PROGRAMS for BEGINNERS on the TRS-80™

FRED BLECHMAN

extensive explanations, line-by-line descriptions of programming techniques for both Model I (level I and level II) and Model III illustrating how programs work

Unlock the mysteries of BASIC!

Many useful, simple programs . . . no printers, disks, or fancy interfaces needed . . . (also, useful Appendices containing helpful aids for beginners on the TRS-80™)

HAYDEN

A cassette containing the programs in this book is available from the publishers for $10.95. Please address orders to Hayden Book Company, Inc., 50 Essex Street, Rochelle Park, NJ 07662.

PROGRAMS FOR BEGINNERS ON THE TRS-80™

PROGRAMS FOR BEGINNERS ON THE TRS-80™

FRED BLECHMAN

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

To my patient and understanding wife, Ev, who—although she doesn't know a bit from a byte—allowed me to spend countless hours in my "cave" with my "mistress" (the TRS-80).

Library of Congress Cataloging in Publication Data

Blechman, Fred.
Programs for beginners on the TRS-80™.

Includes index.
1. TRS-80 (Computer)—Programming. I. Title.
QA76.8.T18B57 001.64'2 80-27365
ISBN 0-8104-5182-4

Printed in the United States of America

1 2 3 4 5 6 7 8 9 PRINTING

81 82 83 84 85 86 87 88 89 YEAR

PREFACE

Professional programmers might laugh at this book for being "too simple." But, if you're a *beginner* in personal computer programming, this book is for you! You'll learn step-by-step how 21 sample TRS-80 programs "work." Various program techniques are described line-by-line within the programs, and the unique Matri-Dex™ matrix index will help you find other programs using the same BASIC commands and statements. If you have a computer *other* than the TRS-80, the BASIC language you use is probably very similar so that you can still learn a lot. A slight modification of the sample programs given here will make them suitable for running on your computer.

For the Programs section of this book, I am going to assume that you've had *no* past experience in either computers or programming but that you have a TRS-80 and the Level I Basic manual (and the Level II manual as well if you have a Level II machine). You see, it is not my purpose to teach you what the BASIC language *is* but to show you how to *use* the essentials of the BASIC language. If BASIC were a foreign language, this might be considered as a course in *conversational* BASIC, not in BASIC vocabulary, syntax, and grammar.

While some of the sample programs are relatively simple, others are more challenging; all of them illustrate various programming methods. With a little imagination, you'll be able to apply the examples in this book to generate programs fitting your specific needs.

Most of these programs are designed to run in 4K RAM in either Level I or Level II. Therefore, most Level I abbreviations are not used (since they are not available in Level II). Also, many math functions directly available in Level II are not used. Since these are pointed out in the discussion of each program, however, you can substitute Level II language where appropriate. In addition, *none* of the programs in this book *requires* the use of a printer, disk, or expansion interface. Some programs have printer options if you *do* have a printer. All sample programs are short enough for you to enter them via the keyboard.

If you're a *non*beginner, you might find Appendices B, C, and D particularly valuable. Appendix A is a video display worksheet that may be used as a reference for many of the programs. Appendix B covers various cassette loading aids and includes a complete schematic and description of a simple cassette audio/visual control monitor you can build from a kit. Appendix C will save you lots of head-scratching when converting Level I programs to RUN in Level II. Appendix D has six extremely handy, short, simple programs that will make a world of difference in using and programming your TRS-80, especially if you have a printer. Appendix E (Matri-Dex™) provides you with a quick way to determine which programs contain operations or statements you want to know more about.

To help you understand the programs, photos show how they will appear on your screen. The introduction to each program tells you what new things you'll learn. A description explains the purpose and use of each program. A complete listing (and usually a run) is followed by a line-by-line explanation of any special programming techniques not covered in detail in previous programs. A list of variables and suggested modifications are also included for each program.

This book is not intended to be an exhaustive treatise on all the TRS-80 will do. Its BASIC vocabulary includes only the most common and easy-to-use commands and statements. For more elaborate or sophisticated techniques, you should consult any of various books that go into BASIC programming in great detail.

Once more, let me make the point that this book is aimed at the beginner, *not* the professional programmer; thus it has the advantage that beginners will be able to *understand* it and *use* it right away!

FRED BLECHMAN

CONTENTS

PROGRAMS FOR BEGINNERS ON THE TRS-80™

PROGRAM 1
DISPLAY ALIGNMENT
(4K, Level I or II)

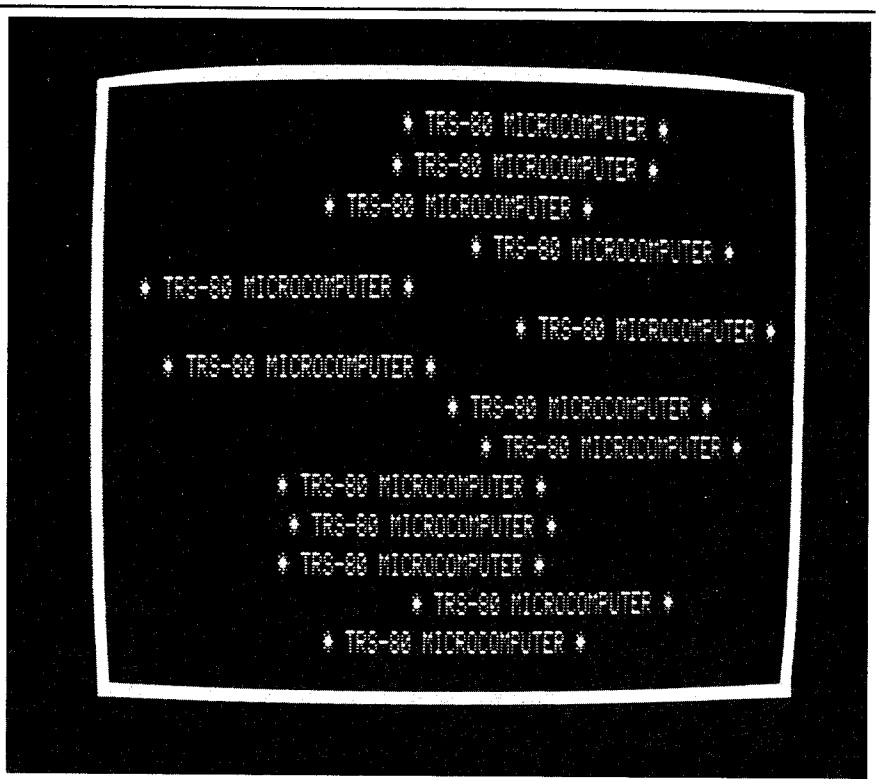
You'll learn: Loops; simple graphics; random numbers; print formatting and blanking

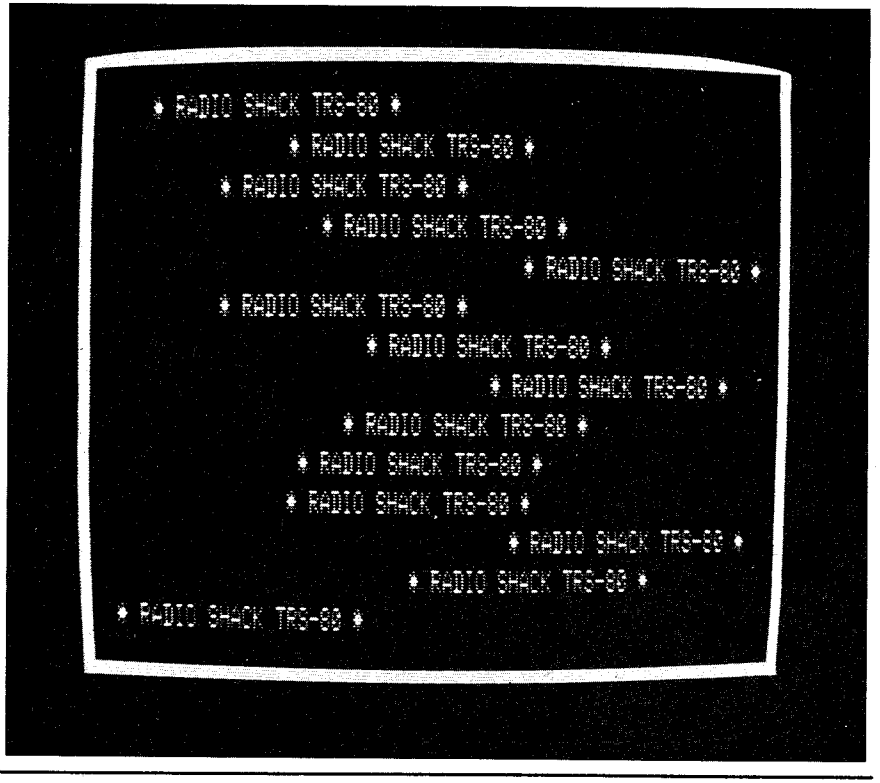
Description: If you have a TRS-80 Video Display to use with your TRS-80 Microcomputer System, you may have noticed that the top or bottom line is partially cut off by the display bezel. This program will not only help you properly position the display on the screen, it also contains several little programming tricks that you might want to add to programs of your own.

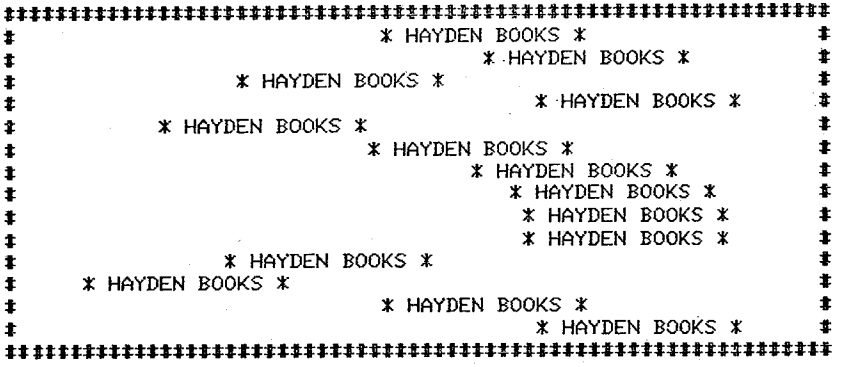
Explanation: The program first draws a border at the graphic limits, using SET commands. Then a short message is printed 14 times within the border. The message can have as many as 61 characters and spaces, but the shorter the better. Each line is printed in a random position from the left border but does not break the right border. After the 14 lines are printed—each line with the same "message"—the area inside the border is cleared, and the randomly positioned 14 lines are printed again. All this happens five times; then the entire screen is cleared and the sequence repeated.

Before we explain how the program does all this, type the program as listed here into your TRS-80. Be sure to observe every punctuation mark carefully. For lines 117Ø and 118Ø, use exactly 30 spaces between the quotation marks. Note that PRINT@ is PRINTAT in Level I.

Once the program is entered, type RUN and enter. First you'll see the top and bottom borders being drawn, then the left and right side borders. If you are using Level I, watch carefully and you'll see the upper left corner disappear just as the border is completed and then reappear after the first line of printing (this has to do with cursor control in Level I only). Let the program run a few sequences; it's fascinating to watch, since it keeps changing. If it doesn't run properly, and you have entered via the keyboard, you could easily have made a punctuation error. The omission of a semicolon, for example, can change the entire display! To stop the


♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦
♦ TRS-80 MICROCOMPUTER ♦


♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦
♦ RADIO SHACK TRS-80 ♦



program, hit BREAK. Changing the message line is easy, and we'll discuss that operation later.

Now let's put the border of this program to use. BREAK the program after the border has been drawn. Are all four corners

LEVEL I/II LISTING

```
1000  REM * COPYRIGHT FRED BLECHMAN 1978 *
1001 REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
1002 REM * TRS-80 LEVEL I OR LEVEL II 4K *
1003 REM * GRAPHIC ALIGNMENT PROGRAM *
1005  CLS
1009 REM * DRAW THE BORDER AT THE GRAPHIC LIMITS *
1010  FOR X=0 TO 127
1020  SET (X,0):SET(X,47)
1030  NEXT X
1040  FOR Y=0 TO 47
1050  SET(0,Y):SET(1,Y):SET(126,Y):SET(127,Y)
1060  NEXT Y
1069 REM * COUNT THE FRAMES *
1070  FOR X=1 TO 5
1079 REM * ESTABLISH PRINT FORMAT *
1080  FOR P=66 TO 898 STEP 64
1089 REM * REFILL UPPER LEFT CORNER *
1090  SET(0,0):SET(1,0)
1100  SET(0,1):SET(1,1)
1110  SET(0,2):SET(1,2)
1119 REM * PRINT MESSAGE AT RANDOM LINE POSITIONS *
1120 R=RND(37)
1130 PRINT@ P,TAB(R);"* TRS-80 MICROCOMPUTER *";
1139 REM * NEXT LINE CONTROLS PRINTING INTERVAL *
1140  FOR D=1 TO 100:NEXT D
1149 REM * ADVANCE PRINT FORMAT COUNTER *
1150  NEXT P
1159 REM * CLEAR AREA INSIDE THE BORDER *
1160  FOR P=66 TO 898 STEP 64
1170  PRINT@ P,"                                    ";
1180  PRINT@ P+30,"                                    ";
1190  NEXT P
1199 REM * ADVANCE FRAME COUNTER *
1200  NEXT X
1209 REM * RERUN *
1210  GOTO 1005
```

visible, or have they been cut off from view by the TRS-80 Video Display front bezel? If you can see all four corners, then your display is properly positioned. If not, carefully remove the back of the display (it's held on with five screws) after unplugging it. Leave the microcomputer on, however, so as not to lose the program. After you remove the display back, look at the back end of the picture tube and you'll find two metal tabs sticking out radially. These are the centering tabs, of which one primarily effects the horizontal positioning of the display, and the other the vertical positioning. Move them slightly until the display is positioned so that all four corners of the border appear on the screen. Of course, since you'll have to plug in the Video Display to do this, be sure you keep away from everything but the centering tabs.

Although you could adjust two potentiometers in the microcomputer to achieve the same effect, this method of centering the display is easy and does not involve opening the sealed microcomputer case (thus voiding the warranty).

Since the program itself may puzzle you, let's go through it more or less line by line. Lines 1ØØØ thru 1ØØ3 are REMARKS that don't affect the program operation. Since there's plenty of memory available for this short program, lots of REM statements have been used to help guide you through the program. Refer to Appendix A for the Video Screen layout system.

Line 1ØØ5 clears the screen. Line 1Ø1Ø starts a FOR-NEXT loop to "light" spots on the screen defined by line 1Ø2Ø. The TRS-80 has 128 vertical "columns" and 48 horizontal "rows" that can be addressed with a SET (column, row) command. Line 1Ø1Ø establishes X = Ø for the first "loop." Line 1Ø2Ø therefore starts with position Ø,Ø (the upper left corner) and Ø,47 (the lower left corner), lighting the screen at these locations. Now line 1Ø3Ø increases the value of X by 1 (since it was not directed in line 1Ø1Ø to step by any other value) so that the next SET command from line 1Ø2Ø is 1,Ø and 1,47. This process continues drawing horizontal borders across the top and bottom of the screen, one column at a time, until the maximum X of 127 established by line 1Ø1Ø is reached. Now the program proceeds to lines 1Ø4Ø, 1Ø5Ø, and 1Ø6Ø, which draw the left and right screen borders in a similar fashion by setting columns Ø, 1, 126, and 127 from row Ø at the top to row 47 at the bottom of the screen. (Since the columns are narrower than the rows, we SET two columns on each side for an equal-width border all around).

Line 1Ø7Ø starts counting another FOR-NEXT loop that repeats five times on the command of line 12ØØ. All lines between 1Ø7Ø and 12ØØ are "nested" within that primary loop. Line 1Ø8Ø tells

the computer that you want to let the variable P start at the value of 66 and advance, when told to, in steps of 64 to the maximum value of 898. Line 115Ø tells P when to advance. You'll see the significance of this in a moment.

Lines 1Ø9Ø, 11ØØ and 111Ø can be deleted in Level II; in Level I, however, they "patch" the upper left corner of the border, which gets blanked by a built-in Level I command that returns the cursor to that location for an instant at the first PRINT command.

The print formatting is controlled by lines 112Ø and 113Ø. Line 112Ø selects a random number from 1 to 37, and line 113Ø tells the computer to print the statement enclosed in quotes (in this example, * TRS-80 MICROCOMPUTER *) at a particular location on the screen. The TRS-80 screen is divided into 1024 printing locations, numbered Ø to 1023. This figure represents 16 lines of 64 locations each. (See the TRS-80 Video Display Worksheet in your Level I or Level II manual). Line 113Ø instructs the computer to print at location P (which is 66 the first time around) and then to indent this position ("TAB") by an additional R spaces. Since R is a random number up to 37 (from line 112Ø), this means that the printed message will be indented up to 37 spaces from position P. The semicolon at the end of line 113Ø keeps the right border from being deleted at the end of that line.

Line 114Ø is a timing-delay FOR-NEXT loop. You can speed up the printing by eliminating this line altogether. To increase the time delay between printed lines, increase the top value of D. For Level I, a top value of 500 will give about a one-second delay as the computer counts to itself. For Level II, use a value of about 400 for a one-second delay. Incidentally, these counts vary with what the computer is doing at the time, as well as the temperature. As you'll see in a later program (Program 8), however, this kind of a loop can actually be used to keep track of real time with fair accuracy.

Line 115Ø now increases the value of P by 64, as instructed by line 1Ø8Ø, and the sequence repeats until P equals 898 (14 message lines). Then the program falls through to lines 116Ø through 119Ø, which clear the screen inside the border.

You may wonder why the blanking is done with two lines, 117Ø and 118Ø, rather than just one line. In Level I, there is not enough room on a program line for the 60 needed spaces; that's why both lines 117Ø and 118Ø have exactly 30 spaces between the quotation marks. In Level II, which allows program lines up to 255 characters in length, a single line with 60 spaces between quotation marks is possible.

Line 12ØØ sends the program back to line 1Ø7Ø for four more printing sequences. At the end of the fifth sequence, the program falls through to line 121Ø, which sends it back to line 1ØØ5. The screen clears and the whole program runs again.

Modifications: Whether you understand the inner workings of the program or not, it's quite easy to customize the message. You'll probably, for example, want to have the display print out your name in random fashion. The only lines that will have to be changed are lines 112Ø and 113Ø. First change line 113Ø by putting what you want printed between the quotation marks. Be very careful not to change anything else on that line; every punctuation mark has a purpose! Now add the number of characters *and* spaces between the quotation marks. Subtract this number from 61, and the result is the random number you need to put in line 112Ø. In other words, your message and the maximum random number must add up to exactly 61. (Actually, you can go to 62, but a space has intentionally been left at the left and right sides so that the printing will not come too close to the border). If you exceed 62, you'll find occasional chunks taken out of the right border!

If you'd like to program the message very easily (limited to 16 characters and spaces in Level I; 50 characters and spaces in Level II), and or change the following lines:

```
 999  CLS
1ØØØ  INPUT"TYPE IN THE DESIRED MESSAGE AND
      ENTER. . .";A$
1ØØ1  PRINT:INPUT"HOW MANY LETTERS AND SPACES
      IN THE MESSAGE";A
1ØØ4  B=61-A
112Ø  R=RND(B)
113Ø  PRINT@P,TAB(R);A$;
```

(Note: Remember that PRINT@ in Level II becomes PRINTAT in Level I.)

Variables:

X—horizontal position counter
X—frame counter
Y—vertical position counter
P—printing location
P—print blanking location
R—random number from 1 to 37
D—delay time counter

Note: X and P are each used twice in this program for different purposes. For example, X is first used with the SET command, then used to count the number of times the screen is filled within the border. The same variable may be used any number of times within a program for different purposes, but for only one purpose at a time. This fact may be important for you to realize, especially in Level I where nonarray variables are limited to 26 (A–Z)

PROGRAM 2
COMPUTER INTERVIEW
(4K, Level I or II)

You'll learn: Screen prompts; strings; branching; variations in Level I/II use.

Description: The computer asks for a lady from the observing group. When she hits ENTER, the computer asks her name, which she types in and enters. Then the computer asks her five questions that she answers with a Y for yes or an N for no. The last question is a request for a date!

Explanation: This program, simple as it is, will not run the same way in Level I and Level II. Let's first discuss how it works in Level I, and then we'll see what changes are needed for Level II.

Lines 4, 5, and 6 format the printing of the computer "introducing" itself and asking for a lady from the audience. While P.AT statements are used here, you could use regular PRINT (or P.) statements with blanks to space the text toward the middle of the screen and P. to space lines. If you have a printer, you will have to do it that way, or with TAB instructions, since most printers ignore the PRINTAT command (even in Level II). They just print at the beginning of the next line if you use the PRINTAT command (PRINT@ in Level II). However, the TRS-80 makes print locations easy for *screen* use. This was mentioned briefly in Program 1 and is discussed in detail in Chap. 22 of your Level I Manual if you need help.

Line 7 is just a short time-delay loop to allow you to read and understand the screen before line 8 clears the screen (C.) and then prints I'M WAITING. . . . . in the center of the screen. Line 9 skips three display lines (P.:P.:P.:) and then uses an input prompt (I.) followed by a semicolon and B$. The input statement automatically does two things: It puts a question mark on the screen and brings the program to a halt to wait for a keyboard input. Since there's text and a semicolon after the text, the question mark appears right after the text. If you leave out the text, the question mark will

LEVEL I LISTING

```
1  REM * COPYRIGHT 1978 FRED BLECHMAN *
2  REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
3  C.
4  P.AT 155;"HELL0";AT270;"I'M YOUR FRIENDLY TRS-80 COMPUTER"
5  P.AT460;"IS THERE SOME LADY OUT THERE THAT WILL"
6  P.AT525;"ANSWER A FEW SIMPLE QUESTIONS???"
7  FOR X=1 TO 2500:N.X
8  C.:P.AT468;"I'M WAITING....."
9  P.:P.:P.:I."              (PRESS ENTER WHEN READY)";B$
10  Y = 1
20  N = 0
21  C.
25  I."WHAT IS YOUR NAME, MY DEAR";A$
26  P.:P.
30  P.A$;:I.", ARE YOU OVER 18 YEARS OLD";A
40  IF A=1 T.60
50  P."EGADS, ";A$;", YOU'RE JAIL-BAIT!"
55  P.:G.70
60  P."HEY, THAT'S SWELL, ";A$;", YOU'RE AN ADULT."
65  P.
70  I."ARE YOU SHORTER THAN 5 FEET 10 INCHES";B
80  IF B=1 T.100
90  P."WOW, ";A$;", YOU'RE A GIANT!"
95  P.:G.110
100  P."THAT'S NICE, ";A$;", YOU'RE JUST THE RIGHT SIZE."
105  P.
110  P.A$;:I.", DO YOU HAVE BLUE EYES";C
120  IF C=1 T.140
130  P."TOO BAD....I REALLY HAVE A THING FOR BLUE EYES."
135  P.:G.150
140  P."THAT'S GREAT, ";A$;", SINCE BLUE EYES TURN ME ON!"
145  P.
150  P."HOW ABOUT YOUR HAIR, ";A$;:I."? ARE YOU A BLONDE";D
160  IF D=1 T.180
170  P."AW, HECK! I REALLY DIG BLONDES....."
175  P.:G.185
180  P."THAT'S TERRIFIC! I'VE GOT TO GET TO KNOW YOU BETTER."
185  FOR X=1 TO 1500:N.X
186  CLS:P.
190  P."ARE YOU BUSY TONIGHT, ";A$;:I.E
200  IF E=1 T.220
210  P."I'LL PICK YOU UP AT EIGHT 0'CLOCK SHARP! JUST YOU AND"
211  P."                ME, BABY....."
215  G.230
220  P."OKAY, ";A$;", IF THAT'S THE WAY YOU WANT TO BE! BUT,"
221  P."REMEMBER, UNDER THIS SLICK GRAY EXTERIOR, I'VE GOT"
222  P."FEELINGS, TOO.....  MAYBE NEXT TIME.....  (SOB) ..."
230  P.:P.:P."(TYPE IN 'RUN' AND ENTER FOR ANOTHER INTERVIEW)
240  END
```

appear at the beginning of the next line. The semicolon is one of the most powerful formatting tools in BASIC. It simply means "Don't go to the next line until there's no more space left on this line!"

The B$ is a "string variable," and there are only two of these (A$ and B$) in Level I. It allows you to use numbers or letters (*alphanumerics*), such as names and addresses, rather than just numbers. Level I is restricted to 16 character strings; Level II, to

```
1  REM * COPYRIGHT 1978 FRED BLECHMAN *
2 REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
3  CLS
4 PRINT@155,"HELLO";:PRINT@270,"I'M YOUR FRIENDLY TRS-80 COMPUTER"
5 PRINT@460,"IS THERE SOME LADY OUT THERE THAT WILL"
6 PRINT@525,"ANSWER A FEW SIMPLE QUESTIONS???"
7  FOR X=1 TO 2500:NEXTX
8 CLS:PRINT@468,"I'M WAITING....."
9  PRINT:PRINT:PRINT:INPUT"                (PRESS ENTER WHEN READY)";B$
21  CLS
25  INPUT"WHAT IS YOUR NAME, MY DEAR";A$
26  PRINT:PRINT
30 PRINTA$;:INPUT", ARE YOU OVER 18 YEARS OLD";B$
40 IF B$="Y" GOTO 60
50  PRINT"EGADS, ";A$;", YOU'RE JAIL-BAIT!"
55  PRINT:GOTO70
60  PRINT"HEY, THAT'S SWELL, ";A$;", YOU'RE AN ADULT."
65  PRINT
70 INPUT"ARE YOU SHORTER THAN 5 FEET 10 INCHES";C$
80 IFC$="Y" GOTO 100
90  PRINT"WOW, ";A$;", YOU'RE A GIANT!"
95  PRINT:GOTO110
100  PRINT"THAT'S NICE, ";A$;", YOU'RE JUST THE RIGHT SIZE."
105  PRINT
110 PRINTA$;:INPUT", DO YOU HAVE BLUE EYES";D$
120 IF D$="Y" GOTO 140
130  PRINT"TOO BAD...I REALLY HAVE A THING FOR BLUE EYES."
135  PRINT:GOTO150
140  PRINT"THAT'S GREAT, ";A$;", SINCE BLUE EYES TURN ME ON!"
145  PRINT
150 PRINT"HOW ABOUT YOUR HAIR, ";A$;:INPUT"? ARE YOU A BLONDE";E$
160 IF E$="Y" GOTO 180
170  PRINT"AW, HECK! I REALLY DIG BLONDES....."
175 PRINT:GOTO 185
180  PRINT"THAT'S TERRIFIC! I'VE GOT TO GET TO KNOW YOU BETTER."
185  FOR X=1 TO 1500:NEXTX
186  CLS:PRINT
190 PRINT"ARE YOU BUSY TONIGHT, ";A$;:INPUTF$
200 IF F$="Y" GOTO 220
210  PRINT"I'LL PICK YOU UP AT EIGHT 0'CLOCK SHARP! JUST YOU AND"
211  PRINT"                ME, BABY....."
215  GOTO230
220  PRINT"OKAY, ";A$;", IF THAT'S THE WAY YOU WANT TO BE! BUT,"
221  PRINT"REMEMBER, UNDER THIS SLICK GRAY EXTERIOR, I'VE GOT"
222  PRINT"FEELINGS, TOO.....MAYBE NEXT TIME.....(SOB)..."
230  PRINT:PRINT:PRINT"(TYPE IN 'RUN' AND ENTER FOR ANOTHER INTERVIEW)
240  END
```

255. When the computer is expecting a string entry, all you have to do to continue the program is hit ENTER. This actually enters a blank for that string value, and the program proceeds. This is a very handy programming technique, not mentioned in the Level I Manual, for "paging" text or tables so that they won't scroll up off the screen until you press ENTER.

Lines 1Ø and 2Ø tell the computer that from here on (unless changed later) the variable Y will be equal to 1 and the variable N will be equal to Ø. In Level I, since the variables are not all set to Ø

PROGRAM 2 RUN

```
                    HELLO

          I'M YOUR FRIENDLY TRS-80 COMPUTER

     IS THERE SOME LADY OUT THERE THAT WILL
      ANSWER A FEW SIMPLE QUESTIONS???


            I'M WAITING.....


     (PRESS ENTER WHEN READY)?



WHAT IS YOUR NAME, MY DEAR? MARY


MARY, ARE YOU OVER 18 YEARS OLD? N
EGADS, MARY, YOU'RE JAIL-BAIT!

ARE YOU SHORTER THAN 5 FEET 10 INCHES? N
WOW, MARY, YOU'RE A GIANT!

MARY, DO YOU HAVE BLUE EYES? Y
THAT'S GREAT, MARY, SINCE BLUE EYES TURN ME ON!

HOW ABOUT YOUR HAIR, MARY? ARE YOU A BLONDE? N
AW, HECK! I REALLY DIG BLONDES.....



ARE YOU BUSY TONIGHT, MARY? Y
REMEMBER, UNDER THIS SLICK GRAY EXTERIOR, I'VE GOT
FEELINGS, TOO..... MAYBE NEXT TIME..... (SOB) ...


(TYPE IN 'RUN' AND ENTER FOR ANOTHER INTERVIEW)
```

when RUN, as they *are* in Level II, it's necessary to assign values in the program to all of the variables used—even those that are equal to zero.

Line 21 clears the screen. Line 25 asks the first question with an input statement and waits for the keyboard entry of A$, the lady's name. The lady "victim" (let's call her Mary) simply types in her name first and then ENTER. The computer now keeps her name in memory as A$ for the rest of the program.

Line 26 "linefeeds" twice, for spacing to the next question. Line 30 prints MARY and follows immediately on the same line (see

the semicolon?) with the input statement for the next question. The computer then waits for variable A from the keyboard.

Mary responds by pressing the Y or N key. In Level I, each key is "initialized" at computer turn-on with some random value (usually around plus or minus .5). You can verify this by turning on your Level I machine and typing PRINT A,B,C,D,E,F,G,H,I,J,K, L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z and then ENTER. You'll get four columns of values, with each figure representing the computer's memory value for that letter. (Read across and then down. C is the third value on the top row; F is the second column, second row; etc.

Since you assigned specific values for Y and N in lines 1Ø and 2Ø, they are no longer the computer turn-on values. If Mary presses Y, the computer sets variable A equal to 1; if she presses N, the computer sets variable A equal to Ø. Now line 4Ø looks at the value of A. If it is equal to 1, then the program jumps to line 6Ø and prints that computer reply. If A is *not* equal to 1, the program ignores line 4Ø and "falls through" to the next line, 5Ø, printing that comment instead.

Since IF-THEN and GOTO statements are so important for "branching" any program, let's follow this example through in detail. Suppose Mary types N and ENTER, setting the value of A to Ø. She is telling the computer that she's *under* 18 years old. The computer ignores line 4Ø, since A is *not* equal to 1, goes on to line 5Ø and prints EGADS, MARY, YOU'RE JAIL-BAIT!, and then moves to the next instruction. If, however, Mary answered the question with a Y, the computer sees A equal to 1 and obeys line 4Ø, jumping over the "wrong" response to line 6Ø, and the screen displays HEY, THAT'S SWELL, MARY, YOU'RE AN ADULT, and moves to the next instruction. Notice, incidentally, the spaces in the proper places in lines 5Ø and 6Ø so that the text doesn't run together. The quotation marks define the beginning and end of what's printed, and blank spaces must be included where needed for proper spacing.

Now that we have a computer response to her entry, what happens next? Go back to Mary entering an N. The next operable line is 5Ø, followed by 55. Line 55 spaces one line and then goes to line 7Ø (G. is the abbreviation for GOTO). Do you see how this bypasses the incorrect response, which appears in line 6Ø? Line 7Ø is the next question.

If, however, Mary answered with a Y, the computer obeys line 4Ø, jumps lines 5Ø and 55, prints the message in line 6Ø, followed by the line space directed by line 65, and then goes on to ask the question in line 70.

In either case, the program stops after asking the question in line 70 and waits for the next answer.

The remainder of the program proceeds in the same manner, with IF-THEN and GOTO statements routing the program around incorrect computer responses. A variation occurs in line 175, where the computer is told to go to a timing loop (line 185) instead of the next question. Why? By this time the screen is full, and the timing loop allows you to see the answer to the last question before the screen is cleared (line 186) and the next question is asked (line 190).

Variables A, B, C, and D are used for the answers to the five questions arbitrarily. Actually, any variable—even the same one each time—could be used for each question. However, false responses could be generated more easily using only one variable, since no "error-trapping" techniques—covered in a later program—are used here.

Level II is different from Level I in many ways affecting the programming. One difference is that Level II will *not* accept a letter entry for a simple variable (such as A, B, C, etc.). Letter entries (such as Y or N) will be accepted only by *string* variables in Level II. Look at the Level II version of this program and you'll see that A, B, C, etc., have been replaced with B$, C$, D$, etc. Also, the IF-THEN statements use string *comparisons* to recognize a Y or N entry. Note that the comparison under inspection must be enclosed in parentheses (such as line 40). Lines 10 and 20 of the Level I program are therefore no longer needed.

The other obvious difference in the Level I and Level II listings is that hardly any abbreviations are allowed in the Level II "language." Commands like PRINT and GOTO must by typed out in full. Also, let us remind you again of the following facts because they are so important: PRINTAT in Level I is PRINT@ in Level II, and the @ is *not* shifted with the shift key as on a regular typewriter. (If you *do* shift the @, it will appear to be normal on your screen, but the program will reject it with an SN—syntax—error message.) Also, very important is the requirement in Level II that PRINT@ be followed by a *comma;* Level I also allows a semicolon.

Modifications: Obviously, you can ask other questions and program other computer responses, limited only by your imagination and the memory of your computer. You could make the program a "compatibility test" by adding variables that count the Y and N answers and come up with a compatibility score. The basic idea here of interaction with the computer to access stored responses is the heart of most computer education programs.

Variables:

X—timing loop counter
A$—name
B$-F$—Y or N responses to questions
B$—ready for next screen

Note: B$ is first used for continuing the program, then for response to the question in line 3Ø. Since these two distinct functions occur at different points in the program, there is no conflict.

PROGRAM 3
FORTUNE TELLER
(4K, Level I or II)

You'll learn: Pseudo-random numbers; computerized questions and answers; ON-GOTO; automatic rerun

Description: After you type in your name and birth month, the computer prints a group of questions on the screen and asks you to choose one or type in your own question. When you do, the screen clears and responds with a two-line answer, after which it prints another question. You can keep on asking questions and getting (usually) different answers until you terminate the program by hitting the BREAK key.

Explanation: This program will run in either Level I or Level II; note, however, that PRINTAT is used in Level I rather than PRINT@.

Line 1Ø7 clears the screen, and lines 11Ø–18Ø provide the instructions. When you press ENTER, the screen clears (line 19Ø), and line 2ØØ asks your name. Type in your name (anything over 16 characters will be "truncated"—cut-off—in Level I). When you press ENTER again, line 2Ø5 asks for your birth-month number, such as 7 for July. Actually, this number provides a "seed" for advancing the random number generator, which is not really random at all!

When the computer is turned on, a series of numbers is held in the read-only-memory (ROM) for every random number limit you choose, but these numbers remain the *same* every time you turn on the computer.

What? You don't believe this? You think random numbers should *really* be random? Alas, they should be, but your TRS-80 uses a *pseudo*-random number technique. Here's how you can prove it to yourself. Turn off your computer (you can leave the display on); then turn it on again and get a READY. (For Level II, you'll have to answer "MEMORY SIZE?__" by pressing ENTER.) Now type and RUN the following program:

LEVEL I/II LISTING

```
100  REM * COPYRIGHT FRED BLECHMAN 1978 *
105  REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
106 REM * FOR TRS-80 LEVEL I OR LEVEL II 4K *
107 CLS
110  PRINT:PRINT"                FORTUNE TELLING PROGRAM"
120  PRINT:PRINT"     THE COMPUTER WILL LIST SEVERAL TYPICAL QUESTIONS"
130  PRINT"FOR YOU TO ASK THE 'FORTUNE TELLER'.  YOU MAY CHOOSE TO "
140 PRINT"ASK YOUR OWN QUESTION INSTEAD.  JUST ENTER THE APPROPRIATE"
150  PRINT"NUMBER.....(FOR ENTERTAINMENT PURPOSES ONLY). "
180  PRINT:INPUT"WHEN READY TO START, PRESS ENTER";A$
190  CLS
200  PRINT:INPUT"WHAT IS YOUR NAME";B$
205  INPUT"WHAT MONTH WERE YOU BORN? (MONTH NUMBER)";N
206  N=10*N
207  FORI=1TON:J=RND(32767):NEXTI
210  PRINT:PRINT"   ";B$;", CHOOSE ONE OF THE NUMBERS BELOW:"
220  PRINT:PRINT"(1) SHOULD I MAKE THAT TRIP I'M THINKING ABOUT?"
230  PRINT"(2) IS HE/SHE SERIOUS, OR JUST PLAYING THE FIELD?"
240  PRINT"(3) SHOULD I TELL HIM/HER THE WHOLE TRUTH?"
250  PRINT"(4) SHOULD I ASK MY BOSS FOR A RAISE?"
260  PRINT"(5) WILL I BE RICH AND FAMOUS SOME DAY?"
270  PRINT"(6) IS THE INVESTMENT I'M CONSIDERING A GOOD ONE?"
280  PRINT"(7) SHOULD I ACCEPT HIS/HER INVITATION?"
290  PRINT"(8) SHOULD I GET OUT OF TOWN FAST???"
300  PRINT"(9) IS IT TIME FOR A JOB CHANGE?"
310  PRINT"(10) ASK ANY YES OR NO TYPE OF QUESTION"
320  PRINT:PRINT:PRINT"WHAT NUMBER DO YOU WANT, ";B$;:INPUTA
330  IF A=10 GOTO 560
340  X=RND(10)
345  CLS
350 ON X GOTO 400,405,410,415,420,425,430,435,440,445
400  PRINT@453,"YES, IT APPEARS SO  BUT BE AWARE THAT":GOTO450
405  PRINT@453,"CERTAINLY! HOWEVER, ON THE OTHER HAND,":GOTO450
410  PRINT@453,"NO, IT DOESN'T LOOK LIKE IT, BUT":GOTO450
415  PRINT@453,"THAT'S FOR SURE! BUT, ON THE OTHER HAND,":GOTO450
420  PRINT@453,"I'D SAY YOU COULD COUNT ON IT, BUT":GOTO450
425  PRINT@453,"ARE YOU KIDDING? I'D TELL YOU EXCEPT THAT":GOTO450
430  PRINT@453,"THE FUTURE IS CLOUDY, ESPECIALLY SINCE":GOTO450
435  PRINT@453,"HOW SHOULD I KNOW? I'VE GOT PROBLEMS, LIKE":GOTO450
440  PRINT@453,"HECK, I DON'T KNOW! YOU MUST CONSIDER THAT":GOTO450
445  PRINT@453,"ACCORDING TO MY CRYSTAL BALL, YES! BUT"
450  Y=RND(10)
460 ON Y GOTO 510,515,520,525,530,535,540,545,550,555
510  PRINT"ALL MY SECRETS ARE IN MY TURBAN - WHICH WAS JUST STOLEN!":GOTO580
515  PRINT"I NEED SOME WINDEX FOR MY CRYSTAL BALL,";B$;"!":GOTO580
520  PRINT"UNDER THIS SLICK GRAY EXTERIOR, I'M ONLY WIRES & STUFF!":GOTO580
525  PRINT"IT WOULD TAKE A $300 PERSONAL SEANCE TO BE SURE!":GOTO580
530  PRINT"MY CRYSTAL BALL IS - LIKE SOME PEOPLE - CRACKED!!":GOTO580
535  PRINT"IN CASE I'M WRONG, REMEMBER WHAT THIS READING COST YOU!":GOTO580
540  PRINT"THAT'S A REALLY DIFFICULT QUESTION,";B$;"!":GOTO580
545  PRINT"MY MAGIC CARPET COULD BE STEERING ME WRONG, ";B$;"!":GOTO580
550  PRINT"THE FUTURE IS REALLY WHAT YOU MAKE OF IT,";B$;"...":GOTO580
555  PRINT"IF YOU DON'T LIKE MY ANSWER OH, FAITHLESS ONE, TRY ASTROLOGY!"
556  GOTO580
560  CLS:PRINT:PRINT:PRINT"TYPE IN QUESTION, THEN HIT 'ENTER'";:INPUTA$
570  GOTO 340
580 FOR D=1 TO 1500:NEXT D:GOTO 210
590  END
```

10 FOR X=1 TO 20:R=RND(25):PRINT R;:NEXT X

The screen will display 20 numbers. Write them down on a piece of paper, and RUN the program again. There will be 20

PROGRAM 3 RUN

```
                FORTUNE TELLING PROGRAM

     THE COMPUTER WILL LIST SEVERAL TYPICAL QUESTIONS
FOR YOU TO ASK THE 'FORTUNE TELLER'. YOU MAY CHOOSE TO
ASK YOUR OWN QUESTION INSTEAD. JUST ENTER THE APPROPRIATE
NUMBER.....(FOR ENTERTAINMENT PURPOSES ONLY).

WHEN READY TO START, PRESS ENTER?

WHAT IS YOUR NAME? HARRY
WHAT MONTH WERE YOU BORN? (MONTH NUMBER)? 7

   HARRY, CHOOSE ONE OF THE NUMBERS BELOW:

(1) SHOULD I MAKE THAT TRIP I'M THINKING ABOUT?
(2) IS HE/SHE SERIOUS, OR JUST PLAYING THE FIELD?
(3) SHOULD I TELL HIM/HER THE WHOLE TRUTH?
(4) SHOULD I ASK MY BOSS FOR A RAISE?
(5) WILL I BE RICH AND FAMOUS SOME DAY?
(6) IS THE INVESTMENT I'M CONSIDERING A GOOD ONE?
(7) SHOULD I ACCEPT HIS/HER INVITATION?
(8) SHOULD I GET OUT OF TOWN FAST???
(9) IS IT TIME FOR A JOB CHANGE?
(10) ASK ANY YES OR NO TYPE OF QUESTION


WHAT NUMBER DO YOU WANT,HARRY? 4
NO, IT DOESN'T LOOK LIKE IT, BUT
IN CASE I'M WRONG, REMEMBER WHAT THIS READING COST YOU!

   HARRY, CHOOSE ONE OF THE NUMBERS BELOW:

(1) SHOULD I MAKE THAT TRIP I'M THINKING ABOUT?
(2) IS HE/SHE SERIOUS, OR JUST PLAYING THE FIELD?
(3) SHOULD I TELL HIM/HER THE WHOLE TRUTH?
(4) SHOULD I ASK MY BOSS FOR A RAISE?
(5) WILL I BE RICH AND FAMOUS SOME DAY?
(6) IS THE INVESTMENT I'M CONSIDERING A GOOD ONE?
(7) SHOULD I ACCEPT HIS/HER INVITATION?
(8) SHOULD I GET OUT OF TOWN FAST???
(9) IS IT TIME FOR A JOB CHANGE?
(10) ASK ANY YES OR NO TYPE OF QUESTION


WHAT NUMBER DO YOU WANT,HARRY? 10


TYPE IN QUESTION, THEN HIT 'ENTER'? WILL TOMORROW BE A BETTER DAY?
ACCORDING TO MY CRYSTAL BALL, YES! BUT
IF YOU DON'T LIKE MY ANSWER OH, FAITHLESS ONE, TRY ASTROLOGY!
```

more numbers. Write them down as well. Now turn your computer off and wait about 5 seconds for the internal capacitors and the power supply to discharge. Then turn the computer on again, type in and RUN the above program, and check the numbers on the screen against those on your paper from the last time. They will be the *same* numbers in the *same sequence.* Not only that, but they will also be exactly the same in Level I and Level II!

Somewhere in some dark cave some genius developed random number tables that simulate actual randomness. These have been programmed into ROM for every random number limit. For example, run the foregoing tests after inserting any number from 1 to 32767 in the program in place of the 25 located between the parentheses after RND. The resulting number series will differ but will remain the *same* every time you turn on the computer.

This random number series apparently repeats after every string of 32767 numbers; thus, if we can somehow advance the series to some point beyond the beginning—and do it in such a way that it's likely to be different at each trial—then we can eliminate nonrandom consistency every time we turn on the computer and use a random number command. To do so is particularly important in games, where random numbers supposedly prevent prediction of the next event. If you've noticed a certain "winner" sequence in some games, this consistency is the probable reason!

Assuming that you're now convinced that your random number statement does not *really* generate random numbers, you'll want to know how to minimize the effects of this on your programs. Let us go back to the "seed" in line 205. When you enter your birth-month number, you enter a number from 1 to 12. This is multiplied by 10 in line 206 and then used the same number of times in line 207 to run up the random number list. (The 32767 in line 207 could be any number from 1 to 32767; it doesn't make any difference which. The computer simply moves up—or is it down?—the internal random number table N times, as instructed by the FOR-NEXT loop in line 207.)

So much for all this random chatter. Now the screen prints lines 210–320 (personalized with your B$ name in lines 210 and 320) and waits for your choice of the "menu"—list—of choices. Notice that line 320 has two PRINT commands for line spacing, another PRINT command for the question, and then INPUT A. An input statement can generate text or follow a PRINT statement. The choice is yours, but don't forget to use semicolons to prevent "carriage return" and colons to separate statements on the same line.

Type in your chosen number and press ENTER. Line 330 takes a look to see if you've chosen number 10. If you have, the program jumps to line 560 and asks you to type in your own question. Level I limits your question to 16 characters, but Level

II allows you 255 characters. Type in your question, hit ENTER, and line 57Ø will send the program to line 34Ø.

Notice that *regardless* of the number you type in from the menu (1 through 10), you finally get to line 34Ø. (If you typed in any number from 1 through 9 in response the the question in line 32Ø, the program falls through line 33Ø to line 34Ø.)

Line 34Ø sets the value of X at some integer (nondecimal) number from 1 to 10. Line 345 clears the screen. Line 35Ø contains a new and very useful command, ON-GOTO. The 10 numbers listed here, separated by commas and following GOTO, represent line numbers to which the program is to jump, depending on the value of X. If X is 1, the program jumps to the first line number, 4ØØ; if X is 2, it jumps to 4Ø5; if X is 3, it jumps to 41Ø; and so forth.

To illustrate how this works, let's say you pick question 4 as your menu choice: "Should I ask my boss for a raise?" The computer selects a random number for X in line 34Ø; let's say 7. Since X is now equal to 7, the computer looks for the seventh number listed after ON X GOTO in line 35Ø and finds 43Ø. The program jumps over all intervening lines to line 43Ø and prints THE FUTURE IS CLOUDY, ESPECIALLY SINCE near the center of the screen (print location 453) and then goes to line 45Ø. Here Y is set equal to another random number from 1 to 10; let's say 3. The program now jumps to the third number listed in line 46Ø, or line 52Ø, and prints, right below the previous line, UNDER THIS SLICK GRAY EXTERIOR I'M ONLY WIRES & STUFF!; it then goes to line 58Ø.

At line 58Ø, a time-delay loop keeps the printing on-screen for a few seconds (determined by the top value of D). The program then goes to line 21Ø and reprints the menu for another choice.

"Foul!" you say. "This is fixed!" Well, since the numbers in lines 34Ø and 45Ø are random—unknown in advance—aren't the computer responses "fate"?

If you analyze the responses in lines 4ØØ–445, you'll see that they are yes, no, or maybe statements, followed by a "cop-out" that's completed in lines 51Ø–555. Another way to put it: each statement in lines 4ØØ–445 giveth; each statement in lines 510–555 taketh away!

Notice that the ending of lines 4ØØ–44Ø are all the same—GOTO45Ø. Line 445 doesn't need this command since the next line *is* 45Ø. You might wonder why line 556 is not just added to

line 555, like the previous lines. Not enough room! In Level I, a line is limited to 70 characters *after* the line number prompt; Level II allows up to 240 characters after the prompt (*not* 255 as you might expect).

Modifications: Once again, as in Program 2, your imagination and computer memory size are the only limitations to enlarging this program. It's fun to play at parties, but providing more choices of answers—about 25 instead of 10—would result in less frequently repeated wisecracks. Also, obviously, you could ask much more personal questions—all the questions you've been afraid to ask—and program much more revealing answers!

Variables:

A$—continue
B$—name
N—birth month (random seed)
I—random seed loop
J—random series advance
A—menu selector
X—random number (first response line)
Y—random number (second response line)
D—time delay loop counter

PROGRAM 4
ASK YOUR GOVERNMENT
(4K, Level I or II)

You'll learn: Entry-error trap; using a subroutine

Description: Have you ever written a government agency or politician and gotten back a letter that was full of odd words that made no sense at all? Well, now you can generate these kind of letters on your TRS-80, using any words you like. This program produces a *similar* but *different* letter each time it is run.

Explanation: The program starts by printing on the screen the title and instructions (lines 101–104) followed by a menu of five questions (lines 105–109). It then stops and waits for your keyboard selection (line 110). Line 111 is a simple entry-error "trap," since it rejects any entry greater than 5 with an error message and returns you to line 110 for another choice. Line 111 does *not* reject 0 or negative number entries. If you want it to do so, change the beginning of line 111 to read IF N=<0 or N>5 PRINT . . . etc. (The word THEN is not always needed in an IF-THEN statement; it is not used in line 111, for example.)

Line 112 checks to see if you entered number 5 as your menu selection. If you have, the computer asks you to type in your question, B$. Remember that B$ is limited to 16 characters in Level I and to 255 in Level II.

Lines 113 and 114 clear the screen and ask for your name, A$. Line 115 clears the screen again, and line 120 starts printing out the governmental letter regardless of the question! Isn't this just like bureaucracy in action? The A$ in line 120 personalizes the letter with your last name. Line 130 prints the first four words of the letter text and then goes to the subroutine starting at line 500.

The first thing that happens in this subroutine is that random numbers from 1 to 10 are established for variables A, B, and C. Line 505 uses the value of variable A in an ON A GOTO statement to send the program to one line of the group of lines

LEVEL I/II LISTING

```
99  CLS
100 REM * LEVEL I OR II 4K * COPYRIGHT FRED BLECHMAN 1978 *
101  PRINT:PRINT"              ASK YOUR GOVERNMENT!"
102  PRINT:PRINT"     WHAT WOULD YOU LIKE TO ASK YOUR CONGRESSMAN?"
103  PRINT"JUST PICK A NUMBER, OR ENTER 5 TO ASK YOUR OWN QUESTION."
104  PRINT"THE COMPUTER WILL PRINT A TYPICAL REPLY...."
105  PRINT:PRINT"(1) WHY ARE MY TAXES SO HIGH?"
106  PRINT"(2) WHAT'S HOLDING UP THE NEW FREEWAY?"
107  PRINT"(3) WILL WE BE GETTING STREET LIGHTS SOON?"
108  PRINT"(4) WHAT ARE YOU DOING ABOUT CRIME IN THE STREETS?"
109  PRINT"(5) ASK YOUR OWN QUESTION....."
110  PRINT:INPUT"WHAT NUMBER, PLEASE";N
111  IF N>5 PRINT"ENTER 1 TO 5 ONLY, PLEASE!":GOTO 110
112  IF N=5 THEN CLS:PRINT:INPUT"TYPE AND ENTER QUESTION";B$
113  CLS:PRINT@260,"OH, BY THE WAY, ";
114  INPUT"WHAT'S YOUR LAST NAME";A$
115  CLS
120  PRINT"DEAR MR. OR MRS. (OR MS.)";A$;","
130  PRINT"     IN THE INTEREST OF ";:GOSUB 500
140  PRINT"IT HAS BEEN DECIDED THAT THE ";:GOSUB 500
150  PRINT"PLAN BE IMPLEMENTED IMMEDIATELY. THIS ";:GOSUB 500
160  PRINT"APPROACH TO ";:GOSUB 500
170  PRINT"MEETS WITH THE CONCEPTUAL APPROVAL OF HEW, SBA, ";
180  PRINT"FHA, FAA, CAB, IRS, OSHA, FTC AND FDA. IN ORDER TO ";
190  PRINT"MINIMIZE THE EFFECTS OF THE ";:GOSUB 500
200  PRINT"ON THE ";:GOSUB 500
210  PRINT"IT HAS BEEN NECESSARY TO REDUCE THE ";:GOSUB 500
220  PRINT"AND ITS AFFILIATED CONTINGENCIES."
230  PRINT"     WE KNOW YOU WILL AGREE THAT THIS ";:GOSUB 500
240  PRINT"CAN ONLY IMPROVE THE OVERALL ";:GOSUB 500
250  PRINT"IMMEASURABLY. THANKS FOR YOUR INQUIRY."
260  PRINT"                        SINCERELY,"
270  PRINT"                                YOUR CONGRESSMAN."
280  GOTO 280
499  REM * RANDOMLY SELECT THREE WORD-STRINGS *
500  A=RND(10):B=RND(10):C=RND(10)
505  ON A GOTO 510,520,530,540,550,560,570,580,590,600
510  PRINT"INTEGRATED ";:GOTO 605
520  PRINT"TOTAL ";:GOTO 605
530  PRINT"SYSTEMATIZED ";:GOTO 605
540  PRINT"PARALLEL ";:GOTO 605
550  PRINT"FUNCTIONAL ";:GOTO 605
560  PRINT"RESPONSIVE ";:GOTO 605
570  PRINT"OPTIMAL ";:GOTO 605
580  PRINT"SYNCHRONIZED ";:GOTO 605
590  PRINT"COMPATIBLE ";:GOTO 605
600  PRINT"BALANCED ";
605  ON B GOTO 610,620,630,640,650,660,670,680,690,700
610  PRINT"MANAGEMENT ";:GOTO 705
620  PRINT"ORGANIZATIONAL ";:GOTO 705
630  PRINT"MONITORED ";:GOTO 705
640  PRINT"RECIPROCAL ";:GOTO 705
650  PRINT"DIGITAL ";:GOTO 705
660  PRINT"LOGISTICAL ";:GOTO 705
670  PRINT"TRANSITIONAL ";:GOTO 705
680  PRINT"INCREMENTAL ";:GOTO 705
690  PRINT"THIRD-GENERATION ";:GOTO 705
700  PRINT"LINEAR ";
705  ON C GOTO 710,720,730,740,750,760,770,780,790,800
710  PRINT"OPTIONS ";:RETURN
720  PRINT"FLEXIBILITY ";:GOTO 805
730  PRINT"CAPABILITY ";:GOTO 805
740  PRINT"MOBILITY ";:GOTO 805
750  PRINT"PROGRAMMING ";:GOTO 805
```

```
760  PRINT"CONCEPT ";:GOTO 805
770  PRINT"TIME-PHASE ";:GOTO 805
780  PRINT"PROJECTION ";:GOTO 805
790  PRINT"HARDWARE ";:GOTO 805
800  PRINT"CONTINGENCY ";
805  RETURN
```

PROGRAM 4 RUN

```
     WHAT WOULD YOU LIKE TO ASK YOUR CONGRESSMAN?
JUST PICK A NUMBER, OR ENTER 5 TO ASK YOUR OWN QUESTION.
THE COMPUTER WILL PRINT A TYPICAL REPLY....

(1) WHY ARE MY TAXES SO HIGH?
(2) WHAT'S HOLDING UP THE NEW FREEWAY?
(3) WILL WE BE GETTING STREET LIGHTS SOON?
(4) WHAT ARE YOU DOING ABOUT CRIME IN THE STREETS?
(5) ASK YOUR OWN QUESTION.......

WHAT NUMBER, PLEASE? 2
OH, BY THE WAY, WHAT'S YOUR LAST NAME? SMITH
DEAR MR. OR MRS. (OR MS.)SMITH,
     IN THE INTEREST OF FUNCTIONAL DIGITAL PROJECTION
IT HAS BEEN DECIDED THAT THE SYSTEMATIZED MONITORED CONCEPT
PLAN BE IMPLEMENTED IMMEDIATELY. THIS OPTIMAL LOGISTICAL MOBILITY
APPROACH TO SYNCHRONIZED THIRD-GENERATION PROGRAMMING
MEETS WITH THE CONCEPTUAL APPROVAL OF HEW, SBA,
FHA, FAA, CAB, IRS, OSHA, FTC AND FDA. IN ORDER TO
MINIMIZE THE EFFECTS OF THE TOTAL TRANSITIONAL CONTINGENCY
ON THE BALANCED THIRD-GENERATION FLEXIBILITY
IT HAS BEEN NECESSARY TO REDUCE THE INTEGRATED MANAGEMENT CONTINGENCY
AND ITS AFFILIATED CONTINGENCIES.
     WE KNOW YOU WILL AGREE THAT THIS BALANCED LOGISTICAL FLEXIBILITY
CAN ONLY IMPROVE THE OVERALL RESPONSIVE INCREMENTAL PROJECTION
IMMEASURABLY. THANKS FOR YOUR INQUIRY.
                    SINCERELY,
                              YOUR CONGRESSMAN.
```

510–600. For example, if the random number chosen for A is 3, then the program jumps to the third line number in line 505—line 530—and the computer prints the word SYSTEMATIZED and goes to line 605. Line 605 selects a line number based on the random value established for B. The word on that line number is printed, and the program then goes to line 705 where the same action takes place with the random number chosen for C. The program then goes to line 805 where RETURN sends it back to the line where the subroutine was called—line 130. Since there is nothing after GOSUB 500 on line 130, the program moves on to line 140 and continues with more of the same bureaucratic skulduggery.

Confused? This all happens so fast on the screen that it seems unbelievable, but read through the last paragraph again, follow it through in the program listing, and everything will fall into place. The subroutine simply picks out three words—one each from three groups of ten words each—and the semicolons hold the whole thing together on the screen, except at the end of lines, where words are sometimes broken off and continued on the next line. These broken words could be avoided, but the programming to do so would be somewhat too sophisticated for this book.

Lines 14Ø, 15Ø, 16Ø, 19Ø, 2ØØ, 21Ø, 23Ø, and 24Ø also use the subroutine to add randomly selected word groupings into the text at the designated points. Some lines (17Ø, 18Ø, 22Ø, 25Ø, 26Ø, and 27Ø) do not use the subroutine, being complete in themselves.

Line 28Ø puts the computer into an endless loop, which allows the 15-line letter to reside on the screen for an indefinite period. If line 28Ø were END, the READY prompt would appear at the bottom left of the screen and the text would scroll up, losing the top line.

To end the program and regain keyboard control, hit the BREAK key.

Modifications: The fixed text and randomly chosen words can be changed any way you like. You could even change the whole letter to respond to bill collectors, say, with some well-chosen words! Be careful that the groups from which individual words are selected follow a reasonable structure. For example, group 1 could be superlatives (very, extremely, highly, extraordinarily, etc.), group 2 could be adjectives (high, short, fat, wide, etc.), and group 3 could be nouns (house, farm, building, mountain, etc.).

Variables:

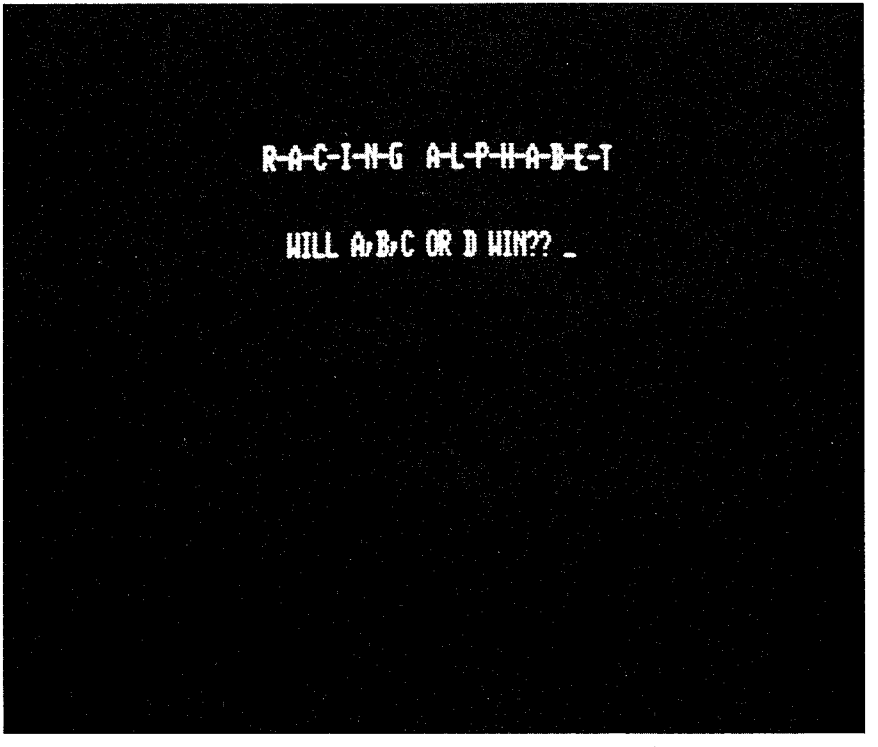
N—menu selection
B$—Specific question
A$—last name
A—selects word from first group
B—selects word from second group
C—selects word from third group

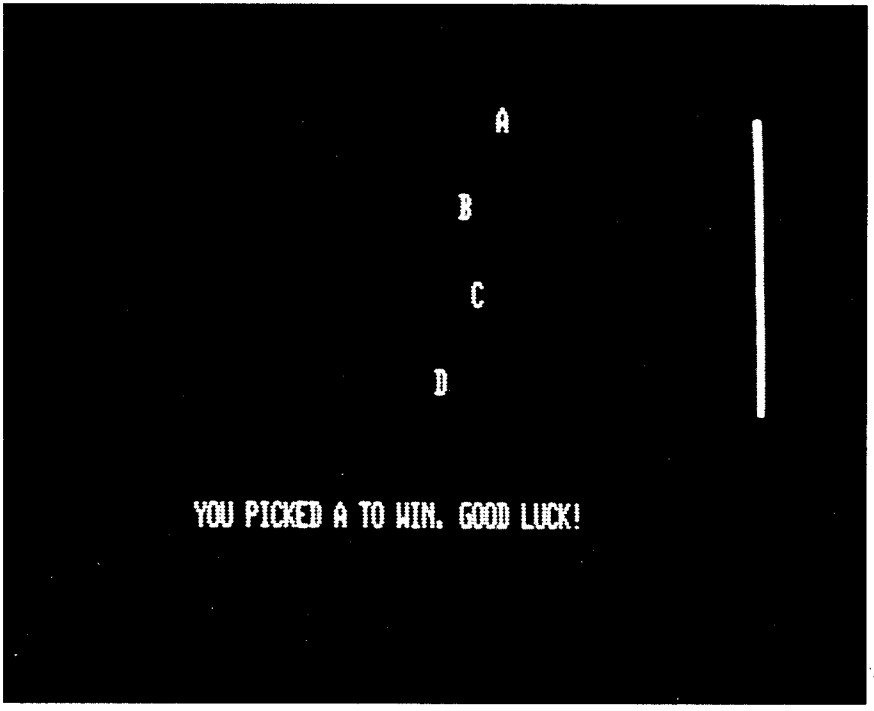
PROGRAM 5
RACING ALPHABET
(4K, Level I or II)

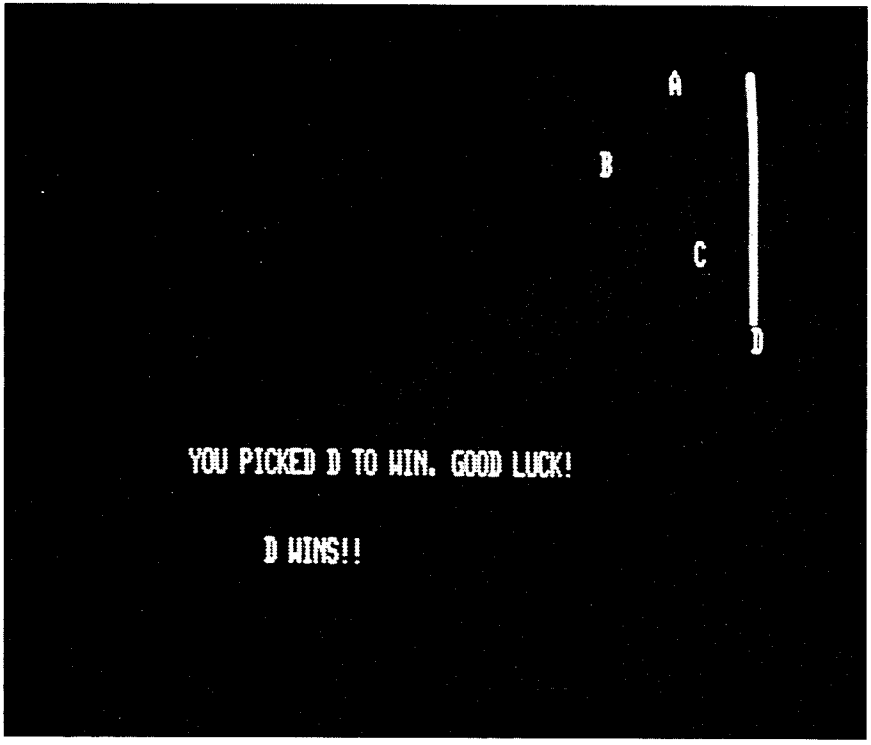
You'll learn: Simple graphic movement; speed control; logic statements

Description: We're off to the races! Choose A, B, C, or D as your winner, and each letter races to a finish line. When any letter reaches the finish line ahead of the others, the program stops and announces the winner, waits a short time, and asks for another choice. Press BREAK to exit the program at any time.

Explanation: We'll explain the Level I listing first and then the changes that must be made for Level II. You'll find several REM




A
B
C
D
YOU PICKED A TO WIN. GOOD LUCK!


A
B
C
D
YOU PICKED D TO WIN. GOOD LUCK!
D WINS!!

```
3  REM * ORIGINAL PROGRAM BY LEONARD V. DUBOSE*
4  REM * REVISED EXTENSIVELY BY FRED BLECHMAN *
5  GOSUB 100
9  REM * ADVANCE EACH LETTER 1 TO 4 SPACES *
10  L=L+RND(4):M=M+RND(4):N=N+RND(4):O=O+RND(4)
14  REM * PRINT NEW POSITION OF EACH LETTER *
15  P. AT(64+L),"A";
20  P. AT(192+M),"B";
25  P. AT(320+N),"C";
30  P. AT(448+O),"D";
34  REM * CHECK TO SEE IF FINISH LINE REACHED *
35  IF(64+L)>122 G. 75
40  IF(192+M)>250 G. 75
45  IF(320+N)>378 G. 75
50  IF(448+O)>506 G. 75
51  REM * LINE 52 SETS RACING SPEED. ELIMINATE FOR FAST RACE *
52  FOR X=1 TO 100:NEXT X
54  REM * DELETE LETTERS, BUT LEAVE FINISH LINE *
55  P. AT(64+L),"      ";
60  P. AT(192+M),"      ";
65  P. AT(320+N),"      ";
70  P. AT(448+O),"      ";
71  GOTO 80
74  REM * HIGHEST LETTER VALUE WINS. TIES LOOP AGAIN *
75  IF(L>M)*(L>N)*(L>O)P. AT788,"A WINS!!":G. 90
76  IF(M>L)*(M>N)*(M>O)P. AT788,"B WINS!!":G. 90
77  IF(N>L)*(N>M)*(N>O)P. AT788,"C WINS!!":G. 90
78  IF(O>L)*(O>M)*(O>N)P. AT788,"D WINS!!":G. 90
80  GOTO 10
90  FOR X=1 TO 1500:N. X
95  GOTO 5
100  CLS:P. AT140,"R-A-C-I-N-G  A-L-P-H-A-B-E-T"
105  A=1:B=2:C=3:D=4
109  REM * USE WIN CHOICE TO CYCLE RANDOM NUMBERS *
110  P. :IN. "              WILL A,B,C OR D WIN?";E
111  IF (E<1)+(E>4) P. "PICK A,B,C OR D":G. 110
115  E=10*E
116  FOR X=1 TO E:J=RND(4):NEXT X
130  CLS:P. AT 654,"YOU PICKED ";
131  IF E=10 P. "A";
132  IF E=20 P. "B";
133  IF E=30 P. "C";
134  IF E=40 P. "D";
135  P. " TO WIN. GOOD LUCK!"
140  L=0:M=0:N=0:O=0
150  FOR Y=3 TO 23:SET(120,Y):NEXT Y
160  RETURN
```

statements throughout to clarify what's happening in each section of the program. These remarks have no effect on the program itself.

Line 5 sends the program to a subroutine. Why a GOSUB subroutine rather than a simple GOTO? In this case, there is no particular reason, since the subroutine (lines 1ØØ–16Ø) is used only once for each race. Normally, subroutines are used several times in a program.

The next obvious question is, why weren't lines 1ØØ–165 put at the *beginning* of the program in the first place? The answer is that the author got so involved in generating the "code" for making the

```
1 REM * RACING ALPHABET PROGRAM *
2 REM * TRS-80 LEVEL II 4K *
3  REM * ORIGINAL PROGRAM BY LEONARD V. DUBOSE*
4  REM * REVISED EXTENSIVELY BY FRED BLECHMAN *
5  GOSUB 100
9  REM * ADVANCE EACH LETTER 1 TO 4 SPACES *
10  L=L+RND(4):M=M+RND(4):N=N+RND(4):O=O+RND(4)
14  REM * PRINT NEW POSITION OF EACH LETTER *
15  PRINT@(64+L),"A";
20  PRINT@(192+M),"B";
25  PRINT@(320+N),"C";
30  PRINT@(448+O),"D";
34  REM * CHECK TO SEE IF FINISH LINE REACHED *
35  IF(64+L)>122 GOTO75
40  IF(192+M)>250 GOTO75
45  IF(320+N)>378 GOTO75
50  IF(448+O)>506 GOTO75
51  REM * LINE 52 SETS RACING SPEED. ELIMINATE FOR FAST RACE *
52  FOR X=1 TO 100:NEXT X
54  REM * DELETE LETTERS, BUT LEAVE FINISH LINE *
55  PRINT@(64+L),"      ";
60  PRINT@(192+M),"      ";
65  PRINT@(320+N),"      ";
70  PRINT@(448+O),"      ";
71  GOTO 80
74  REM * HIGHEST LETTER VALUE WINS. TIES LOOP AGAIN *
75  IF(L>M)*(L>N)*(L>O)PRINT@788,"A WINS!!":GOTO90
76  IF(M>L)*(M>N)*(M>O)PRINT@788,"B WINS!!":GOTO90
77  IF(N>L)*(N>M)*(N>O)PRINT@788,"C WINS!!":GOTO90
78  IF(O>L)*(O>M)*(O>N)PRINT@788,"D WINS!!":GOTO90
80  GOTO 10
90  FOR X=1 TO 1500:NEXTX
95  GOTO 5
100 CLS:PRINT@140,"R-A-C-I-N-G  A-L-P-H-A-B-E-T"
109  REM * USE WIN CHOICE TO CYCLE RANDOM NUMBERS *
110  PRINT:INPUT"                WILL A,B,C OR D WIN?";E$
111 IF(E$="A")OR(E$="B")OR(E$="C")OR(E$="D") THEN115
112 PRINT"PICK A,B,C, OR D!!":GOTO110
115 E=RND(40)
116  FOR X=1 TO E:J=RND(4):NEXT X
130  CLS:PRINT@ 654,"YOU PICKED ";E$;" TO WIN. GOOD LUCK!"
140  L=0:M=0:N=0:O=0
150  FOR Y=3 TO 23:SET(120,Y):NEXT Y
160  RETURN
165 END
```

program work that he didn't leave room for instructions. It's really a better idea to start programs at line 10Ø so that you will have plenty of room *before* line 10Ø for any instructions or explanation you might like to add after the program is running to your satisfaction.

Incidentally, "renumber" programs are now available that will clean up your line number sequence. The programs in the book were written and listed without that luxury.

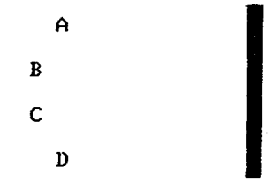
To get back to the Racing Alphabet Program, line 10Ø simply prints out the title, with dashes between the letters for graphic effect. Line 1Ø5 sets the values of A through D at 1 through 4, respectively. Line 11Ø asks you to type in the letter of your choice for

PROGRAM 5 RUN

```
            R-A-C-I-N-G  A-L-P-H-A-B-E-T

              WILL A,B,C OR D WIN?? F
PICK A,B,C, OR D!!

              WILL A,B,C OR D WIN?? C
```




```
YOU PICKED C TO WIN. GOOD LUCK!
```

the winner and assigns this value to variable E. In other words, if you enter A, the computer assigns the value of 1 to E, since line 1Ø5 assigned the value of 1 to A. Similarly, if you typed in B, the E would equal 2, and so on. However, suppose that you type in some letter other than A, B, C, or D. The value of E would then probably be plus or minus .5, the typical Level I start-up values for all variables. Line 111 is an entry-error trap to see if you entered the values of 1, 2, 3, and 4 for A, B, C, and D, respectively. This line says, in effect, that IF the value assigned to E is less than 1 OR greater than 4, the entry is wrong and therefore rejected; the screen then prints PICK A, B, C, OR D and goes back to line 11Ø for another entry for variable E. (The plus sign in line 111 represents the logical OR.) Notice the use of parentheses in line 111 to collect the terms. While parentheses can sometimes be omitted, it's safest to use them when in doubt.

Line 115 multiplies E by 10 and then uses this value in line 116 as a random number seed to move up the random number series as explained in Program 3. The FOR-NEXT loop uses X to count E random numbers in the RND(4) internal random number table. Each number is assigned to variable J just for counting purposes.

To follow the screen formatting from this point on, refer to the Video Display Worksheet in Appendix A.

Line 13Ø clears the screen and prints the beginning of a line near the bottom of the screen (print location 654). The semicolon stops the printing until the program determines what to print next. Lines 131–134 then determine which letter was picked by looking at the value of E (which was multiplied by 10 in line 115). The proper letter is printed, and the semicolon here reserves further printing for line 135, which finishes the statement. This is a good example of how the semicolon allows derived values or text to be printed within an otherwise predetermined sentence.

Now line 14Ø sets L, M, N, and O to Ø. Line 15Ø puts the "finish line" on the screen with a FOR-NEXT loop that lights screen blocks from (12Ø, 3) to (12Ø, 23). Remember that there are 128 horizontal block positions (Ø–127) and 48 vertical positions (Ø–47), as defined by the instruction SET (horizontal number, vertical number). Usually, X is used for the horizontal dimension, Y for the vertical, although any variables can be used.

Line 16Ø simply returns the program to the line where the subroutine was called—line 5. Since there is no additional statement on line 5, the program moves on past the REM to line 1Ø.

Line 14Ø established each of the variables L, M, N, and O equal to zero. Now line 1Ø increases each of them by a separately chosen random number from 1 to 4. Line 15 prints A—on the second screen line—at print position 64 plus the number of spaces equal to the new value of L. Similarly, B, C, and D are printed on the screen on separate lines, with each at an initial position determined by the value of M, N, and O established in line 1Ø and the directions given in lines 2Ø, 25, and 3Ø, respectively.

To determine when there is a winner, the program checks in lines 35, 4Ø, 45, and 5Ø to see if any letter has reached or crossed the finish line. It does so by determining whether the values of L, M, N, and O, when added to the starting values on each line, have reached more than 58 spaces beyond the starting point. Why "more than 58" spaces? Well, the vertical finish line drawn in line 15Ø (SET 12Ø, Y) is on the left side of print locations 124, 188, 252, 316, 38Ø, 444, and 5Ø8. (See the Video Display Worksheet in Appendix A or page 1Ø6 of the Level I Manual or Page E/1 of the Level II Manual.) The initial letter print locations (64, 192, 32Ø, and 448) are chosen so that the letters will be separated vertically by two lines as they move horizontally across the screen.

Take the letter A as an example. It starts out at print position 64 plus L and moves ahead each time the program is incremented (as you'll see later) by a random number from 1 to 4. The finish line is at print location 124. If A has reached position 123,

which is the first number "greater than" 122, it would have just reached the finish line. If A is past that point, it has *crossed* the finish line. Line 35 checks to see if A has reached or crossed the finish line. If 64 plus L is greater than 122, it has, and the program then goes to line 75. If A has not, then line 40 checks to see if B has reached or crossed the finish line at location 252. Remember that "greater than 250" is 251 or more since 251 touches the finish line at 252. Similarly, lines 45 and 50 check the positions of C and D.

If none of the letters has reached the finish line, the program falls through to line 51 and then line 52, which uses the variable X in a time-delay FOR-NEXT loop of about two-tenths of a second. You can change the 100 in line 52 to a smaller number for a faster race (shorter time delay) or a larger number for a slower race. For the fastest race, leave line 52 out altogether.

Lines 55–70 are very important. They "erase" each letter on the screen by printing blanks at the last location of each letter. Were it not for this, a trail of letters would be left across the screen. (Although five blanks are shown between the quotation marks, one is enough.)

Line 71 sends the program to line 80, which sends it back to line 10. This process keeps repeating, with the letters advancing across the screen until one reaches or crosses the finish line, as determined by lines 35–50. (Since each letter can be incremented by as many as four spaces on each "move," it could cross the finish line and go three spaces beyond it before being caught by lines 35–50.)

Ah, but suppose two or more letters reach the finish lines together—a tie? The first one caught by lines 35–50 sends the program to line 75. Here computer logic comes into play, since the asterisk (*) means AND. Line 75 says "IF L is greater than M, AND L is greater than N, AND L is greater than 0, PRINT at location 788, A WINS!!, and then GOTO line 90." In other words, A wins *only* if it is the farthest advanced of the letters. If this is *not* so, then B's advance is checked by line 76, C's is checked by line 77, and D's is checked by line 78. Unless one letter is "ahead" of *all* the others, there is no winner, and the program falls through to line 80, goes back to line 10 for another advance, and is then checked again for a single winner.

Finally, after one letter wins, the program "freezes" on the screen for about 3 seconds (line 90 time-delay FOR-NEXT loop) and then goes to line 5 for another race.

The Level II version of this program is similar but requires some necessary changes in the handling of Level I. Of course, P. becomes PRINT, P.AT becomes PRINT@, and G. becomes GOTO.

At line 11Ø, we must use a string variable (E$) for the chosen letter since, as discussed in Program 2, we cannot enter a letter as the input for a simple variable. This alteration changes the entry-error trap in line 111. It now must use string comparisons to make sure that an A, B, C, or D has been entered. If none has been, line 111 falls through to line 112, which prints a reminder and returns the program to line 11Ø for another input. If line 111 is satisfied, however, the program jumps to line 115, picks a random number from 1 through 40 as a "seed," and runs up the random number table in line 116. Line 13Ø clears the screen, prints the chosen letter message, and proceeds in the same manner as Level I.

Modifications: You could use numbers instead of letters. You could also allow for more "racers"—perhaps as many as 10—without slowing the program too much and still leaving room for a winner statement. You could add a variable to be used as a counter to note the number of each race on the screen. You could make provisions for betting and even for race odds. And, in the next program, we'll show you how to use graphic symbols for the racers.

Variables:

L—A track position
M—B track position
N—C track position
O—D track position
X— counter, FOR-NEXT loops
E— chosen letter (Level I)
E$— chosen letter (Level II)
E—random seed
J—random series advance
Y—finish line vertical limits

PROGRAM 6
RUN, SPOT, RUN!
(4K, Level II Only)

You'll learn: Level II graphic code

Description: Five graphic "dogs" race across the screen to a finish line. When one "wins," movement stops and the winner is announced.

Explanation: Although this program appears to do about the same thing as Program 5 (Racing Alphabet), it does it quite differently. The most obvious change is the use of graphic symbols, which are not available in Level I.

To understand the Level II Graphic Code, take a look at Fig. 6-1. Each printing location on the TRS-80 display is a rectangle composed of six segments—two wide by three high. Lighting one or more segments at the same time, or none at all (leaving a blank), provides 64 possible permutations. Each permutation is assigned a special number (starting at 128) within the TRS-80 ROM and is called by this CHR$ number.

Incidentally, the lighted segments within the rectangle each have a "value" that will allow you to determine the CHR$ number without referring to Fig. 6-1. Just add the values of the lighted segments in Fig. 6-2 to 128 to get the CHR$ number.

Example: Suppose that segments 2, 8, and 16 in Fig. 6-2 are lighted. Then 128 plus 2 plus 8 plus 16 equals 154. Now check this value against Fig. 6-1. They agree. Isn't that amazing? Here's a simple but effective binary coding system.

With that under your belt, let's see how we can come up with a symbol for a dog. Use the Video Display Worksheet in Appendix A or one in your manual and lightly fill in adjoining rectangles to form a figure of a dog. Of course, you could also create a turtle, an airplane, a boat, or any shape on the screen (with resolution limited to the size of the rectangular segments)—but we're doing a dog. Now, using either Fig. 6-1 or the value system just described, determine

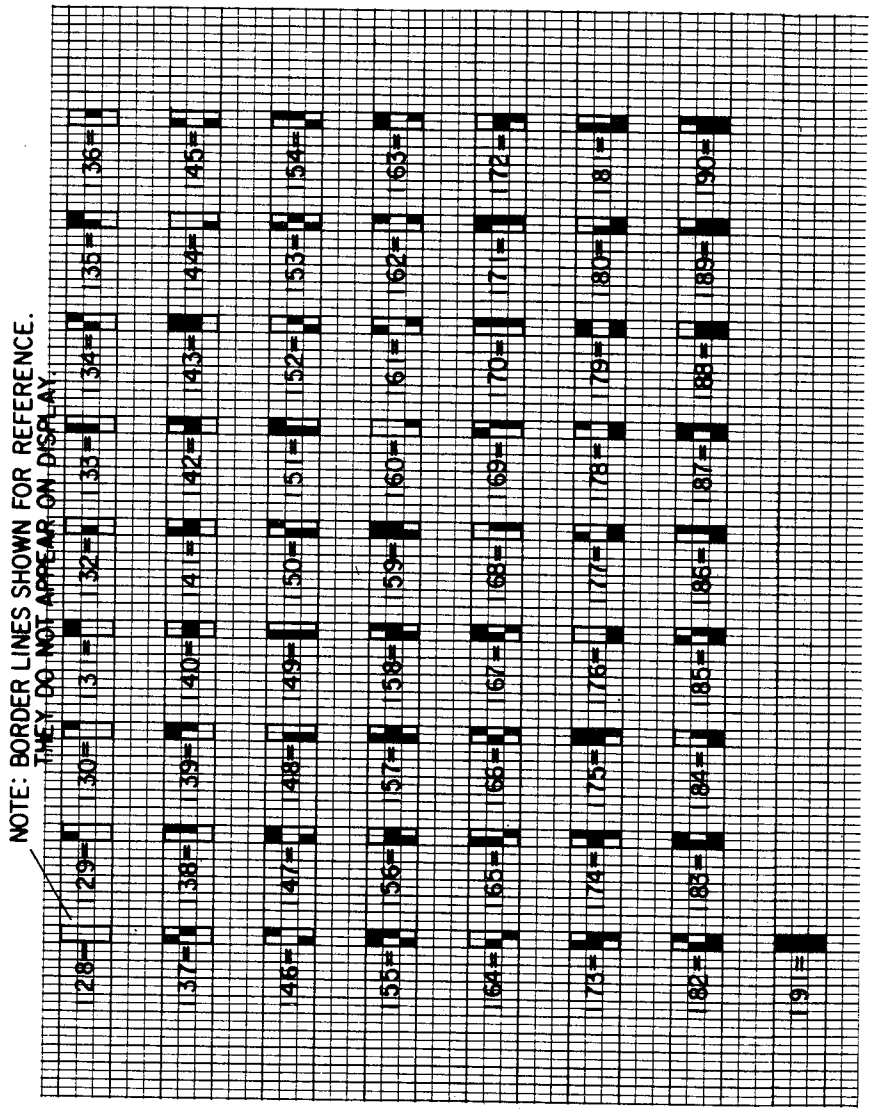


Fig. 6-1. Level II graphic block code

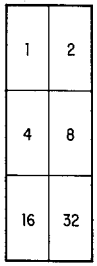


Fig. 6-2. Values of lighted segments

which graphic code numbers represent the required rectangle configurations, reading from left to right. For example, our dog is composed of 157, 140, 172, 131, and 129, as shown in Fig. 6-3. To display this dog on the screen, simply type PRINT CHR$(157); CHR$(140); CHR$(172); CHR$(131); CHR$(129) and then ENTER. Notice the parentheses and semicolons.

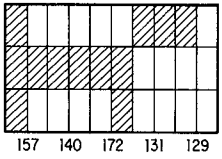
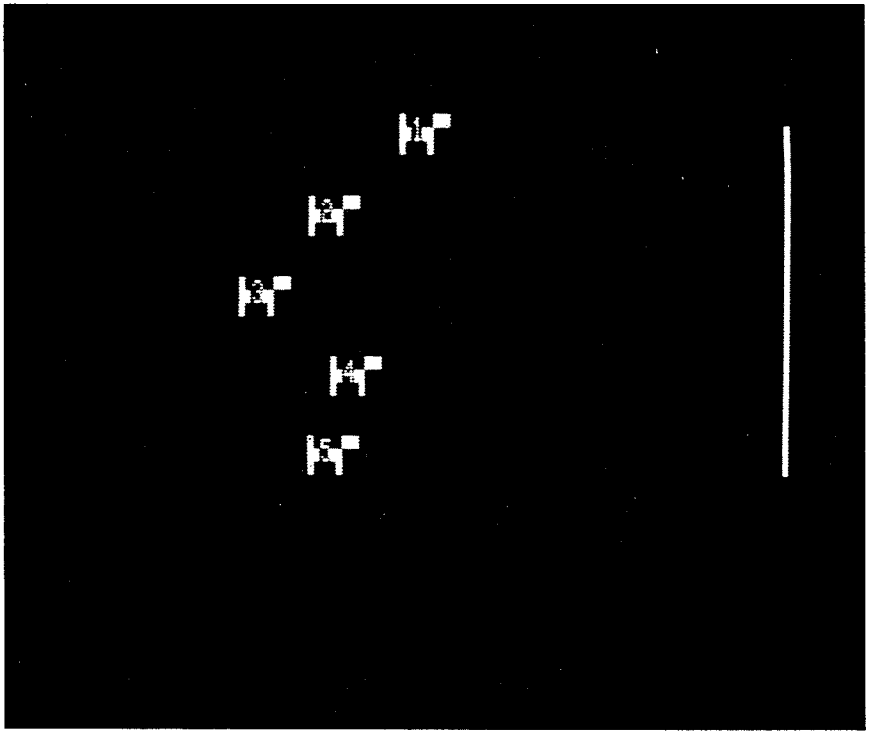
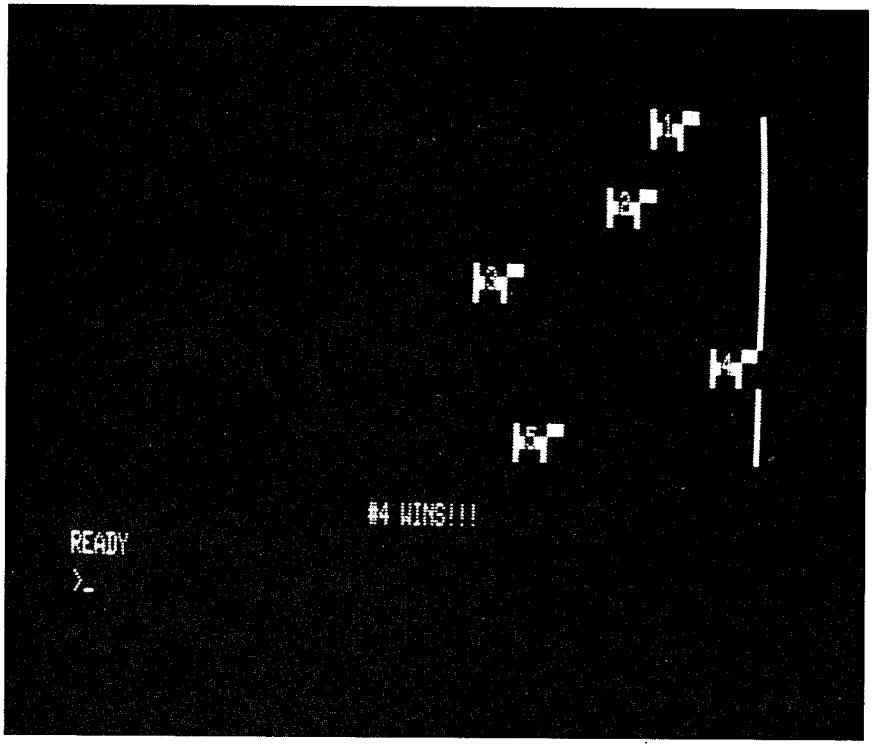
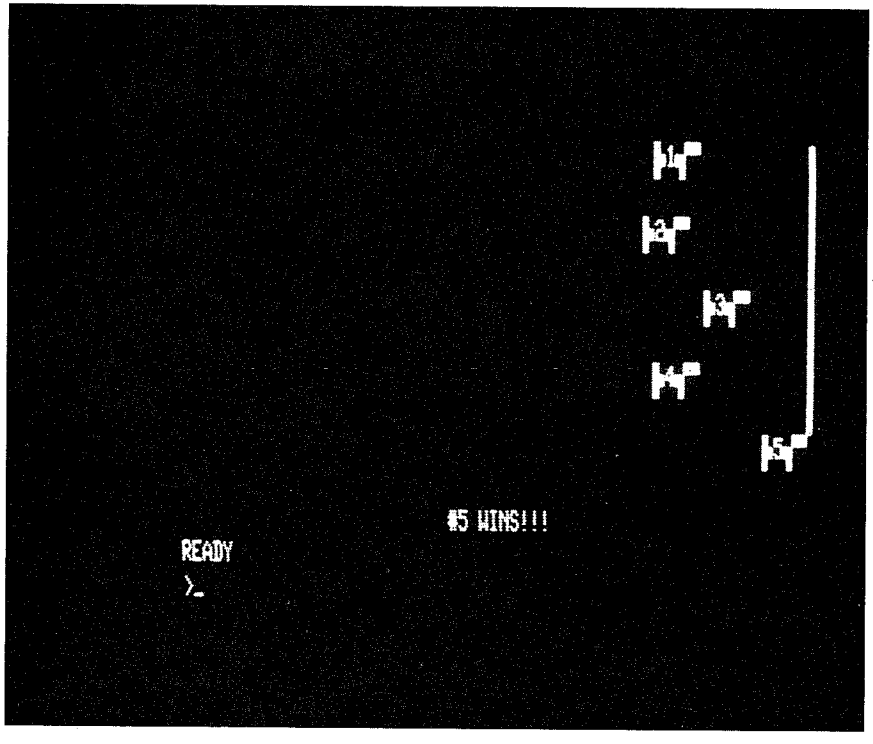


Fig. 6-3. Graphic dog




#4 WINS!!!
READY
>_


#5 WINS!!!
READY
>_

```
5 REM * DOGRACE WITH GRAPHIC DOGS - LEVEL II *
6 REM * COPYRIGHT 1978 FRED BLECHMAN *
7 REM * SET SPEED AT LINES 60-100 (A=A+1 OR A=A+2, ETC.)
10 CLS
15 FOR Y=3 TO 29:SET(123,Y):NEXT
20 A=64:B=192:C=320:D=448:E=576
21 PRINT@A,CHR$(157);"1";CHR$(172);CHR$(131);CHR$(129);
22 PRINT@B,CHR$(157);"2";CHR$(172);CHR$(131);CHR$(129);
23 PRINT@C,CHR$(157);"3";CHR$(172);CHR$(131);CHR$(129);
24 PRINT@D,CHR$(157);"4";CHR$(172);CHR$(131);CHR$(129);
25 PRINT@E,CHR$(157);"5";CHR$(172);CHR$(131);CHR$(129);
26 GOTO 55
30 PRINT@A,CHR$(157);"1";CHR$(172);CHR$(131);CHR$(129);:GOTO50
31 PRINT@B,CHR$(157);"2";CHR$(172);CHR$(131);CHR$(129);:GOTO50
32 PRINT@C,CHR$(157);"3";CHR$(172);CHR$(131);CHR$(129);:GOTO50
33 PRINT@D,CHR$(157);"4";CHR$(172);CHR$(131);CHR$(129);:GOTO50
34 PRINT@E,CHR$(157);"5";CHR$(172);CHR$(131);CHR$(129);
50 IF A>120 PRINT@730,"#1 WINS!!!":END
51 IF B>248 PRINT@730,"#2 WINS!!!":END
52 IF C>376 PRINT@730,"#3 WINS!!!":END
53 IF D>504 PRINT@730,"#4 WINS!!!":END
54 IF E>632 PRINT@730,"#5 WINS!!!":END
55 X=RND(5)
56 ON X GOTO 60,70,80,90,100
60 PRINT@A,"      ";:A=A+1:GOTO30
70 PRINT@B,"      ";:B=B+1:GOTO31
80 PRINT@C,"      ";:C=C+1:GOTO32
90 PRINT@D,"      ";:D=D+1:GOTO33
100 PRINT@E,"      ";:E=E+1:GOTO34
```

Now we can explain the program at hand. Although REM statements are not functional within the program, they can contain some vital information. Look at line 7, for example, which tells how to change the speed of the race (as you'll see further on).

Line 1Ø clears the screen, and line 15 draws the finish line at horizontal location 123 between the vertical locations of 3 and 29. Line 2Ø sets the values of A, B, C, D, and E at numbers equal to print locations at the left side of the screen. Lines 21–25 print our graphic dogs at these locations, with a minor change. Notice that each "dog" has a number in the location formerly held by CHR$(14Ø). You can mix regular characters with graphic codes so long as you don't forget parentheses, quotation marks, and semicolons as shown!

Up to this point we've placed the dogs on the starting line, but that's all. Now line 26 jumps the program to line 55, where X, a random number from 1 to 5, is chosen. Line 56 looks at X and sends the program to a line number determined by the value of X; if X is 1, the program goes to line 6Ø; if X is 2, the program goes to line 7Ø; and so on. Let's assume the random number chosen for X in this instance is 4. The program jumps to line 9Ø, since this is the fourth ON-GOTO number in line 56. Line 9Ø tells the computer to print five

blank spaces (since the graphic dog uses five spaces) at the present position of dog No. 4, thus "erasing" dog 4 from the screen. Then the computer is instructed to add 1 to the existing value of D for a new D, then to go to line 33. Line 33 prints a new dog 4 image on the screen one space ahead of where it was a moment ago. The last instruction on line 33 is to go to line 5Ø.

Lines 5Ø–54 check to see if any of the dogs have reached or crossed the finish line, prints a winner message on the screen if one has, and then ends the program. If there is no winner yet, the program drops through to line 55, selects another X, and advances one dog by one space, continuing in this fashion until one dog crosses the finish line. Since each dog is five spaces "long," the "greater than" numbers in lines 5Ø–54 are five *less* than the printing location of the finish line, allowing any dog to "win by a nose."

For a faster race, increase the add-on number in lines 6Ø–1ØØ from 1 to 2 or 3.

Modifications: Make up your own graphic symbols and have a tortoise/hare race. You can program the advances so that the hare jumps four spaces, but only on every fourth turn.

Also, you can assign the code for each dog as a string variable, such as F$ = CHR$(157) + "1" + CHR$(172) + CHR$(131) + CHR$(129) for dog 1. From then on, you need use only the variable F$ to print dog 1. Notice the use of plus signs instead of semicolons in this case, a practice called "string concatenation"—adding strings together.

You may also want to "seed" the random number table. If you *don't,* your winning dog sequence from computer turn-on will probably be 5-5-1-2-5-5-3-4-4-4-5-2-1-1-4-3-3-5-1-3-3-4-3-1-3 for the first 25 races. That's five wins for dog 1, two for dog 2, seven for dog 3, five for dog 4, and six for dog 5.

Of course, betting and odds are a natural thing to add to any race, but since we don't want to encourage gambling, we'll leave that up to you!

Variables:

Y—finish line vertical limits
A—print location, dog 1
B—print location, dog 2
C—print location, dog 3
D—print location, dog 4
E—print location, dog 5
X—random number to advance one dog

PROGRAM 7
B–I–N–G–O
(4K, Level I or II)

You'll learn: Single-dimension array; READ-DATA statements; selecting nonrepetitive random numbers

Description: The numbers 1 through 75 are displayed on the screen in five columns of 15 consecutive numbers under B-I-N-G-O headings. Lines separate the columns to improve readability. The computer selects and displays random numbers and then puts an asterisk (*) on the screen next to each number when it is chosen (see the display format in Fig. 7–1).

You use regular BINGO cards and markers to play the game. The purpose of this program is just to call a different number each time and show the numbers that have already been chosen at any point in the game.

Why use a computer to pick and post numbers? It is faster, doesn't make posting errors, and creates some suspense in long games as it takes time to "seek" a number not previously chosen.

Explanation: This program is *almost* the same for Levels I and II. Let's discuss Level I first and then the minor changes necessary for Level II.

Lines 5–12 introduce the program on the screen. When there is sufficient memory, it's a good idea to do so. If the instructions are very long, they can be bypassed with a yes/no input statement such as the following, in line 3:

```
INPUT "INSTRUCTIONS? (1 = YES, 2 = NO)"; I
```

Of course, the proper GOTO statement must also be used, such as the following, in line 4:

```
IF I = 2 GOTO 13
```

In this case, since the program introduction fits on one screen-full ("page"), why not include it?

Line 13 stops the program with a request for a number from 5 to 25. When this number is entered, it is used as the random generator "seed" later in the program (line 23Ø).

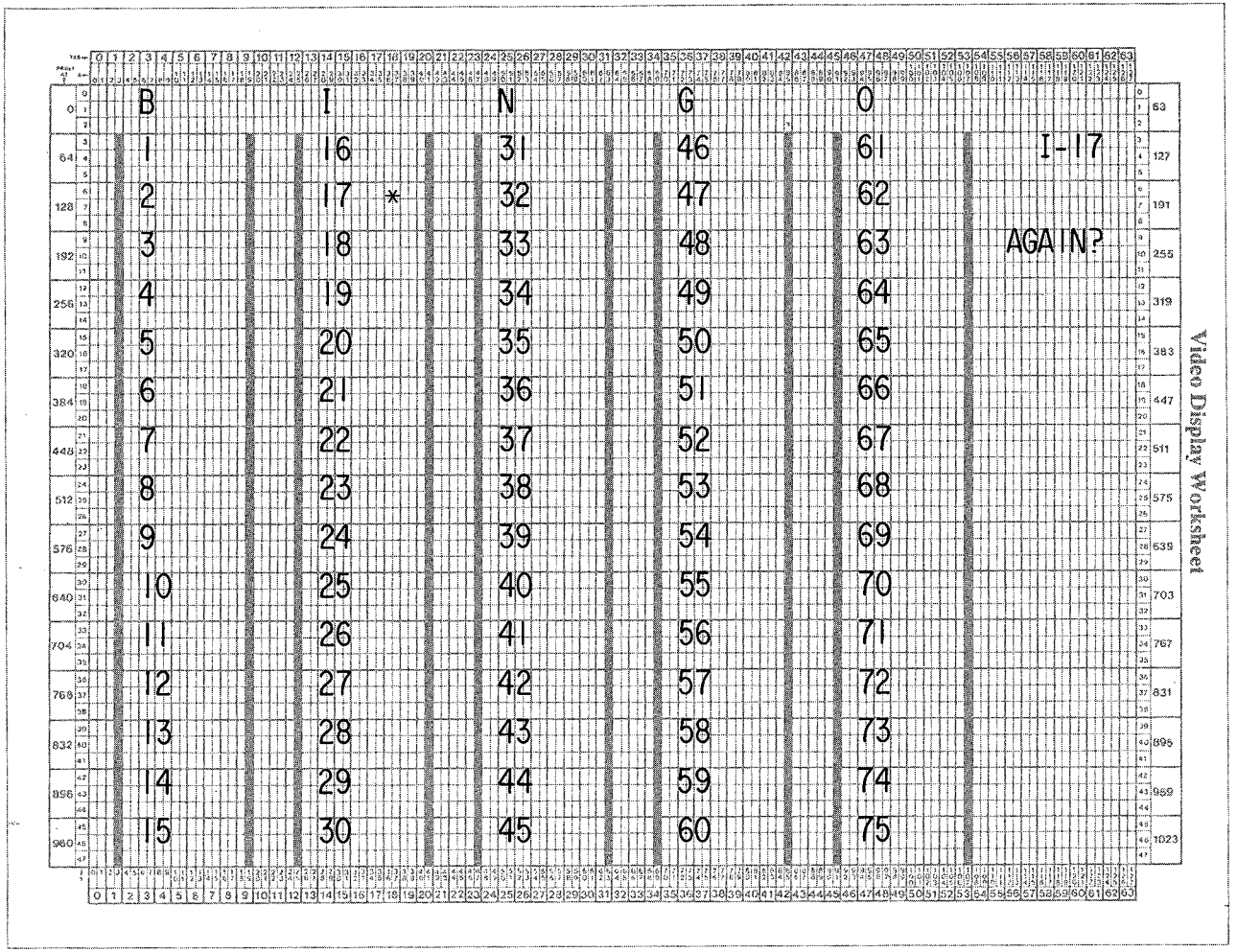


Fig. 7-1. Display format

LEVEL I LISTING

```
1  REM * COPYRIGHT FRED BLECHMAN 1978 *
2  REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
5  CLS:P. :P. "            BINGO NUMBER CALLER & VERIFIER"
6  P. :P. "      THIS PROGRAM WILL SELECT AND NOTE BINGO NUMBERS"
7  P. "FROM B-1 TO O-75, WITHOUT REPEATING ANY NUMBER.  THE *"
8  P. "NEXT TO A NUMBER IN THE TABLE INDICATES IT HAS BEEN CALLED. "
9  P. :P. "      WHEN THE COMPUTER IS LOOKING FOR A NUMBER NOT"
10  P. "ALREADY CHOSEN, IT WILL PRINT 'SEEK!' UNTIL IT FINDS ONE!"
11  P. :P. "      PLAYERS USE BINGO CARDS AND MARKERS IN THE"
12  P. "                  REGULAR WAY....."
13  P. :P. :IN. "TO PLAY, ENTER A NUMBER FROM 5 - 25...";R
14  REM * SET ARRAYS FOR NUMBERS AND ASTERISK POSITIONS *
15  CLS
16  M=101:N=115:S=71:T=S
17  FOR X=M TO N
18  A(X)=S
19  S=S+64
20  N. X
21  M=M+15:N=N+15:S=T+11:T=S
22  IF N=190 GOTO 24
23  GOTO 17
24  B=66:C=111
25  P. "   B           I           N           G           O"
35  D=1:A=D:G. 40
36  D=D+1:A=D
40  F. Y=B TO C STEP 11
60  P. AT Y, A;
70  A=A+15
80 N. Y
90  B=B+64:C=C+64
100  IFY>1010 G. 150
110  G. 36
140  REM * DRAW VERTICAL COLUMNS *
150  FOR Y=3 TO 47
155  FOR Z=1 TO 10
160  READ X
170  SET(X, Y)
175  N. Z
176  RESTORE
180  N. Y
190  D. 3, 19, 25, 41, 47, 63, 69, 85, 91, 107
195  REM * SET ARRAY 1-75 TO ZERO *
200  FOR X=1 TO 75
210  A(X)=0
220  N. X
225  REM * RANDOMIZE AND SELECT A NUMBER NOT ALREADY USED *
230  FOR X=1 TO R:N=RND(75):N. X
231  X=RND(75)
235  IF A(X)<>0 P. AT123, "SEEK!";
240  IF A(X)<>0 THEN 230
245  REM * PRINT NUMBER AND ASTERISK *
250  F. A=1 TO X
260  P. AT 123, A;
270  N. A
275  A=A-1
280  A(X)=A
285  P. AT A(100+A), "*";
286  IF X<16 P. AT122, "B-"; X; :G. 295
287  IF X<31 P. AT122, "I-"; X; :G. 295
288  IF X<46 P. AT122, "N-"; X; :G. 295
289  IF X<61 P. AT122, "G-"; X; :G. 295
290  P. AT122, "O-"; X;
295  P. AT248, " ";
300  IN. "AGAIN"; A$
305  P. AT248, "        ";
310  P. AT121, "        ";
320  G. 230
```

```
1  REM * COPYRIGHT FRED BLECHMAN 1978 *
2  REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
3 REM * TRS-80 LEVEL II *
4 DIMA(175)
5 CLS:PRINT"                BINGO NUMBER CALLER AND VERIFIER"
6  PRINT:PRINT"      THIS PROGRAM WILL SELECT AND NOTE BINGO NUMBERS"
7  PRINT"FROM B-1 TO O-75, WITHOUT REPEATING ANY NUMBER.  THE *"'
8  PRINT"NEXT TO A NUMBER IN THE TABLE INDICATES IT HAS BEEN CALLED."
9  PRINT:PRINT"      WHEN THE COMPUTER IS LOOKING FOR A NUMBER NOT"
10  PRINT"ALREADY CHOSEN, IT WILL PRINT 'SEEK!' UNTIL IT FINDS ONE!"
11  PRINT:PRINT"      PLAYERS USE BINGO CARDS AND MARKERS IN THE"
12  PRINT"                  REGULAR WAY....."
13  PRINT:PRINT:INPUT"TO PLAY, ENTER A NUMBER FROM 5 - 25...";R
14  REM * SET ARRAYS FOR NUMBERS AND ASTERISK POSITIONS *
15  CLS
16  M=101:N=115:S=71:T=S
17  FOR X=M TO N
18  A(X)=S
19  S=S+64
20  NEXTX
21  M=M+15:N=N+15:S=T+11:T=S
22  IF N=190 GOTO 24
23  GOTO 17
24  B=66:C=111
25  PRINT"   B          I          N          G          O"
35  D=1:A=D:GOTO40
36  D=D+1:A=D
40  FORY=B TO C STEP 11
60  PRINT@Y,A;
70  A=A+15
80 NEXTY
90  B=B+64:C=C+64
100  IFY>1010 GOTO150
110  GOTO36
140  REM * DRAW VERTICAL COLUMNS *
150  FOR Y=3 TO 46
155  FOR Z=1 TO 10
160  READ X
170  SET(X,Y)
175  NEXTZ
176  RESTORE
180  NEXTY
190  DATA3,19,25,41,47,63,69,85,91,107
195  REM * SET ARRAY 1-75 TO ZERO *
200  FOR X=1 TO 75
210  A(X)=0
220  NEXTX
225  REM * RANDOMIZE AND SELECT A NUMBER NOT ALREADY USED *
230  FOR X=1 TO R:N=RND(75):NEXTX
231  X=RND(75)
235  IF A(X)<>0 PRINT@123,"SEEK!";
240  IF A(X)<>0 THEN 230
241 PRINT@123,"      ";
245  REM * PRINT NUMBER AND ASTERISK *
250  FORA=1 TO X
260  PRINT@ 123,A;
270  NEXTA
275  A=A-1
280  A(X)=A
285  PRINT@A(100+A),"*";
286  IF X<16 PRINT@122,"B-";X; :GOTO295
287  IF X<31 PRINT@122,"I-";X; :GOTO295
288  IF X<46 PRINT@122,"N-";X; :GOTO295
289  IF X<61 PRINT@122,"G-";X; :GOTO295
```

```
290  PRINT@122,"O-";X;
295 PRINT@248,"AGAIN?";
300 A$=INKEY$:IF A$="" THEN 300
305 PRINT@248,"       ";
310 PRINT@121,"       ";
320  GOTO230
```

PROGRAM 7 RUN

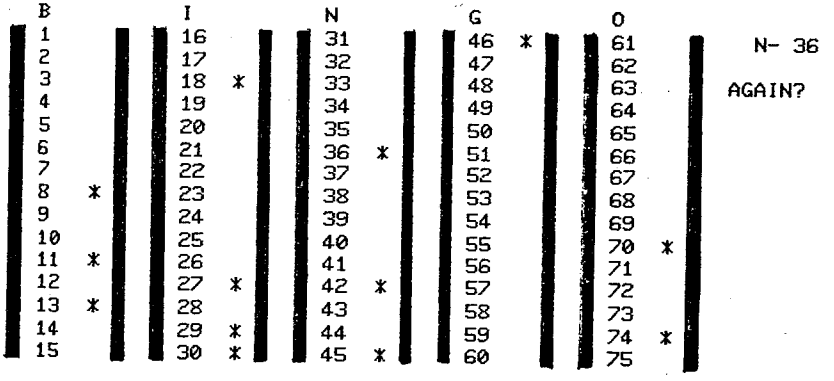
```
                BINGO NUMBER CALLER AND VERIFIER

     THIS PROGRAM WILL SELECT AND NOTE BINGO NUMBERS
FROM B-1 TO O-75, WITHOUT REPEATING ANY NUMBER. THE *
NEXT TO A NUMBER IN THE TABLE INDICATES IT HAS BEEN CALLED

     WHEN THE COMPUTER IS LOOKING FOR A NUMBER NOT
ALREADY CHOSEN, IT WILL PRINT 'SEEK!' UNTIL IT FINDS ONE!

     PLAYERS USE BINGO CARDS AND MARKERS IN THE
                  REGULAR WAY.....

TO PLAY, ENTER A NUMBER FROM 5 - 25....? 16
```



Line 15 clears the screen, and line 16 assigns values to variables M, N, S, and T. These variables put numbers into an "array" for later use. An array is merely a way of storing data in an organized fashion. Think of it as a bunch of empty boxes that can be stacked in the desired number of columns (up and down) and rows (across). In this program you can think of the array as being 35 rows high and 5 columns across for a total of 175 "array locations." We won't use all these locations, but in some we'll store data. First, we are going to use Array A, the only one available in Level I, to hold the printing locations of the 75 possible asterisks we may need later

in the program for numbers chosen. At this point, we'll ignore the first 100 array locations and start with number 101.

Since this is tricky, dig in your heels and let's go! Lines 17-20 are a FOR-NEXT loop using X as the counter, with limits from M (equals 101) to N (equals 115). In other words, we start out with X equal to 101. Line 18 sets array location A (101)—since X equals 101—to the value of S, which is 71 (line 16). This puts a value of 71 into "box" number 101 in our array.

The value of 71, in this case, occupies a print location in which we may need to print an asterisk later. This is a way of putting this print location in memory to be retrieved later, if needed.

Why a value of 71? You'll need to refer to Fig. 7-1 to make any sense out of what follows. The print locations start at 0 in the upper left corner and first go left to right, and then from top to bottom, with 64 spaces per line, and finally end with print location 1023 in the lower right corner. Print location 71 is eight spaces from the left edge on the second line. Later on, if the computer chooses number 1 when selecting random numbers, an asterisk will be printed at print location 71. This location number is now stored in A-array 101.

Line 19 increases the value of S by 64—one line on the display—to 135. Line 20 tells the computer to go back to line 17 and advance the loop counter, X, by 1 since no other STEP was specified in line 17. Thus, X is now equal to 102. Line 18 sets array location 102 to the new value of S, which is 135. Line 19 advances S again by 64 to 199, line 20 sends the program back to line 17, and the next value of X, 103, is established. The program continues this process until X is equal to 116, since line 17 set the limit of X to 115, the value of N set in line 16. What has all this accomplished? It has stored numbers in array locations 101-115. These numbers represent print locations along the left side of the screen, each eight spaces in from the left side, and one below the other. The program may call for these locations later (the array is where they're stored in an organized fashion so that they can be "found" later). Put another way, the program can be directed to a particular array location and use whatever number is stored there.

Confused? Read the last few paragraphs over again until you follow what's happening. Arrays are superpowerful in programming and can be used as variables—an especially important role for them in Level I where conventional variables are limited to A through Z.

Line 21 simply advances the array location limits by 15—think of it as the next column—and the print locations by 11, further

to the right. The variable T was initially set to equal the original value of S in line 16; now line 21 increases the original S by 11 to 82, and then T is set to the new value of S.

Line 22 checks to see if the new value of N is equal to 19Ø. Since N equals 13Ø at this point, the program continues to line 23, which sends the program back to that blasted FOR-NEXT loop in line 17! Take it easy, however; things aren't as bad as they appear. All that happens is that the computer now stuffs array locations 116–13Ø with 15 numbers starting at 82 and increasing by 64 to 978. If you look at Fig. 7–1, you'll see that these are print locations 19 spaces from the left side of the screen, one below the other, from the second to last row. These form the second column of asterisk locations that might be needed later.

Similarly, the program continues operating between lines 17 and 23 until N equals 19Ø (after five times through line 21); then the program jumps to line 24. By this time, A-array locations 1Ø1–175 are all filled with numbers.

So far, nothing has happened on the screen while A-array locations 1Ø1–175 were being loaded (a job that takes only a little over a second). Fortunately, even if you didn't understand a thing you just read about loading the array, the computer does, and fast!

Line 24 assigns values of 66 to B and 111 to C. These will be used in line 4Ø. Line 25 prints B I N G O spaced across the screen as column headings on the first line. Be very careful about the spacing on this line. Leave three spaces after the starting quotation mark and 10 spaces between letters.

Line 35 sets D equal to 1 and A equal to D and then sends the program to line 4Ø, skipping over line 36 at this point. This process of establishing values for variables is called "initialization." Now line 4Ø sets up another FOR-NEXT loop (through line 11Ø). This one will print the numbers 1 through 75 on the screen in five columns of 15 consecutive numbers. Line 4Ø starts with Y equal to 66 (initialized in line 24). Line 6Ø prints the value of A (initialized in line 35 as 1) at print location Y, which is 66. However, since the TRS-80 leaves a blank in front of numbers (for a possible plus or minus sign), the number 1 is actually printed at location 67. Line 7Ø increases the value of A by 15 to 16, and line 8Ø sends the program back to line 4Ø. Here, the value of Y is increased from 66 to 77, since the end of line 4Ø says STEP 11. Now line 6Ø prints the number 16 at print locations 77 and 78 (actually, because of the blank space, 78 and 79). Then A increases by 15 again, and so forth, printing the numbers 31, 46, and 61 across the second screen line, 11 spaces apart. However, the next execution of line 4Ø sets the value of Y to

121. Since this is greater than the value of C (set at 111 in line 24), the program moves on to line 9Ø, where B and C are both increased by 64 to move the printing locations down one line. Line 1ØØ inspects the value of Y to see if it exceeds 1Ø1Ø. Since it doesn't, line 11Ø sends the program to line 36. Here D is advanced by 1 to a value of 2, and A is set equal to the new D. Now the program goes through the FOR-NEXT routine again, printing the numbers 2, 17, 32, 47, and 62 across the next screen line. Do you see what's happening? The screen is being printed from left to right with five columns of numbers (which read from top to bottom).

This looping continues until all 75 numbers are on the screen. The last one is printed at screen location 1ØØ7 (Y = 1ØØ6). Since Y next becomes greater than 1Ø1Ø, line 1ØØ sends the program to line 15Ø.

Lines 15Ø–19Ø draw 1Ø lines down the screen to separate the 75 numbers into five distinct columns. It does so with the SET command and DATA statements. We've run into the SET command before, but DATA statements are a new, and extremely handy, programming tool. They will be used extensively in some later programs.

Line 15Ø sets up a Y-variable FOR-NEXT loop, with values for Y of from 3 to 47. Line 155 uses Z in a FOR-NEXT loop to count from 1 to 1Ø. Line 16Ø says READ X. What does that mean? It means that somewhere in the program there's a program line that starts with DATA (or D.). Conveniently, it is right nearby—at line 19Ø. Actually it could be anywhere in the program.

Now that the computer has found DATA, it reads the first value of that line (in this case, the number 3) and sets the variable X (since line 16Ø said READ X) equal to that value. So X equals 3. Line 17Ø says to use this value, together with the current value of Y (which is also 3, from line 15Ø), and to set (light) that point on the screen. Now line 175 says to go back to the Z FOR-NEXT statement (line 155) and increase Z to 2. Line 16Ø reads the next DATA value (19) and assigns this value to X, and line 17Ø lights point (19,3) on the screen (X equals 19; Y is still equal to 3). Check Fig. 7–1 to see where these X and Y positions are located; they are the *small* numbers on the chart.

This process continues until all 1Ø DATA numbers have been used. At this point, Z becomes 11, so that line 175 falls through to line 176, RESTORE. RESTORE is another new command; it simply means to put the DATA numbers back in the DATA memory as if they had never been "used." Thus, line 19Ø is "restored" with all the values shown.

Line 18Ø sends the program back to line 15Ø, where Y is increased to 4. Now 10 more spaces are lighted across the screen at X locations 3, 19, 25, 41, 47, 63, 69, 85, 91, and 107 and Y location 4.

This whole process keeps drawing 10 lines down the screen until Y equals 48 and Z equals 11. The program then moves on to lines 2ØØ–22Ø, another FOR-NEXT loop. This one is simple. It quickly sets the values in A-array locations 1–75 equal to zero. You'll see why later.

Let's pause for a moment to see what's happened so far. The screen shows 10 vertical lines, with the numbers 1 through 75 in columns of 15 numbers, with each column headed by a letter. Unseen, but in the A-array memory, array locations 1–75 are all set at zero, and array locations 1Ø1–175 contain asterisk print locations.

It's finally time for the computer to choose a number from 1 to 75. But first, the simple FOR-NEXT loop in line 23Ø "seeds" the random-number sequence R times (almost instantly). R is the number you entered back in line 13. This precaution makes it unlikely that games starting from computer turn-on will choose the same numbers (unless you make R the same).

Line 231 selects a number from 1 to 75. Lines 235 and 24Ø check to see if this number has been selected previously by looking at the value stored in the array location corresponding to that number. If the value is zero, then this is a "new" number, and the program falls through to line 25Ø, which starts yet another FOR-NEXT loop. The purpose of lines 25Ø–27Ø is to print numbers near the upper right corner of the screen, starting with number 1, and increasing by 1 up to, and stopping at, the randomly chosen number, X. The variable A is used, and the FOR-NEXT loop is completed when A is equal to X plus 1 in line 27Ø. Line 275 brings the value of A back to the chosen number, X. Now line 28Ø inserts this value in A-array location X so that that location is no longer zero.

For example, suppose that the random number X chosen by the computer in line 231 is 17. Since all A-array locations 1–75 were previously set at zero (lines 2ØØ–22Ø), then lines 235 and 24Ø are ignored. Lines 25Ø–27Ø print at location 123 (actually, at location 124 because of the blank) first the number 1, then 2, then 3, etc., up to 17. Line 27Ø sets the value of A at 18, which stops the FOR-NEXT loop (since A is now larger than X in line 25Ø), and line 275 resets the value of A to 17. Line 28Ø inserts the value 17 into A-array location 17.

The next time the number 17 is chosen, it will be "caught" by lines 235 and 240. The word SEEK! will be printed (however briefly) and another random number chosen. You do see why, don't

you? Because that array location is not zero anymore, as a result of line 28Ø.

Another thing happens when the program moves on to line 285. The asterisk we mentioned earlier is printed on the screen to the right of the chosen number. The printing location is the value held in the A-array location corresponding to the number chosen plus 1ØØ. (Remember that we stored the asterisk print locations in array locations 1Ø1-175.) In this example, since A is 17, the computer looks in array location 117 and finds the number 146, then prints an asterisk at print location 146—three spaces to the right of the 7 in number 17. It's like magic!

Next, lines 286-29Ø determine if the number chosen should be preceded by a B, I, N, G, or O as a column designator. They do so simply with IF LESS THAN inspections, PRINT AT, and GOTO. Obviously, as soon as the proper prefix is found, it is printed with a hyphen just before the number selected, and the program jumps to line 295. Are you puzzled by line 295? This prints a blank two lines below the selected number, and the semicolon establishes the printing position of AGAIN? as called by the input statement of line 3ØØ, which halts the program. When ready for another number, press ENTER. Lines 3Ø5 and 31Ø clear the printing in the upper right corner, and the program then goes back to line 23Ø to pick another number.

You can continue doing this until all 75 numbers are chosen. As you begin to fill up the display with asterisks, the word SEEK! will tell you that the computer is frantically looking for a previously unselected number, and this search can take several seconds.

Press BREAK to exit the program.

The Level II version of this program removes most of the abbreviations, adds line 4, and changes lines 295 and 3ØØ. Line 4 adds the DIM statement needed by Level II whenever more than 1Ø array locations are used to "dimension" the array. In this case, line 4 tells the computer that the program will use as many as 175 A-array locations. The DIM statement is not used in Level I.

A peculiarity of Level II is that an input query, such as line 3ØØ of the Level I program, wipes out the entire next line of the display. In Level II you should thus use a print location to ask the question and the INKEY$ function, as shown in line 3ØØ of the Level II program, to make the computer wait for your response. Almost *any* keyboard entry—not just ENTER—allows the program to continue.

Modifications: With a graphic subroutine, you could print the chosen number larger on the screen, but this trick is not exactly for beginners (translation: I don't know how!).

Variables:

A (X)—A-array location
R—random number seed
M—asterisk array location (column start)
N—asterisk array location (column end)
S—asterisk print location
T—asterisk location shift
X—array counter; horizontal SET coordinate; random number
B—number print location (start)
C—number print location (end)
D—number advance
A—numbers 1 through 75
Y—number print location; vertical SET coordinate
Z—READ-DATA counter
A$—select number command

PROGRAM 8
MY $600 DIGITAL CLOCK!
(4K, Level I or II)

You'll learn: Setting timing loop duration; counting to a limit

Description: This program displays hours, minutes, and seconds on the screen in a 12- or 24-hour format. You set the starting time from the keyboard and can adjust the running speed within the program.

Explanation: With only minor changes, this program will run in Level I or Level II. The listing shown is in Level II language and needs the following changes for Level I:

Lines 168 and 175: Change PRINT@ to PRINTAT
Line 17Ø: Change 34Ø to 45Ø

You can also use the normal abbreviations for Level I, such as P. for PRINT and I. for INPUT.

Keyboard in the program; then type RUN and ENTER. The REM statements, lines 96–1Ø3, provide information not displayed on the screen. It is common for programmers to imbed nonessential information in REM statements so that if you're having difficulty running or understanding a program, it's always a good idea to LIST the program and look for REM statements that could provide additional information.

Lines 11Ø–123 clear the screen and print instructions near its top. When you press ENTER, the program continues on line 125, where the PRINT command prints a blank line (since this is totally unnecessary, it must have been "left over" from program development). Next, H, M, S, F, and X are set to zero in Level I but not in Level II since the RUN command sets all variables to zero in Level II but not Level I. If you had run a previous program in Level I using these variables, they would be held in the computer memory at the last value set. Line 13Ø defines the meanings of H, M, S, and F.

Line 131 clears the screen, and the three PRINT commands print three blank lines on the screen. This has the effect of moving

```
96 REM * DIGITAL CLOCK PROGRAM *
97 REM * TRS-80 LEVEL I OR LEVEL II *
98 REM * SET CLOCK SPEED AT LINE 170 *
99 REM * APPROXIMATE COUNT FOR LEVEL I IS 450; FOR LEVEL II 340*
100  REM * COPYRIGHT FRED BLECHMAN 1978 *
101  REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
103  REM * CLOCK SPEED IS SET AT LINE 170 *
110  CLS:PRINT
120  PRINT"                  MY $600 DIGITAL CLOCK!"
121  PRINT:PRINT:PRINT"      SET THE HOURS, MINUTES AND SECONDS YOU WANT"
122  PRINT"AS A STARTING TIME, IN EITHER 12 OR 24 HOUR FORMAT...."
123  PRINT:PRINT:INPUT"WHEN READY TO SET STARTING TIME, HIT 'ENTER' ";A$
125  PRINT:H=0:M=0:S=0:F=0:X=0
130  REM * LET H=HOURS,M=MINUTES,S=SECONDS,F=FORMAT *
131  CLS:PRINT:PRINT:PRINT
135  INPUT"12 OR 24 HOUR FORMAT";F
136  IF (F<>12)*(F<>24) PRINT"INVALID ENTRY! TRY AGAIN":GOTO 135
140  INPUT"STARTING HOURS";H
145  IF F=12 THEN IF H=0 PRINT"INVALID ENTRY! TRY AGAIN":GOTO140
146  IF F=12 THEN IF H>12 PRINT"INVALID ENTRY! TRY AGAIN":GOTO140
147  IF H>23 PRINT"INVALID ENTRY! TRY AGAIN":GOTO 140
150  INPUT"STARTING MINUTES";M
155  IF M>59 PRINT"INVALID ENTRY! TRY AGAIN":GOTO 150
160  PRINT:PRINT"THE CLOCK WILL START COUNTING WITH THE NEXT ENTRY...."
164  PRINT:INPUT"STARTING SECONDS";S
165  IF S>59 PRINT"INVALID ENTRY! TRY AGAIN":GOTO 164
166  CLS
167 PRINT:PRINT:PRINT"            MY $600 ";F;"HOUR DIGITAL CLOCK!"
168 PRINT@404,"HOURS","MINUTES","SECONDS"
169 GOTO180
170 FOR X=1 TO 340:NEXT X
175 PRINT@468,H,M,S;
180  S=S+1
190  IF S=60 THEN M=M+1
200  IF S=60 THEN S=0
210  IF M=60 THEN H=H+1
220  IF M=60 THEN M=0
225  IF H=24 THEN H=0
226  IF F=24 GOTO 250
230  IF H=13 THEN H=1
250  GOTO 170
```

the first printed line down from the very top of the screen. Line 135 asks you to enter 12 or 24 to define the time format. Most people in the United States use the familiar 12-hour AM or PM method of describing time. However, the military, and many foreign countries, use the 24-hour format, where "0" hours is midnight, "12" hours is noon, and "15" hours, for example, is 3 PM. Figure 8-1 shows this relationship. Minutes and seconds are the same in both systems.

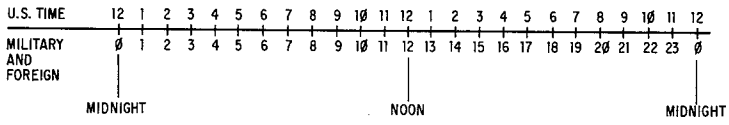


Fig. 8-1. Time relationship

PROGRAM 8 RUN

```
              MY $600 DIGITAL CLOCK!

    SET THE HOURS, MINUTES AND SECONDS YOU WANT
AS A STARTING TIME, IN EITHER 12 OR 24 HOUR FORMAT....

WHEN READY TO SET STARTING TIME, HIT 'ENTER' ?



12 OR 24 HOUR FORMAT? 12
STARTING HOURS? 7
STARTING MINUTES? 34

THE CLOCK WILL START COUNTING WITH THE NEXT ENTRY....

STARTING SECONDS? 15
          MY $600  12 HOUR DIGITAL CLOCK!



          HOURS          MINUTES          SECONDS
            7              34               21
```

Line 136 is an entry-error trap to prevent you from entering any number other than 12 or 24. If you do, the program prints the error message and sends you back to line 135.

Line 140 asks for the starting hours. If you establish that you want the 12-hour format in line 135 (variable F), then you must enter a number from 1 to 12 here. If you choose a 24-hour format, then the number you enter here cannot be greater than 23. Lines 145–147 check your entry to be sure you haven't "goofed"; they won't accept an improper entry.

Lines 150 and 155 allow you to set starting minutes, rejecting anything over 59 since 60 and over are invalid entries.

Line 160 prints a blank line and then the warning that counting starts with the next entry. Line 164 prints a blank line and asks for starting seconds. Enter a number a few seconds ahead of actual time, and then press ENTER when the two coincide. Line 165 checks to be sure you didn't enter a number over 59. Line 166 clears the screen. Line 167 prints two blank lines and then the title on the third line from the top of the screen. Line 168 prints headings starting at print location 404. Since the commas place the printed headings at the beginning of printing "zones" located 16 spaces apart, don't forget them. Line 169 sends the program to line 180.

Line 180 adds 1 to S. Line 190 checks the new value of S to see if it has reached 60. If it has, 1 is added to the value of M, since

60 seconds equal 1 minute. Line 200 then sets S equal to zero if it has reached 60. Therefore, S goes from zero to 59, which represents 60 counts.

Line 210 looks at the new value of M to see if it has reached 60, since there are 60 minutes in an hour. If it has, then 1 is added to H, and line 220 sets M equal to zero.

Line 225 inspects the new value of H. If it has reached 24, then it is reset to zero. Line 226 examines the value of F to see if you asked for a 12- or 24-hour format. If you did, the program jumps to line 250. If not, line 230 checks to see if it equals 13. Since, in the 12-hour mode, the hour after 12 is 1, not 13, H is set equal to 1 if H has reached 13.

Line 250 sends the program back to line 170.

Line 170 is a timing loop to insert a delay so that all processing from line 170 to line 250 takes 1 second. This timing loop is different for Levels I and II. Surprisingly, Level II loops at a slower rate; that is, each loop takes longer than the same loop in Level I. As a result, the TRS-80 in Level I must count to a higher number—around 450. Changes in this number (450 in Level I, 370 in Level II) determine the counting accuracy. Even a change of 1 count, such as changing 370 to 369 or 371, will have an observable effect after about 30 seconds if you compare the seconds-counting with an accurate clock or chronograph.

You may have to experiment a while with the number you use in line 170 in order to get an exactly 1-second time loop from lines 170 to 250. The proper number depends on the running speed of your particular machine as well as on temperature and humidity. Unfortunately, the TRS-80 was not designed to be a precision timepiece!

Line 175 prints the values of hours, minutes, and seconds under the respective headings, properly spaced by the commas. Lines 180-250 have already been described.

The process of adding seconds and inspecting, changing, and printing values for S, M, and H continues until you press the BREAK key.

Modifications: You could use graphic commands (SET or CHR$) to draw boxes around the numbers, making the display look fancy. You could even use subroutines and the CHR$ command to create large graphic digits and call these subroutines from the program for each digit. To do so adds considerable complexity to an otherwise simple program, without increasing accuracy.

Variables:

A$—any keyboard entry to continue program
H—hours
M—minutes
S—seconds
F—12- or 24-hour format designator
X—timing loop counter

PROGRAM 9
MY $600 ADDING MACHINE!
(4K, Level I or Level II)

You'll learn: Column placement; simple branching; number accumulation

Description: This is an example of "computer overkill," but if you don't have a "printing" calculator or adding machine, it will do the job!

This program adds and subtracts, displaying the amounts one below the other. You ENTER a "Ø" to get a subtotal and a "1" to continue with some more numbers. ENTER a "Ø" after a subtotal and you will get a total. You can then use the program again or exit the program.

The only change in the listing to make it suitable for Level I occurs in line 62Ø. Use PRINTAT instead of PRINT@.

Explanation: This program illustrates a number of simple techniques that will be used in later programs. Its primary purpose is to serve as an introduction to adding numbers and formatting their display.

Notice that the first line number is 5ØØ. Where is it written that you must start a program at line 1Ø or 1ØØ? Although most programmers start with low line numbers, you can start anywhere within the line number range of the computer (32767 for Level I, 65529 for Level II).

Lines 5ØØ–5Ø2 identify the programmer, computer language, and memory. It's a good idea to incorporate this feature in every program. If you keyboard in a program designed for another version of BASIC, it may not work on your TRS-80. Also, some programs that look short enough to run in 4K may use large arrays or lots of "strings," and they may require a larger memory. Line 5Ø2 makes it clear that this program will operate with a TRS-80 Level I or Level II machine with a 4K memory. Line 5Ø8 starts out with CLS for "clear screen." Although we've used this statement in earlier programs, we haven't talked about it. CLS is one of the most

```
500  REM * COPYRIGHT FRED BLECHMAN 1978 *
501  REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
502 REM * TRS-80 LEVEL I OR LEVEL II 4K *
508  CLS:PRINT:PRINT
509  PRINT"            MY $600 ADDING MACHINE":PRINT
510  PRINT:PRINT"     THIS PROGRAM DISPLAYS AND ADDS OR SUBTRACTS"
511  PRINT"AND GIVES YOU SUBTOTALS AND TOTALS.  JUST FOLLOW THE"
512  PRINT"INSTRUCTIONS.  REMEMBER, TO GET A SUBTOTAL, JUST ENTER"
513  PRINT"A ZERO AND FOR THE GRAND TOTAL, ENTER ZERO AGAIN...
514  PRINT:PRINT:PRINT:INPUT"WHEN YOU'RE READY TO START, HIT 'ENTER' ";A$
515  CLS
520  PRINT:PRINT"ENTER EACH AMOUNT.  USE A NEGATIVE SIGN FOR MINUS"
521  PRINT"NUMBERS.  ENTER 0 FOR SUBTOTAL....."
525  B=0:PRINT
530  PRINT"                    ";:INPUTA
535  IF A=0 GOTO 560
540  B=B+A
550  GOTO 530
560  PRINT:PRINT"THE SUBTOTAL IS";B
565  PRINT
570  INPUT"TO CONTINUE, ENTER 1.  FOR FINAL TOTAL, ENTER 0 ";C
580  IF C=1 GOTO 530
590  PRINT:PRINT"THE FINAL TOTAL IS";B
595  PRINT
600  INPUT"AGAIN? YES=1, NO=0 ";D
610  IF D=1 GOTO 515
620  CLS:PRINT@404,"GLAD TO HELP.  GOODBYE.....":PRINT:PRINT:PRINT
630  END
```

powerful formatting statements in BASIC. What? A simple thing like clearing the screen is "powerful?" Yes! It accomplishes several things: (1) You start anew with a blank screen, thus eliminating previous distracting text, tabulations, graphics, or whatever; (2) you have the upper left corner of the screen as a new "starting point" for whatever you wish to display next; and (3) you can highlight what comes next, since there's nothing else on the screen. It's just like starting a new page in a book. Properly used, CLS will make your programs easier to understand and use.

The two PRINT statements in line 5Ø8 print blank lines so that the PRINT statement in line 5Ø9 will not be at the very top of the screen. When formatting instructions, such as those on lines 51Ø–514, place them somewhere near mid-screen. Notice how the ending PRINT on line 5Ø9 and the first PRINT on line 51Ø "skip" two lines. Also notice how the three PRINTs at the beginning of line 514 move the "ready to start" prompt to the fourth line below the instructions. While these formatting considerations make no difference in program operation, they do create a more professional appearance. Furthermore, there are times when knowing how to space lines properly can prevent unwanted erasure of those already on the screen or allow you to intentionally erase those no longer needed.

PROGRAM 9 RUN

```
            MY $600 ADDING MACHINE


     THIS PROGRAM DISPLAYS AND ADDS OR SUBTRACTS
AND GIVES YOU SUBTOTALS AND TOTALS. JUST FOLLOW THE
INSTRUCTIONS. REMEMBER, TO GET A SUBTOTAL, JUST ENTER
A ZERO AND FOR THE GRAND TOTAL, ENTER ZERO AGAIN....


WHEN YOU'RE READY TO START, HIT 'ENTER' ?



ENTER EACH AMOUNT. USE A NEGATIVE SIGN FOR MINUS
NUMBERS. ENTER 0 FOR SUBTOTAL.....

                    ? 123.45
                    ? 53.36
                    ? -17.65
                    ? 0

THE SUBTOTAL IS 159.16

TO CONTINUE, ENTER 1. FOR FINAL TOTAL, ENTER 0 ? 1



                    ? 14.78
                    ? -34.56
                    ? 456.66
                    ? 0

THE SUBTOTAL IS 596.04

TO CONTINUE, ENTER 1. FOR FINAL TOTAL, ENTER 0 ? 0

THE FINAL TOTAL IS 596.04

AGAIN? YES=1, NO=0 ? 0

                  GLAD TO HELP. GOODBYE.....
```

Now that we've covered these general points, let's get back to the purpose of the program—to add and display numbers in a column. When you press ENTER, line 515 clears the screen again, and lines 52Ø and 521 summarize the instructions near the top of the screen. Line 525 assigns a value of Ø to variable B and skips one line. Line 53Ø prints a series of blank spaces that move the following INPUT prompt "?" almost halfway across the screen. Why? Because it looks better than having the question marks and numbers you enter hugging the left side of the screen. The number of blank

spaces in line 53Ø determines the spacing across the screen. Don't overlook the semicolon and colon before INPUT A. Incidentally, you could also change the word PRINT in line 53Ø to INPUT and delete the colon and the word INPUT near the end of the line (but leave the semicolon and A) with no change in operation. Even BASIC frequently allows more than one way "to skin a cat."

The INPUT A statement in line 53Ø stops everything until you make a keyboard entry of a number. Type 123.45 and then ENTER, for example. Line 535 checks to see if you entered a zero; if you did, the program jumps to 56Ø to give you a subtotal. However, since you typed in a number other than zero (even a negative number is okay), the program moves on to line 54Ø. Here the peculiar notation B=B+A appears. To anyone familiar with algebra, this equation is ridiculous. However, to your friendly TRS-80 BASIC interpreter, it simply means "make B a new value equal to the present value of B plus the present value of A." Since B was set to zero in line 525 and has not changed in value until now, and since you entered a value of 123.45 for A, the computer assigns a new value to B of Ø plus 123.45, or 123.45.

Line 55Ø sends the program back to line 53Ø, which asks for another INPUT A with a question mark. Let's say that you type in 53.36 and then ENTER. Since you didn't enter a zero, line 535 is ignored. Line 54Ø now establishes a new value for B of 123.45 plus 53.36, or 176.81. Although this number doesn't show on your screen, the computer holds the value for variable B in its memory.

Once again the computer goes back to line 53Ø and displays a question mark on the next line to await your entry. Type and enter −17.65. Line 535 is ignored, but line 54Ø calculates a new value for B again. This time it's the last value of B (176.81) plus a negative 17.65, for a result of 159.16 that is held in memory for the new value of B.

You can do this as many times as you like, with numbers scrolling up and off the screen after the fifteenth entry. However, we'll make the next entry a zero. Now line 535 jumps the program to line 56Ø, where a blank line is printed, followed by "THE SUBTOTAL IS";B. Finally, here's where the variable B comes out of hiding. The value printed is, of course, 159.16.

Line 565 skips a line. Line 57Ø gives you a choice of continuing by entering 1 or getting a final total by entering Ø. Up to 11 previous entries will still be displayed. If you enter 1, line 58Ø sends the program back to line 53Ø for another entry. If you enter Ø, line 59Ø skips a line and prints the final total, B.

Line 595 skips a line. Line 600 gives you an opportunity to add a whole new series of numbers from zero, and line 610 sends the program back to line 515. If you don't want to go again, the computer politely exits the program in lines 620 and 630.

While much of the above is very elementary and quite basic to computer programming, its very simplicity tends to make it overlooked in many texts; consequently, we've covered it here in detail. This is, after all, a beginners' book!

Modifications: Programs 10 and 11 show how the basic column adding technique can be expanded to many columns and how more totals can be held in memory. As an exercise, try to develop your own multiplication or division program but don't try dividing by zero!

Variables:

A$—start program
B—total of entries
A—this entry
C—select continue or total
D—select repeat or exit program

PROGRAM 10
ORDER FORM TOTALS
(4K, Level I or II)

You'll learn: Multiple column formatting; bottom-of-screen prompting; line erasing; print formatting.

Description: With this program you can add four columns of figures, such as are used in ordering or invoicing forms. The figures are displayed in their respective columns under headings. The entry prompt appears at the bottom of the screen, and the entries print on lines one below another on the screen. After the eleventh line, the figures scroll upward with each new line.

When totals are requested, a line is drawn beneath the last entries, totals for each column are displayed, tax and handling are added, and a grand total is shown.

Explanation: Lines 95–1Ø7 are introductory REM statements that *don't* appear on the screen (except when the LIST command is used) and program instructions that *do* appear on the screen when the RUN command is used. The INPUT and A$ on line 1Ø7 hold the instructions on the screen until ENTER is pressed.

Line 1Ø8 clears the screen, and line 11Ø prints column headings on the top line. Other headings, such as "WHOLESALE," "DISCOUNT," "DEALER COST," "SUGG. RETAIL," or "AREA#1," "AREA#2," "AREA#3," "AREA#4," can be used to suit your needs. The important thing to know is that print zones, each 15 characters wide, are being used because of the commas between the quotation marks. Therefore, don't use column headings with more than 15 characters, or you'll bump into the next zone.

You might also wonder why PRINT@Ø is used, instead of just PRINT. Since the screen was cleared in the previous line, either will work.

Line 111 sets variables P, B, C, and R to Ø and variable X to 64. Any letters can be used for these variables, but it's always a good idea, when you can, to use variable letters that bear some relation to the use. For example, when this program was originated, P was

```
95 REM * TRS-80 LEVEL I OR LEVEL II 4K *
96  REM * THIS PROGRAM TOTALS FOUR COLUMNS *
97  REM * TAX AND HANDLING CAN BE CHANGED OR OMITTED *
98  REM * COPYRIGHT FRED BLECHMAN 1978 *
99  REM * 7217 BERNARDINE AVE., CANOGA PARK, CA  91307 *
100  CLS
101  PRINT:PRINT"                    ORDER FORM TOTALS"
102  PRINT:PRINT"  ENTER THE FIGURES FOR EACH OF THE COLUMNS ACROSS"
103  PRINT"EACH LINE OF THE ORDER FORM AS REQUESTED, WITH A COMMA"
104  PRINT"BETWEEN EACH FIGURE. WHEN YOU'VE REACHED THE END OF THE"
105  PRINT"FIGURES, ENTER 0,0,0,0 FOR TAX, HANDLING AND TOTALS."
106  PRINT"    (TAX AND HANDLING CALCULATED IN LINES 240-260)"
107  PRINT:INPUT"WHEN READY TO START, PRESS ENTER...";A$
108  CLS
110  PRINT@0,"COLUMN 1","COLUMN 2","COLUMN 3","COLUMN 4"
111  P=0:B=0:C=0:R=0:X=64
115 PRINT@832,"                                                "
116 PRINT@896,"                                                ";
121 PRINT@896," ";
125  INPUT"ENTER COL.1,COL.2,COL.3,COL.4";E,F,G,H
126  IF G=0 GOTO 200
127  IF X=768 GOTO 136
130  PRINT@ X,E,F,G,H
135  X=X+64: GOTO 140
136  PRINT:PRINT@ 704,E,F,G,H
140  P=P+E:B=B+F:C=C+G:R=R+H
160 GOTO115
200 PRINT@896,"                                                "
210 PRINT@X,"------------------------------------------------------------"
230  PRINTP,B,C,R
240  T=.06*R
250  S=.02*R
260  M=C+T+S
265  PRINT"          6% SALES TAX....";
270  PRINTTAB(33);T
276  PRINT"          HANDLING........";
280  PRINTTAB(33);S
290  PRINT"          GRAND TOTAL.....";
300  PRINTTAB(31);M
310  PRINT:INPUT"ANOTHER ORDER?(YES=1,NO=0)";A
320  IF A=1 THEN 108
330  PRINT:PRINT"SEE YOU ANOTHER TIME. 'BYE...."
340  END
```

chosen as a variable because the first of the four columns to be added was "points." Since the second column was "bonus," a B was used. C stood for "cost," and R for "retail."

Lines 115 and 116 are used to clear the screen for 47 spaces from the beginning of screen locations 832 and 896. These locations are the beginning of screen lines 14 and 15 (see Appendix A). While this area of the screen is clear right now, it won't be for long, and lines 115 and 116 will be used again later. The semicolon at the end of line 116 is not required.

Line 121 merely serves to put the printing cursor back at location 896 and prints one blank space. Since the semicolon prevents cursor movement, the INPUT statement prints at that

PROGRAM 10 RUN

```
                ORDER FORM TOTALS

  ENTER THE FIGURES FOR EACH OF THE COLUMNS ACROSS
EACH LINE OF THE ORDER FORM AS REQUESTED, WITH A COMMA
BETWEEN EACH FIGURE. WHEN YOU'VE REACHED THE END OF THE
FIGURES, ENTER 0,0,0,0 FOR TAX, HANDLING AND TOTALS.
    (TAX AND HANDLING CALCULATED IN LINES 240-260)

WHEN READY TO START, PRESS ENTER...?


COLUMN 1        COLUMN 2        COLUMN 3        COLUMN 4
 2.15            3.25            2.67            3.8
 2.95            4.5             3.38            4.95
 4.85            7.75            5.99            8.7
 10.45           18.9            16.72           20.5
-------------------------------------------------------
 20.4            34.4            28.76           37.95
        6% SALES TAX....          2.277
        HANDLING.......           .759
        GRAND TOTAL.....        31.796

ANOTHER ORDER?(YES=1,NO=0)? 0

SEE YOU ANOTHER TIME. 'BYE....
```

location. You now type in four numbers, separated by commas, and press ENTER. The program holds these four numbers as variables E, F, G, and H, in the same order as entered.

Line 126 looks to see if the third number, assigned to variable G, is zero. If it is, the program jumps to line 200 for totaling. Since we're not ready for that yet, we'll assume you typed in some number (positive or negative) for the third entry, and line 126 is ignored. Line 127 checks the value of X to see if it is 768. Since it isn't, having been set to 64 in line 111, the program goes right to line 130.

After all this investigation, line 130 finally does something. It prints, at screen location X (which is 64 at this time), the values of E, F, G, and H. The commas insure that these values will be printed under the column headings. This practice is called "zone printing." Line 135 adds 64 to the present value of X for a new total of 128, and the program branches to line 140.

Here's some more of that "new math" you'll get used to in BASIC programming. Line 140 simply takes the existing values for each column and adds them together to get new values. You might think of P, B, C, and R as the accumulated values, and E, F, G, and H as the last added values.

Line 160 sends the program back to line 115 for another row of entries. (Vertical lines on the screen are called "columns," and horizontal lines are called "rows.") Now you can see that lines 115

and 116 are necessary. They erase the printing created by line 125. Actually, line 115 isn't really necessary until the screen fills with enough rows to scroll upward with each new set of entries. If you substitute REM on this line, you'll see the effect after eleven rows of entries. In Level II, line 116 also isn't necessary, although it is in Level I. Level I and Level II do not handle print formatting exactly the same way.

As you keep making entries in response to line 125 and the program keeps adding 64 to the value of X in line 135, X equals 768 after the eleventh go-around and line 127 branches the program to line 136. The value of X no longer increases, and all further rows of inputs start at the twelfth screen line (location 7Ø4). The PRINT statement at the beginning of line 136 causes the entire screen to move up one line so that nothing is erased as the new line is printed at location 7Ø4.

When you want a total, enter Ø, Ø, Ø, Ø (or Ø,,,). Now line 126 sees G equal to zero and jumps the program to line 2ØØ. The last prompt is erased by line 2ØØ. Line 21Ø prints a dashed line right below the last entry, and the last values of P, B, C, and R—the column totals—are printed below the line.

Lines 24Ø–26Ø calculate tax and shipping charges. The variable T, which stands for "tax," is shown as 6 percent of the "retail," R. The variable S, which stands for "shipping" or handling, is shown in line 25Ø as 2 percent of the retail. Obviously, you can make this a different percentage, and use cost or any other basis you desire. Line 26Ø simply adds the cost, tax, and shipping for the grand total, M.

Lines 265–3ØØ title and print the tax, handling, and total. The PRINTTAB statements put the results under the third column. This position can be changed to suit your needs.

Lines 31Ø–34Ø allow you to handle another order or end the program.

Modifications: You might want fewer columns. If you do, just leave out one or more. You can also add more columns, but be careful! If you use large numbers (over 99,999.99), don't use more than four columns if you want to assure proper space separation. For smaller numbers, you can go to six columns—perhaps even eight—but you'll have to work at formatting the columns with blank spaces, PRINT@ or TAB. The advantage of only four columns is the simplicity of zone printing.

What if you want to make changes after the totals? Add the following lines:

```
305 PRINT:INPUT"CHANGES?(YES=1,NO=Ø)";D
306 IF D=1 GOTO 115
```

This will allow you to add or subtract and get new totals, tax, and handling. Some "old" lines won't erase, but that's only an aesthetic consideration; the program will still give the correct figures.

Variables:

A$—ready to continue
P—total, column 1
B—total, column 2
C—total, column 3
R—total, column 4
X—print position locator
E—column 1 entry
F—column 2 entry
G—column 3 entry
H—column 4 entry
T—tax
S—shipping/handling
M—grand total
A—repeat prgram control

PROGRAM 11
SIMPLIFIED BOOKKEEPING
(4K, Level I or II)

You'll learn: How to stop initial overscrolling in Level I; formatting more than four columns with selected-column entry

Description: This program helps you to do simple income or expense bookkeeping using a 12-column ledger. You can use it for a small business or for personal finance accounting. Column 1 is used for the amount of an income or expense. The other 11 columns are "categories" into which this income or expense falls. This program does not actually make ledger entries, but it allows you to add all columns sequentially and display the totals at any time, including the grand total.

Explanation: The listing for this program is the same for Level I and Level II, expect that no Level I abbreviations are shown. If you enter this program in a Level I machine and then LIST the program, it will fill the screen and stop. To continue the listing, you must press the up-arrow for each new line. However, you may find that some line listings actually take up two lines on the screen, and when they do, the displayed portion of the list scrolls up more than one line. As a result, on initial listing, the first line or two may actually get scrolled up and off the screen before you can read them. Unless the screen prompt and LIST (or L.) are at the top of the screen, you may not be able to see the first line or two of your program. The remedy is to insert REM statements as line numbers *before* the regular program. You'll need from two to four of these, depending on the lengths of the lines that list at the bottom of the screen. In this program, lines 96–99 serve this purpose.

This precaution is not required for Level II listings, which scroll up automatically but can be stopped by pressing the SHIFT and @ keys at the same time.

Lines 1Ø0 and 1Ø1 identify the program author. Lines 1Ø5–129 clear the screen and print the program title and purpose. Notice that the totals are limited to 9999.99. This restriction is due to

LEVEL I/II LISTING

```
96 REM * THESE REM STATEMENTS KEEP THE LIST FROM
97 REM SCROLLING IN LEVEL I LISTING *
98 REM * TRS-80 LEVEL I OR LEVEL II *
99  REM
100  REM * COPYRIGHT FRED BLECHMAN 1978 *
101  REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
105  CLS:PRINT
120  PRINT"              SIMPLIFIED BOOKKEEPING":PRINT:PRINT
121  PRINT"      THE PURPOSE OF THIS PROGRAM IS TO ALLOW YOU TO"
122  PRINT"ADD UP TO TWELVE COLUMNS OF FIGURES AT ONCE, SUCH AS IN"
123  PRINT"A SIMPLE LEDGER BOOK.  FOR EXAMPLE, USE COLUMN 1 AS THE"
124  PRINT"AMOUNT OF AN EXPENSE, AND THE OTHER ELEVEN COLUMNS FOR"
125  PRINT"EXPENSE CATEGORIES.  COLUMN 2 MIGHT BE MERCHANDISE, COLUMN"
126  PRINT"3 MIGHT BE UTILITIES, AND SO ON.  THE AMOUNTS ENTERED ALL"
127  PRINT"FALL IN COLUMN 1 AND ARE TOTALLED WHEN 0 ENTERED."
128  PRINT:PRINT"TOTALS ARE LIMITED TO 9999.99!":PRINT:PRINT
129  INPUT"WHEN READY TO PROCEED, HIT 'ENTER' ";A$
130  CLS:PRINT:PRINT
135  PRINT"      ENTER THE AMOUNT AND COLUMN # FOR EACH EXPENSE":PRINT
140  PRINT"            ENTER 0 FOR COLUMN TOTALS
141  REM * INITIALIZE VALUES AT ZERO *
145  A=0:B=0:C=0:D=0:E=0:F=0:G=0:H=0:I=0:J=0
146  K=0:L=0:S=0:X=0:Y=0:Z=0
150  PRINT:PRINT
160  INPUT"AMOUNT";A
170  IF A=0 THEN 340
180  S=S+A
190  INPUT"                          COLUMN";X
191  IF(X<2)+(X>12)PRINT"COLUMN ENTRY ERROR!! ONLY 2-12 VALID":GOTO190
192  REM * SELECT COLUMN AND ADD TO PREVIOUS TOTAL *
195  ON X-1 GOTO 200,210,220,230,240,250,260,270,280,290,300
200  B=B+A:GOTO310
210  C=C+A:GOTO310
220  D=D+A:GOTO310
230  E=E+A:GOTO310
240  F=F+A:GOTO310
250  G=G+A:GOTO310
260  H=H+A:GOTO310
270  I=I+A:GOTO310
280  J=J+A:GOTO310
290  K=K+A:GOTO310
300  L=L+A
310  GOTO 160
320  REM * FORMAT HEADINGS AND PRINT COLUMN TOTALS *
340  PRINT"COLUMN #";TAB(10),"1";TAB(19);"2";TAB(28);"3";
350  PRINTTAB(37);"4";TAB(46);"5";TAB(55);"6"
360  PRINT"TOTALS=";TAB(9);S;TAB(18);B;TAB(27);C;
370  PRINTTAB(36);D;TAB(45);E;TAB(54);F
375  PRINT:PRINT
380  PRINT"COLUMN #";TAB(10);"7";TAB(19);"8";TAB(28);"9";
390  PRINTTAB(37);"10";TAB(46);"11";TAB(55);"12"
400  PRINT"TOTALS=";TAB(9);G;TAB(18);H;TAB(27);I;
410  PRINTTAB(36);J;TAB(45);K;TAB(54);L:PRINT
411  REM * DRAW LINES TO SET ASIDE GRAND TOTAL *
415  V=30:FORW=20 TO 34:SET(W,V):NEXTW
416  FORV=24 TO 30:SET(W,V):NEXTV
420  INPUT"DO YOU WANT TO ADD MORE ENTRIES? YES=1,NO=0";Y
430  IF Y=1 GOTO 150
450  PRINT
460  INPUT" WOULD YOU LIKE TO START OVER? YES=1,NO=0";Z
470  IF Z=1 GOTO 129
475  CLS:PRINT:PRINT:PRINT:PRINT
480  PRINT"       END OF PROGRAM - SEE YOU AGAIN SOMETIME..."
490  PRINT:PRINT:PRINT:PRINT:PRINT
500  END
```

PROGRAM 11 RUN

```
            SIMPLIFIED BOOKKEEPING


     THE PURPOSE OF THIS PROGRAM IS TO ALLOW YOU TO
ADD UP TO TWELVE COLUMNS OF FIGURES AT ONCE, SUCH AS IN
A SIMPLE LEDGER BOOK. FOR EXAMPLE, USE COLUMN 1 AS THE
AMOUNT OF AN EXPENSE, AND THE OTHER ELEVEN COLUMNS FOR
EXPENSE CATEGORIES. COLUMN 2 MIGHT BE MERCHANDISE, COLUMN
3 MIGHT BE UTILITIES, AND SO ON. THE AMOUNTS ENTERED ALL
FALL IN COLUMN 1 AND ARE TOTALLED WHEN 0 ENTERED.

TOTALS ARE LIMITED TO 9999.99!


WHEN READY TO PROCEED, HIT 'ENTER' ?


     ENTER THE AMOUNT AND COLUMN # FOR EACH EXPENSE

          ENTER 0 FOR COLUMN TOTALS


AMOUNT? 123.45
                        COLUMN? 1
COLUMN ENTRY ERROR!! ONLY 2-12 VALID
                        COLUMN? 3
AMOUNT? 12.45
                        COLUMN? 7
AMOUNT? 314.56
                        COLUMN? 5
AMOUNT? 0

COLUMN #  1        2        3        4        5        6
TOTALS=   450.46   0        123.45   0        314.56   0


COLUMN #  7        8        9        10       11       12
TOTALS=   12.45    0        0        0        0        0

DO YOU WANT TO ADD MORE ENTRIES? YES=1,NO=0? 0

 WOULD YOU LIKE TO START OVER? YES=1,NO=0? 0



       END OF PROGRAM - SEE YOU AGAIN SOMETIME....
```

formatting, as you'll see later, not the limits of the TRS-80 calculating ability.

The A$ at the end of the INPUT statement in line 129 gives you time to press ENTER, although any combination of as many as 15 letters or numbers can be typed before ENTER. Line 130 clears the screen, and the two PRINT statements move the first line of text down two lines, for a pleasing display.

Lines 135 and 140 print the instructions, with a reminder to enter 0 for column totals. Notice how spaces are used after the first quotation mark in these two lines to approximately center the

printing from left to right. Although these spaces use memory, they are an easy way to format the screen display, and they also work the same way with a printer, whereas TAB or PRINTAT statements do not.

Lines 141–146 set various variables to zero. This operation is not necessary in Level II for the *initial* RUN, but it *is* necessary for reruns. Line 15Ø skips two lines, and line 16Ø asks for an amount to be entered—either a positive or negative number. The computer will hold this number in memory as variable A. Line 17Ø looks at this value of A to see if it's zero; if it is, that means you want totals, and the program branches to line 34Ø. But let's say, instead, that you type and ENTER 123.45. Line 17Ø is therefore ignored, and Line 18Ø adds the value of A, 123.45, to the value in memory for S (set to zero in line 149) and comes up with a "new" value of S equal to 123.45. Not very exciting so far, but just wait a bit.

Line 19Ø now asks you into what column this amount (A in line 16Ø) should go and calls this column number variable X. The blank spaces in this line move the word "COLUMN" to the right to make it very noticeable. Line 191 rejects any entry here except the numbers through 2 through 12. Since the plus sign means OR, the line reads, in plain English, "IF the value of X is less than 2 or greater than 12, print 'COLUMN ENTRY ERROR!! ONLY 2–12 VALID' and then go to line 19Ø." Thus, if you enter any number other than 2 through 12, you'll be asked for the column number again. Why isn't the number 1 a valid entry? Because *all* numbers entered as "AMOUNT?" fall in column 1 automatically. The object in this ledger system is to break down all column 1 entries into categories, called columns 2–12. If you don't understand this scheme, go back and read lines 12Ø–129.

Now we come to another ON X GOTO statement, first encountered in Program 3, but here there's a slight twist. Notice that line 195 starts with ON X-1 GOTO. What's that X-1 doing in there? Well, remember that since we've eliminated 1 as a valid column entry, the first valid column number, which the program calls variable X, is 2. Therefore, by subtracting 1 from X, we've modified the ON X GOTO statement so that an X value of 2 sends the program to line 2ØØ; a value of 3 to line 21Ø; a value of 4 to line 22Ø; and so on. This method of offsetting X is also used in multiple-line ON X GOTO statements.

Lines 2ØØ–3ØØ keep track of the totals of each column with variables B through L—B for column 2, C for column 3, and so on. Since variables B through L were set to zero in lines 145 and 146,

each column total starts out at zero. Each new entry, A, is directed to a particular column by line 190, and line 195 adds this value to the variable for that column. In every case, line 310 is the next instruction, which in turn sends the program back to line 160 for another "AMOUNT?" entry. Actually, line 160 could be specified directly at the end of lines 200–300 instead of at line 310, and line 310 could be deleted. Unnecessary residual jumping is common in programs as they develop from a simple framework.

Remember that at line 170 the computer looked at the amount entered to see if it was zero. Assume that you enter a zero now (after various amount and column entries, such as those shown in the sample RUN). The program then leaps ahead to lines 340 and 350, which print column numbers formatted by TAB statements. Column numbers 1–6 are printed nine spaces apart, and then line 360 prints the column totals (S, B, C, D, E, and F) under these headings. Line 375 skips two lines, and lines 380–410 print column numbers and totals for columns 7–12. Notice how the TAB statements line up everything neatly.

Lines 415 and 416 use SET statements in two simple FOR-NEXT loops to place a horizontal line under, and a vertical line to the right of, the "grand total," column 1. Program 1 explained how SET statements are used.

Lines 420 and 430 allow you to add additional entries. If you choose to, notice that the program goes back to line 150 and *does not* reset the variables to zero. If you would like to start over, however, lines 460 and 470 send the program back to line 129 and all variables affecting column totals are reset to zero as the program moves on to lines 145 and 146.

Lines 475–500 end the program gracefully with an invitation to return. All the print statements do is to put the message down from the top of the screen and to move the READY prompt five lines below the message.

Modifications: You can hold the final column totals in memory by assigning them different variable names and thus have "page" totals, with a "month" total where requested. You can also keep "month" totals in memory for "year" totals. This practice, of course, complicates programming but can be accomplished without too much difficulty by using arrays.

Also, if you use fewer columns per line, you can handle larger numbers. With six columns per screen line, 9999.99 is the practical limit, unless you let numbers from one column extend into

the next column or even into the next line. Arrange your format for, say, four columns across, with three sets of column headings—a total of 12 columns.

Variables:

A$—ready to continue
A—amount, column 1
B—total, column 2
C—total, column 3
D—total, column 4
E—total, column 5
F—total, column 6
G—total, column 7
H—total, column 8
I—total, column 9
J—total, column 10
K—total, column 11
L—total, column 12
S—grand total, column 1
X—column entry number
Y—additional entry control
Z—rerun control
V—vertical screen position
W—horizontal screen position

PROGRAM 12
INTEREST CALCULATION AND TABULATION
(4K, Level I or II)

You'll learn: How to make the TRS-80 calculate from a formula and tabulate the results; handling integer exponents in Level I the easy way; basic interest formula

Description: This program calculates and tabulates the amount of interest for a given principal, interest rate, and period—daily, monthly, quarterly, or annually. It displays the total value, interest total, and, if requested, the interest and totals for each period.

Explanation: Here we go into the world of high finance. This will involve a little mathematics if you really want to follow program operation, and it will also give you a healthy respect for the way in which your TRS-80 can tirelessly grind through formulae.

Probably the most often requested information among savers is "How much will I make if I invest my money in a savings account?" For example, suppose that you deposit $10,000 in a savings account that pays 6 percent annual interest, compounded daily, for 30 days. The formula, with the variables used in this program, is as follows:

$$G = A\,[B/(N+1)]^C$$

where:

G = total principal plus interest at end of C periods
A = initial deposit ($10,000)
B = annual interest rate, in percent (6%)
N = number of compounding periods per year (365)
C = number of periods in this calculation (30)

Notice that since we've established values for A, B, N, and C, the formula can be solved for G. You could use a log slide rule (remember them?) or a scientific calculator to solve this equation, since an exponent (C) is involved. Or, of course, you could do it longhand, but your TRS-80 will do it very quickly once it has been programmed. Let's follow the approach of this program.

```
2  REM * COPYRIGHT FRED BLECHMAN 1978 *
3 REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
4 REM * TRS-80 LEVEL I OR LEVEL II 4K *
5  CLS:PRINT:PRINT"           INTEREST CALCULATION & TABULATION"
10  PRINT:PRINT:PRINT"      THIS PROGRAM WILL ALLOW YOU TO DETERMINE"
11  PRINT"AND TABULATE THE AMOUNT OF INTEREST ON ANY PRINCIPAL"
12  PRINT"FOR ANY PERIOD AND AT ANY INTEREST RATE COMPOUNDED"
13  PRINT"ANNUALLY, QUARTERLY, MONTHLY OR DAILY.  (SOME "
14  PRINT"NUMBERS MAY EXCEED THE LIMITS OF THE TRS-80, SO"
15  PRINT"BE REASONABLE.....)":PRINT
20  INPUT"WHAT IS THE PRINCIPAL ($) ";A
30  INPUT"WHAT IS THE ANNUAL INTEREST RATE (%)";B
35  INPUT"1, 4, 12 OR 365 COMPOUNDING PERIODS PER YEAR ";N
36  IF(N<>1)*(N<>4)*(N<>12)*(N<>365)PRINT"ERROR! TRY AGAIN":GOTO35
40  INPUT"TOTAL NUMBER OF PERIODS";C
45  REM * CONVERT ANNUAL INTEREST RATE TO PERIOD RATE *
46  IF N=1 THEN D=B/100
47  IF N=4 THEN D=B/400
48  IF N=12 THEN D=B/1200
49  IF N=365 THEN D=B/36500
50  REM * INTEREST CALCULATION *
51  CLS
52  IF N=1 PRINT"INTEREST IS BEING COMPOUNDED ANNUALLY"
53  IF N=4 PRINT"INTEREST IS BEING COMPOUNDED QUARTERLY"
54  IF N=12 PRINT"INTEREST IS BEING COMPOUNDED MONTHLY"
55  IF N=365 PRINT"INTEREST IS BEING COMPOUNDED DAILY"
56  PRINT:PRINT"        HOLD EVERYTHING.... I'M CALCULATING....."
57  PRINT:PRINT
58  REM * INTEREST CALCULATION *
60  E=D+1
70  IF C=1 GOTO 110
80  FOR X=1 TO (C-1)
90  E=E*(D+1)
100  NEXT X
110  G=E*A
120  PRINT"THE VALUE OF $";A;"AT";B;"% FOR";C;"PERIODS IS $";G
125  PRINT
126  PRINT"            THE INTEREST ALONE IS $";G-A
127  PRINT
128  INPUT"WANT A TABULAR PRINTOUT? YES=1,NO=0";J
129  IF J=1 GOTO 165
130  INPUT"AGAIN? YES=1,NO=0";H
140  IF H=1 GOTO 5
150  CLS:PRINT:PRINT:PRINT:PRINT
155  PRINT"              SEE YOU AGAIN SOMETIME.   BYE....."
156  PRINT:PRINT:PRINT:PRINT
160  END
165  I=A*D
166  M=0:M=M+I
167  Z=0
168  K=A+I
175  CLS
176  PRINT"PERIOD","INT/PERIOD","TOTAL INT.","TOTAL VALUE"
180  FOR X=1 TO 12
181  Z=Z+1
182  IF Z=C+1 PRINT:PRINT"      END OF TABULATION":PRINT:GOTO 130
183  PRINTZ,I,M,K
185  I=K*D
186  M=M+I
187  K=A+M
190  NEXT X
191  PRINT
195  INPUT"TO CONTINUE THE TABULATION, HIT 'ENTER' ";A$
196  GOTO 175
```

PROGRAM 12 RUN

```
          INTEREST CALCULATION & TABULATION


     THIS PROGRAM WILL ALLOW YOU TO DETERMINE
AND TABULATE THE AMOUNT OF INTEREST ON ANY PRINCIPAL
FOR ANY PERIOD AND AT ANY INTEREST RATE COMPOUNDED
ANNUALLY, QUARTERLY, MONTHLY OR DAILY. (SOME
NUMBERS MAY EXCEED THE LIMITS OF THE TRS-80, SO
BE REASONABLE.....)

WHAT IS THE PRINCIPAL ($) ? 10000
WHAT IS THE ANNUAL INTEREST RATE (%)? 6
1, 4, 12 OR 365 COMPOUNDING PERIODS PER YEAR ? 365
TOTAL NUMBER OF PERIODS? 30

INTEREST IS BEING COMPOUNDED DAILY

        HOLD EVERYTHING....I'M CALCULATING.....


THE VALUE OF $ 10000 AT 6 % FOR 30 PERIODS IS $ 10049.4

           THE INTEREST ALONE IS $ 49.4336

WANT A TABULAR PRINTOUT? YES=1,NO=0? 1
PERIOD          INT/PERIOD      TOTAL INT.      TOTAL VALUE
 1               1.64384         1.64384         10001.6
 2               1.64411         3.28794         10003.3
 3               1.64438         4.93232         10004.9
 4               1.64465         6.57697         10006.6
 5               1.64492         8.22188         10008.2
 6               1.64519         9.86707         10009.9
 7               1.64546         11.5125         10011.5
 8               1.64573         13.1583         10013.2
 9               1.646           14.8043         10014.8
 10              1.64627         16.4505         10016.5
 11              1.64654         18.0971         10018.1
 12              1.64681         19.7439         10019.7

TO CONTINUE THE TABULATION, HIT 'ENTER' ?
 25              1.65033         41.1771         10041.2
 26              1.6506          42.8277         10042.8
 27              1.65088         44.4785         10044.5
 28              1.65115         46.1297         10046.1
 29              1.65142         47.7811         10047.8
 30              1.65169         49.4328         10049.4

      END OF TABULATION

AGAIN? YES=1,NO=0? 0

             SEE YOU AGAIN SOMETIME.  BYE.....
```

As usual, the initial REM lines identify the author and machine requirements. The listing is designed to run in *either* Level I

or Level II with no changes (although you can use abbreviations for many statements in Level I).

Lines 5–15 clear the screen, introduce the program, and point out that the TRS-80 does have some number limits; for numbers over a million, the output will be in exponential notation (such as 1.025E+06). Be reasonable, therefore, with your inputs.

Line 2Ø asks you to enter the principal and stores this as variable A. Line 3Ø asks for the annual interest rate, in percent, and stores this as variable B. Line 35 gives you a choice of four typical yearly compounding periods. The 1 represents annual; 4, quarterly; 12, monthly; and 365, daily periods. This data is stored in the computer memory as variable N. Line 36 makes sure that you don't enter any number other than 1, 4, 12, or 365. The asterisks (*) in this case mean "AND"; some Level II machines may require you to type in AND instead of the asterisk.

Line 4Ø asks for the total number of periods for this calculation and stores this input as variable C.

Now we get into a simple conversion—dividing the annual interest rate (B) by 100 to get a percentage and then dividing by the number of compounding periods (N) to get a "period rate," D. These operations are accomplished in lines 46–49; of course, only *one* of these lines is used, depending on the value of N chosen in response to line 35.

Now line 51 clears the screen, and the TRS-80 prints a quick one-liner (line 52, 53, 54, or 55), skips a line, and prints the text in line 56 so that you don't think the program has "bombed" while the interest is being calculated. If you ask for a year of daily compounding (365 periods), the machine has to loop through lines 8Ø–1ØØ 364 times, which takes about 6 seconds. Obviously, five years of daily compounding (365 × 5 = 1825 periods) would take about 30 seconds.

To explain how the formula is handled, let's use the numbers from our example, as shown in the sample RUN. A is 10000, B is 6 percent, N is 365, and C is 30. At this point, then, D, from line 49, is equal to B/36500, or .000164384 (which is the same as 1.64384E-04 in TRS-80 notation). Line 6Ø simply adds 1 to this and calls the result (1.000164384) the variable E at this point. You should recognize that this is the value inside the parentheses of our formula.

Now, somehow we have to get the computer to multiply this value of (D+1) by itself for (C−1) times—in other words, raise (D+1) to the C power. Doing so is easy in Level II. You just use the up-arrow, and the program reads E=(D+1)↑C. However, since Level I has no exponent function, an involved series of subroutines

would normally be necessary, as shown in the back of the Level I Users' Manual. But these are *not necessary* here since the exponent is a full number—an integer. All you need do is multiply E by (D + 1) for (C – 1) times, "updating" E to its latest value each time.

Line 7Ø looks at C. If C is 1, the program jumps to line 11Ø. In our example, since C is 30, line 7Ø is ignored. Line 8Ø starts a FOR-NEXT loop, using X as the variable, starting at 1, and counting to 1 *less* than C. (This procedure may seem strange, but since (D + 1) is raised to the *second* power the *first* time through this loop, and so on, we must stop at C – 1). Line 9Ø establishes a new value of E equal to the old value of E (from line 6Ø) times the value of (D + 1). Line 1ØØ loops back and does this calculation again, except that now the second E in line 9Ø is the last value calculated for E. Notice that (D + 1) remains unchanged. Finally, after 29 loops (since C – 1 = 30 – 1 = 29), the program falls through to line 11Ø. Here, the last value of E [which is (D + 1) raised to the C power] is multiplied by the original principal, A, to yield the principal plus interest, G.

Line 12Ø prints the results, line 125 skips a line, and line 126 prints the interest alone (the total minus the original principal). Line 127 skips a line, and line 128 asks if you want a tabulation. If you answer Ø (for NO), you are asked by line 13Ø if you want to use the program again. A 1 (for YES) triggers line 14Ø to send the program back to line 5 to start over. A Ø (for NO) ends the program with lines 15Ø–16Ø.

Suppose, however, that you want a tabulation and you respond to the request in line 128 with a 1. Line 129 then sends the program to line 165. Here a new variable, I, is introduced and made equal to the principal, A, times the period rate, D. Using our numbers, I is equal to 10,000 times .000164384, or 16.4384. This is the interest for one period (day) for $10,000 at 6 percent, compounded daily.

However, compounding means that for each new period the interest is calculated on a *new* principal, consisting of the last principal plus the latest period's interest. We're therefore going to need some more variables to keep track of things. Let's use M as the total interest, Z as the period number, and K as the total of principal and interest. Line 166 sets the total interest at zero and then immediately adds the interest, I. (This line could have been written simply as M = I). Line 167 sets Z to zero. Line 168 establishes K as equal to the principal plus the first period interest. Line 175 clears the screen.

Line 176 prints tabulation headings on the screen's top lines, using zone printing (note the commas). Line 18Ø starts a FOR-NEXT

loop to count to 12. Line 181 advances the value of Z by 1, making it now equal to 1, since it was set to zero in line 167. Line 182 checks to see if the required number of periods, C, has been exceeded. If not, line 183 prints the values of Z, I, M, and K under the four headings—zone printing again. Line 185 calculates the *new* interest, which is the latest total, K, times the period rate, D. Line 186 calculates the new total interest, line 187 calculates the new total of principal and interest, and line 190 sends the program back to line 180 for another loop, until the X in line 190 has advanced to 13. At this point, 12 lines of tabulation are on the screen, and the program falls through line 190 to lines 191 and 195, where everything stops until you hit ENTER. This is a method of "paging" your tabulation so that it doesn't just keep running and scroll up and off the screen as new lines appear at the bottom. Line 196 sends the program to line 175 to clear the screen, reprint the column headings, and display up to 12 more rows of tabulation.

However, as soon as line 182 finds that you've tabulated the required number of periods, C, the "END OF TABULATION" statement is printed, and the program is routed to line 130 to see if you want another calculation. Your TRS-80 awaits your pleasure.

Modifications: It's easy to eliminate the tabulation "paging" if you'd like to do so. Just change line 180 to FOR X = 1 TO C + 1 and delete lines 195 and 196. Remember that you can stop the tabulation at any time in Level I by holding down the up-arrow and in Level II by holding down the SHIFT key and pressing the @ key.

Variables:

A—principal
B—annual interest rate, in percent
N—number of compounding periods per year
C—number of periods for this calculation
D—period rate $\left(\frac{B}{100 \times N}\right)$
E—$(D+1)^C$
X—loop counter
G—final principal plus interest
J—tabulation control
H—rerun control
I—interest for each period
M—total interest for each period
Z—period number
K—total principal plus interest for each period
A$—pause control

PROGRAM 13
INVEST—OR SAVE?
(4K, Level I or II)

You'll learn: Subroutine for counting days between two dates; simple daily interest calculation; comparison of values.

Description: Before making an investment, you may want to compare its present value with that of money left in an interest-compounded-daily savings account. This program allows you to do this and also to compare the future value of savings to those of a proposed investment. The program also has a subroutine that counts the days between two dates.

Explanation: Line 1ØØ clears the screen and the top line. Lines 1Ø5–1Ø7 specify the author and the equipment required. As listed, this program will run in both Levels I and II, except for the PRINT@ in line 36Ø, which must be changed to PRINTAT in Level I. Of course, if you're using Level I, you may use the permitted abbreviations.

Lines 11Ø–12Ø print the introductory text, and line 125 waits for you to type and ENTER your first name. Line 126 asks for your social security number. Why? Computers *always* want your social security number, especially when finances are involved. Actually, you could eliminate these two lines, but they personalize later results.

Lines 13Ø–132 set variables A through Z to zero, even though some are not used in the program. In Level I, they comprise all the numerical variables you have, except for the A-array. Initialized at zero on the first run, they are reset to zero by the same lines on subsequent evaluations.

Lines 14Ø–15Ø allow you to input the amount to be invested (or being considered for that purpose) and the annual interest rate of your regular savings. The program assumes daily compounding, which is virtually standard in savings institutions. If you have other compounding (monthly, quarterly, or annual), this program will be slightly in error.

```
100  CLS:PRINT
105  REM *COPYRIGHT 1978 FRED BLECHMAN *
106 REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
107 REM * TRS-80 LEVEL I OR LEVEL II *
110 PRINT"              INVEST - OR SAVE????"
111  PRINT
112  PRINT"     THIS PROGRAM COMPARES THE INTEREST IN A SAVINGS"
113  PRINT"ACCOUNT, COMPOUNDED DAILY, WITH THE VALUE OF AN"
114  PRINT"INVESTMENT.  IN OTHER WORDS, IT ANSWERS THE QUESTION,"
115  PRINT"'HAVE I MADE MORE IN MY INVESTMENT, OR WOULD I HAVE"
116  PRINT"BEEN BETTER OFF LEAVING THE MONEY IN SAVINGS?' YOU"
117  PRINT"CAN ALSO COMPARE THE FUTURE VALUE OF SAVINGS COMPARED"
118  PRINT"TO A PROPOSED INVESTMENT....."
120  PRINT
125  INPUT"WHAT IS YOUR FIRST NAME";A$
126  INPUT"WHAT IS YOUR SOCIAL SECURITY NUMBER";B$
130  A=0:B=0:C=0:D=0:E=0:F=0:G=0:H=0:I=0
131  J=0:K=0:L=0:M=0:N=0:O=0:P=0:Q=0:R=0
132  S=0:T=0:U=0:V=0:W=0:X=0:Y=0:Z=0
140  INPUT"WHAT IS THE DOLLAR AMOUNT INVESTED";P
150  INPUT"WHAT IS YOUR REGULAR SAVINGS INTEREST RATE(%)";R
155  CLS:PRINT
160  PRINT"              HOW MANY INVESTMENT DAYS?"
165  PRINT"(IF YOU WANT THIS NUMBER OF DAYS CALCULATED - WITHIN THE"
170  PRINT"20TH CENTURY ONLY - THEN ENTER 0)":PRINT:INPUTD
175  IF D>0 GOTO190
180  IF D=0 GOSUB 500
190  PRINT".....PATIENCE!...I'M CALCULATING THE ANSWER..."
195 PRINT"(LEVEL I: TAKES ABOUT 10 SECONDS FOR 365 DAYS)"
196 PRINT"(LEVEL II: TAKES ABOUT 15 SECONDS FOR 365 DAYS)"
200  S=R/36500:V=P
205  REM * CALCULATE INTEREST AND ADD TO PRINCIPAL *
210  FOR X=1 TO D
220  I=V*S:V=V+I:T=T+I
230  NEXT X
235  CLS:PRINT
240  PRINT"          THE TOTAL INTEREST IS $";T
250  PRINT"THE VALUE OF $";P;"AFTER";D;"DAYS AT";R;"% IS $";V
254  PRINT
255  PRINT"OKAY, NOW LET'S COMPARE THIS WITH YOUR INVESTMENT..."
256  REM * COMPARE PRESENT VALUE OF INVESTMENT TO SAVINGS *
257  PRINT
260  PRINT"     HOW MANY OUNCES, BARS, SHARES OR WHATEVER ARE"
261  INPUT"THERE IN THE INVESTMENT YOU WANT EVALUATED ";H
275  PRINT
280  INPUT"WHAT IS THE PRESENT VALUE OF EACH SHARE,BAR,ETC. ";M
290  Q=H*M
295  CLS:PRINT:PRINT:PRINT
300  PRINT"     THE VALUE OF YOUR INVESTMENT IS  $";Q
310  Z=V-Q
315  PRINT
320  IF Z>0 PRINTA$;"-";B$;"-YOU HAVE LOST $";Z;"COMPARED TO SAVING!"
330  IF Z<0 PRINTA$;"-";B$;"-YOU HAVE EARNED $";-Z;"MORE THAN SAVING!
335  PRINT:PRINT"THE 'BREAK-EVEN' POINT IS' $";V/H;"PER SHARE,BAR,ETC.
340  PRINT:PRINT:INPUT"ANOTHER EVALUATION? YES=1 NO=0 ";X
350  IF X=1 CLS:PRINT:PRINT:PRINT:GOTO130
355  CLS
360  PRINT@ 390,"WELL, IT WAS NICE TO HELP YOU.  SEE YOU SOMETIME..."
370  PRINT:PRINT:PRINT
380  END
500  REM * SUBROUTINE FOR CALCULATING NUMBER OF DAYS *
501  REM * BASED ON JULIAN CALENDAR *
502  REM * RANGE: MARCH 1, 1900 TO FEBRUARY 28, 2100 *
510  CLS: E=1
```

```
520  INPUT "WHAT IS THE INVESTMENT START DATE(MM,DD,YYYY)";A,B,C
530  IFA<=2 THEN C=C-1: A=A+13: GOTO550
540  A=A+1
550  F=365*C+INT(.25*C)+INT(30.6001*A)+B+1720982
560  IFE=1 THENG=F:E=2:PRINT:PRINT:INPUT"END DATE(MM,DD,YYYY)";A,B,C:GOTO530
570  D=F-G
580  PRINT"THE NUMBER OF INVESTMENT DAYS IS";D
590  RETURN
```

PROGRAM 13 RUN

```
              INVEST - OR SAVE????

     THIS PROGRAM COMPARES THE INTEREST IN A SAVINGS
ACCOUNT, COMPOUNDED DAILY, WITH THE VALUE OF AN
INVESTMENT. IN OTHER WORDS, IT ANSWERS THE QUESTION,
'HAVE I MADE MORE IN MY INVESTMENT, OR WOULD I HAVE
BEEN BETTER OFF LEAVING THE MONEY IN SAVINGS?' YOU
CAN ALSO COMPARE THE FUTURE VALUE OF SAVINGS COMPARED
TO A PROPOSED INVESTMENT.....

WHAT IS YOUR FIRST NAME? FRED
WHAT IS YOUR SOCIAL SECURITY NUMBER? 123-45-6789
WHAT IS THE DOLLAR AMOUNT INVESTED? 1000
WHAT IS YOUR REGULAR SAVINGS INTEREST RATE(%)? 5.25

            HOW MANY INVESTMENT DAYS?
(IF YOU WANT THIS NUMBER OF DAYS CALCULATED - WITHIN THE
20TH CENTURY ONLY - THEN ENTER 0)

?0
WHAT IS THE INVESTMENT START DATE(MM,DD,YYYY)? 1,1,77


END DATE(MM,DD,YYYY)? 1,1,79
THE NUMBER OF INVESTMENT DAYS IS 730
.......PATIENCE!...I'M CALCULATING THE ANSWER....
(LEVEL I: TAKES ABOUT 10 SECONDS FOR 365 DAYS)
(LEVEL II: TAKES ABOUT 15 SECONDS FOR 365 DAYS)


        THE TOTAL INTEREST IS $ 110.702
THE VALUE OF $ 1000 AFTER 730 DAYS AT 5.25 % IS $ 1110.7

OKAY, NOW LET'S COMPARE THIS WITH YOUR INVESTMENT...

     HOW MANY OUNCES, BARS, SHARES OR WHATEVER ARE
THERE IN THE INVESTMENT YOU WANT EVALUATED ? 202

WHAT IS THE PRESENT VALUE OF EACH SHARE,BAR,ETC.? 7.65


     THE VALUE OF YOUR INVESTMENT IS  $ 1545.3

FRED-123-45-6789-YOU HAVE EARNED $ 434.598 MORE THAN SAVING!

THE 'BREAK-EVEN' POINT IS $ 5.49852 PER SHARE,BAR,ETC.


ANOTHER EVALUATION? YES=1 NO=0 ? 0
WELL,IT WAS NICE TO HELP YOU. SEE YOU SOMETIME...
```

Line 155 clears the screen and the first line. Lines 160–170 ask you how many investment days you want considered, with the option of having the number of days calculated from the starting and ending dates. If you enter any positive number, line 175 sends the program to line 190, and the calculations begin. Let's assume, however, that you want the number of days calculated. Just enter zero in response to the line 170 prompt, and line 180 jumps the program to a subroutine starting at line 500.

Lines 500 to 502 point out that since this subroutine for calculating the number of days is based on the Julian Calendar, it is correct only from March 1, 1900 to February 28, 2100. The Julian Calendar was prescribed by Julius Caesar, which, although it uses a form of leap year, contains an error of one day in 128 years and is now 13 days behind the Gregorian Calendar, which is presently in use in most parts of the world. In any case, lines 510 through 570 ask you for the starting and ending dates and go through the necessary calculations; line 580 prints the final results and returns the program to line 190.*

We get to line 190 with a value for D (days) either input in line 170 or calculated by the subroutine in lines 500–590. Lines 190–196 let you know that you might have to wait awhile, especially if you're dealing with years rather than weeks. Oddly, Level I runs through the calculations faster than Level II.

Line 200 divides the interest rate by 36500 to come up with a daily period rate and assigns this value to variable S. Also, V, the running total of principal and interest, is set equal to P, the principal (amount invested).

Lines 210–230 set up a FOR-NEXT loop to calculate the interest for each day (previous total times daily period rate), adds the new interest to the last running total to get a new total, and adds the new interest to the previous total interest to get a new total interest, T. Line 230 repeats the calculations for D times and then allows the program to fall through to lines 235–280, which display the results and ask the amount and value of your investment.

Line 290 simply multiplies the quantity of shares, ounces, or whatever by the value per share or ounce to come up with a total value, Q. Lines 295 and 300 display this total. Line 310 subtracts this total from V, the value of savings after this period. If the result is negative (less than 0), then line 320 prints its message. If the result is positive, line 330 is printed. Line 335 calculates and prints the "break-even" point—the value per share or ounce that would be

*This subroutine was suggested by a correspondent, G. A. Barton of Teheran, Iran, and it works beautifully, although the author must admit he doesn't know exactly how!

equal to the savings total. Lines 340–380 allow you to perform another evaluation or exit the program. Notice that if you choose another evaluation, in line 340, line 350 sends the program to line 130 so that all variables will be reset to zero.

Modifications: You could use the approach of Program 12 to allow monthly, quarterly, or annual compounding.

Variables:

A$—first name
B$—social security number
A—subroutine (month)
B—subroutine (day)
C—subroutine (year)
D—investment days
E—used within subroutine
F—used within subroutine
G—used within subroutine
H—amount of shares, ounces, etc., to be evaluated
I—daily interest
M—present value of each share, ounce, etc.
P—dollar amount invested
Q—investment value
R—saving interest rate, percent
S—daily period rate
T—running total of interest
V—running total of principal and interest
X—loop counter; evaluation repeat control
Z—value of earnings or loss compared to savings

PROGRAM 14
MORTGAGE LOAN AMORTIZATION PROGRAM
(4K, Level I or II)

You'll learn: Loan calculations; putting a printer option in your program; rounding off numbers to two decimal places; converting programs from Level I to Level II

Description: This program calculates the monthly payment on a typical loan and tabulates the principal and interest for each payment, as well as the running totals. If you prefer, you can select a monthly payment, and the program computes the final "balloon" payment.

Explanation: If you've ever purchased a home or taken out a large loan, the term "amortization schedule" may be familiar to you. It's just a tabulation of each monthly payment, showing how much of the payment is for interest, how much to reduce the principal, and the balance still owed after the payment. This new balance is then used to determine the next amount of interest due, and so on. Each month's interest is therefore lower, since it is based on a declining balance. Since the monthly payment is the same each month, the amount paid toward the principal (the amount originally borrowed) increases each month. What a natural for a computer to churn through!

Ah, but there's a problem! How do you figure out the amount of the monthly payment? You could, of course, set an arbitrary amount, but if it's to small, you will end up with a big "balloon" payment at the end. If you choose a payment too large, you'll pay off the loan sooner than you planned. This program allows you to take this approach, if you desire, but why not let the computer calculate the correct payment for you?

Skip the following comments if you're not into mathematics, since the water gets a bit deep. The traditional formula for calculating the payment on a declining balance (which bankers call a "present value annuity") is

$$\text{Monthly payment} = [\text{Original loan amount}] \left[\frac{R}{1 - (1 + R)^{-Y}}\right]$$

where:

R = monthly interest rate
Y = number of months

Multiply the numerator and denominator of the fraction by $(1 + R)^Y$ and you get the following:

$$\text{Monthly payment} = [\text{Original loan amount}] \left[\frac{(R)(1 + R)^Y}{(1 + R)^Y - 1}\right]$$

Now, we can let $Q = (1 + R)^Y$, and the bracketed part of the equation becomes

$$\frac{RQ}{Q - 1}$$

If we let T equal the monthly payment and F equal the original loan amount, then we have

$$T = F\left(\frac{RQ}{Q - 1}\right)$$

This is the equation we'll use in the program for calculating the monthly payment.

To calculate the monthly interest, we use the following equation:

$$I = L \times R$$

where:

I = monthly interest
L = balance still due
R = monthly interest rate

With the mathematics under control and the proper program to do the calculations, all we have to do is enter the figures into the computer. The Level I listing for doing this provides only a screen display since printer commands are not supported by Level I. We'll explain this listing line by line and then go into the changes for Level II, which allows a printer option. Follow the explanation by referring to the screen printout.

As usual, the initial lines (100–135) introduce the program author, title, and purpose. The INPUT statement (abbreviated IN.) of line 135 holds the text on screen until ENTER is pressed. Line 140 clears the screen, skips a line, and then requests the loan amount, F.

```
100  REM * COPYRIGHT FRED BLECHMAN 1978 *
101  REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
105  CLS:P.AT140;"MORTGAGE LOAN AMORTIZATION PROGRAM"
110  P.:P."      THIS PROGRAM WILL CALCULATE YOUR PAYMENT ON"
115  P."A MORTGAGE LOAN AND TABULATE THE AMOUNT OF PRINCIPAL"
120  P."AND INTEREST FOR EACH PAYMENT, AS WELL AS THE RUNNING"
125  P."TOTALS.  IF YOU PREFER, YOU CAN SELECT A MONTHLY"
130  P."PAYMENT AND THE BALLOON PAYMENT WILL BE COMPUTED."
135  P.:P.:IN."WHEN READY, PRESS 'ENTER' ";A$
140  CLS:P.:P.:IN."WHAT IS THE AMOUNT OF THE LOAN";F
145  P.
150  IN."WHAT IS THE INTEREST RATE (%) ";R
155  P.
160  IN."WHAT IS THE LENGTH OF THE LOAN IN MONTHS ";Y
162  IF Y<>INT(Y) P."ENTRY ERROR! TRY AGAIN":G.160
163  P.
164  IN."DO YOU WANT PAYMENT CALCULATED? YES=1, NO =0";D
169  CLS
170  P.AT136;"THE LOAN IS FOR $";F;"FOR";Y;"MONTHS AT";R;"%"
171  IF D=1 GOTO 175
172  IF D<>0 GOTO 164
173  P.:IN."WHAT MONTHLY PAYMENT WOULD YOU LIKE ";T
175  REM CONVERT INTEREST RATE FROM PCT/YR TO DECIMAL/MONTH
180  R=R/1200
185  IF D<>1 THEN 230
190  REM CALCULATE MONTHLY PAYMENT AND ROUND UP TO CENTS
191  P.:P."              .....PATIENCE! I'M CALCULATING....."
195  X=1+R
196  U=1
197  FOR Z=1 TO Y
198  U=U*X
199  NEXT Z
200  Q=U
210  T=F*((R*Q)/(Q-1))
215  IF T<327.67 THEN T=INT((T+.005)*100)/100
230  P.:P."         THE MONTHLY PAYMENT IS $";T
240  REM * PRINT A MONTHLY TABULATION OR ONLY TOTALS *
245  P.
250  IN."DO YOU WANT A MONTHLY BREAKDOWN? (YES=1, NO=0)";D
260  IF D=0 P.:P."      .....CALCULATING";Y;"MONTHLY PAYMENTS....."
261  IF D=0 P."        (AT THE RATE OF 67 EVERY 10 SECONDS)"
262  IF D=0 GOTO 290
270  IF D<>1 GOTO 250
275  CLS:P.:P."PRESS ↑ TO HOLD TABLE, RELEASE TO CONTINUE..."
276  P.
280  P."MONTH   BALANCE   INTEREST   INT. T. D.   PRINCIPAL   PRIN. T. D.
290  L=F:N=0:V=0:W=0
300  FOR K=1 TO Y
310  I=L*R
315  IF I<327.67 THEN I=(I*100)+1:I=INT(I):I=I/100
320  IF K=Y THEN T=L+I
330  W=W+T
350  S=T-I:L=L-S:N=N+I:V=V+S
355  IF D=0 THEN 370
360  P.TAB(1);K;
361  P.TAB(6);L;
362  P.TAB(16);I;
363  P.TAB(26);N;
364  P.TAB(36);S;
365  P.TAB(47);V
370  NEXT K
375  P.:P."      THE FINAL PAYMENT WILL BE $";T
376  P."(NOTE:  IF NEGATIVE NUMBER, SELECT LOWER PAYMENT)"
380  P.:P."      THE TOTAL PAYMENTS ARE $";W:P.:P.
```

```
390  IN."WANT TO DO ANOTHER ONE? YES=1 NO=0 ";D
400  IF D=1 THEN 140
410  IF D<>0 THEN 390
420  CLS:P. :P. :P. :P. "              SEE YOU AGAIN.  'BYE....."
430  P. :P. :P. :P.
440  END
```

SCREEN OUTPUT

```
MORTGAGE LOAN AMORTIZATION PROGRAM

     THIS PROGRAM WILL CALCULATE YOUR PAYMENT ON
A MORTGAGE LOAN AND TABULATE THE AMOUNT OF PRINCIPAL
AND INTEREST FOR EACH PAYMENT, AS WELL AS THE RUNNING
TOTALS. IF YOU PREFER, YOU CAN SELECT A MONTHLY
PAYMENT AND THE BALLOON PAYMENT WILL BE COMPUTED.


WHEN READY, PRESS 'ENTER' ?


WHAT IS THE AMOUNT OF THE LOAN? 50000

WHAT IS THE INTEREST RATE (%) ? 9.75

WHAT IS THE LENGTH OF THE LOAN IN MONTHS ? 358

DO YOU WANT PAYMENT CALCULATED? YES=1,NO =0? 1
THE LOAN IS FOR $ 50000 FOR 358 MONTHS AT 9.75 %

              .....PATIENCE! I'M CALCULATING.....

         THE MONTHLY PAYMENT IS $ 429.981

DO YOU WANT A MONTHLY BREAKDOWN? (YES=1,NO=0)? 1

PRESS UP-ARROW TO HOLD TABLE, RELEASE TO
CONTINUE IN LEVEL I. FOR LEVEL II, PRESS SHIFT-@ TO
STOP TABLE, ANY KEY TO CONTINUE.....

MONTH   BALANCE   INTEREST   INT.T.D.   PRINCIPAL   PRIN.T.D.
  1     49976.3    406.25     406.25     23.7305     23.7305
  2     49952.3    406.057    812.307    23.9233     47.6538
  3     49928.2    405.863    1218.17    24.1177     71.7715
  4     49903.9    405.667    1623.84    24.3136     96.0851
  5     49879.4    405.469    2029.31    24.5112     120.596
  6     49854.7    405.27     2434.58    24.7103     145.307
  7     49829.8    405.069    2839.65    24.9111     170.218
  8     49804.7    404.867    3244.51    25.1135     195.331

  351   2915.35    26.97      103839     403.011     47084.7
  352   2509.06    23.69      103862     406.291     47490.9
  353   2099.47    20.39      103883     409.591     47900.5
  354   1686.55    17.06      103900     412.921     48313.5
  355   1270.28    13.71      103913     416.271     48729.7
  356   850.63     10.33      103924     419.651     49149.4
```

```
 357  427.57     6.92       103931     423.061     49572.4
 358  0          3.48       103934     427.57      50000

     THE FINAL PAYMENT WILL BE $ 431.05
(NOTE: IF NEGATIVE NUMBER, SELECT LOWER PAYMENT)

     THE TOTAL PAYMENTS ARE $ 153935


WANT TO DO ANOTHER ONE? YES=1 NO=0 ? 0


            SEE YOU AGAIN. 'BYE.....
```

LEVEL II LISTING

```
100  REM * COPYRIGHT FRED BLECHMAN 1979 *
101 REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
103 REM * TRS-80 LEVEL II 4K, WITH PRINTER OPTION *
105  CLS:PRINT@140,"MORTGAGE LOAN AMORTIZATION PROGRAM"
110  PRINT:PRINT"      THIS PROGRAM WILL CALCULATE YOUR PAYMENT ON"
115  PRINT"A MORTGAGE LOAN AND TABULATE THE AMOUNT OF PRINCIPAL"
120  PRINT"AND INTEREST FOR EACH PAYMENT, AS WELL AS THE RUNNING"
125  PRINT"TOTALS.  IF YOU PREFER, YOU CAN SELECT A MONTHLY"
130  PRINT"PAYMENT AND THE BALLOON PAYMENT WILL BE COMPUTED."
135  PRINT:PRINT:INPUT"WHEN READY, PRESS 'ENTER' ";A$
136 CLS:PRINT:PRINT:INPUT"PRINTER ON-LINE? (YES=1,NO=0)";P
140  CLS:PRINT:PRINT:INPUT"WHAT IS THE AMOUNT OF THE LOAN";F
145  PRINT
150  INPUT"WHAT IS THE INTEREST RATE (%) ";R
155  PRINT
160  INPUT"WHAT IS THE LENGTH OF THE LOAN IN MONTHS ";Y
162  IF Y<>INT(Y) PRINT"ENTRY ERROR! TRY AGAIN":GOTO160
163  PRINT
164  INPUT"DO YOU WANT PAYMENT CALCULATED? YES=1,NO=0";D
169 CLS:PRINT@136,"THE LOAN IS FOR $";F;"FOR";Y;"MONTHS AT";R;"%"
170 IF P=1 LPRINT"THE LOAN IS FOR $";F;"FOR";Y;"MONTHS AT";R;"%"
171  IF D=1 GOTO 175
172  IF D<>0 GOTO 164
173  PRINT:INPUT"WHAT MONTHLY PAYMENT WOULD YOU LIKE ";T
175  REM CONVERT INTEREST RATE FROM PCT/YR TO DECIMAL/MONTH
180  R=R/1200
185  IF D<>1 THEN 230
190  REM CALCULATE MONTHLY PAYMENT AND ROUND UP TO CENTS
191  PRINT:PRINT"              .....PATIENCE! I'M CALCULATING....."
195  X=1+R
196  U=1
197  FOR Z=1 TO Y
198  U=U*X
199  NEXT Z
200 Q=U
210 T=F*((R*Q)/(Q-1))
215  IF T<327.67 THEN T=INT((T+.005)*100)/100
230  PRINT:PRINT"         THE MONTHLY PAYMENT IS $";T
235 IF P=1 LPRINT:LPRINT"THE MONTHLY PAYMENT IS $";T:LPRINT
240  REM * PRINT A MONTHLY TABULATION OR ONLY TOTALS *
245  PRINT
250  INPUT"DO YOU WANT A MONTHLY BREAKDOWN? (YES=1,NO=0)";D
260  IF D=0 PRINT:PRINT"      .....CALCULATING";Y;"MONTHLY PAYMENTS.  ."
```

```
261 IF D=0 PRINT"          (AT THE RATE OF 123 EVERY 10 SECONDS)"
262 IF D=0 GOTO 290
270  IF D<>1 GOTO 250
275 CLS:PRINT:PRINT"PRESS SHIFT AND @ KEYS TO STOP TABLE, ANY KEY TO CONTINUE    "
276 PRINT
280  PRINT"MONTH  BALANCE  INTEREST  INT.T.D.  PRINCIPAL  PRIN.T.D."
285 IF P=1 LPRINT"MONTH  BALANCE  INTEREST  INT.T.D.  PRINCIPAL  PRIN.T.D."
290  L=F:N=0:V=0:W=0
300  FOR K=1 TO Y
310  I=L*R
315  IF I<327.67 THEN I=(I*100)+1:I=INT(I):I=I/100
320  IF K=Y THEN T=L+I
330  W=W+T
350  S=T-I:L=L-S:N=N+I:V=V+S
355  IF D=0 THEN 370
360 PRINTTAB(1)KTAB(6)LTAB(16)ITAB(26)NTAB(36)STAB(47)V
366 IF P=1 LPRINTTAB(1)KTAB(6)LTAB(16)ITAB(26)NTAB(36)STAB(47)V
370  NEXT K
375  PRINT:PRINT"      THE FINAL PAYMENT WILL BE $";T
376  PRINT"(NOTE: IF NEGATIVE NUMBER, SELECT LOWER PAYMENT)"
378 IF P=1 LPRINT"    THE FINAL PAYMENT WILL BE $";T:LPRINT
380  PRINT:PRINT"      THE TOTAL PAYMENTS ARE $";W:PRINT:PRINT
385 IF P=1 LPRINT"    THE TOTAL PAYMENTS ARE $";W:LPRINT:LPRINT
390  INPUT"WANT TO DO ANOTHER ONE? YES=1,NO=0 ";D
400  IF D=1 THEN 140
410  IF D<>0 THEN 390
420  CLS:PRINT:PRINT:PRINT:PRINT"                SEE YOU AGAIN. 'BYE....."
430  PRINT:PRINT:PRINT:PRINT
440  END
```

PRINTER OUTPUT

```
THE LOAN IS FOR $ 50000 FOR 358 MONTHS AT 9.75 %

THE MONTHLY PAYMENT IS $ 429.981
```

MONTH	BALANCE	INTEREST	INT.T.D.	PRINCIPAL	PRIN.T.D.
1	49976.3	406.25	406.25	23.7305	23.7305
2	49952.3	406.057	812.307	23.9233	47.6538
3	49928.2	405.863	1218.17	24.1177	71.7715
4	49903.9	405.667	1623.84	24.3136	96.0851
5	49879.4	405.469	2029.31	24.5112	120.596
6	49854.7	405.27	2434.58	24.7103	145.307
7	49829.8	405.069	2839.65	24.9111	170.218
8	49804.7	404.867	3244.51	25.1135	195.331
9	49779.4	404.663	3649.18	25.3176	220.649
10	49753.8	404.457	4053.63	25.5233	246.172
37	48980.5	398.225	14889.8	31.756	1019.5
38	48948.5	397.966	15287.7	32.014	1051.52
39	48916.2	397.706	15685.5	32.2742	1083.79
40	48883.7	397.444	16082.9	32.5364	1116.33

```
41   48850.9   397.18    16480.1   32.8008   1149.13
42   48817.8   396.913   16877     33.0673   1182.19


351  2915.35   26.97     103839    403.011   47084.7
352  2509.06   23.69     103862    406.291   47490.9
353  2099.47   20.39     103883    409.591   47900.5
354  1686.55   17.06     103900    412.921   48313.5
355  1270.28   13.71     103913    416.271   48729.7
356  850.63    10.33     103924    419.651   49149.4
357  427.57    6.92      103931    423.061   49572.4
358  0         3.48      103934    427.57    50000
 THE FINAL PAYMENT WILL BE $ 431.05

 THE TOTAL PAYMENTS ARE $ 153935
```

When this is entered, line 145 skips a line, and line 15Ø asks for the interest rate, R, which should be the annual percentage. Line 155 skips a line, and line 16Ø asks for the loan period, Y, in months. Note that this is *months,* not years, since monthly payments are assumed.

Line 162 is an entry-error trap that makes sure that a fractional number is not entered. If anything but a whole number (integer) is entered, the program prints the error message and returns to line 16Ø.

Line 163 skips a line,and line 164 asks if you want the monthly payment calculated. Your response, a 1 or zero, is held in memory as variable D. Line 169 clears the screen. Line 17Ø prints a summary of your inputs starting at print location 136 (third line down, ninth space from left side, as shown in appendix A). Line 171 looks in memory to see if the variable D is a 1, which means that you want the payment calculated (see line 164). If so, the program skips over lines 172 and 173. However, if you did not type a 1 in response to line 164, line 172 now checks to see whether you entered a zero. If you entered a number less than or greater than zero, the program goes back to line 164 and asks you again.

Line 173 is used only if you entered a zero in line 164, which means that you want to enter the monthly payment yourself. Line 173 lets you do so.

Line 175 explains line 18Ø, which divides the *annual* interest rate percentage by 1ØØ (to convert from percentage to rate) and then divides again by 12 to get the *monthly* rate. It's all done at once by dividing by 12ØØ. From here on in the program, R equals the monthly interest rate.

Line 185 looks at D again. If the value is less than or greater than 1, it means that you *don't* want the payment calculated, and the program goes to line 23Ø. Let's assume, however, that you want the

payment calculated in order to avoid a "balloon" payment at the end. This calculation is performed by lines 190–215, using the formula derived earlier:

$$T = F\left(\frac{RQ}{Q - 1}\right)$$

Line 191 skips a line and then reminds the viewer that any delay is due to calculating time. Line 195 sets variable X equal to 1 plus the monthly interest rate, R. This value, (1 + R), must be raised to the Y power (see formula derivation) to give us the value of Q. First, we set variable U in line 196 equal to 1. Then line 197 sets up a loop, with variable Z counting from 1 to Y, the number of months of the loan. Line 198 establishes a new value for U equal to the old value of U (which is 1 at this point) times the value of (1 + R), which was set as X in line 195. Since line 199 sees the value of Z as 1 at this time, the program goes back to line 197, where Z is set at 2. Line 198 multiplies the last value of U by (1 + R) again and establishes a new value for U. This new value of U is actually (1 + R) raised to the second power. Can you see now that each time the program loops through lines 197–199 the value of (1 + R) is raised to the next higher power? If Y is 24, for example, then (1 + R) is raised to the 24th power before line 199 sees a value of Z greater then Y and drops through to line 200.

Why this Mickey Mouse way of raising a value to an exponent? Because Level I, unlike Level II, cannot do it directly. An elaborate subroutine is needed in Level I for raising a number to a fractional power, but if the exponent is an integer (whole number), then lines 197–199 actually provide a fast, efficient way to do the job.

Now that we have U, which is (1 + R) raised to the Y power, line 200 sets Q equal to U, since our formula uses Q for this value. Line 210 simply puts all the variables of the formula together, and the result, T, is the calculated monthly payment.

Line 215 in Level I is a tricky way to display the value of T as a two-place decimal. The line starts out with "IF T is less than 327.67. . ." The TRS-80 can't handle input integers greater than 32767. (Later in the line the value of T is multiplied by 100.) However, if T is less than 327.67, then line 215 is enabled and says, in effect, "take the present value of T, add .005 to it, then multiply the sum by 100, then divide the result by 100, and call this the new value of T." This calculation is simpler than it sounds if we consider an example. Suppose that T comes into line 215 as 137.516. First .005 is added to 137.516, making it 137.521. Now this is multiplied

by 100, making it 13752.1. The *integer* (whole number) portion is now extracted, giving 13752. This is divided by 100, and the end result, the new value of T, is 137.52. Notice how the .005 causes T to be "rounded up" to the next penny. If the entry number had been 137.514, then T plus .005 would be 137.519; the integer, after being multiplied by 100, would be 13751, and the result for T would be 137.51. This is a common method for changing decimal numbers to two places.

After all this, we're back to line 23Ø, which skips a line and prints the monthly payment—either calculated or entered. Line 24Ø gives you an indication of what's coming next. Line 245 skips a line, and line 25Ø asks if you want a monthly breakdown. Lines 26Ø–262 look to see if you entered a zero (for NO). If you did, the computer tells you that it's calculating and jumps to line 29Ø. If you answer line 25Ø with a 1 (for YES), it means you want a tabulation. Line 27Ø makes sure that you didn't enter something other than 1 by mistake; if you did, back to line 25Ø! Of course, if you enter a zero, line 262 causes the program to bypass the entry-error trap in line 27Ø.

We'll assume that you want a breakdown. Line 275 clears the screen, skips a line, and then prints instructions on how to stop the tabulation about to take place. For Level I, holding down the up-arrow halts the program until the up-arrow key is released. Line 276 skips another line, and line 28Ø prints the tabulation headings. "T.D" means "TO DATE," or totals, including the last shown payment.

Line 29Ø sets variable L equal to F (principal) and variables N, V, and W to zero. Line 3ØØ starts a loop to make the monthly interest calculations, determine the table values, and print them. Variable K is the loop counter, which starts at 1 and goes to Y, the number of loan months.

Line 31Ø takes the initial principal, L, multiplies it by the monthly interest rate, R, and sets this product equal to the monthly interest, I. Line 315 is merely another way of making I a two-place decimal. (Actually, the "plus 1" in this line should be "plus .5" for greater accuracy.)

Line 32Ø checks to see if the loop has reached the last month yet. If so, it makes T, now used for the final payment, equal to the last balance plus interest. Line 33Ø sets variable W equal to the last value of W plus the monthly payment. Line 35Ø updates various tabular values. Variable S is set equal to the monthly payment minus the interest—that's the amount going to pay off the principal for that month. The new balance, L, equals the last balance minus the principal paid this month. The total interest paid to date, N, is equal

to the previous interest total plus the latest interest payment. The total principal paid to date, V, is the last total principal paid plus the latest monthly principal paid.

Line 355 looks at variable D to see if you wanted a tabulation. If D is equal to zero, you didn't, and thus the next lines are bypassed and the program jumps to line 37Ø, where the loop is repeated.

However, since we're assuming that you *do* want a tabulation, line 355 is ignored by the program (D is equal to 1), and lines 36Ø–365 print out the tabular values under the headings, using TAB statements. On each of these lines, the semicolon *before* each variable is optional, but don't omit the semicolon *after* K, L, I, N, and S. Actually, these statements could all be put on one program line *without any* semicolons, as you'll see in the Level II version.

Line 37Ø repeats the loop until K is greater than Y, calculating and displaying a new line of values each time.

Finally, the program moves on to line 375, where a line is skipped and the final payment (calculated in line 32Ø) shown. Line 376 is printed to remind you that, if you *selected* a monthly payment arbitrarily and this final payment is a negative number, you should try again with a lower payment amount.

Line 38Ø skips a line, prints the total payment, and then skips two more lines. Lines 39Ø–44Ø allow you to use the program again or to exit.

Level II offers many features superior to those of Level I, including PRINT USING (to format numbers), LPRINT (to printer instead of screen), and direct calculation of exponents. Rather than extensively reprogram for Level II, however, we'll just use the same framework and line numbers, examining only those changes that must be made to the Level I program. This will show you how easy, relatively speaking, it is to convert a Level II-compatible program. Refer to both the Level II listing and its printer output during this explanation of the Level I/Level II conversion.

To begin with, most abbreviations are not recognized by Level II. Therefore, where the Level I listing shows P., Level II requires PRINT. Likewise, IN. becomes INPUT and P.AT must be changed to PRINT@. (Note: Do *not* hold down the shift-key when pressing the @ key!) In Level I, P.AT number can be followed by a semicolon or a comma. In Level II, PRINT@ number must be followed by a comma, *not* a semicolon. Line 1Ø3 identifies this version of the program.

Level II allows the use of a printer, and the command LPRINT will direct the output to the printer instead of the screen.

However, there will be a problem if you don't have a printer or printer-driver software "online," that is, connected to the TRS-80 or loaded in memory. Any LPRINT statement causes the TRS-80 to look for the printer. If the printer isn't there, the TRS-80 "locks up" in an endless loop, the program will not move forward, and you have no control at the keyboard. You must press the RESET button (left rear of keyboard, under the expansion port PC board) to regain control. Fortunately, this situation does not destroy your program, which remains in memory, but it does stop the program from running past the LPRINT statement.*

To avoid it, you must tell the TRS-80 when to ignore LPRINT statements. Line 136 allows you to do so. If you respond to variable P with a zero (for "NO, there is no printer on line"), then the program jumps over all line numbers containing an LPRINT statement, since LPRINT is preceded by "IF P = 1" (lines 17Ø, 235, 285, 366, 378, and 385). If P is 1 (for "YES, a printer is on line"), then these lines are operative.

Notice that not all PRINT statements have LPRINT statements following. The printer should summarize the program results and need not print every question. For example, line 17Ø prints the results of the entries made in lines 14Ø, 15Ø, and 16Ø.

Look at line 36Ø. This is a combination of lines 36Ø–365 of the Level I program with all semicolons removed.

Changes were also made to line 169 (added PRINT statement), line 17Ø (P.AT becomes LPRINT), line 261 (different calculation rate), and line 275 (different operator instruction, since in Level II you must press the SHIFT and @ key *simultaneously* to halt program operations).

You can see, then, that getting a Level I program to run properly in Level II is not always (in fact, not even usually) just a matter of running the Level I program through the Level I program conversion tape supplied with Level II.

Modifications: You might prefer to have the screen display show a "page" of the tabulation and then stop. To do so, add a loop counter that allows, for example, 12 lines of the tabulation to be shown at one time on the screen. (See Program 12 for an example of how this works.) Typically, mortgage loans are for over 120 months, and formatting 12 lines at a time can be a bother. It should definitely be avoided when printer output is enabled.

*If you have an expansion interface connected to your keyboard but do *not* have a printer on line, the RESET button will *destroy* your program in memory. Be careful!

Variables:

A$—pause control
P—printer control
F—amount of loan
R—annual interst rate, monthly interest rate
Y—length of loan in months
D—payment calculation option
T—monthly payment
X—(1 + R)
U—running value of $(1 + R)^Y$
Z—exponent counter
Q—final value of $(1 + R)^Y$
L—monthly balance still due
N—interest paid to date
V—principal paid to date
W—total of all payments
K—month number
I—interest part of payment
S—principal part of payment

PROGRAM 15
PAY NOW OR PAY MONTHLY?
(4K, Level I or II)

You'll learn: Proper use of explanatory text; zone-printed multiple-line tabulation headings; subroutine loop calculations.

Description: If you pay the full amount of a doctor, dentist, or lawyer bill, you will lose the interest this money would make if saved in your own bank account (professionals rarely charge you interest). This program allows you to determine how much you can save each month by deferring payments and the point at which this saving becomes so insignificant that you should pay off the balance due.

Explanation: Lines 9Ø–96 identify the program author and the equipment needed and also keep the Level I version from overscrolling (as explained in Program 11).

Line 1ØØ clears the screen, and lines 11Ø–22Ø explain the purpose of the program. Since this program is short, even a 4K memory will hold a lot of explanatory text, thus allowing the user to brief himself quickly rather than having to look for documentation elsewhere. Where memory is not a problem, providing such text is a good idea.

Line 22Ø waits for the user to hit ENTER, thus holding the text on the screen. Lines 225–25Ø clear the screen, skip two lines, and then ask the three basic questions for which the computer needs input. B is the balance owed, M is the desired monthly payment, and I is the annual savings interest rate percentage. Line 26Ø divides the amount owed by the intended monthly payment, and the result is reduced to the next smallest whole number by the INT (short for "integer") statement. This value, D, is the number of full monthly payments needed to pay off the bill. Line 27Ø divides the annual interest rate, I, by 100 (since it was entered as a percentage) and by 365 (daily compounded interest is virtually standard at savings institutions these days). It does this all at once by dividing by 365 times 100, or 36500. The result, R, is the daily interest rate.

```
90 REM * THESE FEW LINES AT THE TOP OF THE LISTING.....
91 REM .....KEEP THE LISTING FROM SCROLLING UPWARD.....
92 REM .....IN LEVEL I, UNTIL UP-ARROW ENTERED.. *
93 REM * TRS-80 LEVEL I OR LEVEL II 4K *
95 REM * COPYRIGHT FRED BLECHMAN 1978 *
96 REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
100  CLS
110  PRINT:PRINT"                    PAY NOW OR PAY MONTHLY???"
120  PRINT
130  PRINT"      THE OBJECTIVE OF THIS PROGRAM IS TO HELP YOU MAKE THE "
140  PRINT"CHOICE OF WHETHER TO PAY THE DOCTOR OR DENTIST BILL"
150  PRINT"(OR ANY OTHER BILL THAT DOESN'T CHARGE INTEREST) IN ONE LUMP"
160  PRINT"SUM, OR MONTHLY PAYMENTS OF AN AMOUNT AGREED UPON.  SINCE "
170  PRINT"THE MONEY YOU LEAVE IN YOUR SAVINGS ACCOUNT WILL EARN "
180  PRINT"INTEREST DURING THE PAY-BACK PERIOD, YOU'LL ALWAYS COME"
190  PRINT"OUT AHEAD BY MAKING PAYMENTS, BUT YOU MAY THINK PAYMENTS "
200  PRINT"ARE TOO MUCH BOTHER.  THIS PROGRAM WILL GIVE YOU THE"
210  PRINT"FACTS ON WHICH YOU CAN BASE THE DECISION....."
215  PRINT"INCIDENTALLY, MAKE THE SMALLEST PAYMENTS ALLOWED. "
220  PRINT:INPUT"WHEN READY TO START, PRESS ENTER";A$
225  CLS:PRINT:PRINT
230  INPUT"WHAT IS THE AMOUNT OF MONEY YOU OWE";B
240  INPUT"WHAT IS THE MONTHLY PAYMENT YOU INTEND TO MAKE";M
250  INPUT"WHAT IS YOUR SAVINGS INTEREST RATE(%)";I
260  D=INT(B/M)
270  R=I/36500
280  PRINT"PAYMENT","BALANCE","LEFT IN","INTEREST"
290  PRINT"NUMBER","REMAINING","SAVINGS","THIS MONTH"
300  X=0:V=B:G=B
310  GOSUB 500
320  FOR X=1 TO D
330  V=V-M:B=B-M
340  GOSUB 500
350  NEXT X
360  PRINT
370  PRINT"      PAYING THE TOTAL OF $";G;"AT $";M;"PER MONTH"
380  PRINT"WILL TAKE";D;"MONTHS, AND A FINAL PAYMENT OF $";B;" "
390  PRINT:PRINT"TOTAL INTEREST SAVED IS $";V-B;"COMPARED TO PAYING"
400  PRINT"A SINGLE LUMP SUM WHEN ORIGINALLY BILLED. "
410  PRINT:PRINT"      REMEMBER - YOU MAY WANT TO PAY THE BALANCE "
420  PRINT"OFF COMPLETELY WHEN THE MONTHLY INTEREST BECOMES SMALL..."
425  PRINT
430  INPUT"WANT TO TRY ANOTHER RUN (YES=1,NO=0)";C
440  IF C=1 THEN 225
450  PRINT"SEE YOU SOME OTHER TIME.  'BYE....."
460  END
500  REM * CALCULATE DAILY INTEREST FOR 1 MONTH (30 DAYS) *
505  T=0
510  FOR Y=1 TO 30
520  S=V*R
530  V=V+S
540  T=T+S
550  NEXT Y
560  PRINTX,B,V,T
570  RETURN
```

Lines 280 and 290 print four two-line column headings. Since the commas between the printing in quotes tell the computer to use zone printing, no TAB statements are needed, and the second line of each column heading starts right below the first line. This is a

PROGRAM 15 RUN

```
                PAY NOW OR PAY MONTHLY???

     THE OBJECTIVE OF THIS PROGRAM IS TO HELP YOU MAKE THE
CHOICE OF WHETHER TO PAY THE DOCTOR OR DENTIST BILL
(OR ANY OTHER BILL THAT DOESN'T CHARGE INTEREST) IN ONE LUMP
SUM, OR MONTHLY PAYMENTS OF AN AMOUNT AGREED UPON. SINCE
THE MONEY YOU LEAVE IN YOUR SAVINGS ACCOUNT WILL EARN
INTEREST DURING THE PAY-BACK PERIOD, YOU'LL ALWAYS COME
OUT AHEAD BY MAKING PAYMENTS, BUT YOU MAY THINK PAYMENTS
ARE TOO MUCH BOTHER. THIS PROGRAM WILL GIVE YOU THE
FACTS ON WHICH YOU CAN BASE THE DECISION.....
INCIDENTALLY, MAKE THE SMALLEST PAYMENTS ALLOWED.

WHEN READY TO START, PRESS ENTER?


WHAT IS THE AMOUNT OF MONEY YOU OWE? 1000
WHAT IS THE MONTHLY PAYMENT YOU INTEND TO MAKE? 25
WHAT IS YOUR SAVINGS INTEREST RATE(%)? 5.25
PAYMENT          BALANCE          LEFT IN          INTEREST
NUMBER           REMAINING        SAVINGS          THIS MONTH
 0                1000             1004.32          4.32408
 1                975              983.559          4.23468
 2                950              962.704          4.14489
 3                925              941.759          4.05471
 4                900              920.723          3.96414
 5                875              899.596          3.87318
 6                850              878.378          3.78182
 7                825              857.068          3.69007
 8                800              835.666          3.59793
 ↯                ↯                ↯                ↯
 38               50               148.716          .64029
 39               25               124.251          .534957
 40               0                99.6798          .429168

     PAYING THE TOTAL OF $ 1000 AT $ 25 PER MONTH
WILL TAKE 40 MONTHS, AND A FINAL PAYMENT OF $ 0 .

TOTAL INTEREST SAVED IS $ 99.6798 COMPARED TO PAYING
A SINGLE LUMP SUM WHEN ORIGINALLY BILLED.

     REMEMBER - YOU MAY WANT TO PAY THE BALANCE
OFF COMPLETELY WHEN THE MONTHLY INTEREST BECOMES SMALL...

WANT TO TRY ANOTHER RUN (YES=1,NO=0)? 0
SEE YOU SOME OTHER TIME. 'BYE.....
```

useful technique when several lines are needed for column headings. Remember, however, that zone printing on the TRS-80 is limited to four zones, with starting points at print locations Ø, 16, 32, and 48 on each line.

Once the headings are nicely printed, we need to do some calculations to put figures under those headings. Line 3ØØ sets variable X to zero and variables V and G equal to the balance owed,

B. Variable G will be retained in memory unchanged for use later in line 37Ø. Variable V will be used to show how much the balance, left in savings, will grow each month as interest is added.

Line 31Ø sends the program to the subroutine starting at line 5ØØ and ending at line 57Ø. This subroutine assumes that daily interest is compounded for 30 days each month, thus introducing a very slight error, since some months have 31 days. Since the nature of this program is not critical, this simplification makes sense. The alternative of entering the billing start date and providing additional programming to specify compounding for the specific number of days in each month would be overkill. In the banking industry, such a simplification would be a no-no; for us, it makes sense.

Line 5Ø5 sets the accumulated daily interest, T, to zero. Line 51Ø starts a loop, using Y as the loop counter, running from 1 to 30 (30 days). Lines 52Ø–54Ø take the latest balance owed, V, and mutliply it by the daily interest rate, R, to get the daily interest, S. This is added to the previous balance to get a new and slightly higher balance, V. The daily interest, S, is added to the accumulated daily interest, T, for a new T.

This sequence (lines 52Ø–54Ø) is repeated 30 times by the loop counter (line 55Ø). After 30 calculations, the payment number, X, the balance remaining, B, the amount left in savings, V, and the monthly interest, T, are printed out under the column headings by line 56Ø. Line 57Ø returns the program to line 31Ø, where it had branched away. Since there is no additional instruction on line 31Ø, the program moves on the line 32Ø, the beginning of another loop. This loop counts from 1 to D, the number of payments, using X as the counter. Line 33Ø deducts the monthly payment from the bank balance for a new balance (since we assume you'll make this payment from your savings account). Also, the balance owed, B, is reduced by the monthly payment. Then line 34Ø sends the program to the subroutine again for another 30-day interest calculation and prints another line of results. This process continues until X in line 35Ø exceeds D. Then the program falls though to lines 36Ø–46Ø, which print the summary information and allow you to do another calculation or exit the program.

Modifications: You could complicate the program considerably by printing more columns of data, but why bother? You could also add one line (line 555) and change another (line 56Ø) to format the "Left In Savings" column to dollars and cents, as follows:

```
555  Z=V: Z=INT(Z*1ØØ+.5)/1ØØ
56Ø  PRINTX,B,Z,T
```

[*Note:* Line 555 is *yet another* way to use the INT (integer) statement to round off numbers. Two other way were shown in Program 14. Also notice that the value of V is *not* rounded off, since it is used in calculations.]

Variables:

A$—pause control
B—amount owed
M—monthly payment
I—annual interest, percent
D—number of monthly payments
R—daily interest rate
X—loop counter (payments)
V—savings balance
G—original amount owed
C—rerun control
T—accumulated daily interest
Y—loop counter (days)
S—daily interest

PROGRAM 16
PHONE TOLL-CHARGE PROGRAM
(4K, Level I or II)

You'll learn: Telephone billing structure; using a timing loop for a practical purpose; combining graphics with text; bypassing instructions.

Description: This program may be used whenever you make a call out of your local area. The display shows both the time and charges for the call and "counts down" by the second to the next charge. Instructions and information to reduce the expense of your calls are included in the program text.

Explanation: This program will operate in Level I or Level II. For Level I, the regular abbreviations may be used, and PRINT@ should be PRINTAT.

Lines 5–22 introduce the program, with line 22 waiting for your input. Type and ENTER either a number or a name since B$ is nonfunctional in this program; it is used later on (in line 6Ø) merely to show the called name or number on the screen, as a reference.

Line 25 allows you to select or bypass instructions. If you type and ENTER a 1 (for YES), line 26 sends the program to lines 3ØØ–4ØØ. At line 4ØØ you select or bypass discount information. If you type and ENTER a 2 (for NO), line 41Ø sends the program back to line 28. If you type and ENTER a 1 (for YES), the program continues through lines 42Ø–48Ø, and line 49Ø sends the program back to line 28. Back in line 25, if you had typed and ENTERed a 2 (for NO), the program would ignore line 26 and proceed to line 28. Thus, no matter *what* you do, you'll eventually get to line 28.

Line 28 clears the screen and skips three lines. Line 29 prints the program title on the screen and skips two lines. Line 3Ø asks for the initial time period, which will be either 1 or 3 minutes—1 minute if you dial directly, 3 minutes if you use an operator to assist you. Type and ENTER a 1 or 3; this number is then assigned to variable P. Line 35 skips two screen lines, and

LEVEL I/II LISTING

```
5 REM * TRS-80  LEVEL I OR LEVEL II 4K *
10  REM * COPYRIGHT 1978 FRED BLECHMAN *
11  REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
12  REM * LINE 225 CONTROLS THE SECOND-COUNTING ACCURACY *
13  CLS:PRINT:PRINT:PRINT
14  PRINT"                PHONE TOLL-CHARGE PROGRAM":PRINT:PRINT
15  PRINT"    USE THIS PROGRAM WHENEVER YOU MAKE A TELEPHONE CALL"
16  PRINT"OUT OF THE LOCAL AREA - ESPECIALLY LONG DISTANCE. THE DISPLAY"
17  PRINT"WILL SHOW YOU HOW MANY MINUTES YOU HAVE BEEN CHARGED,"
18  PRINT"THE COST SO FAR, AND THE NUMBER OF SECONDS TO THE NEXT"
19  PRINT"ADDITIONAL CHARGE, COUNTING DOWN........"
20  PRINT:PRINT
22  INPUT"WHO ARE YOU CALLING.....";B$
25  INPUT"DO YOU WANT SPECIFIC INSTRUCTIONS? YES=1,NO=2";A
26  IF A=1 GOTO 300
28  CLS:PRINT:PRINT:PRINT
29  PRINT"                    TELEPHONE TOLL TOTALIZER":PRINT:PRINT
30  INPUT "WHAT IS THE INITIAL TIME PERIOD (MINUTES)";P
35  PRINT:PRINT
40  INPUT"WHAT IS THE INITIAL CHARGE (CENTS)";I
45  PRINT:PRINT
50  INPUT"WHAT IS THE ADDITIONAL CHARGE PER MINUTE (CENTS)";M
51  CLS:PRINT:PRINT:PRINT:PRINT
55  PRINT "WHEN THE PARTY AT OTHER END PICKS UP THE RECEIVER"
57  INPUT"     PRESS ENTER TO START TIMING....";A$
60  CLS:PRINT:PRINT"           ON YOUR CALL TO  ";B$;"....."
70  PRINT"THE INITIAL COST OF THE FIRST";P;"MINUTE(S) IS: $";I/100
80  PRINT"     THE ADDITIONAL COST PER MINUTE IS:$";M/100
90  PRINT@ 847,"PRESS BREAK TO STOP COUNTING."
100  C=I/100:D=1
110  FOR X=22 TO 93
115  SET(X,25):SET(X,31):NEXT X
120  FOR Y=25 TO 31
125  SET (22,Y): SET(93,Y):NEXT Y
130  T=57
140  IF P=3 THEN T=177
150  GOTO 221
200  PRINT@ 590,"THE TOTAL CHARGE IS NOW:$";C;
210  SET(93,27):SET(93,28):SET(93,29)
215  PRINT@ 715,"TOTAL TIME CHARGED IS NOW:";P+D;"MINUTES"
220  T=59:D=D+1
221  C=C+(M/100)
222 REM * LINE 225 COUNT: 485 FOR LEVEL I, 263 FOR LEVEL II *
223 REM * CHANGE THIS COUNT IF NECESSARY FOR ACCURACY *
225 FOR X=1 TO 263:NEXT X
230  PRINT@ 330,"SECONDS TO THE NEXT ADDITIONAL CHARGE:";T
240  T=T-1
250  IF T=-1 GOTO 200
255  GOTO 225
300  CLS:PRINT"YOUR CHARGES ARE BASED UPON THREE THINGS:"
310  PRINT"     (1) INITIAL TIME PERIOD (1 OR 3 MINUTES)"
320  PRINT"     (2) INITIAL CHARGE (FOR THE INITIAL PERIOD)"
330  PRINT"     (3) ADDITIONAL CHARGE PER MINUTE AFTER INITIAL PERIOD"
340  PRINT:PRINT"IF YOU USE AN OPERATOR TO ASSIST YOU, THE INITIAL"
350  PRINT"TIME PERIOD IS 3 MINUTES. DIRECT DIAL IS 1 MINUTE."
360  PRINT:PRINT"THE CHARGES ARE BASED ON THE DESTINATION CALLED...."
370  PRINT"....THESE ARE USUALLY LISTED IN THE FRONT OF PHONE BOOK"
380  PRINT"...OR....CALL OPERATOR FOR THE RATES."
390  PRINT" DO YOU WANT INFORMATION ON DISCOUNT PERIODS?"
400  PRINT:INPUT"YES=1, NO=2";B
410  IF B=2 GOTO 28
420  CLS:PRINT"THERE ARE TWO DISCOUNT RATES IN THE CONT. U.S.A.:"
430  PRINT"    35% DISCOUNT:  5PM - 11PM SUNDAY - FRIDAY"
440  PRINT"                   8AM - 11PM HOLIDAYS":PRINT
```

```
450  PRINT"    60% DISCOUNT:  11PM - 8AM EVERY NIGHT"
460  PRINT"                   8AM - 11PM SATURDAY"
470  PRINT"                   8AM -  5PM SUNDAY"
475  PRINT:PRINT"CHARGES ARE BASED ON TIME AT CALLING POINT!"
476  PRINT:PRINT:PRINT
480  INPUT"PRESS ENTER TO INPUT TIME AND CHARGE DATA....";A$
490  GOTO 28
```

PROGRAM 16 RUN

```
    USE THIS PROGRAM WHENEVER YOU MAKE A TELEPHONE CALL
OUT OF THE LOCAL AREA - ESPECIALLY LONG DISTANCE. THE DISPLAY
WILL SHOW YOU HOW MANY MINUTES YOU HAVE BEEN CHARGED,
THE COST SO FAR, AND THE NUMBER OF SECONDS TO THE NEXT
ADDITIONAL CHARGE, COUNTING DOWN.......


WHO ARE YOU CALLING.....? HARRY JONES
DO YOU WANT SPECIFIC INSTRUCTIONS? YES=1,NO=2? 1

YOUR CHARGES ARE BASED UPON THREE THINGS:
     (1) INITIAL TIME PERIOD (1 OR 3 MINUTES)
     (2) INITIAL CHARGE (FOR THE INITIAL PERIOD)
     (3) ADDITIONAL CHARGE PER MINUTE AFTER INITIAL PERIOD

IF YOU USE AN OPERATOR TO ASSIST YOU, THE INITIAL
TIME PERIOD IS 3 MINUTES. DIRECT DIAL IS 1 MINUTE.

THE CHARGES ARE BASED ON THE DESTINATION CALLED....
....THESE ARE USUALLY LISTED IN THE FRONT OF PHONE BOOK
...OR....CALL OPERATOR FOR THE RATES.
 DO YOU WANT INFORMATION ON DISCOUNT PERIODS?

YES=1, NO=2? 1


THERE ARE TWO DISCOUNT RATES IN THE CONT. U.S.A.:
    35% DISCOUNT:  5PM - 11PM SUNDAY - FRIDAY
                   8AM - 11PM HOLIDAYS

    60% DISCOUNT:  11PM - 8AM EVERY NIGHT
                   8AM - 11PM SATURDAY
                   8AM -  5PM SUNDAY

CHARGES ARE BASED ON TIME AT CALLING POINT!


PRESS ENTER TO INPUT TIME AND CHARGE DATA....?



                  TELEPHONE TOLL TOTALIZER

WHAT IS THE INITIAL TIME PERIOD (MINUTES)? 1

WHAT IS THE INITIAL CHARGE (CENTS)? 22
```

```
WHAT IS THE ADDITIONAL CHARGE PER MINUTE (CENTS)? 16

WHEN THE PARTY AT OTHER END PICKS UP THE RECEIVER
     PRESS ENTER TO START TIMING....?



          ON YOUR CALL TO  HARRY JONES.....
THE INITIAL COST OF THE FIRST 1 MINUTE(S) IS: $ .22
     THE ADDITIONAL COST PER MINUTE IS:$ .16

          SECONDS TO THE NEXT ADDITIONAL CHARGE: 47

     THE TOTAL CHARGE IS NOW:$ .38
TOTAL TIME CHARGED IS NOW: 2 MINUTES

     PRESS BREAK TO STOP COUNTING.
```

line 40 asks for the initial charge. The front section of your telephone book shows rates for various nonlocal calls. They are shown for both operator-assisted and unassisted calls and for three different time periods. The time periods are based on the time at the calling point. Find the rates that apply to your specific call (location called, assisted or unassisted, time-of-day). There will be one rate for the first period and another rate for "each additional minute." In our program, these are assigned variables I (initial charge) and M (additional minutes rate). Type and ENTER the initial rate (no decimal) in response to line 40. Line 45 skips two lines, and line 50 asks for the minute rate, also entered without a decimal.

Line 51 clears the screen and prints four blank lines. Lines 55 and 57 tell you to press ENTER when the party at the other end picks up the phone, thus starting the TRS-80 timing at the same time as the telephone company billing equipment without any connection to the phone lines!

As soon as you press ENTER, the screen is cleared again by line 60. Lines 60-90 summarize your inputs near the top of the screen and print PRESS BREAK TO STOP COUNTING near the bottom of the screen.

Line 100 establishes the value of variable C as variable I divided by 100, thus converting the initial charge, entered as a whole number, to cents and saving you the trouble of entering a decimal point in response to line 40. Also, variable D, used to keep track of elapsed minutes, is set to 1, since from the instant the receiving party picks up the phone, you are charged for *at least* one minute.

Lines 115–125 "draw" a rectangular box n the screen with SET statements. (Go back to Program 1 for an explanation of how this works.)

Line 13Ø sets variable T, the seconds counter, to 57. Why not 59? Because from the time you press ENTER (line 57) until the appearance of the seconds counter on the screen (line 23Ø) about 2 seconds elapse, reducing the count down from 59 to 57. However, the next line, 14Ø, looks at the value of P that was entered in line 3Ø. If the initial time period is 3 minutes, then T is set to 177 instead of 57 (3 minutes is 180 seconds). In either case, line 15Ø sends the program to line 221.

Line 221 first sets a new value for C equal to the initial charge (lines 4Ø and 1ØØ) plus the cents value of the additional charge M (entered in line 5Ø). This, in effect, establishes the total charge for the initial time period plus one additional minute, the result being held in memory as variable C but not yet displayed.

Lines 222 and 223 give you the approximate loop-counter number to use in line 225, the seconds counting loop. Be warned that Level I and Level II complete a single loop at different rates. Moreover, the specific rate for your TRS-80 will depend on actual circuit constants and the equipment temperature. The TRS-80 was not designed to be used as a clock, but you'll find it fairly accurate if you "customize" the loop-counter number. Suppose, for example, that you find the seconds counting fast by comparison with a real-time clock. If so, just increase the loop-counter number. If your program is counting seconds too slowly, decrease the counter number. Change the number one digit at a time, and RUN the program after each change to see the effect. You'll have to let it run at least a minute, when you're close to the correct timing, to see much deviation.

Incidentally, you'll find that the timing of this loop is affected by earlier program entries during that run. The memory builds a variable "look-up" table, and you *cannot* use GOTO 6Ø or RUN 6Ø to avoid test entries. If you do, your "clock" will run faster than it would during a normal run, which has more variables in memory.

Anyhow, don't expect perfection, but you should be able to get the timing accurate to within one-half second per minute. Remember, however, that this accuracy will vary slightly with temperature.

At the completion of the first line-225 timing loop, line 23Ø prints the value of T (which starts at 57 or 177, from lines 13Ø and 14Ø). Line 24Ø now subtracts 1 from T for a new T of 56. Line 25Ø

checks to see if T has gone below zero. Since it hasn't, line 255 sends the program back to the timing line, line 225. As the program continues circulating between lines 225 and 255, the seconds —as displayed by line 23Ø—count down to zero. Now line 24Ø makes T equal to −1; line 25Ø notes this negative value and returns the program to line 2ØØ.

Line 2ØØ prints the total charge, C, which has been held in memory from the last pass through line 221. Line 21Ø replaces the right border of the rectangle if it has been blanked out by a large value of C, and, right below the rectangle, line 215 prints the total time charged. The value P + D is equal to the initial time period plus the additional minutes counted so far. The next line, 22Ø, resets the seconds count to 59 (to start another minute countdown) and increments the additional-minutes counter, D, by one.

This whole process of counting down and adding minutes and charges continues until you hit the BREAK key. When you do so, the BREAK command wipes out two screen lines in Level I, or three screen lines (including the top line of the rectangle) in Level II. In either case, since all the data remains on the screen, you can keep a written record in a toll-call log to compare with your phone bill.

Modifications: A major modification of this program is to add automatic phone dialing when you type in a name, with redial capability if the number dialed is busy. Manual dialing and alphabetical screen listing of up to 500 names in memory can also be added. To do so requires a Level II 16K TRS-80 and a simple telephone interface (under $5) using five common Radio Shack parts built into a small plastic box. No modification whatever is required to the TRS-80. (This "AUTO-DIALER II" program, together with the Phone Toll-Charge program, is available on cassette, with documentation, for $10, postpaid USA, from the author at 7217 Bernadine Avenue, Canoga Park, CA 91307.)

Variables:

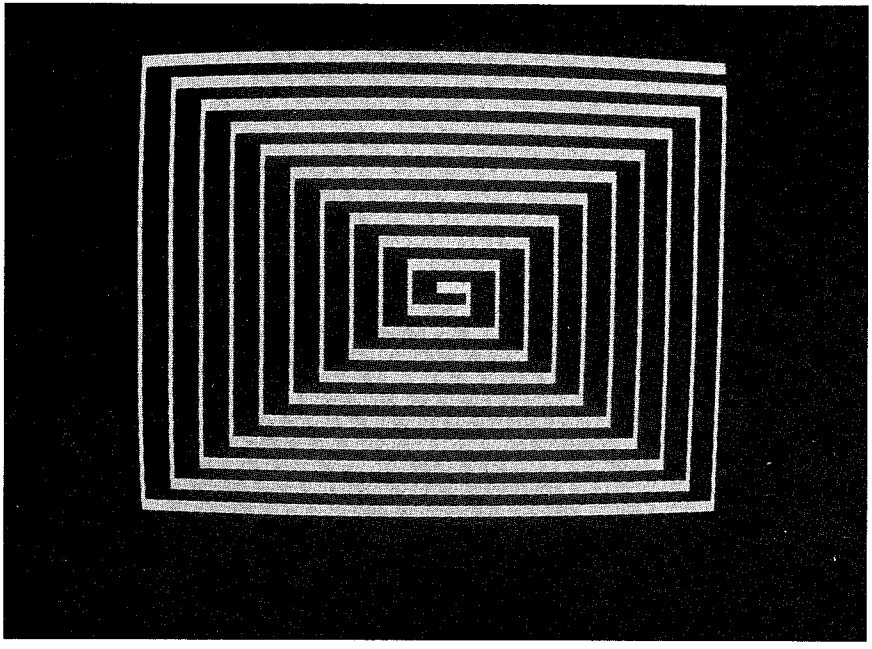
B$—name or number called
A—instruction call
P—initial time period (minutes)
I—initial charge (cents)
M—additional charge per minute (cents)
A$—pause control

C—total charge (dollars)
D—additional minutes
X—rectangle horizontal coordinates; timing loop counter
Y—rectangle vertical coordinates
T—countdown (seconds)
B—discount info call

PROGRAM 17
SPIRANGLE
(4K, Level I or II)

You'll learn: How to draw a simple graphic pattern on the screen by using loops

Description: The "spirangle" is a rectangular-shaped spiral that starts near the center of the screen and progresses clockwise until it fills the screen and "freezes."



Explanation: Lines 5–8 introduce the listing with REM statements. The program is identical for Levels I and II, with the listing shown for Level II (no abbreviations).

Line 1Ø clears the screen. Line 2Ø assigns values to variables X and Y. If you check these on the Video Display Worksheet in Appendix A, you'll see that these values fall near

```
5 REM * COPYRIGHT FRED BLECHMAN 1978 *
6 REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
7 REM * SPIRANGLE PROGRAM *
8 REM * FOR TRS-80 LEVEL I OR LEVEL II 4K *
10  CLS
20  X=62:Y=23
25  A=6:B=2
30  FOR Z=1 TO A
40  SET(X,Y)
45  X=X+1
46  IF X=123 GOTO 170
50  NEXT Z
60  FOR Z=1 TO B
70  SET (X,Y)
75  Y=Y+1
80  NEXT Z
85  C=A+6
90  FOR Z=1 TO C
100  SET(X,Y)
105  X=X-1
110  NEXT Z
115  D=B+2
120  FOR Z= 1 TO D
130  SET(X,Y)
140  Y=Y-1
150  NEXT Z
155  A=C+6:B=D+2
160  GOTO 30
170  GOTO 170
```

the center of the screen (X is 63 units from the left, out of a possible 128, and Y is 24 units down, out of a possible 48. Remember that Ø is the starting number in each case.)

Next line 25 sets variable A equal to 6 and variable B equal to 2. These are horizontal and vertical limits for the first two "legs" of the graphic pattern to be generated. Line 3Ø starts a FOR-NEXT loop with Z as the variable. It says, in effect, "Let Z vary from 1 to 6 in steps of 1." You see, A is equal to 6, and since no other STEP is specified, then 1 is automatically used.

Line 4Ø tells the computer to light the graphic block (SET) at location X across and Y down (that's 62 and 23) from line 2Ø.

Now line 45 increases the value of X by 1, to 63. Line 46 checks to see if the "drawing" is complete yet; it looks for an X value of 123. Since X is only 63 at this time, the program ignores this line. Line 5Ø looks at the value of Z to see if it has yet reached 6 (the value of A in line 3Ø, established in line 25). Since it hasn't, the program returns to the beginning of the Z loop, line 3Ø, and Z increases by 1 to a value of 2. Line 4Ø now lights graphic block 63 across and 23 down. Notice that X has increased by 1 (line 45) but Y has remained the same.

This process continues for a total of six times (since A equals 6 from line 25), at which point graphic block (67,23) is SET, X is advanced to 68 by line 45, line 46 is still not valid, and line 5Ø sees Z equal to 6 and thus allows the program to proceed to line 6Ø.

Confused? If so, go back over this loop in lines 3Ø–5Ø again, step by step, since it is a relatively simple example of an incrementing loop. Line 3Ø advances Z each time to count the number of main loops, whereas X is advanced each time by line 45. This form of looping is used throughout the rest of the program.

Line 6Ø resets loop-counter Z to 1 and tells the compuer to advance it by steps of 1 to the value of B, which was established as 2 in line 25. Line 7Ø now lights graphic block (68,23). Notice that Y has not changed and that X was advanced to 68 by line 45 during the last pass through the loop in lines 3Ø–5Ø.

Now line 75 advances Y by 1, to 24, but X is *not* changed. Since line 8Ø sees that Z has not yet reached 2, the program goes back to line 6Ø for another loop, and Z is advanced to 2. Next, the graphic block (68,24) is lighted by line 7Ø, and Y is then advanced to 25 by line 75. Since line 8Ø now sees Z equal to 2, the program continues to line 85.

A new variable, C, is introduced in line 85 and set equal to the current value of A (which is 6) plus 6, or 12. Line 9Ø starts another Z loop, using C as the maximum count (12). Line 1ØØ lights graphic block (68,25), and line 1Ø5 *subtracts* 1 from X for a new value of 67. In effect, the computer is counting *backwards,* and the graphic blocks are lighted from *right to left.* This process goes on for 12 counts, and the program then goes to line 115, where another new variable, D, is established equal to the present value of B (which is 2) plus 2, or 4.

Since loop 12Ø–15Ø subtracts 1 from the value of Y each time through, the graphic blocks are lighted in an upward-moving sequence.

So far, we've completed a right-going leg, using A to establish the number of blocks to be lighted; a down-going leg, with B counting; a left-going leg, with C counting; and an upward-going leg, with D counting. Thus, to continue, line 155 merely adds a constant value of 6 to the last value of A (which gives horizontal limits) and a constant value of 2 to the last value of B (which gives vertical limits). The program now is sent back to line 3Ø by line 16Ø, with the new values for A and B in

memory. The "backward" counting variables C and D are derived from the latest A and B values (lines 85 and 115) as the program progresses.

The last graphic block drawn is (122,3), in line 40. Line 45 now sets X equal to 123, so the following line 46 is now operational, and the program is sent to line 170.

Line 170 looks peculiar. It is an "endless loop" since it keeps calling itself as the next line. This is a simple technique for stopping the program without a READY or cursor scrolling the screen or blanking out one or two lines of the display. The penalty, however, is that you lose keyboard control, except for BREAK. Press BREAK to regain keyboard control.

The entire "drawing" on the screen takes less than 30 seconds—a lot faster than trying to explain it!

Modifications: You can use different values in line 155 for the constants—try A=C+4 and B=D+4—to draw a lopsided spiral that will continue running in Level I (until X=123) but that will come up with an FC (function call) error in Level II when the graphic line hits the bottom screen limit after six and one-half "turns" of the spiral. Level I graphics have "wraparound," that is, lines going off the screen at one limit of the screen automatically start at the opposite limit. In other words, a graphic line going down past the bottom appears at the top and continues downward; a line going off the right side of the screen next appears on the left and continues to the right. Level II graphics do not have this wraparound feature; a graphic command that would light the screen beyond its limits results in program interruption with an FC error. The program also "bombs" in either Level I or Level II if the X or Y values of a SET statement are negative numbers.

Variables:

X—horizontal locator
Y—vertical locator
A—right-going block counter
B—down-going block counter
Z—loop counter
C—left-going block counter
D—upward-going block counter

PROGRAM 18
CUSTOM SCREEN DISPLAYS
(4K, Level I or II)

You'll learn: A relatively easy way to display large letters or figures on the TRS-80 screen using DATA and SET statements; also a way to display black letters or figures on a white background using RESET.

Description: The words "RADIO SHACK TRS-80 COMPUTER" are displayed in three lines of 1-inch high letters on a black screen in a dot-dash fashion. Then the screen goes blank, is "painted" white, and sections are "punched out" to display the same letters in black on a white background. This sequence keeps repeating.

Explanation: Looking at the listing for this program is enough to confuse even a veteran programmer. All those DATA statements

seem impossible to digest or interpret. However, even an elephant can be eaten one bite at a time, and we can go through this program one line at a time to see how really simple the idea is. Once you see how it works, you will be limited only by your imagination.

The program was originally designed for Level I, which has limited graphic statements, such as SET and RESET. CHR$(X) is available in Level II for much faster graphics—especially when used with POKE and PEEK—but this program will use only SET and RESET and will run fine in Level II. It's fascinating to watch the letters' being formed, erased, reformed on a contrasting background, and then run again. They make a very nice, unattended attention-getting window, counter, or show-booth display.

Before we go into a line-by-line explanation, look at the screen layout in Fig. 18–1. The task is to program the TRS-80 to make this display. The letters are shown as black on a white background. Since the blank screen is black and since SET statements *light* specified segments of the screen, let's first assume that the black block letters are actually *white* and program that approach first.

Each letter is actually composed of many individual lighted segments. The first letter in RADIO, the "R," is made up of 39 individual lighted segments, each of which can be lighted with a SET (X,Y) statement. The extreme upper left corner is (6,9), that is, the

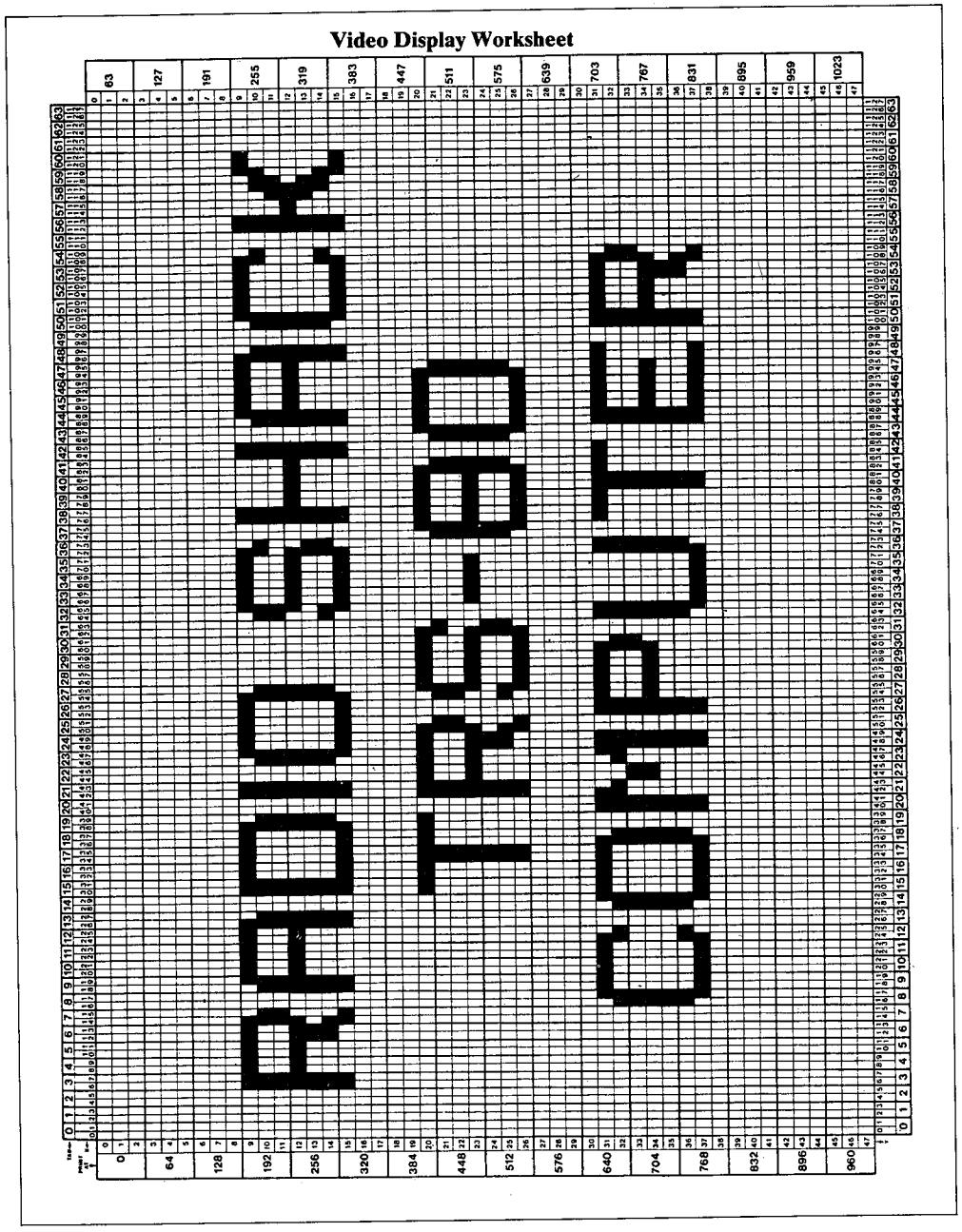


Fig. 18-1. Screen lay-out

horizontal position, X, is 6 and the vertical position, Y, is 9. You can therefore program this display by using a SET statement to light every segment needed to form each letter.

A shortcut would be to use DATA statements to hold X and Y coordinates and some kind of looping to feed these coordinates into SET statements. In this program, two different types of loops are used. The program could have been formulated using either one alone, but using both gives you a choice for future programs of your own.

Lines 4–7 introduce the program. Lines 8–17Ø contain DATA for the first programming approach, and lines 2ØØ–44Ø contain DATA for the second approach. Line 5ØØ starts the main program.

Line 5Ø4 sets variable Z equal to zero. You'll see later that assigning this value, in effect, tells the computer to display white letters on a black background. Line 5Ø5 clears the screen. The RESTORE is not needed the first time through the program; however, as the program progresses, the DATA is accessed, and a DATA pointer moves on to the next DATA item. The RESTORE statement moves the DATA pointer back to the very first DATA item for the next rerun.

Line 5Ø6 looks at the value of Z, and if it is zero, as it is in this case, the program jumps to line 52Ø.

Line 52Ø starts a FOR-NEXT loop, using J to count from 1 to 89. Why 89? Because we will have to go exactly 89 times through this loop to do what we want, that is, to light all segments of the display on screen lines 9, 12, 15, 20, 23, 26, 31, 34, and 37 of the layout in Fig. 18–1. This will light the top, bottom, and center lines of all screen letters, omitting the other lines for the effect. As this takes place, it looks like a secret dot-dash code of some sort—very mysterious. Later on, the program fills in the missing portion of all the letters.

Line 53Ø tells the computer to go to the *first* DATA line—*wherever* it is in the program—and read the first three values as variables A, B, and Y, respectively. Looking at line 1Ø, we see that A will be 6, B will be 14, and Y will be 9. Now line 54Ø starts *another* loop—a loop-within-a-loop, also called a "nested loop"—with X counting from A to B. Since A is 6 and B is 14, as just established by the READ statement in line 53Ø, X will initially equal 6 and progress in steps of 1 (since no other STEP is specified) to a value of 14.

Line 545 is ignored, since Z is still zero, and line 55Ø lights segment (6,9) since X is 6 and Y is 9. Line 56Ø has two loop completion statements, NEXT X and NEXT J. However, until X

LEVEL I/II LISTING

```
4 REM * TRS-80 LEVEL I OR LEVEL II 4K *
5  REM * RADIO SHACK TRS-80 COMPUTER SCREEN DISPLAY *
6  REM * COPYRIGHT FRED BLECHMAN 1978 *
7 REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
8  REM * LOOP VALUES *
10  DATA6,14,9,19,26,9,30,38,9,42,43,9,47,54,9,65,72,9
20  DATA76,77,9,84,85,9,89,96,9,101,108,9,112,113,9,119,121,9
30  DATA6,14,12,18,27,12,30,31,12,38,39,12,42,43,12,46,47,12
40  DATA54,55,12,65,72,12,76,85,12,88,97,12,100,101,12,112,117,12
50  DATA6,7,15,14,15,15,18,19,15,26,27,15,30,38,15,42,43,15
60  DATA47,54,15,65,72,15,76,77,15,84,85,15,88,89,15
70  DATA96,97,15,101,108,15,112,113,15,119,121,15
80  DATA30,39,20,42,50,20,55,62,20,75,82,20,87,94,20
90  DATA34,35,23,42,50,23,55,62,23,68,71,23,74,83,23,86,87,23
100  DATA94,95,23
110  DATA34,35,26,42,43,26,50,51,26,55,62,26,75,82,26,87,94,26
120  DATA17,24,31,29,36,31,40,42,31,47,49,31,52,60,31
130  DATA64,65,31,72,73,31,76,85,31,88,97,31,100,108,31
140  DATA16,17,34,28,29,34,36,37,34,40,41,34,44,45,34,48,49,34
150  DATA52,60,34,64,65,34,72,73,34,80,81,34,88,95,34,100,108,34
160  DATA17,24,37,29,36,37,40,41,37,48,49,37,52,53,37
170  DATA65,72,37,80,81,37,88,97,37,100,101,37,108,109,37
200  REM * SINGLE SETS *
210  DATA39,6,7,14,15,18,19,26,27,30,31,38,39,42,43,46,47,54,55
220  DATA64,65,72,73,76,77,84,85,88,89,96,97,100,101
230  DATA108,109,112,113,117,118,119
240  DATA35,6,7,14,15,18,19,26,27,30,31,38,39,42,43,46,47,54,55
250  DATA64,65,76,77,84,85,88,89,96,97,100,101,112,113,116,117,118
255  DATA1,6
260  DATA35,6,7,12,13,18,19,26,27,30,31,38,39,42,43,46,47,54,55
270  DATA72,73,76,77,84,85,88,89,96,97,100,101,112,113,116,117,118
280  DATA40,6,7,12,13,14,18,19,26,27,30,31,38,39,42,43,46,47,54,55
290  DATA64,65,72,73,76,77,84,85,88,89,96,97,100,101,108,109
300  DATA112,113,117,118,119
310  DATA18,34,35,42,43,50,51,54,55,62,63,74,75,82,83,86,87,94,95
320  DATA16,34,35,42,43,50,51,54,55,74,75,82,83,86,87,94,95
330  DATA1,34
340  DATA16,34,35,42,43,48,49,62,63,74,75,82,83,86,87,94,95
350  DATA19,34,35,42,43,48,49,50,54,55,62,63,74,75,82,83,86,87,94,95
360  DATA32,16,17,24,25,28,29,36,37,40,41,42,43,46,47,48,49
370  DATA52,53,60,61,64,65,72,73,80,81,88,89,100,101,108,109
380  DATA28,16,17,28,29,36,37,40,41,44,45,48,49,52,53,60,61,64,65
390  DATA72,73,80,81,88,89,100,101,108,109
400  DATA1,16
410  DATA24,16,17,28,29,36,37,40,41,48,49,52,53,64,65,72,73
420  DATA80,81,88,89,100,101,106,107
430  DATA27,16,17,24,25,28,29,36,37,40,41,48,49,52,53
440  DATA64,65,72,73,80,81,88,89,100,101,106,107,108
500  REM * MAIN PROGRAM *
504  Z=0
505  CLS:RESTORE
506  IF Z=0 GOTO520
510  CLS:RESTORE
511  FOR Y=0 TO 47
512  FOR X=0 TO 127
513  SET(X,Y)
514  NEXTX:NEXTY
520  FOR J=1 TO 89
530  READ A,B,Y
540  FORX=A TO B
545  IF Z=1 THEN RESET(X,Y):GOTO560
550  SET(X,Y)
560  NEXTX:NEXTJ
600  GOSUB680
```

```
610  GOSUB690
620  GOSUB700
630 GOTO905
680  FOR Y=10 TO 14
685  GOTO705
690  FOR Y=21 TO 25
695  GOTO705
700  FOR Y=32 TO 36
705  READ A
710  FORX=1 TO A
715  IF Z=1 THEN READ B:RESET(B,Y):GOTO730
720  READ B:SET(B,Y)
730  NEXTX:NEXTY
740  RETURN
905  IF Z=1 THEN Z=0:GOTO910
906  IF Z=0 THEN Z=1
910  FOR X=1 TO 1000:NEXTX
920  GOTO505
```

advances to the limiting value (14, in this example), the program is sent back to line 540 for another X. This time X is 7, but Y has not changed; it is still 9. Line 550 therefore lights segment (7,9). Do you see what is happening? Line 550 next lights segment (8,9), then (10,9), (11,9), (12,9), (13,9), and (14,9). This completes the top line of the letter R in RADIO. With X now equal to 14, line 560 executes NEXT J, and the program goes back to line 520. Here J is advanced to 2. Line 530 reads the *next three data* values (still on line 10). These are 19, 26, and 9. Thus A becomes 19, B is 26, and Y is 9. Now the X-loop in lines 540–560 lights segments 19 through 26 on screen line 9, the top line of the letter A in RADIO.

If you analyze the DATA statements, you'll see that they provide the computer with SET values to light the top, center, and bottom lines of all display letters—just as we planned! This process takes 89 trips through program lines 520–560 and uses up all the DATA in lines 10–170. Finally, J equals 89, and the program falls through line 560 to 600, where a slightly different loop technique is used.

Line 600 sends the program to the subroutine starting at line 680. Here Y is a loop counter, with the value limits representing vertical screen lines—in this case, lines 10 to 14. Line 685 jumps the program to line 705, where the next DATA value (the first one in line 210, since all prior values have been used) is read. This value, 39, now becomes the value of A. Line 710 starts a loop with X counting from 1 to 39. Line 715 is ignored since Z is still zero. Line 720 now READs the next DATA value, 6, and sets B equal to this; then it lights segment (6,10). (B is 6 and Y is 10.) Line 730 sends the program back to line 710 (until X is equal to 39), and another segment, (7,10), is lighted. This process continues, using the DATA

in program lines 21Ø, 22Ø, and 23Ø, until all the necessary 39 segments on line 1Ø are lighted. Then line 73Ø executes NEXT Y, and the program goes back to line 68Ø, where Y is advanced to 11. DATA from lines 24Ø and 25Ø is used in 35 X-loops to light the necessary segments of screen line 11.

Line 255 needs some explanation. It comes into play as Y in line 68Ø is advanced to 12. Since the necessary segments of screen line 12 have already been lighted by the earlier part of the program (DATA lines 3Ø and 4Ø), line 255 lights the already lighted segment (6,12) to fulfill the Y = 12 loop, and Y moves on to 13.

When the Y = 14 loop is completed, line 73Ø is satisfied and line 74Ø RETURNs the program to line 61Ø. Now subroutine 69Ø–74Ø lights the required segments on screen lines 21–25; then the subroutine in lines 7ØØ–74Ø lights segments of screen lines 32–36 and sends the program to line 63Ø.

Line 63Ø sends the program to line 9Ø5. Since Z is still zero, the program ignores this line, and line 9Ø6 changes the value of Z to 1. From here on, until line 9Ø5 is reached again, Z is equal to 1.

Line 91Ø is a time-delay loop, counting to itself for about 2 seconds to allow you to see the display. Line 92Ø then shoots the program back to line 5Ø5, where the screen is cleared and all DATA RESTOREd.

Since line 5Ø6 now sees that Z is 1, it ignores line 5Ø6 and goes on to line 51Ø. The latter line is *not needed* since it merely repeats line 5Ø5 (it appears to have been left over from program development).

Lines 511–514 paint the screen white by sequentially lighting all 128 segments on all 48 screen lines (a total of 6144 segments!), one at a time. Slow, but easy to program. Level II can do this several faster ways.

Once the screen is white, the program runs as before *except* that program lines 545 and 715 are now active (since Z is equal to 1) and RESET is used instead of SET. RESET turns *off* a lighted segment with the effect of punching a black hole in the white screen. The loops work exactly the same as before.

Since line 9Ø5 returns Z to a value of zero, the next run through the program is the same as the first. Lines 9Ø5 and 9Ø6 act like a flip-flop to alternate the display background each time through.

The program keeps running until you press BREAK.

Modifications: Obviously, by changing the DATA and loop values, you can create whatever you want on the screen. Try making

up your own program to show your name in giant letters. First draw it out on a copy of the TRS-80 Graphics Worksheet (Radio Shack Catalog #26-2105) or the Appendix A worksheet. Then program the loops and DATA lines to light the required segments.

Variables:

Z—black or white background control
Y—vertical screen line
X—horizontal screen location; loop counter
J—loop counter
A—start of segment location; X-loop limit
B—end of segment location

PROGRAM 19
CHANGING BILLBOARD PROGRAM
(4K, Level I or II)

You'll learn: Control of screen printing—location, duration, and erasure; custom programming; subroutine convenience

Description: A changing one-line message is enclosed in a graphic rectangle, with a fixed 11-line message above and below the rectangle. You can easily program both the fixed and changing message lines for your own eye-catching display.

Explanation: With this program your TRS-80 will "advertise" anything you want with an attention-getting display. You can custom program the text to promote any product or service or to tell passers-by when the next "live" demonstration will take place, or when the store will be open, or the "specials" for this week, or whatever.

The sample program contains instructions for its use in the printed statements. It can be run in either Level I or Level II, provided you use the correct instruction for the printing location—PRINTAT for Level I, PRINT@ for Level II. (Be sure *not* to use the SHIFT key when you press the @ key!)

Lines 45–55 clear the screen and introduce the program with REM statements; these don't appear on the screen except when you LIST. Lines 6Ø–71 print programming instructions on the screen and wait for you to press ENTER when you want to see the sample display.

Lines 75–85 print the "fixed" text, that is, the text that doesn't change during this display. The three PRINTs at the beginning of line 78 leave three blank lines between the text of line 77 and the text that follows in line 78. Why? Because this is the space we're reserving for the rectangle.

Lines 11Ø–16Ø draw the rectangle—using FOR-NEXT loops and SET statements—in the empty space, between Y locations of 13 and 19 and X locations of 1 and 11Ø (see Appendix A).

Print location 322 is near the left end of the rectangle, halfway between the top and bottom of the rectangle; this is where

```
45  CLS
50  REM * COPYRIGHT FRED BLECHMAN 1978 *
51 REM * 7217 BERNADINE AVE., CANOGA PARK, CA  91307 *
55 REM * TRS-80 LEVEL I OR LEVEL II  4K *
60  PRINT:PRINT"          CHANGING BILLBOARD PROGRAM"
65  PRINT:PRINT"     YOU MUST DO SOME OF YOUR OWN PROGRAMMING FOR "
66  PRINT"THIS BILLBOARD TO BE EFFECTIVE FOR YOUR OWN USE.  LINES"
67  PRINT"75 TO 85 ARE THE FIXED COPY, WHILE LINES 170 TO 490"
68  PRINT"ARE THE CHANGING COPY.  JUST TYPE IN THE TEXT YOU WANT"
69  PRINT"FOR THOSE LINES AND YOU'VE GOT A CUSTOM PROGRAM.  THE"
70  PRINT"FOLLOWING IS JUST A SAMPLE OF THE RESULTS...."
71  PRINT:INPUT"WHEN READY TO SEE SAMPLE DISPLAY, HIT ENTER ";A$
75  CLS:PRINT:PRINT"     THIS IS A SAMPLE OF HOW YOU CAN PLACE"
76  PRINT"FIXED COPY ON THE SCREEN TO SAY WHATEVER YOU WISH.  THIS"
77  PRINT"COPY IS DETERMINED BY WHAT YOU TYPE IN LINES 75 TO 85."
78  PRINT:PRINT:PRINT:PRINT"YOU MUST REMEMBER TO DUPLICATE THE PUNCTUATION"
79  PRINT"YOU FIND IN THE EXISTING LISTING FOR THESE 'PROGRAMMABLE'"
80  PRINT"LINES.  THE SPACING IS IMPORTANT, SINCE YOU MUST LEAVE"
81  PRINT"ROOM FOR THE RECTANGLE, IN WHICH THE CHANGING MESSAGE"
82  PRINT"GOES.....FOR THE CHANGING MESSAGE, INSERT ANY TEXT, NO"
83  PRINT"LONGER THAN 50 CHARACTERS AND SPACES, IN LINES 170 TO 490"
84  PRINT"...AND DON'T FORGET THE SEMI-COLONS TO END EACH MESSAGE LINE!"
85  PRINT"                    HAVE FUN!"
110  FOR X=1 TO 110
120  SET(X,13):SET(X,19)
130  NEXTX
140  FOR Y=13 TO 19
150  SET(1,Y):SET(110,Y)
160  NEXTY
170  PRINT@322,"THIS IS A SAMPLE OF THE TEXT YOU CAN PUT IN THE";
175  GOSUB500
180  PRINT@322,"BOX.  ACTUALLY, IT'S JUST A MATTER OF TYPING";
185  GOSUB500
190  PRINT@322,"AS MANY LINES AS YOU WANT - UNTIL THE MEMORY";
195  GOSUB 500
200  PRINT@322,"IS FULL! YOU TYPE THE CHANGING STATEMENTS IN";
205  GOSUB 500
210  PRNT@322,"LINES 170, 180, 190, 200, ETC., AND TYPE THE";
215  GOSUB500
220  PRINT@322,"GOSUB500 INSTRUCTION IN BETWEEN EACH.  YOU CAN";
225  GOSUB500
230  PRINT@322,"GO ALL THE WAY TO LINE 490 BEFORE YOU RUN INTO";
235  GOSUB500
240  PRINT@322,"THE SUBROUTINE AT LINE 500.  DON'T FORGET THE";
245  GOSUB500
250  PRINT@322,"SEMI-COLON AT THE END OF EACH BILLBOARD LINE!";
255  GOSUB500
260  PRINT@322,"   WHEN YOU GET EVERYTHING THE WAY YOU WANT IT";
265  GOSUB500
270  PRINT@322,"THEN PUT IT ON A CASSETTE TAPE FOR LATER USE....";
275  GOSUB500
490  PRINT@322,".....NOW WE'LL REPEAT THE ENTIRE SEQUENCE.....";
495  GOSUB500
496  GOTO170
500  FOR X=1 TO 1000:NEXTX
505 PRINT@322,"                                                    ";
506  FOR X=1 TO 250:NEXTX
510  RETURN
```

the *changing* text starts. Line 170 prints the first line of this changing text. The semicolon at the end of this line is extremely important. Without it, part of the right border of the rectangle would be

PROGRAM 19 RUN

```
          CHANGING BILLBOARD PROGRAM

     YOU MUST DO SOME OF YOUR OWN PROGRAMMING FOR
THIS BILLBOARD TO BE EFFECTIVE FOR YOUR OWN USE. LINES
75 TO 85 ARE THE FIXED COPY, WHILE LINES 170 TO 490
ARE THE CHANGING COPY. JUST TYPE IN THE TEXT YOU WANT
FOR THOSE LINES AND YOU'VE GOT A CUSTOM PROGRAM. THE
FOLLOWING IS JUST A SAMPLE OF THE RESULTS....

WHEN READY TO SEE SAMPLE DISPLAY, HIT ENTER ?


     THIS IS A SAMPLE OF HOW YOU CAN PLACE
FIXED COPY ON THE SCREEN TO SAY WHATEVER YOU WISH. THIS
COPY IS DETERMINED BY WHAT YOU TYPE IN LINES 75 TO 85.
###########################################################
# THIS IS A SAMPLE OF THE TEXT YOU CAN PUT IN THE       #
###########################################################
YOU MUST REMEMBER TO DUPLICATE THE PUNCTUATION
YOU FIND IN THE EXISTING LISTING FOR THESE 'PROGRAMMABLE'
LINES. THE SPACING IS IMPORTANT, SINCE YOU MUST LEAVE
ROOM FOR THE RECTANGLE, IN WHICH THE CHANGING MESSAGE
GOES.....FOR THE CHANGING MESSAGE, INSERT ANY TEXT, NO
LONGER THAN 50 CHARACTERS AND SPACES, IN LINES 170 TO 490
...AND DON'T FORGET THE SEMI-COLONS TO END EACH MESSAGE LINE!
                    HAVE FUN!
```

blanked out in Level I; in Level II, the bottom of the rectangle (the next printing line) would be erased. Line 175 tells the program to jump to a subroutine starting at line 500.

Line 500 is a simple time-delay loop—about 2½ seconds in Level II, shorter in Level I. Line 505 then *erases* the printing in the rectangle by printing blanks. Notice that the ending quotation mark encloses exactly 50 blank spaces and is followed by a semicolon to avoid destroying part of the rectangle.

Line 506 is another short time-delay loop. It leaves the inside of the rectangle blank for about a half second to provide some anticipation of the next line of text to be printed in the rectangle. You can leave this program line out, if you prefer.

Line 510 sends the program back to line 175 for the next instruction. Since there's nothing after GOSUB500, the program moves on to line 180 and prints another line of text in the blanked-out rectangle. This procedure keeps repeating until line 496 sends the program back to line 170, and the whole changing-message portion of the text is repeated.

The program continues repeating until you press BREAK (or push the RESET button, or shut off the computer, or unplug the computer, or there's a power outage, or. . . .)

CSAVE the finished program on tape for future use.

Modifications: Obviously, you'll want to change all the text for your own use. When you program the changing lines of text, be sure that each line does not exceed a total of 50 characters and spaces. If you use more than 50 characters and spaces, the erase instruction, program line 505, will only erase the first 50 spaces—thus leaving the remainder on the screen—and your text also might bust through the right border of the rectangle.

You might wish to have several completely different screen displays, including new fixed text each time. Move lines 110–160 so that they start at, say, line 600 and make them a subroutine. "DELETE 60–71" will remove the instructions, giving you more available memory. Now, with a 4K memory you should be able to get two complete displays with 11 fixed lines and 12 changing lines; with a 16K memory you'll probably be able to program seven complete displays this size. By having a GOTO instruction at the *end* of the *last* display that sends the program back to the *start* of the *first* display, the program will keep repeating all displays in sequence.

Also, each display can have more changing-text lines. You've got all the line numbers from 170 to 496 to use. The GOSUB500 instruction does not need to be on a separate line; it can follow a colon at the end of each changing-text line.

Variables:

A$—pause control
X—graphic horizontal position; delay-loop counter
Y—graphic vertical position

PROGRAM 20
THE MAGIC SQUARE
(4K, Level I or II)

You'll learn: Computation from a formula and screen layout of results

Description: This program generates a 16-number square whose numbers add in any direction—vertically, horizontally, or diagonally—to total a chosen number. It works with positive or negative numbers.

Explanation: *Creative Computing* (October 1979, p. 115) describes a large square containing 16 smaller, numbered squares whose numerical values add up to the same total in every direction. A "secret" formula calculates the number for each small square that will produce the chosen total, which we'll call X.

Our program converts this formula into TRS-80 BASIC and instructs the computer to display the results in an almost-square pattern on the screen. To understand the programming, you must first see how the formula is used. The large square in Fig. 20-1 is divided into 16 smaller squares, each labeled with a

SQUARE VALUES

A= D+7	B= D+10	C= D+13+R	D= D+0
E= D+12+R	F= D+1	G= D+6	H= D+11
I= D+2	J= D+15+R	K= D+8	L= D+5
M= D+9	N= D+4	O= D+3	P= D+14+R

BASIC FORMULA:

$$\frac{X-30}{4} = D + R$$

WHERE:
X = CHOSEN NUMBER
D = DIVIDEND (INTEGER)
R = REMAINDER (0, 1, 2, OR 3)

Fig. 20-1. Magic square

letter from A to P. The letter for each small square represents the value of the number desired for that square.

Each square contains a simple addition formula involving D and sometimes R. These formulas are derived from the equation:

$$\frac{X - 30}{4} = D + R$$

where:

X = chosen number
D = dividend [largest whole number that can be divided into (X − 30)]
R = remainder after division (Ø, 1, 2, or 3)

To keep all numbers positive, X should be above 40. However, our clever TRS-80 ingeniously performs negative-integer rounding *downward,* to the next *lower* number, so that any numbers, positive or negative, will work in this program. For simplicity, our discussion assumes positive numbers when dealing with the INT (integer) function.

Now all we need to do is somehow enter these formulas into a program and format the results on the screen. This program will run "as is" in either Level I or Level II, since no sophisticated calculations or formatting are involved.

Lines 1ØØ–13Ø are REM statements that give the program source. Line 14Ø clears the screen and prints two blank lines so that the next lines, 15Ø and 16Ø, print several lines below the top of the screen. Line 17Ø skips two more screen lines and asks you to choose any number. The input statement waits for your entry. When you type any reasonable whole number (positive or negative) and press ENTER, the computer will store this number as variable X. You may have noticed the word "reasonable" in the last sentence. What's reasonable? I've used numbers as large as 10 million, both positive and negative, and the program continues to work in Level II, although the numbers must be expressed in exponential notation when they get extremely large. The author must confess that he doesn't know the actual limits.

Line 18Ø subtracts 30 from X and divides the result by 4; the integer function then cuts off any remainder so that D ends up as a whole number. We now have the dividend, D, but how do we get a remainder? That is provided by line 19Ø.

```
100 REM * MAGIC SQUARE - DERIVED FROM CREATIVE COMPUTING *
110 REM * OCTOBER 1979 ISSUE, PAGE 115 *
120 REM * PROGRAMMED BY FRED BLECHMAN!*
130 REM * COPYRIGHT FRED BLECHMAN 1979 *
140 CLS:PRINT:PRINT
150 PRINT"                .....THE MAGIC SQUARE....."
160 PRINT"      .....TOTALS THE GIVEN NUMBER IN ANY DIRECTION.
    ..."
170 PRINT:PRINT:INPUT"ENTER ANY NUMBER..!."!X
180 D=INT((X-30)/4)
190 R=(X-30)/4:R=R-INT(R)
200 IFR>.74THENR=3:GOTO 240
210 IFR>.49THENR=2:GOTO 240
220 IFR>.24THENR=1:GOTO 240
230 R=0
240 A=D+7:B=D+10:C=D+13+R:D=D+0
250 E=D+12+R:F=D+1:G=D+6:H=D+11
260 I=D+2:J=D+15+R:K=D+8:L=D+5
270 M=D+9:N=D+4:O=D+3:P=D+14+R
280 CLS:PRINT:PRINT"THE MAGIC NUMBER IS";X;"!":PRINT
290 PRINTA,B,C,D
300 PRINT:PRINT
310 PRINTE,F,G,H
320 PRINT:PRINT
330 PRINTI,J,K,L
340 PRINT:PRINT
350 PRINTM,N,O,P
360 PRINT
370 INPUT"ANOTHER MAGIC NUMBER? (YES=1, NO=0)";Z
380 IF Z=1 GOTO100
390 END
```

First, line 190 subtracts 30 from X and then divides the result by 4 just as line 180 did, but *without* the INT. This means that the value of R is *not* a whole number, but the whole-number portion is not important, only the remainder. The next statement on line 190 therefore subtracts INT(R)—the whole-number portion—from R, leaving only the decimal remainder.

The decimal remainder must mathematically be zero, .25, .50, or .75, since we started with a whole number (X − 30) and divided by 4. However, the TRS-80 *sometimes* comes up with numbers like .250001 or .740009. Lines 200–220 inspect R, the remainder, to see if it's greater than .74, .49 or .24. If greater than .74, then it must be .75; R is thus set to the value of 3 and the program skips to line 240. Similarly, the value of R may be set to 2 by line 210 or 1 by line 220. If none of the lines 200, 210, or 220 are true, then line 230 sets the value of R to zero.

The program now establishes the value for each small square simply by setting the letter for that square equal to the addition formula shown for it in Fig. 20-1. For example, the value in Square A always is equal to the dividend, D, plus 7. Square B has a value of D plus 10. Square C has a value of D plus 13 plus R. Square D

PROGRAM 20 RUN

```
              .......THE MAGIC SQUARE.....
       .......TOTALS THE GIVEN NUMBER IN ANY DIRECTION....

ENTER ANY NUMBER....? 1345

THE MAGIC NUMBER IS 1345 !

 335            338            344            328

 343            329            334            339

 330            346            336            333

 337            332            331            345

ANOTHER MAGIC NUMBER? (YES=1, NO=0)? 1

              .......THE MAGIC SQUARE.....
       .......TOTALS THE GIVEN NUMBER IN ANY DIRECTION....

ENTER ANY NUMBER....? -66

THE MAGIC NUMBER IS-66 !

-17            -14            -11            -24

-12            -23            -18            -13

-22            -9             -16            -19

-15            -20            -21            -10

ANOTHER MAGIC NUMBER? (YES=1, NO=0)? 0
```

always has just the value of the dividend, with nothing added. Lines 240–270 calculate these values for all sixteen squares and hold the results in memory as variables A through P.

Line 280 clears the screen, skips a line, and prints the chosen number, X; then it skips another line. Lines 290–350 "zone-print" each variable (remember that the comma directs the next variable value to print at the beginning of the next zone) with two horizontal blank lines between each printed line, forming a large square (actually, a squashed-down-a-bit square!).

Line 360 skips a line before line 370 asks if you want to enter another number. If you do, line 380 recognizes the "1" you entered for variable Z and sends the program back to line 100. If not, line 390 ends the program.

Modifications: You could use graphic lines generated by SET statements to outline all the squares and then use PRINT@ (or PRINTAT in Level I) to print the results in each square. However, if you do this, arrange the program to draw the lines *only once* and use PRINT@ with blank spaces to "erase" previous numbers on reruns to save "drawing" time.

Some advanced high-speed graphic techniques can be used to reduce graphics time to an acceptable value for reruns. Program 6 offers one approach; there are faster ones, but these are beyond the scope of this book.

Variables:

X—chosen number
D—dividend
R—remainder
A-P—individual square values
Z—rerun control

PROGRAM 21
PIANO KEYBOARD FREQUENCIES
(4K, Level II Only)

You'll learn: Fractional-exponent calculation; printing contents of screen on printer (optional)

Description: This program calculates and displays eight octaves of the musical scale on the screen. The display may also be output to a printer.

Explanation: There are actually many different musical scales in use around the world, but the most dominant one in the United States is the "American Equal Temperament Musical Scale," in which the standard pitch of A is 440 Hz (cycles per second) and the frequency of each next higher note (including the black keys) is exactly 2 raised to the 1/12 power greater.

This apportionment means that the frequency *doubles* every 12 notes, an octave. Look at the piano keyboard layout in Fig. 21-1. Although there are 88 keys on a piano, only two octaves are shown here—a total of 24 keys starting with A (220 Hz). Notice that each black note is shown as a "sharp" (#). (We will not refer to "flats" since that would just confuse matters; the frequency of each black key remains the same regardless of a

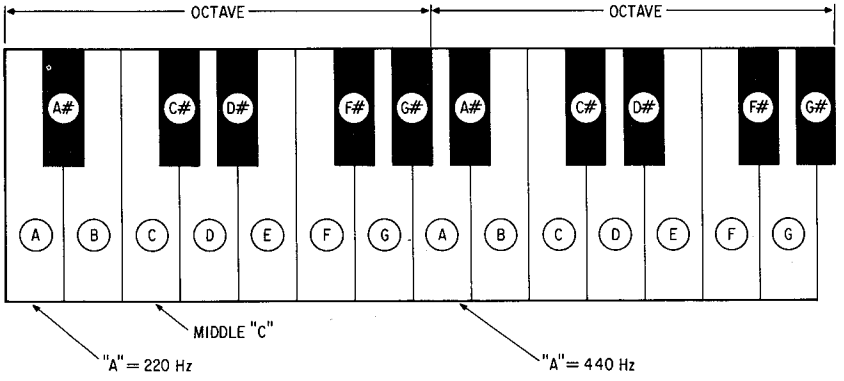


Fig. 21-1. Portion of piano keyboard

sharp or flat reference.) The designation of each twelfth note is a repeat, doubling in frequency as you go upward in pitch (to the right) and halving in frequency as you go down the keyboard (to the left). Therefore, the lowest key on the piano, A (four octaves below the A of 440 Hz) has a frequency of 27.5 Hz (440/2 = 220/2 = 110/2 = 55/2 = 27.5). We'll use this as the starting point in our program.

The listing for this program operates only in Level II BASIC because of the fractional exponent calculation. (If Level I were used, extensive subroutines would be needed.) Lines 10–80 clear the screen and introduce the program. Lines 100–107 clear the screen and print the tabulation title and octave headings.

Line 110 sets the value of A equal to 27.5, Z equal to 1, E equal to 0, and P equal to 192. A is the variable that represents the frequency of the leftmost piano key. Variable Z, as we will see shortly, is the A-array counter. Variable E is the value of the numerator of the E/12 exponent. Variable P is the print-location number for display purposes.

Line 115 starts a FOR-NEXT loop in which R, the note counter, will loop 12 times (an octave). Line 120 starts a FOR-NEXT loop, with X counting the eight octaves. Line 130 establishes a value of A-array location Z equal to A times 2 to the E/12 power. (The up-arrow means "raise to the following power.") With the values from line 110, A(1) equals 27.5 times 2 to the 0/12 power, or 27.5 (since any number to the zero power equals 1). Now line 135 uses the integer-rounding technique explained in Program 15 (Modifications) to make A(Z) a two-place decimal. This process stores 27.5 in A(1).

Line 140 now increments Z by 1 and multiplies A by 2; Z is therefore 2 and A is 55. Line 150 sends the program back to line 120, where X becomes 2. Lines 130 and 135 now store 55 in A-array A(2). This process continues, with A(3) equal to 110, A(4) equal to 220, and so on, up to A(8), which is equal to 3520. Notice that in all these cases, since E is equal to zero, 2 to the E/12 power is equal to 1 and has no effect on the calculation in line 130.

When line 150 makes X equal to 9, exceeding the top value set in line 120, the program continues to line 165, which READs the first DATA entry, A in line 300, and puts this in memory as A$. Remember that a READ statement looks for the first DATA line it can find anywhere in the program.

Line 170 prints a single line, starting at print position 192 (set in line 110). First it prints A$, which is A in this case. The

```
10 REM * COPYRIGHT 1979 FRED BLECHMAN *
11 REM * 7217 BERNADINE AV., CANOGA PARK, CA  91307 *
15 CLS
20 PRINT@256,"     THIS PROGRAM CALCULATES AND DISPLAYS THE FREQUENCIES"
30 PRINT"OF THE EQUAL TEMPERAMENT MUSICAL SCALE, USING THE AMERICAN
40 PRINT"STANDARD PITCH OF A=440 HERTZ. THE FREQUENCY OF EACH NOTE"
50 PRINT"IS EXACTLY 2 RAISED TO THE 1/12 POWER TIMES THE PRECEDING NOTE."
60 PRINT:PRINT"     IF YOU ARE NOT USING A PRINTER, BE SURE TO DELETE LINES"
70 PRINT"200 AND 210 BEFORE RUNNING!!"
80 PRINT:INPUT"     WHEN READY TO RUN, PRESS ENTER....";Z$
100 CLS
105 PRINT"     PIANO KEYBOARD FREQUENCIES FOR EIGHT OCTAVES"
106 PRINT"                (EQUAL TEMPERAMENT SCALE)
107 PRINT"      1     2     3     4     5     6     7     8"
110 A=27.5:Z=1:E=0:P=192
115 FOR R=1 TO 12
120 FOR X=1 TO 8
130 A(Z)=A*2↑(E/12)
135 A(Z)=INT((A(Z)*100)+.5)/100
140 Z=Z+1:A=A*2
150 NEXT X
165 READA$
170 PRINT@P,A$:PRINT@P+3,A(1):PRINT@P+9,A(2):PRINT@P+16,A(3):PRINT@P+23,A(4)
172 PRINT@P+30,A(5):PRINT@P+37,A(6):PRINT@P+45,A(7):PRINT@P+53,A(8)
176 P=P+64
180 E=E+1:A=27.5:Z=1
190 NEXT R
200 V=15360:LPRINTCHR$(29):FORC=0TO15:FORR=0TO63
210 LPRINT CHR$(PEEK(V+64*C+R));:NEXT:LPRINT" ":NEXT:END
300 DATA A ,A♯,B ,C ,C♯,D ,D♯,E ,F ,F♯,G ,G♯
```

remainder of program line 170 prints each of the values held in A(1) through A(4), formatted by incremented PRINT@ statements. Line 172 continues the process with values A(5) through A(8). This is yet another means of formatting screen printing; TAB statements could also be used. The increments (P+3, P+9, P+16, etc.) are *not* equal, since the number of digits to be printed in each column varies.

Line 176 increments P, the print locator, by 64, to prepare for printing on the next screen line the next time through the R-loop. Similarly, line 180 increments E by 1, resets A to the starting value of 27.5, and resets the octave counter, Z, to 1. Line 190 returns the program to line 115, and R is now equal to 2.

This time, since line 130 sees E equal to 1, 2 is raised to the 1/12 power. Therefore, A(1) is 27.5 times 2 to the 1/12 power, or 27.5 times 1.05946, which equals 29.135232. This figure is rounded up to 29.14 by line 135 and stored in memory as A(1). Similarly, if you follow through the X-loop (lines 120–150), A(2) equals 55 times 1.05946, or 58.270465. This is rounded down to 58.27 and stored as A(2). Similarly, A(3) is 116.54, A(4) is 233.08, and so on.

PROGRAM 21 RUN

```
    THIS PROGRAM CALCULATES AND DISPLAYS THE FREQUENCIES
OF THE EQUAL TEMPERAMENT MUSICAL SCALE, USING THE AMERICAN
STANDARD PITCH OF A=440 HERTZ. THE FREQUENCY OF EACH NOTE
IS EXACTLY 2 RAISED TO THE 1/12 POWER TIMES THE PRECEDING NOTE.

    IF YOU ARE NOT USING A PRINTER, BE SURE TO DELETE LINES
200 AND 210 BEFORE RUNNING!!

    WHEN READY TO RUN, PRESS ENTER....?



    PIANO KEYBOARD FREQUENCIES FOR EIGHT OCTAVES
              (EQUAL TEMPERAMENT SCALE)
     1      2       3       4       5       6       7       8
A   27.5   55      110     220     440     880     1760    3520
A#  29.14  58.27   116.54  233.08  466.16  932.33  1864.66 3729.31
B   30.87  61.74   123.47  246.94  493.88  987.77  1975.53 3951.07
C   32.7   65.41   130.81  261.63  523.25  1046.5  2093    4186.01
C#  34.65  69.3    138.59  277.18  554.37  1108.73 2217.46 4434.92
D   36.71  73.42   146.83  293.66  587.33  1174.66 2349.32 4698.64
D#  38.89  77.78   155.56  311.13  622.25  1244.51 2489.02 4978.03
E   41.2   82.41   164.81  329.63  659.26  1318.51 2637.02 5274.04
F   43.65  87.31   174.61  349.23  698.46  1396.91 2793.83 5587.65
F#  46.25  92.5    185     369.99  739.99  1479.98 2959.96 5919.91
G   49     98      196     392     783.99  1567.98 3135.96 6271.93
G#  51.91  103.83  207.65  415.3   830.61  1661.22 3322.44 6644.88
```

After A(8) is calculated and stored, line 165 reads A# from DATA and calls it A$. Lines 170 and 172 print—starting at the new location of P (= 192 + 64, or 256)—the new values of A(1) through A(8).

You should be able to see now that each pass through the R-loop (lines 115–190) calculates the eight octave values for each successive key, A through G#, as shown in DATA line 300.

Finally, after twelve lines of printing, line 190 increases the value of R to 13. Since this value exceeds the limit set in line 115, the program proceeds to lines 200 and 210. Instead of explaining LPRINT and PEEK here, look at Appendix D and you'll find out how they work. All you need to know here is that lines 200 and 210 print the contents of the video display on a printer exactly as they are shown on the screen except for graphics. Therefore, if you don't have a printer, these lines are not needed. Furthermore, if you attempt to run this program *with* lines 200 and 210 but *without* a printer, the program will appear to work normally, but you will *lose keyboard control* after the last line is displayed on the screen. Why? The reason is that the computer is looking frantically for a printer because of the

LPRINT statement in line 2ØØ and can't find one. The only escape from this condition, keeping the program intact, is to press the RESET button (rear left of keyboard unit, under the flip-up door). This will give you a READY but won't destroy the program in memory.* It is thus wise to delete lines 2ØØ and 21Ø unless you are using a printer.

Since the screen actually displays 96 frequencies, it goes "above" the highest piano key (which is the C shown in the eighth octave, with a value of 4186.Ø1 Hz).

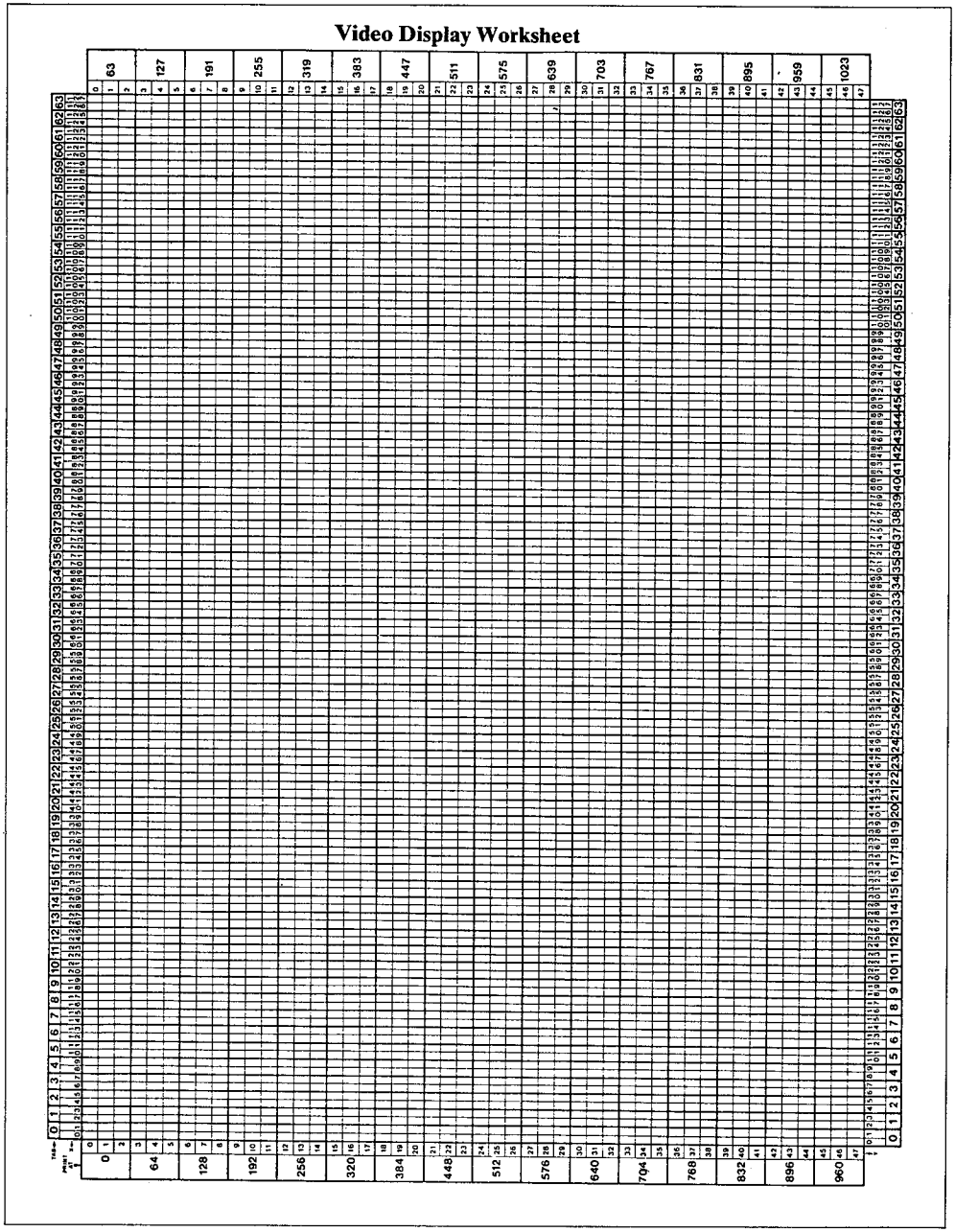
Modifications: Let A = 55 in lines 11Ø and 18Ø, and you'll show one additional octave *above* the keyboard but will lose the bottom octave.

Variables:

Z$—pause control
A—frequency of musical note A
Z—A-array location counter
E—numerator of exponent E/12
P—print location
R—note-loop counter
X—octave-loop counter
A( )—A-array
A$—musical note designation

*With an expansion interface, pressing the RESET button will destroy the resident BASIC program.

APPENDIX A
VIDEO DISPLAY WORKSHEET



APPENDIX B
CASSETTE LOADING AIDS

1. Cassette Loading Time Charts (Levels I and II)

When CLOADing a cassette into your TRS-80, the total time needed depends on several factors, the most important of which is program length. Level I tape loading takes roughly twice as long as Level II loading for programs of the same size. Even though the cassette recorder speed is the same, Level I and Level II BASIC have different "baud" (signal transmission) rates.

To give you an idea of how long it might take you to CLOAD (or CSAVE) various programs, look at the Cassette Loading Time Charts in Figs. B1 and B2. You'll notice that both the Level I and Level II lines start flat along the time axis for about 3 seconds. This segment represents a string of zeros put on the tape by the CSAVE operation to act as a "leader" on CLOADing so that the TRS-80 input electronics can attempt to adjust to the incoming signal level. The leader is the same for all program lengths and is

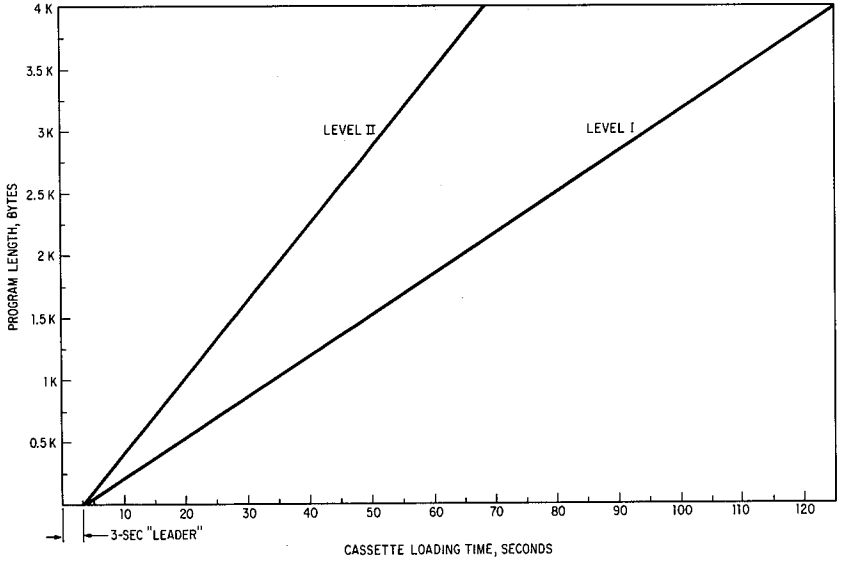


Fig. B-1. Cassette loading time chart

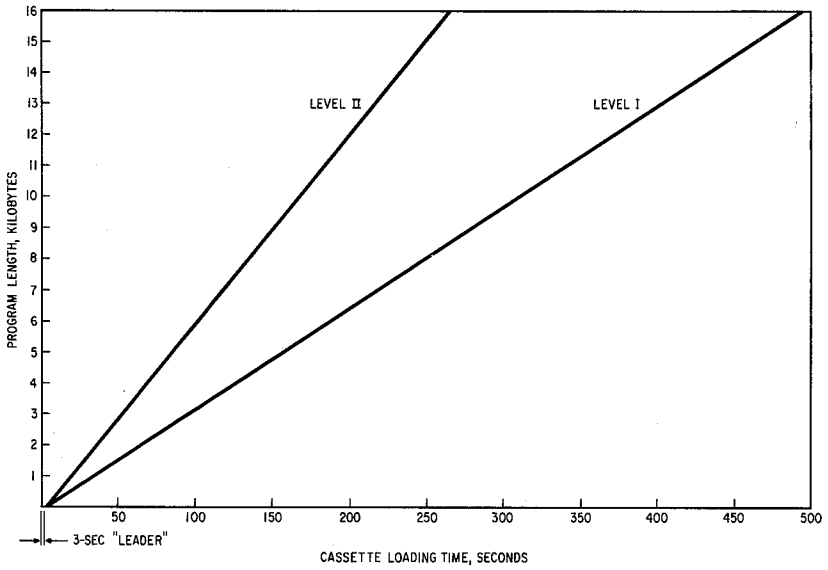


Fig. B-2. Cassette loading time chart

about the same for both Level I and Level II. The charts can be used two ways:

1. If you know how many bytes are in the program to be CSAVEd or CLOADed, enter the appropriate chart (Fig. B1 for programs under 4K, Fig. B2 for programs 4K to 16K long) at the left scale and move horizontally to the Level I or Level II line. Drop straight down from this intersection and read "seconds" from the bottom scale.
2. If you measure the loading time in seconds, enter the graph at the bottom scale, and move straight up to the Level I or II line, then over horizontally to the left-side scale and read the program byte length.

These graphs are correct enough for general use. Since individual tape recorder speed can effect graph accuracy, especially on long programs, use them only as a guide.

2. Audio/Visual Control Box

Although the Radio Shack TRS-80 Model I Microcomputer System offers many features for a low price, various design compromises were made to keep the price down. The recording and playback of cassettes is clumsy at best, requiring that you pull plugs from the recorder to move the tape when it's not under computer

control or to monitor the signals. Also, the loading signal voltage is critical, especially in Level II.

The Audio/Visual Control Box described here does the following:

1. Eliminates plug handling by providing a manual switch to control the recorder motor.
2. Allows you to hear the tape signals during both record and playback (CSAVE and CLOAD).
3. Provides a "signal level meter" for reliably loading programs from tape, especially for Level II machines.
4. Visually alerts you when any recorder buttons are depressed and the computer has stopped the recorder motor.

No modification whatever is required of the TRS-80 system since the control box is simply inserted between the existing TRS-80 cable plugs and the recorder jacks. Also, the total cost, using all new parts, is under $15!

How It Works The schematic shows the basic no-frills design. No batteries or active circuits are involved. To describe how it works, assume that the control box is installed between the computer plugs and the recorder jacks and that you're playing a tape into the computer (CLOAD or INPUT#). The tape signal comes out of the recorder earphone jack into P1. The earphone makes the signal audible, and the output transformer, T1, operating "backwards," multiplies the output voltage. Diode D1 rectifies the signal to pulsating dc for indication on meter M1. Potentiometer R1 is used to calibrate the meter reading (more on this later). By adjusting the recorder volume control for the proper meter reading, you can be sure (with a good tape) of getting a proper "load" into the computer.

Nothing at all is happening at J2 at this time. Switch SW1 would be in the "automatic" (open) position since you are letting the computer control tape movement. The TRS-80 internal relay is closed, to run the recorder motor, and the relay contacts act as a direct short circuit across J3. Therefore, no voltage appears across the series combination of resistor R2 and the light-emitting diode LED with the result that LED is dark. When the tape reaches the end of its "message" to the computer, the TRS-80 relay contacts open, making J3 an open circuit. Since the recorder "play" key is depressed, voltage appears across R2-LED, and the LED glows. This reminds you to press the recorder "stop" key. Leaving the recorder keys depressed in the play or record mode for long periods without the motor running is mechanically harmful to the recorder.

Now suppose that you want to rewind the tape. Just place SW1 in the "manual" (closed) position and press the recorder rewind button. By just putting the switch in the manual position, you can exercise full recorder keyboard control, bypassing the computer relay.

How about recording on tape? The signal comes from the computer gray mini-plug through J2 to P2 and into the recorder auxiliary jack. As the signal is being recorded, a monitor signal appears at the earphone jack. This is heard from the earphone and also appears on the meter. In this case, volume adjustment is not required since the recorder has automatic volume control and the meter reading is not critical. The switch should be in the automatic position.

Construction Any small box will do for a cabinet. Cut and file a rectangular hole in the top face of the cabinet and

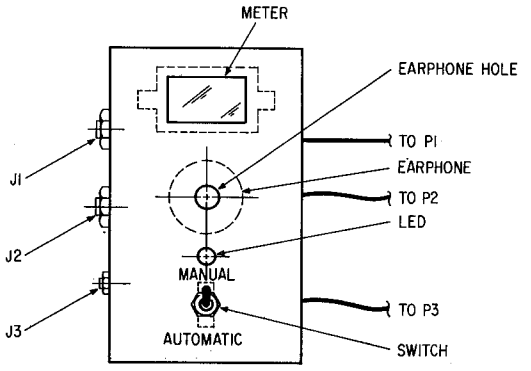


Fig. B-3. Suggested lay-out of audio/visual control box

mount the meter in position (see Fig. B3 for suggested layout). I used glue to hold the meter, but mounting ears on some small meters allow the use of small screws.

Mount the three input jacks (they may be either open-circuit or closed-circuit types) on the left side of the box, with the LED and switch on the top. Also, make a ¼-in. diameter hole in the top of the box for the earphone to "play" through. Prepare the earphone by cutting off the projecting part that normally goes in your ear, and glue the earphone into the case directly below the earphone hole. This will allow the sound from the earphone to be loud enough to be useful, but not loud enough to be annoying.

The transformer, diode, and potentiometer can be mounted on a small piece of perforated board as a subassembly

and wired to the meter and J1. Position the potentiometer so that it can be set during calibration. Wire according to the schematic, being careful to observe polarity for D1, LED, and M1, and don't forget the LED dropping resistor, R2.

Construction is completed by bringing wires through small holes in the right side of the box and soldering them to the plugs

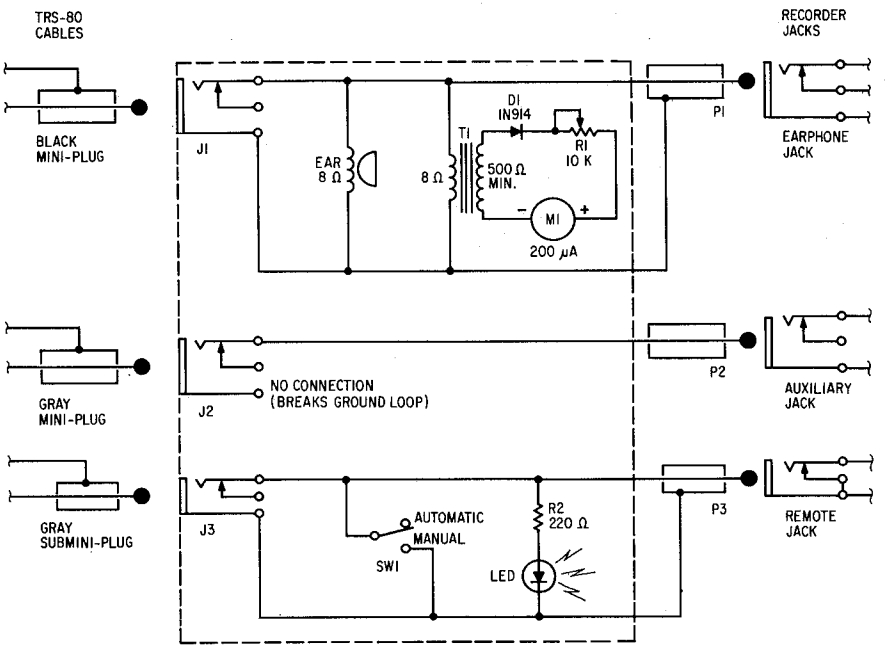


Fig. B-4. Schematic of audio/visual control box

as shown in the schematic of Fig. B4, being careful to connect the correct wire to the tip and sleeve of each plug. No shielded wire is required, and J2–P2 are connected with only a single wire to avoid a ground loop.

Calibration and Use If you have a calibrated oscilloscope, connect it across the earphone, J1 or P1, and play a known-good computer tape into the computer. Adjust the recorder volume control for 2 volts peak-to-peak on the oscilloscope; then adjust potentiometer R1 for the meter to read about ¾ scale. Note this meter reading since it will be your desired recorder-to-computer setting regardless of whether you're operating in Level I or Level II.

If you don't have an oscilloscope, you must simply find the proper loading level by trial and error and then set R1 for the desired meter reading at that signal level. From then on, any good

tape fed into the computer at that meter reading should load properly. Since the output level of tapes can vary, all you have to do is adjust the recorder playback volume unitl the meter reading is where you know it should be and the computer is receiving the proper signal voltage.

You'll very quickly find that this control box is worth ten times its cost by eliminating the frustration and unreliability of tape operation, especially in Level II. You'll be able to do other things while you *listen* to the tape playing into the computer during loading, and a bad tape is very obvious from the abnormal sound it makes. The meter will allow you to load with confidence. During CSAVE, you'll also hear the computer "talking" to the recorder, putting the signal on tape. No more watching the display, bleary-eyed, for a READY! The switch gives you positive recorder control without constantly pulling (and wrecking) plugs and jacks. Also, the glowing LED reminds you to release the recorder keyboard, thus avoiding strained springs and flat spots on the pinch-roller. Now, isn't all that worth $15?

PARTS LIST	*Radio Shack No.*	*Calectro No.*
Plastic cabinet (approx 3″ × 2″ × 1″)	270-230	H4-723
EAR (8-ohm dynamic earphone)	33-175	Q4-215
T1 (subminiature audio output xfmr, 8:500 or 8:1000)	273-1380	D1-712
M1 (200-μA subminiature meter)	—	D1-901
D1 (signal diode, 1N914 or similar)	276-1122	J4-1610
R1 (10-kilohm subminiature trimmer pot)	271-218	B1-644
J1, J2 (miniature open or closed circuit jacks)	274-292	F2-842
J3 (subminiature open or closed circuit jack)	274-251	F2-845
P1, P2 (miniature phone plug)	274-286	F2-821
P3 (subminiature phone plug)	274-289	F2-826
SW1 (SPST subminiature toggle switch)	275-612	E2-116
LED (jumbo red light-emitting diode)	276-041	J4-940

R2 (½-watt, 220-ohm carbon resistor)	271-000	B1-386
Perforated board (1¼" × 1¼")	276-139	J4-601

For the convenience of readers, a complete kit of all the above parts, plus solder and wire, is available for $14.95 plus $1 postage and handling (USA only; California residents must add 6% sales tax) from:

PPG Electronics Co.
Dept. FB
14663 Lanark St.
Van Nuys, CA 91402

3. The DATA DUBBER

You might not wish to build the Audio/Visual Control Box if you're not into electronic construction. Also, you may prefer a unit with additional versatility—the DATA DUBBER. It comes completely assembled and tested.

With too little volume, the TRS-80 misses data bits; too much volume causes an overload. The upper and lower volume limits create a loading "window." Good tapes have a window several numbers wide on the volume knob. Troublesome tapes have a very narrow window, which makes the volume setting critical.

The DATA DUBBER opens the window so that setting the volume is much less critical. It connects between your cassette player and the TRS-80 to filter the incoming data pulses. It actually ignores noise, volume variations, static, and distortion. A tiny integrated circuit chip (incorporating several hundred transistors, diodes, and resistors) takes the incoming data and reproduces (or regenerates) the original CSAVE data pulses. The data is "laundered" by the DATA DUBBER for a clean CLOAD. No modifications whatever are required to your TRS-80.

You can also copy any TRS-80 program cassette with the DATA DUBBER. While you can CSAVE BASIC programs without a DATA DUBBER, it can take a long time to load and save an entire cassette or program. By connecting the DATA DUBBER between two recorders, you can copy any tape in minutes. You can even make perfect replicas of SYSTEM tapes that supposedly can't be copied.

By connecting a Radio Shack speaker/amplifier (277-1008)) to the DATA DUBBER, you can monitor your tapes

as they are loading without modifying your CTR-41. This is the same amplifier that is used with game programs featuring sound. With this monitor, you can listen for data dropouts or gaps that make the tape unusable.

Since a data-actuated power switch is built-in, you don't have to turn the unit on and off. The DATA DUBBER draws power from the battery only when you input data. The inexpensive 9-volt battery (supplied) lasts for many months.

The DATA DUBBER is available ($49.95 postpaid; Washington residents add sales tax) from:

The Peripheral People
Dept. FB
P.O. Box 524
Mercer Island, WA 98040

APPENDIX C
Level I/II Conversions

The TRS-80 BASIC used in Level I and Level II has a lot of similarities in "language." However, some significant differences can prevent an apparently compatible Level I program from running in Level II.

The most obvious difference is in tape format. Level I tapes will *not* load in a Level II machine without some form of conversion tape. Radio Shack supplies PROGRAM and DATA conversion tapes to allow loading Level I tapes into a Level II TRS-80. These tapes accept the Level I tape format, expand Level I abbreviations to full "words," and change PRINTAT to PRINT@. However, these tapes do *not* correct for some different Level II syntax requirements.

Using the keyboard of a Level II TRS-80 to enter a Level I listing presents yet another problem. No abbreviations are allowed in Level II. If the Level I abbreviation is a single letter followed by a period (such as L. for LIST, P. for PRINT, R. for RUN, etc.), it is *not* valid for Level II.

Thus, whether using the Radio Shack conversion tapes or keyboard, you finally have the Level I program loaded into your Level II TRS-80 in Level II language, or so you think until you try to run it! The "error messages" generated by Level II will challenge your ingenuity unless you pay attention to the following Level II rules:

1. PRINT@ in Level II must be followed by a *comma, not* a semicolon, which is allowed in Level I. When typing in PRINT@, do *not* use the SHIFT key for any of the letters or the @.
2. PRINTAB can be followed by a comma in Level I. In Level II, this has to be changed to a semicolon or omitted altogether!
3. Level I uses only single-letter variables. Since Level II allows two letters (or a letter and a number) as a variable, some statements need proper spacing in Level II. For example, Level II will not properly interpret FORATOB to mean FOR A TO B. If in doubt, use spaces!

4. Arrays do not require DIMensioning in Level I, regardless of size. In Level II, you must use DIM (array designation) for any array larger than the first 11 elements (Ø–1Ø). Furthermore, if the program loops back for a rerun, the DIM(x) statement must appear on a program line *prior* to the rerun line or a ?DD ERROR message will tell you that the array was already DIMensioned earlier in the same program.
5. Level I graphics automatically "wrap around" the screen. That is, if SET coordinates exceed (127,47), they are reset to zero. However, this is not the case in Level II, which comes up with an ?FC ERROR if SET (127,47) is exceeded in either the X or Y direction. You must edit or rewrite the program to avoid this.
6. BASIC allows you to use the POINT(X,Y) to determine if a screen location (X,Y) is lighted (SET). In Level I, a SET location yields a 1. However, in Level II, a SET location yields a −1. Both Level I and Level II return a Ø if the POINT location is *not* SET.
7. FOR-NEXT loops are executed at a slower speed in Level II than in Level I. However, if the loop variable is defined as an integer with DEFINT, then the Level II loop is *faster* than Level I. Therefore, if a loop is used in a Level I program for timing, the limit number needs to be changed for Level II or the timing will be different.
8. The logical operators * and + of Level I *may* have to be replaced by AND and OR respectively in some Level II machines. Some Level II BASIC ROMs recognize both forms, whereas *all* Level II ROMs recognize AND and OR.
9. Level II will *not* accept a letter entry for a numerical variable. Letter variables, such as Y for YES, or N for NO, must be input to a string variable. See Program 2 (Level II version) for some examples.
10. When assigning values to strings in Level II, the value must be enclosed in quotes. For example, A$ = "HARRY".

There's an even easier way of handling the whole "problem" of Level I cassette programs if you move up to Level II 16K. "BASIC-1P" is a machine-language cassette tape that loads into the top 4K of RAM, leaving about 12K for your Level I programs. If you CLOAD in any BASIC Level I program or data tape, your Level II TRS-80 will act like a Level I machine. All Level I commands and abbreviations are supported, and there are four

new printer commands that don't even exist in Level I. They allow you to LLIST, or LPRINT, on printer and to LPRINT on both screen *and* printer!

BASIC 1P is available for $19.95, postpaid (plus sales tax for California residents), from:

Small System Software
Dept. FB
P.O. Box 366
Newbury Park, CA 91320

APPENDIX D
"SLICK TRICKS"
(Level II Only)

As a beginner with the TRS-80, you probably had your hands full just learning BASIC, Level I or Level II. However, as you advance further, you'll be looking for more sophisticated programs, with fancy subroutines, and you'll probably get a printer. And once you have a printer, you'll be looking for ways to print the contents of the screen, including the graphics. You can, and easily! Here are some "tricks" and short routines I have picked up along the way. They will work in cassette-based Level II but may not work the same way (or at all) with DISK.

1. Keyboard Debounce Program

```
3000 V=15360:LPRINTCHR$(29)
3010 FOR C=0 TO 15:FOR R=0 TO 63
3020 A=PEEK(V+64*C+R)
3030 IF A<192 AND A>128 LPRINT"*";:GOTO3050
3040 LPRINTCHR$(A);
3050 NEXT:LPRINT"":NEXT
3060 RETURN
```

This is so short and simple you can put it at the beginning of any of your programs. I keep it on a very short tape and make it a habit to load it into the TRS-80 whenever I turn it on. I have no idea how it works, but it entirely eliminates undesired repeating characters when I use my keyboard and without any excessive entry delays. The entry delay time, incidentally, is set by the value "20" in line 40 of the program. This could be a larger or smaller number, but 20 seems just right. If you change it, also change the value of "20" in line 50 to the new number. Line 50, of course, could be eliminated entirely, but I prefer the security of the TRS-80 looking (PEEK) into the value set in memory location 16486 and seeing if it's the specified value. This is assurance that the required POKEs (establishing DATA values at specified memory locations) have been accepted. I understand that these POKEs are in the DOS (disk operating system)

area of memory; consequently, this probably won't work with a disk system.

2. Double CSAVE

When CSAVEing programs on cassette, it's a good idea to make two copies, with a short space between them. Normally, you do this by entering CSAVE"1" (or any other single number or letter inside the quotes), then advancing the tape after the CSAVE is complete, and doing it again.

The following is easier (type in and ENTER with recorder in RECORD mode; no line number is required):

```
CSAVE"1":OUT255,4:FOR I=1 TO 2000:NEXT:CSAVE"1"
```

The first CSAVE operates normally, but instead of the recorder stopping at the end of the CSAVE, the OUT255,4 keeps it running for a period determined by the interval FOR-NEXT loop—about 2 seconds. Then the next CSAVE is performed.

To verify, rewind the tape to the beginning and CLOAD? each CSAVE separately. I have not found a way to *double* CLOAD? If *you* do, please let me know how.

3. Merging BASIC Programs

This trick is really handy when you have two related BASIC programs on cassette that you'd like to combine or you have a long subroutine on tape that you'd like to add to a BASIC program in memory or on cassette, without all the keyboard time and debugging hassle.

Let's say that you have a BASIC Program A to which you'd like to add Program B, a subroutine. First, make sure that *all* the line numbers of Program B are *higher* than any of the Program A line numbers. If not, change the Program B line numbers to make them higher. This is most easily done with any of several RENUMBER programs. The simplest and least expensive RENUMBER program, perhaps, is "TRS-80 Utility I" (No. 0081R) for 16K Level II machines, available for $7.95 (plus $1.00 shipping) from

Instant Software
Dept. FB
Peterborough, NH 03458

Renumbering programs is less trouble than making keyboard changes and less likely to cause *other* problems since any decent renumber program also changes GOTO, IF-THEN, and other operational line number references within the program as it changes the line numbers.

Anyhow, we'll assume that Program B has higher line numbers than Program A and that Program A is in memory, either from keyboard or CLOAD. Verify that Program A operates properly. Type in and ENTER the following line (no line number):

```
POKE 16548,PEEK(16633) – 2:POKE16549,PEEK(16634)
```

The screen should respond with a READY prompt. If the screen shows an error message, simply add two characters anywhere in Program A and try again. In effect, this instruction looks at the values that determine the program pointer location and changes them to move the pointer to the *end* of Program A. If you try to LIST Program A, you'll get nothing, but it's still there, below the program pointer location.

Now CLOAD Program B in the normal manner. However, it is not actually appended to Program A until the program pointer is moved back to the *beginning* of Program A. This is done by simply typing and ENTERing the following line (no line number):

```
POKE16548,233:POKE16549,66
```

You may now LIST, RUN, or CSAVE the combined programs in the normal way. Isn't that worth knowing about?

4. Screen Print (No Graphics)

After you've had your printer a while and used LLIST to print program listings and LPRINT within program runs, you'll become annoyed at the things your printer does *not* show. For example, most printers ignore graphic codes completely. Also, PRINT@ statements appear at the beginning of the next printer line. Furthermore, if you use TAB for forms or tabulations, your printer may interpret TAB differently than the screen does, and you will end up with a mess.

Ideally, you'd like the printer to show everything *exactly* as it appears on the screen. "Screen printers" will do this, but they are expensive. However, this simple program will output to your printer all characters (except graphic) as they appear on your screen:

```
3ØØØ  V = 15360:LPRINT CHR$(29)
3Ø1Ø  FOR C = Ø TO 15:FOR R = Ø TO 63
```

```
3020  LPRINT CHR$(PEEK(V+64*C+R));
3030  NEXT:LPRINT " ":NEXT
3040  END -OR- 3040 RETURN
```

The line numbers, of course, may be changed to anything you like. This can be treated as a subroutine and embedded anywhere within your program, but placing it at the end of the program is considered the best "style."

Let's examine, line by line, how it works. What we want to do, somehow, is examine every screen printing location, tell the printer what's at that location, and then have it printed there. Line 3000 starts by using variable V and setting it equal to 15360. Why 15360? That is the decimal memory location of the first printing space (upper left corner of your screen, print position 0 in Appendix A). Print position 1 is memory location 15361, and so on to the last location (lower right corner, print position 1023), which has a decimal memory location of 16383.

The next instruction on line 3000 tells the computer to LPRINT CHR$(29).* That commands the printer to return to the beginning of the next line—like a carriage return on an electric typewriter. Now the printer is poised at the beginning of a line, waiting for further instructions.

Line 3010 sets up two FOR-NEXT loops. The C-loop counts 16 screen lines, 0 to 15. The R-loop counts 64 printing positions on a line, 0 to 63. In effect, these loops are used to "scan" the screen memory locations, one location at a time, from left to right, until a line is completed, and line by line from screen top to bottom.

Line 3020 determines the ASCII code designation—computer character codes—for each memory location by PEEKing. The V is 15360, the 64*C is the line number, and the +R is the character location along the line. The LPRINT CHR$ instruction preceding the PEEK tells the printer to print that character. The semicolon at the end of the line allows the printer to advance only to the next print position and stops it there.

Line 3030 starts with NEXT, which advances R to the next higher number. Therefore, every position along that screen line is inspected, one by one, and the printer outputs a character or space. Graphic codes are ignored; blank spaces are printed in those locations.

*Your printer may interpret this differently. The OKIDATA Microline 80, for example, interprets LPRINT CHR$(29) as a command to print narrow letters. Use the CHR$ number your printer sees as a carriage return command.

When NEXT advances the R to 64, the LPRINT " " on line 3Ø3Ø executes a carriage return, and the following NEXT advances C by 1 to the next screen line memory locations when calculated by the PEEK inspection on line 3Ø2Ø. Ingenious, isn't it? The formula in line 3Ø2Ø, together with the C and R loops, provides a simple way to inspect every screen memory location and gives this information to the printer one character at a time.

Line 3Ø4Ø may either END the program or RETURN it to a GOSUB call in the program. This little program is an extremely useful subroutine when you have a main program with lots of text and you don't want to add LPRINT statements throughout the program. Just add a line with GOSUB 3ØØØ after each screenful.

Since each character is being determined and printed one at a time, this program usually prints out more slowly than a regular LPRINT statement. However, the convenience is worth the extra time. Furthermore, PRINT@ statements and tabulations appear on the printer just as they do on the screen.

5. Screen Print (Using # for Graphics)

For those programs heavy in graphics, especially games, use this subroutine:

```
10 FORN=16480TO16492:READK:POKEN,K:NEXT
20 FORN=16435TO16437:READK:POKEN,K:NEXT
30 POKE16405,0
40 DATA205,227,3,133,200,14,20,16,254,13,32,251,201,195,96,64
50 IF PEEK(16486)=20 PRINT"DEBOUNCE PROGRAM INSTALLED"
```

The similarity with the nongraphic screen printing program (Section 4) is obvious. All we have done here is to determine if the ASCII code stored in a screen memory location is a visible graphic code (129–191; 128 is a blank space). If so, a # is printed at that location. Of course, you can replace the # with any other character you please, including ! or * or + or whatever. Just enclose your desired character in the quotes in line 3Ø3Ø.

If this is *not* used as a subroutine, then line 3Ø6Ø should be END or GOTO (line number) instead of RETURN; otherwise, the computer will terminate your RUN with an ?RG ERROR.

6. Software Printer/Screen Switch

If you have a program with a lot of PRINT statements and you want to output to a printer, the LPRINT statement is

used. This could mean either *changing* some or all of the PRINT statements within the program to LPRINT or *adding* a bunch of LPRINT statements after PRINT statements to get output to both screen and printer.

The following simple one-line program acts like a switch. It changes all PRINT statements in a program to LPRINT. Used a second time, it changes back to PRINT.

```
FOR I=16414 TO 16415: K=PEEK(I): J=PEEK(I+8): POKE I, J:
  POKE I+8, K: NEXT
```

This can be entered from the keyboard without a line number before a program run or embedded within the program in one or more places *with* line numbers. It can also be used as a subroutine by adding RETURN at the end of the line when used within a program. Remember that this line, when used only *once,* converts PRINT to LPRINT. Therefore, you must use it a *second time* to regain screen output. If used only once, all output—including keyboard entry—that normally goes to the screen will instead go to the printer. This makes your printer an electric typewriter, with no backspacing, and nothing shown on the screen. Simply type the "switch" line in from the keyboard, or GOSUB the line number in your program where the "switch" resides in memory.

Unfortunately, there's one limitation to the use of this simple "switch," at least on my printer. It doesn't work properly if more than one PRINTTAB statement is used on a printed line. The printer counts TAB spaces from the last printing location instead of the beginning of the line. If you try to bring the cursor back to the beginning of the line with a CHR$(29), the printer returns all right, but advances to the next line.

If you find out how to handle several TABs on one line using this "switch," please let me know. Of course, the Screen Prints (sections 4 and 5) handle TABs with no problem.

MATRI-DEX™

PROGRAM NO.	LEVEL I	LEVEL II	PROGRAM TITLE / STATEMENT OR OPERATION	AND	Arrays	Entry-error Trap	Erase Printing	FOR-NEXT	GOSUB-RETURN	GOTO	Graphic Symbols	Greater Than >	IF-GOTO or IF-THEN	INPUT	INT (X)	Less Than <	LPRINT	ON-GOTO	OR	PEEK	POKE	PRINTAT or PRINT@	READ-DATA	Rerun Program	RESTORE	RND(X)	Screen Printing	SET (x,y)	String Comparison	Strings	TAB(X)	Timing Loop
1	●	●	DISPLAY ALIGNMENT				●															●		●				●		●	●
2	●	●	COMPUTER INTERVIEW					●		●			●	●								●								●	
3	●	●	FORTUNE TELLER					●										●						●					●		●
4	●	●	ASK YOUR GOVERNMENT!			●								●																	
5	●	●	RACING ALPHABET	●		●	●	●	●	●		●	●	●								●						●			
6		●	RUN, SPOT, RUN!				●	●		●	●	●	●						●			●									
7	●	●	BINGO CALLER/VERIFIER		●		●	●		●		●	●			●						●	●								
8	●	●	$600 DIGITAL CLOCK	●		●		●				●										●									
9	●	●	$600 DIGITAL ADDING MACHINE																			●									
10	●	●	ORDER FORM				●															●									
11	●	●	BOOKKEEPING			●		●											●												
12	●	●	INTEREST CALCULATIONS	●		●		●																							
13	●	●	INVEST OR SAVE?					●																							
14	●	●	MORTGAGE LOAN			●																									
15	●	●	PAY NOW OR PAY MONTHLY?																												
16	●	●	PHONE TOLL-CHARGE																												
17	●	●	SPIRANGLE																												
18	●	●	CUSTOM SCREEN DISPLAY																												
19	●	●	CHANGING BILLBOARD																												
20	●	●	MAGIC SQUARE																												
21		●	PIANO KEYBOARD FREQUENCIES		●																										
–	●	●	APPENDIX D–SLICK TRICKS																												

5182-4
$8.95

PROGRAMS FOR BEGINNERS ON THE TRS-80™

Fred Blechman

Here's a valuable book of practical and interesting programs for home use that can be understood and used immediately by the beginner in personal computer programming. You'll learn step-by-step how 21 sample TRS-80 programs work. Program techniques are described line-by-line within the programs, and a unique Matri-Dex™ matrix index will enable you to locate other programs using the same BASIC commands and statements. Each program includes a detailed description, a complete listing, an explanation of what the program does, and instructions for modification.

Other Books of Interest . . .

TEN EASY PIECES: Creative Programming for Fun and Profit

Carl Meyer, Jr., and Hans Sagan

". . . a nicely written book, of interest to anybody who wants to write games, or who wants to know more about the subtleties of advanced programming in BASIC. . . ." *Creative Computing.* Requires little knowledge of elementary mathematics. #5160-3, paper, 192 pages

BASIC COMPUTER PROGRAMS FOR THE HOME

Charles D. Sternberg

Over 75 practical home application programs that are documented with a description of operations and functions. Each program includes a listing in BASIC language, a symbol table, sample data, and one or more output samples. #5154-9, paper, 336 pages

BASIC BASIC: An Introduction to Computer Programming in BASIC Language, Second Edition

James S. Coan

". . . an excellent introduction to the use of BASIC . . . clearly written and well-organized." *Computing Reviews.* "It is a well-written book . . . there are many good examples, complete with results." *Computer World.* Contains over 100 sample programs. #5106-9, paper, 288 pages

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

ISBN 0-8104-5182-4