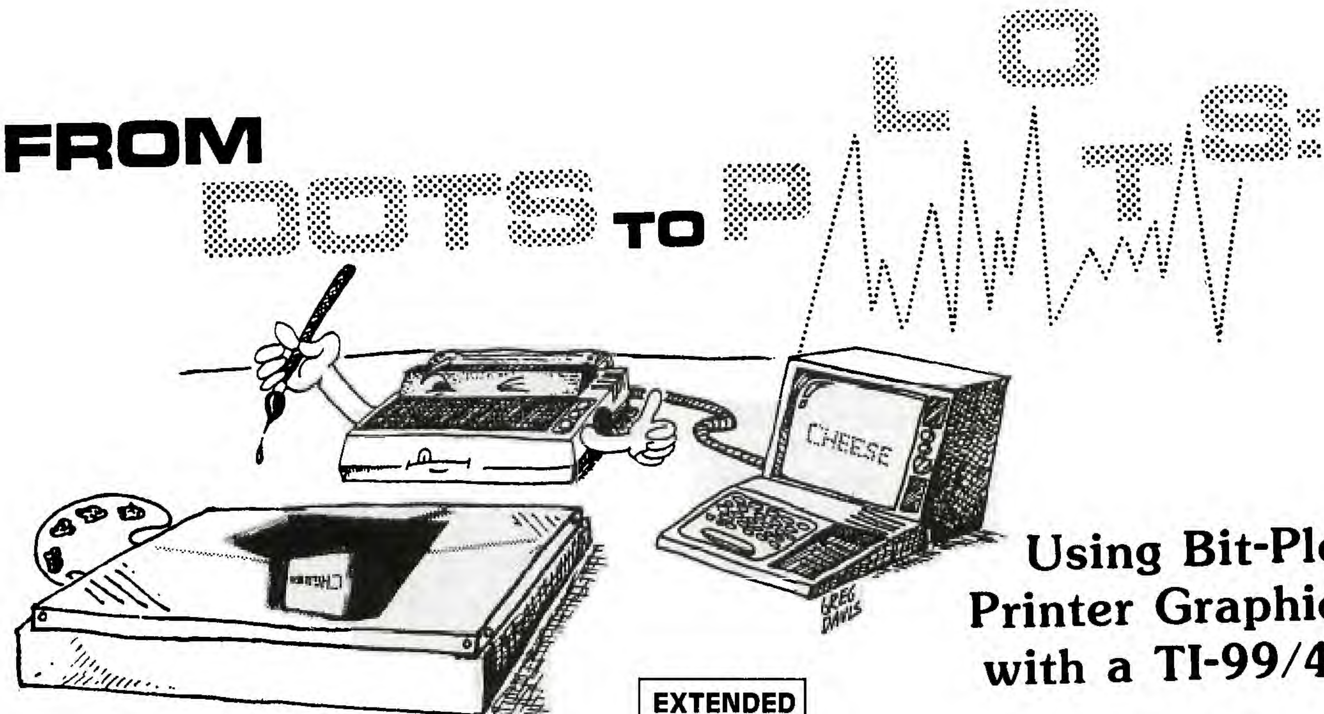
Line Nos.

Line Nos.	
200-390	Contains DATA formated to print the *99'er Magazine* letterhead.
400	OPENs a line to the RS-232 interface for output to the printer.
410	Sets the printer for "Double Strike" mode.
420	Sets the printer for "Emphasized" mode.
430	READs the DATA statement and stores the result in A.
440	Tests A; if A is greater than 100, then PRINT the character stored in A.
450	Tests A; if A is greater than 0 and less than 100, then READ B; PRINT B, A times.
460	Tests A; if A is equal to zero, then do a Carriage Return. This marks the end of a line.
470	A equals a negative number at this point; ABS(A) controls the branching of subroutines for special tasks, e.g., PRINT text.
490-500	Subroutine to execute the Carriage Return.
510-520	Subroutine to PRINT the value in A.
530-570	Subroutine to READ B, and PRINT B, A times.
580-710	Subroutine to print normal text instead of graphics.
720-740	End-of-print message on the screen; END of program.

```
100 REM   **************
110 REM   **          **
120 REM   **   99'ER  **
130 REM   **          **
140 REM   **LETTERHEAD**
150 REM   **          **
160 REM   **************
170 REM
180 REM
190 REM
200 DATA 10,223,2,160,10,223,160,223,221,0
210 DATA 2,223,6,160,2,223,2,160,2,223,6,160,2,223,160,162,223,0
220 DATA 2,223,6,160,2,223,2,160,2,223,6,160,2,223,0
230 DATA 2,223,6,160,2,223,2,160,2,223,6,160,2,223,3,160,8,223,2,160,2,223,2,160,216,2,223,0
240 DATA 10,223,2,160,10,223,3,160,2,223,4,160,2,223,2,160,2,223,192,222,167,2,163,0
250 DATA 7,160,3,223,9,160,3,223,3,160,2,223,4,160,2,223,2,160,3,223,161,0
260 DATA 7,160,3,223,9,160,3,223,3,160,8,223,2,160,3,223,0
270 DATA 7,160,3,223,9,160,3,223,3,160,3,223,7,160,3,223,-4,0
280 DATA 7,160,3,223,9,160,3,223,3,160,3,223,7,160,3,223,-5,0
290 DATA 7,160,3,223,9,160,3,223,3,160,8,223,2,160,3,223,-6,0
300 DATA -7,0
310 DATA 25,160,183,180,182,181,160,192,166,196,2,160,183,2,163,165,160,192,166,196,2,160,2,163,211
320 DATA 165,160,162,203,163,160,183,196,160,181,160,183,2,163,165,0
330 DATA 25,160,181,162,160,181,160,189,172,172,181,160,213,208,210,181,160,189
340 DATA 2,172,181,160,220,211,208,176,160,192,218,208,160,181,162,196,181,160,215,211,209,180,0
350 DATA 0
360 DATA -1,0
370 DATA -2,0
380 DATA -3,0
390 DATA -8
400 OPEN #1:"RS232.BA=9600.DA=8.PA=N"
410 PRINT #1:CHR$(27);"G"
420 PRINT #1:CHR$(27);"E"
430 READ A
440 IF A>100 THEN 510
450 IF A>0 THEN 530
460 IF A=0 THEN 490
470 ON ABS(A)GOSUB 580,600,620,640,660,680,700,720
480 GOTO 430
490 PRINT #1
500 GOTO 430
510 PRINT #1:CHR$(A);
520 GOTO 430
530 READ B
540 FOR X=1 TO A
550 PRINT #1:CHR$(B);
560 NEXT X
570 GOTO 430
580 PRINT #1:TAB(25);"EMERALD VALLEY PUBLISHING CO.";
590 RETURN
600 PRINT #1:TAB(33);"P.O. BOX 5537";
610 RETURN
620 PRINT #1:TAB(29);"EUGENE. OREGON 97405";
630 RETURN
640 PRINT #1:TAB(41);"FOR USERS OF TI-99/4";
650 RETURN
660 PRINT #1:TAB(41);"AND OTHER TMS9900-BASED";
670 RETURN
680 PRINT #1:TAB(41);"PERSONAL COMPUTER SYSTEMS";
690 RETURN
700 PRINT #1:TAB(63);"TM";
710 RETURN
720 PRINT "ALL DONE WITH LETTER HEAD"
730 CLOSE #1
740 END
```

Copyright © 1983 Emerald Valley Publishing Co.

FROM DOTS TO PLOTS:



Using Bit-Plot Printer Graphics with a TI-99/4A

EXTENDED BASIC

With TI's 99/4 Impact Printer, we can explore the world of *bit plot* printer graphics. The following program will also work with other printers (Epson's MX-80 with the Graftrax-80 option, for instance) when suitable modifications are made to the program.

Bit-Plot Graphics

In bit-image mode, the printer produces in one dot-column a character which may have any combination of the eight dots in the printhead. This makes it possible to duplicate exactly the 8 × 8 pixel graphics characters of the TI-99/4 and TI 99/4A by printing 8 columns of up to 8 dots on the printer. The printer dots are turned on in accordance with a binary format. For example, sending CHR$(0) to the printer will produce a blank space, one dot-column wide; CHR$(1) will print only the bottom dot; CHR$(7) will print the bottom three dots, CHR$(255) will print all 8 dots, and so forth (see Figure 1). Under software control you may select either 480 or 960 dot-per-line resolution. This means that to print a full line in the 960 dot mode

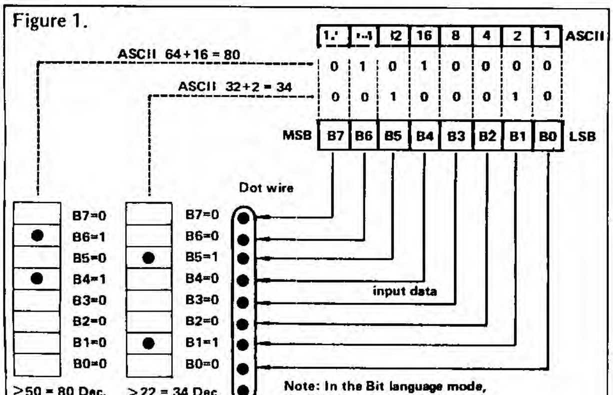


Figure 1.

you would have to print a dot-column character 960 times. The following program, for example, would print a line with only the bottom dot "on" across the entire page width:

```
10 FOR X = 1 TO 960
20 PRINT #1: CHR$(1)
30 NEXT X
40 END
```

In the 960 mode, the line would appear solid with no space between the dots. In the 480 mode, the line would have small but visible spaces between the dots.

> [**Note**: There are two options in the 960 mode: the first at *half* the speed of the 480 mode, and the second at the *same* speed. This second mode may only be used by high-speed Assembly Language driver routines. BASIC Interpreters are too slow in execution to print in this mode. If you try this mode using Extended BASIC you will lose many of the dots on each line. When you use this high-speed mode, there is still another restriction: The same needle may not be struck twice in a row because the needles take 2 microseconds to hit and return to seat. Printing at 480 speed, the print head passes over a dot portion every microsecond. For this reason it is impossible to strike the same needle twice in a row at this high speed. If you attempt a second stike, the printer will automatically toss away the second consecutive dot. The printer will also print bi-directionally in this mode. It should be noted that there is some misalignment between passes of the printhead from opposite directions. This will vary from printer to printer and must be compensated for with computer software.—Ed.]

To leave the TI-99/4's standard text mode and enter the bit-image graphics mode, you must first send CHR$(27); "L" or "K"; CHR$(X); CHR(Y); to the printer. The ESCape "K" code will assign the 480 mode to the printer; "L" will assign it the 960 mode. You must then tell the printer how many graphics columns or characters are to be printed. This is done with CHR$(X);CHR$(Y) where $0<X<255$, and $0<Y<3$. The number of columns of dots or characters to follow is equal to $(Y*256)+X$.

The only problem I have encountered with this convention is a difficulty with intermixing graphics and standard characters on the same line without complicated program-

Copyright © 1983 Emerald Valley Publishing Co.

ming and file structures. The simplest method is to store a CHR$(X) in a file or data statement and then print CHR$(X) for each dot column across the page. This may be time consuming and require more disk, tape or data space, but it allows for the least complicated program [and consequently is the simplest way to get you started using this versatile graphics mode.—Ed.]

To program the graphics you must know how to format the OPEN statement. First, you must tell the RS232 port to output 8 bits instead of 7; then tell it to supress the automatic carriage return and linefeed with the .CRLF software switch. The statement in line 560 will read:

OPEN #3: "RS232.BA=9600.DA=8.CRLF"

[If you have the Epson MX-80 with the Graftrax-80 option, it will read:

OPEN #3: "RS232.BA=9600.DA=8.PA=N.CRLF".—Ed.]

The Program

There are three main sections in the program. The first part is a disk initialization subroutine. This routine will open a file on a blank disk with the following parameters: RELATIVE—random access of file records, and INTERNAL,FIXED 24—a fixed record length of 24 to store 12 CHAR$(X) values or 12 dot-columns of information.

It is possible to store up to 3570 such records on one 5¼" single-density disk. The process of initialization is therefore very slow and takes about half an hour. The initialization program will open the file and print CHR$(0) to all records. This helps speed file building in the second section of the program. When a large clear space on the paper is required, you do not need to enter all these zeros.

The second section of the program is a form of "word processor," only here it is designed to handle *numbers* from 0 to 255. The program works with 20 file records at a time (240 character variables). Each group of 240 variables will be called one *created line*. The present line being worked on is displayed at the top of the screen. The next value displayed is the position in that line, from 1 to 240. Below the position indicator, the present CHR$ value at that position is displayed. Below that, the computer asks you for the new value—from 0 to 255—that you want to assign to that position. Several single-keystroke commands are available to help you manipulate the data. If you merely press ENTER without touching any other key, the value of 0 will be assigned to the position indicated. The following is a list of commands and their explanations:

P—prints one line of data (240 dot positions of the line you are presently working on).
L—lists all 240 variables on the screen for inspection.
N—lets you jump to a new line number and position—a process that would take too long with the arrow keys alone (below). Screen prompts will guide you. If you just hit ENTER, the program will default to the current line number or position without changing anything.
E—decrements the line number by 1.
X—increments the line number by 1.
S—decrements the position in a line by 1
D—increments the position in a line by 1.
Z—returns the user to the main menu screen.

After you enter a valid numeric value and press ENTER, the position in the line will automatically increment by 1 to the next record. After you enter the 240th record of the line, the previous line number will increment to the next line, and the position will return to number 1. The previous line will also be automatically stored on disk. (Note: Any time you change line numbers, the current line will also automatically go into disk storage. If you plan to exit to the main menu or to turn off the system, you should first go to any other line so that the data are stored; otherwise the data on that line will be lost.)

The final part of the program is the routine that prints your graphics. There are several options in this section. First is the option to print single density (480 dots per line) or double density (960 dots per line).

The second option is the line width: A line width of 240 will print one of the created lines in the create-file section, (240 dots); a line width of 480 will print 2 of your created lines according to the chosen parameters.

The last option is the number of lines you want printed: You should specify the actual number of lines to appear on the paper, not necessarily the number of created lines. For example, if you want to print 5 lines with 480 width, you will actually be printing 10 created lines.

Print Line	"created lines"			
1 –	Line 0	Line 1		5 printer lines = 10 "created lines" @ 480 width
2 –	Line 2	Line 3		
3 –	Line 4	Line 5		
4 –	Line 6	Line 7		
5 –	Line 8	Line 9		
1 –	Line 0	Line 1	Line 2	3 printer lines = 9 "created lines" @ 720 width
2 –	Line 3	Line 4	Line 5	
3 –	Line 6	Line 7	Line 8	

EXPLANATION OF THE PROGRAM
DOTS TO PLOTS

Line Nos.	
100-170	REM.
180-190	Initializes variables and arrays.
210-290	Data statement.
300	Subroutine to read data and display.
310	Subroutine to read data, display and accept input.
320	Reads data only.
330	Initializes colors.
340-380	Checks for proper disk in drive #1.
390-400	Opens file #2 on disk #1.
410	Clears screen with left to right scroll.
420-430	Controls subroutine to check for proper disk in drive #1.
440	Reads multiple data statements and display.
450-470	Inputs record from disk #1.
480-500	Prints record on disk #1.
510-550	Improper disk in drive #1 message; option to try again.
560	Opens a port to the printer if not already open.
570	Prints title page.
580	Prints option page and input option.
590	Checks input limits on option.
600	Branches to subroutines.
610-1120	Creates file subroutines.
610	Checks for proper disk in drive #1.
620	Initializes variables.
630	Branches to subroutines.
640-690	Inputs option for density (480 or 960).

Continued

Copyright © 1983 Emerald Valley Publishing Co.

EXPLANATION Continued

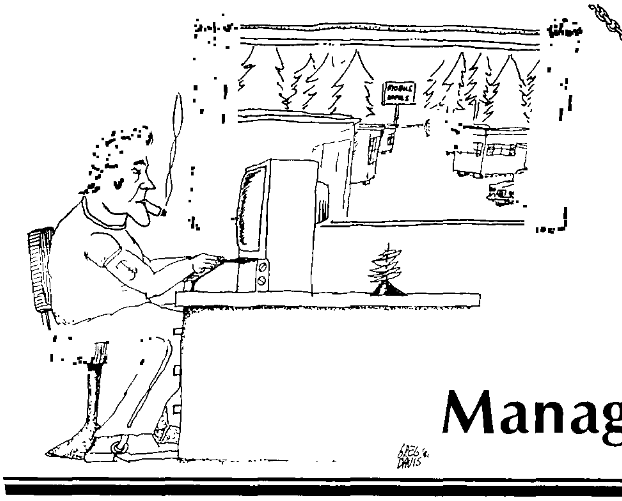
700	Clears screen. Inputs first record.
710-720	Displays record variables.
730	Inputs new value or command.
740-830	Checks for a command input and does necessary logic.
840-860	Checks that all characters in the new value are numeric.
870-920	Assigns new value to array; advances to next line position, and checks for end of line.
940-970	Subroutine to enter new line number and new position.
980-1030	Subroutine to print one line (CHR$(X)=240).
1040-1110	Subroutine to display array contents on the screen.
1110	Subroutine to convert the input string into ASCII form and store it in the array.
1120	Subroutine to re-convert the array into a string for output to the disk.
1130-1270	Subroutine to print entire graphics page from information stored on disk.
1130	Checks for file on disk.
1140	Inputs density (480, 960).
1150-1160	Inputs width of graphics in dot columns.
1170-1180	Inputs number of lines to be printed.
1190-1270	Prints file from disk onto the printer in the form of bit-image graphics.
1280-1340	Initializes a new disk with all CHR$(0); will destroy any records stored on that file.
1350	Closes files and ends.

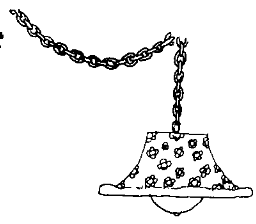
```
100 REM ****************
110 REM * DOTS TO PLOTS *
120 REM ****************
130 REM
140 REM
150 REM
160 REM
170 REM
180 OP2=0 :: OP3=0
190 DIM Z(12,20)
200 GOTO 570
210 DATA 1,8,DOTS TO PLOTS,3,9,SUBROUTINES
220 DATA 1,11,MENU,3,3,1.CREATE DATA FIELD,5,3,2.PRINT DATA FIELD,7,3,3.INITIALIZE NEW DISK,9,3,4.EXIT
230 DATA 1,5,CREATE DATA FIELD
240 DATA 1,9,PRINT DATA,3,3,FILE NAME?
250 DATA 23,3,YOUR CHOICE?
260 DATA 20,3,FILE NOT ON DISK
270 DATA 3,5,PLACE DISK IN DSK1,6,5,DISK IS NOT BLANK
280 DATA 24,3,PRESS ENTER TO CONTINUE
290 DATA 12,1,NEW LINE:,13,1,NEW POS.:,12,1,"",13,1,""
300 READ A1,A2,A$ :: DISPLAY AT(A1,A2):A$ :: RETURN
310 READ A1,A2,A$ :: DISPLAY AT(A1,A2):A$ :: ACCEPT AT(A1,A2+LEN(A$)+1):ANS$ :: RETURN
320 READ A1,A2,A$ :: RETURN
330 CALL SCREEN(5):: FOR A=1 TO 8 :: CALL COLOR(A,2,8):: NEXT A :: RETURN
340 OPEN #1:"DSK1.",RELATIVE,INTERNAL,INPUT
350 INPUT #1:A$,A,A,A
360 INPUT #1:A$,:: IF A$="" THEN CLOSE #1 :: GOTO 510
370 IF A$=ANS$ THEN CLOSE #1 :: GOTO 390 ELSE IF A$<>"" THEN 540
380 RESTORE 260 :: GOSUB 300 :: RETURN
390 IF OP2=1 THEN 400 :: OPEN #2:"DSK1."&ANS$,RELATIVE,INTERNAL,FIXED 24 :: OP2=1
400 RETURN
410 CALL COLOR(9,5,5):: CALL VCHAR(1,31,99,96):: CALL VCHAR(1,3,32,672):: RETURN
420 GOSUB 410 :: RESTORE 270 :: GOSUB 300 :: RESTORE 280 :: GOSUB 310
430 RESTORE 240 :: GOSUB 320 :: GOSUB 310 :: GOSUB 340 :: RETURN
440 FOR T1=1 TO T :: READ A1,A2,A$ :: DISPLAY AT(A1,A2):A$ :: NEXT T1 :: RETURN
450 FOR X=1 TO 20
460 INPUT #2,REC L*20+X:@$ :: GOSUB 1110
470 NEXT X :: RETURN
480 FOR X=1 TO 20
490 GOSUB 1120 :: PRINT #2,REC L*20+X:@$
500 NEXT X :: RETURN
510 IF PG=2 THEN GOSUB 390 :: RETURN ELSE DISPLAY AT(6,5):"DISK IS BLANK"
520 DISPLAY AT(7,1):"DO YOU WISH TO TRY AGAIN?      (Y/N)." :: ACCEPT AT(8,8)VALIDATE("YN"):AN$
530 IF AN$="Y" THEN 340 ELSE 1350
540 IF PG=1 THEN DISPLAY AT(6,1):"FILE NOT PRESENT" :: GOTO 520
550 DISPLAY AT(6,1):"DISK IS NOT BLANK" :: GOTO 520
560 IF OP3=1 THEN RETURN ELSE OPEN #3:"RS232.BA=9600.DA=8.PA=N.CRLF" :: OP3=1 :: RETURN
570 GOSUB 330 :: GOSUB 410 :: RESTORE 210 :: T=2 :: GOSUB 440 :: RESTORE 280 :: GOSUB 310
580 GOSUB 410 :: RESTORE 220 :: T=5 :: GOSUB 440 :: RESTORE 250 :: GOSUB 310
590 IF ASC(ANS$)<48 OR ASC(ANS$)>55 THEN RESTORE 250 :: GOSUB 310 :: GOTO 590
600 ON VAL(ANS$)GOTO 610,1130,1280,1350
610 PG=1 :: GOSUB 410 :: RESTORE 230 :: GOSUB 300 :: GOSUB 420
620 L=0 :: P=1 :: P1=1
630 GOSUB 390 :: GOSUB 640 :: GOTO 700
640 DISPLAY AT(2,1):"WHAT DENSITY PER LINE?"
650 DISPLAY AT(3,1):"1. 480":"2. 960"
660 ACCEPT AT(5,1)VALIDATE("12"):D$ :: D=VAL(D$)
670 IF D=1 THEN D$="K" ELSE D$="L"
680 D=D*2
690 RETURN
700 GOSUB 410 :: GOSUB 450
710 DISPLAY AT(1,1):"LINE #:":"":"POSITION:":"":"OLD VALUE:":"":"NEW VALUE:"
720 DISPLAY AT(1,9):L :: DISPLAY AT(3,11):(P1-1)*12+P :: DISPLAY AT(5,12):Z(P,P1)
730 ACCEPT AT(7,12):NV$
740 IF NV$<>"S" THEN 770 ELSE IF P>1 THEN P=P-1 :: GOTO 720
750 P1=P1-1 :: P=12 :: IF L=0 THEN 720 ELSE IF P1<1 THEN GOSUB 480 :: L=L-1 :: P1=20 :: GOSUB 450 :: GOTO 720
760 GOSUB 450 :: GOTO 720
770 IF NV$="D" THEN IF P<12 THEN P=P+1 :: GOTO 720 ELSE P1=P1+1 :: P=1 :: IF P1>20 THEN GOSUB 480 :: L=L+1 :: P1=1 :: GOSUB 450 :: GOTO 720
780 IF NV$="E" AND L>0 THEN GOSUB 480 :: L=L-1 :: GOSUB 450 :: P1=1 :: P=1 :: GOTO 720
```


Copyright © 1983 Emerald Valley Publishing Co.

```
790 IF NV$="X" AND L<100 THEN GOSUB 480 :: L=L+1 :: GOSUB 450 :: P1=1 :: P=1 :: GOTO 720
800 IF NV$="P" THEN GOSUB 560 :: GOSUB 980 :: GOTO 720
810 IF NV$="L" THEN GOSUB 1040 :: GOTO 710
820 IF NV$="Z" THEN 580
830 IF NV$="N" THEN GOSUB 940 :: GOSUB 300 :: GOSUB 300 :: GOTO 710
840 FOR TL=1 TO LEN(NV$)
850 IF ASC(SEG$(NV$,TL,1))<48 OR ASC(SEG$(NV$,TL,1))>57 THEN GOTO 720
860 NEXT TL
870 IF NV$="" THEN NV$="0"
880 NV=VAL(NV$):: IF NV<0 OR NV>255 THEN 720
890 Z(P,P1)=NV
900 P=P+1
910 IF P>12 THEN P1=P1+1 :: P=1 :: IF P1>20 AND L<100 THEN GOSUB 480 :: L=L+1 :: GOSUB 450 :: P1=1 :: P=1 :: GOTO 720
920 GOTO 720
930 REM
940 RESTORE 290 :: GOSUB 310 :: IF ANS$="" THEN NLINE=L :: GOTO 950 ELSE IF VAL(ANS$)<=189/D THEN NLINE=VAL(ANS$)ELSE GOTO 940
950 GOSUB 310 :: IF ANS$="" THEN LPOS=(P1-1)*12+P ELSE IF VAL(ANS$)<=240 THEN LPOS=VAL(ANS$)ELSE GOTO 950
960 IF NLINE>189/D OR NLINE<0 OR LPOS>240 OR LPOS<1 THEN 940
970 GOSUB 480 :: L=NLINE :: GOSUB 450 :: P1=INT((LPOS+1)/12):: P=((LPOS+1)/12-(P1*20))*12 :: RETURN
980 PRINT #3:CHR$(27);D$;CHR$(240);CHR$(0);
990 FOR X=1 TO 20
1000 PRINT #3:CHR$(Z(1,X));CHR$(Z(2,X));CHR$(Z(3,X));CHR$(Z(4,X));CHR$(Z(5,X));CHR$(Z(6,X));CHR$(Z(7,X));
1010 PRINT #3:CHR$(Z(8,X));CHR$(Z(9,X));CHR$(Z(10,X));CHR$(Z(11,X));CHR$(Z(12,X));
1020 NEXT X
1030 CLOSE #3 :: OP3=0 :: RETURN
1040 TB=1 :: LN=1 :: LI=9 :: GOSUB 1060 :: LN=10 :: LI=18 :: GOSUB 1060 :: LN=19 :: LI=20 :: GOSUB 1060
1050 GOSUB 410 :: RETURN
1060 FOR Y=LN TO LI
1070 FOR X=1 TO 12 :: PRINT TAB(TB*5-4);Z(X,Y);:: TB=TB+1 :: IF TB=6 THEN TB=1
1080 NEXT X :: NEXT Y :: PRINT :: PRINT
1090 RESTORE 280 :: GOSUB 310
1100 RETURN
1110 FOR @=1 TO 12 :: Z(@,X)=ASC(SEG$(@$,@,1)):: NEXT @ :: RETURN
1120 @$="" :: FOR @=1 TO 12 :: @$=@$&CHR$(Z(@,X)):: NEXT @ :: RETURN
1130 GOSUB 410 :: RESTORE 240 :: GOSUB 300 :: GOSUB 310 :: GOSUB 390
1140 GOSUB 640
1150 DISPLAY AT(7,1):"WIDTH OF GRAPHICS":"":"1. 240":"2. 480":"3. 720 (960 RES. ONLY)":"4. 960 (960 RES. ONLY)"
1160 ACCEPT AT(12,1)VALIDATE("1234"):WIDTH$ :: W=VAL(WIDTH$)
1170 DISPLAY AT(14,1):"NUMBER OF LINES? (1-";INT(189/W);")"
1180 ACCEPT AT(15,1)VALIDATE(DIGIT):NOL :: IF NOL>INT(189/W)THEN 1180
1190 DISPLAY AT(18,10):"PRINTING"
1200 GOSUB 560 :: PRINT #3:CHR$(27);"A";CHR$(8);
1210 FOR PRNT=0 TO NOL-1
1220 FOR PRIT=1 TO W
1230 PRINT #3:CHR$(27);D$;CHR$(240);CHR$(0);
1240 L=PRNT*W+PRIT-1 :: FOR X=1 TO 20 :: INPUT #2,REC L*20+X:@$ :: PRINT #3:@$;:: NEXT X
1250 NEXT PRIT :: PRINT #3:""
1260 NEXT PRNT
1270 CLOSE #3 :: OP3=0 :: GOTO 580
1280 GOSUB 410 :: DISPLAY AT(1,1):"INITIALIZATION WILL DESTROY":"ANY RECORDS BEING STORED":"ON THE FILE"
1290 DISPLAY AT(4,1):"DO YOU WISH TO CONTINUE?":"(Y/N)"
1300 ACCEPT AT(5,7)VALIDATE("YN"):AN$ :: IF AN$="N" THEN 570
1310 PG=2 :: GOSUB 410 :: DISPLAY AT(2,1):"PLACE BLANK DISK IN DRIVE #1" :: RESTORE 280 :: GOSUB 310
1320 GOSUB 430
1330 FOR X=1 TO 12 :: @$=@$&CHR$(0):: NEXT X
1340 FOR X=1 TO 3570 :: PRINT #2:@$ :: NEXT X :: CLOSE #2 :: GOTO 570
1350 IF OP2=1 THEN CLOSE #2
1360 IF OP3=1 THEN CLOSE #3
1370 END
```

Copyright © 1983 Emerald Valley Publishing Co.





Personal Record Keeping

Managing a Mobile Home Park

First of all, this true story has a moral to it, so we might as well get it out of the way now:

"Before going to all the work of writing a program to do a job, find out if a TI Command Cartridge can do the job for you."

The TI Command Cartridges are well written, almost totally error-free, and have been engineered for ease of use by non-programmers. Let's talk about one of these little jewels:

The *Personal Record Keeping (PRK)* Command Cartridge, when combined with your imagination, is a very flexible and powerful tool. In order to fully utilize this power, however, your TI-99/4A system should include a printer. The TI Thermal Printer works well and is probably the easiest and least expensive to use, but I chose a more expensive route: an Epson MX-80 printer operating through the RS232 Interface. This gives me a bit more power (e.g., longer print lines) for the *PRK*'s report formating. For most applications, you will also need either a cassette recorder or disk system to store your data files.

Before trying to set up and work with a data file using the *PRK*, you should carefully read the manual and all the examples that come with the cartridge. When you have done that, take a break, come back a little later, and do it again. That mild-mannered little *PRK* manual contains the answers to questions that will surely pop up when you start designing the solution to *your* problem. So keep it handy!

OK, now comes the real challenge. How do we decide that a problem can be solved using the *PRK* cartridge? First, we must completely describe the problem. Second, we must break the problem down into subproblems or tasks. Third, we identify the tasks that can be performed by the *PRK* cartridge. Fourth, we see if enough of the tasks can be handled by the *PRK* cartridge to make its use a reasonable solution to the overall problem. The rest of this article is about such a challenge and a way to solve it with the *PRK* cartridge.

And Now For Our Story . . .

Recently I had a customer ask me how much I would charge to write a program for him. I told him that it is difficult to determine until the program is completed, but he could figure on $15.00 per hour for a minimum of 10 hours. At that point, he decided to be brave and tackle the program himself. Of course, I was curious, so I asked him what he wanted the TI BASIC program to do.

He told me that he was the owner of a mobile home park. Each month he had to figure out the bill for each individual renter in the park. He wanted the TI-99/4A to save him time and decrease the chance for errors. After thinking over this problem for a minute, I asked him for details: What did he do to accomplish the job himself?

First, he walked around to each trailer space and copied the electric meter and gas meter readings into his notebook. A computer system could be designed to do this task but it would take extremely expensive peripheral hardware. . .

When I asked him what else was in the notebook that he used for this job, he said that it contained all the previous electric and gas meter readings. It also contained miscellaneous charges for each renter, the electric and gas rates, and the actual space rental fees. At this point it was obvious to me that the computer could easily act as a notebook and store all that data on cassette tape or floppy disk.

Next, he sat down at his desk with the notebook, pencil, paper and a calculator. For each trailer space, he performed the following calculations:

```
     GAS BILL = (CUR. GAS METER - PREV. GAS METER) x GAS RATE
                + GAS METER USE FEE
ELECTRIC BILL = (CUR. KWH METER - PREV. KWH METER) x KWH RATE
                + KWH METER USE FEE
   TOTAL BILL = GAS BILL + ELECTRIC BILL + MISC. CHARGES
                + SPACE RENTAL FEE
```

He recorded each of the items in the notebook for bookkeeping purposes, and then made out a statement for each tenant. Finally, he figured the total gas bill, the total electric bill, and the total income for the trailer park. Of course, the computer could also record all the bill items, perform the calculations, and make out statements.

After reading the *PRK* manual a couple of times, I noted it offered the following capabilities:

1. Maintains files of data in a structured fashion.
2. Allows data additions and updates within the files.
3. Permits mathematical operations on any numerical data structure or between numerical data structures.

Copyright © 1983 Emerald Valley Publishing Co

4. Permits all data structures to be sorted in various ways.
5. Permits printing of data structures as reports, lists or what have you.

After further consideration, I decided that most of the tasks related to the "trailer park monthly billing problem" could be solved using the TI-99/4A with the *PRK* cartridge and a printer.

With my TI-99/4A fired up, *PRK* cartridge installed, and manual in hand, I started toying around, setting up the data structure of the file to use on this problem. I finally settled on the structure shown in Table 1.

FILE STRUCTURE

ITEM	TYPE	WIDTH	DEC	DESCRIPTION OF ITEM
1 SPACE #	CHAR	4	0	The trailer space number
2 RENTER	CHAR	15	0	Name of the trailer space renter
3 LAST GAS	DEC	10	3	The previous gas meter reading
4 CUR. GAS	DEC	10	3	The current gas meter reading
5 GAS RATE	DEC	6	4	The cost of the gas per unit volume
6 LAST KWH	DEC	10	3	The previous electric meter reading
7 CUR. KWH	DEC	10	3	The current electric meter reading
8 RATE/KWH	DEC	6	4	The cost of the electricity per KWH
9 G.M. CHRG	DEC	5	2	The monthly gas meter charge
10 GAS TOTAL	DEC	6	2	The cost of gas used plus the meter charge
11 E.M. CHRG	DEC	6	2	The monthly electric meter charge
12 ELEC.TOTL	DEC	6	2	The cost of KWH used plus the meter charge
13 RENT/MO.	DEC	6	2	The trailer space monthly rental cost
14 MISC.CHRG	DEC	7	2	Any other charge (maybe damage to lot...)
15 MO. TOTAL	DEC	8	2	Grand total of gas, KWH, rent, and misc.

TABLE 1

Once a structure has been defined, you can't go back and change it without redefining the entire file structure. In order to minimize this problem, the best policy to follow is to try out the file structure with a small amount of test data. It is a real pain to spend 4 hours entering real data into a file and then discover that one oddball piece of data is too big! By the way, the smaller you define the width of a data item, the more data items you can keep in memory. As you can see, some care must be given to the design of the file structure.

Look at Table 2. It shows my three sample file "pages" of test data. This is the way the data would look after putting in the initial values. Now look at Table 3. The current utility meter readings and any miscellaneous charges have now been entered as the trailer park operator would do once a month.

At this point, I realized I had to figure out how to use the *PRK* cartridge's *math transformations*. That sounds pretty ominous, doesn't it? But study of the manual revealed that it is nothing more than a set of simple equation *templates*. These are shown on page 25 of the *PRK* manual and included here in Table 4. By substituting an item name for the appropriate A, B, or C in the equations, I built up a set of math transformations to figure out the electric, gas, and total bills. The *PRK* cartridge guides you through this process nicely. The tailored set of math transformations is shown in Table 5 (in the order of execution).

Notice that the tailored math transformations set up the next month's LAST GAS, CUR. GAS, LAST KWH, CUR. KWH item fields after the current data was used. This means that next month the user won't have to worry about moving the old "current" values to the "last" fields for the next month too. (That ought to get your imagination working!)

Now for the big test: Run the tailored math transformations on the file of test data and see if it works. The results are shown in Table 6. It is interesting to compare Table 6 to Table 3. The comparison better illustrates the work of these tailored equation templates.

With all the real data in the file, it takes about half an hour to a full hour to process all the math. Sure, that is slow, but it is *accurate*—and the manager can be eating dinner while the *PRK* cartridge processes the data. After dinner, he can start the *PRK* cartridge printing out a report for each *file page*, as shown in Table 6. Finally, after a nice relaxed dessert or brandy, he can cut apart the pages of the report and tape them in the appropriate spot of the form shown in Figure 1. There is a separate form page for each space in the trailer park. By using tape only at the *top* of the little *PRK* page, he can flip through previous month's data (since the little pages are overlapping).

An Automatic *Manual* Feature

By using the ANALYZE PAGES mode of the *PRK* cartridge, you can read the total gas, electric, and monthly income. After selecting the mode, select 5 SEE ITEM

FILE: RENTALS
DATE: 6/20/81
TITLE: TABLE 2

PAGE #	1	PAGE #	2	PAGE #	3
1.SPACE #	A-23	1.SPACE #	B-44	1.SPACE #	B-45
2.RENTER	SMITH,C.W.	2.RENTER	JONES,SAM	2.RENTER	HEIM,WILLIAM
3.LAST GAS	799992.465	3.LAST GAS	830.592	3.LAST GAS	990498.328
4.CUR. GAS	0.000	4.CUR. GAS	0.000	4.CUR. GAS	0.000
5.GAS RATE	.1130	5.GAS RATE	.1130	5.GAS RATE	.1130
6.LAST KWH	128176.263	6.LAST KWH	18841.212	6.LAST KWH	130392.249
7.CUR. KWH	0.000	7.CUR. KWH	0.000	7.CUR. KWH	0.000
8.RATE/KWH	.0231	8.RATE/KWH	.0231	8.RATE/KWH	.0231
9.G.M. CHRG	2.50	9.G.M. CHRG	2.50	9.G.M. CHRG	2.50
10.GAS TOTAL	0.00	10.GAS TOTAL	0.00	10.GAS TOTAL	0.00
11.E.M. CHRG	5.00	11.E.M. CHRG	5.00	11.E.M. CHRG	5.00
12.ELEC.TOTL	0.00	12.ELEC.TOTL	0.00	12.ELEC.TOTL	0.00
13.RENT/MO.	98.00	13.RENT/MO.	105.00	13.RENT/MO.	105.00
14.MISC.CHRG	0.00	14.MISC.CHRG	0.00	14.MISC.CHRG	0.00
15.MO. TOTAL	0.00	15.MO. TOTAL	0.00	15.MO. TOTAL	0.00

TABLE 2

Copyright © 1983 Emerald Valley Publishing Co.

FILE: RENTALS
DATE: 6/20/81
TITLE: TABLE 3

PAGE #	1	PAGE #	2	PAGE #	3
1.SPACE #	A-23	1.SPACE #	B-44	1.SPACE #	B-45
2.RENTER	SMITH,C.W.	2.RENTER	JONES,SAM	2.RENTER	HEIM,WILLIAM
3.LAST GAS	799992.465	3.LAST GAS	830.592	3.LAST GAS	990498.328
4.CUR. GAS	800124.732	4.CUR. GAS	891.947	4.CUR. GAS	990674.998
5.GAS RATE	.1130	5.GAS RATE	.1130	5.GAS RATE	.1130
6.LAST KWH	128176.263	6.LAST KWH	18841.212	6.LAST KWH	130392.249
7.CUR. KWH	131002.097	7.CUR. KWH	23622.609	7.CUR. KWH	134305.045
8.RATE/KWH	.0231	8.RATE/KWH	.0231	8.RATE/KWH	.0231
9.G.M. CHRG	2.50	9.G.M. CHRG	2.50	9.G.M. CHRG	2.50
10.GAS TOTAL	0.00	10.GAS TOTAL	0.00	10.GAS TOTAL	0.00
11.E.M. CHRG	5.00	11.E.M. CHRG	5.00	11.E.M. CHRG	5.00
12.ELEC.TOTL	0.00	12.ELEC.TOTL	0.00	12.ELEC.TOTL	0.00
13.RENT/MO.	98.00	13.RENT/MO.	105.00	13.RENT/MO.	105.00
14.MISC.CHRG	0.00	14.MISC.CHRG	17.50	14.MISC.CHRG	0.00
15.MO. TOTAL	0.00	15.MO. TOTAL	0.00	15.MO. TOTAL	0.00

TABLE 3

ITEM TRANSFORMATIONS

1. A = B
2. A = B + C
3. A = B − C
4. A = B × C
5. A = B / C
6. A = B^C
7. A = ABS(B)
8. A = LOG10(B)
9. A = LOGE(B)
10. A = EXP(B)
11. A = ATAN(B)
12. A = TAN(B)
13. A = SIN(B)
14. A = COS(B)
15. A = INT(B)
16. A = SGN(B)
17. A = PI
18. A = RND

(See the *User's Reference Guide* for a discussion of these functions.)

TABLE 4

TAILORED MATH TRANSFORMATIONS FOR TRAILER PARK BILLING

```
LAST GAS  =  CUR. GAS  -  LAST GAS
GAS TOTAL  =  LAST GAS  X  GAS RATE
GAS TOTAL  =  G.M. CHRG  +  GAS TOTAL
LAST GAS  =  CUR. GAS
CUR. GAS  =  0.000
LAST KWH  =  CUR. KWH  -  LAST KWH
ELEC.TOTL  =  LAST KWH  X  RATE/KWH
ELEC.TOTL  =  E.M. CHRG  +  ELEC.TOTL
LAST KWH  =  CUR. KWH
CUR. KWH  =  0.000
MO. TOTAL  =  GAS TOTAL  +  ELEC.TOTL
MO. TOTAL  =  MO. TOTAL  +  RENT/MO.
MO. TOTAL  =  MO. TOTAL  +  MISC.CHRG
```

TABLE 5

FILE: RENTALS
DATE: 6/20/81
TITLE: TABLE 6

PAGE #	1	PAGE #	2	PAGE #	3
1.SPACE #	A-23	1.SPACE #	B-44	1.SPACE #	B-45
2.RENTER	SMITH,C.W.	2.RENTER	JONES,SAM	2.RENTER	HEIM,WILLIAM
3.LAST GAS	800124.732	3.LAST GAS	891.947	3.LAST GAS	990674.998
4.CUR. GAS	0.000	4.CUR. GAS	0.000	4.CUR. GAS	0.000
5.GAS RATE	.1130	5.GAS RATE	.1130	5.GAS RATE	.1130
6.LAST KWH	131002.097	6.LAST KWH	23622.609	6.LAST KWH	134305.045
7.CUR. KWH	0.000	7.CUR. KWH	0.000	7.CUR. KWH	0.000
8.RATE/KWH	.0231	8.RATE/KWH	.0231	8.RATE/KWH	.0231
9.G.M. CHRG	2.50	9.G.M. CHRG	2.50	9.G.M. CHRG	2.50
10.GAS TOTAL	17.45	10.GAS TOTAL	9.43	10.GAS TOTAL	22.46
11.E.M. CHRG	5.00	11.E.M. CHRG	5.00	11.E.M. CHRG	5.00
12.ELEC.TOTL	70.28	12.ELEC.TOTL	115.45	12.ELEC.TOTL	95.39
13.RENT/MO.	98.00	13.RENT/MO.	105.00	13.RENT/MO.	105.00
14.MISC.CHRG	0.00	14.MISC.CHRG	17.50	14.MISC.CHRG	0.00
15.MO. TOTAL	185.72	15.MO. TOTAL	247.38	15.MO. TOTAL	222.85

TABLE 6

Copyright © 1983 Emerald Valley Publishing Co.

STATISTICS. Then choose an item—such as GAS TOTAL—and a display like Figure 2 will appear. The gas total for the entire trailer park is contained in the value of SUM. See what reading the manual reveals to you.

Before getting out of the *PRK* cartridge, you must save the data file on cassette tape or floppy disk for next time. Yes, the math transformations are also saved automatically at the same time.

Well, that's the story. I guess the only thing to add is that the *Personal Record Keeping* Command Cartridge isn't the solution to *all* problems. But if you study it and experiment enough, you will be ready to wield this valuable and flexible tool when the appropriate situation arises. So go ahead—give the cartridge a try. I'll bet that soon you too will be witnessing a "Command" performance.

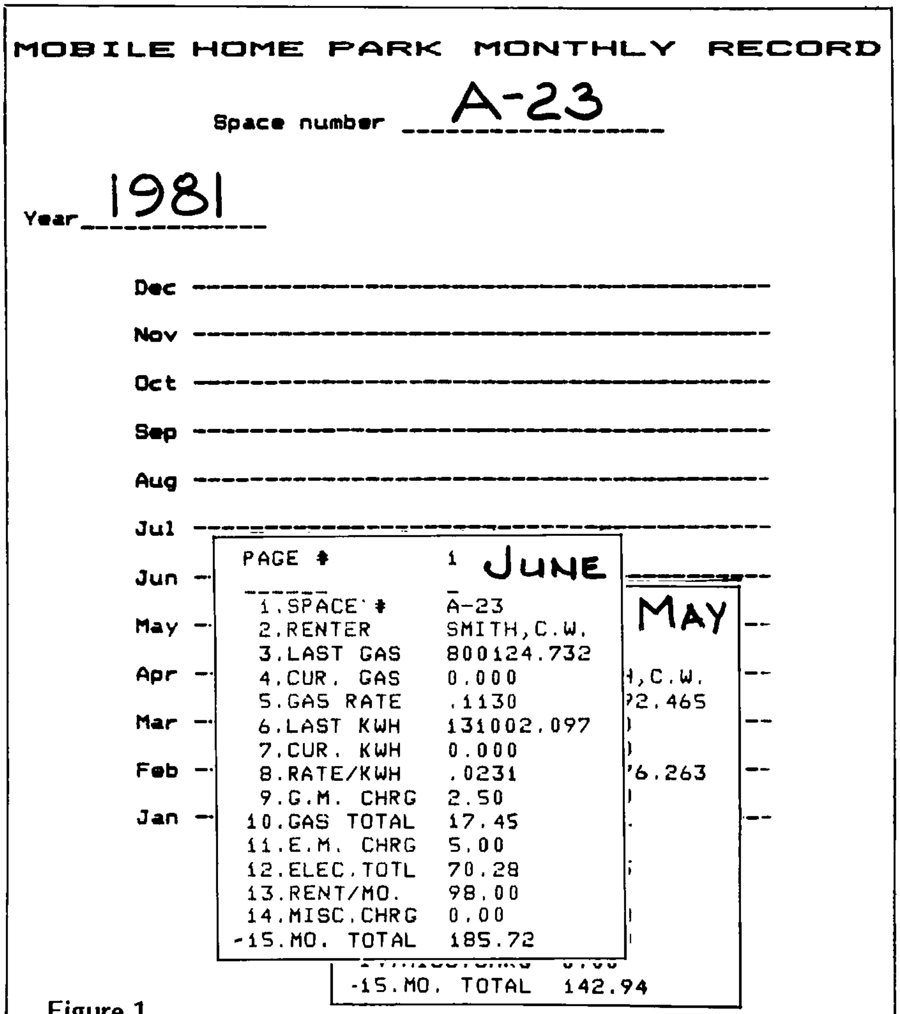


Figure 1.

```
ITEM STATISTICS

    ITEM = GAS TOTAL
    MEAN = 16.44666667
 STD DEV = 6.57268844
 MAX VAL = 22.46
 MIN VAL = 9.43
     SUM = 49.34
    DATA = 3
 MISSING = 0
```

Figure 2.

Copyright © 1983 Emerald Valley Publishing Co.

The Small Investor & the **TI-99/4A**

A LOOK AT THE DOW JONES NEWS SERVICE

Information utilities such as The Source and MicroNet allow any individual with a microcomputer and modem to tap into a rich vein of information resources. These databases, however, are aimed almost exclusively toward the general consumer population and as such cannot adequately cover the needs of serious, small investors. That's where the Dow Jones News Service (DJNS) comes in: The combination of the DJNS and the TI-99/4A may be the most significant advance in investment analysis since the electronic calculator made its debut. . .

In addition to giving you historical stock quotes, DJNS gives you current-day quotes for all listed stocks, bonds, options and U.S. Treasury issues. The DJNS also has some specialized databases which you can access for information about particular companies, market sectors or market indicators.

For a comprehensive review of a stock or industry, the Media General database provides detailed technical and fundamental indicators on the item of your choice.

The conservative investor can access the Disclosure On-line database for a profile on most major companies, plus a 10-K report that lists almost all the important (to the investor) information that can be found in a corporation's financial statement.

The Money Market Service database is a new service introduced by Dow Jones in February 1981. Commentary, tables and graphs on the economy are displayed for most of the important indicators used in determining the current business climate. Of course, the ever-popular Dow Jones averages are also available, as are Trading Activity, The Market Diary, Market Volume, and many other valuable market statistics.

With everything there comes a price tag, and the news service is no exception. During the business day (6:00 a.m. to 7:00 P.M. EST) the charge for news is $1.20 per minute. After 7:00, this rate is reduced drastically! Until the next morning, news can be accessed for 20 cents per minute, and historical market quotes for 15 cents. The start-up fee for the service is $50, but there are no monthly charges or minimum on-line times. For high-volume users there is pricing option A. Under this option, there is a $75 monthly fee in exchange for lower prime-time rates during the business day. Pricing option B should be satisfactory for most individual investors.

[To access the Dow Jones New Sevice and its databases you will need the TI *Terminal Emulator II* Command Cartridge to send and receive the appropriate signals, as well as the TI RS232 Interface and an RS232C-compatible telephone coupler (or modem).—Ed.]

After news has been obtained on the News Service, there are really only two things that can be done with it: (1) it can be kept temporarily, or (2) kept permanently. News that is to be kept temporarily is best stored on a disk or printed copy for ease of access and readability. When keeping news permanently, cassette tapes can be both cost effective and reasonably efficient, especially if bought in volume.

For aspects of the service other than news, there are many different ways to use both the historical and current quote databases. The historical quotes are available in either monthly or quarterly format for any given item. While a weekly format would be desirable, the monthly quotes can be used to determine most long- and intermediate-term trends. For the very short-term, one month of daily quotes is always available. These can be used to develop a 10-, 15- or 20-day moving average of prices for the item being researched, and if saved over a period of time, can be used in any format.

For the novice investor, the Media General database provides a sufficient amount of both technical and fundamental analysis. *Fundamental analysis* refers to information concerning aspects of a particular company or industry, such as assets, net worth, or earnings. *Technical analysis* refers to the study of the chart or graph of a company, industry, or the market in general—in the hope that past behavior as revealed in graphs can be used to predict future price movements.

The serious investor may prefer to develop his or her own analytical tools. One current theory on Wall Street

Copyright © 1983 Emerald Valley Publishing Co.

today maintains that about half of a stock's performance is due to movement of the market in general, and about half of the movement is due to characteristics peculiar to that particular stock. Naturally, anyone who can predict the movement of the market, even for a short time, has a very powerful financial tool.

For this reason, my own predilection is for analyzing the leading market indices. This analysis can be facilitated by the TI *Personal Record Keeping* Command Cartridge (*PRK*). Each page you set up with the *PRK* can represent one day, and the first few lines can label the index to be tracked. The remaining lines can be the 10-, 15-, or 20-day averages of the aforementioned indices. The use of math transformations in the *PRK* cartridge allows you to compute the average for each of the indices, but you must enter the average manually with the Change Page option. The average has a useful by-product which the *PRK* computes automatically: the standard deviation. I have found this statistic to be a good indicator of market volatility. It too can be entered and tracked with the average. The ability of the *Statistics* Command Cartridge to analyze data produced with the *PRK* cartridge is a definite plus. Even though the *Statistics* cartridge is a more sophisticated analytical device, and offers more tools to work with than the *PRK* cartridge, I do not feel that it is essential to index analysis—only helpful.

Investors with access to a TI-59 programmable calculator as well as a TI-99/4A can perform some rather astounding mathematical computations without a strong math background. Quotes obtained through the News Service can be processed in a *Least Squares Curve Fit* program detailed in a Texas Instruments publication, *Sourcebook for Programmable Calculators*. This will result in a series of simultaneous equations which can be solved with either the *Master Library-2* program on the TI-59 or the *Math Library-2* program on the TI-99/4A. In theory, the resulting equation should be a reasonably accurate description of the line from which the datapoints were taken, and it can be used to predict the future behavior of the line. Naturally, the number and quality of the datapoints chosen determine the accuracy of the predictive equation, and any conclusion drawn from such analysis is at best highly speculative.

Fundamental analysis using the TI-99/4A also has many applications. You can program balance sheet and income statement analyses, and then compare them to an "ideal" or average analysis in order to determine the variances which may reveal the strengths or weaknesses of a particular company or industry. The information for these analyses can be found in the 10-K section of the Disclosure On-Line database of the News Service.

Of course, these are only a few of the applications that are possible with the TI-99/4A and the Dow Jones News Service. In the past, this mathematical analysis of the market and its component stocks was inaccessible or simply incomprehensible to the small investor. But now, with the help of your TI-99/4A, it's both possible and easy to take a sophisticated approach to market analyses.

I would recommend that any investor with a TI-99/4A computer call Dow Jones on their toll-free number (800-257-5114 except N.J.) to request their free information packet detailing prices and services.

Good luck, 99'ers! If this works for you, your only problem may be writing a suitable income tax program!

Copyright © 1983 Emerald Valley Publishing Co.

TI BASIC

Interactive Forms Generator

When I started in business, I decided to utilize my TI-99/4A as much as possible. One of the things I wanted to do with the computer was to generate customized business forms: purchase orders, price lists, invoices, and sales orders.

Right away you may be thinking: "He could buy all those forms ready made. . ." Yes, but that's not challenging or really as much fun. Not only that, but printing up custom forms (ones that bear your company name and address) is not cheap. Around here a minimum order of triplicate invoices costs about $40 for 500. (And I probably wouldn't use all 500 before wanting to modify the form anyway. . .). Furthermore, multiplying that $40 figure by the 12 *different* forms (including price list pages) I presently use gives a starting cost of $480! That is almost enough money to buy an Epson MX-80 printer!

Well, you guessed it: I bought the printer—plus the serial interface, the RS232 cable, and the TI RS232 interface. The whole setup did cost more than the original estimate, but I can write off the added cost as "hobby money" for now. With the right software I could sit down at the TI-99/4A keyboard, activate a program that would prompt me to fill in the blanks of a form that was in memory, and finally print out as many copies as I wanted on the MX-80.

I wrote such a program and I called it the *Interactive Forms Generator*. It is written in a general fashion to work with any correctly formated data file. I then made up a Form Data File for each of my forms. A Form Data File is just a bunch of ASCII text lines stored in a string array. Each text line may be written as a DATA LINE to be printed on the MX-80 or as a COMMAND LINE to direct the *Interactive Forms Generator* program.

How Does It Work?

The *Interactive Forms Generator* (*IFG*) program asks questions of the operator via the TI-99/4A screen. *IFG* accepts inputs from the operator via the keyboard and interprets instructions from the Form Data File's COMMAND LINES. In other words, the *IFG* program works with you to load your Form Data File, fill out the form, and finally print it out on the MX-80.

Let's say I am generating a Sales Order Acknowledgment form to send to a customer. First, I load the *IFG* program for diskette (or casette). Second, I type RUN and hit ENTER. Third, the *IFG* program asks:

```
MAKE A CHOICE--
1. LOAD NEW FORM FILE
2. FILL OUT SAME FORM
3. PRINT COPIES
4. TERMINATE
```

I enter 1 and follow instructions from the *IFG* program to load the Sales Order Acknowledgment Form Data File. Fourth, the program asks:

```
MAKE A CHOICE--
1. LOAD NEW FORM FILE
2. FILL OUT SAME FORM
3. PRINT COPIES
4. TERMINATE
?
```

I enter 2. Fifth, *IFG* will look through the Form Data File for the COMMAND LINES. Interpreting the lines, *IFG* will prompt me via the screen for the information needed to fill out the form's blanks. Also, in interpreting the COMMAND LINES, *IFG* may perform simple math functions on fields of DATA LINES to calculate tax, totals, etc. After all the COMMAND LINES have been used, *IFG* again asks:

```
MAKE A CHOICE--
1. LOAD NEW FORM FILE
2. FILL OUT SAME FORM
3. PRINT COPIES
4. TERMINATE
?
```

This time I enter 3 and the *IFG* program asks:

```
ENTER NUMBER OF COPIES TO PRINT-
```

I enter some number and *IFG* sends only the DATA LINES of the Form Data File to the MX-80, which does the rest! See Figure 2 for a look at the completed form sample.

Copyright © 1983 Emerald Valley Publishing Co.

Boy, isn't that slick. . .just like the big guys—perhaps a little slower, but that's OK until the business grows to the point that speed is important. (By the way, for Christmas I can generate a very long form letter with a year's worth of family news, then use *IFG* to fill out a separate salutation for each relative. So the whole family gets the latest without my getting writer's cramp! I'll bet that with your imagination and creativity you will come up with some other neat applications for *IFG*, too. . .)

OK, OK. You want to know how you can make one of these Form Data Files, don't you? Well then, there are a couple ways:

Building a Form Data File: Method 1

If you have some kind of editor program that will build an ASCII text string array, you are all set. All you have to do is make sure it will output the special ASCII control codes used by the printer to do its tricks. It must also output the Form Data File to cassette or diskette in a compatible format. Listings 1 and 2 for subroutines CASSOUT and DISKOUT illustrate what is needed. If you don't have an editor program, see Method 2, below:

Building a Form Data File: Method 2

This is a real simple—but much more tedious—method of building the Form Data File.

STEP 1.

Sit down with pad of paper and a pencil. Now design each character-string line of the form. Use the CHR$ ( ) function to put in the string special codes that can't be directly entered by a key on the 99/4A keyboard. The codes can be looked up in the MX-80 (or other printer's) manual. The samples shown below are: CHR$(27), ESCape code; CHR$(13), Carriage Return code; CHR$(10), Line Feed code:

CHR$(27)&"E"&"THE DOG RAN HOME QUICKLY"&CHR$(10)&CHR$(13)

STEP 2

Now fire up your 99/4A. Enter the following program lines:

```
100 REM
110 REM  "FILEBUILD" PROGRAM
120 REM
130 OPTION BASE 1
140 DIM A$(70)
150 REM
```

Then enter your character-string lines from paper into the string array via the TI-99/4A keyboard as follows:

```
160 REM  NOW FOR THE CHARACTER STRINGS
     IN THE ARRAY
170 REM
180 A$(1)=CHR$(27)&"E"&"THE DOG RAN HO
    ME QUICKLY"&CHR$(10)&CHR$(13)
190 A$(2)="AFTER DINNER THE DOG BURPED
     AND SCRATCHED HIS EAR."&CHR$(10)&
    CHR$(13)
200 A$(3)="******"
??? A$(??)="THAT'S ALL FOLKS!
    "&CHR$(10)&CHR$(10)&CHR$(13)
1000 REM
```

Now enter the following lines of program code:

```
1010 REM  MAKE X EQUAL TO THE NUMBER OF
      LINES IN A$
1030 X=??
1040 PRINT "WRITE FILE TO?"
1050 PRINT "     1  - CS1"
1060 PRINT "     2  - DSK"
1070 INPUT CHOICE
1080 IF (CHOICE<1)+(CHOICE>2)=-1 THEN 1
     040
1090 ON CHOICE GOSUB 2000,3000
1100 PRINT " C O M P L E T E"
1110 PRINT "NOW SAVE ME ON TAPE OR DISK
     .........."
1120 END
1130 REM
1140 REM OUTPUT SUBROUTINES FOLLOW.....
1150 REM
```

Finally, enter the two subroutines CASSOUT and DISKOUT starting at lines 2000 and 3000.

STEP 3.

Type in RUN. You should end up with your own Form Data File on tape or diskette. This can now be used with the *IFG* program,

STEP 4.

Hold it! Don't turn off the TI-99/4A yet! SAVE your *Filebuild* program on tape or diskette too. Chances are you will want to modify that form because of errors or change of design in the future. OK, now you can turn off the computer and hit the sack. (Notice that this kind of work is always done at midnight. . .)

To help clarify the above process, I generated a simple *Filebuild* program (Listing 3). Note that text lines A$(1) – A$(13) are DATA LINES and text lines A$(14) – A$(19) are COMMAND LINES (more on these next). Data File 2 shows the resulting Form Data File (as printed by my editor program). Figure 1 shows the results of running *IFG* using this Form Data File.

```
KOMPUTAR WORKS
P.O.BOX 483
143 SUNSET DRIVE
ELECTRIC CITY, WASHINGTON
     99123



                    SHIP TO:
                               MR. CHIP BUGGS
                               9900 SEAMOSS AVE.    APT. # 102
                               ELECTRIC CITY
                               WASHINGTON
                                    99123
```

Figure 1.

Power to the *IFG*!

How do we get the Form Data File to tell the *IFG* what to do? By making up COMMAND LINES. What makes a COMMAND LINE special? It must start with these two characters: !!. What can a COMMAND LINE tell *IFG* to do? It can tell it to output a message to the TI-99/4A display. How? Here's a sample:

!!"THIS MESSAGE WILL BE WRITTEN ON THE 99/4A DISPLAY"

Note that anything between quotes will be displayed.

What about telling it to get something from the 99/4A keyboard? OK—whenever *IFG* does this, it stuffs the information obtained into a line of the Form Data File either right-justified or left-justified. To get input from the keyboard and stuff it left-justified, use this FIELD DEFINITION syntax:

Copyright © 1983 Emerald Valley Publishing Co.

!! :28:1:32: (!!: *:line#:first character:last character:*)

To get information right-justified (which is needed for lining up decimals) do the same except add a ">" sign after the first ":". Example:

!! :>28:1:32:

By the way, *IFG* will show you on the display how much space you have to write in and will let you know if you overflow.

Didn't I say something about *IFG* doing math calculations, you ask? Right. You can write COMMAND LINES to add, subtract, multiply, and divide. Each term and operator simply must be enclosed in parentheses. A term may be a FIELD DEFINITION or a constant. Here are some samples:

```
!! (:28:1:32:) (*) (.05) (=) (:34:17:24:)
!! (:2:1:4:) (+) (:3:1:4:) (+) (:4:1:4:) (=) (:6:1:4:)
!! (:34:57:68:) (*) (:34:22:32:) (=) (:>34:70:82:)
!! (12.1) (/) (:2:22:32:) (-) (33.3) (=) (:2:22:32:)
```

I know what you're probably saying right now: "Wow, that is really a lot of power! Is that all *IFG* can do?" Well, there is one more small thing. You can write a COMMAND LINE sequence that will repeat a given number of times. Each time the sequence is repeated, all included FIELD DEFINITION line numbers are incremented. IFG always asks after each repeat cycle if you want to do another. This last feature makes it simple to fill out a form with a multi-line list. Here is a sample repeat sequence:

```
!!@10; "stock #?" :28:2:10: "description?" :28:12:44:
!!"scheduled ship date?" :28:46:56: "quantity?"
:>28:58:62:
!!"unit price?" :>28:65:70:
!! (:28:58:62:) (*) (:28:65:70:) (=) (:>28:72:79:) @
```

This sequence will start at Form Data File line 28 and go to line 38. Notice the repeat sequence is bracketed by "@" symbols and the number between the first "@" and the ";" tells how many repeat cycles. Study this sample for a bit and figure out what it does. Then you can look over Data File 1. It is the whole Form Data File produced by my editor program for my Sales Order Acknowledgment form. Figure 2 shows the results of running *IFG* with the Sales Order Acknowledgment Form Data File. The above repeat sequence is from lines 60-62 of that Sales Order Acknowledgement Form Data File.

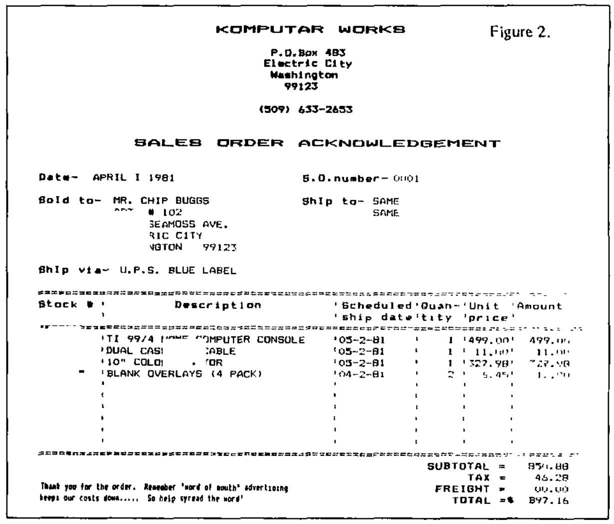


Figure 2.

Finally, Listing 4 is the *Interactive Forms Generator* program. I recommend loading it without all the comment lines to save memory. If you use the disk drive system, you should use CALL FILES(1) and NEW prior to loading *IFG*. That will give space for about a 70-line Form Data File.

Listing 1

```
2000 REM
2010 REM    SUBROUTINE "CASSOUT"
2020 REM
2030 OPEN #1:"CS1",INTERNAL,OUTPUT,FIXED 192
2040 REM
2050 REM    "X" MUST EQUAL THE NUMBER OF TEXT LINES
2060 REM
2070 PRINT #1:X
2080 FOR I=1 TO X+1 STEP 2
2090 PRINT #1:A$(I),A$(I+1)
2100 NEXT I
2110 CLOSE #1
2120 RETURN
```

Listing 2

```
3000 REM
3010 REM    SUBROUTINE "DISKOUT"
3020 REM
3030 PRINT "ENTER WHICH DISK, 1-3?"
3040 INPUT DISK
3050 IF (DISK<1)+(DISK>3)=-1 THEN 3030
3060 PRINT "ENTER FILENAME:"
3070 INPUT NAME$
3080 IF (LEN(NAME$)<1)+(LEN(NAME$)>10)=-1 THEN 3060
3090 OPEN #1:"DSK"&STR$(DISK)&"."&NAME$,OUTPUT,INTERNAL,VARIABLE 132
3100 REM
3110 REM    "X" MUST EQUAL THE NUMBER OF TEXT LINES
3120 REM
3130 PRINT #1:X
3140 FOR I=1 TO X
3150 PRINT #1:A$(I)
3160 NEXT I
3170 CLOSE #1
3180 RETURN
```


Copyright © 1983 Emerald Valley Publishing Co.

Data File 1

```
1 ESCFESCHDC4DC2LFCR
2 SISOESCEESCG            KOMPUTAR WORKSLFLFCR
3                               P.O.Box 483LFCR
4                              Electric CityLFCR
5                               WashingtonLFCR
6                                 99123LFLFCR
7                            (509) 633-2653LFLFLFLFCR
8 DC2DC4ESCFESCHLFCR
9 SOESCE        SALES ORDER ACKNOWLEDGEMENTLFLFLFCR
10 ESCFSISODate-                          S.O.number-CR
11 DC4DC2                                                                        LFCR
12 LFCR
13 SISOSold to-                         Ship to-CR
14 DC4DC2                                                                        LFCR
15 HT                                                                         LFCR
16 HT                                                                         LFCR
17 HT                                                                         LFCR
18 HT                                                                         LFCR
19 LFSISOShip via-CR
20 DC4DC2                                                                  LFLFCR
21 ==========================================================================LFCR
22 SISOStock #          Description          Scheduled Quan- Unit  AmountCR
23 DC4DC2        !                                  !        !      !     !        LFCR
24 SISO                                      ship date tity  priceCR
25 DC4DC2        !                                  !        !      !     !        LFCR
26 DC4DC2CR
27 ==========================================================================LFCR
28 HT          !                            !          !      !     !        LFCR
29 HT          !                            !          !      !     !        LFCR
30 HT          !                            !          !      !     !        LFCR
31 HT          !                            !          !      !     !        LFCR
32 HT          !                            !          !      !     !        LFCR
33 HT          !                            !          !      !     !        LFCR
34 HT          !                            !          !      !     !        LFCR
35 HT          !                            !          !      !     !        LFCR
36 HT          !                            !          !      !     !        LFCR
37 HT          !                            !          !      !     !        LFCR
38 ==========================================================================LFCR
39 SISO                                                SUBTOTAL =CR
40 DC4DC2                                                                        LFCR
41 SISO                                                     TAX =CR
42 DC4DC2                                                                        LFCR
43 SISO                                                 FREIGHT =CR
44 DC4DC2                                                                        LFCR
45 SISO                                                   TOTAL =$CR
46 DC4DC2                                                                        LFCR
47 ESCB:NULCR
48 VTSIThank you for the order.  Remember "word of mouth" advertising LFCR
49 keeps our costs down.....  So help spread the word!LFCR
50 FFCR
51 !!"SALES ORDER ACKNOWLEDGEMENTLFLF" LFCR
52 !!"date?" :11:11:30: "S.O. number?" :11:55:64:CR
53 !!"sold to?" :14:14:37: "sold to address lines=4LF"CR
54 !!"address #1" :15:13:36: "address#2" :16:13:36:CR
55 !!"address #3" :17:13:36: "address #4" :18:13:36:CR
56 !!"ship to?" :14:52:75: "ship to address lines=4LF"CR
57 !!"address #1" :15:51:75: "address #2" :16:51:75:CR
58 !!"address #3" :17:51:75: "address #4" :18:51:75:CR
59 !!"ship via?" :20:15:54:CR
60 !!@10; "stock #?" :28:2:10: "description?" :28:12:44:CR
61 !!"scheduled ship date?" :28:46:56: "quantity?" :)28:58:62:CR
62 !!"unit price?" :)28:65:70: (:28:58:62:)(*)(:28:65:70:)(=)(:)28:72:79:) @CR
63 !! (:28:72:79:)(+)(:29:72:79:)(+)(:30:72:79:)(+)(:31:72:79:)(+)CR
64 !!(:32:72:79:)(+)(:33:72:79:)(+)(:34:72:79:)(+)(:35:72:79:)(+)CR
65 !! (:36:72:79:)(+)(:37:72:79:)(=)(:)40:73:80:)CR
66 !!"tax rate (ie: .0544)?" :42:73:80: (:40:73:80:)(*)(:42:73:80:)(=)CR
67 !! (:)42:73:80:) "freight charge?" :)44:73:80: (:40:73:80:)(+)CR
68 !! (:42:73:80:)(+)(:44:73:80:)(=)(:)46:73:80:)CR
```

Data File 2

```
1 ESCESISOKOMPUTAR WORKSLFCR
2 P.O.BOX 483LFCR
3 143 SUNSET DRIVELFCR
4 ELECTRIC CITY, WASHINGTONLFCR
5      99123LFCR
6 LFLFLFLFCR
7 SO         SHIP TO:LFCR
8                                                                   LFCR
9                                                                   LFCR
10                                                                  LFCR
11                                                                  LFCR
12                                                                  LFCR
13 LFLFLFLFLFCR
14 !!"SHIPPING LABELLFLF"CR
15 !!"CUSTOMER NAME":8:35:70:CR
16 !!"STREET ADDRESS":9:35:70:CR
17 !!"CITY":10:35:70:CR
18 !!"STATE":11:35:70:CR
19 !!"ZIP CODE":12:40:60:CR
```

Listing 3

```
110 REM  "FILEBUILD" PROGRAM--VERSION 3: SHIPPING LABELS
120 REM
130 OPTION BASE 1
140 DIM A$(70)
150 REM
160 REM  NOW FOR THE CHARACTER STRINGS IN THE ARRAY
170 REM
180 A$(1)=CHR$(27)&"E"&CHR$(15)&CHR$(14)&"KOMPUTAR WORKS"&CHR$(10)&CHR$(13)
190 A$(2)="P.O. BOX 483"&CHR$(10)&CHR$(13)
200 A$(3)="143 SUNSET DRIVE"&CHR$(10)&CHR$(13)
210 A$(4)="ELECTRIC CITY, WASHINGTON"&CHR$(10)&CHR$(13)
220 A$(5)="          99123"&CHR$(10)&CHR$(13)
230 A$(6)=CHR$(10)&CHR$(10)&CHR$(10)&CHR$(10)&CHR$(10)&CHR$(13)
240 A$(7)=CHR$(14)&"                    SHIP TO:"&CHR$(10)&CHR$(13)
250 A$(8)="                                        "&CHR$(10)&CHR$(13)
260 A$(9)="                                        "&CHR$(10)&CHR$(13)
270 A$(10)="                                        "&CHR$(10)&CHR$(13)
280 A$(11)="                                        "&CHR$(10)&CHR$(13)
290 A$(12)="                                        "&CHR$(10)&CHR$(13)
300 A$(13)=CHR$(10)&CHR$(10)&CHR$(10)&CHR$(10)&CHR$(10)&CHR$(13)
310 A$(14)="!!"&CHR$(34)&"SHIPPING LABEL"&CHR$(10)&CHR$(10)&CHR$(34)&CHR$(13)
320 A$(15)="!!"&CHR$(34)&"CUSTOMER NAME"&CHR$(34)&":8:35:70:"&CHR$(13)
330 A$(16)="!!"&CHR$(34)&"STREET ADDRESS"&CHR$(34)&":9:35:70:"&CHR$(13)
340 A$(17)="!!"&CHR$(34)&"CITY"&CHR$(34)&":10:35:70:"&CHR$(13)
350 A$(18)="!!"&CHR$(34)&"STATE"&CHR$(34)&":11:35:70:"&CHR$(13)
360 A$(19)="!!"&CHR$(34)&"ZIP CODE"&CHR$(34)&":12:40:60:"&CHR$(13)
370 REM
380 REM  MAKE X EQUAL THE NUMBER OF LINES IN A$
390 REM
400 X=19
410 PRINT "WRITE FILE TO?"
420 PRINT "     1 - CS1"
430 PRINT "     2 - DSK"
440 INPUT CHOICE
450 IF (CHOICE<1)+(CHOICE>2)=-1 THEN 410
460 ON CHOICE GOSUB 530,660
470 PRINT "C O M P L E T E"
480 PRINT "NOW SAVE ME ON TAPE OR DISK......"
490 END
500 REM
510 REM  OUTPUT SUBROUTINES FOLLOW...
520 REM
530 REM
540 REM  SUBROUTINE "CASSOUT"
550 REM
560 OPEN #1:"CS1",INTERNAL,OUTPUT,FIXED 192
570 REM
580 REM  "X" MUST EQUAL THE NUMBER OF TEXT LINES
```

```
590 REM
600 PRINT #1:X
610 FOR I=1 TO X+1 STEP 2
620 PRINT #1:A$(I),A$(I+1)
630 NEXT I
640 CLOSE #1
650 RETURN
660 REM
670 REM   SUBROUTINE  "DISKOUT"
680 REM
690 PRINT "ENTER WHICH DISK, 1-3?"
700 INPUT DISK
710 IF (DISK<1)+(DISK>3)=-1 THEN 690
720 PRINT "ENTER FILENAME:"
730 INPUT NAME$
740 IF (LEN(NAME$)<1)+(LEN(NAME$)>10)=
    -1 THEN 720
750 OPEN #1:"DSK"&STR$(DISK)&"."&NAME$
    ,OUTPUT,INTERNAL,VARIABLE 132
760 REM
770 REM  "X"  MUST EQUAL THE NUMBER
    OF TEXT LINES
780 REM
790 PRINT #1:X
800 FOR I=1 TO X
810 PRINT #1:A$(I)
820 NEXT I
830 CLOSE #1
840 RETURN
```

This program (Listing 4) scans the file for lines that start with !!. Then these Interactive COMMAND LINES are parsed for four types of commands:

1. *Comments* or messages to prompt the interactive user. This type of command is in the form of text preceded by a quote and followed by a quote.
2. *Field-definition* type commands define the physical field into which the user's keyboard input will be stored. The field has the form—
   :<*line number*>:<*start position*>:
   <*end position*>: Example— :23:5:22
   A Sample COMMAND LINE is:
   !!"Enter the serial number—" :19:7:22:
3. Repeat Command Sequence starts with—
   !!@<Numeric Value>:- and must end with a @.
   Everything in between will be repeated the number of times specified by the numeric value. A sample might be:
   (Line 20) Serial Number— Model—
   (Line 21) Serial Number— Model—
   (Line 22) Serial Number— Model—
   (Line ??) !!"Fill in the table values that follow:"
   (Line? + 1) !!@3:"Enter serial number;" :20:15:24
   (Line? + 2) !!"Enter the model number:" '20'31'40' @
4. Math Transformations are made up of terms and operators. Terms may be Field-definition or constant types. Operators are "*", "/", "+", "=", and "-". All terms and operators must each be enclosed in parentheses.
   !!(:23:5:22) (*) (.0544) (=) (:>23:17:35:)
   Note the ">" in the last term which causes the answer to be right justified in the field.

The *IFG* program is set up for use with an EPSON MX-80 printer connected as device:

"RS232.CR.EC.DA=8.BA=9600"

If you are using a different baud rate, the OPEN statements for the printer on line number 1380 must be changed.

You can use a different RS232 printer with the *IFG* program but first check lines 190-360 to make sure these character sequences are compatible with your printer. Especially check RESETEPSON, which initializes the printer.

Listing 4

```
100 REM  * INTERACTIVE FORMS *
110 REM  *    GENERATOR      *
130 REM
140 REM
170 OPTION BASE 1
180 DIM A$(70)
190 RESETEPSON$=CHR$(18)&CHR$(20)&CHR$
    (27)&CHR$(70)&CHR$(27)&CHR$(72)&CH
    R$(13)
200 BLANK$="
    "
210 QUOTE$=CHR$(34)
220 BANG$=CHR$(33)&CHR$(33)
230 COLON$=CHR$(58)
240 SEMICOLON$=CHR$(59)
250 RIGHTARROW$=CHR$(62)
260 AMPERSAND$=CHR$(64)
270 OPENPAREN$=CHR$(40)
280 CLOSEPAREN$=CHR$(41)
290 DECIMAL$=CHR$(46)
300 ZERO$=CHR$(48)
310 PLUS$=CHR$(43)
320 MINUS$=CHR$(45)
330 MULTIPLY$=CHR$(42)
340 DIVIDE$=CHR$(47)
350 EQUAL$=CHR$(61)
360 SPACE$=CHR$(32)
370 YES=1
380 NO=0
390 REM  CENTRAL CONTROL MENU
400 CALL CLEAR
410 PRINT "MAKE A CHOICE--"
420 PRINT
430 PRINT "   1 - LOAD NEW FORM FILE"
440 PRINT "   2 - FILL OUT SAME FORM"
450 PRINT "   3 - PRINT COPIES"
460 PRINT "   4 - TERMINATE"
470 PRINT
480 INPUT CHOICE
490 IF (CHOICE>0)+(CHOICE<5)=-2 THEN 5
    20
500 PRINT "ERROR, TRY AGAIN..."
510 GOTO 410
520 ON CHOICE GOSUB 550,840,1340,3020
530 GOTO 400
540 REM    1 - LOAD NEW FORM FILE SUBR
    OUTINE
550 CALL CLEAR
560 PRINT "ENTER DEVICE:"
570 PRINT "              1 FOR  CS1"
580 PRINT "              2 FOR  DISK"
590 INPUT DEVICE
600 IF (DEVICE<1)+(DEVICE>2)=-1 THEN 5
    60
610 IF DEVICE=2 THEN 690
620 OPEN #4:"CS1",INTERNAL,INPUT ,FIXE
    D 192
630 INPUT #4:X
640 FOR I=1 TO X STEP 2
650 INPUT #4:A$(I),A$(I+1)
660 NEXT I
670 CLOSE #4
680 RETURN
690 REM   -- READ FROM DISK
700 PRINT "ENTER WHICH DISK, 1-3?"
710 INPUT DISKNO
720 IF (DISKNO<1)+(DISKNO>3)=-1 THEN 7
    00
730 PRINT "ENTER FILENAME:"
740 INPUT FILENAME$
750 IF (LEN(FILENAME$)<1)+(LEN(FILENAM
    E$)>10)=-1 THEN 730
760 OPEN #5:"DSK"&STR$(DISKNO)&"."&FIL
    ENAME$,INPUT ,SEQUENTIAL,INTERNAL,
    VARIABLE 132
770 INPUT #5:X
780 FOR I=1 TO X
```


Copyright © 1983 Emerald Valley Publishing Co.

```
790 INPUT #5:A$(I)
800 NEXT I
810 CLOSE #5
820 RETURN
830 REM 2 - FILL OUT SAME FORM
840 REM MAIN SCANNER- LOOKS FOR COMMANDS
850 CALL CLEAR
860 TERM=1
870 FOR I=1 TO X
880 REM >>>> COMMAND LINE?>>>
890 IF SEG$(A$(I),1,2)=BANG$ THEN 910 ELSE 1310
900 REM >>>> IF SO, START OF REPEAT SEQUENCE?>>>
910 IF SEG$(A$(I),3,1)=AMPERSAND$ THEN 920 ELSE 990
920 IF REPEAT=YES THEN 990
930 REM >>>> INITIALIZE THE REPEAT SEQUENCE.>>>
940 REPEAT=YES
950 MAXREPS=VAL(SEG$(A$(I),4,POS(A$(I),SEMICOLON$,4)-4))
960 REPSTART=I
970 REPS=0
980 REM LINE PARSER
990 FOR J=3 TO LEN(A$(I))
1000 P$=SEG$(A$(I),J,1)
1010 IF P$=QUOTE$ THEN 1030 ELSE 1060
1020 REM >>>> GO SUBROUTINE "COMMENT">>>
1030 GOSUB 1480
1040 J=K
1050 GOTO 1300
1060 IF P$=COLON$ THEN 1080 ELSE 1140
1070 REM >>>> GO SUBROUTINE "FIELDINPUT">>>
1080 IF SEG$(A$(I),J+1,1)=RIGHTARROW$ THEN 1090 ELSE 1110
1090 RIGHTJUSTIFY=YES
1100 J=J+1
1110 GOSUB 1650
1120 J=K
1130 GOTO 1300
1140 IF P$=OPENPAREN$ THEN 1160 ELSE 1190
1150 REM >>>> GO SUBROUTINE "MATH TERM">>>
1160 GOSUB 1860
1170 J=K
1180 GOTO 1300
1190 IF P$=AMPERSAND$ THEN 1210 ELSE 1300
1200 REM >>>> IS IT THE 1ST AMPERSAND?>>>
1210 IF J=3 THEN 1300
1220 REM >>>> NO, IT IS THE END OF REPEAT SEQUENCE MARK.>>>
1230 REPS=REPS+1
1240 IF REPS=MAXREPS THEN 1290
1250 INPUT "REPEAT ENTRY? (1=YES 0=NO)":MORE
1260 IF MORE=NO THEN 1290
1270 I=REPSTART
1280 GOTO 890
1290 REPEAT=NO
1300 NEXT J
1310 NEXT I
1320 RETURN
1330 REM 3 - PRINT COPIES
1340 REM FORM PRINT SECTION
1350 CALL CLEAR
1360 PRINT "ENTER NUMBER OF COPIES"
1370 INPUT "TO PRINT-":Z
1380 OPEN #3:"RS232.CR.EC.DA=8.BA=9600",VARIABLE 132
1390 PRINT #3:RESETEPSON$
1400 FOR I=1 TO Z
1410 FOR J=1 TO X
1420 IF SEG$(A$(J),1,2)=BANG$ THEN 1440
1430 PRINT #3:A$(J)
1440 NEXT J
1450 NEXT I
1460 CLOSE #3
1470 RETURN
1480 REM "COMMENT" SUBROUTINE
1490 COMMENT$=""
1500 FOR K=J+1 TO LEN(A$(I))
1510 P$=SEG$(A$(I),K,1)
1520 IF P$=QUOTE$ THEN 1620
1530 P=ASC(P$)
1540 IF P>96 THEN 1550 ELSE 1570
1550 P=P-32
1560 P$=CHR$(P)
1570 COMMENT$=COMMENT$&P$
1580 NEXT K
1590 PRINT "*** ERROR IN LINE #":I
1600 PRINT "*** MISSING QUOTE..."
1610 GOTO 1640
1620 PRINT COMMENT$
1630 PRINT ""
1640 RETURN
1650 REM "FIELDINPUT" SUBROUTINE
1660 FRONT$=""
1670 BACK$=""
1680 REM >>>> DECODE THE FIELD PARAMETERS >>>
1690 GOSUB 2680
1700 PRINT
1710 MIDDLE$=SEG$(A$(LINE),START,LENGTH)
1720 PRINT "**"&MIDDLE$&"**"
1730 PRINT
1740 INPUT TEXT$
1750 IF SEG$(TEXT$,1,1)=RIGHTARROW$ THEN 1760 ELSE 1780
1760 RIGHTJUSTIFY=YES
1770 TEXT$=SEG$(TEXT$,2,LEN(TEXT$))
1780 IF LEN(TEXT$)>LENGTH THEN 1790 ELSE 1830
1790 PRINT "-- TEXT STRING TOO LONG..."
1800 PRINT "PLEASE ENTER SHORTER LINE"
1810 GOTO 1700
1820 REM >>>> GO STUFF THE FIELD >>>
1830 GOSUB 2860
1840 RETURN
1850 REM "MATH TERM" SUBROUTINE
1860 J=J+1
1870 ON TERM GOTO 1890,1990,2050
1880 REM >>>> PROCESS FIRST TERM >>>
1890 IF SEG$(A$(I),J,1)=COLON$ THEN 1900 ELSE 1930
1900 GOSUB 2680
1910 FIRSTTERM$=SEG$(A$(LINE),START,LENGTH)
1920 GOTO 1960
1930 ENDFIELD=POS(A$(I),CLOSEPAREN$,J)
1940 FIRSTTERM$=SEG$(A$(I),J,ENDFIELD-J)
1950 K=ENDFIELD
1960 TERM=2
1970 GOTO 2660
1980 REM >>>> PROCESS SECOND TERM >>>
1990 ENDFIELD=POS(A$(I),CLOSEPAREN$,J)
2000 SECONDTERM$=SEG$(A$(I),J,ENDFIELD-J)
2010 TERM=3
2020 K=ENDFIELD
2030 GOTO 2660
2040 REM >>>> PROCESS THIRD TERM AND CALCULATE >>>
2050 IF SECONDTERM$=EQUAL$ THEN 2070 ELSE 2200
2060 REM >>>> PROCESS ANSWER AND STORE AT 3RD TERM LOCATION >>>
2070 IF SEG$(A$(I),J,1)=COLON$ THEN 2080 ELSE 2170
```

Copyright © 1983 Emerald Valley Publishing Co.

```
2080 IF SEG$(A$(I),J+1,1)=RIGHTARROW$ THEN 2090 ELSE 2110
2090 RIGHTJUSTIFY=YES
2100 J=J+1
2110 GOSUB 2680
2120 TEXT$=FIRSTTERM$
2130 GOSUB 2860
2140 K=K+1
2150 TERM=1
2160 GOTO 2660
2170 PRINT "ERROR IN MATH- FILE LINE#":I
2180 GOTO 2660
2190 REM >>>> GET THE THIRD TERM .....>>>
2200 IF SEG$(A$(I),J,1)=COLON$ THEN 2210 ELSE 2240
2210 GOSUB 2680
2220 THIRDTERM$=SEG$(A$(LINE),START,LENGTH)
2230 GOTO 2270
2240 ENDFIELD=POS(A$(I),CLOSEPAREN$,J)
2250 THIRDTERM$=SEG$(A$(I),J,ENDFIELD-J)
2260 K=ENDFIELD
2270 TERM=2
2280 IF POS(FIRSTTERM$,DECIMAL$,1)+POS(THIRDTERM$,DECIMAL$,1)=0 THEN 2310
2290 ALIGN=YES
2300 GOTO 2330
2310 ALIGN=NO
2320 REM >>>> OK, NOW DO MATH >>>
2330 IF POS(BLANK$,FIRSTTERM$,1)=NO THEN 2350
2340 FIRSTTERM$=ZERO$
2350 IF POS(BLANK$,THIRDTERM$,1)=NO THEN 2370
2360 THIRDTERM$=ZERO$
2370 FIRSTTERM=VAL(FIRSTTERM$)
2380 THIRDTERM=VAL(THIRDTERM$)
2390 IF SECONDTERM$=PLUS$ THEN 2400 ELSE 2420
2400 TEMPTERM=FIRSTTERM+THIRDTERM
2410 GOTO 2530
2420 IF SECONDTERM$=MINUS$ THEN 2430 ELSE 2450
2430 TEMPTERM=FIRSTTERM-THIRDTERM
2440 GOTO 2530
2450 IF SECONDTERM$=MULTIPLY$ THEN 2460 ELSE 2480
2460 TEMPTERM=FIRSTTERM*THIRDTERM
2470 GOTO 2530
2480 IF SECONDTERM$=DIVIDE$ THEN 2490 ELSE 2510
2490 TEMPTERM=FIRSTTERM/THIRDTERM
2500 GOTO 2530
2510 PRINT "MATH OPERATOR BAD - FILE LINE #":I
2520 GOTO 2660
2530 TEMPTERM=INT(TEMPTERM*100)/100
2540 TEMPTERM$=STR$(TEMPTERM)
2550 IF ALIGN=NO THEN 2650
2560 IF POS(TEMPTERM$,DECIMAL$,1)=0 THEN 2590
2570 ADJUST=LEN(TEMPTERM$)-POS(TEMPTERM$,DECIMAL$,1)+1
2580 ON ADJUST GOTO 2610,2630,2650
2590 FIRSTTERM$=TEMPTERM$&DECIMAL$&ZERO$&ZERO$
2600 RETURN
2610 FIRSTTERM$=TEMPTERM$&ZERO$&ZERO$
2620 RETURN
2630 FIRSTTERM$=TEMPTERM$&ZERO$
2640 RETURN
2650 FIRSTTERM$=TEMPTERM$
2660 RETURN
2670 REM      GET FIELD DEF. SUBROUTINE
2680 K=J+1
2690 NEXTCOLON=POS(A$(I),COLON$,K)
2700 LINE=VAL(SEG$(A$(I),K,NEXTCOLON-K))
2710 K=NEXTCOLON+1
2720 NEXTCOLON=POS(A$(I),COLON$,K)
2730 START=VAL(SEG$(A$(I),K,NEXTCOLON-K))
2740 K=NEXTCOLON+1
2750 NEXTCOLON=POS(A$(I),COLON$,K)
2760 LEND=VAL(SEG$(A$(I),K,NEXTCOLON-K))
2770 K=NEXTCOLON+1
2780 IF REPEAT=NO THEN 2800
2790 LINE=LINE+REPS
2800 LENGTH=LEND-START+1
2810 IF LENGTH<1 THEN 2820 ELSE 2840
2820 PRINT "*** ERROR IN LINE #":I
2830 PRINT "*** FIELD LENGTH NEGATIVE..."
2840 RETURN
2850 REM      FIELD STUFFER SUBROUTINE
2860 IF LEN(TEXT$)=0 THEN 3010
2870 IF RIGHTJUSTIFY=NO THEN 2930
2880 FOR M=LEN(TEXT$)TO LENGTH-1
2890 TEXT$=SPACE$&TEXT$
2900 NEXT M
2910 RIGHTJUSTIFY=NO
2920 GOTO 2960
2930 FOR M=LEN(TEXT$)TO LENGTH-1
2940 TEXT$=TEXT$&SPACE$
2950 NEXT M
2960 IF START=1 THEN 2980
2970 FRONT$=SEG$(A$(LINE),1,START-1)
2980 IF LEND=LEN(A$(LINE))THEN 3000
2990 BACK$=SEG$(A$(LINE),LEND+1,LEN(A$(LINE)))
3000 A$(LINE)=FRONT$&TEXT$&BACK$
3010 RETURN
3020 END
```


Copyright © 1983 Emerald Valley Publishing Co.

Getting Down to Business

Risks and Benefits

You don't need to be reminded that microcomputers are having more than a micro impact on business. If you are reading this, it is because you would like some of that impact to benefit you. In this series of articles, we will explore some of those benefits and show you how to incorporate them in your business or professional work. They will be at least partly cautionary—written to try to keep you out of trouble. And don't expect only *success* stories. After all, *failures* can be most instructive. . .

Planning Use vs. Integrated Use

It is important to distinguish between two major and very different categories of business and professional use of the computer. The first I will call *planning* use. This category includes a lot of activities that are helpful to business and professional people. Applications in this category tend to be analytical or evaluative. They need not be done on a regular basis, but can often be a dramatic help in charting future direction and improving the profitability of a business. Some applications require rather little in the way of input data and are essentially projections; others analyze whatever body of historical data that might be available. Some common examples are the following:

- Comparisons of ROI (Return On Investment) for the various options.
- Interest calculations (e.g., effective interest rates on installment loans).
- Profitability analyses for comparing charges and costs of providing various services.
- Lease vs. purchase analyses.

The second category of use is what I call *integrated* use. This category includes a lot of functions that support a business on a minute-to-minute or day-to-day basis. These are, for example:

- Maintenance of inventory records.
- Preparation of invoices, orders, service contracts, bills, etc.
- Accounts payable and accounts receivable.
- Maintenance of customer or mailing lists.
- Payroll records.
- General ledger and other accounting records.

The potential benefits to your business of applications like these are enormous. But then, so are the risks! Before you allow your business to become dependent on a microcomputer (or any other computer) and set of computer programs, there are a number of steps you must take to safeguard it against the small and large catastrophes that could be (at the least) a major setback for you. This is not to *discourage* you from integrated uses, but rather, to *encourage* you to be very careful about implementing them. [You should also look at "Murphy's Law," which has some steps we recommend you take to protect yourself against this ubiquitous and insidious law: "If anything can go wrong, it will!" It *may* apply (and indeed *has* applied—more frequently than most would care to admit) to integrated computer applications.—Ed.]

A good place for you to start using the power of your microcomputer is in planning applications. They don't require extensive systems of programs or comprehensive detailed business records. They don't need to be done at any given moment, at peril of disaster to your business. And you don't have to chase down some itinerant programmer or software house to update your program upon change of, say, some federal tax formula—again, at peril of disaster. Furthermore, you can implement some planning applications yourself, without extra software, disk drives, extensive data files, or a lot of time.

Projections: A Planning Use

Perhaps you've heard the story of the wealthy Indian maharajah who was challenged to a chess match by a shrewd foreign merchant. The merchant put up one hundred gold coins as his part of the wager, but only asked for rice if the maharajah lost the game: one grain of rice on the first square of the chessboard, two grains on the second square, four on the third, eight on the fourth, and so on. The maharajah was amused and somewhat skeptical that the merchant would ask for only a few grains of rice, but nevertheless accepted the challenge. Naturally—or there would be no point to the story—the maharajah lost. And as the prize was being paid, the full impact became shockingly clear: So much rice did not exist in the world! And even if it did, the immense wealth of the maharajah could have paid for only a tiny fraction of it. . .

Copyright © 1983 Emerald Valley Publishing Co.

You are not often confronted with this type of wager. But you do have opportunities to evaluate, just as the maharajah *should* have. A computer can help you to project events into the future, vary the assumptions and tabulate the projected results. Using a computer program, you can analyze a much more complex situation than you would be willing to do with just a pencil, calculator and paper. You can change your assumptions and let the computer recalculate and reprint the projections, and thus gain much more understanding of the consequences of various contingencies as you play what is essentially a game of "What if. . . ?" As an extra benefit, consider this effect: The necessity of making clear and explicit assumptions usable by the computer may force you to think more clearly and objectively than you might have done otherwise. (I wonder whether baseball clubs would pay as much for some of their benchwarmers and stars if they evaluated the consequences and contingencies objectively.)

A Program Outline

Perhaps the best thing about a projection program is the ease with which you can write it yourself in BASIC. The fundamental tool is a two-dimensional array. If you thought anything connected with arrays was necessarily complex and difficult, please read on. You'll soon discover that an array application can be a lot easier than you imagined.

An array is nothing more than a table in computer storage; a two-dimensional array has rows and columns. We must assign meaning to each, and write our program to honor those meanings. In a projection program, I let each column represent a year (or month?). If the problem requires, I let the numbers in the first column represent initial values, investments, or costs; the numbers in the last column represent residual values, or perhaps totals over all years in the projections. Each row represents a significant quantity that we want to project over the time span.

	1 1981	2 1982	3 1983	4 1984
1 Rents				
2 Vacancy loss				
3 Gross revenue				
4 Property tax				
5 Insurance				
6 Interest				
7 Maintenance				
8 Management				
9 Depreciation				
10 Gross expenses				
11 Net income				

Figure 1. Use of a Table for a Rental Projection

Figure 1 shows a simple projection of a rental operation. There are four columns in the array; they represent 1981, 1982, 1983, and 1984. Each row represents a quantity necessary to the projection of rental results. The array can be declared in BASIC by:

```
60 DIM T(11,4)
```

A BASIC program can refer to any number in the array: For example, to refer to the maintenance expense in 1982 we refer to T(7,2).

1. Set initial (1981) values
2. FOR each additional year compute the projected values
3. FOR each row of the table PRINT a row of the table

Figure 2. Outline of a Projection Program

The outline of the program is shown in Figure 2. Let us examine each of the steps of the outline and show how it would be programmed in BASIC. A bunch of LET statements takes care of the first step. If rental income for 1980 is projected to be $20,000, with a vacancy rate of 5 percent, property tax of $3200, insurance or $700, etc., the first several BASIC statements would be:

```
1010 LET T(1,1)=20000
1020 LET T(2,1)=.05*T(1,1)
1030 LET T(3,1)=T(1,1)-T(2,1)
1040 LET T(4,1)=3200
1050 LET T(5,1)=700
```

These illustrate several ways of assigning values:

- directly as a given number (as in statements 1010, 1040, 1050);
- as a multiple of another number (as in statement 1020)
- as sum or difference of other numbers (as in statement 1030);

It will be clear from your application how to assign each of your values.

```
  10 REM  SKELETON OF A PROJECTION PROG
        RAM
  20 REM  ROWS OF T REPRESENT INCOME OR
          EXPENSE ITEMS
  30 REM  COLUMNS OF T REPRESENT YEARS
  60 DIM T(11,4)
1000 REM  STEP 1.  INITIAL VALUES
1010 LET T(1,1)=20000
1020 LET T(2,1)=.05*T(1,1)
1030 LET T(3,1)=T(1,1)-T(2,1)
1040 LET T(4,1)=3200
1050 LET T(5,1)=700
1990 REM  STEP 2.  PROJECT TO FUTURE YEA
        RS
2000 FOR J=2 TO 4
2010 LET T(1,J)=T(1,J-1)*1.08
2020 LET T(2,J)=.05*T(1,J)
2030 LET T(3,J)=T(1,J)-T(2,J)
2040 LET T(4,J)=T(4,J-1)*1.06
2050 LET T(5,J)=T(5,J-1)*1.10
2200 NEXT J
2990 REM  STEP 3.  PRINT THE RESULTS
3000 FOR K=1 TO 11
3010 PRINT K,T(K,1),T(K,2),T(K,3),T(K,4
        )
3020 NEXT K
9990 END
```

The second step of the program is probably the most complex. The idea is to march across the table, usually deriving each number from the one to its left—that is, from the corresponding entry for the previous year. However, some of these entries, too, will be multiples, sums, or differences of other numbers in the same columm. We can use the BASIC statement FOR to good advantage here; it easily specifies a repetition for each year. In our rental example, these statements could be:

```
2000 FOR J=2 TO 4
2010 LET T(1,J)=T(1,J-1)*1.08
2020 LET T(2,J)=.05*T(1,J)
```


Copyright © 1983 Emerald Valley Publishing Co.

```
2030 LET T(3,J)=T(1,J)–T(2,J)
2040 LET T(4,J)=T(4,J-1)*1.06
2050 LET T(5,J)=T(5,J–1)*1.10
  :    :
2200 NEXT J
```

These statements reflect assumptions that:

- Rental income increases at an 8% inflation rate.
- Vacancy continues at 5%.
- Property taxes increase at only (!) a 6% inflation rate.
- Insurance costs increase at a 10% inflation rate.

If you are not sure how all this works, take out your pencil and, for J with a value of 2, play computer by filling in numbers in the table yourself as the computer would.

Note how all the assumptions are built into the program; each one can be changed at will. You should, in fact, change several, and re-RUN the program several times in order to see the effect of each of your assumptions. This is sometimes call *sensitivity analysis*, but don't let big words scare you.

You may also use the full capabilities of BASIC for special situations. For example, we might project that in the third year, the property will be annexed to the city and taxes will go up 30 percent instead of 6 percent. We could replace statement 2040 by:

```
2040 IF J=3 THEN 2047
2043 LET T(4,J)=T(4,J–1)*1.06
2044 GOTO 2050
2047 LET T(4,J)=T(4,J–1)*1.30
```

Also, suppose that in the same year we expect to have to put on a new roof for $8000; this is a maintenance expense, but one in addition to the regular budgeted maintenance. And unlike the taxes, the extra maintenance does not continue into 1984. We may use another form of the multiplication here:

```
2070 IF J=3 THEN 2077
2073 LET T(7,J)=T(7,1)*1,08 (J–1)
2074 GOTO 2080
2077 LET T(7,J)=T(7,1)*10.8 2+8000
```

In the last step we display the table. The print-out can be prettied up with column headings, a description of each row, and other features. A bare-bones approach is sufficient, really, and could look like this:

```
3000 FOR K=1 TO 11
3010 PRINT K,T(K,1),T(K,2),T(K,3),T(K,4)
3020 NEXT K
```

This segment prints the four numbers of each row of the table on one line, so the table appears on paper just the way we have been thinking about it; each row of numbers is preceded by the row number (K), which at least helps you to identify and keep track of your output.

Listing 1 gathers these program segments into one skeleton. With this as as guideline, you should now be able to sit down and develop your own useful projection programs—applied to sales, production, commissions, or whatever else you need.

Getting Down to Business

Evaluating a Software Package

In the first section, I defined two categories of computer applications for business: (1) *planning*—concerned mostly with projections, and not having to be done at particular moments at peril to a business; and (2) *integrated use*—applications such as invoices, accounts payable and receivable, mailing list maintenance, general ledger, inventory, or others upon which a business crucially depends at particular times. In this article, we'll explore some of the implicatons of integrated use.

Programs for integrated use are likely to be rather extensive. After all, most such applications involve organization and management of significant quantities of data. This means that the programs must help you with the data entry, help you monitor the validity and correctness of the data, and help you update the data. The programs must also be able to retrieve data for processing, summarizing, and answering inquiries. Depending on the application, the programs may also have to generate controls for audit purposes, and provide tax reports.

The programs for an integrated use application must be well-designed and form what we would call an *information system*. To develop such a system takes a substantial amount of work probably several *months*, if not *years*, of programmer time. If your application is small enough for you to think about doing it on a TI-99/4A or other micro, it would be quite a mismatch of investment for you to pay for even six months of a programmer's time to develop a system. Therefore, you will want to buy a system that is already developed, packaged, and ready to install and use. You actually have a better chance of getting a good working product by buying a package than by having it done to your specifications by a programmer.

OK, you're in the market for a package. Besides cost, the most obvious criterion is whether a proposed package will meet your needs. Now is the time—even before seeing the details of a proposed package—to make yourself a checklist of the features you want your package to include. List each processing action that you think necessary in your system. Consider the data elements you think would have to be stored and related to each other in order to provide the information you will need at any given mo-

Copyright © 1983 Emerald Valley Publishing Co.

ment. If done in a detailed and comprehensive way, this would be close to what we would call a *systems analysis* of your application.

Great detail and comprehensiveness are not needed; the idea is to give you a starting point for judging the adequacy of a package you may be offered. You will probably find that a particular package is organized differently and operates differently from your outline. There's nothing wrong with that. Concentrate on the *results* produced and whether they are appropriate: Does the proposed package provide the information you consider essential? Then, of course, you can also judge whether the proposed package is convenient or awkward, and flexible or rigid.

A second suggestion is to talk to other users of the proposed package, and get their opinions of the package's strengths and weaknesses. You may be surprised how willing other users are to share their experiences. Even if you have to phone a couple of users long-distance, it will be well worth the trouble and cost.

You should not expect your needs in an information system to always remain the same. Your business changes; auditors make new demands; federal or state regulations change. This is where flexibility of a system comes in. Chances are that there will come a time when you will want your system to do something it was not designed to do. Then, you will need help in modifying the system. The supplier of the package is in the best position to know how to modify your system. But will he be around when you need him? Find out whether the *source* program is supplied and accessible to you. If it is, then you have a chance of getting someone near you to modify it when needed. Try to find out from the supplier and users how much trouble a minor modification would be. You may not be able to trust an answer you get absolutely, because judging how hard it will be to modify a program is difficult, but this is the best suggestion I can make.

In the next section I will review some business-related software. This will provide an opportunity for some more specific suggestions about the analysis of a package.

Now let us turn our attention to something more tangible—a program that should be of practical use to many of you.

Effective Interest Rate or Return On Investment

Suppose you have an opportunity to buy an investment for $1500. The investment is expected to pay $140 at the end of each of the next five years, and at the end of five years return a lump sum of $2000. What is the effective interest rate or total yield on this investment? Or, put another way, what is the return on this investment? This problem can be stated in terms of capital in your business: If you invest some amount in a certain piece of equipment or in a higher level of inventory or. . . ,you expect some estimated improvement in revenues. What is your expected return on this investment?

Since you have many opportunities and a limited amount of capital, you need to compare the expected rates of return on each of several opportunities in order to be able to make the best decision. Of course, there are usually intangible benefits, as well as variations in the risks of different investments. A return-on-investment calculation is, therefore, not the only—or necessarily the deciding—criterion in your decisions. Nevertheless, it will certainly provide valuable input in your decision-making process.

The program presented here is a relatively simple one. I define a component of the investment as one or more payments of equal amounts made at regular intevals. An investment will have two or more components; they are the main input to the program. Each component is described by:

(a) the amount of each payment (there may be only one).
(b) the time at which the first of these payments is made. Time is measured in months from the current moment, which is understood to be time zero.
(c) the number of months between payments. This is irrelevant if there is only one payment in a component, but we require a number anyway.
(d) the number of payments in this component.

For instance, the example above includes three components:

	(a)	(b)	(c)	(d)
1st Component	**1500**	**0**	**1**	**1**
2nd Component	**– 140**	**12**	**12**	**5**
3rd Component	**– 2000**	**60**	**1**	**1**

Note that the investment amount is given as a positive number, but the returns on the investment are given as negative numbers. The second component represents the five annual payments (12 months apart) starting 12 months after the current time. The first and third components represent single payments: the initial payment and the final payoff after five years (60 months).

The program makes provision for up to ten components; the number of components is the first input the program asks for.

The program strategy is to compute the residual present value at one interest rate higher and one lower than the effective interest rate. We use an interpolation formula to produce a better estimate of the effective interest rate, then narrow the range of possible effective interest rates, and repeat the process. The program stops when the residual value is less than some fraction of the total of the numbers used in computing the residual value, or when the range of possible effective interest rates is less than some tolerance. There are four parameters set in statements 200-230 of the program that you may want to change, depending on your requirements:

U9—starting upper bound for effective interest rate, set now at 30%.
L9—starting lower bound for effective interest rate, set now at 0%.
T9—tolerance for range of effective interest rate, set now at .05%. When the possible range is less than this, we conclude you have the rate closely enough.
P9—tolerance for residual present value, set now at .0001. Because of round-off error during the calculations, this tolerance should not be reduced much below this value.

Copyright © 1983 Emerald Valley Publishing Co.

Figure 1

```
ENTER NUMBER OF PAYMENT COMPONENTS? 3
ENTER AMOUNT OF PAYMENT? 1500
ENTER TIME OF FIRST OF THESE PAYMENTS? 0
ENTER PERIOD BETWEEN THESE PAYMENTS, IN MONTHS? 1
ENTER NUMBER OF THESE PAYMENTS? 1

ENTER AMOUNT OF PAYMENT? -140
ENTER TIME OF FIRST OF THESE PAYMENTS? 12
ENTER PERIOD BETWEEN THESE PAYMENTS, IN MONTHS? 12
ENTER NUMBER OF THESE PAYMENTS? 5

ENTER AMOUNT OF PAYMENT? -2000
ENTER TIME OF FIRST OF THESE PAYMENTS? 60
ENTER PERIOD BETWEEN THESE PAYMENTS, IN MONTHS? 1
ENTER NUMBER OF THESE PAYMENTS? 1

RESIDUAL PRESENT VALUE AT 0% IS -1200
RESIDUAL PRESENT VALUE AT 30% IS 731.7656652
RESIDUAL PRESENT VALUE AT 18.63580073% IS 290.8235145
RESIDUAL PRESENT VALUE AT 15.00040794% IS 93.29345296
RESIDUAL PRESENT VALUE AT 13.91833345% IS 27.69506322
RESIDUAL PRESENT VALUE AT 13.60435554% IS 8.02691232
RESIDUAL PRESENT VALUE AT 13.5139594% IS 2.310160891
RESIDUAL PRESENT VALUE AT 13.48799321% IS .6635205027
RESIDUAL PRESENT VALUE AT 13.48053936% IS .1904640003
EFFECTIVE INTEREST RATE COMPOUNDED MONTHLY,
    IS 13.48053936
```

Figure 1 shows a transcript of the execution of the program with the sample data given above.

Note that the program uses a subroutine starting at line 720; a parameter R is supplied to the subroutine, and parameters V and V3 are returned. If you have Extended BASIC, you can make these parameters explicit in the subroutine call. You can also rephrase some of the control structures using IF-THEN-ELSE and multi-line statements, and make the program much more readable. I leave this for you to explore.

Lease vs. Purchase Analysis

Quite complex programs are available to determine whether leasing or purchasing some piece of equipment is more advantageous. The effective interest rate program can be used for lease vs. purchase analysis, though it requires you to do some side calculation. One method of the analysis would be essentially to calculate the return on *purchasing* the equipment and *leasing* it back to someone else. You would include:

- cost of purchase(+)
- tax benefits from claimed depreciation (−)
- lease payments (−)
- maintenance cost, if maintenance is provided under the lease (+)
- any difference in insurance or other costs between purchasing and leasing (+ or −)
- expected cost of purchase at the end of lease period (−) *or* trade-in value at the end of lease period (−)

The rate of return indicated by this analysis can be compared with your borrowing cost, and the comparison would give you an indication of whether purchase or lease would be more advantageous to you.

As a small example, suppose you are going to get a widget-grinder. You can buy it for $12,000, or lease it for three years at $300 per month. No maintenance is involved, and the insurance cost is the same under lease or purchase. You expect that after three years you would need to trade this one in on a larger model. If you buy it, the trade-in allowance will be $6000. Assuming that either depreciation or lease payments would cost a net of only 60 percent of the actual amounts because of an assumed 40 percent tax rate, the input to the program would therefore be:

	(a)	(b)	(c)	(d)
1st Component	12000	0	1	1
2nd Component	−1200	12	12	3
3rd Component	−180	0	1	36
4th Component	−6000	36	1	1

If you want to check, this example gives an effective interest rate of about 14.1%. Presumably, it would be advantageous to purchase the widget grinder instead of leasing it.

Effective Interest Rate Program: Table of Variables

Arrays:

A1: amount of each payment in an investment component*

T1: time at which the first payment of that component is made (in months, from current time = 0)

F1: number of months between the payments in this component

N1: number of payments in this component

***An investment component is a series of one or more equal payments made at fixed intervals. Payments may be paid out (+) or received (−).**

Parameters:

U9: upper limit for effective annual rate

L9: lower limit for effective annual rate

T9: tolerance: when the interval between upper and lower limits (L1, U1) is less than this, the program stops

U9: tolerance—when the residual present value at a trial interest rate, divided by the sum of the absolute values of all components, is less than this, the program stops

C: number of components

I: index of the current investment component under consideration (always goes from 1 to C)

L1: current lower bound on effective rate

U1: current upper bound on effective rate

R: trial interest rate, on which to calculate residual present value V

V: residual present value, based on trial interest rate R

L2: residual present value at lower limit L1

U2: residual present value at upper limit U1

V3: sum of absolute values of component present values

V4: present value of a component at rate R

V5: temporary variable used in computing V4

R1: monthly increase factor, using rate R

Copyright © 1983 Emerald Valley Publishing Co.

PROGRAM OUTLINE:
Effective Interest Rate

Line Nos.	
200-230	Set Parameters.
250-370	Obtain input data from user.
400-560	Set lower and upper limits, and the residual present value at each.
590-700	Iterate: interpolate to get a new trial interest rate R, replace either upper or lower bound by R.
720-920	Subroutine: computes residual present value at the trial rate R; also computes V3.
930-950	Report final result.

```
100 REM ************************
110 REM * EFFECTIVE INTEREST RATE *
120 REM ************************
130 REM
140 REM
150 REM
160 REM
170 REM
180 REM
190 DIM A1(10),T1(10),F1(10),N1(10)
200 U9=30.0
210 L9=0
220 T9=0.05
230 P9=1.0E-4
240 REM ACCEPT INPUT
250 PRINT "ENTER NUMBER OF PAYMENT COMPONENTS";
260 INPUT C
270 FOR I=1 TO C
280 PRINT
290 PRINT "ENTER AMOUNT OF PAYMENT";
300 INPUT A1(I)
310 PRINT "ENTER TIME OF FIRST OF THESE PAYMENTS";
320 INPUT T1(I)
330 PRINT "ENTER PERIOD BETWEEN THESE PAYMENTS, IN MONTHS";
340 INPUT F1(I)
350 PRINT "ENTER NUMBER OF THESE PAYMENTS";
360 INPUT N1(I)
370 NEXT I
380 PRINT
390 REM SET LOWER & UPPER BOUNDS FOR EFFECTIVE RATE
400 L1=L9
410 U1=U9
420 REM GET RESIDUAL VALUE AT LOWER BOUND
430 R=L1
440 GOSUB 720
450 L2=V
460 IF ABS(V/V3)<P9 THEN 930
470 REM GET RESIDUAL VALUE AT UPPER BOUND
480 R=U1
490 GOSUB 720
500 U2=V
510 IF ABS(V/V3)<P9 THEN 930
520 REM RESIDUAL VALUES MUST HAVE OPPOSITE SIGNS AT THE BOUNDS
530 IF U2*L2<0 THEN 590
540 PRINT "EFFECTIVE RATE NOT BETWEEN ";L9;" AND ";U9
550 PRINT "CHECK YOUR INPUT OR CHANGE BOUNDS L9 AND U9"
560 GOTO 950
570 REM INTERPOLATE BETWEEN LOWER & UPPER BOUNDS
580 REM FOR NEW TRIAL RATE R
590 R=(L1*U2-U1*L2)/(U2-L2)
600 GOSUB 720
610 IF ABS(V/V3)<P9 THEN 930
620 REM TRIAL RATE REPLACES WHICHEVER BOUND HAS RESIDUAL VALUE WITH THE SAME SIGN
630 IF V*L2>0 THEN 670
640 U1=R
650 U2=V
660 GOTO 690
670 L1=R
680 L2=V
690 IF U1-L1<T9 THEN 930
700 GOTO 590
710 REM SUBROUTINE TO COMPUTE RESIDUAL VALUE V AT RATE R
720 V=0
730 V3=0
740 FOR I=1 TO C
750 IF N1(I)>1 THEN 790
760 REM COMPUTE RESIDUAL VALUE IF ONLY ONE PAYMENT
770 V4=(1+R/1200)^(-T1(I))*A1(I)
780 GOTO 880
790 IF R<>0 THEN 840
800 REM SPECIAL CASE WHEN R=0
810 V4=N1(I)*A1(I)
820 GOTO 880
830 REM COMPUTE RESIDUAL VALUE OF SERIES OF PAYMENTS
840 R1=1+R/1200
850 V5=(1-R1^(-N1(I)*F1(I)))/(1-R1^(-F1(I)))
860 V4=A1(I)*R1^(-T1(I))*V5
870 REM IN ALL CASES, INCLUDE V4 IN V AND V3
880 V=V+V4
890 V3=V3+ABS(V4)
900 NEXT I
910 PRINT "RESIDUAL PRESENT VALUE AT ";R;"% IS ";V
920 RETURN
930 PRINT
940 PRINT "EFFECTIVE INTEREST RATE, COMPOUNDED MONTHLY, IS ";R
950 END
```


Copyright © 1983 Emerald Valley Publishing Co.

Getting DOWN to Business

Inventory

In this section, let's consider an inventory system for your computer. I don't have a particular system to review, but I want to discuss what should be involved in an inventory system, and why. This has implications for a number of other applications you might like for your business, such as order processing, accounts payable, and even a general ledger system.

We will address the kind of system you might use in a sales organization—either in a store or for mail or phone order. Some of the description also fits the situation of a raw materials inventory or even miscellaneous supplies. Since some of the activities may not fit you if your business is small, be prepared to discount some of the benefits.

First, it is important to know *why* you apply a computer to some task. You should have specific advantages in mind and know what you have to do to attain those advantages. And of course you should be prepared to change your operations as necessary. I have seen organizations that wanted to "put it on the computer" without any clear reason. Often such organizations waste time and resources changing the specifications, design, and operation of a system as they struggle to develop reasons for their system on the fly. Others merely wind up with a system which is a burden to run, with no advantages except an imagined prestige.

On the other hand, I did some work not long ago with a company that was going to get a computer. I expected that payroll would be one of their first applications, as it is in so many companies. But no, they had payroll near the bottom of their list. Because they had only 60 or so employees, they were able to do their payroll manually quite well and had other uses in mind that would give them definite advantages. For them, one of the first priorities was inventory.

What are some of the possible benefits of keeping your inventory records by computer?

1. If you are processing orders by computer, you can improve the efficiency of your warehouse operation in several ways:

   a. The computer can recognize which orders cannot be filled, and thus avoid sending the warehouse crew to look for the items.

   b. The computer can produce "pick slips" or a "picking list" arranged in a sequence to make the picking of the items from the warehouse efficient.

   c. The computer can help manage back orders; when new stock arrives, the computer automatically scans the file of back orders and fills any back orders for the items before allowing new orders a chance.

2. The computer can help you manage your inventory levels effectively and save you money. To do this, you must have good projections of future demand for each item. You can then time your reorders and calculate optimal reorder quantities. At least in theory, you should be able to reduce your working capital tied up in inventory, and at the same time be out of stock less often and therefore be able to fill more orders and keep customers happier.

Next, let's consider the information you must keep in your inventory file in order to have an effective inventory system. This file has a record for each product (and perhaps for each size, color, model and style). There is inventory status information: current quantity on hand, quantity on order from vendors, date expected, quantity on back order to customers, and quantity sold since last update. There is also historical demand information, such as quantity sold in each month in perhaps the last year. Finally, there is reorder and forecast information: i.e., preferred vendor, vendor's product number, vendor lead time, order quantity, order frequency or reorder point, and demand forecast.

If your computer is processing orders, you also maintain files of back orders (if permitted). The order processing programs obviously use and update the inventory file. If your computer is not used for processing orders, you must find some other means of updating your inventory data. One of the troubles with this is that your input data to the inventory system are likely to be much less reliable than the order-processing input would be.

An inventory system must include a number of other functions. There are simple updates to price, cost, and warehouse location, as well as addition and deletion of products. There are also inventory adjustments caused by events such as return of an item from a customer, or the removal of an item for product testing. The function of receiving into inventory is complex: Quantities on hand and on order must be updated. A payable transaction is generated—with its necessary comparison of actual arrival amount with invoiced quanitity—so there is an interface with your accounts payable system, if you are using one. Then your system must be sure to trigger the filling of back orders from the new stock *before* letting any new orders have access to it.

Periodically, you must count your physical inventory and adjust your computer inventory accordingly, since you need your inventory file to reflect reality, not wishful thinking. Many events can cause a discrepancy in inventory counts—things such as pilferage, mislabeling, or failure to make the minor adjustments necessitated by the odd-but-authorized removal or replacement of items. The computer should help the physical inventory process by

Copyright © 1983 Emerald Valley Publishing Co.

printing the stock list, and by making it easy to adjust the inventory for discrepancies found.

And then there are the functions involved in reordering: About once a week your system should sweep through all products and determine what to reorder. You of course have the opportunity to override the computer's suggestions, but any such decisions must be recorded (e.g., in quantity on order). Perhaps once a month your system should update some analysis programs that keep your demand history current and recompute forecasts, reorder quantities, etc.

There is even a connection to your general ledger system. After all, inventory is an asset, and any activity that affects the value of that asset should be reflected in your profit and loss, assets and liabilities.

All this is a great deal of work. Not only are there a lot of things to do, but they must be done accurately. I knew a company that went bankrupt, *primarily* because the order processing and inventory control they did by computer was not accurate. They tremendously overstocked some items because the computer said there were none on hand (and of course didn't fill orders because it thought there was no stock), and ran out of stock on other items because the computer thought there were plenty. Naturally the company couldn't fill those orders either! One of the causes of the snafu was the company's lack of understanding of how the system was supposed to work, how to ensure its accuracy, and how to diagnose inaccuracies.

On the other hand, another company, where I helped install order and inventory processing, listened very carefully to what we told them about operation for accuracy. They were not only willing to tighten some of their operations, but were also eager to be able to control their warehouse functions more closely. That company is still prospering.

There's another side to consider too. If you have a small list of products to keep track of, you probably do rather well keeping track of them already. And as for calculating optimal inventory levels, reorder quantities, etc., you can probably do that rather well with a pocket calculator and formulas you can find in many textbooks. So honestly, *would* the computer help you do a better job of managing inventory than you already do (or could do manually with the same effort you would have to put into a computer system)? If the answer is no, then save yourself money, time, and management energy by *not* doing computer inventory. If there are real benefits you would receive, I hope this section will make you a little more aware of what you must prepare for and strengthen your resolve to do it carefully, and do it accurately. The stakes are too high for you to wander casually into a computer inventory system!

Getting DOWN to Business

When Random Does Not Mean By Chance

Random-access files are extremely important in any conversational application that requires a data base of some kind. This includes any kind of *business* information system, but also includes a lot of others as well. Unfortunately, the concept of what *random access* actually is often gives rise to misunderstanding and even fear—that is, the fear that using random access is too complex to be attempted. In this article I will try to correct some of the misunderstandings and start you on your way to using random-access files.

The dictionary I took to college told me that random meant "going, made, occurring, etc., without definite aim, purpose, or reason." Synonyms given are *haphazard, chance, casual, aimless*. Thus, when I first heard of random access in reference to computer data, it didn't sound like anything I would want. The good people didn't *mean* haphazard or chance, or any of those other things; they meant access *directly* to a piece of data specifically wanted, *without* having to pass sequentially by a lot of other unwanted data to get there. To me, this is much better described by the phrase *direct access*, and I have been using direct access and talking against the term *random access* for years. But enough. The terminology *random-access* appears in my newer dictionary, and is generally understood in computer circles to mean "permitting access to stored data in any order the user desires." From the standpoint of the storage unit, access *is* random in the older sense, since the sequence of access requests is not at all predictable (compared with sequential access, which is entirely predictable). So this is the point—direct (I still like that word) access to whatever data we want, in any sequence.

Why is this important? Suppose you are using an inventory system. You have a transaction for product 539. Your last transaction was for product 762. What must you do to retrieve, update, and rewrite the record for product 539? If your inventory file is an ordinary sequential file, you must start at the beginning of the file, read all the records up to product 539, and rewrite each to a new file. Impossibly slow, yet it gets worse: After you do your thing with product 539, you either have to finish copying the rest of the records to the new files or postpone that, in hope that the next transaction will be for a product after 539 so we can save a trip through the whole file. What we clearly need is the ability to go directly to record 539, read it, and write the updated record back in the same place. Random-access files permit you to do just that, and the savings in time are what make a data-

Copyright © 1983 Emerald Valley Publishing Co.

based system feasible—not only for inventory, but for accounts payable or receivable, general ledger, etc.

Implementation in TI BASIC

In a random-access file, in TI BASIC and in every other system I know, all records must be the same length. The operating system knows the length of each record, knows where the file begins on disk, and therefore can calculate the exact location of the 367th record, or any other record. This calculation is used whenever we ask to read or write a particular record.

Let's look at the statements we use on random-access files. They are the same statements we use on ordinary (sequential) files, but some parameters are different. First, when we OPEN a random-access file, we must declare:

- file organization is RELATIVE
- file type is DISPLAY or INTERNAL
- open mode is INPUT, OUTPUT, or UPDATE
- record type is FIXED

Don't ask why the word RELATIVE is chosen to specify random access, but it may have something to do with the address calculation: The location of each record is computed relative to the beginning of the file. You may well want to construct your random-access files as INTERNAL, to save space and time required for converting DISPLAY (ASCII) files for internal use. An INTERNAL file cannot be listed directly, but you probably need a program to list a random-access file anyway. An *open mode* of UPDATE allows you to read and write records in your file, and this is what you want most of the time. UPDATE is also the default if you don't specify an open mode. As well as specifying FIXED record type, you may also specify the record length, and I recommend that you do. As an example,

```
OPEN #1:"DSK1.INVENTORY",RELATIVE,
  INTERNAL, UPDATE, FIXED 92.
```

opens the INVENTORY file on your DSK1 as your #1 file; the file has 92-byte records in internal format, for random-access reads and writes. When you first create a file, you can and should specify the number of records to be allocated initially; the number follows the word RELATIVE. For example, the program that first established this file could have used:

```
OPEN #2: "DSK1.INVENTORY",RELATIVE
  150,INTERNAL,OUTPUT,FIXED 92
```

To read a particular record, include the record number (the first record is numbered zero) in the INPUT statement; if N = 119, for example,

```
INPUT #2,REC N: PN,D$,Q,PR
```

reads the 119th record from the file into the variables PN, D$, Q, PR. The PRINT statement similarly includes the word REC and the record number of the record to be written:

```
PRINT #2,REC N: PN,D$,Q,PR
```

You can use the EOF function for a random-access file, but this is not the best way. Better is to use the record 0 to hold special information about the file, especially the length of the file. As soon as you open the file, read that record:

```
INPUT #2, REC 0: FL
```

Then, before accessing any record, compare its record number with FL:

```
230 IF N>FL THEN 260
240 INPUT #2,REC N: PN,D$,Q,PR
250 GOTO 280
260 PRINT "INVALID RECORD NUMBER.
    REENTER"
```

When we wish, we are allowed to read or write records sequentially in random-access file. And of course we should CLOSE a file at the end of the program.

Which Record Contains What?

Okay, so you can easily get the 119th record in your random-access file. But how do you know that the information you want is in the 119th record? That is the hard part. If you are willing to assign product numbers 1 to 200 to the 200 items in your inventory file, you have no problem. At least, not until you discontinue some products and add others. In many cases, you can't assign the key to your file (product number, social security number, account number, or whatever) like this at all. So we need some scheme that associates a record number with each of your keys.

There are a lot of ways to do this. I will show one here: an index, which I keep in a file of its own. Actually, it could be kept in the first several records of your random-access file if you wish. Let's suppose an inventory system with up to 200 products. The product numbers are already assigned, as integers like 17, 29, 83, 104, 105, etc. We can keep our index in a pair of arrays in main storage while we run our system: these arrays don't take a lot of room.

```
 60 DIM IPN(200),ILOC(200)
 70 OPEN #1: "DSK1.INVINDEX",SEQUENTIAL
    INTERNAL
      :
      :
180 FOR I=1 to 200
190 INPUT #1: IPN(I),ILOC(I)
200 NEXT I
```

The IPN array holds the product numbers, and the ILOC array the record numbers in the random-access file for the corresponding products. When we want to access a product, we search the IPN array, find the record number, then use it to access the product record directly.

Using this scheme, the sequence of records in the random-access file matters very little. The sequence in the index file (and therefore in the arrays) matters more. The easiest thing, but least efficient, is to search the IPN array sequentially, with the product numbers in either ascending sequence or no particular sequence. One better idea is to put the most frequently used records at the front of the index file, thus cutting down on the average number of index entries your program must search. Studies have shown that in situations like this, 80 percent of the desired accesses are to 20 percent of the items. A still more efficient (but longer to program) method is a binary search, requiring that the index be in ascending sequence by product number. But let's come back to that idea another time.

Copyright © 1983 Emerald Valley Publishing Co.

Putting It all Together

Let's see how some of this works. We will see, at least in outline, how to (1) update a particular record, using the index, and (2) how to add a new record to the file (and of course to the index). First, let's be a little more precise about how we keep information, again using an inventory system as the context.

1. The RELATIVE file is named INVENTORY; its first record (numbered 0) contains the allocated length of the file; the number of records actually used must not exceed that number. If the allocated length is 201 records, for example, we might at some time be using 160, and these would be numbered 1 to 160.

2. The index file is named INVINDEX; it contains an index entry for each of the allocated records in INVENTORY. The index entries are in sequence by product number. The unused records are identified in the index by a product number like 32767, which is larger than any actual product number. In addition at the very beginning of the INVINDEX file are:

(a) the number of allocated records
(b) the number of currently active records

As part of our program initialization, we must open the files and read the index into our arrays:

```
 60 DIM IPN(200),ILOC(200)
 70 OPEN #1: "DSK1.INVINDEX",SEQUENTIAL,
    INTERNAL
 80 OPEN #2: "DSK1.INVENTORY",RELATIVE,
    INTERNAL,UPDATE,FIXED 92
 90 INPUT #1: NALLOC,NACTV
100 FOR I=1 TO NALLOC
110 INPUT #1: IPN(I),ILOC(I)
120 NEXT I
```

Now, suppose the program has accepted a product number APN, and needs to retrieve the INVENTORY record for that product; we will show for simplicity a sequential, rather than a binary search through the index file:

```
310 FOR J=1 TO NACTV
320 IF APN=IPN(J) THEN 370
330 IF ANP>IPN(J) THEN 350
340 NEXT J
350 PRINT "PRODUCT NOT ON FILE"
360 GOTO . .
370 INPUT #2,REC ILOC(J): PN,D$,Q,PR,. . .
```

If the program goes on to update some fields of the record, the record can be rewritten with its updated contents very simply:

```
470 PRINT #2,REC ILOC(J): PN,D$,Q,PR,. . .
```

Inserting a new record for a new product number is a little tricky. Where to put it in the inventory file is no problem; it can go right after the last active record. The index will make it accessible at the right time, with no problem. But we have more to do with the index. We must insert a new index entry in its proper place in sequence. Let's look at that process. Suppose that the product number to be inserted is PN, and that we have ascertained that such a number is not in the file.

```
600 IN NACTV<NALLOC THEN 630
610 PRINT "NO MORE SPACE IN THE
    INVENTORY FILE"
620 GOTO . .
630 NACTV=NACTV+1
640 PRINT #2,REC NACTV: PN,D$,Q,PR
650 REM ADJUST INDEX
660 FOR J=1 TO NACTV
670 IF PN>IPN(J) THEN 690
680 NEXT J
690 FOR K=NACTV TO J+1 STEP -1
700 IPN(K)=IPN(K-1)
710 ILOC(K)=ILOC(K-1)
720 NEXT K
730 IPN(J)=PN
740 ILOC(J)=NACTV
750 REM REWRITE THE UPDATED INDEX FILE
760 RESTORE #1
770 PRINT #1: NALLOC,NACTV
780 FOR I=1 TO NALLOC
790 PRINT #1: IPN(I),ILOC(I)
800 NEXT I
```

None of these operations takes very long. We always have the index file, the index arrays, and the random-access file itself in sync.

Do you have a better scheme? You may very well have, especially for your particular application. There is a lot of room for different ways of using and managing random-access files. After all, what we have is really the capability of managing large arrays—kept on disk instead of main storage. I hope you can see the importance of, and get some idea of how to use, random-access files from this introduction.

Copyright © 1983 Emerald Valley Publishing Co.

Getting DOWN to Business

DIVIDE AND CONQUER

In an earlier section we discussed random access files, and explored some details of using them. This section will review a few of the main points, develop a further idea or two, and then show a full example program using random access files.

A random access file is essentially a big array stored on disk. We can treat it much as we would treat an array; we access an element of an array by using a subscript, whose job is to select one particular element of the array:

```
350 K = A(SUBS)
360 B(SUBS) = L
```

These are ordinary BASIC statements that retrieve a value from the A array and store one in the B array. Similarly, we can specify exactly which record we want to read from or write to a random access file:

```
440 SUBS = 37
450 INPUT #1,REC SUBS: PN,D$,Q,R
460 PRINT #2,REC SUBS: L$,AVE,RET
```

These statements read the 37th record from the #1 file and write a record into the 37th record position of the #2 file. So SUBS is used just like a subscript to select which record to read or write.

The Index to the Random-Access File

In random-access files, the problem is knowing which record should be stored (or found) in which location. In my last section, I described a small inventory system in which the key to each record was the product number, an integer. There are at most 200 product numbers, but they are *not* just the numbers 1 to 200. My storage scheme stores product records arbitrarily, in the random-access file, but includes an index file also. The index file contains a pair of numbers for each record: the product number and the record number (the subscript in the random-access file where the record for the product number is stored). For example, Figure 3-a shows the index and the file after four entries have been made to the file. The records were inserted for products 67, 105, 29, and 84, and the records are stored in the random-access file in the sequence in which the records were created. The function of the index is to keep a list of the product numbers and the position of each record in the random-access file.

Early in any program that uses the random-access file, I read the index file into a pair of arrays, IPN and ILOC. When I then want to access a product record, I can search the IPN array at electronic speeds for the desired product record, and go directly to the product record.

Binary Search

In my last column, I showed a sequential search of this table, and hinted that a binary search would be faster. Let's take a look at a binary search: It is not very complex, is really a time saver, and can be applied in many table-search situations.

First, let's be clear about the context. We have an array, whose elements may be numbers or strings. Let's use numbers in our example, but strings can work exactly the same way. The elements in the table must be arranged in ascending sequence. We also have a number, called the *search key*, that we want to find in the table. So the objective in the search is to find the subscript for which the table element matches the search key. If no match for the search key exists in the table, we say the search fails.

The idea is a divide-and-conquer strategy. At all times, we keep track of the lower bound and the upper bound of the possible subscript in the table. At the start of the search, these bounds are the beginning and end of the table, of course. The central idea is that each time through the search loop, we compare the search key with the table element *half-way between the upper and lower bounds*. If that element matches the search key, the search is finished successfully. Otherwise, if the search key is less than this middle element, the desired table element must be in the lower half of the currently remaining portion of the table. So, we bring the upper bound down to the entry just below the middle one. The final case is the one in which the search key is greater than then middle element. So, the desired element must be in the upper half, and we bring the *lower* bound up to the one just above the middle. We repeat this process; each time through, we reduce the remaining possible portion of the table by half. The search ends either when we have found our table element or when we find the lower bound is greater than the upper bound; this last condition shows that the search has failed.

Let's look at an example. Figure 1 shows a table of twelve numbers (product numbers?) in ascending sequence. Suppose we are searching for 135 in the table. Our search starts with a lower bound of 1 and an upper bound of 12. Our first iteration computes a middle of (1 + 12)/2 = 6 (rounded down). The 6th table entry is 119; the search key is larger so the lower bound is set to 7, and we have eliminated the entire lower half of the table from further consideration. In our second iteration (7 + 12)/2 = 9, and we compare the search key with the 9th entry, 158. This time, the search key is less, so the *upper* bound becomes 8. The third iteration tests the 7th element itself and sets the lower limit to 8. The fourth

Copyright © 1983 Emerald Valley Publishing Co.

Figure 1 A table of numbers in ascending sequence.

17
29
83
104
105
119
130
135
158
183
197
262

iteration finds that the 8th element matches the search key. Suppose our search key were instead 139; the iterations would be exactly the same, except that in the fourth iteration, lower and upper bounds are both 8, and we find that the search key is larger than the 8th table element. Thus the lower bound exceeds the upper bound, and the search fails.

Figure 2 A subroutine for a binary search.

```
1000 LOWB = 1
1010 REM 1000. SUBROUTINE TO BINARY SEARCH THE TABLE IPN.
1020 REM OF LENGTH NACTV, FOR SEARCH KEY APN.
1030 REM RETURNS ISUB = SUBSCRIPT OF MATCHING ENTRY,
1040 REM OR 0 IF THE SEARCH FAILS
1050 UPB=NACTV
1060 IF UPB < LOWB THEN 1140
1070 ISUB = INT ((LOWB + UPB)/2)
1080 IF APN = IPN (ISUB) THEN 1150
1090 IF APN < IPN (ISUB) THEN 1120
1100 LOWB = ISUB + 1
1110 GO TO 1060
1120 UPB = ISUB - 1
1130 GO TO 1060
1140 ISUB = 0
1150 RETURN
```

Figure 2 shows a subroutine that searches a table named IPN for a search key named APN. The length of the table in NACTV. The subroutine returns the value of the subscript ISUB of the matching element in the table, or returns ISUB = 0 if the search fails. Trace the subroutine, using the table shown in Fig. 1, to verify that the subroutine follows the logic described above.

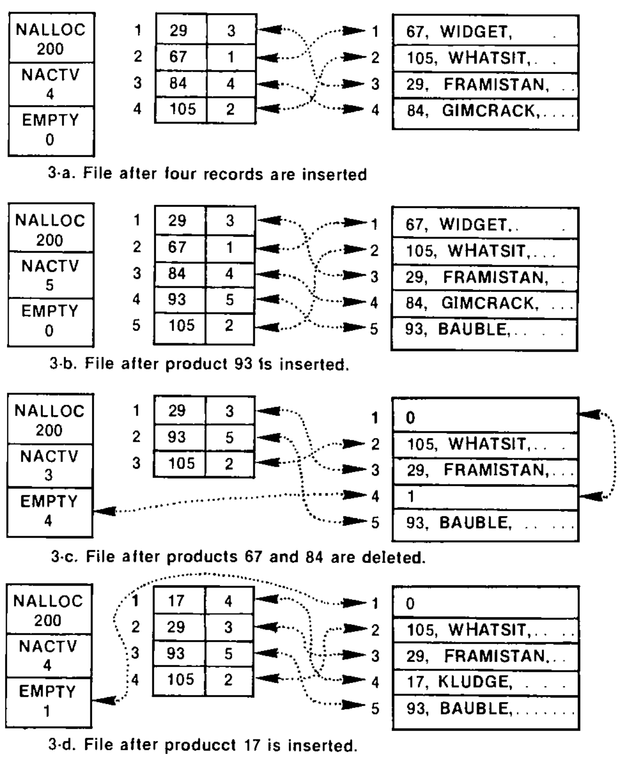
Inserting and Deleting Records

If we are only inserting records into a random-access file, there isn't much problem. We use the next record position in the random-access file, and adjust the index by moving the entries up to make room, in order to put the new product number where it should go in sequence. We showed this process in the earlier section. The problem is more complex if we also delete entries from the file from time to time. And, of course, we want to be able to reuse vacated file positions.

To keep track of these positions, we use what the computer scientists call a *linked list*. We keep a variable, called EMPTY, which gives the first available file position; if there are no deleted record positions currently available (they may all have been used up again, or maybe no record have been deleted at all), EMPTY equals zero. Suppose the record stored in the 11th position of the random-access file has been deleted. Then EMPTY = 11. And the 11th record *now* contains the *next available* file position, which might be 37. Then the 37th record contains the next available file position, etc. The last available file record contains a zero to mark the end of the list.

When deleting a record, we store in that record position the current value of EMPTY, and record the file position of this newly deleted record as the new value of EMPTY. When we need to insert a new record in the file, we look at EMPTY first, to see whether there are any previously deleted records whose positions we can recycle. If so, we use the first one in the list, but first read from it a new value to place in EMPTY, so the second available file position is now first. If EMPTY = 0, we are using all file position up to NACTV. In that case we just use the next file position after NACTV, unless NACTV = NALLOC, in which case we've run out of space.

Figure 3



You can see the process in Fig 3. First, Fig. 3-c shows the result of deleting first product 67 and then product 84 from the file. The list of empty file positions starts at position 4 (as EMPTY tells us), and then continues to position 1. The zero, stored at position 1, indicates that there are no more empty records. The index table records only the currently active products, and NACTV tells how many there are.

When product 17 is inserted (Fig. 3-d), it is put into the first available empty position, which is position 4. The list of empty positions is reduced, and the index, of course, is expanded. If two more products are inserted, the first will go into file position 1, but then the EMPTY list will itself be empty, so the next product will be put in position 6.

Copyright © 1983 Emerald Valley Publishing Co.