dilithium Press

Tom Rugg, Phil Feldman and Raymond Alen

32 BASIC PROGRAMS FOR THE TI-99/4A®

32 BASIC Programs for the TI-99/4A Computer

Programs for either TI BASIC or TI Extended BASIC

32 BASIC Programs for the TI-99/4A Computer

Programs for either TI BASIC or TI Extended BASIC

Tom Rugg, Phil Feldman,
and Raymond Alen

dilithium Press
Beaverton, Oregon

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data

Rugg, Tom.
32 BASIC programs for the TI-99/4A computer.

Bibliography: p.
1. T1 99/4A (Computer) — Programming. 2. Basic (Computer program language) I. Feldman, Phil, Phil. II. Alen, Raymond, 1965– . III. Title. IV. Title: Thirty-two BASIC programs for the TI-99/4A computer.

QA76.8.T133R84 1984 001.64'2 83-23962
ISBN 0-88056-136-X

Cover: Vernon G. Groff

Printed in the United States of America

dilithium Press
8285 S.W. Nimbus
Suite 151
Beaverton, Oregon 97005

Acknowledgements

Our thanks to the following for their help and encouragement: Our wives and families, Merl and Patti Miller, Asenatha McCauley, Herb Furth, and Freddie.

AN IMPORTANT NOTE

Preface

You have bought yourself a Texas Instruments TI-99/4A computer (or maybe you just have access to one at school or work). You will soon find that the most frequent question you are asked goes something like this: "Oh, you got a computer, eh? Uh. . . what are you going to do with it?"

Your answer, of course, depends on your own particular situation. Maybe you got it for mathematical work, or for your business, or for home usage, or to enable you to learn more about computers. Maybe you got it for a teaching/learning tool or for playing games.

Even if you got the computer specifically for only one of these reasons, you should not neglect the others. The computer is such a powerful tool that it can be used in many different ways. If it is not being used for its "intended" function right now, why not make use of it in some other way?

The TI-99/4A is so small and portable that you can, say, take it home from work over the weekend and let the kids play educational games. They will have fun *and* learn a lot. After they go to bed, you can use it to help plan your personal finances. Or, you can let your guests at a party try to outsmart the TI-99/4A (or each other) at some fascinating games. The possibilities go on and on.

All these things can be done with the TI-99/4A, but it cannot do any of them without the key ingredient – a computer program. People with little or no exposure to computers may be in for a surprise when they learn this. A computer without a program is like a car without a driver. It just sits there.

So you ask, "Where can I get some programs to do the things I want my computer to do?" Glad you asked. There are several alternatives.

1. Hire a computer programmer. If you have a big budget, this is the way to go. Good programmers are expensive and hard to find (and you will not know for sure if they're really good until after the job is finished). Writing a couple of programs that are moderately complex will probably cost you more than you paid for your TI-99/4A.
2. Learn to program yourself. This is a nice alternative, but it takes time. There are lots of programming books available—some are good, some not so good. You can take courses at local colleges. If you can afford the time and you have a fair amount of common sense and inner drive, this is a good solution.
3. Buy the programs you want. This is cheaper than hiring your own programmer because all the buyers share the cost of writing the programs. You still will not find it very cheap, especially if you want to accumulate several dozen programs. Each program might cost anywhere from a few dollars to several hundred dollars. The main problem is that you cannot be sure how good the programs are, and, since they are generalized for all possible buyers, you may not be able to easily modify them to do exactly what *you* want. Also, they have to be written in a computer language that *your* computer understands. Even if you find a program written in the BASIC language, you will soon learn that TI's BASIC is not the same as other versions. Variations between versions of the same language typically result in the program not working.

This book gives you the chance to take the third alternative at the lowest possible cost. If you divide the cost of the book by the number of programs in it (use your computer if you like), you will find that the cost per program is amazingly low. Even if there are only a few programs in the book that will be useful to you, the cost is pretty hard to beat.

Just as important is the fact that these programs are written specifically for your TI-99/4A. If you type them in exactly as shown, they will work! No changes are needed. In addition, we show you exactly what to change in order to make some simple modifications that may suit your taste or needs. Plus, if you have learned a little about BASIC, you can go even further and follow the suggestions about more extensive changes that can be made. This approach was used to try to make every program useful to you, whether you are a total beginner or an old hand with computers.

But enough of the sales pitch. Our main point is that we feel a computer is an incredibly flexible machine, and it is a shame to put it to only one or two limited uses and let it sit idle the rest of the time. We are giving you a pretty wide range of things to do with your TI-99/4A, and we are really only scratching the surface.

So open your eyes and your mind. Play a mental game against the computer (WARI, JOT). Evaluate your next financial decision (LOAN, DECIDE). Expand your vocabulary or improve your reading speed (VOCAB, TACHIST). Solve mathematical equations (DIFFEQN, SIMEQN).

But please, don't leave your TI-99/4A asleep in the corner too much. Give it some exercise.

How to Use This Book

Each chapter of this book presents a computer program that runs on the TI-99/4A with either "regular" TI BASIC or TI Extended BASIC. Two of the programs (ROADRACE and WALLOONS) *require* TI Extended BASIC. All will run on a 16K TI-99/4A. Each chapter is made up of eight sections that serve the following functions:

1. **Purpose:** Explains what the program does and why you might want to use it.
2. **How To Use It:** Gives the details of what happens when you run the program. Explains your options and the meanings of any responses you might give. Provides details of any limitations of the program or errors that might occur.
3. **Sample Run:** Shows you what you will see on your screen when you run the program.
4. **Program Listing:** Provides a "listing" (or "print-out") of the BASIC program. These are the instructions to the computer that you must provide so it will know what to do. You must type them in extremely carefully for correct results.
5. **Easy Changes:** Shows you some very simple changes you can make to the program to cause it to work differently, if you wish. You do not have to understand how to program to make these changes.
6. **Main Routines:** Explains the general logic of the program, in case you want to figure out how it works. Gives the BASIC line numbers and a brief explanation of what each major portion of the program accomplishes.
7. **Main Variables:** Explains what each of the key variables in the program is used for, in case you want to figure out how it works.

8. **Suggested Projects:** Provides a few ideas for major changes you might want to make to the program. To try any of these, you will need to understand BASIC and use the information provided in the previous two sections (Main Routines and Main Variables).

To use any of these programs on your TI-99/4A, you need only use the first four sections. The last four sections are there to give you supplementary information if you want to tinker with the program.

RECOMMENDED PROCEDURE

Here is our recommendation of how to try any of the programs in this book:

1. Read through the documentation that came with the TI-99/4A to learn the fundamentals of communication with the computer. This will teach you how to turn the computer on, get into TI BASIC, enter a program, correct mistakes, run a program, etc.
2. Pick a chapter and read Section 1 ("Purpose") to see if the program sounds interesting or useful to you. If not, move on to the next chapter until you find one that is. If you are a beginner you might want to try one of the short "Miscellaneous Programs" first.
3. Read Sections 2 and 3 of the chapter ("How To Use It" and "Sample Run") to learn the details of what the program does.
4. Enter the NEW command to eliminate any existing program that might already be in your TI-99/4A's memory. Using Section 4 of the chapter ("Program Listing"), carefully enter the program into the TI-99/4A. Be particularly careful to get all the punctuation characters right (i.e., commas, semicolons, colons, quotation marks, etc.). *Be sure you have the ALPHA LOCK key set* (down) *when entering and running these programs.* You may find it convenient to use the NUMBER command to have the computer generate line numbers for you.
5. *After the entire program is entered into the TI's memory, use the LIST command to display what you have entered so you can double check for typographical errors, omitted lines, etc.* Don't mistake a semicolon for a colon, or an alphabetic I or O for a numeric 1 or 0 (zero). Take a minute to note the differences in these characters before you begin.
6. Before trying to RUN the program, use the SAVE command to save the program temporarily on cassette or disk. This could prevent a lot of wasted effort in case something goes wrong (power failure, computer malfunction, etc.).

7. Now RUN the program. Is the same thing happening that is shown in the Sample Run? If so, accept our congratulations and go on to step 9. If not, stay cool and go to step 8.
8. If you got an INCORRECT STATEMENT error or a SYNTAX ERROR in a line, LIST that line and look at it closely. Something is not right. Maybe you interchanged a colon and a semicolon. Maybe you typed a numeric 1 or 0 instead of an alphabetic I or O. Maybe you misspelled a word or omitted one. Keep looking until you find it, then correct the error and go back to step 7.

   If you got some other kind of error message, consult the TI-99/4A documentation for an explanation. Keep in mind that the error might not be in the line that is pointed to by the error message. It is not unusual for the mistake to be in a line immediately preceding the error message line. Another possibility is that one or more lines were omitted entirely. In any event, fix the problem and go back to step 7.

   If there are no error messages, but the program is not doing the same thing as the Sample Run, there are two possibilities. First, maybe the program isn't *supposed* to do exactly the same thing. Some of the programs are designed to do unpredictable things to avoid repetition (primarily the game programs and graphic displays). They should be doing the same *types* of things as the Sample Run, however.

   The second possibility is that you made a typing error that did not cause an error message to be displayed, but simply changed the meaning of one or more lines in the program. These are a little tricky to find, but you can usually narrow it down to the general area of the problem by noting the point at which the error takes place. Is the first thing displayed correct? If so, the error is probably after the PRINT or DISPLAY statement that caused the first thing to be displayed. Look for the same types of things mentioned before. Make the corrections and go back to step 7.
9. Continue running the program, trying to duplicate the Sample Run. If you find a variation that cannot be accounted for in the "How To Use It" section of the chapter, go to step 8. Otherwise, if it seems to be running properly, SAVE the program on cassette or disk.
10. Read Section 5 of the chapter ("Easy Changes"). Try any of the changes that look interesting. If you think the changed version is better, SAVE it on cassette or disk, too. You will probably want to give it a slightly different title in the first REM statement to avoid future confusion.

AN NOTE ON THE PROGRAM LISTINGS

A line of text on the screen of the TI-99/4A is 28 characters wide. The printer that was used to create the Program Listing section of each chapter prints lines up to 80 characters long. For best reproduction in this book, it was preferable that each published line be no longer than 46 characters. This combination of facts might cause you a little confusion when you are entering the programs into your TI-99/4A. Here's the way it works.

Wherever there is a line in a program that is longer than 46 characters, it has been divided into two or more lines that are each no more than 46 characters. You can recognize this easily because the second part has no line number at the left-hand side. This division is only for the purpose of printing the book. You should think of a divided line like this as one long line and enter it into your TI-99/4A as a single line.

Don't be fooled by the fact that the cursor on your TI-99/4A jumps down to the next line after you enter the 28th character—it's just one long line until you press ENTER.

USING A DISKETTE OR CASSETTE

If you purchased this book as a software package you will need loading instructions to get the programs into your computer. The loading instructions for both diskette and cassette are in the back of this book. Once you have loaded the programs you can follow the recommendations supplied earlier except that you have saved lots of time by not having to type in the programs yourself.

Contents

How to Use This Book xi

Section 1—APPLICATIONS PROGRAMS
Practical uses at home or work.

BIORHYTHM	*Can you predict your good and bad days?*	3
CHECKBOOK	*Balance your checkbook.*	13
DECIDE	*Choose the best decision from your list of alternatives.*	25
LOAN	*Calculate payments and interest for mortgages, car loans, etc.*	41
MILEAGE	*Analyze your car's gasoline usage.*	51
QUEST/EXAM	*Determine the results of questionnaires and examinations.*	61

Section 2—EDUCATIONAL PROGRAMS
To help yourself or others to learn.

ARITHMETIC	*Math drills for school-aged children.*	75
FLASHCARD	*Create your own flashcards, then practice.*	87
METRIC	*Learn the metric system.*	95
NUMBERS	*Help pre-school children to learn numbers.*	105
TACHIST	*Increase your reading speed with this tachistoscope.*	111
VOCAB	*Expand your vocabulary.*	121

Section 3—GAME PROGRAMS
Match wits with the computer or a friend.

DECODE	*Figure out the computer's secret code.*	131

GROAN	*This dice game will make you do just that.*	141
JOT	*Challenge the computer to this word and logic game.*	157
OBSTACLE	*Play a friend at this arcade-like game.*	171
ROADRACE	*Try to keep the car on the road.*	179
WARI	*Attempt to beat the computer at this ancient African skill game.*	187

Section 4—GRAPHICS DISPLAY PROGRAMS

Dazzling visual diversions.

KALEIDO	*A kaleidoscope with eight point symmetry.*	201
SPARKLE	*Hypnotic sparkling patterns.*	207
SQUARES	*Overlaying concentric squares.*	213
WALLOONS	*A classic from the TI-99/4A Theatre.*	217

Section 5—MATHEMATICS PROGRAMS

For math, engineering, and statistical uses.

CURVE	*Perform least-squares curve fitting.*	229
DIFFEQN	*Solve ordinary differential equations.*	243
GRAPH	*Display a simple graph of a function.*	253
INTEGRATE	*Determine the area under a curve.*	265
SIMEQN	*Solve simultaneous linear equations.*	273
STATS	*Perform standard "statistics of population" calculations.*	281

Section 6—MISCELLANEOUS PROGRAMS

Short programs that do interesting things.

BIRTHDAY	*What are the odds of identical birthdays in a group of people?*	299
PI	*Calculate an approximation of pi.*	303

POWERS	*Calculate powers of integers—up to 250 digits long.*	**309**
PYTHAG	*Generate Pythagorean triplets ($a^2 + b^2 = c^2$).*	**315**

Bibliography 321

Loading Instructions for Diskette 323

Loading Instructions for Cassette 325

Section 1

Applications Programs

Good practical applications are certainly a prime use of personal computers. There are a myriad of ways the TI-99/4A can help us to do useful work. Here are six programs for use around the home or business.

Financial considerations are always important. LOAN will calculate interest, payment schedules, etc., for mortgages, car loans, or any such business loan. Do you ever have trouble balancing your checkbook(s)? CHECKBOOK will enable you to rectify your monthly statements and help you find the cause of any errors.

Fuel usage is a constant concern for those of us who drive. MILEAGE will determine and keep track of a motor vehicle's general operating efficiency.

The tedium of analyzing questionnaires and examinations can be greatly relieved with the aid of your computer. In particular, teachers and market researchers should find QUEST/EXAM useful.

Often we are faced with difficult decisions. DECIDE transforms the TI-99/4A into a trusty advisor. Help will be at hand for any decision involving the selection of one alternative from several choices.

Before anything else, you might want to consult BIORHYTHM each day. Some major airlines, and other industries, are placing credence on biorhythm theory. If you agree, or "just in case," simply turn on your TI-99/4A and load this program.

BIORHYTHM

PURPOSE

Did you ever have one of those days when nothing seemed to go right? All of us seem to have days when we are clumsy, feel depressed, or just cannot seem to force ourselves to concentrate as well as usual. Sometimes we know why this occurs. It may result from the onset of a cold or because of an argument with a relative. Sometimes, however, we find no such reason. Why can't we perform up to par on some of those days when nothing is known to be wrong?

Biorhythm theory says that all of us have cycles, beginning with the moment of birth, that influence our physical, emotional, and intellectual states. We will not go into a lot of detail about how biorhythm theory was developed (your local library probably has some books about this if you want to find out more), but we will summarize how it supposedly affects you.

The physical cycle is twenty-three days long. For the first 11½ days, you are in the positive half of the cycle. This means you should have a feeling of physical well-being, strength, and endurance. During the second 11½ days, you are in the negative half of the cycle. This results in less endurance and a tendency toward a general feeling of fatigue.

The emotional cycle lasts for twenty-eight days. During the positive half (the first fourteen days), you should feel more cheerful, optimistic, and cooperative. During the negative half, you will tend to be more moody, pessimistic, and irritable.

The third cycle is the intellectual cycle, which lasts for thirty-three days. The first half is a period in which you should have greater success in learning new material and pursuing creative, intellectual activities. During the second half, you are supposedly better off reviewing old material rather than attempting to learn difficult new concepts.

The ups and downs of these cycles are relative to each individual. For example, if you are a very self-controlled, unemotional person to begin with, your emotional highs and lows may not be very noticeable. Similarly, your physical and intellectual fluctuations depend upon your physical condition and intellectual capacity.

The day that any of these three cycles changes from the plus side to the minus side (or vice versa) is called a "critical day." Biorhythm theory says that you are more accident-prone on critical days in your physical or emotional cycles. Critical days in the intellectual cycle aren't considered as dangerous, but if they coincide with a critical day in one of the other cycles, the potential problem can increase. As you might expect, a triple critical day is one on which you are recommended to be especially careful.

Please note that there is quite a bit of controversy about biorhythms. Most scientists feel that there is not nearly enough evidence to conclude that biorhythms can tell you anything meaningful. Others believe that biorhythm cycles exist, but that they are not as simple and inflexible as the 23, 28, and 33 day cycles mentioned here.

Whether biorhythms are good, bad, true, false, or anything else is not our concern here. We are just presenting the idea to you as an interesting theory that you can investigate with the help of your TI-99/4A computer.

HOW TO USE IT

The program first asks for the birthday of the person whose biorhythm cycles are to be charted. You provide the month and day as you might expect. For the year, you only need to enter the last two digits if it is between 1900 and 1999. Otherwise, enter all four digits.

Next the program asks you for the start date for the biorhythm chart. Enter it in the same way. Of course, this date cannot be earlier than the birth date.

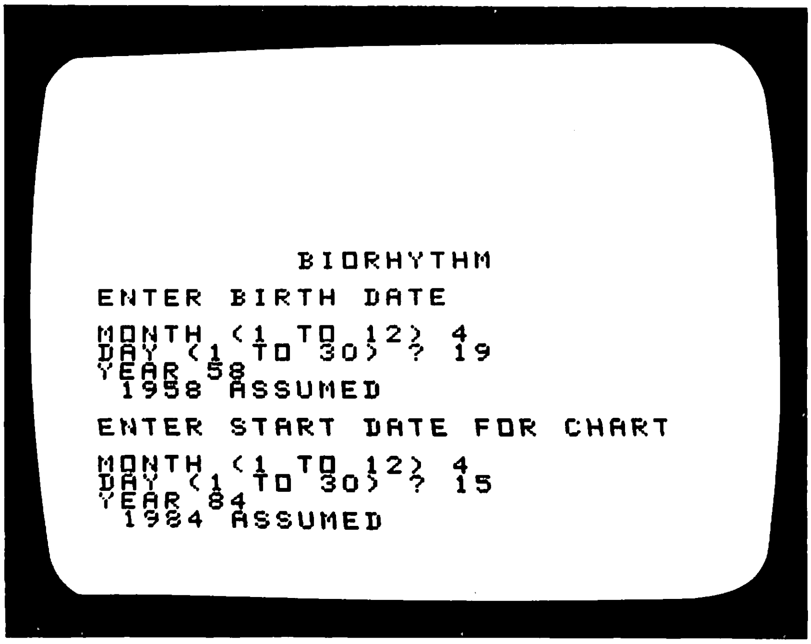
After a delay of a few seconds, the program clears the screen and begins plotting the biorhythm chart, one day at a time. The left side of the screen displays the date, while the right side displays the chart. The left half of the chart is the "down" (negative) side of each cycle. The right half is the "up" (positive) side. The center line shows the critical days when you are at a zero point (neither positive nor negative).

Each of the three curves is plotted with an identifying letter – P for physical, E for emotional, and I for intellectual. When the curves cross, an asterisk is displayed instead of either of the two (or three) letters.

Eighteen days of the chart are displayed on one screen, and then the program waits for you to press a key. If you press the E key, the current chart ends. If you press the **SPACE** key, the program clears the screen and displays the next eighteen days of the chart. If you press the FUNCTION key and CLEAR key at the same time, the program will end.

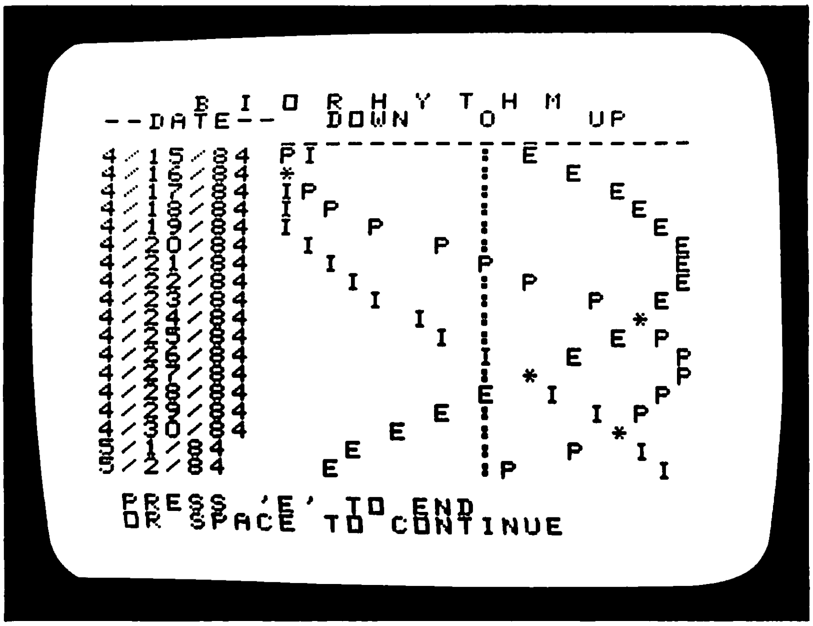
The program will allow you to enter dates from the year 100 A.D. and on. We make no guarantees about any extreme future dates, however, such as entering a year greater than 3000. We sincerely hope that these limitations do not prove to be too confining for you.

SAMPLE RUN



```
          BIORHYTHM
ENTER BIRTH DATE
MONTH (1 TO 12) 4
DAY (1 TO 30) ? 19
YEAR 58
 1958 ASSUMED
ENTER START DATE FOR CHART
MONTH (1 TO 12) 4
DAY (1 TO 30) ? 15
YEAR 84
 1984 ASSUMED
```

The operator enters his or her birth date and the date for the beginning of the chart.



```
    B I O R H Y T H M
--DATE--  DOWN   O    UP
4/15/84 PI-------:-E--------
4/16/84 *        :   E
4/17/84 IP       :     E
4/18/84 I P      :       E
4/19/84 I   P    :        E
4/20/84  I     P :         E
4/21/84   I      P         E
4/22/84    I     : P       E
4/23/84     I    :    P   E
4/24/84       I  :       *
4/25/84        I :      E P
4/26/84          I   E     P
4/27/84          : *       P
4/28/84          E  I     P
4/29/84        E :    I  P
4/30/84      E   :      *
5/1/84     E     :   P   I
5/2/84    E      :P       I
 PRESS 'E' TO END
 OR SPACE TO CONTINUE
```

The program responds with the first 18 days of the operator's biorhythm chart, then waits for a key to be pressed.

PROGRAM LISTING

```
100 REM BIORHYTHM
110 REM (C) 1984 DILITHIUM PRESS
120 L=0
130 Z=.9999
140 T=9
150 P=3.1415926535
160 CALL CLEAR
170 GOSUB 1660
180 PRINT TAB(10);"BIORHYTHM"
190 PRINT
200 PRINT "ENTER BIRTH DATE"
210 PRINT
220 INPUT "MONTH (1 TO 12) ":M
230 IF M<1 THEN 220
240 IF M>12 THEN 220
250 RESTORE
260 FOR J=1 TO M
```

```
270 READ X
280 NEXT J
290 DATA 31,29,31,30,31,30,31,31,30,31,30,31
300 A$="DAY (1 TO "&STR$(X)&") "
310 PRINT A$;
320 INPUT D
330 IF D<1 THEN 310
340 IF D>X THEN 310
350 INPUT "YEAR ":Y
360 IF Y<0 THEN 350
370 IF Y>=100 THEN 400
380 Y=Y+1900
390 PRINT Y;"ASSUMED"
400 GOSUB 1000
410 JB=JD
420 PRINT
430 PRINT "ENTER START DATE FOR CHART"
440 PRINT
450 INPUT "MONTH (1 TO 12) ":M
460 IF M<1 THEN 450
470 IF M>12 THEN 450
480 RESTORE
490 FOR J=1 TO M
500 READ X
510 NEXT J
520 A$="DAY (1 TO "&STR$(X)&") "
530 PRINT A$;
540 INPUT D
550 IF D<1 THEN 530
560 IF D>X THEN 530
570 INPUT "YEAR ":Y
580 IF Y<0 THEN 570
590 IF Y>=100 THEN 620
600 Y=Y+1900
610 PRINT Y;"ASSUMED"
620 GOSUB 1000
630 JC=JD
640 PRINT
650 IF JC>=JB THEN 710
660 PRINT "CHART DATE CAN'T BE EARLIER"
670 PRINT "THAN BIRTH DATE."
680 FOR J=1 TO 1000
690 NEXT J
```

```
700 GOTO 120
710 FOR J=1 TO 300
720 NEXT J
730 GOSUB 1100
740 N=JC-JB
750 V=23
760 GOSUB 1190
770 GOSUB 1220
780 V=28
790 GOSUB 1190
800 GOSUB 1220
810 V=33
820 GOSUB 1190
830 GOSUB 1220
840 GOSUB 1480
850 L=L+1
860 PRINT C$;TAB(9);L$
870 JC=JC+1
880 IF L<18 THEN 740
890 PRINT
900 PRINT " PRESS 'E' TO END"
910 PRINT " OR SPACE TO CONTINUE";
920 CALL KEY(0,KEY,STATUS)
930 IF STATUS=0 THEN 920
940 IF KEY<>69 THEN 970
950 CALL CLEAR
960 END
970 IF KEY<>32 THEN 920
980 L=0
990 GOTO 730
1000 W=INT((M-14)/12+Z)
1010 JD=INT(1461*(Y+4800+W)/4)
1020 B=367*(M-2-W*12)/12
1030 IF B>=0 THEN 1050
1040 B=B+Z
1050 B=INT(B)
1060 JD=JD+B
1070 B=INT(INT(3*(Y+4900+W)/100)/4)
1080 JD=JD+D-32075-B
1090 RETURN
1100 CALL CLEAR
1110 PRINT TAB(5);"B I O R H Y T H M"
```

```
1120 PRINT "--DATE--  DOWN";
1130 PRINT TAB(18);"O";TAB(23);"UP"
1140 PRINT TAB(9);
1150 FOR J=1 TO T+T+1
1160 PRINT CHR$(95);
1170 NEXT J
1180 RETURN
1190 W=INT(N/V)
1200 R=N-W*V
1210 RETURN
1220 IF V<>23 THEN 1250
1230 L$=Q$
1240 C$="P"
1250 IF V<>28 THEN 1270
1260 C$="E"
1270 IF V<>33 THEN 1290
1280 C$="I"
1290 W=R/V
1300 W=W*2*P
1310 W=T*SIN(W)
1320 W=W+T+1.5
1330 W=INT(W)
1340 A$=SEG$(L$,W,1)
1350 IF A$="P" THEN 1390
1360 IF A$="E" THEN 1390
1370 IF A$="*" THEN 1390
1380 GOTO 1400
1390 C$="*"
1400 IF W=1 THEN 1440
1410 IF W=19 THEN 1460
1420 L$=SEG$(L$,1,W-1)&C$&SEG$(L$,LEN(L$)-(19-
W)+1,(19-W))
1430 RETURN
1440 L$=C$&SEG$(L$,2,18)
1450 RETURN
1460 L$=SEG$(L$,1,18)&C$
1470 RETURN
1480 W=JC+68569
1490 R=INT(4*W/146097)
1500 W=W-INT((146097*R+3)/4)
1510 Y=INT(4000*(W+1)/1461001)
```

```
1510 Y=INT(4000*(W+1)/1461001)
```

```
1510 Y=INT(4000*(W+1)/1461001)
1520 W=W-INT(1461*Y/4)+31
1530 M=INT(80*W/2447)
1540 D=W-INT(2447*M/80)
1550 W=INT(M/11)
1560 M=M+2-12*W
1570 Y=100*(R-49)+Y+W
1580 A$=STR$(M)
1590 C$=A$&"/"
1600 A$=STR$(D)
1610 C$=C$&A$&"/"
1620 A$=STR$(Y)
1630 W=LEN(A$)-1
1640 C$=C$&SEG$(A$,W,2)
1650 RETURN
1660 Q$=""
1670 FOR J=1 TO T
1680 Q$=Q$&CHR$(32)
1690 NEXT J
1700 Q$=Q$&":"&Q$
1710 RETURN
```

EASY CHANGES

1. Want to see the number of days between any two dates? Insert this:

```
745 PRINT "DAYS=";N
746 END
```

   Then enter the earlier date as the birth date, and the later date as the start date for the chart. This will cause the program to display the difference in days and then end.

2. To alter the number of days of the chart on each screen, alter the 18 in line 880.

MAIN ROUTINES

120– 190	Initializes variables. Displays title.
200– 410	Asks for birth date and converts to Julian date format (i.e., the number of days since January 1, 4713 B.C.).

420– 640	Asks for start date for chart and converts to Julian date format.
650– 700	Checks that chart date is not sooner than birth date.
710– 720	Delays about one second before displaying chart.
730	Displays heading at top of screen.
740	Determines number of days between birth date and current chart date.
750– 830	Plots points in L$ string for each of the three cycles.
840	Converts Julian date back into month-day-year format.
850– 860	Displays one line on the chart.
870– 990	Adds one to chart date. Checks to see if screen is full, and what operator wants to do if so.
1000–1090	Subroutine to convert month, day, and year into Julian date format.
1100–1180	Subroutine to clear screen and display headings.
1190–1210	Subroutine to calculate remainder R of N/V.
1220–1470	Subroutine to plot a point in L$ based on V and R.
1480–1650	Subroutine to convert Julian date JC back into month, day, year format.
1660–1710	Subroutine to create Q$.

MAIN VARIABLES

L	Counter of number of lines shown on the screen.
T	Number of characters on one side of the chart.
P	Pi.
JB	Birth date in Julian format.
JD	Julian date calculated in subroutine.
JC	Chart start date in Julian format.
J, K	Loop and work variables.
N	Number of days between birth and current chart date.
V	Number of days in present biorhythm cycle (23, 28, or 33).
C$	String with date in month/day/year format. Also work variable.
L$	String with one line of the biorhythm chart.
Q$	Used to initialized L$ before P, E, and I plotted.
M	Month (1–12).
D	Day (1–31).
Y	Year (100 or greater).

W, B, X	Work variables.
R	Remainder of N/V (number of days into cycle).
A$	Work variable.

SUGGESTED PROJECTS

Investigate the biorhythms of some famous historical or athletic personalities. For example, are track and field athletes usually in the positive side of the physical cycle on the days that they set world records? Where was Lincoln in his emotional and intellectual cycles when he wrote "The Gettysburg Address"? Do a significant percentage of accidents befall people on critical days?

CHECKBOOK

PURPOSE

Many people consider the monthly ritual of balancing the checkbook to be an irritating and error-prone activity. Some people get confused and simply give up after the first try, while others give up the first time they cannot reconcile the bank statement with the checkbook. Fortunately, you have an advantage—your computer. This program takes you through the necessary steps to balance your checkbook, doing the arithmetic for you, of course.

HOW TO USE IT

The program starts off by giving you instructions about how to verify that the amount of each check and deposit are the same on the statement as they are in your checkbook. Sometimes the bank will make an error in reading the amount that you wrote on a check (especially if your handwriting is not too clear), and sometimes you will copy the amount incorrectly into your checkbook. While you are comparing these figures, make a check mark in your checkbook next to each check and deposit listed on the statement. A good system is to alternate the marks you use each month (maybe an "x" one month and a check mark the next) so you can easily see which checks and deposits came through on which statement.

Next, the program asks for the ending balance shown on the bank statement. You are then asked for the *check number* (not the amount) of the most recent check shown on the statement. This will generally be the highest numbered check the bank has processed, unless you like to write checks out of sequence. Your account balance after this most recent check will be reconciled with the statement balance, so that is what the program asks for next—your checkbook balance after the most recent check.

The program must compensate for any differences between what your checkbook has in it prior to the most recent check and what the statement has on it. First, if you have any deposits that are not shown on the statement before the most recent check, you must enter them. Generally, there are none, so you just enter "0" (zero).

Next you have to enter the amounts of any checks that have not yet "cleared" the bank and that are prior to the most recent check. Look in your checkbook for any checks that do not have your check mark next to them. Remember that some of these could be several months old.

Next you enter the amount of any service charges or debit memos that are on the statement, but which have not been shown in your checkbook prior to the most recent check. Typically, this is just a monthly service charge, but there might also be charges for printing new checks for you or some other adjustment that takes money away from you. Credit memos (which give money back to you) are not entered until later. Be sure to make an entry in your checkbook for any of these adjustments so that next month's statement will balance.

Finally, you are asked for any recent deposits or credit memos that were *not* entered in your checkbook prior to the most recent check, but that *are* listed on the bank statement. It is not unusual to have one or two of these, since deposits are generally processed by banks sooner than checks.

Now comes the moment of truth. The program tells you whether or not you are in balance and displays the totals. If so, pack things up until next month's statement arrives.

If not, you have to figure out what is wrong. You have seven options of what to do next which allow you to review the numbers you entered in case of a typing error. If you find an error, go back to the beginning and try again. Of course, if it is a simple error that precisely accounts for the amount by which you are out of balance, there is no need to go through the whole thing again.

If you entered everything correctly, the most likely cause of that out-of-balance condition is an arithmetic error in your checkbook. Look for errors in your addition and subtraction, with subtraction being the most likely culprit. This is especially likely if the amount of the error is a nice even number like one dollar or ten cents.

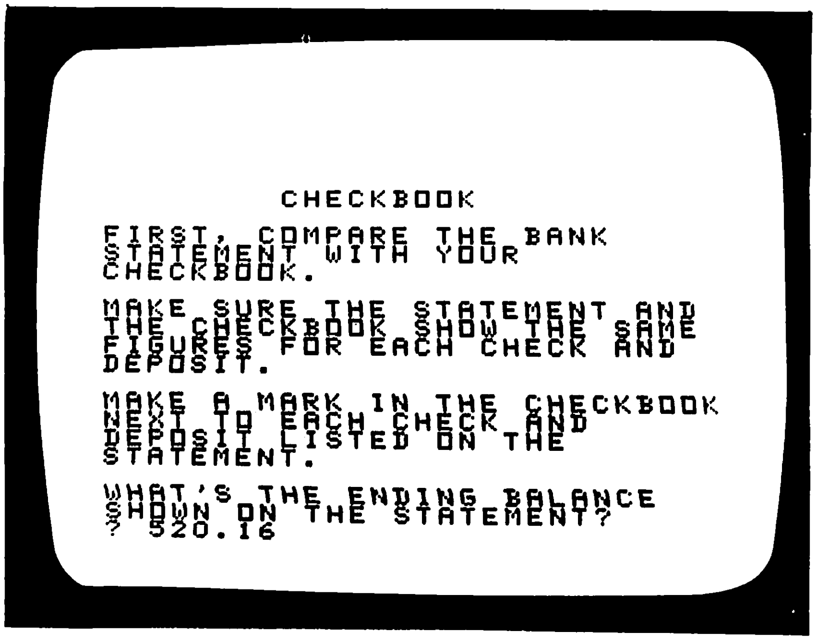
Another common error is accidentally adding the amount of a check in your checkbook instead of subtracting it. If you did this, your error will be twice the amount of the check (which makes it easy to find).

If this still does not explain the error, check to be sure you subtracted *last* month's service charge when you balanced your checkbook with the previous statement. And, of course, if you did not balance your checkbook last month, you cannot expect it to come out right this month.

The program has limitations of how many entries you can make in each category (checks outstanding, deposits outstanding, etc.), but these can be changed easily. See "Easy Changes" below.

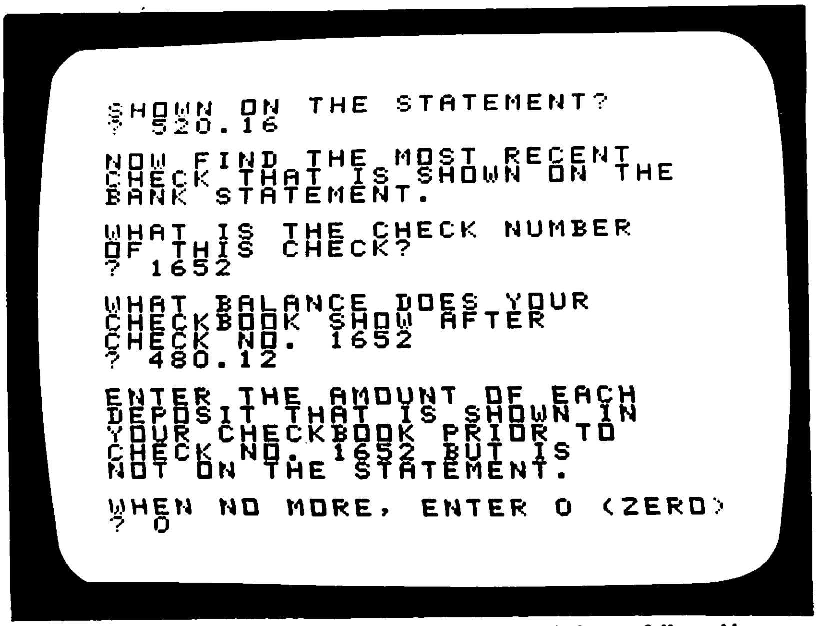
NOTE: SEE DISCLAIMER IN FRONT PART OF BOOK.

SAMPLE RUN



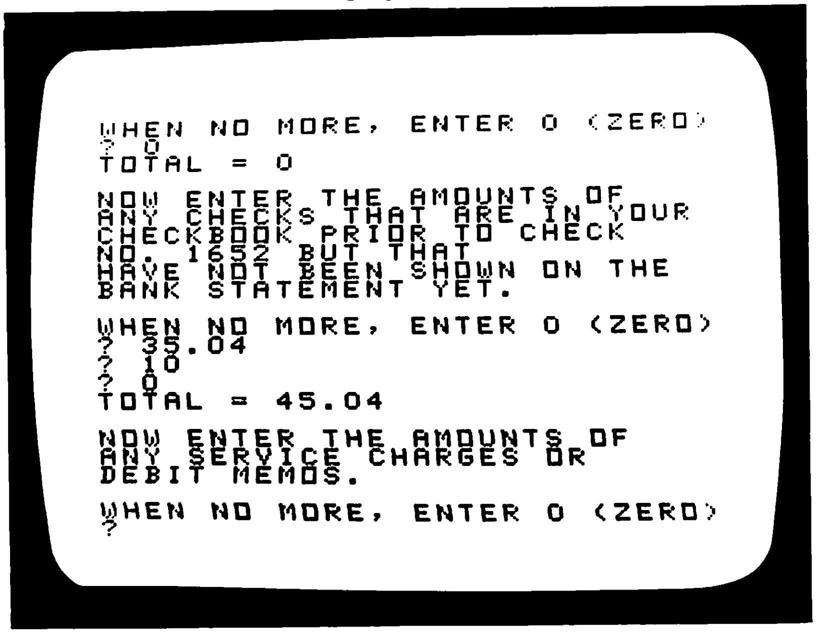
```
        CHECKBOOK
FIRST, COMPARE THE BANK
STATEMENT WITH YOUR
CHECKBOOK.
MAKE SURE THE STATEMENT AND
THE CHECKBOOK SHOW THE SAME
FIGURES FOR EACH CHECK AND
DEPOSIT.
MAKE A MARK IN THE CHECKBOOK
NEXT TO EACH CHECK AND
DEPOSIT LISTED ON THE
STATEMENT.
WHAT'S THE ENDING BALANCE
SHOWN ON THE STATEMENT?
? 520.16
```

The program displays an introduction, and the operator begins providing the necessary information.



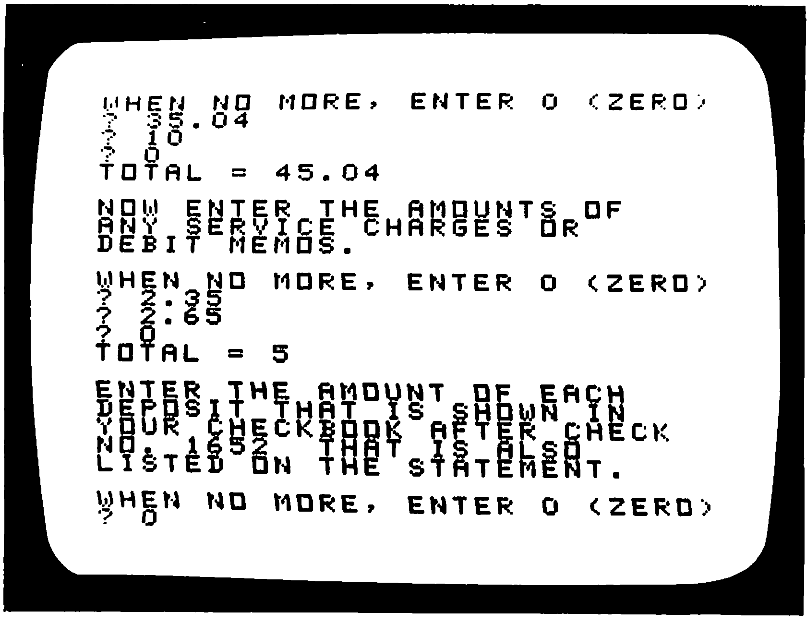
```
SHOWN ON THE STATEMENT?
? 520.16
NOW FIND THE MOST RECENT
CHECK THAT IS SHOWN ON THE
BANK STATEMENT.
WHAT IS THE CHECK NUMBER
OF THIS CHECK?
? 1652
WHAT BALANCE DOES YOUR
CHECKBOOK SHOW AFTER
CHECK NO. 1652
? 480.12
ENTER THE AMOUNT OF EACH
DEPOSIT THAT IS SHOWN IN
YOUR CHECKBOOK PRIOR TO
CHECK NO. 1652 BUT IS
NOT ON THE STATEMENT.
WHEN NO MORE, ENTER 0 (ZERO)
? 0
```

The operator continues by entering the checkbook balance, followed by zero to indicate no outstanding deposits.



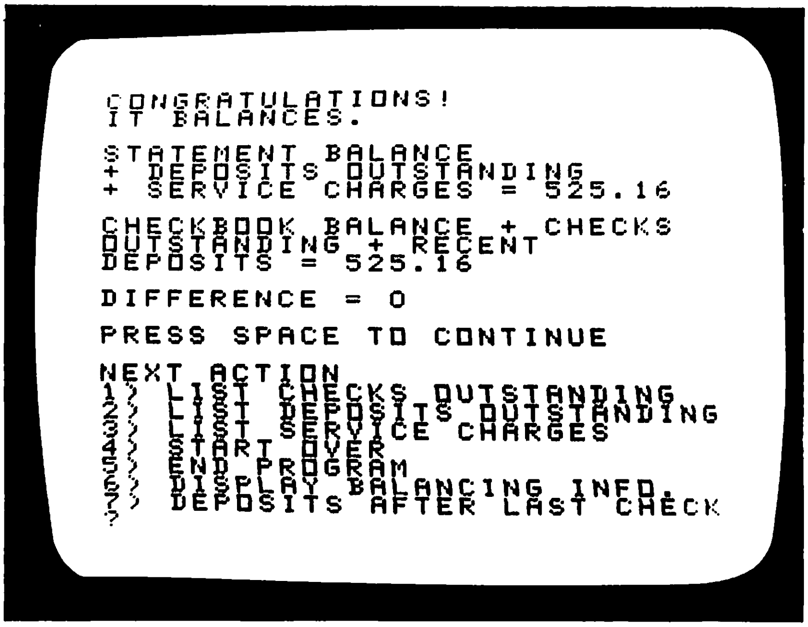
```
WHEN NO MORE, ENTER 0 (ZERO)
? 0
TOTAL = 0
NOW ENTER THE AMOUNTS OF
ANY CHECKS THAT ARE IN YOUR
CHECKBOOK PRIOR TO CHECK
NO. 1652 BUT THAT
HAVE NOT BEEN SHOWN ON THE
BANK STATEMENT YET.
WHEN NO MORE, ENTER 0 (ZERO)
? 35.04
? 10
? 0
TOTAL = 45.04
NOW ENTER THE AMOUNTS OF
ANY SERVICE CHARGES OR
DEBIT MEMOS.
WHEN NO MORE, ENTER 0 (ZERO)
?
```

The operator enters the outstanding checks and prepares to enter service charges.



```
WHEN NO MORE, ENTER 0 (ZERO)
? 35.04
? 10
? 0
TOTAL = 45.04

NOW ENTER THE AMOUNTS OF
ANY SERVICE CHARGES OR
DEBIT MEMOS.

WHEN NO MORE, ENTER 0 (ZERO)
? 2.35
? 2.65
? 0
TOTAL = 5

ENTER THE AMOUNT OF EACH
DEPOSIT THAT IS SHOWN IN
YOUR CHECKBOOK AFTER CHECK
NO. 1652 THAT IS ALSO
LISTED ON THE STATEMENT.

WHEN NO MORE, ENTER 0 (ZERO)
? 0
```

After the service charges are entered, the operator indicates no late deposits.



```
CONGRATULATIONS!
IT BALANCES.

STATEMENT BALANCE
+ DEPOSITS OUTSTANDING
+ SERVICE CHARGES = 525.16

CHECKBOOK BALANCE + CHECKS
OUTSTANDING + RECENT
DEPOSITS = 525.16

DIFFERENCE = 0

PRESS SPACE TO CONTINUE

NEXT ACTION
1) LIST CHECKS OUTSTANDING
2) LIST DEPOSITS OUTSTANDING
3) LIST SERVICE CHARGES
4) START OVER
5) END PROGRAM
6) DISPLAY BALANCING INFO.
7) DEPOSITS AFTER LAST CHECK
?
```

Finally, the program displays balancing information and, after space is pressed, the seven continuation options.

PROGRAM LISTING

```
100 REM CHECKBOOK
110 REM (C) 1984 DILITHIUM PRESS
120 MC=20
130 MD=10
140 MS=10
150 MR=10
160 DIM C(20),S(10),R(10),D(10)
170 E$="ERROR. RE-ENTER, PLEASE."
180 F$="NO MORE ROOM"
190 G$="WHEN NO MORE, ENTER 0 (ZERO)"
200 CALL CLEAR
210 TC=0
220 TD=0
230 TS=0
240 TR=0
250 NC=0
260 ND=0
270 NS=0
280 NR=0
290 PRINT TAB(9);"CHECKBOOK"
300 PRINT
310 PRINT "FIRST, COMPARE THE BANK"
320 PRINT "STATEMENT WITH YOUR"
330 PRINT "CHECKBOOK."
340 PRINT
350 PRINT "MAKE SURE THE STATEMENT AND"
360 PRINT "THE CHECKBOOK SHOW THE SAME"
370 PRINT "FIGURES FOR EACH CHECK AND"
380 PRINT "DEPOSIT."
390 PRINT
400 PRINT "MAKE A MARK IN THE CHECKBOOK"
410 PRINT "NEXT TO EACH CHECK AND"
420 PRINT "DEPOSIT LISTED ON THE"
430 PRINT "STATEMENT."
440 PRINT
450 PRINT "WHAT'S THE ENDING BALANCE"
460 PRINT "SHOWN ON THE STATEMENT?"
470 INPUT SB
480 PRINT
490 PRINT "NOW FIND THE MOST RECENT"
500 PRINT "CHECK THAT IS SHOWN ON THE"
```

```
510 PRINT "BANK STATEMENT."
520 PRINT
530 PRINT "WHAT IS THE CHECK NUMBER"
540 PRINT "OF THIS CHECK?"
550 INPUT LC
560 PRINT
570 PRINT "WHAT BALANCE DOES YOUR"
580 PRINT "CHECKBOOK SHOW AFTER"
590 PRINT "CHECK NO.";LC
600 INPUT CB
610 PRINT
620 PRINT "ENTER THE AMOUNT OF EACH"
630 PRINT "DEPOSIT THAT IS SHOWN IN"
640 PRINT "YOUR CHECKBOOK PRIOR TO"
650 PRINT "CHECK NO.";LC;"BUT IS"
660 PRINT "NOT ON THE STATEMENT."
670 PRINT
680 PRINT G$
690 INPUT X
700 IF X>0 THEN 740
710 IF X=0 THEN 790
720 PRINT E$
730 GOTO 690
740 ND=ND+1
750 D(ND)=X
760 TD=TD+D(ND)
770 IF ND<MD THEN 690
780 PRINT F$
790 PRINT "TOTAL =";TD
800 PRINT
810 PRINT "NOW ENTER THE AMOUNTS OF"
820 PRINT "ANY CHECKS THAT ARE IN YOUR"
830 PRINT "CHECKBOOK PRIOR TO CHECK"
840 PRINT "NO.";LC;"BUT THAT"
850 PRINT "HAVE NOT BEEN SHOWN ON THE"
860 PRINT "BANK STATEMENT YET."
870 PRINT
880 PRINT G$
890 INPUT X
900 IF X>0 THEN 940
910 IF X=0 THEN 990
920 PRINT E$
```

```
930 GOTO 890
940 NC=NC+1
950 C(NC)=X
960 TC=TC+C(NC)
970 IF NC<MC THEN 890
980 PRINT F$
990 PRINT "TOTAL =";TC
1000 PRINT
1010 PRINT "NOW ENTER THE AMOUNTS OF"
1020 PRINT "ANY SERVICE CHARGES OR"
1030 PRINT "DEBIT MEMOS."
1040 PRINT
1050 PRINT G$
1060 INPUT X
1070 IF X>0 THEN 1110
1080 IF X=0 THEN 1160
1090 PRINT E$
1100 GOTO 1060
1110 NS=NS+1
1120 S(NS)=X
1130 TS=TS+S(NS)
1140 IF NS<MS THEN 1060
1150 PRINT F$
1160 PRINT "TOTAL =";TS
1170 PRINT
1180 PRINT "ENTER THE AMOUNT OF EACH"
1190 PRINT "DEPOSIT THAT IS SHOWN IN"
1200 PRINT "YOUR CHECKBOOK AFTER CHECK"
1210 PRINT "NO.";LC;" THAT IS ALSO"
1220 PRINT "LISTED ON THE STATEMENT."
1230 PRINT
1240 PRINT G$
1250 INPUT X
1260 IF X>0 THEN 1300
1270 IF X=0 THEN 1350
1280 PRINT E$
1290 GOTO 1250
1300 NR=NR+1
1310 R(NR)=X
1320 TR=TR+R(NR)
```

```
1330 IF NR<MR THEN 1250
1340 PRINT F$
1350 PRINT "TOTAL =";TR
1360 PRINT
1370 W=SB+TD+TS-CB-TC-TR
1380 W=ABS(W)
1390 IF W>.001 THEN 1410
1400 W=0
1410 IF W<>0 THEN 1450
1420 PRINT "CONGRATULATIONS!"
1430 PRINT "IT BALANCES."
1440 GOTO 1460
1450 PRINT "SORRY, IT'S OUT OF BALANCE."
1460 PRINT
1470 PRINT "STATEMENT BALANCE"
1480 PRINT "+ DEPOSITS OUTSTANDING"
1490 PRINT "+ SERVICE CHARGES =";SB+TD+TS
1500 PRINT
1510 PRINT "CHECKBOOK BALANCE + CHECKS"
1520 PRINT "OUTSTANDING + RECENT"
1530 PRINT "DEPOSITS =";CB+TC+TR
1540 PRINT
1550 PRINT "DIFFERENCE =";W
1560 PRINT
1570 PRINT "PRESS SPACE TO CONTINUE"
1580 CALL KEY(0,KEY,STATUS)
1590 IF STATUS=0 THEN 1580
1600 IF KEY<>32 THEN 1580
1610 PRINT
1620 PRINT "NEXT ACTION"
1630 PRINT "1) LIST CHECKS OUTSTANDING"
1640 PRINT "2) LIST DEPOSITS OUTSTANDING"
1650 PRINT "3) LIST SERVICE CHARGES"
1660 PRINT "4) START OVER"
1670 PRINT "5) END PROGRAM"
1680 PRINT "6) DISPLAY BALANCING INFO."
1690 PRINT "7) DEPOSITS AFTER LAST CHECK"
1700 INPUT X
1710 X=INT(X)
1720 IF X<1 THEN 1580
```

```
1730 IF X>7 THEN 1580
1740 PRINT
1750 ON X GOTO 1760,1810,1860,1910,1940,2010,1
960
1760 PRINT "CHECKS OUTSTANDING"
1770 FOR J=1 TO NC
1780 PRINT C(J)
1790 NEXT J
1800 GOTO 1560
1810 PRINT "DEPOSITS OUTSTANDING"
1820 FOR J=1 TO ND
1830 PRINT D(J)
1840 NEXT J
1850 GOTO 1560
1860 PRINT "SERVICE CHARGES"
1870 FOR J=1 TO NS
1880 PRINT S(J)
1890 NEXT J
1900 GOTO 1560
1910 CALL CLEAR
1920 PRINT
1930 GOTO 200
1940 CALL CLEAR
1950 END
1960 PRINT "RECENT DEPOSITS"
1970 FOR J=1 TO NR
1980 PRINT R(J)
1990 NEXT J
2000 GOTO 1560
2010 PRINT
2020 GOTO 1460
```

EASY CHANGES

Change the limitations of how many entries you can make in each category. Lines 120 through 160 establish these limits. If you have more than 20 checks outstanding at some time, change the value of MC to 50 in line 120 and C(20) to C(50) in line 160, for example. The other three variables can also be changed if you anticipate needing more than 10 entries. They are: the number of deposits outstanding (MD(10)), the number of service charges and debit memos (MS(10)), and the number of recent deposits and credit memos (MR(10)). You may use more than the 16K that is available if you make the values too large.

MAIN ROUTINES

120– 440	Initializes variables and displays first instructions.
450– 470	Gets balance from statement.
480– 550	Gets most recent check number.
560– 600	Gets checkbook balance after most recent check number.
610– 790	Gets outstanding deposits.
800– 990	Gets outstanding checks.
1000–1160	Gets service charges and debit memos.
1170–1350	Gets recent deposits and credit memos.
1360–1600	Performs and displays balancing calculation. Waits for a key to be pressed to continue.
1610–1750	Shows seven actions to choose from.
1760–1800	Displays checks outstanding.
1810–1850	Displays deposits outstanding.
1860–1900	Displays service charges.
1910–1930	Starts program over.
1940–1950	Ends program.
1960–2000	Displays recent deposits.
2010–2020	Goes back to display balancing information.

MAIN VARIABLES

MC	Maximum number of checks outstanding.
MD	Maximum number of deposits outstanding.
MS	Maximum number of service charges, debit memos.
MR	Maximum number of recent deposits, credit memos.
C	Array for checks outstanding.
S	Array for service charges.
R	Array for recent deposits.
D	Array for deposits outstanding.
TC	Total of checks outstanding.
TD	Total of deposits outstanding.
TS	Total of service charges and debit memos.
TR	Total of recent deposits and credit memos.
NC	Number of checks outstanding actually entered.
ND	Number of deposits outstanding.
NS	Number of service charges and debit memos.

NR	Number of recent deposits and credit memos.
E$	Error message.
F$	Message indicating an array is full.
G$	Message saying how to indicate no more data.
SB	Statement balance.
LC	Number of last check on the statement.
CB	Checkbook balance after last check on statement.
X	Reply from operator.
W	Amount by which checkbook is out of balance.
KEY	Key pressed by operator to continue program.
STATUS	Status indicator showing if key was pressed.
J	Loop variable.

SUGGESTED PROJECTS

1. Add more informative messages and a more complete introduction to make the program a tutorial for someone who has never balanced a checkbook before.
2. Allow the operator to modify any entries that have been discovered to be in error. This could be done by adding another option to the "NEXT ACTION" list, which would then ask the operator which category to change. This would allow the operator to correct an error without having to re-enter everything from the beginning.
3. If the checkbook is out of balance, have the program do an analysis (as suggested in the "How To Use It" section) and suggest the most likely errors that might have caused the condition.
4. Allow the operator to find arithmetic errors in the checkbook. Ask for the starting balance, then ask for each check or deposit amount. Add or subtract, depending on which type the operator indicates. Display the new balance after each entry so the operator can compare with the checkbook entry.

DECIDE

PURPOSE

"Decisions, decisions!" How many times have you uttered this lament when confronted by a difficult choice? Wouldn't a trusty advisor be helpful on such occasions? Well, you now have one—your TI-99/4A computer, of course.

This program can help you make decisions involving the selection of one alternative from several choices. It works by prying relevant information from you and then organizing it in a meaningful, quantitative manner. Your best choice will be indicated and all of the possibilities given a relative rating.

You can use the program for a wide variety of decisions. It can help with things like choosing the best stereo system, saying yes or no to a job or business offer, or selecting the best course of action for the future. Everything is personalized to your individual decision.

HOW TO USE IT

The first thing the program does is ask you to categorize the decision at hand into one of these three categories:

1) Choosing an item (or thing),
2) Choosing a course of action, or
3) Making a yes or no decision.

You simply press **1**, **2**, or **3** and press **ENTER** to indicate which type of decision is facing you. If you are choosing an item, you will be asked what kind of item it is.

If the decision is either of the first two types, you must next enter a list of all the possibilities under consideration. A question mark will prompt you for each one. When the list is complete, type "END" in response to the last question mark. You must, of course, enter at least two possibilities. (We hope you don't have trouble making decisions from only one possibility!) After the list is finished, it will be re-displayed so that you can verify that it is correct. If not, you must re-enter it.

Now you must think of the different factors that are important to you in making your decision. For example, location, cost, and quality of education might govern the decision of which college to attend. For a refrigerator purchase, the factors might be things like price, size, reliability, and warranty. In any case, you will be prompted for your list with a succession of question marks. Each factor is to be entered one at a time with the word "END" used to terminate the list. When complete, the list will be re-displayed. You must now decide which single factor is the most important and input its number. (You can enter 0 if you wish to change the list of factors.)

The program now asks you to rate the importance of each of the other factors relative to the most important one. This is done by first assigning a value of 10 to the main factor. Then you must assign a value from 0–10 to each of the other factors. These numbers reflect your assessment of each factor's relative importance as compared to the main one. A value of 10 means it is just as important; lesser values indicate how much less importance you place on it.

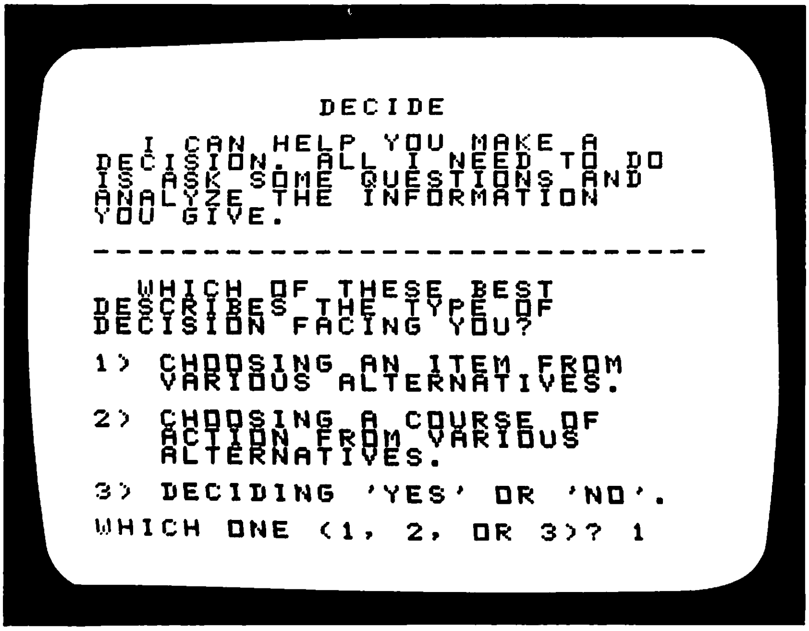
Now you must rate the decision possibilities with respect to each of the importance factors. Each importance factor will be treated separately. Considering *only* that importance factor, you must rate how each decision possibility stacks up. The program first assigns a value of 10 to one of the decision possibilities. Then you must assign a relative number (lower, higher, or equal to 10) to each of the other decision possibilities.

An example might alleviate possible confusion here. Suppose you are trying to decide whether to get a dog, cat, or canary for a pet. Affection is one of your importance factors. The program assigns a value of 10 to the cat. Considering *only* affection, you might assign a value of 20 to the dog and 6.5 to the canary. This means *you* consider a dog twice as affectionate as a cat but a canary only about two-thirds as affectionate as a cat. (No slighting of bird lovers is intended here, of course. Your actual ratings may be entirely different.)

Armed with all this information, the program will now determine which choice seems best for you. The various possibilities are listed in order of ranking. Alongside each one is a relative rating with the best choice being normalized to a value of 100.

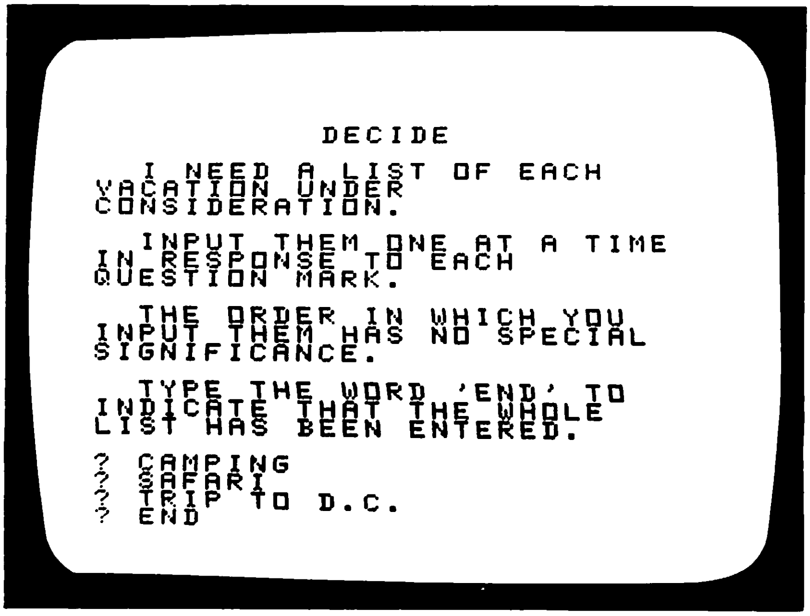
Of course, DECIDE should not be used as a substitute for good, clear thinking. However, it can often provide valuable insights. You might find one alternative coming out surprisingly low or high. A trend may become obvious when the program is re-run with improved data. At least, it may help you think about decisions systematically and honestly.

SAMPLE RUN



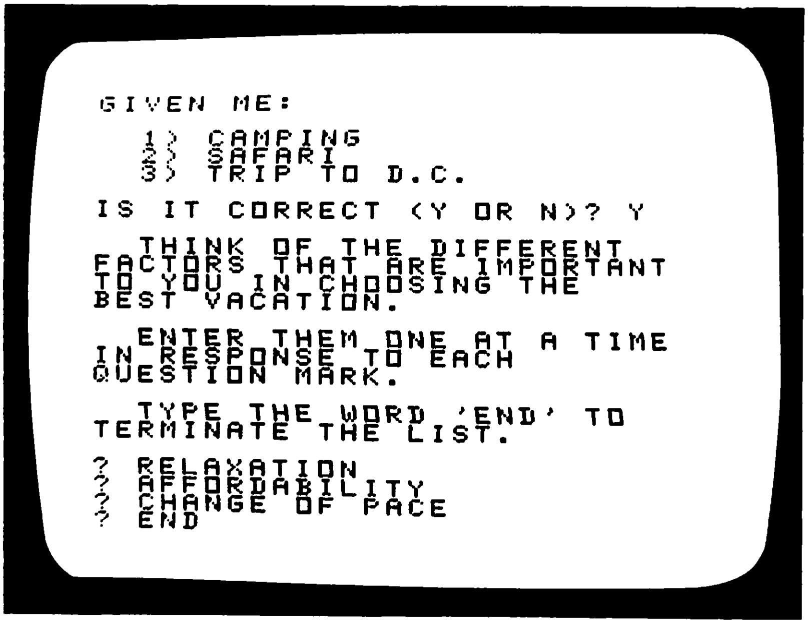
```
                DECIDE
   I CAN HELP YOU MAKE A
DECISION. ALL I NEED TO DO
IS ASK SOME QUESTIONS AND
ANALYZE THE INFORMATION
YOU GIVE.
------------------------------
   WHICH OF THESE BEST
DESCRIBES THE TYPE OF
DECISION FACING YOU?
1) CHOOSING AN ITEM FROM
   VARIOUS ALTERNATIVES.
2) CHOOSING A COURSE OF
   ACTION FROM VARIOUS
   ALTERNATIVES.
3) DECIDING 'YES' OR 'NO'.
WHICH ONE (1, 2, OR 3)? 1
```

After displaying an introduction, the program asks what type of decision is to be analyzed. The operator indicates choice #1.



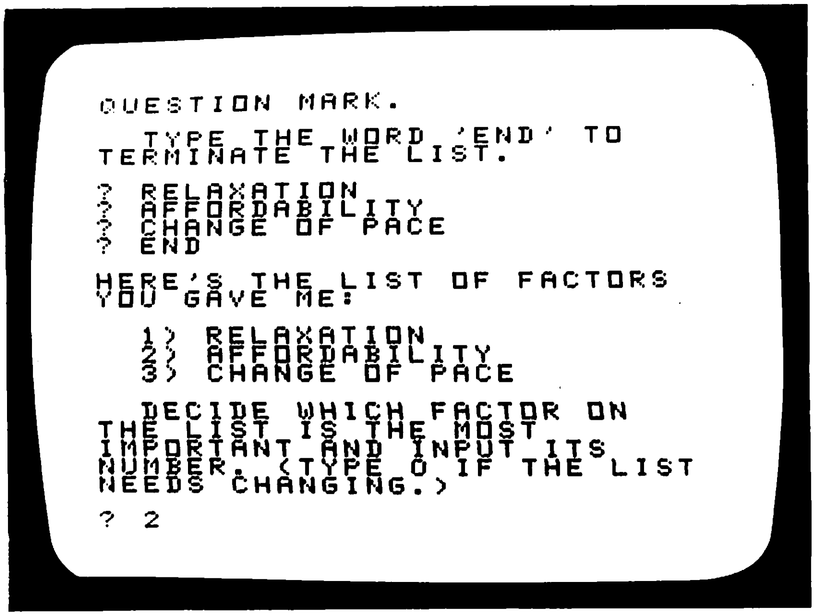
```
                DECIDE
  I NEED A LIST OF EACH
VACATION UNDER
CONSIDERATION.
  INPUT THEM ONE AT A TIME
IN RESPONSE TO EACH
QUESTION MARK.
  THE ORDER IN WHICH YOU
INPUT THEM HAS NO SPECIAL
SIGNIFICANCE.
  TYPE THE WORD 'END' TO
INDICATE THAT THE WHOLE
LIST HAS BEEN ENTERED.
? CAMPING
? SAFARI
? TRIP TO D.C.
? END
```

After indicating he will be choosing a "VACATION," the operator lists the vacations under consideration.



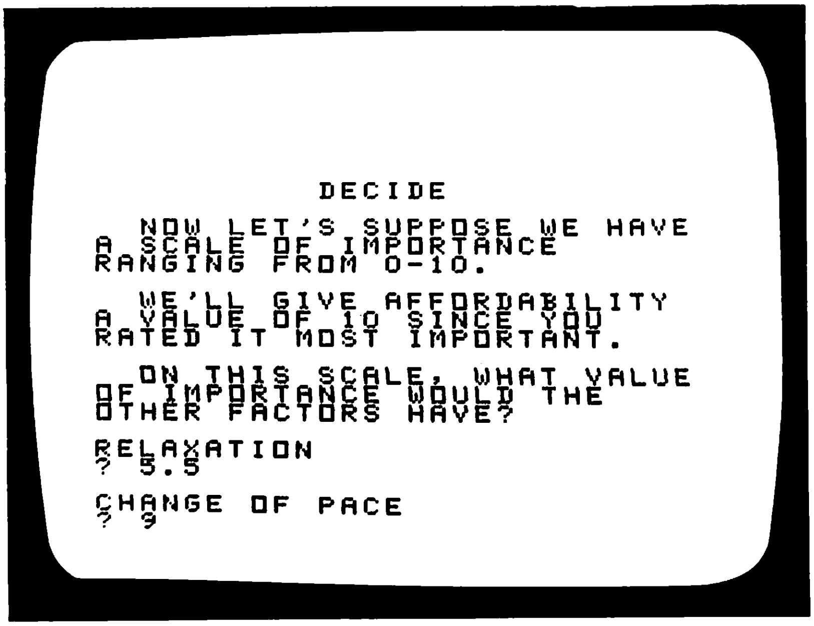
```
GIVEN ME:
  1) CAMPING
  2) SAFARI
  3) TRIP TO D.C.
IS IT CORRECT (Y OR N)? Y
  THINK OF THE DIFFERENT
FACTORS THAT ARE IMPORTANT
TO YOU IN CHOOSING THE
BEST VACATION.
  ENTER THEM ONE AT A TIME
IN RESPONSE TO EACH
QUESTION MARK.
  TYPE THE WORD 'END' TO
TERMINATE THE LIST.
? RELAXATION
? AFFORDABILITY
? CHANGE OF PACE
? END
```

The program re-displays the list of vacations and the operator verifies its correctness. He then enters the importance factors as requested.



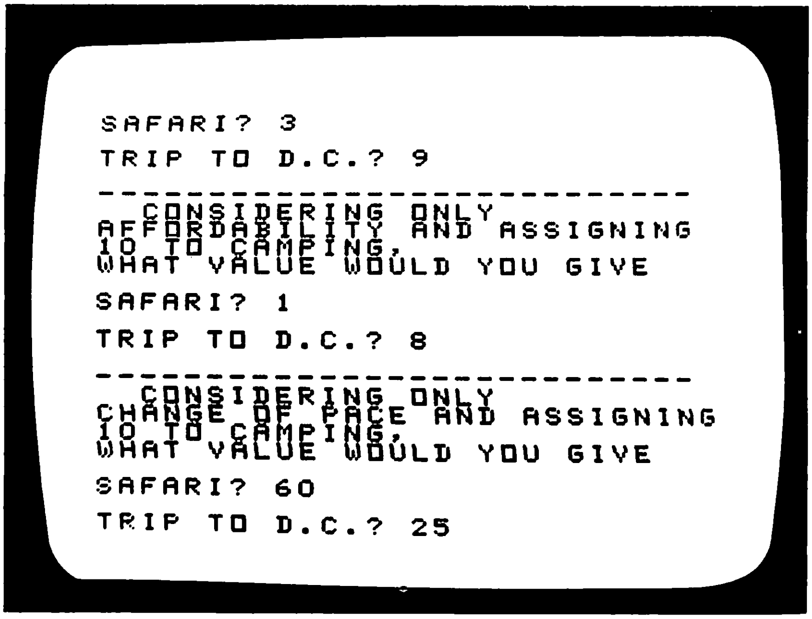
```
QUESTION MARK.
  TYPE THE WORD 'END' TO
TERMINATE THE LIST.
? RELAXATION
? AFFORDABILITY
? CHANGE OF PACE
? END
HERE'S THE LIST OF FACTORS
YOU GAVE ME:
  1) RELAXATION
  2) AFFORDABILITY
  3) CHANGE OF PACE
  DECIDE WHICH FACTOR ON
THE LIST IS THE MOST
IMPORTANT AND INPUT ITS
NUMBER. (TYPE 0 IF THE LIST
NEEDS CHANGING.)
? 2
```

The operator chooses "AFFORDABILITY" as the primary importance factor.



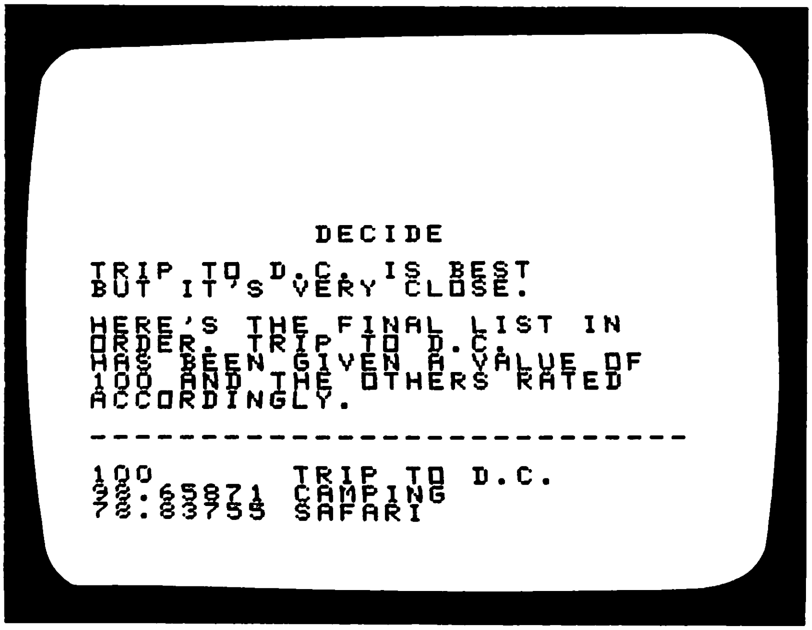
```
          DECIDE
  NOW LET'S SUPPOSE WE HAVE
A SCALE OF IMPORTANCE
RANGING FROM 0-10.
  WE'LL GIVE AFFORDABILITY
A VALUE OF 10 SINCE YOU
RATED IT MOST IMPORTANT.
  ON THIS SCALE, WHAT VALUE
OF IMPORTANCE WOULD THE
OTHER FACTORS HAVE?
RELAXATION
? 5.5
CHANGE OF PACE
? 9
```

The operator selects numerical values for the other two factors.



```
SAFARI? 3
TRIP TO D.C.? 9
----------------------------
  CONSIDERING ONLY
AFFORDABILITY AND ASSIGNING
10 TO CAMPING,
WHAT VALUE WOULD YOU GIVE
SAFARI? 1
TRIP TO D.C.? 8
----------------------------
  CONSIDERING ONLY
CHANGE OF PACE AND ASSIGNING
10 TO CAMPING,
WHAT VALUE WOULD YOU GIVE
SAFARI? 60
TRIP TO D.C.? 25
```

After being given instructions on how to rate each vacation with respect to each factor, and after hitting a key to continue, the operator provides the requested data.



```
           DECIDE
TRIP TO D.C. IS BEST
BUT IT'S VERY CLOSE.
HERE'S THE FINAL LIST IN
ORDER. TRIP TO D.C.
HAS BEEN GIVEN A VALUE OF
100 AND THE OTHERS RATED
ACCORDINGLY.
----------------------------
100      TRIP TO D.C.
98.65871 CAMPING
78.83755 SAFARI
```

The program displays the choices in order of desirability.

PROGRAM LISTING

```
100 REM DECIDE
110 REM (C) 1984 DILITHIUM PRESS
120 DIM L$(10),F$(10),V(10)
130 DIM C(10,10),D(10),Z(10)
140 E$="END"
150 B$=CHR$(32)
160 GOSUB 3020
170 PRINT TAB(3);"I CAN HELP YOU MAKE A"
180 PRINT "DECISION. ALL I NEED TO DO"
190 PRINT "IS ASK SOME QUESTIONS AND"
200 PRINT "ANALYZE THE INFORMATION"
210 PRINT "YOU GIVE."
220 PRINT
230 CALL HCHAR(24,3,45,28)
240 PRINT
250 PRINT
260 PRINT TAB(3);"WHICH OF THESE BEST"
270 PRINT "DESCRIBES THE TYPE OF"
280 PRINT "DECISION FACING YOU?"
290 PRINT
300 PRINT "1) CHOOSING AN ITEM FROM"
310 PRINT TAB(4);"VARIOUS ALTERNATIVES."
320 PRINT
330 PRINT "2) CHOOSING A COURSE OF"
340 PRINT TAB(4);"ACTION FROM VARIOUS"
350 PRINT TAB(4);"ALTERNATIVES."
360 PRINT
370 PRINT "3) DECIDING 'YES' OR 'NO'."
380 PRINT
390 INPUT "WHICH ONE (1, 2, OR 3)? ":T
400 T=INT(T)
410 IF T<1 THEN 380
420 IF T>3 THEN 380
430 ON T GOTO 440,480,500
440 PRINT
450 PRINT "WHAT TYPE OF ITEM MUST YOU"
460 INPUT "DECIDE UPON? ":T$
470 GOTO 550
480 T$="COURSE OF ACTION"
490 GOTO 550
500 T$="'YES' OR 'NO'"
```

```
510 NI=2
520 L$(1)="DECIDING YES"
530 L$(2)="DECIDING NO"
540 GOTO 1030
550 GOSUB 3020
560 NI=0
570 PRINT TAB(3);"I NEED A LIST OF EACH"
580 PRINT T$;B$;"UNDER"
590 PRINT "CONSIDERATION."
600 PRINT
610 PRINT TAB(3);"INPUT THEM ONE AT A TIME"
620 PRINT "IN RESPONSE TO EACH"
630 PRINT "QUESTION MARK."
640 PRINT
650 PRINT TAB(3);"THE ORDER IN WHICH YOU"
660 PRINT "INPUT THEM HAS NO SPECIAL"
670 PRINT "SIGNIFICANCE."
680 PRINT
690 PRINT TAB(3);"TYPE THE WORD '";E$;"' TO"
700 PRINT "INDICATE THAT THE WHOLE"
710 PRINT "LIST HAS BEEN ENTERED."
720 PRINT
730 NI=NI+1
740 INPUT L$(NI)
750 IF L$(NI)<>E$ THEN 730
760 NI=NI-1
770 IF NI>=2 THEN 860
780 PRINT
790 PRINT TAB(3);"ERROR -- YOU MUST HAVE"
800 PRINT "AT LEAST TWO CHOICES!"
810 CALL SOUND(300,200,3)
820 PRINT
830 PRINT TAB(3);"TRY AGAIN!"
840 GOSUB 3060
850 GOTO 550
860 PRINT
870 PRINT "O.K. HERE'S THE LIST YOU'VE"
880 PRINT "GIVEN ME:"
890 PRINT
900 FOR J=1 TO NI
910 PRINT TAB(3);STR$(J);")";B$;L$(J)
920 NEXT J
```

```
930 PRINT
940 INPUT "IS IT CORRECT (Y OR N)? ":R$
950 R$=SEG$(R$,1,1)
960 IF R$="Y" THEN 1030
970 IF R$="N" THEN 990
980 GOTO 930
990 PRINT
1000 PRINT "THE LIST MUST BE RE-ENTERED."
1010 GOSUB 3060
1020 GOTO 550
1030 PRINT
1040 PRINT TAB(3);"THINK OF THE DIFFERENT"
1050 PRINT "FACTORS THAT ARE IMPORTANT"
1060 IF T<3 THEN 1100
1070 PRINT "TO YOU IN DECIDING 'YES'"
1080 PRINT "OR 'NO'."
1090 GOTO 1120
1100 PRINT "TO YOU IN CHOOSING THE"
1110 PRINT "BEST";B$;T$;"."
1120 PRINT
1130 PRINT TAB(3);"ENTER THEM ONE AT A TIME"
1140 PRINT "IN RESPONSE TO EACH"
1150 PRINT "QUESTION MARK."
1160 PRINT
1170 PRINT TAB(3);"TYPE THE WORD '";E$;"' TO"
1180 PRINT "TERMINATE THE LIST."
1190 PRINT
1200 NF=0
1210 NF=NF+1
1220 INPUT F$(NF)
1230 IF F$(NF)<>E$ THEN 1210
1240 NF=NF-1
1250 IF NF>=1 THEN 1300
1260 PRINT
1270 PRINT "YOU MUST HAVE AT LEAST ONE."
1280 CALL SOUND(300,200,3)
1290 GOTO 1030
1300 PRINT
1310 PRINT "HERE'S THE LIST OF FACTORS"
1320 PRINT "YOU GAVE ME:"
1330 PRINT
1340 FOR J=1 TO NF
```

```
1350 PRINT TAB(3);STR$(J);")";B$;F$(J)
1360 NEXT J
1370 PRINT
1380 PRINT TAB(3);"DECIDE WHICH FACTOR ON"
1390 PRINT "THE LIST IS THE MOST"
1400 PRINT "IMPORTANT AND INPUT ITS"
1410 PRINT "NUMBER. (TYPE 0 IF THE LIST"
1420 PRINT "NEEDS CHANGING.)"
1430 PRINT
1440 INPUT A
1450 A=INT(A)
1460 IF A=0 THEN 1030
1470 IF A>NF THEN 1300
1480 GOSUB 3020
1490 IF NF=1 THEN 1750
1500 PRINT TAB(3);"NOW LET'S SUPPOSE WE HAVE"
1510 PRINT "A SCALE OF IMPORTANCE"
1520 PRINT "RANGING FROM 0-10."
1530 PRINT
1540 PRINT TAB(3);"WE'LL GIVE";B$;F$(A)
1550 PRINT "A VALUE OF 10 SINCE YOU"
1560 PRINT "RATED IT MOST IMPORTANT."
1570 PRINT
1580 PRINT TAB(3);"ON THIS SCALE, WHAT VALUE"
1590 PRINT "OF IMPORTANCE WOULD THE"
1600 PRINT "OTHER FACTORS HAVE?"
1610 FOR J=1 TO NF
1620 Q=A
1630 IF J=Q THEN 1740
1640 PRINT
1650 PRINT F$(J)
1660 INPUT V(J)
1670 IF V(J)<0 THEN 1700
1680 IF V(J)>10 THEN 1700
1690 GOTO 1740
1700 PRINT
1710 PRINT "IMPOSSIBLE VALUE-TRY AGAIN"
1720 CALL SOUND(300,200,3)
1730 GOTO 1640
1740 NEXT J
1750 V(A)=10
1760 Q=0
```

```
1770 FOR J=1 TO NF
1780 Q=Q+V(J)
1790 NEXT J
1800 FOR J=1 TO NF
1810 V(J)=V(J)/Q
1820 NEXT J
1830 PRINT
1840 IF T=3 THEN 1870
1850 PRINT TAB(3);"EACH";B$;T$
1860 GOTO 1880
1870 PRINT "DECIDING YES OR DECIDING NO"
1880 PRINT "MUST NOW BE COMPARED WITH"
1890 PRINT "RESPECT TO EACH IMPORTANCE"
1900 PRINT "FACTOR."
1910 PRINT TAB(3);"WE'LL CONSIDER EACH FACTOR"
1920 PRINT "SEPARATELY AND THEN RATE"
1930 IF T=3 THEN 1960
1940 PRINT "EACH";B$;T$;B$;"IN TERMS"
1950 GOTO 1970
1960 PRINT "DECIDING YES OR NO IN TERMS"
1970 PRINT "OF THAT FACTOR ONLY."
1980 PRINT TAB(3);"LET'S GIVE";B$;L$(1)
1990 PRINT "A VALUE OF 10 ON EACH SCALE."
2000 IF T=3 THEN 2030
2010 PRINT TAB(3);"EVERY OTHER";B$;T$
2020 GOTO 2040
2030 PRINT TAB(3);"THEN DECIDING NO"
2040 PRINT "WILL BE ASSIGNED A VALUE"
2050 PRINT "HIGHER OR LOWER THAN 10."
2060 PRINT "THIS VALUE DEPENDS ON HOW"
2070 PRINT "MUCH YOU THINK IT IS BETTER"
2080 PRINT "OR WORSE THAN"
2090 PRINT L$(1);"."
2100 PRINT
2110 PRINT "HIT ANY KEY TO CONTINUE."
2120 CALL KEY(0,KK,SS)
2130 IF SS=0 THEN 2120
2140 PRINT
2150 FOR J=1 TO NF
2160 CALL HCHAR(24,3,45,27)
2170 PRINT
2180 PRINT TAB(3);"CONSIDERING ONLY"
```

```
2190 PRINT F$(J);B$;"AND ASSIGNING"
2200 PRINT "10 TO";B$;L$(1);","
2210 PRINT "WHAT VALUE WOULD YOU GIVE"
2220 PRINT
2230 FOR K=2 TO NI
2240 PRINT L$(K);
2250 INPUT C(K,J)
2260 PRINT
2270 IF C(K,J)>=0 THEN 2320
2280 PRINT "NEGATIVE VALUES ILLEGAL!"
2290 CALL SOUND(300,200,3)
2300 PRINT
2310 GOTO 2240
2320 NEXT K
2330 C(1,J)=10
2340 NEXT J
2350 FOR J=1 TO NF
2360 Q=0
2370 FOR K=1 TO NI
2380 Q=Q+C(K
2390 NEXT K
2400 FOR K=1 TO NI
2410 C(K,J)=C(K,J)/Q
2420 NEXT K
2430 NEXT J
2440 FOR K=1 TO NI
2450 D(K)=0
2460 FOR J=1 TO NF
2470 D(K)=D(K)+C(K,J)*V(J)
2480 NEXT J
2490 NEXT K
2500 MX=0
2510 FOR K=1 TO NI
2520 IF D(K)<=MX THEN 2540
2530 MX=D(K)
2540 NEXT K
2550 FOR K=1 TO NI
2560 D(K)=D(K)*100/MX
2570 NEXT K
2580 FOR K=1 TO NI
2590 Z(K)=K
2600 NEXT K
```

```
2610 NM=NI-1
2620 FOR K=1 TO NI
2630 FOR J=1 TO NM
2640 N1=Z(J)
2650 N2=Z(J+1)
2660 IF D(N1)>D(N2)THEN 2690
2670 Z(J+1)=N1
2680 Z(J)=N2
2690 NEXT J
2700 NEXT K
2710 J1=Z(1)
2720 J2=Z(2)
2730 DF=D(J1)-D(J2)
2740 GOSUB 3020
2750 PRINT L$(J1);B$;"IS BEST"
2760 IF DF<5 THEN 2810
2770 IF DF<10 THEN 2830
2780 IF DF<20 THEN 2850
2790 PRINT "QUITE DECISIVELY."
2800 GOTO 2860
2810 PRINT "BUT IT'S VERY CLOSE."
2820 GOTO 2860
2830 PRINT "BUT IT'S FAIRLY CLOSE."
2840 GOTO 2860
2850 PRINT "BY A FAIR AMOUNT."
2860 PRINT
2870 PRINT "HERE'S THE FINAL LIST IN"
2880 PRINT "ORDER.";B$;L$(J1)
2890 PRINT "HAS BEEN GIVEN A VALUE OF"
2900 PRINT "100 AND THE OTHERS RATED"
2910 PRINT "ACCORDINGLY."
2920 PRINT
2930 CALL HCHAR(24,3,45,27)
2940 PRINT
2950 PRINT
2960 FOR J=1 TO NI
2970 Q=Z(J)
2980 PRINT SEG$(STR$(D(Q)),1,8);TAB(10);L$(Q)
2990 NEXT J
3000 GOSUB 3060
3010 END
3020 CALL CLEAR
```

```
3030 PRINT TAB(11);"DECIDE"
3040 PRINT
3050 RETURN
3060 FOR J=1 TO 2000
3070 NEXT J
3080 RETURN
```

EASY CHANGES

1. The word "END" is used to flag the termination of various input lists. If you wish to use something else (because of conflicts with items on the list), change the definition of E$ in line 140. For example, to use the word "DONE," change line 140 to

```
140 E$="DONE"
```

2. Line 3060 contains a timing delay used regularly in the program. If ERROR messages seem to change too fast, you can make the number 2000 larger. Try

```
3060 FOR J=1 TO 4000
```

3. The program can currently accept up to nine decision alternatives and/or nine importance factors. If you need more, increase the dimensioning in lines 120 and 130. Each dimension value is one more than the number the program will actually allow. Thus, to use 14 values, lines 120 and 130 should be

```
120 DIM L$(15),F$(15),V(15)
130 DIM C(15,15),D(15),Z(15)
```

MAIN ROUTINES

120– 150	Initializes and dimensions variables.
160– 420	Determines category of decision.
430– 540	Gets or sets T$.
550–1020	Gets list of possible alternatives from user.
1030–1470	Gets list of importance factors from user.
1480–1740	User rates each importance factor.
1750–2340	User rates the decision alternatives with respect to each importance factor.
2350–2570	Evaluates the various alternatives.
2580–2740	Sorts alternatives into their relative ranking.
2750–3010	Displays results.
3020–3050	Subroutine to clear screen and display header.
3060–3080	Time-wasting subroutine.

MAIN VARIABLES

NI	Number of decision alternatives.
L$	String array of the decision alternatives.
NF	Number of importance factors.
F$	String array of the importance factors.
V	Array of the relative values of each importance factor.
A	Index number of most important factor.
C	Array of relative values of each alternative with respect to each importance factor.
T	Decision category (1=item, 2=course of action, 3=yes or no).
T$	String name of decision category.
E$	String to signal the end of an input data list.
J, K	Loop indices.
R$	User reply string.
Q, N1, N2	Work variables.
D	Array of each alternative's value.
MX	Maximum value of all alternatives.
DF	Rating difference between best two alternatives.
Z	Array of the relative rankings of each alternative.
KK, SS	User input.

SUGGESTED PROJECTS

1. Allow the user to review the numerical input and modify it if desired.
2. Insights into a decision can often be gained by a sensitivity analysis. This involves running the program a number of times for the same decision. Each time, one input value is changed (usually the one you are least confident about). By seeing how the results change, you can determine which factors are the most important. Currently, this requires a complete rerunning of the program each time. Modify the program to allow a change of input after the regular output is produced. Then recalculate the results based on the new values. (Note that many input arrays are clobbered once all the input is given. This modification will require saving the original input in new arrays so that it can be reviewed later.)

LOAN

PURPOSE

One of the most frustrating things about borrowing money from a bank (or credit union or savings and loan) is that it's not easy to fully evaluate your options. When you are borrowing from a credit union to buy a new car, you might have the choice of a thirty-six or a forty-eight month repayment period. When buying a house, you can sometimes get a slightly lower interest rate for your loan if you can come up with a larger down payment. What option is best for you? How will the monthly payment be affected? Will there be much difference in how fast the principal of the loan decreases? How much of each payment will be for interest, which is tax-deductible?

You need to know the answers to all these questions to make the best decision. This program gives you the information you need.

HOW TO USE IT

The program first asks you the size of the loan you are considering. Only whole dollar amounts are allowed—no pennies. Loans of $100,000 or more are rejected, to avoid dealing with numbers too large to fit on the screen. However, you still can get the data you need for larger loans by simply dividing them by ten. For example, to evaluate a $300,000 loan, enter the amount as 30,000 (without entering the comma, however). You can easily add a zero (multiply by ten) mentally to learn the proper amounts for the monthly payment, remaining balance, monthly interest, and interest to-date.

Next you are asked the yearly interest rate for the loan. Enter this number as a percentage, such as "10.8." Next, you are asked to give the period of the loan in months. For a five year loan, enter 60. For a thirty year mortgage, enter 360. The program then displays this information for you and calculates the monthly payment that will cause the loan to be paid off with equal payments each month over the life of the loan.

At this point you have four options. First, you can show a monthly analysis. This displays a month-by-month breakdown, showing the state of the loan after each payment. The four columns of data shown for each month are the last digit of the payment number (or month number) of the loan, the remaining balance of the loan after that payment, the amount of that payment that was interest, and the accumulated interest paid to date. Up to ten lines of data are displayed on the screen, and then you can either press the **T** key to get the final totals for the loan, or any other key to get the data for the next ten months of the loan.

The second option is overriding the monthly payment. It is a common practice with second mortgage loans to make smaller monthly payments each month with a large "balloon" payment as the final payment. You can use this second option to try various monthly payments to see how they affect that big payment at the end. After overriding the monthly payment, you will want to use the first option next to get a monthly analysis and final totals using the new monthly payment. Be careful not to enter such a small payment that you allow the remaining balance of the loan to rise above $100,000. This will cause the program to end abnormally with the error message "BAD VALUE IN 1570."

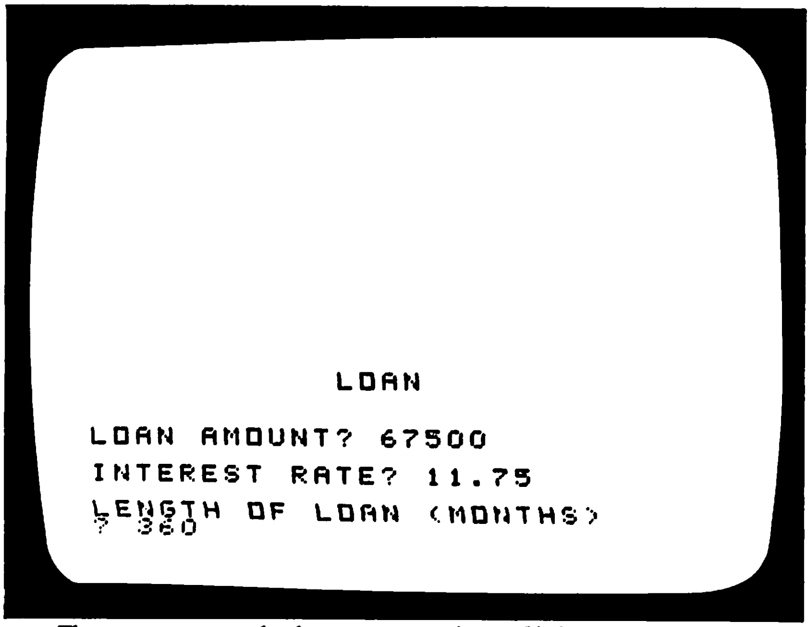
The third option is to simply start over. You will generally use this option if you are just comparing what the different monthly payments would be for different loan possibilities.

The fourth option ends the program.

By the way, there is a chance that the monthly payment calculated by your lender will differ from the one calculated here by a penny or two. We like to think that this is because we are making a more accurate calculation.

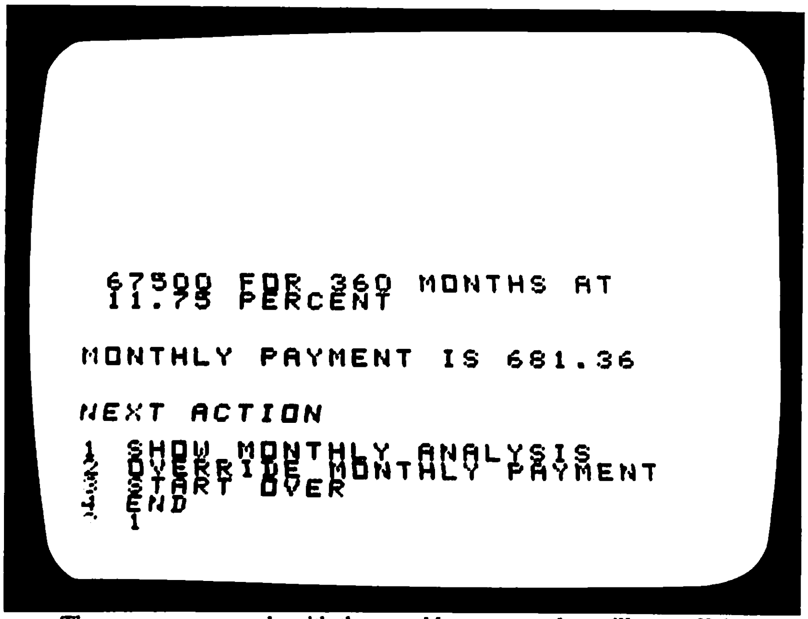
NOTE: SEE DISCLAIMER IN FRONT PART OF BOOK.

SAMPLE RUN



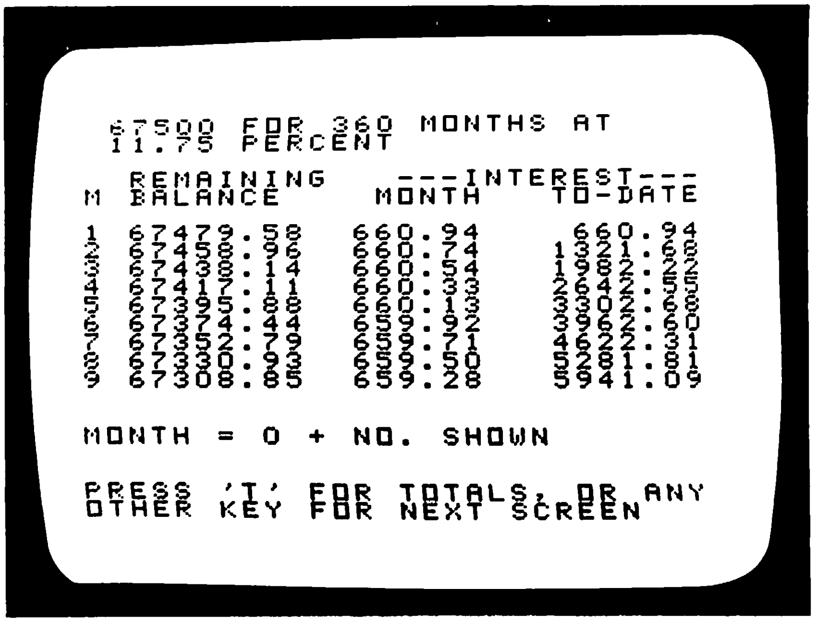
```
            LOAN

LOAN AMOUNT? 67500
INTEREST RATE? 11.75
LENGTH OF LOAN (MONTHS)
? 360
```

The operator enters the three necessary pieces of information about his or her loan.



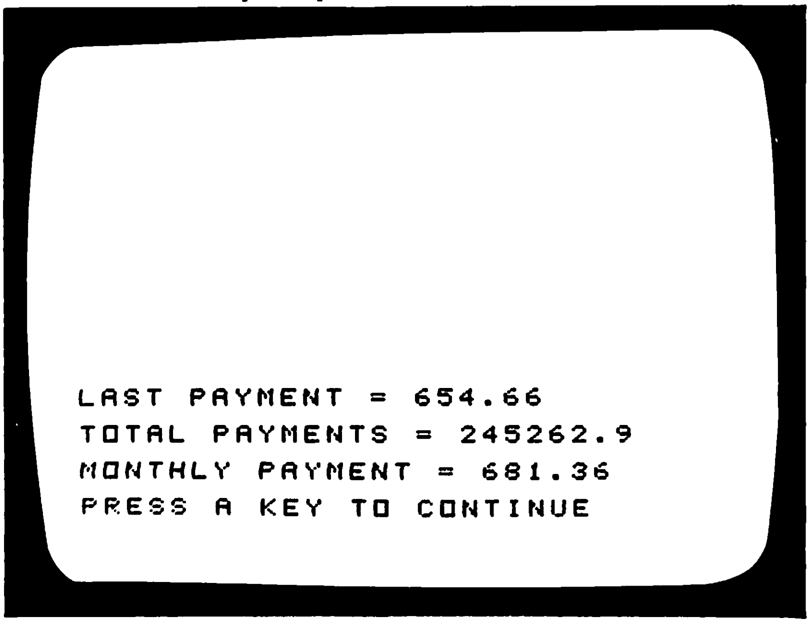
```
 67500 FOR 360 MONTHS AT
 11.75 PERCENT

MONTHLY PAYMENT IS 681.36

NEXT ACTION
1 SHOW MONTHLY ANALYSIS
2 OVERRIDE MONTHLY PAYMENT
3 START OVER
4 END
? 1
```

The program responds with the monthly payment that will pay off the loan with equal payments over its life, then asks the operator what to do next. The operator requests a monthly analysis.



```
67500 FOR 360 MONTHS AT
11.75 PERCENT
  REMAINING        ---INTEREST---
M BALANCE        MONTH     TO-DATE
1 67479.58      660.94      660.94
2 67458.96      660.74     1321.68
3 67438.14      660.54     1982.22
4 67417.11      660.33     2642.55
5 67395.88      660.13     3302.68
6 67374.44      659.92     3962.60
7 67352.79      659.71     4622.31
8 67330.93      659.50     5281.81
9 67308.85      659.28     5941.09

MONTH = 0 + NO. SHOWN

PRESS 'T' FOR TOTALS, OR ANY
OTHER KEY FOR NEXT SCREEN
```

The program responds with information about the first nine months of the loan, then the operator presses "T."



```
LAST PAYMENT = 654.66
TOTAL PAYMENTS = 245262.9
MONTHLY PAYMENT = 681.36
PRESS A KEY TO CONTINUE
```

After a few seconds the program displays totalling information about the loan.

PROGRAM LISTING

```
100 REM LOAN
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 BL$=CHR$(32)&CHR$(32)
140 BL$=BL$&BL$&BL$&BL$
150 PRINT TAB(12);"LOAN"
160 PRINT
170 PRINT
180 INPUT "LOAN AMOUNT? ":A
190 GOSUB 1590
200 IF A=0 THEN 180
210 PRINT
220 INPUT "INTEREST RATE? ":R
230 PRINT
240 PRINT "LENGTH OF LOAN (MONTHS)"
250 INPUT N
260 N=INT(ABS(N))
270 M=R/1200
280 GOSUB 1370
290 REM 'UP-ARROW' IN NEXT LINE
300 REM  IS SHIFTED '6'.
310 W=(1+M)^N
320 P=A*M*W/(W-1)
330 P=(INT(P*100+.99))/100
340 PRINT
350 PRINT "MONTHLY PAYMENT IS";P
360 FP=P
370 PRINT
380 PRINT
390 PRINT "NEXT ACTION"
400 PRINT
410 PRINT "1 SHOW MONTHLY ANALYSIS"
420 PRINT "2 OVERRIDE MONTHLY PAYMENT"
430 PRINT "3 START OVER"
440 PRINT "4 END"
450 INPUT C
460 C=INT(C)
470 IF C<1 THEN 370
480 IF C>4 THEN 370
490 PRINT
500 ON C GOTO 570,530,120,510
```

```
510 CALL CLEAR
520 END
530 CALL CLEAR
540 PRINT "ENTER MONTHLY PAYMENT"
550 INPUT P
560 GOTO 360
570 CALL CLEAR
580 GOSUB 600
590 GOTO 680
600 GOSUB 1370
610 PRINT TAB(3);"REMAINING";
620 PRINT TAB(15);"---INTEREST---"
630 PRINT "M";TAB(3);"BALANCE";
640 PRINT TAB(14);"MONTH";
650 PRINT TAB(22);"TO-DATE"
660 PRINT
670 RETURN
680 B=A*100
690 TT=0
700 TP=0
710 L=0
720 P=P*100
730 R$=""
740 FOR J=1 TO N
750 T=M*B
760 T=INT(T+.5)
770 IF J<>N THEN 790
780 P=B+T
790 TP=TP+P
800 B=B-P+T
810 TT=TT+T
820 IF B>=0 THEN 840
830 GOSUB 1630
840 IF KEY=84 THEN 1240
850 XX=5
860 W=B
870 GOSUB 1430
880 B$=S$
890 W=T
900 GOSUB 1430
910 T$=S$
920 W=TT
```

```
930 XX=7
940 GOSUB 1430
950 XX=5
960 TT$=S$
970 IF L>1 THEN 990
980 FS=J
990 IF J<>1 THEN 1010
1000 L=L+1
1010 PRINT STR$(L);TAB(3);B$;T$;TT$
1020 IF B<>0 THEN 1050
1030 J=N
1040 GOTO 1070
1050 L=L+1
1060 IF L<10 THEN 1240
1070 IF L=10 THEN 1090
1080 L=L+1
1090 PRINT
1100 PRINT
1110 PRINT "MONTH =";INT(FS/10)*10;"+ NO. SHOW
N"
1120 PRINT
1130 PRINT
1140 PRINT "PRESS 'T' FOR TOTALS, OR ANY"
1150 PRINT "OTHER KEY FOR NEXT SCREEN"
1160 CALL KEY(0,KEY,STATUS)
1170 IF STATUS=0 THEN 1160
1180 L=0
1190 CALL CLEAR
1200 GOSUB 600
1210 IF KEY<>84 THEN 1240
1220 PRINT
1230 PRINT "CALCULATING TOTALS..."
1240 NEXT J
1250 CALL CLEAR
1260 PRINT "LAST PAYMENT =";P/100
1270 PRINT
1280 PRINT "TOTAL PAYMENTS =";TP/100
1290 PRINT
1300 PRINT "MONTHLY PAYMENT =";FP
1310 PRINT
1320 PRINT "PRESS A KEY TO CONTINUE"
1330 CALL KEY(0,KEY,STATUS)
```

```
1340 IF STATUS=0 THEN 1330
1350 P=FP
1360 GOTO 370
1370 CALL CLEAR
1380 PRINT
1390 PRINT A;"FOR";N;"MONTHS AT"
1400 PRINT R;"PERCENT"
1410 PRINT
1420 RETURN
1430 W=INT(W)
1440 S$=STR$(W)
1450 K=LEN(S$)
1460 IF K<>1 THEN 1490
1470 S$=SEG$(BL$,1,XX)&".0"&S$
1480 RETURN
1490 IF K<>2 THEN 1520
1500 S$=SEG$(BL$,1,XX)&"."&S$
1510 RETURN
1520 D$="."&SEG$(S$,K-1,2)
1530 S$=SEG$(S$,1,K-2)&D$
1540 IF XX<>7 THEN 1570
1550 S$=SEG$(BL$,1,9-K)&S$
1560 GOTO 1580
1570 S$=SEG$(BL$,1,7-K)&S$
1580 RETURN
1590 A=INT(ABS(A))
1600 IF A<100000 THEN 1620
1610 A=0
1620 RETURN
1630 P=P+B
1640 TP=TP+B
1650 B=0
1660 RETURN
```

EASY CHANGE

1. To include the monthly payment in the heading at the top of each screen of the monthly analysis, make this change:

```
1410 IF FP=0 THEN 1420
1415 PRINT "MONTHLY PAYMENT=";FP
```

MAIN ROUTINES

120– 260	Displays title. Gets loan information.
270– 360	Calculates and displays monthly payment.
370– 500	Asks for next action. Goes to corresponding routine.
530– 560	Gets override for monthly payment.
570–1360	Calculates and displays monthly analysis.
1370–1420	Subroutine to clear screen and display data about the loan at the top.
1430–1580	Subroutine to convert integer amount to fixed-length string with aligned decimal point.
1590– 1620	Edits loan amount (size and whole dollar).
1630–1660	Subroutine to handle early payoff of loan.

MAIN VARIABLES

A	Amount of loan.
R	Interest rate (percentage).
N	Length of loan (number of months).
M	Monthly interest rate (not percentage).
W	Work variable.
P	Monthly payment (times 100).
FP	First monthly payment.
C	Choice of next action.
B	Remaining balance of loan (times 100).
TT	Total interest to date (times 100).
TP	Total payments to date.
L	Number of lines of data on screen.
R$	Reply from operator at keyboard.
J	Work variable for loops.
T	Monthly interest.
B$	String format of B.
T$	String format of T.
TT$	String format of TT.
S$	Work string.
D$	Work string.
K	Work variable.
KEY	Key pressed to continue program.
STATUS	Status of key.
FS	First payment number on this screen.
XX	Flag value for short or long numeric values.

SUGGESTED PROJECTS

1. Display a more comprehensive analysis of the loan along with the final totals. Show the ratio of total payments to the amount of the loan (TP divided by A), for example.
2. Modify the program to show an analysis of resulting monthly payments for a range of interest rates and/or loan lengths near those provided by the operator. For example, if an interest rate of 9.5 percent was entered, display the monthly payments for 8.5, 9.0, 9.5, 10.0, and 10.5 percent.

MILEAGE

PURPOSE

For many of us, automobile operating efficiency is a continual concern. This program can help by keeping track of gasoline consumption, miles driven, and fuel mileage for a motor vehicle. DATA statements are used to hold the vehicle's "data file." Thus a master file can be retained and updated by merely resaving the program after adding new information. The program computes mileage (miles per gallon or MPG) obtained after each gasoline fill-up. A running log of all information is maintained, allowing trends in vehicle operation efficiency to be easily checked.

HOW TO USE IT

Before running the program, you must enter a chronological history of your vehicle's gasoline consumption. This is accomplished by the use of DATA statements beginning at line 1500. For each gasoline fill-up, a record of the date, odometer reading, and number of gallons purchased is needed.

The form of each DATA statement should be:

line number DATA date, odometer value, number of gallons.

Some comments are in order here: the line number of each statement should increase as the information becomes more recent. We recommend starting with line number 1500 and incrementing each new line by five or ten. This allows later insertion to correct mistakes or to add previously missing data. A special DATA statement to signal the end of the data is already provided at line 9999. Leave this line intact and do not use line numbers higher than 9999 when entering your data.

When entering each of your DATA statements, keep the following in mind. The word DATA must be typed exactly as is. The remainder of each DATA line contains the three pieces of information needed by the program. They must be separated by commas. The first item is the date of the gasoline fill-up. It can be comprised of any keyboard characters but should not contain commas, or quotation marks. Only the leftmost eight characters will be used if more than eight are entered. We recommend that you use the general form typified by 12/25/83. However, you might want to use other notations, such as JAN 23, or WEEK 5 or something else. A comma must be typed after the date. The odometer reading and number of gallons purchased are then entered as numeric values separated by a comma. (See the Sample Run for an example of typical data entry.)

Once your data is entered, you can retain it by saving the program on cassette tape or disk. Then, as new data becomes available, you can load the old program, add the new data to it, and save the program again to preserve the entire data file.

Having entered the appropriate data, you are ready to run the program. It operates from a central command menu. The operator branches to one of three available subroutines. When a subroutine completes execution, control returns to the command menu for additional requests. A brief description of each subroutine now follows.

Verify DATA Statements

This scans the DATA statements to look for possible problems with the data. It will test to see if any odometer values are too big or too small (they are presumed to be between 0 and 999999), or if any gallons values are too big or too small (they are presumed to be between 0 and 999). It will look to see that the odometer values increase with each successive entry. Also, it will make sure that you have entered some data. If any of these problems are found, an appropriate error message will be displayed and the program will end. If a bad data record is found (usually more or less than three items on a DATA line, or perhaps a string value for one of the numeric quantities), it will issue a DATA ERROR message and abort the program. If you then type PRINT N, a number will be printed indicating which DATA statement (counting from the first) is in error. The offending statement is usually either the line indicated or the line immediately preceding it. If all data is in the correct form the subroutine will display the beginning and ending dates for the data and the total number of data records found. It will then ask that you hit any key to re-enter the command menu.

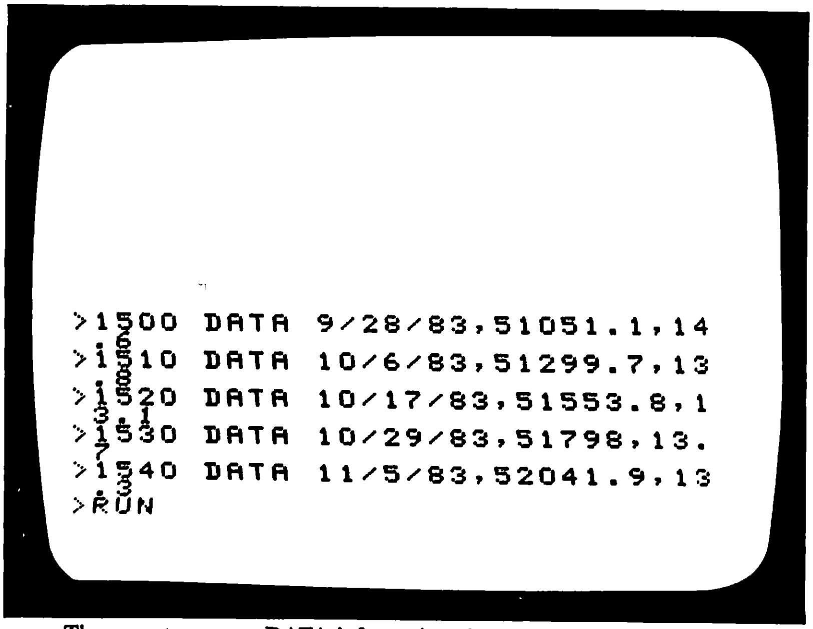
Display Mileage Information

This subroutine computes mileage (miles per gallon) from the available data. It formats all information and displays it in tabular form. Numbers are rounded so that four columns of information can be displayed on one line. When data fills the screen, the user is prompted to hit any key to continue the listing. When all data is displayed, pressing any key will re-enter command mode. An error message will be printed and the program will terminate if a fatal error is found in the DATA records.

Terminate Program

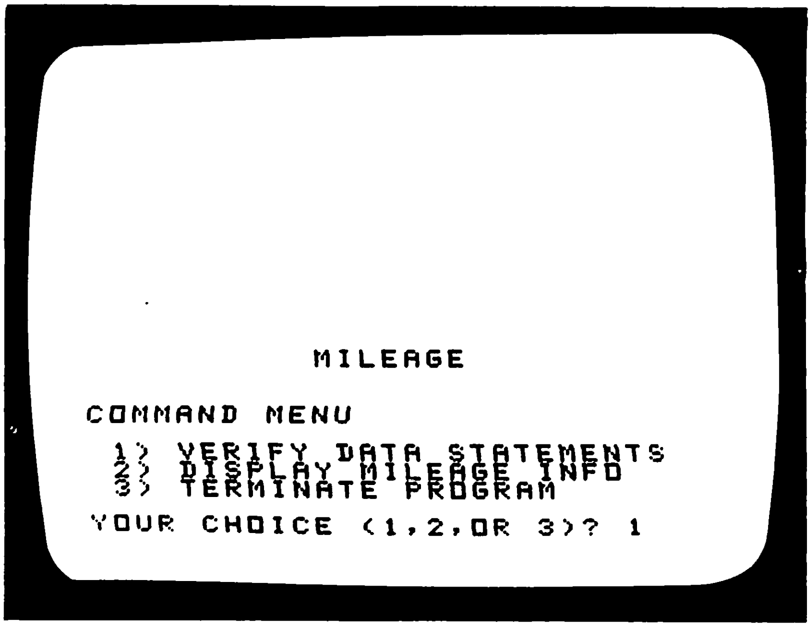
This subroutine ends program execution and returns the computer to direct BASIC.

SAMPLE RUN



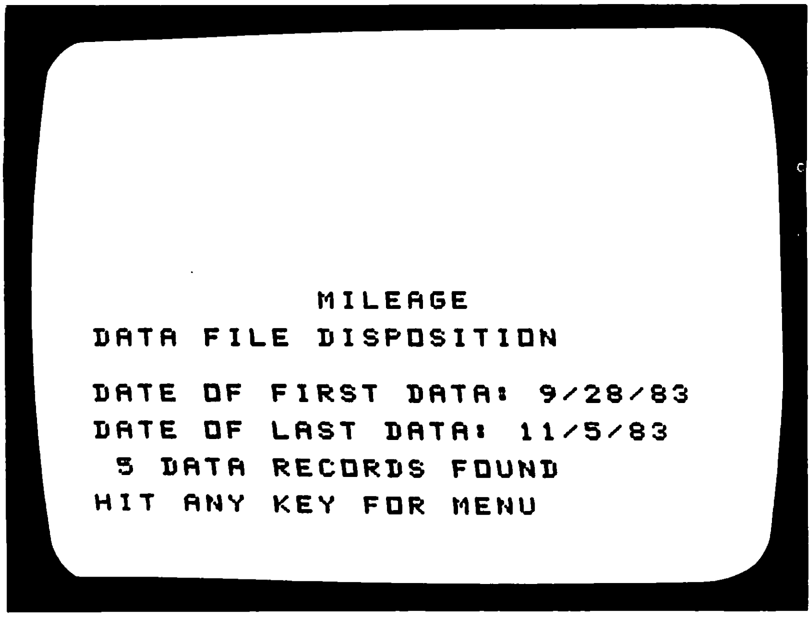
```
>1500 DATA 9/28/83,51051.1,14
>1510 DATA 10/6/83,51299.7,13
>1520 DATA 10/17/83,51553.8,1
>1530 DATA 10/29/83,51798,13.
>1540 DATA 11/5/83,52041.9,13
>RUN
```

The operator enters DATA information about his or her car, and then types RUN to start the program.



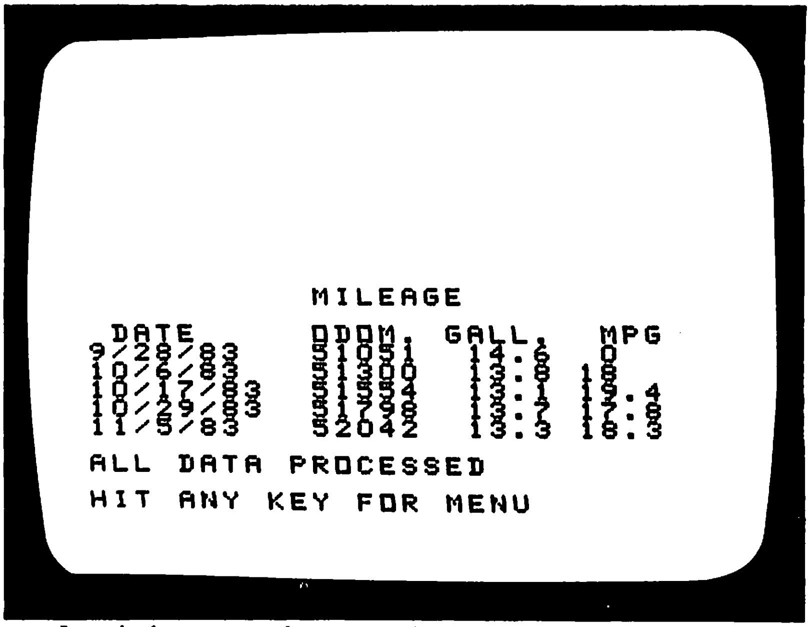
```
          MILEAGE

COMMAND MENU

  1> VERIFY DATA STATEMENTS
  2> DISPLAY MILEAGE INFO
  3> TERMINATE PROGRAM

YOUR CHOICE (1,2,OR 3)? 1
```

The program's menu is displayed and the operator chooses option 1. This requests the program to scan the DATA statements for possible errors.



```
          MILEAGE
DATA FILE DISPOSITION

DATE OF FIRST DATA: 9/28/83
DATE OF LAST DATA: 11/5/83
 5 DATA RECORDS FOUND
HIT ANY KEY FOR MENU
```

All is O.K. with the data. The dates of the first and last DATA statements are shown along with the total number entered.



```
           MILEAGE
  DATE     ODOM.  GALL.   MPG
9/28/83    51051  14.6    0
10/6/83    51300  13.8   18
10/17/83   51554  13.1   19.4
10/29/83   51798  13.7   17.8
11/5/83    52042  13.3   18.3
ALL DATA PROCESSED
HIT ANY KEY FOR MENU
```

Later in the program, the operator chooses option 2. This displays the data along with the fuel MPG obtained. The program will re-enter the command menu when a key is hit.

PROGRAM LISTING

```
100 REM MILEAGE
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 PRINT TAB(11);"MILEAGE"
140 PRINT
150 PRINT
160 PRINT "COMMAND MENU"
170 PRINT
180 PRINT TAB(2);"1) VERIFY DATA STATEMENTS"
190 PRINT TAB(2);"2) DISPLAY MILEAGE INFO"
200 PRINT TAB(2);"3) TERMINATE PROGRAM"
210 PRINT
220 INPUT "YOUR CHOICE (1,2,OR 3)? ":C
230 C=INT(C)
240 IF C<1 THEN 120
250 IF C>3 THEN 120
260 ON C GOTO 290,830,270
270 CALL SOUND(300,200,2)
280 END
```

```
290 CALL CLEAR
300 RESTORE
310 PRINT TAB(11);"MILEAGE"
320 PRINT
330 PRINT "DATA FILE DISPOSITION"
340 PRINT
350 PRINT
360 N=0
370 DD=-0.001
380 READ T$,D,G
390 IF T$="XXX" THEN 710
400 A$=T$
410 N=N+1
420 IF N>1 THEN 450
430 PRINT "DATE OF FIRST DATA: ";A$
440 PRINT
450 IF D<0 THEN 480
460 IF D>999999 THEN 480
470 GOTO 520
480 PRINT "--BAD ODO VALUE=";D
490 PRINT TAB(3);"AT DATE: ";A$
500 CALL SOUND(300,200,2)
510 END
520 IF D>DD THEN 620
530 PRINT "--INCONSISTENT ODO VALUES"
540 PRINT
550 PRINT TAB(3);"ODO READS";D
560 PRINT TAB(3);"AT DATE: ";A$
570 PRINT
580 PRINT TAB(3);"YET READS ";DD
590 PRINT TAB(3);"AT PREVIOUS DATE"
600 CALL SOUND(300,200,2)
610 END
620 IF G<0 THEN 650
630 IF G>999 THEN 650
640 GOTO 690
650 PRINT "--BAD GALLONS VALUE=";G
660 PRINT TAB(3);"AT DATE: ";A$
670 CALL SOUND(300,200,2)
680 END
690 DD=D
700 GOTO 380
```

```
710 IF N>0 THEN 750
720 PRINT "--NO DATA FOUND"
730 CALL SOUND(300,200,2)
740 END
750 PRINT "DATE OF LAST DATA: ";A$
760 PRINT
770 PRINT N;"DATA RECORDS FOUND"
780 PRINT
790 PRINT "HIT ANY KEY FOR MENU"
800 CALL KEY(0,KK,SS)
810 IF SS=0 THEN 800
820 GOTO 120
830 GOSUB 1420
840 RESTORE
850 DD=-1
860 READ A$,D,G
870 IF A$<>"XXX" THEN 910
880 PRINT
890 PRINT "ALL DATA PROCESSED"
900 GOTO 780
910 A$=SEG$(A$,1,8)
920 D=INT(D+0.5)
930 IF D<0 THEN 960
940 IF D>999999 THEN 960
950 GOTO 1010
960 PRINT
970 PRINT "ERROR IN ODOMETER VALUE"
980 PRINT "AT DATE: ";A$
990 CALL SOUND(300,200,2)
1000 END
1010 D$=STR$(D)
1020 L=LEN(D$)
1030 LD=16-L
1040 G=INT(G*10+0.5)/10
1050 IF G<0 THEN 1080
1060 IF G>999 THEN 1080
1070 GOTO 1130
1080 PRINT
1090 PRINT "ERROR IN GALLONS VALUE"
1100 PRINT "AT DATE: ";A$
1110 CALL SOUND(300,200,2)
1120 END
```

```
1130 G$=STR$(G)
1140 L=LEN(G$)
1150 LG=22-L
1160 IF INT(G)<>G THEN 1180
1170 LG=LG-2
1180 M=0
1190 IF DD<0 THEN 1220
1200 IF G=0 THEN 1220
1210 M=(D-DD)/G
1220 IF M<0 THEN 1260
1230 IF M>999 THEN 1260
1240 M=INT(M*10+0.5)/10
1250 GOTO 1270
1260 M=0
1270 M$=STR$(M)
1280 L=LEN(M$)
1290 LM=27-L
1300 IF INT(M)<>M THEN 1320
1310 LM=LM-2
1320 DD=D
1330 PRINT A$;TAB(LD);D$;TAB(LG);G$;TAB(LM);M$
1340 K=K+1
1350 IF K<18 THEN 860
1360 PRINT
1370 PRINT "HIT ANY KEY TO CONTINUE"
1380 CALL KEY(0,KK,SS)
1390 IF SS=0 THEN 1380
1400 GOSUB 1420
1410 GOTO 860
1420 CALL CLEAR
1430 PRINT TAB(11);"MILEAGE"
1440 PRINT
1450 PRINT TAB(2);"DATE";TAB(11);"ODOM.";TAB(1
7);"GALL.";TAB(24);"MPG"
1460 K=0
1470 RETURN
9999 DATA XXX,0,0
```

EASY CHANGES

1. If you would like to give a name to the data file and have that name print out with the command menu, change line 130 and add line

125 as follows:

```
125 B$="VOLVO 1983"
130 PRINT "MILEAGE FOR: ";B$
```

Just set B$ in line 125 to whatever file name you wish to use.

MAIN ROUTINES

120– 260	Command menu. Displays available subroutines and branches to the operator's choice.
270– 280	Subroutine to terminate execution.
290– 820	Subroutine to verify DATA statements.
830–1410	Subroutine to display mileage information.
1420–1470	Subroutine to print column headings.
1500–9998	User-created assigned DATA statements.
9999	DATA statement for end of data.

MAIN VARIABLES

C	Command flag (1=verify data, 2=display mileage, 3=terminate execution).
N	Number of data records read.
A$	Date of current data record.
DD	Previous odometer value.
D	Current odometer value.
G	Current gallons value.
M	Current MPG value.
LD	Print position for odometer value.
LG	Print position for gallons value.
LM	Print position for mileage value.
T$	Temporary string variable.
L	Length of string.
K	Number of lines printed since CALL CLEAR.
D$	String of D.
G$	String of G.
M$	String of M.
KK, SS	User input.

SUGGESTED PROJECTS

1. Calculate and print the average MPG over the whole data file. This will be the total miles driven divided by the total gallons pur-

chased. The total miles driven is the difference between the odometer values of the last and first DATA statements. The total gallons used is the sum of all the gallons values from the second DATA statement to the last DATA statement.

2. Add an option to do statistical calculations over a given subset of the data. The operator inputs a beginning and ending date. He is then shown things like average MPG, total miles driven, total gallons purchased, etc., all computed over the range requested.
3. Write a subroutine to graphically display MPG. A bar graph might work well.
4. Add a new parameter in each data record—the cost of each fill-up. Then compute things like the total cost of gasoline, miles/dollar, etc.

QUEST/EXAM

PURPOSE

If you've ever had to analyze the results of a questionnaire, or grade a multiple-choice examination, you know what a tedious and time-consuming process it can be. This is particularly true if you need to accumulate statistics for each question showing how many people responded with each possible answer.

With this program, you provide the data, and the computer does the work.

HOW TO USE IT

First, enter the number of questions on the questionnaire or exam. The maximum is 25, but a practical maximum is 20 due to the screen width of the TI-99/4A. Then enter the number of choices per question. This is an integer from 2 through 9. Each answer will be a digit from one to this number or left blank.

Finally, you are asked if you want to input names for each entry. Enter Y or N (for yes or no). This is especially useful when grading exams, to enable you to re-check later to verify that you entered the proper data for each student.

At this point, the program asks you for the answer key. If you are scoring an exam, provide the correct answers. The program displays "guide numbers" to help you keep track of which answers you are providing. If you are analyzing a questionnaire, you have no answer key, so just press the **ENTER** key.

Now the program asks you to begin providing the answers for each entry. Again, guide numbers are displayed above the area where you are to enter the data so you can more easily provide the proper answer for the proper question number. If no answer was given for a particular question, leave a blank space. However, if the last question was left blank, you will have to include a dummy answer after the space. See below for a method to do this.

If you make a mistake when entering the data, the program will tell you and ask you to re-enter it. This is most commonly caused by a space in the last position without a dummy answer after it. Remember that each answer must be either a blank or a number from one to the number of choices you allowed per question.

Here is the method to avoid entering blanks for unanswered questions. Suppose you have a maximum of 5 possible answers per question. Simply tell the program there are 6 choices per question. Then, when a question is unanswered, you can enter a 6 instead of leaving it blank.

If you specified that you wanted names for each entry, the program asks you for the name after you have entered each person's answers. Do not use commas in the name.

If you provided an answer key, the program displays the number and percentage correct after each entry before going on to ask for the next one. When you have no more entries, press the **ENTER** key instead of entering a string of answers.

At this point, the program displays four options from which you choose your next action. Here are brief explanations. You can experiment to verify how they work.

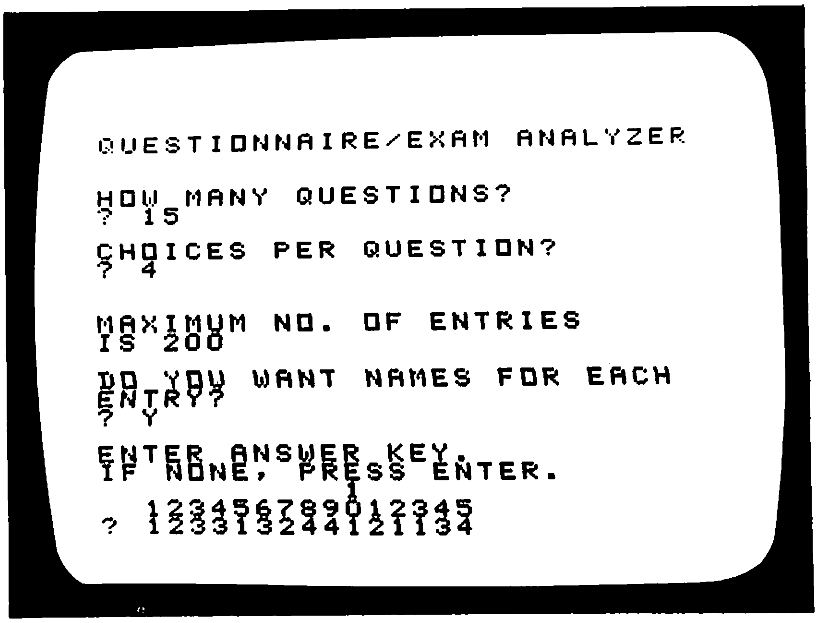
Option one lets you analyze each question to see how many people responded with each answer. The percentage of people who responded with each answer is indicated with the word "RIGHT."

Option two allows you to go back and provide more entries. This allows you to pause after entering part of the data, do some analysis of what you have entered so far, and then go back and continue entering data.

Option three lets you review what you have entered, including the answer key. This permits you to check for duplicate, omitted, or erroneous entries.

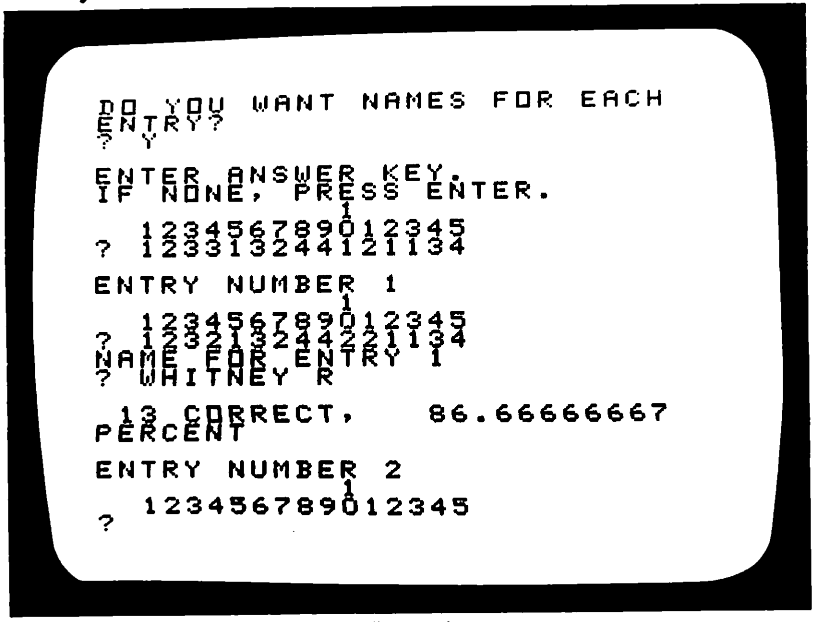
Option four ends the program.

SAMPLE RUN



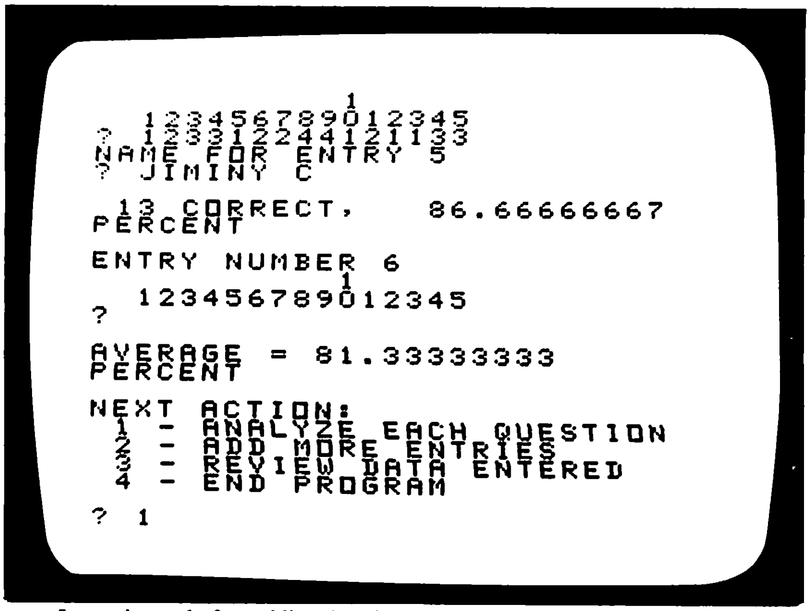
```
QUESTIONNAIRE/EXAM ANALYZER

HOW MANY QUESTIONS?
? 15

CHOICES PER QUESTION?
? 4

MAXIMUM NO. OF ENTRIES
IS 200

DO YOU WANT NAMES FOR EACH
ENTRY?
? Y

ENTER ANSWER KEY.
IF NONE, PRESS ENTER.
           1
  123456789012345
? 123313244121134
```

The operator provides the necessary information, including the answer key.



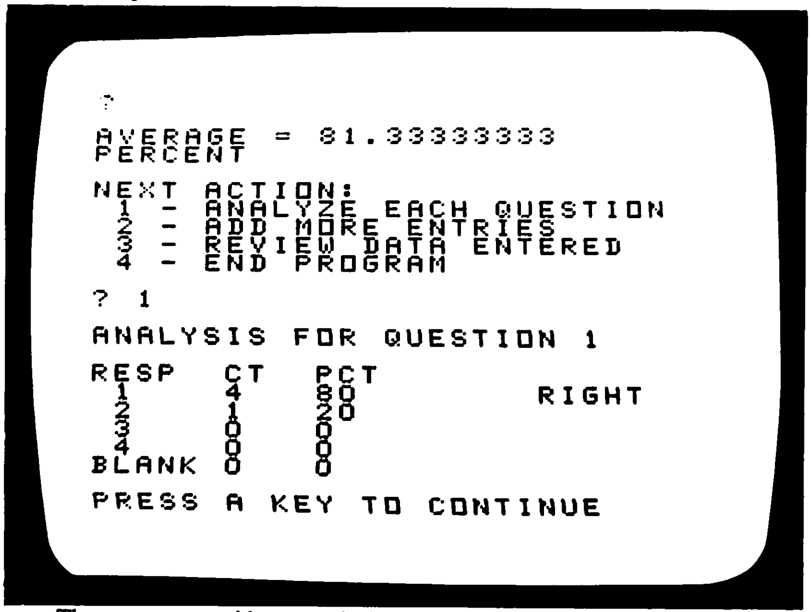
```
DO YOU WANT NAMES FOR EACH
ENTRY?
? Y

ENTER ANSWER KEY.
IF NONE, PRESS ENTER.
           1
  123456789012345
? 123313244121134

ENTRY NUMBER 1
           1
  123456789012345
? 123213244221134
NAME FOR ENTRY 1
? WHITNEY R

 13 CORRECT,    86.66666667
PERCENT

ENTRY NUMBER 2
           1
  123456789012345
?
```

The answers are entered for the first student.



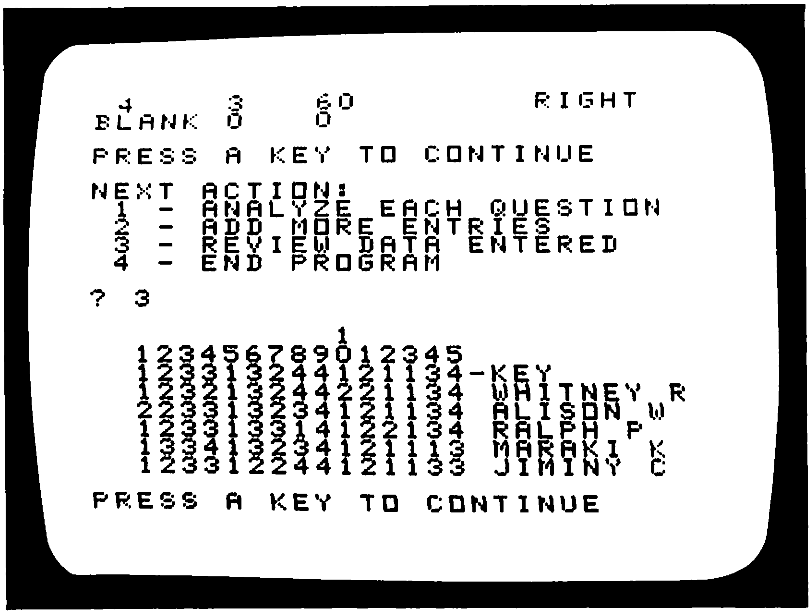
```
            1
  123456789012345
? 123312244121133
NAME FOR ENTRY 5
? JIMINY C

 13 CORRECT,     86.66666667
PERCENT

ENTRY NUMBER 6
            1
  123456789012345
?

AVERAGE = 81.3333333
PERCENT

NEXT ACTION:
 1 - ANALYZE EACH QUESTION
 2 - ADD MORE ENTRIES
 3 - REVIEW DATA ENTERED
 4 - END PROGRAM

? 1
```

Later, instead of providing data for a sixth student, the operator presses the ENTER key, indicating no more entries. The program displays the overall percentage correct, and displays a menu of choices. The operator picks option 1.



```
?

AVERAGE = 81.3333333
PERCENT

NEXT ACTION:
 1 - ANALYZE EACH QUESTION
 2 - ADD MORE ENTRIES
 3 - REVIEW DATA ENTERED
 4 - END PROGRAM

? 1

ANALYSIS FOR QUESTION 1

RESP    CT   PCT
 1      4    80          RIGHT
 2      1    20
 3      0    0
 4      0    0
BLANK   0    0

PRESS A KEY TO CONTINUE
```

The program provides an analysis of the responses for question number one, then waits for a key to be pressed.



```
 4      3     60           RIGHT
BLANK   0     0
PRESS A KEY TO CONTINUE
NEXT ACTION:
 1 - ANALYZE EACH QUESTION
 2 - ADD MORE ENTRIES
 3 - REVIEW DATA ENTERED
 4 - END PROGRAM
? 3
             1
   123456789012345
   123313244121134-KEY
   123321324422113 4 WHITNEY R
   223313234121134 ALISON W
   123313314122134 RALPH P
   133413234121113 MARAKI K
   123312244121133 JIMINY C
PRESS A KEY TO CONTINUE
```

Later, the operator asks for option 3, which lists the data entered for each of the students.

PROGRAM LISTING

```
100 REM QUEST/EXAM
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 PRINT "QUESTIONNAIRE/EXAM ANALYZER"
140 PRINT
150 PRINT
160 E$="ERROR. RE-ENTER PLEASE."
170 P$="PRESS A KEY TO CONTINUE"
180 PRINT "HOW MANY QUESTIONS?"
190 INPUT Q
200 Q=INT(ABS(Q))
210 IF Q<1 THEN 180
220 IF Q>25 THEN 180
230 GOSUB 1850
240 PRINT
250 PRINT "CHOICES PER QUESTION?"
260 INPUT D
270 D=ABS(INT(D))
280 IF D<2 THEN 250
```

```
290 IF D>9 THEN 250
300 PRINT
310 DIM C(10)
320 PRINT
330 N=200
340 PRINT "MAXIMUM NO. OF ENTRIES"
350 PRINT "IS";N
360 DIM Q$(200)
370 PRINT
380 PRINT "DO YOU WANT NAMES FOR EACH"
390 PRINT "ENTRY?"
400 INPUT R$
410 IF R$="N" THEN 440
420 IF R$<>"Y" THEN 380
430 DIM N$(200)
440 PRINT
450 PRINT "ENTER ANSWER KEY."
460 PRINT "IF NONE, PRESS ENTER."
470 GOSUB 1510
480 C$=STR$(D)
490 INPUT A$
500 IF LEN(A$)=0 THEN 570
510 IF LEN(A$)=Q THEN 540
520 PRINT E$
530 GOTO 440
540 T$=A$
550 GOSUB 1750
560 IF T$="B" THEN 520
570 K=1
580 PRINT
590 R=0
600 PRINT "ENTRY NUMBER";K
610 GOSUB 1510
620 INPUT Q$(K)
630 W=LEN(Q$(K))
640 IF W=0 THEN 850
650 IF W<>Q THEN 690
660 T$=Q$(K)
670 GOSUB 1750
680 IF T$<>"B" THEN 710
690 PRINT E$
700 GOTO 590
```

```
710 IF R$="N" THEN 740
720 PRINT "NAME FOR ENTRY";K
730 INPUT N$(K)
740 IF LEN(A$)=0 THEN 820
750 FOR J=1 TO Q
760 IF SEG$(A$,J,1)<>SEG$(Q$(K),J,1)THEN 780
770 R=R+1
780 NEXT J
790 TR=TR+R
800 PRINT
810 PRINT R;"CORRECT,  ";R*100/Q;"PERCENT"
820 PRINT
830 K=K+1
840 IF K<=N THEN 590
850 K=K-1
860 IF LEN(A$)=0 THEN 900
870 IF K=0 THEN 1600
880 PRINT
890 PRINT "AVERAGE =";TR*100/(Q*K);"PERCENT"
900 GOTO 1600
910 FOR J=1 TO Q
920 R=0
930 PRINT
940 PRINT "ANALYSIS FOR QUESTION";J
950 PRINT
960 PRINT "RESP  CT  PCT"
970 FOR L=0 TO D
980 C(L)=0
990 NEXT L
1000 FOR L=1 TO K
1010 T$=SEG$(Q$(L),J,1)
1020 IF T$<>CHR$(32)THEN 1050
1030 W=0
1040 GOTO 1060
1050 W=VAL(T$)
1060 C(W)=C(W)+1
1070 NEXT L
1080 IF K=0 THEN 1600
1090 FOR L=1 TO D
1100 W$=SEG$(STR$(C(L)*100/K),1,7)
1110 PRINT L;TAB(6);C(L);TAB(11);W$;
1120 IF LEN(A$)<>0 THEN 1150
```

```
1130 PRINT
1140 GOTO 1200
1150 T$=STR$(L)
1160 IF T$=SEG$(A$,J,1)THEN 1190
1170 PRINT
1180 GOTO 1200
1190 PRINT TAB(21);"RIGHT"
1200 NEXT L
1210 PRINT "BLANK";TAB(6);C(0);TAB(10);C(0)*10
0/K
1220 PRINT
1230 PRINT P$
1240 CALL KEY(0,KEY,STATUS)
1250 IF STATUS=0 THEN 1240
1260 NEXT J
1270 GOTO 1600
1280 PRINT
1290 L=0
1300 GOSUB 1510
1310 IF LEN(A$)=0 THEN 1330
1320 PRINT TAB(3);A$;"-KEY"
1330 FOR J=1 TO K
1340 IF R$<>"Y" THEN 1370
1350 PRINT TAB(3);Q$(J);CHR$(32);N$(J)
1360 GOTO 1380
1370 PRINT TAB(3);Q$(J);" #";J
1380 L=L+1
1390 IF L<10 THEN 1450
1400 L=0
1410 PRINT
1420 PRINT P$
1430 CALL KEY(0,KEY,STATUS)
1440 IF STATUS=0 THEN 1430
1450 NEXT J
1460 PRINT
1470 PRINT P$
1480 CALL KEY(0,KEY,STATUS)
1490 IF STATUS=0 THEN 1480
1500 GOTO 1600
1510 W=INT(Q/10)
1520 IF W<1 THEN 1570
1530 FOR J=1 TO W
```

```
1540 X=J*10+1
1550 PRINT TAB(J*10+1);J;
1560 NEXT J
1570 PRINT
1580 PRINT TAB(3);H$
1590 RETURN
1600 PRINT
1610 PRINT "NEXT ACTION:"
1620 PRINT " 1 - ANALYZE EACH QUESTION"
1630 PRINT " 2 - ADD MORE ENTRIES"
1640 PRINT " 3 - REVIEW DATA ENTERED"
1650 PRINT " 4 - END PROGRAM"
1660 PRINT
1670 INPUT T
1680 T=INT(ABS(T))
1690 IF T<1 THEN 1600
1700 IF T>4 THEN 1600
1710 ON T GOTO 910,830,1280,1720
1720 CALL CLEAR
1730 END
1740 CALL CLEAR
1750 FOR J=1 TO LEN(T$)
1760 W$=SEG$(T$,J,1)
1770 IF W$=CHR$(32)THEN 1810
1780 IF W$<"1" THEN 1830
1790 IF SEG$(T$,J,1)>C$ THEN 1830
1800 IF W$>C$ THEN 1830
1810 NEXT J
1820 RETURN
1830 T$="B"
1840 RETURN
1850 T$="1234567890"
1860 H$=""
1870 FOR J=1 TO Q/10+1
1880 H$=H$&T$
1890 NEXT J
1900 H$=SEG$(H$,1,Q)
1910 RETURN
```

MAIN ROUTINES

120– 170	Initializes variables.
180– 430	Performs initilization dialog. Selects options. Allocates arrays.
440– 560	Gets answer key (if any) from operator.
570– 700	Gets exam data for Kth entry.
710– 730	Gets name for Kth entry, if applicable.
740– 810	Scores Kth exam, if applicable.
860– 900	Displays average score, if an exam.
910–1270	Analyzes responses to each question.
1280–1500	Displays data entered.
1510–1590	Subroutine to display guide numbers over input data area.
1600–1710	Displays choices for next action. Gets response and goes to appropriate routine.
1720–1730	Ends program.
1740–1840	Subroutine to verify answers are in proper range.
1850–1910	Subroutine to build string of guide digits.

MAIN VARIABLES

E$	Error message.
P$	Message about pressing a key to continue.
Q	Number of questions (1–25).
D	Number of choices per question (2–9).
C	Array for tallying number of people responding with each choice.
N	Maximum number of entries.
Q$	Array of N strings of entries.
R$	Set to N if no names for each entry, or Y otherwise.
N$	Array of Q names (if R$ is Y).
A$	Answer key string (null if not an exam).
C$	String value of highest legal answer choice.
K	Counter of number of exams scored.
R	Number of questions answered right (if exam).
W	Work variable.
J, L, M	Loop variables.
TR	Total right for all entries.
T$	Temporary work string variable.

KEY	User input.
STATUS	Reply from keyboard.
H$	Heading (units digit) for data entry.
W$	Work string variable.

SUGGESTED PROJECTS

1. Add an option to change the answer key after the data for the exams is entered. This would be useful in case a mistake was found when reviewing the data.
2. Add an option to allow the operator to re-score each of the exams after all are entered, in case some were overlooked at the time of entry.
3. Combine some of the capabilities of the STATS program with this one.

Section 2

Educational Programs

Education is one area where computers are certain to have more and more impact. Though a computer cannot completely replace a human teacher, the machine does have certain advantages. It is ready anytime you are, allows you to go at your own pace, handles rote drill effortlessly, and is devoid of any personality conflicts.

With a good software library, the TI-99/4A can be a valuable learning center in the school or at home. Here are six programs to get you started.

Mathematics is certainly a "natural" subject for computers. NUMBERS is designed for pre-school children. While familiarizing youngsters with computers, it provides an entertaining way for them to learn numbers and elementary counting. ARITHMETIC is aimed at older, grade school students. It provides drill in various kinds of math problems. The child can adjust the difficulty factors, allowing the program to be useful for several years.

The TI-99/4A is by no means restricted to mathematical disciplines. We include two programs designed to improve your word skills. VOCAB will help you expand your vocabulary. TACHIST turns the TI-99/4A into a reading clinic, helping you to improve your reading speed.

Do you have trouble familiarizing yourself with the increasingly prevalent metric system? METRIC is the answer.

Need help learning a certain subject? FLASHCARD allows you to create your own "computer flashcards." Then you can drill yourself until you get it right.

ARITHMETIC

PURPOSE

ARITHMETIC provides mathematics drills for grade school children. The student can request problems in addition, subtraction, or multiplication from the program. Also, he or she may ask that the problems be easy, medium, or hard. The program should be useful to a child over an extended period of time. He can progress naturally to a harder category of problems when he begins to regularly perform well at one level. The difficulty and types of problems encompass those normally encountered by school children between the ages of six and ten.

The problems are constructed randomly within the constraints imposed by the degree of difficulty selected. This gives the student fresh practice each time the program is used. After entering answers, he is told whether he was right or wrong. The correct answers are also displayed.

HOW TO USE IT

To begin, the student must indicate what type of problem he wishes to do. The program requests an input of **1**, **2**, or **3** to indicate addition, subtraction, or multiplication, respectively. It then asks whether easy, medium, or hard problems are desired. Again an input of **1**, **2**, or **3** is required.

Now the screen will clear and four problems of the desired type will be displayed. The user now begins to enter his answers to each problem.

A question mark is used to prompt the user for each digit of the answer, one digit at a time. This is done moving right to left, the way arithmetic problems are naturally solved.

To start each problem, the question mark will appear in the spot for the rightmost (or units column) digit of the answer. When the key for a digit from 0–9 is pressed, that digit will replace the question mark on the screen. The question mark moves to the immediate left waiting for a digit for the "tens" column.

Digits are entered in this right-to-left manner until the complete answer has been input. Then the **ENTER** key must be pressed. This will end the answer to the current problem and move the question mark to begin the answer for the next question.

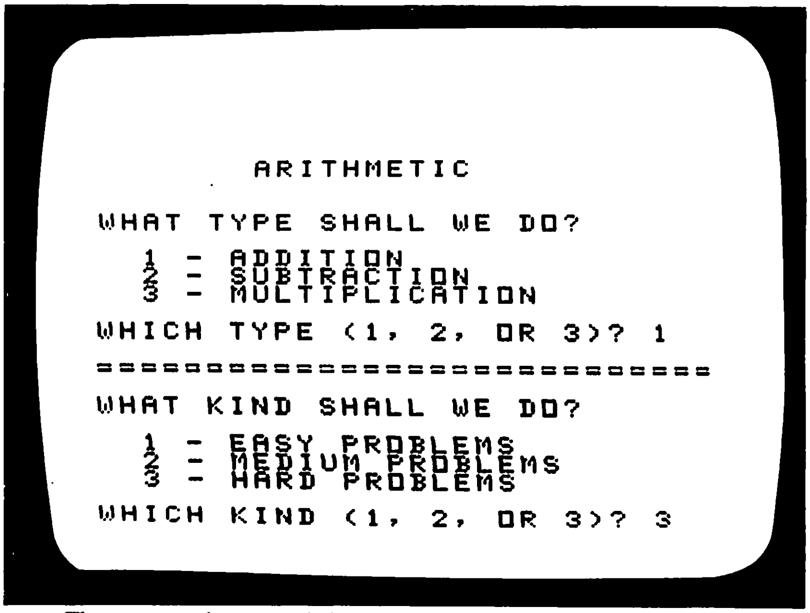
If the **ENTER** key is pressed to begin a problem, an answer of zero is assumed. No problems created by this program have answers of more than three digits. If a four-digit answer is given, the program will accept the answer, but then go immediately to the next problem. Answers to the problems are never negative.

The program will display the correct answers to the four problems on the screen after the student has entered his four answers. The message RIGHT or WRONG will also be displayed below each problem.

Then the message "HIT ANY KEY TO CONTINUE" will be displayed. After the key is pressed, a new set of four problems of the same type will be presented.

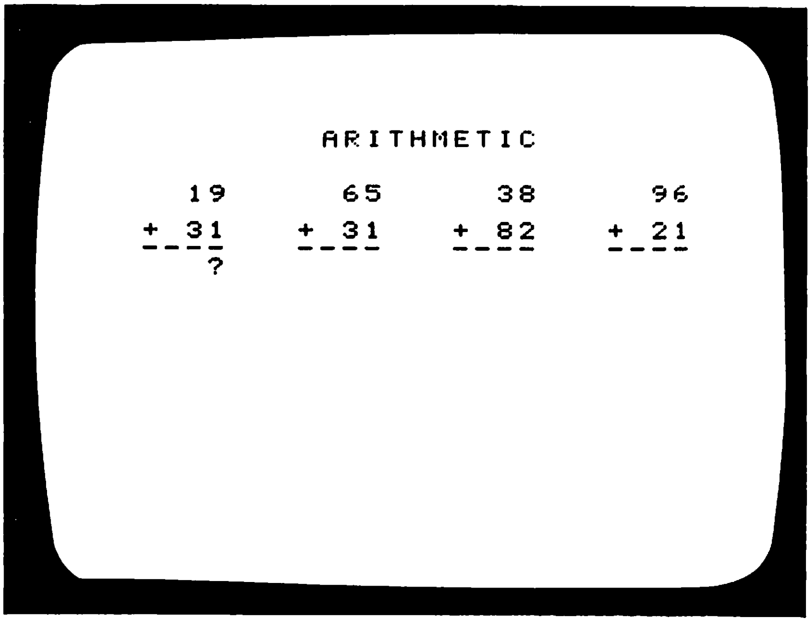
This continues until twenty problems have been worked. Before ending, the program shows what the student's performance has been. This is expressed as the number of problems solved correctly and also as the percentage of problems solved correctly.

SAMPLE RUN



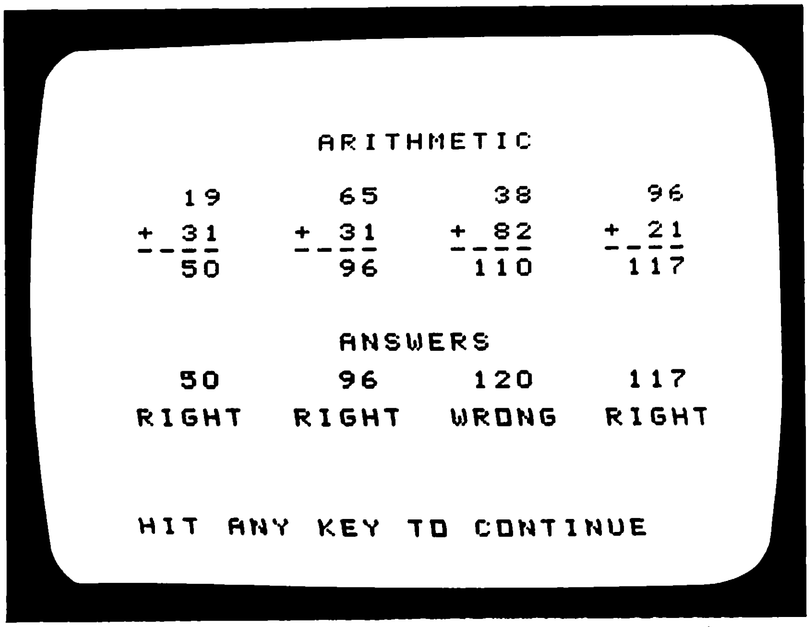
```
             ARITHMETIC

WHAT TYPE SHALL WE DO?
  1 - ADDITION
  2 - SUBTRACTION
  3 - MULTIPLICATION
WHICH TYPE (1, 2, OR 3)? 1
=============================
WHAT KIND SHALL WE DO?
  1 - EASY PROBLEMS
  2 - MEDIUM PROBLEMS
  3 - HARD PROBLEMS
WHICH KIND (1, 2, OR 3)? 3
```

The operator chooses to do hard addition problems.



```
             ARITHMETIC

    19      65      38      96
  + 31    + 31    + 82    + 21
  ----    ----    ----    ----
     ?
```

The initial set of 4 problems is presented. With a question mark, the program prompts the operator for the answer to the first problem.



```
        ARITHMETIC

   19        65        38        96
+  31     +  31     +  82     +  21
 ----      ----      ----      ----
   50        96       110       117

          ANSWERS

   50        96       120       117
RIGHT     RIGHT     WRONG     RIGHT


HIT ANY KEY TO CONTINUE
```

The operator has entered his or her four answers. The program displays the correct answers and indicates whether or not each problem was solved correctly. The program waits for the operator to hit any key in order to continue with the next set of four problems.

PROGRAM LISTING

```
100 REM ARITHMETIC
110 REM (C) 1984 DILITHIUM PRESS
120 ND=0
130 NP=20
140 DIM A(4),B(4),C(4),G(4)
150 RANDOMIZE
160 CALL CLEAR
170 PRINT TAB(8);"ARITHMETIC"
180 PRINT
190 PRINT
200 PRINT "WHAT TYPE SHALL WE DO?"
210 PRINT
220 PRINT TAB(3);"1 - ADDITION"
230 PRINT TAB(3);"2 - SUBTRACTION"
240 PRINT TAB(3);"3 - MULTIPLICATION"
250 PRINT
260 INPUT "WHICH TYPE (1, 2, OR 3)? ":T
```

```
270 T=INT(T)
280 IF T<1 THEN 250
290 IF T>3 THEN 250
300 PRINT
310 CALL HCHAR(24,3,61,28)
320 PRINT
330 PRINT
340 PRINT "WHAT KIND SHALL WE DO?"
350 PRINT
360 PRINT TAB(3);"1 - EASY PROBLEMS"
370 PRINT TAB(3);"2 - MEDIUM PROBLEMS"
380 PRINT TAB(3);"3 - HARD PROBLEMS"
390 PRINT
400 INPUT "WHICH KIND (1, 2, OR 3)? ":D
410 D=INT(D)
420 IF D<1 THEN 390
430 IF D>3 THEN 390
440 ON D GOTO 450,490,580
450 GOSUB 1440
460 GOSUB 1360
470 GOSUB 1400
480 GOTO 640
490 GOSUB 1440
500 GOSUB 1400
510 IF T=3 THEN 550
520 GOSUB 1470
530 GOSUB 1360
540 GOTO 640
550 GOSUB 1500
560 GOSUB 1360
570 GOTO 640
580 GOSUB 1470
590 GOSUB 1360
600 GOSUB 1400
610 IF T<>3 THEN 640
620 GOSUB 1440
630 GOSUB 1400
640 IF T<>2 THEN 710
650 FOR J=1 TO 4
660 IF B(J)<=C(J)THEN 700
670 R=C(J)
680 C(J)=B(J)
```

```
690 B(J)=R
700 NEXT J
710 GOSUB 1530
720 GOSUB 1300
730 GOSUB 1660
740 Y=10
750 FOR J=1 TO 4
760 X=1+J*7
770 GOSUB 1090
780 G(J)=N
790 NEXT J
800 FOR J=1 TO 7
810 K=ASC(SEG$("ANSWERS",J,1))
820 CALL HCHAR(14,J+13,K)
830 NEXT J
840 Y=16
850 FOR J=1 TO 4
860 X=1+J*7
870 GOSUB 1910
880 NEXT J
890 Y=18
900 FOR J=1 TO 4
910 X=-3+J*7
920 P$="WRONG"
930 IF A(J)<>G(J)THEN 960
940 P$="RIGHT"
950 NR=NR+1
960 FOR K=1 TO 5
970 Q=ASC(SEG$(P$,K,1))
980 CALL HCHAR(Y,X+K,Q)
990 NEXT K
1000 NEXT J
1010 PRINT TAB(3);"HIT ANY KEY TO CONTINUE";
1020 CALL KEY(0,R,ST)
1030 IF ST=0 THEN 1020
1040 ND=ND+4
1050 IF ND>=NP THEN 1070
1060 GOTO 440
1070 GOSUB 1980
1080 END
1090 N=0
1100 M=1
```

```
1110 CALL HCHAR(Y,X,63)
1120 CALL KEY(0,KK,SS)
1130 IF SS=0 THEN 1120
1140 IF KK<>13 THEN 1200
1150 IF M<>1 THEN 1180
1160 CALL HCHAR(Y,X,48)
1170 RETURN
1180 CALL HCHAR(Y,X,32)
1190 RETURN
1200 IF KK<48 THEN 1120
1210 IF KK>57 THEN 1120
1220 V=KK-48
1230 CALL HCHAR(Y,X,KK)
1240 N=N+M*V
1250 M=M*10
1260 IF M<=1000 THEN 1280
1270 RETURN
1280 X=X-1
1290 GOTO 1110
1300 CALL CLEAR
1310 FOR J=1 TO 10
1320 K=ASC(SEG$("ARITHMETIC",J,1))
1330 CALL HCHAR(3,J+12,K)
1340 NEXT J
1350 RETURN
1360 FOR K=1 TO 4
1370 C(K)=INT(RND*(H-L+1))+L
1380 NEXT K
1390 RETURN
1400 FOR K=1 TO 4
1410 B(K)=INT(RND*(H-L+1))+L
1420 NEXT K
1430 RETURN
1440 H=9
1450 L=0
1460 RETURN
1470 H=99
1480 L=0
1490 RETURN
1500 H=25
1510 L=1
1520 RETURN
```

```
1530 ON T GOTO 1540,1580,1620
1540 FOR J=1 TO 4
1550 A(J)=B(J)+C(J)
1560 NEXT J
1570 RETURN
1580 FOR J=1 TO 4
1590 A(J)=C(J)-B(J)
1600 NEXT J
1610 RETURN
1620 FOR J=1 TO 4
1630 A(J)=C(J)*B(J)
1640 NEXT J
1650 RETURN
1660 FOR J=1 TO 4
1670 Y=6
1680 X=1+J*7
1690 Q=C(J)
1700 GOSUB 1840
1710 Y=8
1720 Q=B(J)
1730 GOSUB 1840
1740 X=X-3
1750 P=43
1760 IF T=1 THEN 1800
1770 P=45
1780 IF T=2 THEN 1800
1790 P=88
1800 CALL HCHAR(Y,X,P)
1810 CALL HCHAR(Y+1,X,45,4)
1820 NEXT J
1830 RETURN
1840 P$=STR$(Q)
1850 R=LEN(P$)
1860 FOR K=1 TO R
1870 P=ASC(SEG$(P$,K,1))
1880 CALL HCHAR(Y,X-R+K,P)
1890 NEXT K
1900 RETURN
1910 Y=16
1920 FOR W=1 TO 4
1930 X=1+W*7
1940 Q=A(W)
```

```
1950 GOSUB 1840
1960 NEXT W
1970 RETURN
1980 CALL CLEAR
1990 PRINT TAB(8);"ARITHMETIC"
2000 PRINT
2010 PRINT
2020 PRINT "YOU GOT";NR;"RIGHT"
2030 PRINT
2040 PRINT "OUT OF";NP;"PROBLEMS"
2050 P=NR/NP*100
2060 PRINT
2070 PRINT "THAT'S";P
2080 PRINT "PERCENT CORRECT"
2090 RETURN
```

EASY CHANGES

1. The program currently does twenty problems per session. You can change this number by altering the variable NP in line 130. For example,

   130 NP=12

   will cause the program to do only twelve problems per session. The value of NP should be kept a positive multiple of four.
2. Zero is currently allowed as a possible problem operand. If you do not wish to allow this, change lines 1450 and 1480 to read as follows:

   1450 L=1
   1480 L=1

MAIN ROUTINES

120– 170	Initializes constants, displays header.
180– 430	Asks operator for type of problems desired.
440– 720	Sets A, B, C arrays, clears screen.
730–1080	Mainline routine–displays problems, gets operator's answers, displays correct answers and user's performance.
1090–1290	Subroutine to get and display user's answers.
1300–1350	Subroutine to clear screen and display title.

1360–1430	Subroutines to set B, C arrays.
1440–1520	Subroutines to set L, H.
1530–1650	Subroutine to calculate A array from B, C arrays.
1660–1830	Subroutine to display problems.
1840–1900	Subroutine to print the number Q at screen location X,Y.
1910–1970	Subroutine to display the correct answers.
1980–2090	Subroutine to display operator's performance.

MAIN VARIABLES

NP	Number of problems to do in the session.
ND	Number of problems done.
NR	Number of correct answers given.
C, B, A	Arrays of top operand, bottom operand, and correct answer to each problem.
N	Operator's answer to current problem.
G	Array of operator's answers.
T	Type of problems requested (1=addition, 2=subtraction, 3=multiplication).
D	Kind of problem requested (1=easy, 2=medium, 3=hard).
H, L	Highest, lowest integers to allow as problem operands.
KK	ASCII value of input character.
Q	Value to be displayed.
M	Answer column being worked on.
R	Work variable.
V	Decimal value of KK.
X, Y	Horizontal, vertical screen position to print.
J, K, W	Loop indices and work variables.
P	Percentage of correct answers, print character.
P$	Character or string to be printed.
SS, ST	Value indicating if key has been pressed.

SUGGESTED PROJECTS

1. Keep track of problems missed and repeat them quickly for additional practice.
2. No negative operands or answers are currently allowed. Rewrite the problem generation routines and the operator's answer routines to allow the possibility of negative answers.

3. The answers are now restricted to three-digit numbers. However, the program would work fine for four-digit numbers if the operands of the problems were allowed to be large enough. Dig into the routines at lines 440–720 and 1440–1520. See how they work and then modify them to allow possible four-digit answers.
4. The operator cannot currently correct any mistakes he makes while typing in his answers. Modify the program to allow him to do so.
5. Modify the program to allow problems in division.

FLASHCARD

PURPOSE

There are certain things that the human mind is capable of learning only through repetition. Not many people can remember the multiplication tables after their first exposure, for example. The same applies to learning the vocabulary of a foreign language, the capital cities of the fifty states, or famous dates in history. The best way to learn them is to simply review them over and over until you have them memorized.

A common technique for doing this involves the use of flashcards. You write one half of the two related pieces of information on one side of a card, and the other half on the other side. After creating a set of these cards, you can drill yourself on them over and over until you always remember what's on the other side of each card.

But why waste precious natural resources by using cards? Use your computer instead. This program lets you create flashcards, drill using them, and save them for later review.

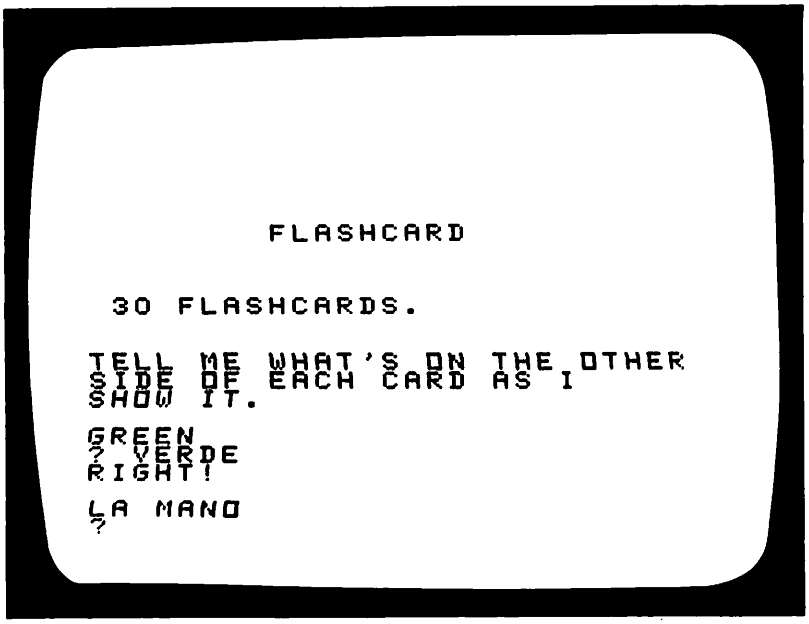
HOW TO USE IT

As currently written, the program immediately begins drilling you on Spanish vocabulary words. After explaining how to use the program with these words, we'll show you how to enter your own flashcards (see Easy Changes).

The program flashes one side of one card on the screen for you. Both are chosen at random—the side and the card. Your job is to respond with the other side. If you enter it correctly, the program says "RIGHT!" If not, it tells you the correct answer. In either event, the program continues by picking another side and card at random. This goes on until you finally respond by simply pressing the **ENTER** key, which tells the program that you do not want to drill any more. It then tells you how many you got right out of the number attempted, as well as the percentage, and gives you the option of drilling more or ending the program.

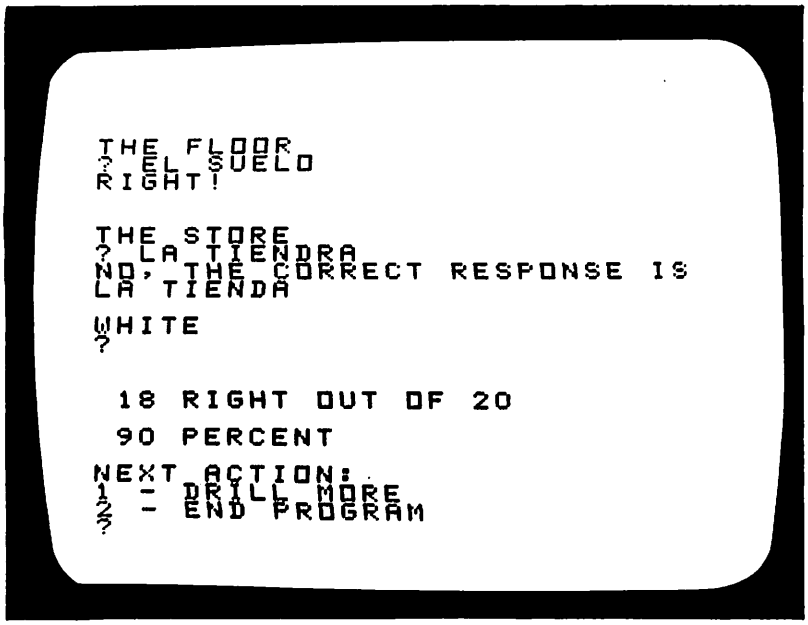
During the drill, the program will not repeat a card that was used in the previous four questions.

SAMPLE RUN



```
               FLASHCARD

   30 FLASHCARDS.

TELL ME WHAT'S ON THE OTHER
SIDE OF EACH CARD AS I
SHOW IT.

GREEN
? VERDE
RIGHT!

LA MANO
?
```

The program begins by asking the operator a question. The operator enters his or her response, the correct answer.



```
THE FLOOR
? EL SUELO
RIGHT!

THE STORE
? LA TIENDRA
NO, THE CORRECT RESPONSE IS
LA TIENDA
WHITE
?

 18 RIGHT OUT OF 20

 90 PERCENT

NEXT ACTION:
1 - DRILL MORE
2 - END PROGRAM
?
```

The operator continues to run the program. Later in the program, the number and percentage of correctly answered questions are shown.

PROGRAM LISTING

```
100 REM FLASHCARD
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 L=50
140 M=5
150 DIM F$(50),B$(50),P(4)
160 RANDOMIZE
170 PRINT TAB(9);"FLASHCARD"
180 PRINT
190 PRINT
200 K=1
210 C=0
220 W=0
230 IF K>L THEN 310
240 READ R$
250 IF R$="XXX" THEN 320
260 F$(K)=R$
270 READ R$
```

```
280 B$(K)=R$
290 K=K+1
300 GOTO 230
310 PRINT "FLASHCARD ARRAY FULL."
320 K=K-1
330 IF K>=M THEN 360
340 PRINT "NOT ENOUGH FLASHCARDS."
350 END
360 PRINT
370 PRINT K;"FLASHCARDS."
380 PRINT
390 PRINT
400 PRINT "TELL ME WHAT'S ON THE OTHER"
410 PRINT "SIDE OF EACH CARD AS I"
420 PRINT "SHOW IT."
430 PRINT
440 R=INT(RND*K)+1
450 FOR J=0 TO M-2
460 IF P(J)=R THEN 430
470 NEXT J
480 J=RND
490 IF J>.5 THEN 530
500 PRINT F$(R)
510 C$=B$(R)
520 GOTO 550
530 PRINT B$(R)
540 C$=F$(R)
550 INPUT R$
560 IF LEN(R$)=0 THEN 690
570 IF R$=C$ THEN 620
580 PRINT "NO, THE CORRECT RESPONSE IS"
590 PRINT C$
600 W=W+1
610 GOTO 640
620 PRINT "RIGHT!"
630 C=C+1
640 FOR J=1 TO M-2
650 P(J-1)=P(J)
660 NEXT J
670 P(M-2)=R
680 GOTO 430
690 PRINT
```

```
700 PRINT
710 IF C+W=0 THEN 760
720 PRINT C;"RIGHT OUT OF";C+W
730 PRINT
740 PRINT C*100/(C+W);"PERCENT"
750 PRINT
760 PRINT "NEXT ACTION:"
770 PRINT "1 - DRILL MORE"
780 PRINT "2 - END PROGRAM"
790 INPUT R
800 R=INT(ABS(R))
810 IF R<>2 THEN 840
820 CALL CLEAR
830 END
840 IF R=1 THEN 360
850 GOTO 790
860 REM FLASCARD DATA
870 DATA THE PEN,LA PLUMA
880 DATA THE DOOR,LA PUERTA
890 DATA THE SCHOOL,LA ESCUELA
900 DATA THE FLOOR,EL SUELO
910 DATA THE STORE,LA TIENDA
920 DATA THE HAND,LA MANO
930 DATA THE HOUSE,LA CASA
940 DATA THE FRIEND,EL AMIGO
950 DATA THE DINNER,LA COMIDA
960 DATA THE CHAIR,LA SILLA
970 DATA TO ARRIVE,LLEGAR
980 DATA TO ASK,PREGUNTAR
990 DATA TO BUY,COMPRAR
1000 DATA TO BRING,LLEVAR
1010 DATA TO COME,VENIR
1020 DATA TO EAT,COMER
1030 DATA TO FIND,HALLAR
1040 DATA TO GO,ANDAR
1050 DATA TO HAVE,TENER
1060 DATA TO KNOW,SABER
1070 DATA BLUE,AZUL
1080 DATA GREEN,VERDE
1090 DATA RED,ROJO
1100 DATA WHITE,BLANCO
1110 DATA YELLOW,AMARILLO
```

```
1120 DATA ENOUGH,BASTANTE
1130 DATA FAR,LEJOS
1140 DATA FEW,POCOS
1150 DATA MANY,MUCHOS
1160 DATA NEAR,CERCA
9999 DATA XXX
```

EASY CHANGES

1. Replace the DATA statements with your own flashcards. A comma is used to separate the two sides of each card. Don't use commas, colons, or quotation marks as part of your cards. Leave 9999 alone as it is to signal the end of the cards.
2. Change the limits of the number of flashcards that can be entered by altering the value of L and M in lines 130 and 140, and the DIM statements in line 150. L is the upper limit and M is the minimum. If you have 16 K of memory, you can make L as large as about four hundred for flashcards this size. Do not make M much larger than about ten or so, or you will slow down the program and use more memory than you might want. But, it increases the number of cards drilled upon before repeating. In line 150, the value following P must be one less than M.
3. To cause the program to always display side one of the flashcards (and ask you to respond with side two), change this line:

    ```
    490 REM
    ```

   To cause it to always display side two, change it this way:

    ```
    490 GOTO 530
    ```

MAIN ROUTINES

120–220	Initializes variables. Creates arrays. Displays title.
230–360	Reads flashcards from DATA statements.
370–750	Drills operator on flashcards in memory.
760–850	Displays options and analyzes response. Branches to appropriate routine.
870–9999	Data statements with flashcard data.

MAIN VARIABLES

L	Upper limit of number of flashcards that can be entered.
M	Minimum number of flashcards that can be entered.

R	Subscript of random flashcard chosen during drill.
K	Number of flashcards entered.
W	Number of wrong responses.
C	Number of correct responses.
F$	Array containing front side of flashcards (side 1).
B$	Array containing back side of flashcards (side 2).
P	Array containing subscripts of M−1 previous flashcards during drill.
J	Loop and subscript variable.
C$	The correct response during drill.
R$	Response from operator. Also temporary string variable.

SUGGESTED PROJECTS

1. Modify the program for use in a classroom environment. Require the operator to drill a fixed number of times (maybe 20 or 50). Don't allow a null response to end the drill. For example, you could make these changes:

```
560 REM
635 IF C=20 THEN 720
745 END
```

This will cause the program to continue until 20 correct answers are given, and then end.

METRIC

PURPOSE

In case you don't realize it, we live in a metric world. The United States is one of the last holdouts, but that is changing rapidly. So if you're still inching along or watching those pounds, it's time to convert.

METRIC is an instructional program designed to familiarize you with the metric system. It operates in a quiz format; the program randomly forms questions from its data resources. You are then asked to compare two quantities—one in our old English units and one in the corresponding metric units. When you are wrong, the exact conversion and the rule governing it are given.

The two quantities to compare are usually within 50% of each other. Thus, you are constantly comparing an "English" quantity and a metric one which are in the same ball park. This has the effect of providing you with some insight by sheer familiarity with the questions.

HOW TO USE IT

The first thing the program does is ask you how many questions you would like to do for the session. Any value of one or higher is acceptable.

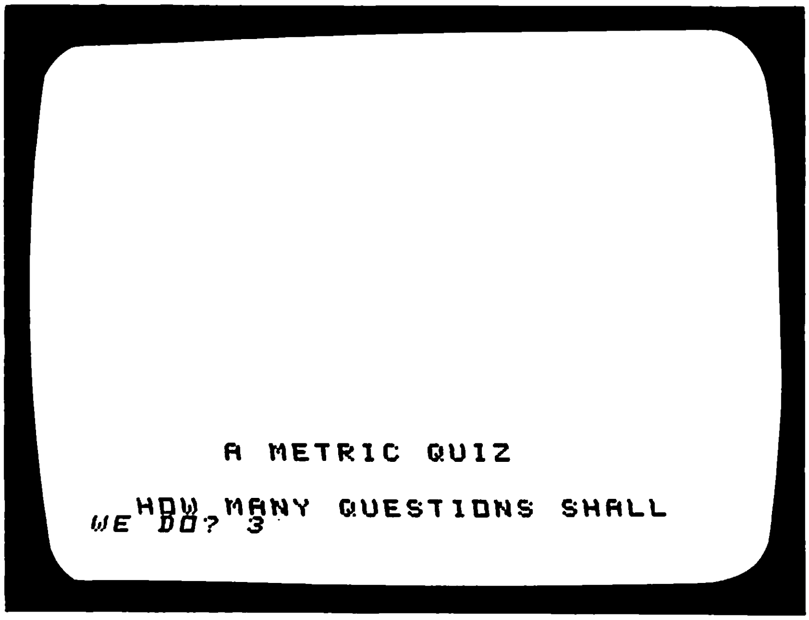
The sample run shows how each question is formulated. A quantity in English units is compared with one in metric units. Either one may appear first in the question. Each quantity will have an integral value. The relating word ("longer," "hotter," "heavier," etc.) indicates what type of quantities are being compared.

There are three possible replies to each question. Pressing Y or N means that you think the answer is yes or no, respectively. Pressing any other key indicates that you have no idea as to the correct answer.

If you answer the question correctly, you will be duly congratulated and the program will proceed to the next question. A wrong answer or a response of "no idea," however, will generate some diagnostic information. The first value used in the question will be shown converted to its exact equivalent in the corresponding units. Also, the rule governing the situation will be displayed. At the end of any question, the program will request that you hit any key to proceed to the next question.

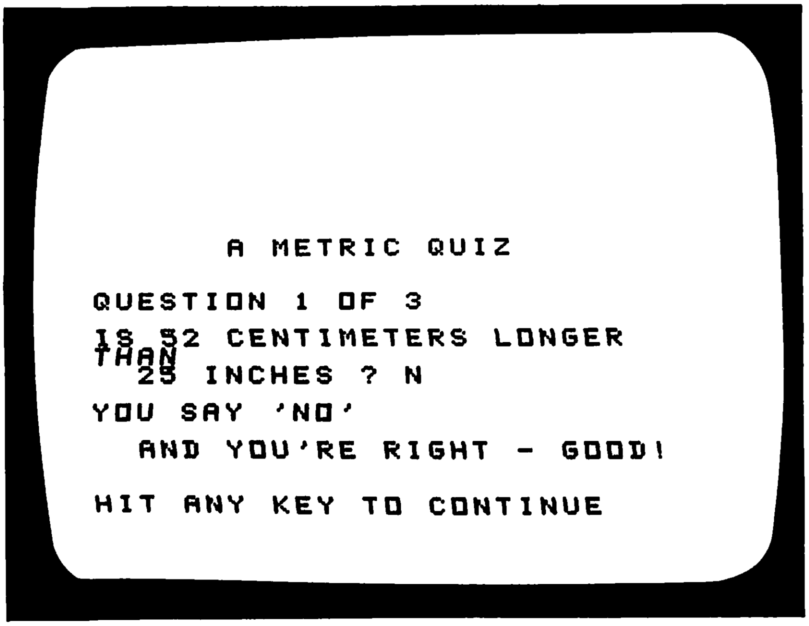
The program will continue generating the requested number of questions. Before ending, it will show you how many correct answers you gave and your percentage correct.

SAMPLE RUN



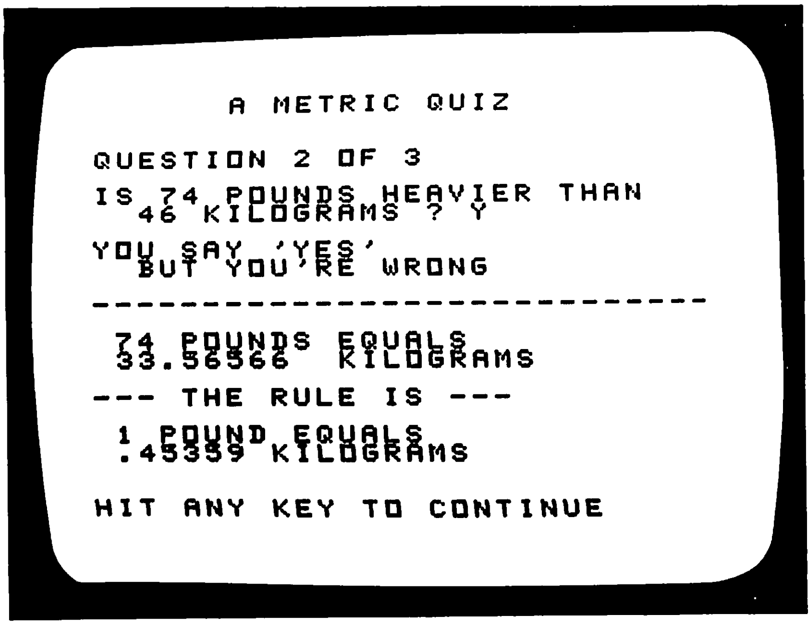
```
      A METRIC QUIZ

HOW MANY QUESTIONS SHALL
WE DO? 3
```

The operator requests a three question quiz.



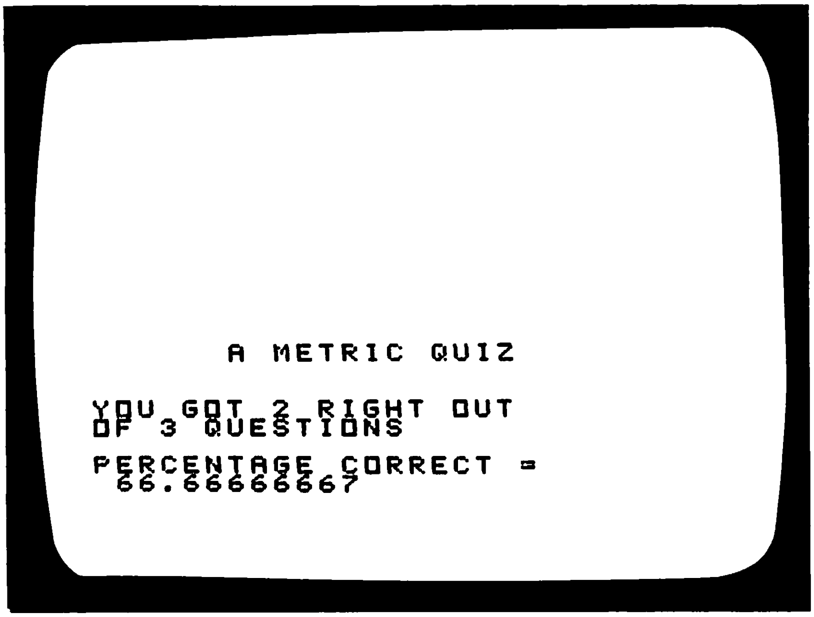
```
        A METRIC QUIZ

QUESTION 1 OF 3
IS 52 CENTIMETERS LONGER
THAN
  25 INCHES ? N
YOU SAY 'NO'
  AND YOU'RE RIGHT - GOOD!

HIT ANY KEY TO CONTINUE
```

The first question is correctly answered "no." The program waits for a key to be pressed before continuing the quiz.



```
        A METRIC QUIZ

QUESTION 2 OF 3
IS 74 POUNDS HEAVIER THAN
  46 KILOGRAMS ? Y
YOU SAY 'YES'
  BUT YOU'RE WRONG
----------------------------
 74 POUNDS EQUALS
 33.56566  KILOGRAMS
--- THE RULE IS ---
 1 POUND EQUALS
 .45359 KILOGRAMS

HIT ANY KEY TO CONTINUE
```

The second question is wrongly answered "yes." The correct conversion and governing rule are displayed.



```
         A METRIC QUIZ

YOU GOT 2 RIGHT OUT
OF 3 QUESTIONS
PERCENTAGE CORRECT =
 66.6666667
```

The program shows the number and percentage of correctly answered questions.

PROGRAM LISTING

```
100 REM METRIC
110 REM (C) 1984 DILITHIUM PRESS
120 DIM ES$(30),MS$(30),R$(30)
130 DIM C(30),EP$(30),MP$(30)
140 B$=CHR$(32)
150 RANDOMIZE
160 GOSUB 340
170 GOSUB 410
180 PRINT TAB(3);"HOW MANY QUESTIONS SHALL"
190 INPUT "WE DO? ":NQ
200 NQ=INT(NQ)
210 IF NQ<1 THEN 170
220 FOR J=1 TO NQ
230 GOSUB 460
240 GOSUB 1260
250 NEXT J
260 GOSUB 410
270 PRINT "YOU GOT";NR;"RIGHT OUT"
280 PRINT "OF";NQ;"QUESTIONS"
```

```
290 PRINT
300 P=100*NR/NQ
310 PRINT "PERCENTAGE CORRECT =";P
320 PRINT
330 END
340 RESTORE
350 ND=0
360 ND=ND+1
370 READ ES$(ND),MS$(ND),R$(ND),C(ND),EP$(ND),
MP$(ND)
380 IF ES$(ND)<>"XXX" THEN 360
390 ND=ND-1
400 RETURN
410 CALL CLEAR
420 PRINT TAB(7);"A METRIC QUIZ"
430 PRINT
440 PRINT
450 RETURN
460 N=INT(ND*RND)+1
470 F=0
480 IF RND<0.5 THEN 500
490 F=1
500 V1=INT(RND*99)+2
510 V3=V1*C(N)
520 IF F=0 THEN 540
530 V3=V1/C(N)
540 IF N<>1 THEN 580
550 V3=(V1-32)/1.8
560 IF F=0 THEN 580
570 V3=(V1*1.8)+32
580 V2=V3*(0.5+RND)
590 V2=INT(V2+0.5)
600 T=0
610 IF V2>=V3 THEN 630
620 T=1
630 GOSUB 410
640 PRINT "QUESTION";J;"OF";NQ
650 PRINT
660 IF F=1 THEN 700
670 PRINT "IS";V1;EP$(N);B$;R$(N);B$;"THAN"
680 PRINT B$;V2;MP$(N);B$;
690 GOTO 720
```

```
700 PRINT "IS";V1;MP$(N);B$;R$(N);B$;"THAN"
710 PRINT B$;V2;EP$(N);B$;
720 INPUT Q$
730 Q$=SEG$(Q$,1,1)
740 IF Q$="Y" THEN 800
750 IF Q$="N" THEN 840
760 PRINT
770 PRINT "YOU HAVE NO IDEA"
780 R=2
790 GOTO 870
800 PRINT
810 PRINT "YOU SAY 'YES'"
820 R=1
830 GOTO 870
840 PRINT
850 PRINT "YOU SAY 'NO'"
860 R=0
870 X=T-R
880 IF R<>2 THEN 910
890 GOSUB 990
900 GOTO 980
910 IF X<>0 THEN 960
920 PRINT
930 PRINT B$;B$;"AND YOU'RE RIGHT - GOOD!"
940 NR=NR+1
950 GOTO 980
960 PRINT B$;B$;"BUT YOU'RE WRONG"
970 GOSUB 990
980 RETURN
990 PRINT
1000 CALL HCHAR(24,3,45,28)
1010 PRINT
1020 PRINT
1030 IF F=1 THEN 1070
1040 PRINT V1;EP$(N);B$;"EQUALS"
1050 PRINT V3;B$;MP$(N)
1060 GOTO 1090
1070 PRINT V1;MP$(N);B$;"EQUALS"
1080 PRINT V3;B$;EP$(N)
1090 PRINT
1100 PRINT "--- THE RULE IS ---"
1110 PRINT
```

```
1120 IF N<>1 THEN 1180
1130 IF F=1 THEN 1160
1140 PRINT B$;"DEG.C = (DEG.F - 32)/1.8"
1150 RETURN
1160 PRINT B$;"DEG.F = (DEG.C * 1.8) + 32"
1170 RETURN
1180 IF F=1 THEN 1220
1190 PRINT B$;"1";B$;ES$(N);B$;"EQUALS"
1200 PRINT C(N);MP$(N)
1210 RETURN
1220 Q=INT(1.E5/C(N))/1.E5
1230 PRINT B$;"1";B$;MS$(N);B$;"EQUALS"
1240 PRINT Q;EP$(N)
1250 RETURN
1260 PRINT
1270 PRINT
1280 PRINT "HIT ANY KEY TO CONTINUE"
1290 CALL KEY(0,KK,ST)
1300 IF ST=0 THEN 1290
1310 RETURN
1320 DATA DEGREE FAHRENHEIT,DEGREE CENTIGRADE,
HOTTER,0.5
1330 DATA DEGREES FAHRENHEIT,DEGREES CENTIGRAD
E
1340 DATA MILE PER HOUR,KILOMETER PER HOUR,FAS
TER,1.60935
1350 DATA MILES PER HOUR,KILOMETERS PER HOUR
1360 DATA FOOT,METER,LONGER,0.3048
1370 DATA FEET,METERS
1380 DATA MILE,KILOMETER,LONGER,1.60935
1390 DATA MILES,KILOMETERS
1400 DATA INCH,CENTIMETER,LONGER,2.54
1410 DATA INCHES,CENTIMETERS
1420 DATA GALLON,LITRE,MORE,3.78533
1430 DATA GALLONS,LITRES
1440 DATA POUND,KILOGRAM,HEAVIER,0.45359
1450 DATA POUNDS,KILOGRAMS
1999 DATA XXX,XXX,XXX,0,XXX,XXX
```

EASY CHANGES

1. To have the program always ask a fixed number of questions, change line 200 to set NQ to the desired value. Also make lines 180 and 190 REM statements. For example:

```
180 REM
190 REM
200 NQ=10
```

   will cause the program to do 10 questions.

2. There are currently seven conversions built into the program:

N	*Type*	*English Unit*	*Metric Unit*
1	temperature	degrees F.	degrees C.
2	speed	miles/hour	kilometers/hour
3	length	feet	meters
4	length	miles	kilometers
5	length	inches	centimeters
6	volume	gallons	litres
7	weight	pounds	kilograms

   If you wish to be quizzed on only one type of question, set N to this value in line 460. Thus,

```
460 N=4
```

   will cause the program to only produce questions comparing miles and kilometers. To add additional data to the program, see the first "Suggested Project."

3. You can easily have the questions posed in one "direction" only. To go only from English to metric units use

```
470 F=0
480 REM
490 REM
```

   while to go from metric to English use

```
470 F=1
480 REM
490 REM
```

4. You might want the converted value and governing rule to be displayed even when the correct answer is given. This is accomplished by adding line 945 as follows:

```
945 GOSUB 990
```

MAIN ROUTINES

120– 150	Dimensions and initializes variables.
160– 330	Mainline routine, drives other routines.
340– 400	Reads and initializes data.
410– 450	Displays header.
460– 980	Forms and asks questions. Processes user's reply.
990–1250	Displays exact conversion and governing rule.
1260–1310	Waits for user to hit any key.
1320–1999	DATA statements.

MAIN VARIABLES

ND	Number of conversions in the data.
ES$, EP$	String arrays of English units' names (singular, plural).
MS$, MP$	String arrays of metric units' names (singular, plural).
R$	String array of the relation descriptors.
C	Array of the conversion factors.
Q	Work variable.
B$	String constant of one blank character.
J	Current question number.
NR	Number of questions answered right.
P	Percentage answered right.
NQ	Number of questions in session.
N	Index number of current question in the data list.
F	Flag on question "direction" (0=English to metric; 1=metric to English).
V1, V2	Numeric values on left, right sides of the question.
V3	The correct value of the right-hand side.
T	Flag on the question's correct answer (1=true; 0=false).
R	User reply flag (0=no; 1=yes; 2=no idea).
X	User's result (0 if correct answer was given).
Q$	Input string.
KK, ST	User input.

SUGGESTED PROJECTS

1. Each built-in conversion requires six elements of data in this order:

Element	*Data Description*
1	English unit (singular)
2	Metric unit (singular)

3	Relation descriptor (e.g., "hotter," "faster," etc.)
4	Conversion factor (from English to metric)
5	English unit (plural)
6	Metric unit (plural)

Each of these elements, except the fourth, is a string. The DATA statements in the listing should make clear how the information is to be provided. You can add new data to the program with appropriate DATA statements in this format. New data should be added after the current data; i.e., just before line 1999. Line 1999 is a special data statement to trigger the end of all data to the program. The program is dimensioned up to thirty entries while only seven are currently used. (Note: this format allows only conversions where one unit is a direct multiple of the other. Temperature, which does not fit this rule, is handled as a special case throughout the program.)

2. Convert the program to handle units conversion questions of any type.
3. Keep track of the questions asked and which ones were missed. Then, do not ask the same questions too soon if they have been answered correctly. However, do re-ask those questions missed for additional practice.

NUMBERS

PURPOSE

This is an educational program for preschool children. After a few weeks watching Sesame Street on television, most three and four-year-old children will learn how to count from one to ten. The NUMBERS program allows these children to practice their numbers and have fun at the same time.

HOW TO USE IT

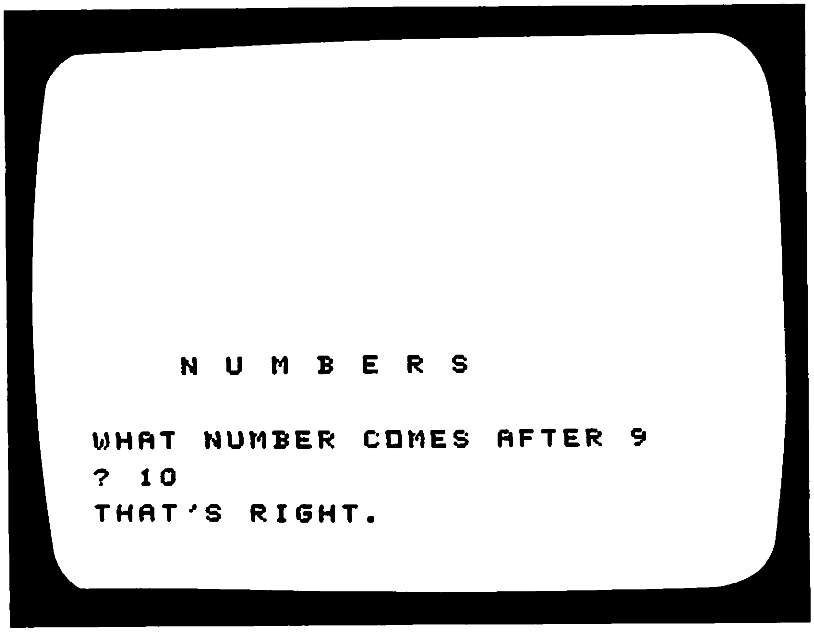
We know a child who learned how to type RUN to get this program started before she turned three, but you'll probably have to help your child with this for a while. The program asks the question, "WHAT NUMBER COMES AFTER n?", where n is a number from one to nine. Even if the child can't read yet, he or she will soon learn to look for the number at the end of the line. The child should respond with the appropriate number, and then press the **ENTER** key.

If the answer is correct, the program displays the message "THAT'S RIGHT!", pauses for a couple of seconds, and then displays three geometric shapes. In the upper left of the screen a rectangle is drawn. In the lower center, a triangle is drawn. Then an asterisk (or a snowflake, perhaps?) is drawn in the lower right portion of the screen. After a short delay, the program clears the screen and asks another question. The same number is never asked twice in a row.

If the child provides the wrong answer, a message indicates the error and the same question is asked again.

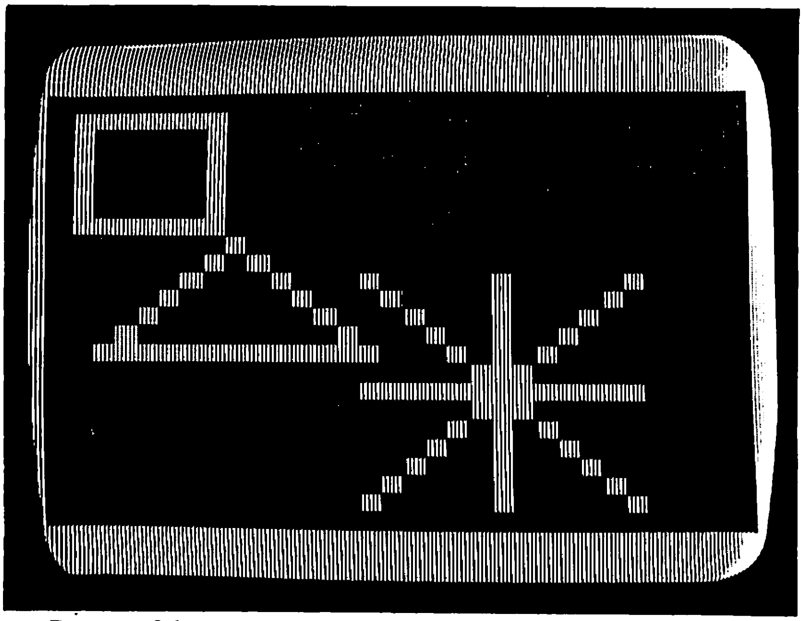
The program keeps on going until you enter "E" instead of a number. Remember that most children have a pretty short attention span, so please do not force your child to continue after his or her interest diminishes. Keep each session short and fun. This way, it will always be a treat to "play" with the computer.

SAMPLE RUN



```
        N U M B E R S

WHAT NUMBER COMES AFTER 9
? 10
THAT'S RIGHT.
```

The program asks what number comes after 9, and waits for a response. The operator says 10, and the program acknowledges that the answer is correct.


Because of the correct response, the program draws three geometric figures.

PROGRAM LISTING

```
100 REM NUMBERS
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 M=9
140 E=10
150 TS=8
160 RANDOMIZE
170 PRINT TAB(5);"N U M B E R S"
180 PRINT
190 PRINT
200 PRINT
210 R=INT(M*RND)+1
220 IF R=P THEN 210
230 PRINT "WHAT NUMBER COMES AFTER";R
240 PRINT
250 INPUT R$
260 IF LEN(R$)=0 THEN 250
270 IF R$<>"E" THEN 300
280 CALL CLEAR
```

```
290 END
300 IF R$<="/" THEN 330
310 IF R$>"9" THEN 330
320 IF VAL(R$)=R+1 THEN 380
330 PRINT
340 PRINT "NO, THAT'S NOT IT."
350 PRINT "TRY AGAIN."
360 PRINT
370 GOTO 230
380 PRINT
390 PRINT "THAT'S RIGHT."
400 FOR J=1 TO 500
410 NEXT J
420 P=R
430 E=6
440 C=INT(11*RND)+2
450 CALL CLEAR
460 CALL COLOR(1,C,C)
470 CALL HCHAR(2,2,31,E)
480 CALL VCHAR(2,2+E,31,E)
490 CALL HCHAR(2+E,2,31,E+1)
500 CALL VCHAR(2,2,31,E)
510 CALL COLOR(2,C+1,C+1)
520 FOR J=1 TO E
530 Y=TS+J
540 X=TS+J
550 CALL HCHAR(Y,X,40)
560 NEXT J
570 FOR J=1 TO E
580 Y=TS+J
590 X=TS-J+2
600 CALL HCHAR(Y,X,40)
610 NEXT J
620 Y=TS+E+1
630 FOR X=TS-E+1 TO TS+E+1
640 CALL HCHAR(Y,X,40)
650 NEXT X
660 CALL COLOR(3,INT((C+1)/2),INT((C+1)/2))
670 B=23-E
680 A=27-E
690 FOR J=1 TO E
700 X=A+J
```

```
710 Y=B+J
720 GOSUB 970
730 Y=B-J
740 GOSUB 970
750 Y=B
760 GOSUB 970
770 X=A
780 GOSUB 970
790 Y=B+J
800 GOSUB 970
810 Y=B-J
820 GOSUB 970
830 X=A-J
840 GOSUB 970
850 Y=B
860 GOSUB 970
870 Y=B+J
880 GOSUB 970
890 NEXT J
900 FOR J=1 TO 1000
910 NEXT J
920 CALL CLEAR
930 CALL COLOR(1,2,1)
940 CALL COLOR(2,2,1)
950 CALL COLOR(3,2,1)
960 GOTO 210
970 CALL HCHAR(Y,X,48)
980 RETURN
```

EASY CHANGES

1. Change the range of numbers that the program asks by altering the value of M in line 130. For a beginner, use a value of 3 for M instead of 9. Later, increase the value of M to 5, and then 8.
2. Alter the delay after "THAT'S RIGHT!" is displayed by altering the value of 500 in statement 400. Double it to double the time delay, etc. The same can be done with the 1000 in line 900 to alter the delay after the figures are drawn.

MAIN ROUTINES

120–200	Initializes variables. Clears screen.
210–220	Picks random integer from 1 to M.

230–390	Asks question. Gets answer. Determines if right or wrong.
400–410	Delays about 1½ seconds.
420–510	Draws a rectangle.
520–650	Draws a triangle.
660–890	Draws an asterisk.
900–910	Delays a few seconds.
920–960	Clears screen. Goes back to ask next question.
970–980	Subroutine to plot point in asterisk.

MAIN VARIABLES

M	Maximum number that will be asked.
E	Edge length of geometric figures.
R	Random integer in range from 1 to M.
P	Previous number that was asked.
R$	Reply given by operator.
X, Y	Coordinates in CRT display.
TS	Triangle's starting location (top).
A,B	X,Y coordinate values.
J	Subscript variable.

SUGGESTED PROJECTS

1. Modify the program to ask the next letter of the alphabet. Use the ASC and CHR$ functions in picking a random letter from A to Y, and to check whether the response is correct or not.
2. Ask each number from 1 to M once (in a random sequence). At the end of the sequence, repeat those that were missed.
3. Add different shapes to the graphics display that is done after a correct answer. Try an octagon, a diamond, and a rectangle. Or, combine this program with one of the graphics display programs.

TACHIST

PURPOSE

This program turns your computer into a tachistoscope (tah-KISS-tah-scope). A tachistoscope is used in reading classes to improve reading habits and, as a result, improve reading speed. The program displays a word or phrase on the screen for a fraction of a second, then asks you what it was. With a little practice, you will find that you can read phrases that are displayed for shorter and shorter time periods.

HOW TO USE IT

The program starts off by displaying a brief introduction and waiting for you to press any key (except the fctn, ctrl, or shift keys, of course, or the E key, which ends the program).

After you press a key, the screen is blanked out except for a horizontal dashed line in the lower left-hand corner. After two seconds, a phrase is flashed on the screen replacing the dashed line. Then the screen is blanked again, and you are asked what the phrase was.

If you respond correctly, the next phrase is displayed for a shorter time period (half as long). If you respond incorrectly, the program shows you the correct phrase, and the next phrase is displayed for a longer period of time (twice as long).

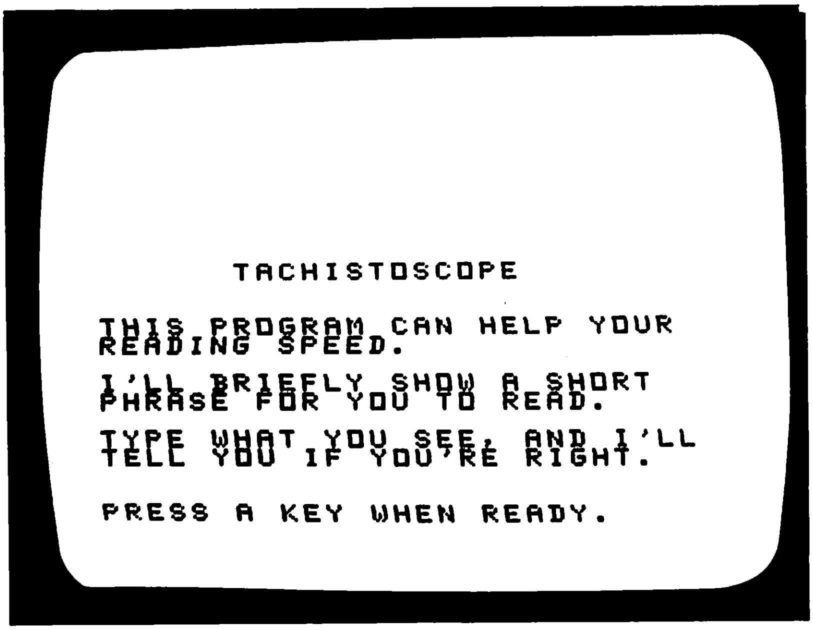
The fastest the computer can display a phrase and erase it is about .02 seconds (one-fiftieth). See if you can reach the top speed and still continue to read the phrases correctly.

A great deal of research has been done to determine how people read and what they should do to read both faster and with better comprehension. We will not try to explain it all (see the bibliography), but a couple of things are worth mentioning.

To read fast, you should not read one word at a time. Instead, you should learn to quickly read an entire phrase at once. By looking at a point in the center of the phrase (and slightly above it), your eyes can see the whole phrase *without* the necessity of scanning it from left to right, word by word. Because the tachistoscope flashes an entire phrase on the screen at once, it forces you to look at a single point and absorb the whole phrase, rather than scanning left to right, word by word.

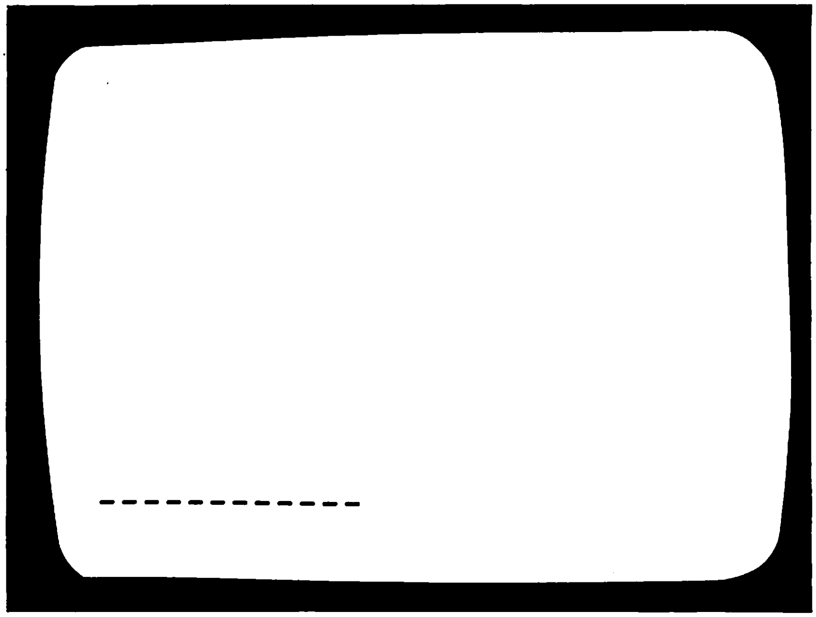
If you can incorporate this technique into your reading and increase the width of the phrases you absorb, your reading speed can increase dramatically.

SAMPLE RUN

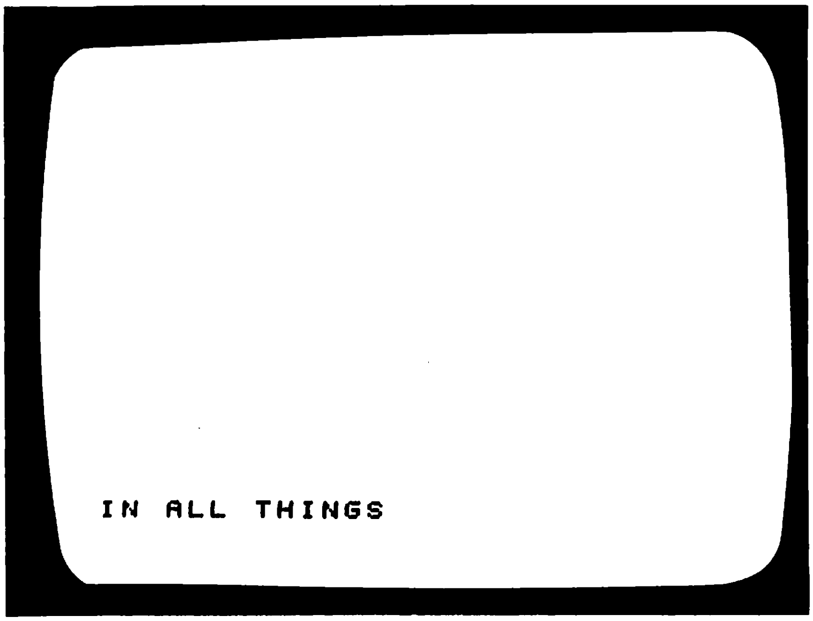


```
        TACHISTOSCOPE

THIS PROGRAM CAN HELP YOUR
READING SPEED.

I'LL BRIEFLY SHOW A SHORT
PHRASE FOR YOU TO READ.

TYPE WHAT YOU SEE, AND I'LL
TELL YOU IF YOU'RE RIGHT.

PRESS A KEY WHEN READY.
```

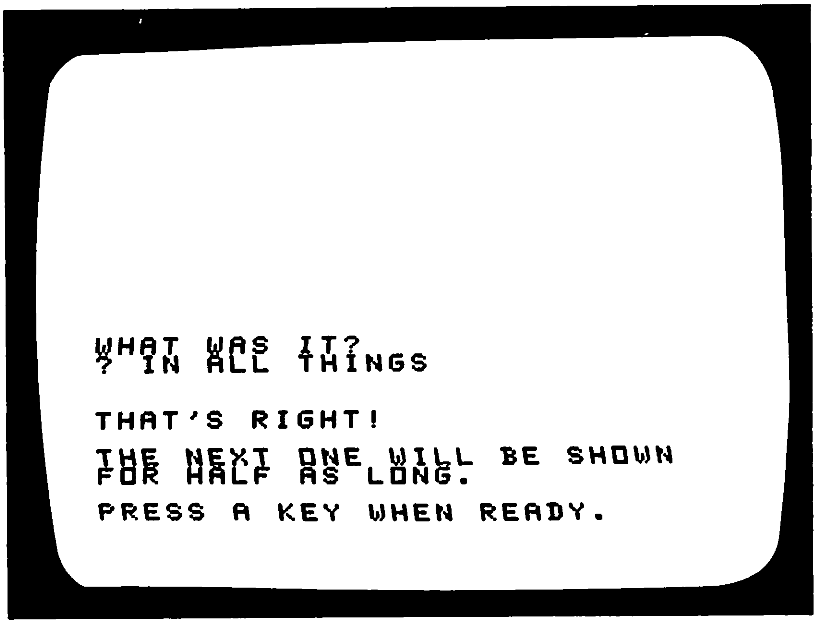
The program displays an introduction, then waits.


```
-------------
```

The program clears the screen and displays a horizontal line that remains in the lower left corner of the screen for a couple of seconds.


```
IN ALL THINGS
```

The program flashes a short phrase (chosen at random) where the line was for a fraction of a second, then clears the screen.



```
WHAT WAS IT?
? IN ALL THINGS

THAT'S RIGHT!
THE NEXT ONE WILL BE SHOWN
FOR HALF AS LONG.
PRESS A KEY WHEN READY.
```

The program asks what the phrase was. The operator responds correctly. The program acknowledges the correct response, and indicates that the next phrase will be shown for half as long.

PROGRAM LISTING

```
100 REM TACHIST
110 REM (C) 1984 DILITHIUM PRESS
120 RANDOMIZE
130 CALL CLEAR
140 T=128
150 L=50
160 DIM T$(50)
170 C=0
180 READ R$
190 IF R$="XXX" THEN 260
200 C=C+1
210 IF C<=L THEN 240
220 PRINT "TOO MANY DATA STATEMENTS."
230 END
240 T$(C)=R$
250 GOTO 180
260 CALL CLEAR
270 PRINT TAB(7);"TACHISTOSCOPE"
```

```
280 PRINT
290 PRINT
300 PRINT "THIS PROGRAM CAN HELP YOUR"
310 PRINT "READING SPEED."
320 PRINT
330 PRINT "I'LL BRIEFLY SHOW A SHORT"
340 PRINT "PHRASE FOR YOU TO READ."
350 PRINT
360 PRINT "TYPE WHAT YOU SEE, AND I'LL"
370 PRINT "TELL YOU IF YOU'RE RIGHT."
380 PRINT
390 PRINT
400 PRINT "PRESS A KEY WHEN READY."
410 CALL KEY(0,K,R)
420 IF R=0 THEN 410
430 IF K<>69 THEN 450
440 END
450 R=INT(C*RND)+1
460 IF R=P1 THEN 450
470 IF R=P2 THEN 450
480 IF R=P3 THEN 450
490 IF R=P4 THEN 450
500 IF R=P5 THEN 450
510 GOSUB 900
520 FOR K=1 TO 700
530 NEXT K
540 CALL CLEAR
550 PRINT T$(R)
560 FOR J=1 TO T
570 NEXT J
580 CALL CLEAR
590 FOR K=1 TO 400
600 NEXT K
610 PRINT "WHAT WAS IT?"
620 INPUT R$
630 IF R$<>T$(R)THEN 800
640 PRINT
650 PRINT "THAT'S RIGHT!"
660 T=T/2
670 R$="FOR HALF AS LONG."
680 P1=P2
690 P2=P3
```

```
700 P3=P4
710 P4=P5
720 P5=R
730 IF T>4 THEN 760
740 T=4
750 R$="AT MAXIMUM SPEED."
760 PRINT
770 PRINT "THE NEXT ONE WILL BE SHOWN"
780 PRINT R$
790 GOTO 390
800 PRINT
810 PRINT "NO, IT WAS"
820 PRINT T$(R)
830 T=T*2
840 IF T<1024 THEN 880
850 T=1024
860 R$="AT THE SAME SPEED."
870 GOTO 680
880 R$="FOR TWICE AS LONG."
890 GOTO 680
900 CALL CLEAR
910 PRINT "-----------"
920 RETURN
930 DATA AT THE TIME
940 DATA THE BROWN COW
950 DATA LOOK AT THAT
960 DATA IN THE HOUSE
970 DATA THIS IS MINE
980 DATA SHE SAID SO
990 DATA THE BABY CRIED
1000 DATA TO THE STORE
1010 DATA READING IS FUN
1020 DATA HE GOES FAST
1030 DATA IN ALL THINGS
1040 DATA GREEN GRASS
1050 DATA TWO BIRDS FLY
1060 DATA LATE LAST NIGHT
1070 DATA THEY ARE HOME
1080 DATA ON THE PHONE
1090 DATA THROUGH A DOOR
1100 DATA WE CAN TRY
```

```
1110 DATA MY FOOT HURTS
1120 DATA HAPPY NEW YEAR
9999 DATA XXX
```

EASY CHANGES

1. Change the phrases that are displayed by changing the DATA statements that start at line 930. Add more and/or replace those shown with your own phrases or words. Lines 150 and 160 must specify a number that is at least as large as the number of DATA statements. So, to allow for up to 100 DATA statements, change lines 150 and 160 to say

    ```
    150 L=100
    160 DIM T$(100)
    ```

    Be sure to enter your DATA statements in the same form as shown in the program listing. To begin with, you may want to start off with shorter phrases or single words. Later, try longer phrases. Do not alter line 9999, which has to be the last DATA statement.
2. To change the length of time the first phrase is displayed, change the value of T in line 140. Double it to double the length of time, etc. Don't make it less than 4.
3. To cause all phrases to be displayed for the same length of time, remove lines 660 and 830, and insert these lines:

    ```
    675 R$="AT THE SAME SPEED"
    835 R$="AT THE SAME SPEED"
    836 GOTO 680
    ```

4. If you want to change the waiting period before the phrase is flashed on the screen, change the 700 in line 520. To make the delay five seconds, change it to 2000. To make it one second, change it to 300.
5. To put the program into a sort of flashcard mode, in which the phrases are flashed, but no replies are necessary, insert these three lines:

    ```
    605 GOTO 800
    675 R$="AT THE SAME SPEED"
    825 GOTO 675
    ```

    This will cause each phrase to be flashed (all for the same length of time), and then to be displayed again so you can verify what it was.

MAIN ROUTINES

120–160	Initializes variables.
170–250	Reads DATA statements into T$ array.
260–380	Displays introduction.
390–440	Waits for operator to press a key.
450–500	Picks random phrase from T$ array. Ensures no duplication from previous five phrases.
510	Clears screen and displays horizontal line.
520–600	Displays phrase for appropriate length of time.
610–620	Asks what phrase was.
630	Determines if typed phrase matches the phrase displayed.
640–790	Shortens time for next phrase if reply was correct. Saves subscript to avoid repetition. Goes back to wait for key to be pressed.
800–890	Shows what phrase was. Lengthens time for next phrase. Ensures that time period does not exceed maximum.
900–920	Subroutine to display horizontal dash line.
930–9999	DATA statements with phrases to be displayed.

MAIN VARIABLES

T	Time that phrase will be displayed.
J	Loop variable.
L	Limit of number of phrases.
T$( )	Array of phrases (read into from DATA statements).
C	Count of number of phrases actually read.
R$	Temporary string variable. Also, reply of operator.
R	Work variable. Also subscript of phrase to be displayed.
P1, P2, P3, P4, P5	Subscripts of the five previous phrases.
K	Temporary work variable.

SUGGESTED PROJECTS

1. Instead of picking phrases at random, go through the list once sequentially. Insert line 265 to set R to zero, and line 450 to add one to R, then check if R is greater than C.
2. Instead of only verifying that the current phrase does not duplicate any of the previous five phrases, modify the program to avoid

duplication of the previous ten or more. Changes will be needed to lines 460–500 and 680–720.

3. Keep score of the number of correct and incorrect replies, and display the percentage each time. Alternatively, come up with a rating based on the percentage correct and the speed attained, possibly in conjunction with a difficulty factor for the phrases used.
4. Add the capability to the program to also have a mode in which it can display a two to seven digit number, chosen at random. Have the operator try several of the numbers first (maybe five digit ones) before trying the phrases. The phrases will seem easy after doing the numbers.

VOCAB

PURPOSE

Did you ever find yourself at a loss for words? Well, this vocabulary quiz can be used in a self-teaching environment or as reinforcement for classroom instruction to improve your ability to remember the jargon of any subject. If allows you to drill at your own pace, without the worry of ridicule from other students or judgment by an instructor. When you make mistakes, only the computer knows, and it's not telling anyone except you. Modifying the program to substitute a different vocabulary list is very simple, so you can accumulate many different versions of this program, each with a different set of words.

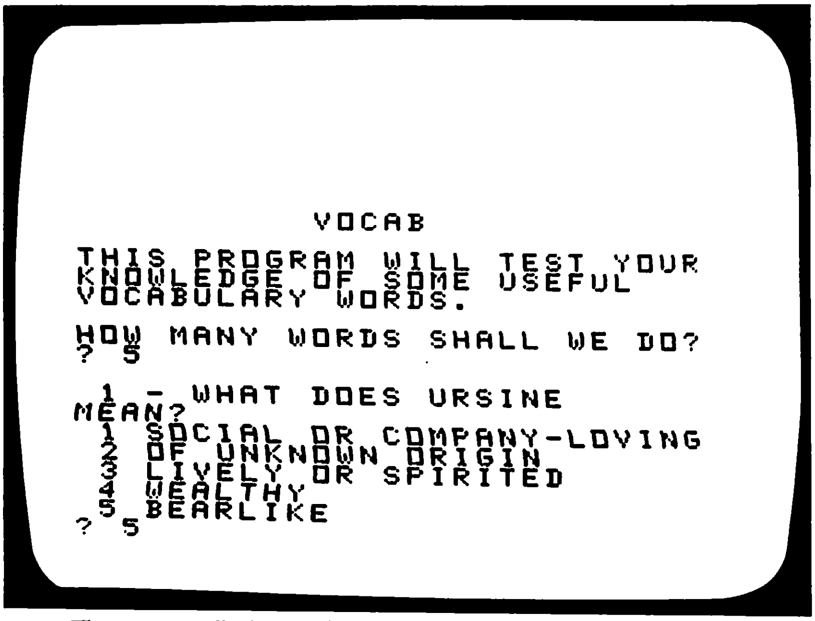
HOW TO USE IT

This program is pretty much self-explanatory from the sample run. After you enter "RUN," it asks you how many questions you would like. If you respond with a number less than five, you will still do five. Otherwise, you will do the number you enter.

Next, you get a series of multiple-choice questions. Each question is formatted in one of two ways—either you are given a word and asked to select from a list of definitions, or you are given a definition and asked to select from a list of words. The format is chosen at random. You respond with the number of the choice you think is correct. If you are right, you are told so. If not, you are shown the correct answer. From the second answer on, you are shown a status report of the number correct out of the number attempted so far.

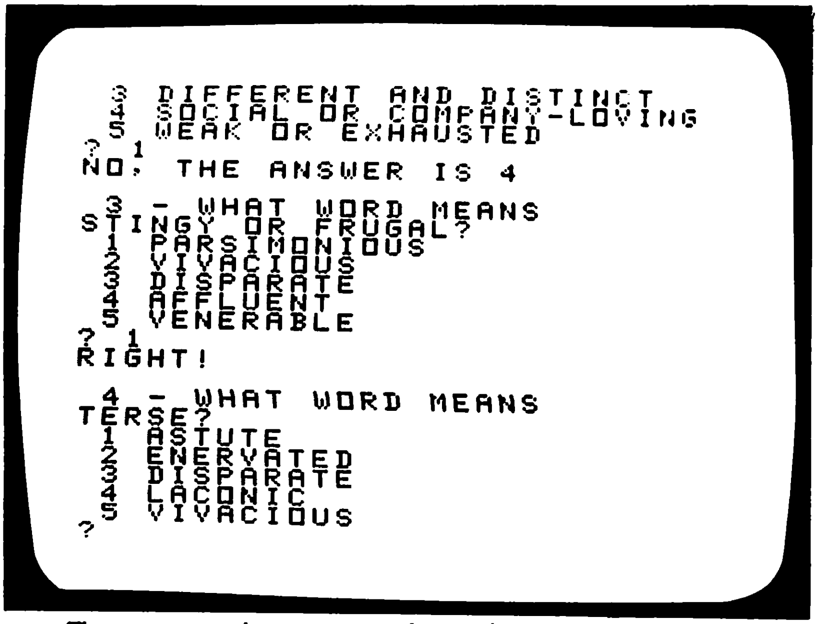
Finally, after the last question, you are shown the percentage you got correct, along with a comment on your performance. Then you have the option of going back for another round of questions or stopping.

SAMPLE RUN



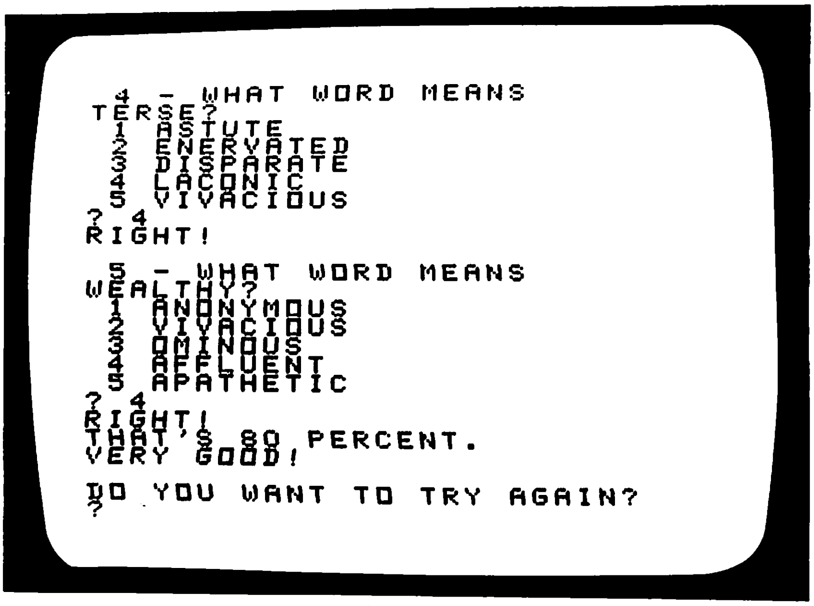
```
          VOCAB
THIS PROGRAM WILL TEST YOUR
KNOWLEDGE OF SOME USEFUL
VOCABULARY WORDS.

HOW MANY WORDS SHALL WE DO?
? 5

 1 - WHAT DOES URSINE
MEAN?
 1 SOCIAL OR COMPANY-LOVING
 2 OF UNKNOWN ORIGIN
 3 LIVELY OR SPIRITED
 4 WEALTHY
 5 BEARLIKE
? 5
```

The program displays an introduction and asks the first question. The operator replies.



```
 3 DIFFERENT AND DISTINCT
 4 SOCIAL OR COMPANY-LOVING
 5 WEAK OR EXHAUSTED
? 1
NO, THE ANSWER IS 4

 3 - WHAT WORD MEANS
STINGY OR FRUGAL?
 1 PARSIMONIOUS
 2 VIVACIOUS
 3 DISPARATE
 4 AFFLUENT
 5 VENERABLE
? 1
RIGHT!

 4 - WHAT WORD MEANS
TERSE?
 1 ASTUTE
 2 ENERVATED
 3 DISPARATE
 4 LACONIC
 5 VIVACIOUS
?
```

The operator continues to answer the questions, and the program indicates right and wrong responses.



```
  4 - WHAT WORD MEANS
TERSE?
 1 ASTUTE
 2 ENERVATED
 3 DISPARATE
 4 LACONIC
 5 VIVACIOUS
? 4
RIGHT!

  5 - WHAT WORD MEANS
WEALTHY?
 1 ANONYMOUS
 2 VIVACIOUS
 3 OMINOUS
 4 AFFLUENT
 5 APATHETIC
? 4
RIGHT!
THAT'S 80 PERCENT.
VERY GOOD!

DO YOU WANT TO TRY AGAIN?
?
```

At the end of five questions, the program gives a final score and asks about trying again.

PROGRAM LISTING

```
100 REM VOCAB
110 REM (C) 1984 DILITHIUM PRESS
120 RANDOMIZE
130 GOSUB 210
140 GOSUB 370
150 GOSUB 550
160 GOSUB 700
170 GOSUB 850
180 GOSUB 960
190 IF E=0 THEN 150
200 GOTO 130
210 IF E<>0 THEN 290
220 CALL CLEAR
230 PRINT TAB(11);"VOCAB"
240 PRINT
250 PRINT "THIS PROGRAM WILL TEST YOUR"
260 PRINT "KNOWLEDGE OF SOME USEFUL"
270 PRINT "VOCABULARY WORDS."
280 PRINT
290 PRINT "HOW MANY WORDS SHALL WE DO?"
```

```
300 INPUT L
310 L=INT(ABS(L))
320 IF L>4 THEN 360
330 PRINT "THAT'S NOT ENOUGH."
340 PRINT "LET'S DO FIVE."
350 L=5
360 RETURN
370 IF E<>0 THEN 510
380 C=5
390 D=26
400 DIM D$(26),E$(26)
410 DIM P(5)
420 J=1
430 READ D$(J)
440 IF D$(J)="XXX" THEN 500
450 READ E$(J)
460 J=J+1
470 IF J<=D THEN 430
480 PRINT "TOO MANY DATA STATEMENTS."
490 PRINT "ONLY FIRST";D;"ARE USED."
500 D=J-1
510 Q=1
520 E=0
530 Q1=0
540 RETURN
550 FOR J=1 TO C
560 P(J)=0
570 NEXT J
580 FOR J=1 TO C
590 PP=INT(D*RND)+1
600 IF PP=P1 THEN 590
610 IF PP=P2 THEN 590
620 IF PP=P3 THEN 590
630 FOR K=1 TO J
640 IF P(K)=PP THEN 590
650 NEXT K
660 P(J)=PP
670 NEXT J
680 A=INT(C*RND)+1
690 RETURN
700 PRINT
```

```
710 M=RND
720 IF M>.5 THEN 790
730 PRINT Q;"- WHAT WORD MEANS"
740 PRINT E$(P(A));"?"
750 FOR J=1 TO C
760 PRINT J;D$(P(J))
770 NEXT J
780 GOTO 840
790 PRINT Q;"- WHAT DOES ";D$(P(A))
800 PRINT "MEAN?"
810 FOR J=1 TO C
820 PRINT J;E$(P(J))
830 NEXT J
840 RETURN
850 INPUT R
860 IF R=A THEN 890
870 PRINT "NO, THE ANSWER IS";A
880 GOTO 910
890 PRINT "RIGHT!"
900 Q1=Q1+1
910 IF Q=1 THEN 920
920 P3=P2
930 P2=P1
940 P1=P(A)
950 RETURN
960 Q=Q+1
970 IF Q>L THEN 990
980 RETURN
990 E=1
1000 Q=Q1*100/(Q-1)
1010 IF Q>0 THEN 1040
1020 PRINT "WELL THAT'S A PERFECT SCORE"
1030 GOTO 1190
1040 PRINT "THAT'S";Q;"PERCENT."
1050 IF Q>25 THEN 1080
1060 PRINT "CONGRATULATIONS ON AVOIDING A SHUT
OUT."
1070 GOTO 1190
1080 IF Q>50 THEN 1120
1090 PRINT "YOU CAN USE SOME MORE"
1100 PRINT "PRACTICE."
```

```
1110 GOTO 1190
1120 IF Q>75 THEN 1160
1130 PRINT "NOT BAD, BUT ROOM FOR"
1140 PRINT "IMPROVEMENT."
1150 GOTO 1190
1160 PRINT "VERY GOOD!"
1170 IF Q<=95 THEN 1190
1180 PRINT "YOU'RE ALMOST AS SMART AS ME"
1190 PRINT
1200 PRINT "DO YOU WANT TO TRY AGAIN?"
1210 INPUT R$
1220 IF SEG$(R$,1,1)<>"N" THEN 1260
1230 PRINT
1240 PRINT "CHECK YOU LATER."
1250 END
1260 IF SEG$(R$,1,1)<>"Y" THEN 1190
1270 CALL CLEAR
1280 RETURN
1290 REM ******NOTE******
1300 REM LINES 390-400: VALUES MUST BE
1310 REM AT LEAST ONE GREATER THAN THE
1320 REM NUMBER OF DIFFERENT WORDS.
1330 REM ****************
1340 DATA ANONYMOUS,OF UNKNOWN ORIGIN
1350 DATA OMINOUS,THREATENING OR MENACING
1360 DATA AFFLUENT,WEALTHY
1370 DATA APATHETIC,"INDIFFERENT, UNINTERESTED
"
1380 DATA LACONIC,TERSE
1390 DATA INTREPID,FEARLESS OR COURAGEOUS
1400 DATA GREGARIOUS,SOCIAL OR COMPANY-LOVING
1410 DATA ENERVATED,WEAK OR EXHAUSTED
1420 DATA VENERABLE,WORTHY OF RESPECT
1430 DATA DISPARATE,DIFFERENT AND DISTINCT
1440 DATA VIVACIOUS,LIVELY OR SPIRITED
1450 DATA ASTUTE,KEEN IN JUDGMENT
1460 DATA URSINE,BEARLIKE
1470 DATA PARSIMONIOUS,STINGY OR FRUGAL
1480 DATA OMNISCIENT,ALL-KNOWING
9999 DATA XXX
```

EASY CHANGES

1. Add more DATA statements statements between lines 1340 and 9999, or replace them all with your own. Be careful not to use two or more words with very similar definitions; the program might select more than one of them as possible answers to the same question. Note that each DATA statement first has the vocabulary word, then a comma, and then the definition or synonym. Be sure there are no commas or colons in the definition (unless you enclose the definition in quotes). If you add more DATA statements, you have to increase the values of the numbers in lines 390 and 400 to be at least one greater than the number of words. The number of DATA statements you can have depends on how long each one is and how much user memory your computer has. Using DATA statements that average the same length as these, you can probably have about 200 of them if you have a 16 K computer. Be sure to leave statement 9999 as it is—it signals that there are no more DATA statements.
2. If you do not want to be given a choice of how many questions are going to be asked, remove lines 290 through 350 and insert the following lines:

```
290 PRINT "WE'LL DO TEN QUESTIONS."
300 L=10
```

   This will always cause ten questions to be asked. Of course, you can use some number other than ten if you want.

MAIN ROUTINES

120- 200	Mainline routine. Calls major subroutines.
210- 360	Displays introduction. Determines number of questions to be asked.
370- 540	Reads vocabulary words and definitions into arrays. Performs housekeeping.
550- 690	Selects choices for answers and determines which will be the correct one.
700- 840	Determines in which format the question will be asked. Asks it.
850- 950	Accepts answer from operator. Determines if right or wrong. Keeps score. Saves subscripts of last three correct answers.

960–1280	Gives final score. Asks about doing it again.
1340–9999	DATA statements with vocabulary words and definitions.

MAIN VARIABLES

E	Set to 1 to avoid repeating introduction after the first round.
L	Limit of number of questions to ask.
R	Operator's reply to each question.
C	Number of choices of answers given for each question.
D	At least one greater than number of DATA statements.
D$	Array of vocabulary words.
E$	Array of definitions.
P	Array for numbers of possible answers to each question.
J	Work variable (subscript for FOR-NEXT loops).
Q	Number of questions asked so far (later used to calculate percent correct).
Q1	Number of questions correct so far.
PP	Work variable.
P1, P2, P3	Last three correct answers.
A	Subscript of correct answer in P array.
M	Work variable to decide which way to ask question.
R$	Yes or no reply about doing another round.

SUGGESTED PROJECTS

1. Modify lines 1010 through 1180 to display the final evaluation messages based on a finer breakdown of the percent correct. For example, show one message if 100 percent, another if 95 to 99, another if 90 to 94, etc.
2. Ask the operator's name in the introduction routine, and personalize some of the messages with his/her name.
3. Instead of just checking about the last three questions, be sure that the next question has not been asked in the last eight or ten questions. (Check lines 600–620 and 920–940.)
4. Keep track of which questions the operator misses. Then, after going through the number of questions he/she requested, repeat those that were missed.

Section 3

Game Programs

Almost everyone likes to play games. Computer games are a fun and entertaining use of your TI-99/4A computer. Besides providing relaxation and recreation, they have some built-in practical bonuses. They often force you to think strategically, plan ahead, or at least be orderly in your thought processes. They are also a good way to help some friends over their possible "computer phobia." We present a collection of games to fit any game-playing mood.

Maybe you desire a challenging all-skill game? Like chess or checkers, WARI involves no luck and considerable thinking. The computer will be your opponent, and a formidable one indeed.

Perhaps you're in the mood for a game with quick action and mounting excitement. GROAN is a fast-paced dice game involving mostly luck with a dash of skill (or intuition) thrown in. The computer is ready for your challenge any time.

JOT is a word game. You and the TI-99/4A each take secret words and then try to home in on each other's selection.

Do you like solving puzzles? If so, try DECODE. The computer will choose a secret code and then challenge you to break it.

Graphic electronic arcade games have been a prevalent landmark of the past few years. We include two such games. ROADRACE puts you behind the wheel of a high-speed race car. You must steer accurately to stay on course. OBSTACLE lets you and a friend compete in a game of cut and thrust. Each of you must avoid crossing the path laid by the other, and by yourself!

DECODE

PURPOSE

Decode is really more of a puzzle than a game, although you can still compete with your friends to see who can solve the puzzles the fastest. Each time you play, you are presented with a new puzzle to solve.

The object is to figure out the computer's secret code in as few guesses as possible. The program gives you information about the accuracy of each of your guesses. By carefully selecting your guesses to make use of the information you have, you can determine what the secret code must be in a surprisingly small number of guesses. Five or six is usually enough.

The first few times you try, you will probably require quite a few more guesses than that, but with practice, you'll discover that you can learn a lot more from each guess than you originally thought.

HOW TO USE IT

The program starts off by displaying a brief introduction. Here are some more details.

The program selects a secret code for you to figure out. The code is a four-digit number that uses only the digits 1 through 6. For example, your TI-99/4A might pick 6153 or 2242 as a secret code.

Your object is to guess the code in the fewest possible guesses. After each of your guesses, the program tells you a "black" and a "white" number. The black number indicates the number of digits in your guess that were correct—the digit was correct *and* in the correct position. So, if the secret code is 6153 and your guess is 4143, you will be told that black is 2 (because the 1 and the 3 are correct). Of course, you aren't told *which* digits are correct. That is for you to figure out by making use of the information that you get from other guesses.

Each of the white numbers indicates a digit in your guess that is correct, but which is in the wrong position. For example, if the secret code is 6153 and your guess is 1434, you will be told that white is 2. The 1 and 3 are correct, but in wrong positions.

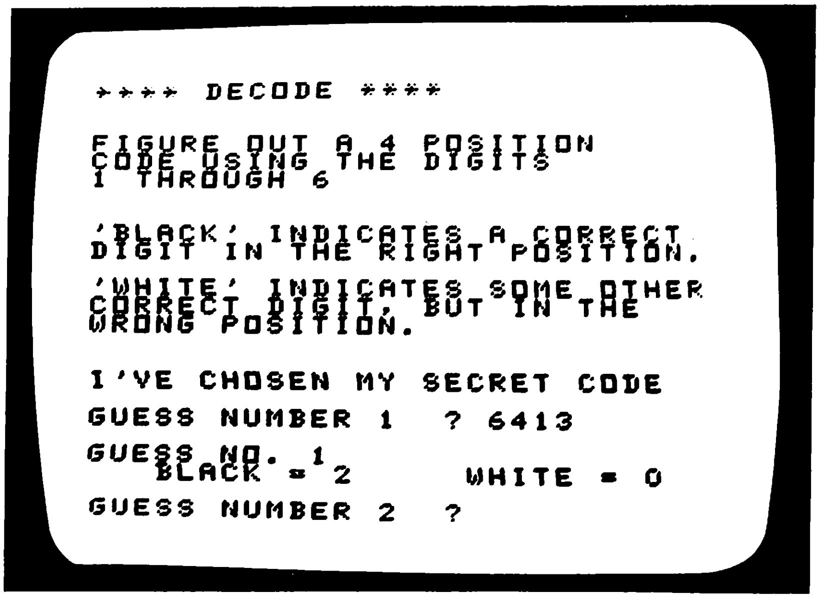
The white number is determined by ignoring any digits that accounted for a black number. Also, a single position in the secret code or guess can only account for one black or white number. These facts become significant when the secret code and/or your guess have duplicate digits. For example, if the code is 1234 and your guess is 4444, there is only one black, and no whites. If the code is 2244 and your guess is 4122, there are no blacks and three whites.

This may sound a little tricky, but you will quickly get the hang of it.

At any time during the game, you can ask for a "SUMMARY" by entering an S instead of a guess. This causes the program to clear the screen and display each guess (with the corresponding result) that has occurred so far.

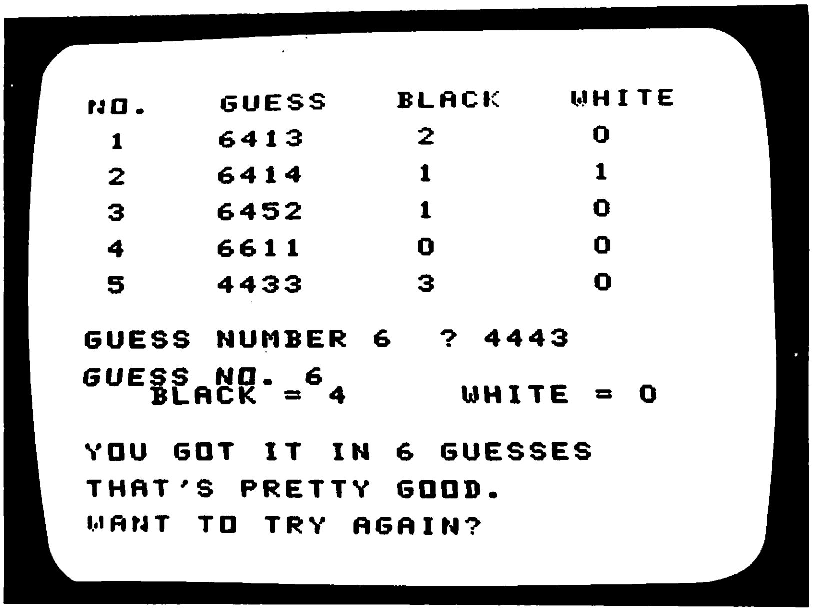
Also, if you get tired of trying and want to give up, you can enter a Q (for "quit") to end your misery and find out the answer. Otherwise, you continue guessing until you get the code right (four black, zero white), or until you have used up the maximum of 12 guesses.

SAMPLE RUN



```
**** DECODE ****

FIGURE OUT A 4 POSITION
CODE USING THE DIGITS
1 THROUGH 6

'BLACK' INDICATES A CORRECT
DIGIT IN THE RIGHT POSITION.

'WHITE' INDICATES SOME OTHER
CORRECT DIGIT, BUT IN THE
WRONG POSITION.

I'VE CHOSEN MY SECRET CODE
GUESS NUMBER 1   ? 6413
GUESS NO. 1
     BLACK = 2        WHITE = 0
GUESS NUMBER 2   ?
```

The program displays an introduction, chooses its secret code, and asks for the operator's first guess. The program responds with a "black" and a "white" number.



```
NO.     GUESS     BLACK     WHITE
 1      6413        2         0
 2      6414        1         1
 3      6452        1         0
 4      6611        0         0
 5      4433        3         0

GUESS NUMBER 6   ? 4443
GUESS NO. 6
     BLACK = 4        WHITE = 0

YOU GOT IT IN 6 GUESSES
THAT'S PRETTY GOOD.
WANT TO TRY AGAIN?
```

Later in the same game, the operator asks for a summary, then makes the guess that turns out to be correct.

PROGRAM LISTING

```
100 REM DECODE
110 REM (C) 1984 DILITHIUM PRESS
120 D=6
130 P=4
140 L=12
150 DIM G$(12),G(4),C(4),B(12),W(12)
160 GOSUB 1110
170 GOSUB 310
180 PRINT "GUESS NUMBER";GG;" ";
190 INPUT A$
200 IF SEG$(A$,1,1)="S" THEN 410
210 IF SEG$(A$,1,1)="Q" THEN 560
220 GOSUB 680
230 IF K=1 THEN 180
240 GOSUB 810
250 GOSUB 1040
260 IF B(GG)=P THEN 1300
270 G$(GG)=A$
280 GG=GG+1
290 IF GG>L THEN 1500
300 GOTO 180
310 GG=1
320 C$=""
330 FOR J=1 TO P
340 RANDOMIZE
350 R=INT(D*RND)+1
360 C$=C$&STR$(R)
370 NEXT J
380 PRINT "I'VE CHOSEN MY SECRET CODE"
390 PRINT
400 RETURN
410 IF GG<>1 THEN 440
420 PRINT "NO GUESSES YET."
430 GOTO 180
440 CALL CLEAR
450 PRINT "****SUMMARY****"
460 PRINT
470 PRINT "NO.   GUESS   BLACK   WHITE"
480 PRINT
490 FOR J=1 TO GG-1
```

```
500 PRINT J;TAB(7);G$(J);TAB(15);B(J);TAB(23);
W(J)
510 IF GG>=10 THEN 530
520 PRINT
530 NEXT J
540 PRINT
550 GOTO 180
560 PRINT
570 PRINT "CAN'T TAKE IT, HUH?"
580 PRINT
590 PRINT "WELL, MY CODE WAS ";
600 FOR J=1 TO 4
610 PRINT ". ";
620 FOR K=1 TO 500
630 NEXT K
640 NEXT J
650 PRINT C$
660 PRINT
670 GOTO 1440
680 K=0
690 IF LEN(A$)<>P THEN 780
700 FOR J=1 TO P
710 T=ASC(SEG$(A$,J,1))
720 IF T<49 THEN 780
730 IF T>57 THEN 780
740 R=VAL(SEG$(A$,J,1))
750 IF R>D THEN 780
760 NEXT J
770 RETURN
780 PRINT "ILLEGAL. TRY AGAIN."
790 K=1
800 RETURN
810 BB=0
820 WW=0
830 FOR J=1 TO P
840 G(J)=VAL(SEG$(A$,J,1))
850 C(J)=VAL(SEG$(C$,J,1))
860 IF G(J)<>C(J)THEN 900
870 BB=BB+1
880 G(J)=0
890 C(J)=0
900 NEXT J
```

```
910 FOR J=1 TO P
920 IF C(J)=0 THEN 1020
930 H=0
940 FOR K=1 TO P
950 IF C(J)=0 THEN 1000
960 IF C(J)<>G(K)THEN 1000
970 H=1
980 G(K)=0
990 C(J)=0
1000 NEXT K
1010 WW=WW+H
1020 NEXT J
1030 RETURN
1040 B(GG)=BB
1050 W(GG)=WW
1060 PRINT
1070 PRINT "GUESS NO.";GG
1080 PRINT TAB(4);"BLACK =";BB;TAB(18);"WHITE
=";WW
1090 PRINT
1100 RETURN
1110 CALL CLEAR
1120 PRINT "**** DECODE ****"
1130 PRINT
1140 PRINT
1150 PRINT "FIGURE OUT A";P;"POSITION"
1160 PRINT "CODE USING THE DIGITS"
1170 PRINT "1 THROUGH";D
1180 PRINT
1190 PRINT
1200 PRINT "'BLACK' INDICATES A CORRECT"
1210 PRINT "DIGIT IN THE RIGHT POSITION."
1220 PRINT
1230 PRINT "'WHITE' INDICATES SOME OTHER"
1240 PRINT "CORRECT DIGIT, BUT IN THE"
1250 PRINT "WRONG POSITION."
1260 PRINT
1270 PRINT
1280 RANDOMIZE
1290 RETURN
1300 PRINT
```

```
1310 PRINT "YOU GOT IT IN";GG;"GUESSES"
1320 IF GG>=5 THEN 1340
1330 B$="OUTSTANDING!"
1340 IF GG=5 THEN 1360
1350 IF GG<>6 THEN 1370
1360 B$="PRETTY GOOD."
1370 IF GG<>7 THEN 1390
1380 B$="NOT TOO GREAT."
1390 IF GG<8 THEN 1410
1400 B$="PRETTY WEAK."
1410 PRINT
1420 PRINT "THAT'S ";B$
1430 PRINT
1440 INPUT "WANT TO TRY AGAIN? ":A$
1450 IF SEG$(A$,1,1)="Y" THEN 160
1460 IF SEG$(A$,1,1)<>"N" THEN 1440
1470 PRINT
1480 PRINT "COWARD."
1490 END
1500 PRINT
1510 PRINT "THAT'S YOUR LIMIT OF"
1520 PRINT L;"GUESSES."
1530 PRINT
1540 PRINT "MY CODE WAS ";C$
1550 GOTO 1430
```

EASY CHANGES

1. Modify lines 120–150 to change the complexity of the code and/or the number of guesses you are allowed. For example, the following lines would allow fifteen guesses at a five-position code using the digits 1 through 8:

```
120 D=8
130 P=5
140 L=15
150 DIM G$(15), G(5), C(5), B(15), W(15)
```

The introduction will automatically reflect the new values for D and P. Be sure that neither D nor P is set greater than 9. Be sure to change the values in the DIM statement at line 150. If you change the value of P, you must also change the value for the arrays G and C. If you change the value of L, you must change the arrays G$, B, and W.

2. To change the program so that it will always display the "Summary" information after each guess automatically, replace line 300 with:

```
300 GOTO 410
```

MAIN ROUTINES

120– 170	Initializes variables. Displays introduction. Chooses secret code.
180– 250	Gets a guess from operator. Analyzes reply. Displays result.
260	Determines if operator guessed correctly.
270– 300	Saves guess. Adds one to guess-counter. Determines if limit on number of guesses was exceeded.
310– 400	Subroutine to initialize variables, choose secret code and inform operator.
410– 550	Subroutine to display summary of guesses so far.
560– 670	Subroutine to slowly display secret code when operator quits.
680– 800	Subroutine to determine if operator's guess was legal.
810–1030	Subroutine to determine number of black and white responses for the guess.
1040–1100	Subroutine to display number of black and white responses for the guess.
1110–1290	Subroutine to display title and introduction.
1300–1490	Subroutine to analyze operator's performance after correct answer is guessed and ask about playing again.
1500–1550	Subroutine to display secret code after operator exceeds limit of number of guesses.

MAIN VARIABLES

D	Number of possible digits in each position of the code (i.e., a digit from 1 to D).
P	Number of positions in the code.
L	Limit of number of guesses that can be made.
G$	Array in which guesses are saved.
G, C	Work arrays in which each guess is analyzed.
B, W	Arrays in which the number of black and white responses is saved for each guess.
R, H, T	Work variables.

GG	Counter of the number of guesses made.
A$	Reply by the operator.
C$	Secret code chosen by the program.
J, K	Loop variables.
BB, WW	Number of black and white responses for this guess.
B$	String with message about operator's performance.

SUGGESTED PROJECTS

1. Change the analysis at the end of the game to take into account the difficulty of the code as well as the number of guesses it took to figure the code out. A four-position code using the digits 1 through 6 has 1296 possibilities, but a five-position code using 1 through 8 has 32,768 possibilities. Change lines 1320 through 1400 to determine the message to be displayed based on the number of possibilities in the code as well as GG.
2. At the beginning of the game, give the operator the option of deciding the complexity of the code. Ask for the number of positions and the number of digits. Make sure only "reasonable" numbers are used—do not try to create a code with zero positions, for example. Another approach is to ask the operator if he/she wants to play the easy, intermediate, or advanced version. Then set the values of D and P accordingly. Suggestions are:

Easy:	D=3 and P=3
Intermediate	D=6 and P=4
Advanced:	D=8 and P=5

3. In addition to using the number of guesses to determine how well the operator did, keep track of the amount of time. This will require some logic to replace the INPUT in line 190. You'll have to build the A$ string one character at a time by doing CALL (0,KK,SS) to see if a key has been pressed. By counting the number of times through the loop when no key was pressed, you can "time" the operator.

GROAN

PURPOSE

Do you like the thrills of fast-paced dice games? If so, GROAN is right up your alley. It is a two-person game with the computer playing directly against you. There is a considerable amount of luck involved. However, the skill of deciding when to pass the dice to your opponent also figures prominently.

The TI-99/4A will roll the dice for both players, but don't worry—it will not cheat. (We wouldn't think of stooping to such depths.)

Why is the game called GROAN? You will know soon after playing it.

HOW TO USE IT

The game uses two dice. They are just like regular six-sided dice except for one thing. The die face where the "1" would normally be has a picture of a frowning face instead. The other five faces of each die have the usual numbers two through six on them.

The object is to be the first player to achieve a score agreed upon before the start of the game. Players alternate taking turns. A turn consists of a series of dice rolls (at least one roll, possibly several) subject to the following rules.

As long as no frown appears on either die, the roller builds a running score for this current series of rolls. After each roll with no frown, he has the choice of rolling again or passing the dice to his opponent. If he passes the dice, his score achieved on the current series is added to any previous total he may have had.

But if he rolls and a frown appears, he will be groaning. A frown on only one die cancels any score achieved for the current series of rolls. Any previous score is retained in this case. However, if he rolls a double frown, his entire previous total is wiped out as well as his current total. Thus, he reverts back to a total score of zero—true despair.

The program begins by asking what the winning score should be. Values between 50 and 100 tend to produce the best games, but any positive value less than 1000 is acceptable. Next, a simulated coin toss randomly decides who will get the first roll.

Each dice roll is portrayed with a short graphics display. The dice are shown rolling and then the outcome is displayed pictorially. During each roll, the TI-99/4A indicates who is rolling.

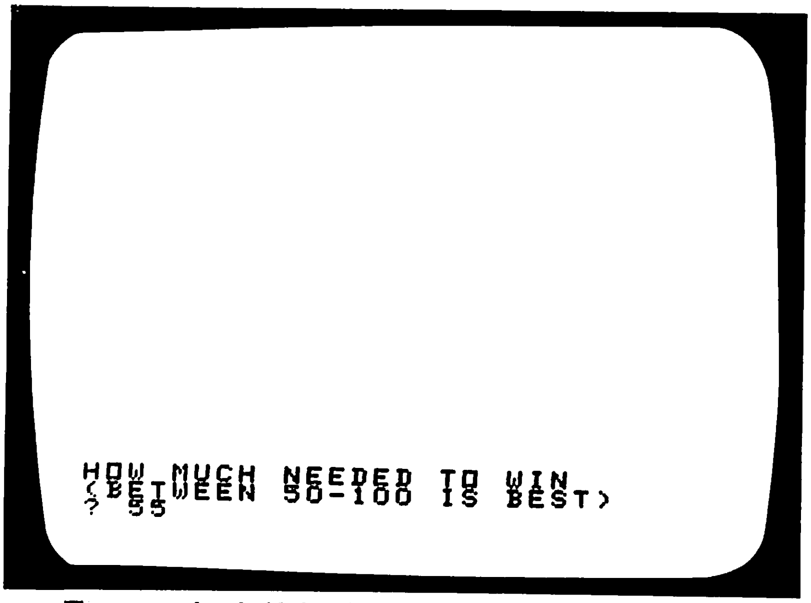
Each roll is followed by a display of the scoreboard. This scoreboard gives all relevant information: score needed to win, both players' scores before the current series of rolls, and the total score for the current series.

If a frown should appear on a die, the scoreboard will indicate the current running total as zero. In addition, the previous total will become zero in the case of the dreaded double frown. In either case, the dice will be passed automatically to the next player.

If a scoring roll results, the roller must decide whether to roll again or to pass the dice. The program has a built-in strategy to decide this for the TI-99/4A. For you, the question will be asked after the scoreboard is displayed. The two legal replies are **P** and **R**. The **R** means that you wish to roll again. The **P** means that you choose to pass the dice to the computer. If you should score enough to win, you must still pass the dice to add the current series to your previous total.

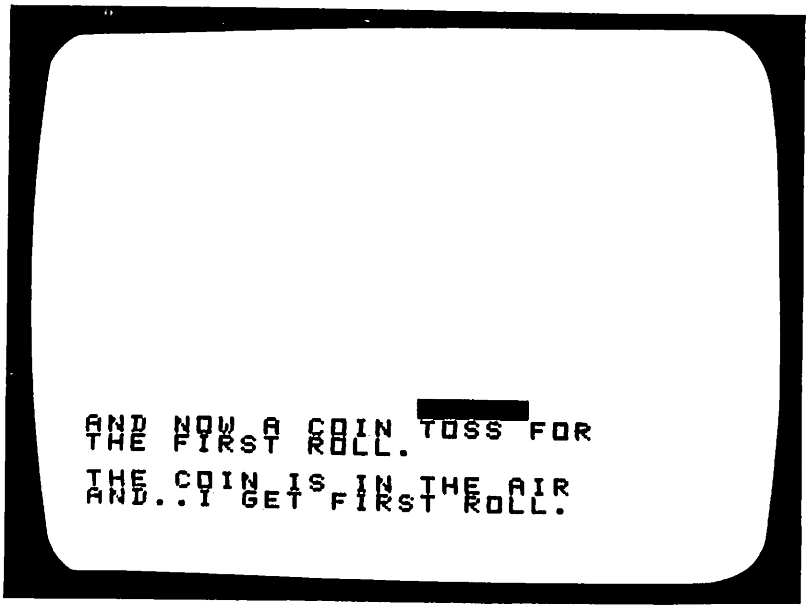
The first player to pass the dice with a score greater than or equal to the winning score is the victor. This will surely cause his opponent to GROAN. The computer will acknowledge the winner before signing off.

SAMPLE RUN



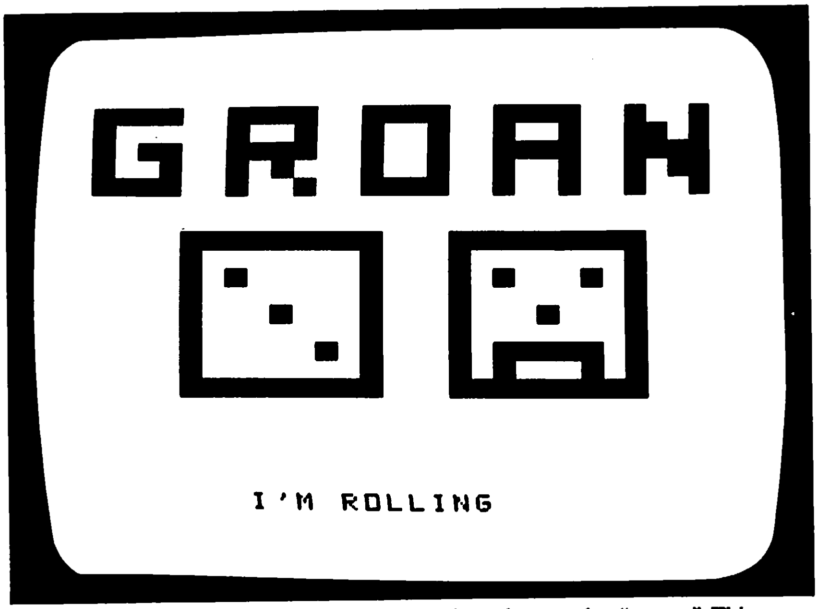
```
HOW MUCH NEEDED TO WIN
(BETWEEN 50-100 IS BEST)
? 55
```

The operator has decided to challenge the computer to a fifty-five-point game on GROAN.



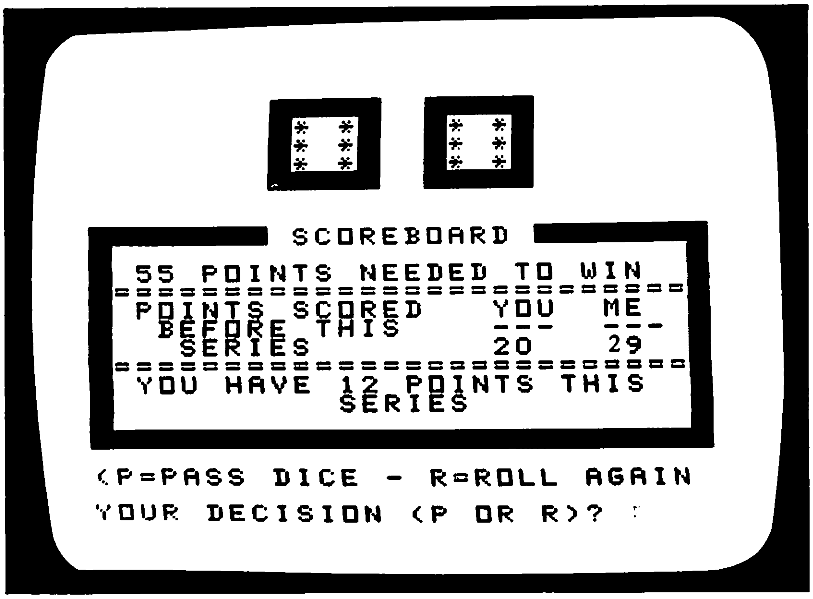
```
AND NOW A COIN TOSS FOR
THE FIRST ROLL.
THE COIN IS IN THE AIR
AND..I GET FIRST ROLL.
```

The computer wins the toss and gets the first dice roll.



```
GROAN

I'M ROLLING
```

The computer's roll, however, results in a three and a "groan." This scores no points and the dice pass to the operator.



```
              SCOREBOARD
 55 POINTS NEEDED TO WIN
===============================
 POINTS SCORED   YOU    ME
  BEFORE THIS    ---    ---
   SERIES        20     29
===============================
 YOU HAVE 12 POINTS THIS
          SERIES

<P=PASS DICE - R=ROLL AGAIN
YOUR DECISION <P OR R>?
```

Much later in the same game, the operator rolls a 12 to start a series of rolls. The score was TI–29, operator–20 before the roll. The operator must now decide whether to pass the dice or risk rolling again.

PROGRAM LISTING

```
100 REM GROAN
110 REM (C) 1984 DILITHIUM PRESS
120 A=0
130 H=0
140 T=0
150 B$=CHR$(32)
160 CC=128
170 CD=136
180 RANDOMIZE
190 CALL CLEAR
200 GOSUB 2140
210 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
220 CALL CHAR(136,"FFFFFFFFFFFFFFFF")
230 XS=3
240 YS=2
250 CALL COLOR(13,C,1)
260 GOSUB 2340
270 GOSUB 2110
280 GOSUB 2310
290 CALL CLEAR
300 PRINT "HOW MUCH NEEDED TO WIN"
310 PRINT "(BETWEEN 50-100 IS BEST)"
320 INPUT W
330 W=INT(W)
340 IF W<1 THEN 290
350 IF W>999 THEN 290
360 GOSUB 2140
370 GOSUB 2580
380 IF Q>0.5 THEN 600
390 T=0
400 GOSUB 2970
410 T=T+R1+R2
420 IF F=0 THEN 440
430 T=0
440 IF F<2 THEN 460
450 H=0
460 CALL CLEAR
470 PRINT TAB(7);"YOU'RE ROLLING"
480 P$="YOU"
490 GOSUB 3590
500 IF F=0 THEN 560
```

```
510 P$="ME"
520 GOSUB 1510
530 DL=400
540 GOSUB 2110
550 GOTO 600
560 GOSUB 1290
570 IF Q$="R" THEN 400
580 H=H+T
590 IF H>=W THEN 950
600 T=0
610 CALL CLEAR
620 GOSUB 2970
630 T=T+R1+R2
640 IF F=0 THEN 660
650 T=0
660 IF F<2 THEN 680
670 A=0
680 PRINT TAB(8);"I'M ROLLING"
690 P$="I"
700 GOSUB 3590
710 IF F=0 THEN 770
720 P$="YOU"
730 GOSUB 1510
740 DL=400
750 GOSUB 2110
760 GOTO 390
770 GOSUB 3800
780 IF AD=0 THEN 830
790 PRINT "I'LL ROLL AGAIN"
800 DL=600
810 GOSUB 2110
820 GOTO 610
830 PRINT "I'LL STOP WITH THIS"
840 A=A+T
850 IF A<W THEN 890
860 DL=350
870 GOSUB 2110
880 GOTO 950
890 YS=YS+2
900 P$="YOU"
910 GOSUB 1510
920 DL=600
```

```
930 GOSUB 2110
940 GOTO 390
950 XS=4
960 YS=2
970 CALL CLEAR
980 P$="WE"
990 T=0
1000 GOSUB 1080
1010 IF A<W THEN 1030
1020 PRINT "I WIN-SKILL TRIUMPHS AGAIN"
1030 IF H<W THEN 1050
1040 PRINT "YOU WIN-IT WAS SHEER LUCK"
1050 GOSUB 2200
1060 END
1070 CALL CLEAR
1080 PRINT TAB(10);"SCOREBOARD"
1090 PRINT
1100 PRINT B$;W;"POINTS NEEDED TO WIN"
1110 CALL HCHAR(24,4,61,26)
1120 PRINT
1130 PRINT TAB(3);"POINTS SCORED";TAB(19);"YOU
";TAB(24);"ME"
1140 PRINT TAB(4);"BEFORE THIS";TAB(19);"---";
TAB(24);"---"
1150 PRINT TAB(5);"SERIES";TAB(18);H;TAB(23);A
1160 CALL HCHAR(24,4,61,26)
1170 PRINT
1180 PRINT TAB(3);P$;B$;"HAVE";T;"POINTS THIS"
1190 PRINT TAB(12);"SERIES"
1200 PRINT
1210 CALL HCHAR(13,3,CC,8)
1220 CALL HCHAR(13,23,CC,8)
1230 CALL VCHAR(13,3,CC,11)
1240 CALL VCHAR(13,30,CC,11)
1250 CALL HCHAR(24,3,CC,28)
1260 PRINT
1270 PRINT
1280 RETURN
1290 PRINT "(P=PASS DICE - R=ROLL AGAIN"
1300 PRINT
1310 INPUT "YOUR DECISION (P OR R)? ":Q$
1320 IF Q$="R" THEN 1350
```

```
1330 IF Q$="P" THEN 1350
1340 GOTO 1310
1350 RETURN
1360 CALL HCHAR(YS-2,XS-2,CC,5)
1370 CALL HCHAR(YS+2,XS-2,CC,5)
1380 CALL VCHAR(YS-2,XS-2,CC,5)
1390 CALL VCHAR(YS-2,XS+2,CC,5)
1400 RETURN
1410 FOR J=1 TO 5
1420 CALL HCHAR(Y,X+J-1,ASC(SEG$("GROAN",J,1))
)
1430 NEXT J
1440 RETURN
1450 FOR J=1 TO 8
1460 CALL HCHAR(Y,X+J-1,ASC(SEG$("DESPAIR!",J,
1)))
1470 NEXT J
1480 DL=300
1490 GOSUB 2110
1500 RETURN
1510 PRINT
1520 PRINT "DICE PASS TO";B$;P$;
1530 RETURN
1540 CALL HCHAR(YS,XS,42)
1550 RETURN
1560 CALL HCHAR(YS-1,XS-1,42)
1570 CALL HCHAR(YS+1,XS+1,42)
1580 RETURN
1590 GOSUB 1540
1600 GOSUB 1560
1610 RETURN
1620 CALL HCHAR(YS-1,XS+1,42)
1630 CALL HCHAR(YS+1,XS-1,42)
1640 GOSUB 1560
1650 RETURN
1660 GOSUB 1540
1670 GOSUB 1620
1680 RETURN
1690 CALL HCHAR(YS,XS-1,42)
1700 CALL HCHAR(YS,XS+1,42)
1710 GOSUB 1620
1720 RETURN
```

```
1730 CALL HCHAR(YS-1,XS-1,64)
1740 CALL HCHAR(YS-1,XS+1,64)
1750 CALL HCHAR(YS,XS,94)
1760 CALL HCHAR(YS+1,XS-1,45,3)
1770 RETURN
1780 CALL HCHAR(Y-4,X-4,CV,9)
1790 CALL HCHAR(Y+4,X-4,CV,9)
1800 CALL VCHAR(Y-4,X-4,CV,9)
1810 CALL VCHAR(Y-4,X+4,CV,9)
1820 RETURN
1830 CALL HCHAR(YS,XS,CC)
1840 RETURN
1850 CALL HCHAR(YS-2,XS-2,CC)
1860 CALL HCHAR(YS+2,XS+2,CC)
1870 RETURN
1880 GOSUB 1830
1890 GOSUB 1850
1900 RETURN
1910 CALL HCHAR(YS+2,XS-2,CC)
1920 CALL HCHAR(YS-2,XS+2,CC)
1930 GOSUB 1850
1940 RETURN
1950 GOSUB 1830
1960 GOSUB 1910
1970 RETURN
1980 CALL HCHAR(YS,XS-2,CC)
1990 CALL HCHAR(YS,XS+2,CC)
2000 GOSUB 1910
2010 RETURN
2020 GOSUB 1950
2030 CALL HCHAR(YS+2,XS-1,CC,3)
2040 CALL HCHAR(YS+3,XS-2,CC)
2050 CALL HCHAR(YS+3,XS+2,CC)
2060 RETURN
2070 FOR J=1 TO BL
2080 CALL SOUND(50,2000*RND+150,10)
2090 NEXT J
2100 RETURN
2110 FOR J=1 TO DL
2120 NEXT J
2130 RETURN
2140 C=INT(RND*11)+5
```

```
2150 RETURN
2160 BL=12
2170 GOSUB 2070
2180 DL=10
2190 GOSUB 2110
2200 BL=20
2210 DL=20
2220 FOR K=1 TO 2
2230 GOSUB 2110
2240 GOSUB 2070
2250 NEXT K
2260 DL=30
2270 BL=30
2280 GOSUB 2110
2290 GOSUB 2070
2300 RETURN
2310 BL=20
2320 GOSUB 2070
2330 RETURN
2340 CALL HCHAR(YS,XS,CC,4)
2350 CALL VCHAR(YS,XS,CC,5)
2360 CALL HCHAR(YS+4,XS,CC,4)
2370 CALL VCHAR(YS+2,XS+3,CC,3)
2380 CALL HCHAR(YS+2,XS+2,CC)
2390 CALL VCHAR(YS,XS+6,CC,5)
2400 CALL HCHAR(YS,XS+6,CC,4)
2410 CALL HCHAR(YS+2,XS+6,CC,4)
2420 CALL HCHAR(YS+1,XS+9,CC)
2430 CALL HCHAR(YS+3,XS+8,CC)
2440 CALL HCHAR(YS+4,XS+9,CC)
2450 CALL HCHAR(YS,XS+12,CC,4)
2460 CALL HCHAR(YS+4,XS+12,CC,4)
2470 CALL VCHAR(YS,XS+12,CC,5)
2480 CALL VCHAR(YS,XS+15,CC,5)
2490 CALL VCHAR(YS,XS+18,CC,5)
2500 CALL VCHAR(YS,XS+21,CC,5)
2510 CALL HCHAR(YS,XS+18,CC,4)
2520 CALL HCHAR(YS+2,XS+18,CC,4)
2530 CALL VCHAR(YS,XS+24,CC,5)
2540 CALL VCHAR(YS,XS+27,CC,5)
2550 CALL VCHAR(YS+1,XS+25,CC,2)
2560 CALL VCHAR(YS+2,XS+26,CC,2)
```

```
2570 RETURN
2580 X=20
2590 Y=19
2600 CALL CLEAR
2610 PRINT "AND NOW A COIN TOSS FOR"
2620 PRINT "THE FIRST ROLL."
2630 PRINT
2640 PRINT "THE COIN IS IN THE AIR"
2650 PRINT "AND..";
2660 DL=1000
2670 GOSUB 2110
2680 DL=100
2690 FOR K=1 TO 3
2700 CALL HCHAR(Y,X-2,CC,5)
2710 GOSUB 2110
2720 CALL HCHAR(Y,X-2,32,5)
2730 CALL VCHAR(Y-5,X,CC,5)
2740 GOSUB 2110
2750 CALL VCHAR(Y-5,X,32,5)
2760 Y=Y-6
2770 NEXT K
2780 Y=1
2790 FOR K=1 TO 3
2800 CALL HCHAR(Y,X-2,CC,5)
2810 GOSUB 2110
2820 CALL HCHAR(Y,X-2,32,5)
2830 CALL VCHAR(Y+1,X,CC,5)
2840 GOSUB 2110
2850 CALL VCHAR(Y+1,X,32,5)
2860 Y=Y+6
2870 NEXT K
2880 CALL HCHAR(Y,X-2,CC,5)
2890 Q=RND
2900 Q$="YOU"
2910 IF Q<=0.5 THEN 2930
2920 Q$="I"
2930 PRINT Q$;B$;"GET FIRST ROLL."
2940 DL=2000
2950 GOSUB 2110
2960 RETURN
2970 R1=INT(RND*6)+1
2980 R2=INT(RND*6)+1
```

```
2990 F=0
3000 IF R1>1 THEN 3020
3010 F=1
3020 IF R2>1 THEN 3040
3030 F=F+1
3040 RETURN
3050 X=8
3060 Y=16
3070 CALL COLOR(13,1,1)
3080 CALL COLOR(14,1,1)
3090 CV=CD
3100 GOSUB 1780
3110 X=21
3120 GOSUB 1780
3130 Y=13
3140 X=11
3150 CV=CC
3160 GOSUB 1780
3170 X=23
3180 GOSUB 1780
3190 FOR K=1 TO 10
3200 CALL COLOR(13,C,1)
3210 CALL COLOR(13,1,1)
3220 CALL COLOR(14,C,1)
3230 CALL COLOR(14,1,1)
3240 NEXT K
3250 CALL COLOR(13,C,1)
3260 RETURN
3270 XS=13
3280 YS=6
3290 GOSUB 1360
3300 Q=R1
3310 GOSUB 3570
3320 XS=20
3330 GOSUB 1360
3340 Q=R2
3350 GOSUB 3570
3360 IF R1>1 THEN 3400
3370 X=11
3380 Y=3
3390 GOSUB 1410
```

```
3400 IF R2>1 THEN 3440
3410 X=18
3420 Y=3
3430 GOSUB 1410
3440 IF F=0 THEN 3460
3450 GOSUB 2310
3460 IF F<2 THEN 3560
3470 DL=40
3480 GOSUB 2110
3490 GOSUB 2160
3500 DL=80
3510 GOSUB 2110
3520 X=13
3530 Y=9
3540 GOSUB 1450
3550 CALL SOUND(1000,150,2)
3560 RETURN
3570 ON Q GOSUB 1730,1560,1590,1620,1660,1690
3580 RETURN
3590 GOSUB 2140
3600 GOSUB 3050
3610 XS=11
3620 YS=13
3630 Q=R1
3640 GOSUB 3780
3650 XS=23
3660 Q=R2
3670 GOSUB 3780
3680 XS=3
3690 YS=2
3700 IF F=0 THEN 3730
3710 GOSUB 2340
3720 CALL SOUND(500,200,2)
3730 DL=400
3740 GOSUB 2110
3750 GOSUB 1070
3760 GOSUB 3270
3770 RETURN
3780 ON Q GOSUB 2020,1850,1880,1910,1950,1980
3790 RETURN
3800 V=A+T
```

```
3810 IF V>=W THEN 3910
3820 IF (W-H)<10 THEN 3910
3830 IF A<H THEN 3860
3840 CT=T/25
3850 GOTO 3900
3860 IF V>=H THEN 3890
3870 CT=T/35
3880 GOTO 3900
3890 CT=T/30
3900 IF RND>CT THEN 3930
3910 AD=0
3920 RETURN
3930 AD=1
3940 RETURN
```

EASY CHANGES

1. If you wish to set the program for a fixed value of the winning score, it can be done by changing line 300 and deleting lines 310 and 320. Simply set W to the winning score desired. For example:

   ```
   300 W=100
   ```

   would make the winning score 100. Don't forget to delete lines 310 and 320.

2. The rolling dice graphics display before each roll can be eliminated by adding line 3185 as follows:

   ```
   3185 GOTO 3250
   ```

   This has the effect of speeding up the game by showing each dice roll immediately.

3. After you play the game a few times, you may wish to change the "pacing" of the game; i.e., the time delays between various messages, etc. To speed up the game try:

   ```
   2110 DL=DL/2
   2115 FOR J=1 TO DL
   2125 DL=2*DL
   ```

   To slow down the pacing, try:

   ```
   2110 DL=2*DL
   ```

```
2115 FOR J=1 TO DL
2125 DL=DL/2
```

4. The color for the graphic is selected randomly at line 2140. To set the color so it is always the same, let's say dark blue, change line 2140 so it reads:

```
2140 C=5
```

You can, of course, select any of the sixteen available colors as described in your BASIC manual or User's Reference Guide under the CALL COLOR command.

MAIN ROUTINES

120– 180	Initializes constants.
190– 380	Initial displays. Gets winning score.
390– 640	Human rolls.
650– 940	Computer rolls.
950–1060	Ending messages.
1070–1280	Displays scoreboard.
1290–1350	Asks user for re-roll decision.
1360–1400	Draws scoreboard dice outline.
1410–1530	Subroutines to print various messages.
1540–1770	Draws scoreboard dice faces.
1780–1820	Draws graphics dice outline.
1830–2060	Draws graphics dice faces.
2070–2330	Loops for delays, colors, and sounds.
2340–2570	Draws graphics "groan."
2580–2960	Performs coin toss.
2970–3040	Determines dice roll.
3050–3260	Controls graphics dice rolling.
3270–3580	Controls scoreboard display.
3590–3790	Subroutine to control scoreboard and dice faces.
3800–3940	Computer strategy. Sets AD=0 to stop rolling or AD=1 to continue rolling.

MAIN VARIABLES

A	Previous score of computer.
H	Previous score of human.
T	Score of current series of rolls.
B$	String of one blank character.

C	Graphics color.
XS, YS	Horizontal, vertical reference print position.
W	Amount needed to win.
Q	Work variable.
R1, R2	Outcome of roll for die 1, die 2.
F	Result of roll (0=no frown, 1=one frown, 2=double frown).
DL	Delay length.
BL	Length of tone generation.
P$	String name of current roller.
Q$	Work string variable.
AD	Computer strategy flag (0=stop rolling, 1=roll again).
J, K	Loop indices.
X, Y	Horizontal, vertical print position.
V	Score computer would have if it passed the dice.
CT	Cutoff threshold used in computer's built-in strategy.
CC, CD, CV	Graphics character.

SUGGESTED PROJECTS

1. The computer's built-in strategy is contained from line 3800 on. Remember, after a no-frown roll, the TI-99/4A must decide whether or not to continue rolling. See if you can improve on the current strategy. You may use, but not modify, the variables A, T, H, and W. The variable AD must be set before returning. Set AD=0 to mean the TI-99/4A passes the dice or AD= 1 to mean the TI-99/4A will roll again.
2. Ask for the operator's name. Then personalize the messages and scoreboard more.
3. Dig into the workings of the graphics routines connected with the dice rolling. Then modify them to produce new, perhaps more realistic, effects.

JOT

PURPOSE

JOT is a two-player word game involving considerable mental deduction. The TI-99/4A will play against you. But be careful! You will find your computer quite a formidable opponent.

The rules of JOT are faily simple. The game is played entirely with three-letter words. All letters of each word must be distinct—no repeats. (See the section on Easy Changes for further criteria used in defining legal words.)

To begin the game, each player chooses a secret word. The remainder of the game involves trying to be the first player to deduce the other's secret word.

The players take turns making guesses at their opponent's word. After each guess, the asker is told how many letters (or hits) his guess had in common with his opponent's secret word. The position of the letters in the word does not matter. For example, if the secret word was "own," a guess of "who" would have two hits. The winner is the first person to correctly guess his opponent's secret word.

HOW TO USE IT

The program begins with some introductory messages while asking you to think of your secret word. It then asks whether or not you wish to make the first guess. This is followed by you and the TI-99/4A alternating guesses at each other's secret word.

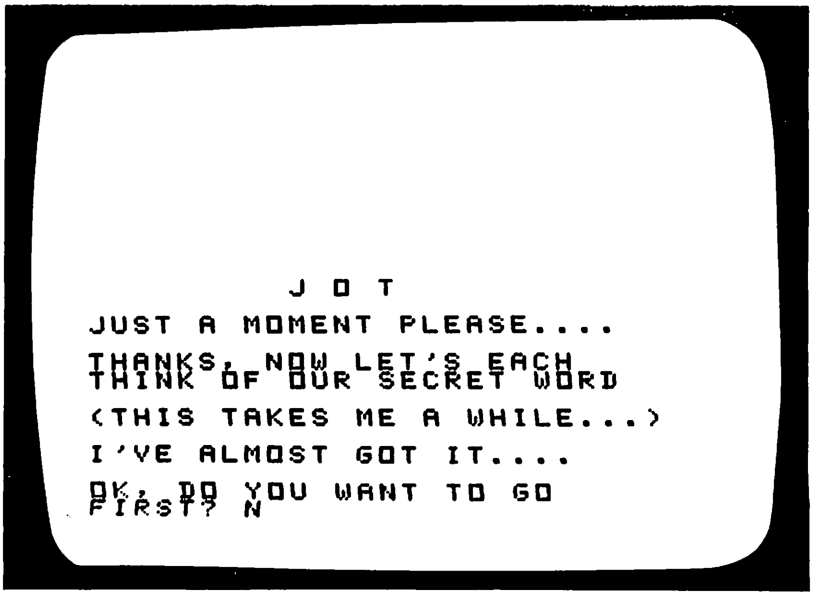
After the computer guesses, it will immediately ask you how it did. Possible replies are **0**, **1**, **2**, **3**, or **R**. The response of **R** (for right) means the TI-99/4A has just guessed your word correctly—a truly humbling experience. The numerical replies indicate that the word guessed by the TI-99/4A had that number of hits in your secret word. A response of 3 means that all the letters were correct, but they need to be arranged to form the actual secret word (e.g., a guess of "EAT" with the secret word being "TEA").

After learning how it did, the computer will take some time to process its new information. If this time is not trivial, the TI-99/4A will display the message: "I'M THINKING" so you do not suspect it of idle daydreaming. If it finds an inconsistency in its information, it will ask you for your secret word and then analyze what went wrong.

When it is your turn to guess, there are two special replies you can make. These are the single letters S and Q. The S, for summary, will display a table of all previous guesses and corresponding hits. This is useful as a concise look at all available information. It will then prompt you again for your next guess. The Q, for quit, will simply terminate the game.

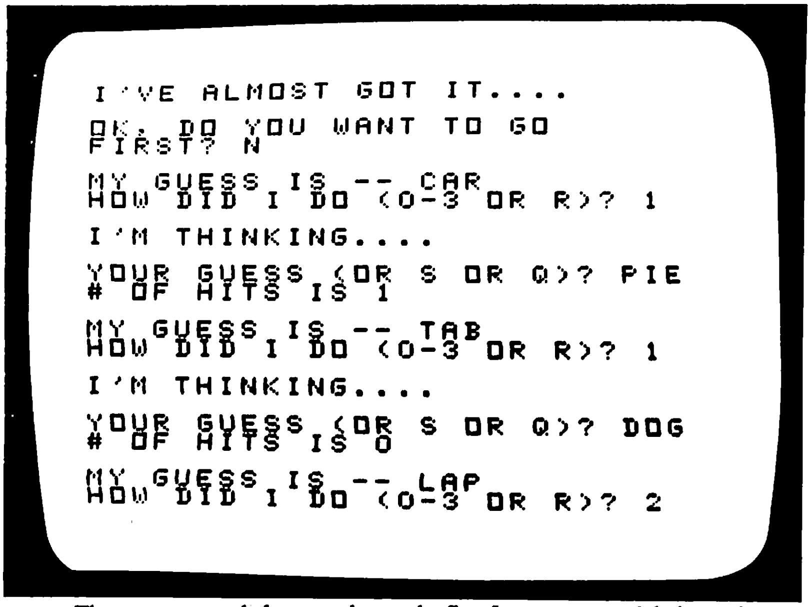
When not making one of these special replies, you will input a guess at the computer's secret word. This will be, of course, a three-letter word. If the word used is not legal, the computer will so inform you. After a legal guess, you will be told how many hits your guess had. If you correctly guess the computer's word, you will be duly congratulated. The TI-99/4A will then ask you for your secret word and verify that all is on the "up and up."

SAMPLE RUN



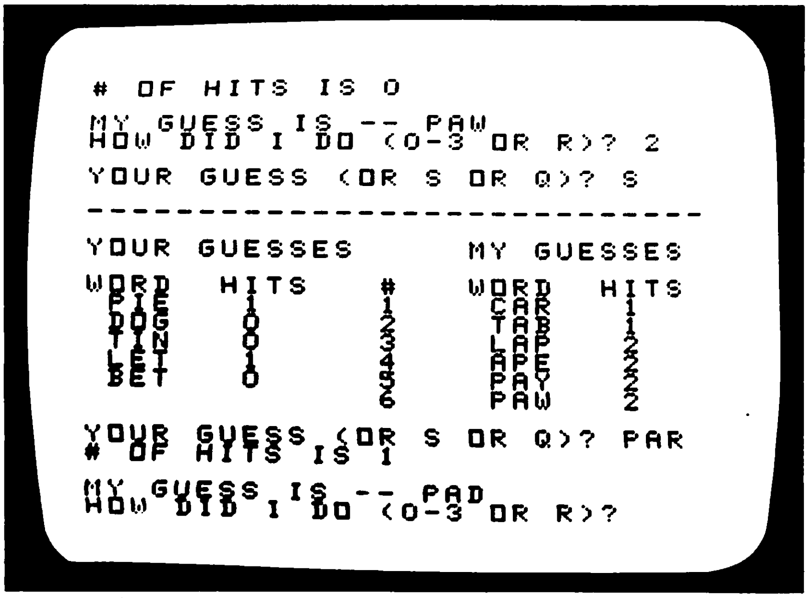
```
          J O T
JUST A MOMENT PLEASE....
THANKS, NOW LET'S EACH
THINK OF OUR SECRET WORD
(THIS TAKES ME A WHILE...)
I'VE ALMOST GOT IT....
OK, DO YOU WANT TO GO
FIRST? N
```

The player and the computer each select their secret words. The computer is given the first guess.



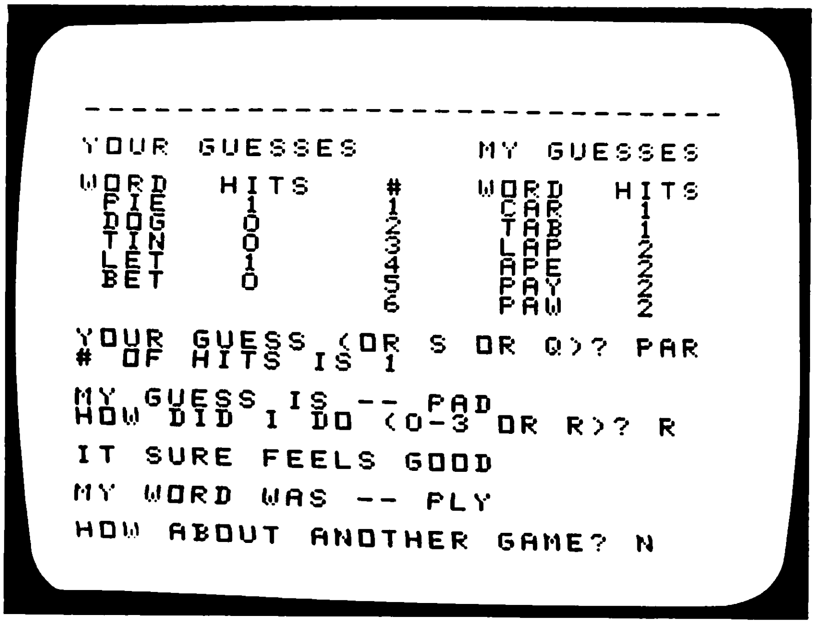
```
I'VE ALMOST GOT IT....
OK, DO YOU WANT TO GO
FIRST? N
MY GUESS IS -- CAR
HOW DID I DO (0-3 OR R)? 1
I'M THINKING....
YOUR GUESS (OR S OR Q)? PIE
# OF HITS IS 1
MY GUESS IS -- TAB
HOW DID I DO (0-3 OR R)? 1
I'M THINKING....
YOUR GUESS (OR S OR Q)? DOG
# OF HITS IS 0
MY GUESS IS -- LAP
HOW DID I DO (0-3 OR R)? 2
```

The computer and player exchange the first few guesses and their results with each other.



```
# OF HITS IS 0

MY GUESS IS -- PAW
HOW DID I DO (0-3 OR R>? 2

YOUR GUESS (OR S OR Q)? S

----------------------------

YOUR GUESSES        MY GUESSES

WORD   HITS    #    WORD  HITS
PIE     1      1    CAR    1
DOG     0      2    TAB    1
TIN     0      3    LAP    2
LET     1      4    APE    2
BET     0      5    PAY    2
               6    PAW    2

YOUR GUESS (OR S OR Q)? PAR
# OF HITS IS 1

MY GUESS IS -- PAD
HOW DID I DO (0-3 OR R)?
```

Later in the same game, the player requests a summary before making his guess.



```
----------------------------

YOUR GUESSES        MY GUESSES

WORD   HITS    #    WORD  HITS
PIE     1      1    CAR    1
DOG     0      2    TAB    1
TIN     0      3    LAP    2
LET     1      4    APE    2
BET     0      5    PAY    2
               6    PAW    2

YOUR GUESS (OR S OR Q)? PAR
# OF HITS IS 1

MY GUESS IS -- PAD
HOW DID I DO (0-3 OR R)? R

IT SURE FEELS GOOD

MY WORD WAS -- PLY

HOW ABOUT ANOTHER GAME? N
```

The computer, however, guesses correctly to win the game. After revealing its secret word, the computer offers another game but the player has had enough.

PROGRAM LISTING

```
100 REM JOT
110 REM (C) 1984 DILITHIUM PRESS
120 M=25
130 N=406
140 DIM A$(406)
150 DIM G1$(25),G2$(25),H1(25),H2(25)
160 RANDOMIZE
170 G1=0
180 G2=0
190 L=N
200 CALL CLEAR
210 PRINT TAB(10);"J O T"
220 PRINT
230 PRINT "JUST A MOMENT PLEASE...."
240 GOSUB 2080
250 Q=RND*N+1
260 PRINT
270 PRINT "THANKS, NOW LET'S EACH"
280 PRINT "THINK OF OUR SECRET WORD"
290 PRINT
300 PRINT "(THIS TAKES ME A WHILE...)"
310 GOSUB 1890
320 M$=A$(Q)
330 PRINT
340 PRINT "OK, DO YOU WANT TO GO"
350 INPUT "FIRST? ":Q$
360 Q$=SEG$(Q$,1,1)
370 IF Q$="N" THEN 700
380 IF Q$="Y" THEN 430
390 PRINT
400 PRINT "YES OR NO PLEASE"
410 CALL SOUND(300,200,2)
420 GOTO 330
430 PRINT
440 INPUT "YOUR GUESS (OR S OR Q)? ":P$
450 IF P$<>"S" THEN 480
460 GOSUB 1300
470 GOTO 430
480 IF P$="Q" THEN 1570
490 IF P$<>M$ THEN 540
500 G1=G1+1
```

```
510 G1$(G1)=P$
520 H1(G1)=9
530 GOTO 2180
540 GOSUB 1810
550 IF F=1 THEN 610
560 PRINT
570 PRINT "THAT'S NOT A LEGAL WORD"
580 PRINT "TRY AGAIN"
590 CALL SOUND(300,200,2)
600 GOTO 430
610 Q$=M$
620 GOSUB 2040
630 Q$=P$
640 GOSUB 1670
650 PRINT "# OF HITS IS";Q
660 G1=G1+1
670 G1$(G1)=Q$
680 H1(G1)=Q
690 IF G1=N THEN 2430
700 Q$=A$(L)
710 G2=G2+1
720 G2$(G2)=Q$
730 PRINT
740 PRINT "MY GUESS IS -- ";Q$
750 INPUT "HOW DID I DO (0-3 OR R)? ":P$
760 IF P$<>"" THEN 780
770 P$="BAD"
780 P$=SEG$(P$,1,1)
790 Q=ASC(P$)
800 IF Q=82 THEN 850
810 IF Q<48 THEN 870
820 IF Q>51 THEN 870
830 P=VAL(P$)
840 GOTO 910
850 H2(G2)=9
860 GOTO 2130
870 PRINT
880 PRINT "BAD ANSWER"
890 CALL SOUND(300,200,2)
900 GOTO 730
910 IF L<100 THEN 940
920 PRINT
```

```
930 PRINT "I'M THINKING...."
940 H2(G2)=P
950 GOSUB 970
960 GOTO 430
970 Q$=G2$(G2)
980 H=H2(G2)
990 J=0
1000 GOSUB 2040
1010 L=L-1
1020 IF L<1 THEN 1150
1030 J=J+1
1040 IF J>L THEN 1140
1050 Q$=A$(J)
1060 GOSUB 1670
1070 IF Q=H THEN 1030
1080 A=J
1090 B=L
1100 GOSUB 2000
1110 L=L-1
1120 IF L<1 THEN 1150
1130 IF L>=J THEN 1050
1140 RETURN
1150 PRINT
1160 PRINT "SOMETHING'S WRONG !!"
1170 PRINT
1180 INPUT "WHAT'S YOUR SECRET WORD? ":P$
1190 GOSUB 1810
1200 IF F=1 THEN 1250
1210 PRINT
1220 PRINT "ILLEGAL WORD !!"
1230 PRINT "I NEVER HAD A CHANCE"
1240 GOTO 1570
1250 PRINT
1260 PRINT "YOU GAVE A BAD ANSWER"
1270 PRINT "SOMEWHERE-CHECK THE SUMMARY"
1280 GOSUB 1300
1290 GOTO 1570
1300 PRINT
1310 Q=G1
1320 IF G2<=G1 THEN 1340
1330 Q=G2
1340 IF Q>0 THEN 1370
```

```
1350 PRINT "NO GUESSES YET"
1360 RETURN
1370 CALL HCHAR(24,3,45,28)
1380 PRINT
1390 PRINT
1400 PRINT "YOUR GUESSES";TAB(18);"MY GUESSES"
1410 PRINT
1420 PRINT "WORD";TAB(7);"HITS";TAB(14);"#";TA
B(18);"WORD";TAB(24);"HITS"
1430 FOR J=1 TO Q
1440 K=1
1450 IF J<=9 THEN 1470
1460 K=0
1470 IF J<=G1 THEN 1500
1480 PRINT TAB(12+K);J;TAB(19);G2$(J);TAB(24);
H2(J)
1490 GOTO 1550
1500 IF J<=G2 THEN 1530
1510 PRINT TAB(2);G1$(J);TAB(7);H1(J);TAB(12+K
);J
1520 GOTO 1550
1530 PRINT TAB(2);G1$(J);TAB(7);H1(J);TAB(12+K
);J;
1540 PRINT TAB(19);G2$(J);TAB(24);H2(J)
1550 NEXT J
1560 RETURN
1570 PRINT
1580 INPUT "HOW ABOUT ANOTHER GAME? ":Q$
1590 Q$=SEG$(Q$,1,1)
1600 IF Q$="Y" THEN 170
1610 IF Q$<>"N" THEN 1630
1620 END
1630 PRINT
1640 PRINT "'YES' OR 'NO' PLEASE"
1650 CALL SOUND(300,200,2)
1660 GOTO 1570
1670 P$=SEG$(Q$,1,1)
1680 Q=0
1690 GOSUB 1750
1700 P$=SEG$(Q$,2,1)
1710 GOSUB 1750
1720 P$=SEG$(Q$,3,1)
```

```
1730 GOSUB 1750
1740 RETURN
1750 IF P$=M1$ THEN 1790
1760 IF P$=M2$ THEN 1790
1770 IF P$=M3$ THEN 1790
1780 RETURN
1790 Q=Q+1
1800 RETURN
1810 F=0
1820 J=0
1830 J=J+1
1840 IF A$(J)=P$ THEN 1870
1850 IF J<N THEN 1830
1860 RETURN
1870 F=1
1880 RETURN
1890 FOR A=N TO 100 STEP -1
1900 B=INT(RND*A)+1
1910 GOSUB 2000
1920 NEXT A
1930 PRINT
1940 PRINT "I'VE ALMOST GOT IT...."
1950 FOR A=99 TO 2 STEP -1
1960 B=INT(RND*A)+1
1970 GOSUB 2000
1980 NEXT A
1990 RETURN
2000 Q$=A$(B)
2010 A$(B)=A$(A)
2020 A$(A)=Q$
2030 RETURN
2040 M1$=SEG$(Q$,1,1)
2050 M2$=SEG$(Q$,2,1)
2060 M3$=SEG$(Q$,3,1)
2070 RETURN
2080 RESTORE
2090 FOR P=1 TO N
2100 READ A$(P)
2110 NEXT P
2120 RETURN
2130 PRINT
2140 PRINT "IT SURE FEELS GOOD"
```

```
2150 PRINT
2160 PRINT "MY WORD WAS -- ";M$
2170 GOTO 1570
2180 PRINT
2190 PRINT "CONGRATULATIONS - THAT'S IT"
2200 PRINT
2210 INPUT "WHAT WAS YOUR WORD? ":P$
2220 GOSUB 1810
2230 J=1
2240 IF F=1 THEN 2300
2250 PRINT
2260 PRINT "ILLEGAL WORD !!"
2270 PRINT "I HAD NO CHANCE"
2280 CALL SOUND(300,200,2)
2290 GOTO 1570
2300 IF A$(J)<>P$ THEN 2350
2310 PRINT
2320 PRINT "NICE WORD !!!"
2330 CALL SOUND(300,200,2)
2340 GOTO 1570
2350 J=J+1
2360 IF J<=L THEN 2300
2370 PRINT
2380 PRINT "YOU MADE AN ERROR SOMEWHERE"
2390 PRINT "-- CHECK THE SUMMARY"
2400 CALL SOUND(300,200,2)
2410 GOSUB 1300
2420 GOTO 1570
2430 PRINT
2440 PRINT "SORRY, I'M OUT OF MEMORY"
2450 CALL SOUND(300,200,2)
2460 PRINT
2470 PRINT "MY WORD WAS - ";M$
2480 GOTO 1570
2490 DATA ACE,ACT,ADE,ADO,ADS,AFT,AGE
2500 DATA AGO,AID,AIL,AIM,AIR,ALE,ALP
2510 DATA AND,ANT,ANY,APE,APT,ARC,ARE
2520 DATA ARK,ARM,ART,ASH,ASK,ASP,ATE
2530 DATA AWE,AWL,AXE,AYE,BAD,BAG,BAN
2540 DATA BAR,BAT,BAY,BED,BEG,BET,BID
2550 DATA BIG,BIN,BIT,BOA,BOG,BOW,BOX
2560 DATA BOY,BUD,BUG,BUM,BUN,BUS,BUT
```

```
2570 DATA BUY,BYE,CAB,CAD,CAM,CAN,CAP
2580 DATA CAR,CAT,COB,COD,COG,CON,COP
2590 DATA COT,COW,COY,CRY,CUB,CUD,CUE
2600 DATA CUP,CUR,CUT,DAB,DAM,DAY,DEN
2610 DATA DEW,DIE,DIG,DIM,DIN,DIP,DOE
2620 DATA DOG,DON,DOT,DRY,DUB,DUE,DUG
2630 DATA DYE,DUO,EAR,EAT,EGO,ELK,ELM
2640 DATA END,ELF,ERA,FAD,FAG,FAN,FAR
2650 DATA FAT,FED,FEW,FIG,FIN,FIR,FIT
2660 DATA FIX,FLY,FOE,FOG,FOR,FOX,FRY
2670 DATA FUN,FUR,GAP,GAS,GAY,GEM,GET
2680 DATA GIN,GNU,GOB,GOD,GOT,GUM,GUN
2690 DATA GUT,GUY,GYP,HAD,HAG,HAM,HAS
2700 DATA HAT,HAY,HEN,HEX,HID,HIM,HIP
2710 DATA HIS,HIT,HER,HEM,HOE,HOG,HOP
2720 DATA HOT,HOW,HUB,HUE,HUG,HUM,HUT
2730 DATA ICE,ICY,ILK,INK,IMP,ION,IRE
2740 DATA IRK,ITS,IVY,JAB,JAR,JAW,JAY
2750 DATA JOB,JOG,JOT,JOY,JUG,JAG,JAM
2760 DATA JET,JIB,JIG,JUT,KEG,KEY,KID
2770 DATA KIN,KIT,LAB,LAD,LAG,LAP,LAW
2780 DATA LAY,LAX,LED,LEG,LET,LID,LIE
2790 DATA LIP,LIT,LOB,LOG,LOP,LOT,LOW
2800 DATA LYE,MAD,MAN,MAP,MAR,MAT,MAY
2810 DATA MEN,MET,MID,MOB,MOW,MUD,MOP
2820 DATA MIX,MUG,NAB,NAG,NAP,NAY,NET
2830 DATA NEW,NIL,NIP,NOD,NOT,NOR,NOW
2840 DATA NUT,OAF,OAR,OAT,ODE,OIL,OAK
2850 DATA OLD,ONE,OPT,ORE,OUR,OUT,OVA
2860 DATA OWE,OWL,OWN,PAD,PAL,PAN,PAR
2870 DATA PAT,PAW,PAY,PEA,PEG,PEN,PET
2880 DATA PEW,PIE,PIG,PIT,PLY,POD,POT
2890 DATA POX,PER,PIN,PRO,PRY,PUB,PUN
2900 DATA PUS,PUT,RAG,RAM,RAN,RAP,RAT
2910 DATA RAW,RAY,RED,RIB,RID,REV,RIG
2920 DATA RIM,RIP,ROB,ROD,ROE,ROT,ROW
2930 DATA RUB,RUE,RUG,RUM,RUT,RYE,RUN
2940 DATA SAD,SAG,SAP,SAT,SAW,SAY,SET
2950 DATA SEW,SEX,SHY,SEA,SIN,SHE,SIP
2960 DATA SIR,SIT,SIX,SKI,SKY,SLY,SOB
2970 DATA SOD,SON,SOW,SOY,SPA,SPY,STY
2980 DATA SUE,SUM,SUN,TAB,TAD,TAG,TAN
```

```
2990 DATA TAP,TAX,TAR,TEA,TEN,THE,THY
3000 DATA TIC,TIE,TIN,TIP,TOE,TON,TOP
3010 DATA TOW,TOY,TRY,TUB,TUG,TWO,URN
3020 DATA USE,UPS,VAN,VAT,VEX,VIA,VIE
3030 DATA VIM,VOW,YAK,YAM,YEN,YES,YET
3040 DATA YOU,WAD,WAG,WAN,WAR,WAS,WAX
3050 DATA WAY,WEB,WED,WET,WHO,WHY,WIG
3060 DATA WIN,WIT,WOE,WON,WRY,ZIP,FIB
```

EASY CHANGES

1. It is fairly common for players to request a summary before most guesses that they make. If you want the program to automatically provide a summary before each guess, change lines 430–460.

```
430 GOSUB 1300
440 PRINT
450 INPUT "YOUR GUESS (OR Q)? ":P$
460 GOTO 480
```

2. The maximum number of guesses allowed, M, can be changed in line 120 and the value of the arrays in line 150. You may wish to increase it in conjunction with Suggested Project 2. You might decrease it to free some memory needed for other program additions. The current value of 25 is somewhat larger than necessary. An actual game almost never goes beyond 15 guesses. To set the number of guesses to 15, change lines 120 and 150 to read:

```
120 M=15
150 DIM G1$(15), G2$(15), H1(15), H2(15)
```

3. Modifying the data list of legal words is fairly easy. Our criteria for legal words were as follows: they must have three distinct letters and *not* be

   –proper names
   –abbreviations
   – interjections (like "ugh", "hey", etc.)
   – specialized words (like "ohm", "sac", "yaw", etc.)

   In line 130, N is set to be the total number of words in the data list. The data list itself is from line 2490 on.

To add word(s) do the following. Enter them in data statements after the current data (use line numbers larger than 3060). Then redefine the value of N to be 406 plus the number of new words added. Also increase the dimension of A$ in line 140 to this new number. For example, to add the words "ohm" and "yaw" into the list, change lines 130 and 140 to read

```
130 N=408
140 DIM A$(408)
```

and add a new line

```
3070 DATA OHM, YAW
```

To delete word(s), the opposite must be done. Remove the words from the appropriate data statement(s) and decrease the value of N accordingly.

MAIN ROUTINES

120– 150	Dimensions arrays.
160– 420	Initializes new game.
430– 690	Human guesses at the computer's word.
700– 960	Computer guesses.
970–1140	Evaluates human's possible secret words. Moves them to the front of A$ array.
1150–1290	Processes inconsistency in given information.
1300–1560	Displays the current summary table.
1570–1660	Inquires about another game.
1670–1800	Compares a guess with key word.
1810–1880	Checks if input word is legal.
1890–1990	Shuffles A$ array randomly.
2000–2030	Swaps elements A and B in the A$ array.
2040–2070	Breaks word Q$ into separate letters.
2080–2120	Fill A$ array from data.
2130–2170	Post-mortem after computer wins.
2180–2420	Post-mortem after human wins.
2430–2480	Error routine—too many guesses.
2490–3060	Data.

MAIN VARIABLES

N	Total number of data words.
M	Maximum number of guesses allowed.

A$	String array holding data words.
G1$, G2$	String arrays of human's, computer's guesses.
H1,H2	Arrays of human's, computer's hits corresponding to G1$, G2$.
G1,G2	Current number of human's, computer's guesses.
M$	Computer's secret word.
M1$, M2$, M3$	First, second, and third letters of a word.
P$, Q$	String temporaries and work variables.
L	Current number of human's possible secret words.
F	Flag for input word legality.
H	Number of hits in last guess.
A, B	A$ array locations to be swapped.
J, P, Q	Temporaries; array and loop indices.
K	Formatting variable for the summary display.

SUGGESTED PROJECTS

1. Additional messages during the course of the game can personify the program even more. After the TI-99/4A finds out how its last guess did, you might try an occasional message like one of these:

   JUST AS I THOUGHT...
   HMM, I DIDN'T EXPECT THAT...
   JUST WHAT I WAS HOPING TO HEAR...

   The value of L is the number of words to which the computer has narrowed down the human's secret word. You might check its value regularly and, when it gets low, come out with something like

   BE CAREFUL, I'M CLOSING IN ON YOU.

2. Incorporate a feature to allow the loser to continue guessing at the other's word. The summary display routine will already work fine even if G1 and G2 are very different from each other. It will display a value of "9" for the number of hits corresponding to the correct guess of a secret word.
3. Incorporate the legalization of words with repeat letters; i.e., make such words legal as both possible secret words and possible guesses. This involves compiling such a word list, adding it to the data, and modifying the program to allow the new kind of words.

OBSTACLE

PURPOSE

This program allows you and a friend (or enemy) to play the game of OBSTACLE, an arcade-like game that's one of our favorites. A combination of physical skills (reflex speed, hand-to-eye coordination, etc.) and strategic skills are needed to beat your opponent. Each game generally takes only a minute or two, so you'll want to play a match of several games to determine the better player.

HOW TO USE IT

The object of the game is to keep moving longer than your opponent without bumping into an obstacle. When the program starts, it asks in turn for the name of the player on the left and on the right. Then it displays the playing field, shows the starting point for each player, and tells you to press any key to start.

After a key is pressed, each player begins moving independently in one of four random directions—up, down, left, or right. As each player moves, he or she builds a "wall" inside the playing field. The computer determines the speed of the move; the player can only control his own direction. The player on the left can change direction to up, down, left, or right by pressing the key **W**, **Z**, **A**, or **S**, respectively. The player on the right does the same by using the keys for **P**, **.** (period), **L**, and **;** (semicolon). Find these keys on the TI-99/4A keyboard and you will see the logic behind these choices.

The first time either player bumps into the wall surrounding the playing field or the obstacle wall built by either player, he loses.

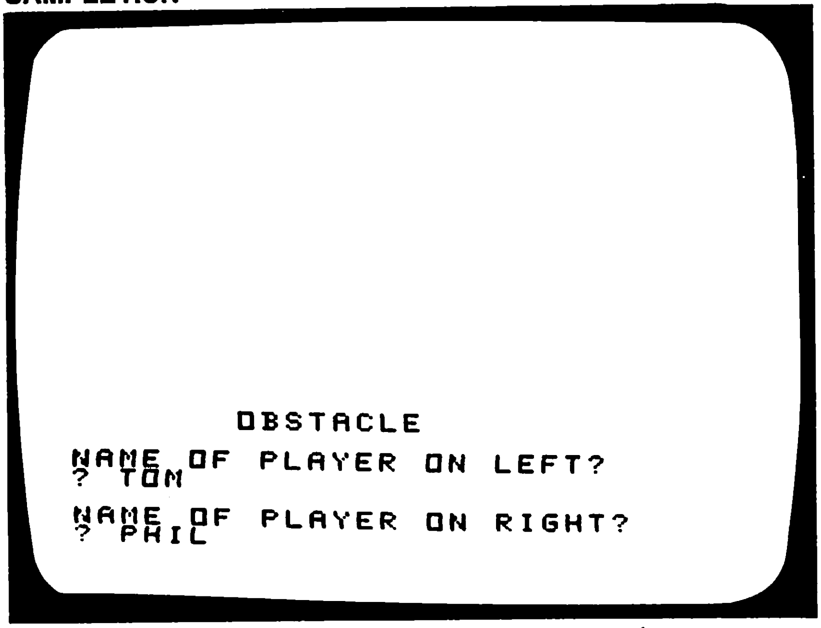
When this happens the program indicates the point of impact for a few seconds and displays the name of the winner. Then the game starts over.

The strategic considerations for this game are interesting. Should you attack your opponent, trying to build a wall around him that he must crash into? Or should you stay away from him and try to make efficient moves in an open area until your opponent runs out of room on his own? Try both approaches and see which yields the most success.

When pressing a key to change direction, be sure to press it quickly and release it. *Do not* hold a key down–you might inhibit the computer from recognizing a move your opponent is trying to make. Once in a while, only one key will be recognized when two are hit at once.

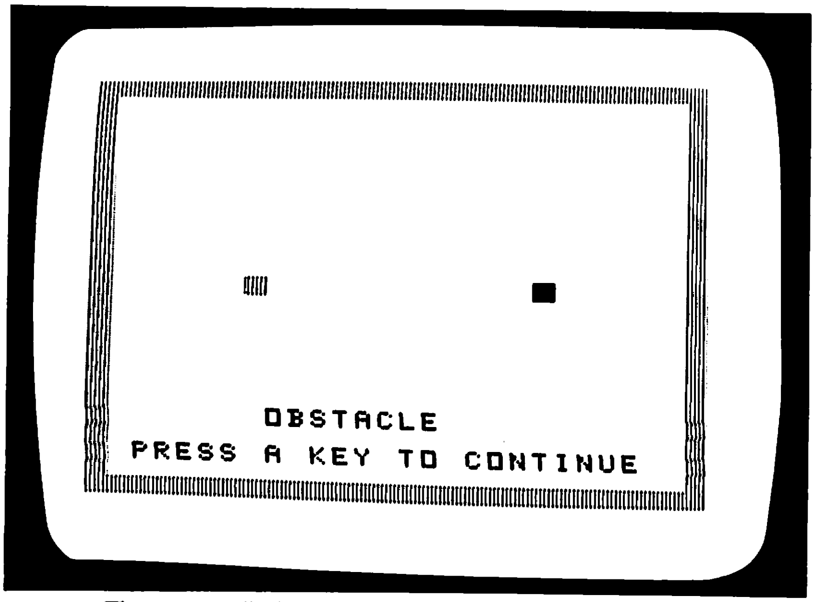
Pressing the ENTER key at the start of a new game ends the program.

SAMPLE RUN



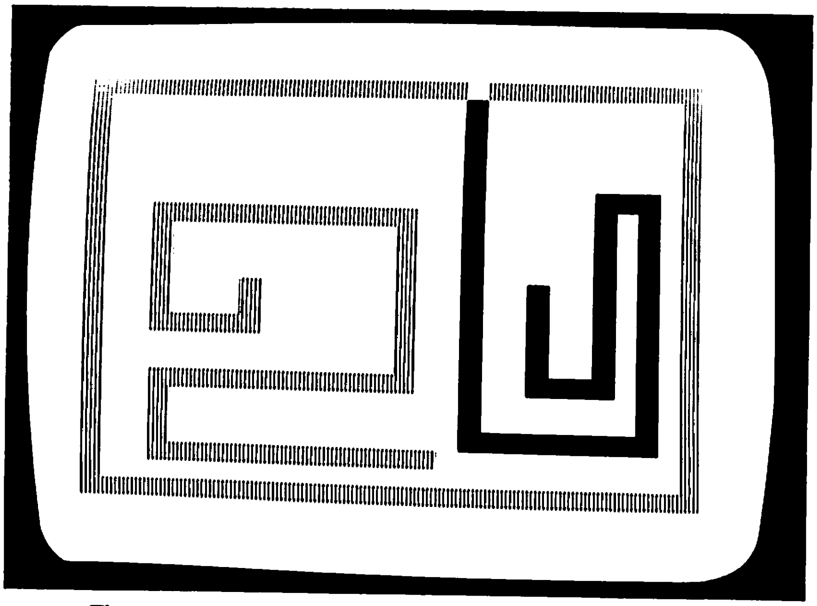
```
          OBSTACLE
NAME OF PLAYER ON LEFT?
? TOM
NAME OF PLAYER ON RIGHT?
? PHIL
```

The program starts off by asking for the names of the two players.



```
OBSTACLE
PRESS A KEY TO CONTINUE
```

The program displays each player's starting position.



The program draws the playing field and starts each player moving in a random direction. Phil, on the right, fails to turn soon enough and hits the wall, losing the game.

PROGRAM LISTING

```
100 REM OBSTACLE
110 REM (C) 1984 DILITHIUM PRESS
120 GOSUB 1010
130 CALL CLEAR
140 PRINT TAB(9);"OBSTACLE"
150 PRINT
160 PRINT "  PRESS A KEY TO CONTINUE"
170 PRINT
180 PRINT
190 Z=15
200 T=23
210 AX=10
220 AY=INT(T/2)+1
230 BX=AX+13
240 BY=AY
250 A=7
260 B=13
270 E=12
280 AD=INT(4*RND)+1
290 BD=INT(4*RND)+1
300 CALL COLOR(9,A,A)
310 CALL COLOR(3,B,B)
320 GOSUB 1260
330 GOSUB 1230
340 CALL KEY(0,C,S)
350 CALL KEY(0,C,S)
360 IF S=0 THEN 350
370 CALL CLEAR
380 IF C<>13 THEN 400
390 END
400 GOSUB 1260
410 GOSUB 1230
420 FOR J=1 TO 5
430 CALL KEY(0,K,S)
440 NEXT J
450 X=AX
460 Y=AY
470 D=AD
480 GOSUB 1320
490 AR=R
```

```
500 AX=X
510 AY=Y
520 X=BX
530 Y=BY
540 D=BD
550 GOSUB 1320
560 BR=R
570 BX=X
580 BY=Y
590 IF AR=1 THEN 820
600 IF BR=1 THEN 820
610 GOSUB 1230
620 FOR J=1 TO 6
630 CALL KEY(0,C,S)
640 IF C<>87 THEN 660
650 AD=1
660 IF C<>90 THEN 680
670 AD=2
680 IF C<>65 THEN 700
690 AD=3
700 IF C<>83 THEN 720
710 AD=4
720 IF C<>80 THEN 740
730 BD=1
740 IF C<>46 THEN 760
750 BD=2
760 IF C<>76 THEN 780
770 BD=3
780 IF C<>59 THEN 800
790 BD=4
800 NEXT J
810 GOTO 450
820 X=AX
830 Y=AY
840 CALL SOUND(200,440,15)
850 IF AR=1 THEN 880
860 X=BX
870 Y=BY
880 FOR J=1 TO 10
890 Z=10
900 CALL COLOR(4,Z,Z)
```

```
910 CALL HCHAR(Y,X,60)
920 FOR K=1 TO 50
930 NEXT K
940 CALL COLOR(4,1,1)
950 CALL HCHAR(Y,X,60)
960 FOR K=1 TO 50
970 NEXT K
980 NEXT J
990 GOSUB 1110
1000 GOTO 130
1010 CALL CLEAR
1020 PRINT TAB(8);"OBSTACLE"
1030 PRINT
1040 PRINT "NAME OF PLAYER ON LEFT?"
1050 INPUT AN$
1060 PRINT
1070 PRINT "NAME OF PLAYER ON RIGHT?"
1080 INPUT BN$
1090 RANDOMIZE
1100 RETURN
1110 CALL CLEAR
1120 IF AR=0 THEN 1160
1130 IF BR=0 THEN 1160
1140 PRINT "YOU BOTH LOSE"
1150 GOTO 1200
1160 R$=AN$
1170 IF AR<>1 THEN 1190
1180 R$=BN$
1190 PRINT R$;" WINS!!"
1200 FOR J=1 TO 500
1210 NEXT J
1220 RETURN
1230 CALL HCHAR(AY,AX,99)
1240 CALL HCHAR(BY,BX,50)
1250 RETURN
1260 CALL COLOR(9,E,E)
1270 CALL HCHAR(1,3,99,28)
1280 CALL HCHAR(T,3,99,28)
1290 CALL VCHAR(1,3,99,T)
1300 CALL VCHAR(1,30,99,T)
1310 RETURN
1320 IF D<>1 THEN 1350
```

```
1330 Y=Y-1
1340 GOTO 1420
1350 IF D<>2 THEN 1380
1360 Y=Y+1
1370 GOTO 1420
1380 IF D<>3 THEN 1410
1390 X=X-1
1400 GOTO 1420
1410 X=X+1
1420 R=0
1430 CALL GCHAR(Y,X,Z)
1440 IF Z=32 THEN 1460
1450 R=1
1460 RETURN
```

EASY CHANGES

1. To speed the game up, change the 6 in line 620 to a 4 or so. To slow it down, make it 8 or 10.
2. To make both players always start moving upward at the beginning of each game (instead of in a random direction), insert the following:

    292 AD=1
    294 BD=1

    To make the players always start off moving toward each other, use this instead:

    292 AD=4
    294 BD=3

3. To change the length of time that the final messages are displayed after each game, modify line 880. Change the 10 to 5 (or so) to shorten it, or to 20 to lengthen it.

MAIN ROUTINES

120- 330	Initializes variables. Gets players' names. Displays titles, playing field.
340- 440	Waits for key to be pressed to start game. Redisplays playing field.
450- 610	Makes move for player A (on left side) and B (on right). Saves results.

620– 810	Accepts moves from keyboard and translates direction.
820–1000	Flashes a square where collision occurred. Goes back to start next game. Displays winner's name.
1010–1100	Subroutine that gets each player's name.
1110–1220	Subroutine that displays winner's name.
1230–1250	Subroutine that displays each graphics character of each player's obstacle on the screen.
1260–1310	Subroutine that displays playing field.
1320–1460	Subroutine that moves marker and determines if space moved to is empty.

MAIN VARIABLES

AX, AY	Coordinates of player A's current position.
BX, BY	Coordinates of player B's current position.
A	A's graphics color.
B	B's graphics color.
S	Status of keyboard input.
T	Height of playing field.
AD, BD	Current direction in which A and B are going (1=up, 2=down, 3=left, 4=right).
E	Graphics color for edge of playing field.
C	Character being read from keyboard.
X, Y	Temporary position on screen.
D	Temporary direction.
AR, BR	Result of A's and B's moves (0=okay, 1=loser).
AN$, BN$	Names of players A and B.
Z	Graphics color displayed when collision is made.
J, K	Loop variables.
R$	Name of winner.
R	Temporary result of A's and B's moves (0=okay, 1=loser)

SUGGESTED PROJECTS

1. Keep score over a seven-game (or so) match. Display the current score after each game. Don't forget to allow for ties.
2. Modify the program to let each player press only two keys—one to turn left from the current direction of travel, and one to turn right.
3. Instead of a game between two people, make it a game of a person against the computer. Develop a computer strategy to keep finding open areas to move to and/or to cut off open areas from the human opponent.

ROADRACE

PURPOSE

Imagine yourself at the wheel of a high-speed race car flying down an infinite straightaway. The traffic is extremely heavy. To stay on course, you must steer accurately or risk collision. How far can you go in one day? How many miles can you survive? Thrills galore without leaving your living room.

The difficuly of the game is completely under your control. By adjusting the road width and visibility conditions, ROADRACE can be made as easy or as challenging as you wish.

HOW TO USE IT

This program requires TI Extended BASIC. The car is controlled with the use of a joystick, but Easy Change #3 details how to operate it from the keyboard if you do not have a joystick.

The program begins by asking you for two inputs; road width and visibility. Road width can be anywhere between 1 and 5. The degree of difficulty changes appreciably with different widths. A very narrow setting will be quite difficult and a wide one relatively easy. Visibility can be set to any of four settings, ranging from "poor" (1) to "good" (4). When visibility is good, the car appears low on the screen. This allows a good view of the road ahead. When visibility is poor, the car appears high on the screen allowing only a brief look at the upcoming cars.

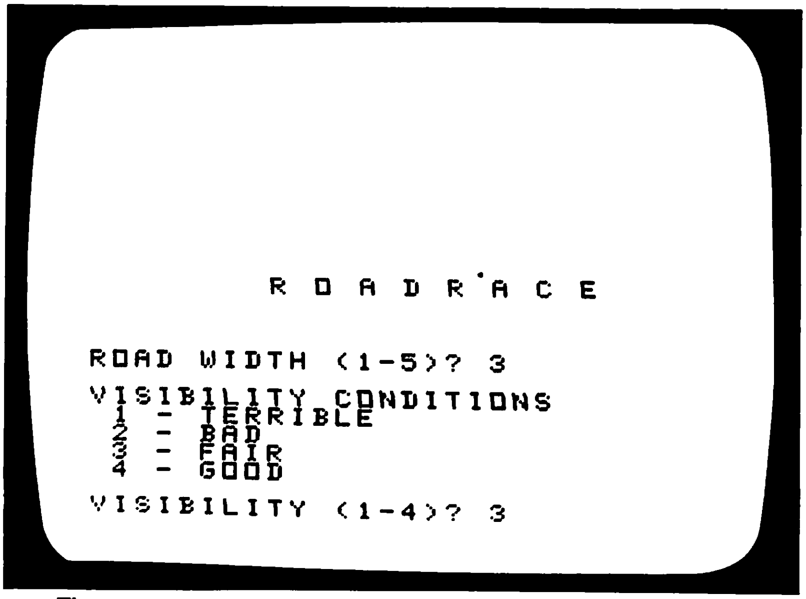
Having set road width and visibility, the race is ready to start. The car appears on the road at the starting line. A five-step starting light counts down to the start. When the bottom light goes on, the race begins. As the road moves continually down the screen, you must steer the car accurately with the joystick, to keep it from colliding with other cars.

The race proceeds until the car collides with another car or an obstacle. Each such collision is considered to terminate one day of the race. After each day, you are shown the number of miles achieved that day along with the cumulative miles achieved for consecutive days of the race.

After each collision, you can proceed by pressing either **C**, **R**, or **Q**. Selecting **C** will continue the race for another day with the same road conditions. Cumulative totals will be retained. **R** will restart the race. This allows changing the road conditions and initializing back to day one. **Q** simply quits the race and returns the TI-99/4A back to direct Extended BASIC. Either of the last two options will produce a display of the average miles traveled per day for the race.

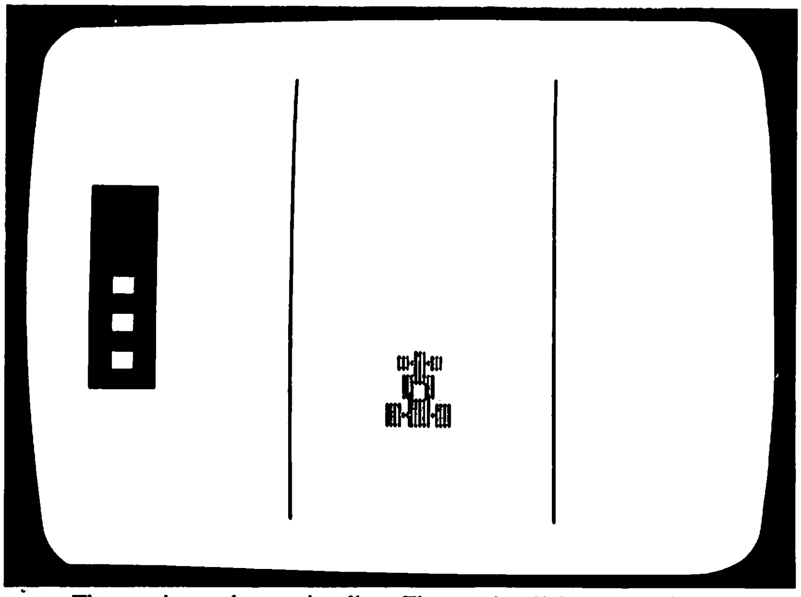
There are several different ways to challenge yourself with the program. You can try to see how far you get in a given number of days. You might see how many days it takes you to go a given number of miles – say 500 miles. As you become proficient at one set of road conditions, make the road narrower and/or the visibility poorer. This will increase the challenge. Different road conditions can also be used as a handicapping aid for two unequally-matched opponents.

SAMPLE RUN

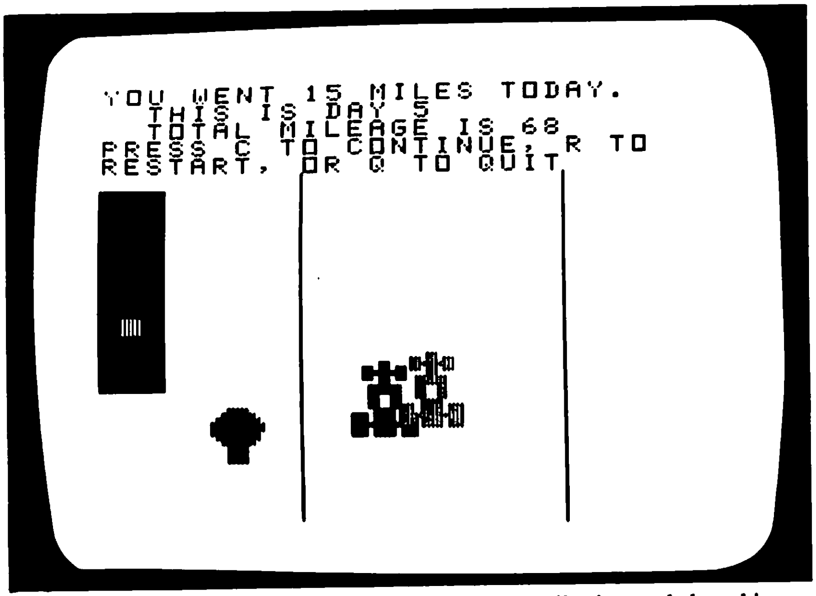


```
         R O A D R A C E

ROAD WIDTH (1-5)? 3
VISIBILITY CONDITIONS
 1 - TERRIBLE
 2 - BAD
 3 - FAIR
 4 - GOOD
VISIBILITY (1-4)? 3
```

The operator selects a course with a moderate road width and a fair visibility of 3.



The car is on the starting line. The starting light counts down the beginning of the race. When the last light goes on, the race will be off and running.



```
YOU WENT 15 MILES TODAY.
THIS IS DAY 5
TOTAL MILEAGE IS 68
PRESS C TO CONTINUE, R TO
RESTART, OR Q TO QUIT
```

The operator finally crashes. A distance of 15 miles is traveled on this leg for a total of 68 miles in 5 days. The options for continuing are displayed while the program waits for the operator's choice.

PROGRAM LISTING

```
100 REM ROADRACE
110 REM (C) 1984 DILITHIUM PRESS
120 RANDOMIZE
130 CALL CHARSET :: CALL CLEAR :: D,M,TM=0
140 C1=9 :: C2=5
150 YV=34 :: XV=24
160 XV2=XV/2
170 GOSUB 750
180 CALL CLEAR :: D=D+1 :: X=0
190 CALL CHAR(60,"03070F1F1F1F0F07")
200 CALL CHAR(61,"0303030300000000")
210 CALL CHAR(62,"C0E0F0F8F8F0F0E0")
220 CALL CHAR(33,"8080808080808080")
230 CALL CHAR(34,"0101010101010101")
240 CALL CHAR(63,"C0C0C0C000000000")
250 CALL CHAR(40,"010D0F0D01070706")
260 CALL CHAR(41,"0606033B3B3F3B3B")
270 CALL CHAR(42,"80B0F0B080E0E060")
```

```
280 CALL CHAR(43,"6060C0DCDCFCDCDC")
290 CALL MAGNIFY(4):: M=0
300 CALL VCHAR(1,XL,33,24)
310 CALL VCHAR(1,XR,34,24)
320 GOSUB 470
330 CALL SPRITE(#1,60,3,1,LT,99,0,#2,60,3,128,
RT,99,0)
340 CALL SPRITE(#4,40,C2,1,150,80,0)
350 CALL JOYST(1,X,Y)
360 M=M+1 :: CALL MOTION(#3,0,X*8)
370 CALL COINC(ALL,C):: IF C<>0 THEN 580
380 CALL POSITION(#3,X3,Y3)
390 IF Y3<LL THEN CALL MOTION(#3,0,0):: CALL L
OCATE(#3,VV,LL)
400 IF Y3>RR THEN CALL MOTION(#3,0,0):: CALL L
OCATE(#3,VV,RR)
410 CALL POSITION(#4,X4,Y4)
420 IF Y4<LL THEN CALL MOTION(#4,80,8)
430 IF Y4>RR THEN CALL MOTION(#4,80,-8)
440 CALL MOTION(#4,30+RND*YV,-XV2+RND*XV)
450 CALL COINC(ALL,C):: IF C<>0 THEN 580
460 GOTO 350
470 CALL SPRITE(#3,40,C1,VV,120,0,0)
480 CALL COLOR(9,2,2)
490 CALL COLOR(10,7,7)
500 CALL COLOR(11,11,11)
510 CALL COLOR(12,13,13)
520 CALL HCHAR(7,3,96,3):: CALL VCHAR(7,3,96,1
1):: CALL VCHAR(7,5,96,11):: CALL
HCHAR(17,3,96,3)
530 DISPLAY AT(9,2):CHR$(96);:: DISPLAY AT(11,
2):CHR$(96);:: DISPLAY AT(13,2):CH
R$(96);:: DISPLAY AT(15,2):CHR$(96);
540 FOR I=8 TO 12 STEP 2 :: FOR J=1 TO 200 ::
NEXT J :: DISPLAY AT(I,2):CHR$(104
);:: NEXT I
550 FOR J=1 TO 200 :: NEXT J :: DISPLAY AT(14,
2):CHR$(112);
560 FOR J=1 TO 200 :: NEXT J :: DISPLAY AT(16,
2):CHR$(120);
570 RETURN
```

```
580 TM=TM+M :: CALL MOTION(#1,0,0,#2,0,0,#3,0,
0,#4,0,0)
590 CALL SOUND(800,-6,2)
600 CALL POSITION(#1,Y1,X1):: IF Y1<50 THEN CA
LL DELSPRITE(#1)
610 CALL POSITION(#2,Y2,X2):: IF Y2<50 THEN CA
LL DELSPRITE(#2)
620 FOR J=1 TO 5 :: DISPLAY AT(J,1):" " :: NEX
T J
630 DISPLAY AT(1,1):"YOU WENT";M;"MILES TODAY.
"
640 DISPLAY AT(2,3):"THIS IS DAY";D
650 DISPLAY AT(3,3):"TOTAL MILEAGE IS";TM
660 DISPLAY AT(4,1):"PRESS C TO CONTINUE, R TO
"
670 DISPLAY AT(5,1):"RESTART, OR Q TO QUIT"
680 CALL KEY(0,K,S):: IF S=0 THEN 680
690 IF K=67 THEN 730
700 IF K=81 THEN 880
710 IF K=82 THEN 880
720 GOTO 680
730 CALL CLEAR :: CALL DELSPRITE(ALL)
740 GOTO 180
750 PRINT TAB(9);"R O A D R A C E" :: PRINT ::
 PRINT
760 PRINT :: INPUT "ROAD WIDTH (1-5)? ":W
770 W=INT(W):: IF W<1 OR W>5 THEN 760
780 PRINT :: PRINT "VISIBILITY CONDITIONS"
790 PRINT " 1 - TERRIBLE":" 2 - BAD"
800 PRINT " 3 - FAIR":" 4 - GOOD"
810 PRINT :: INPUT "VISIBILITY (1-4)? ":V
820 V=INT(V):: IF V<1 OR V>4 THEN 810
830 XL=15-W :: XR=20+W
840 LT=75-8*W :: RT=165+8*W
850 LL=115-8*W :: RR=125+8*W
860 VV=40*V
870 RETURN
880 CALL CLEAR :: CALL DELSPRITE(ALL)
890 DISPLAY AT(5,1):"AVERAGE MILES PER DAY IS"
;TM/D
900 FOR J=1 TO 1500 :: NEXT J
910 IF K=81 THEN END
920 GOTO 130
```

EASY CHANGES

1. The colors used for your car and your opponents' cars are assigned to the variables C1 and C2 respectively. These are set in line 140. Currently, medium red is used for your car and dark blue is used for the opposing cars. You may wish to use different colors for variety. (Don't use transparent, however, for that is the color of the background.) Consult your TI-Extended BASIC manual for the numerical values corresponding to the different colors. For example, to change your car to black and the opposing cars to dark green, change line 140 to read

```
140 C1=2::C2=13
```

2. The speed and motion of the opposing cars is controlled by the variables XV and YV set in line 150. Their motion is random but these two variables control the limit of their horizontal and vertical speeds respectively. If you wish tamer motion, reduce their values. For more exciting, hectic action, you might try

```
150 YV=50::XV=34
```

3. Instead of using a joystick, the keyboard can be used. First, you need to change line 350 to read

```
350 CALL KEY (O,K,S)
```

and insert lines 352 and 354

```
352 IF K=ASC("S") THEN X=-4
354 IF K=ASC("D") THEN X=4
```

This will allow you to use **S** to move left and **D** to move right. To change the keys that are used simply insert the left key letter in 352 and the right key letter in 354.

MAIN ROUTINES

120–290	Variable initialization.
300–320	Initializes road.
330–340	Starts race.
350–460	Moves cars. Checks for collisions.
470–570	Subroutine to draw starting light.
580–720	Processes end of race day.
730–740	Continues race.

750–870	Gets road conditions. Sets variables.
880–920	Displays average miles.

MAIN VARIABLES

XL, XR	Horizontal position of left, right side of road.
VV	Vertical position of player's car.
LT	Horizontal start for left tree.
RT	Horizontal start for right tree.
C	Collision flag variable.
W	Road width.
V	Visibility.
M	Miles driven on current day.
XV, XV2	Range of horizontal car motion.
YV	Range of vertical car motion.
D	Number of days of the race.
TM	Total miles driven for whole race.
LL,RR	Position of left, right side of road.
K, S	User input.
X, Y	Joystick inputs.
I, J	Loop indices and work variables.
C1	Color of player's car.
C2	Color of opposing cars.
X1,X2, X3,X4	Horizontal positions of left tree, right tree, player's car, opposing cars, respectively.
Y1, Y2, Y3, Y4	Vertical positions of left tree, right tree, player's car, opposing cars, respectively.

SUGGESTED PROJECTS

1. Write a routine to evaluate a player's performance after each collision. Display a message rating him anywhere from "expert" to "back-seat driver." This should involve comparing his actual miles achieved against an expected (or average) number of miles for the given road width and visibility. For starters, you might use

$$\text{Expected miles} = 5W^3 + 50V - 100$$

   This formula is crude, at best.
2. Incorporate provisions for two players racing one at a time. Keep cumulative totals separately. After each collision, display the current leader and how far he is ahead.
3. Add physical obstacles or other hazards onto the road in order to increase the challenge. The program will recognize a collision of any two sprites.

WARI

PURPOSE

Wari is an old game with roots that are much older. Its origins go back thousands of years to a variety of other similar games, all classified as being members of the Mancala family. Other variations are Awari, Oware, Pallanguli, Kalah, and countless other offshoots.

The program matches you against the computer. You are probably going to lose a few games before you win one—the computer plays a pretty good game. This may hurt your ego a little bit, since Wari is purely a skill game (like chess or checkers). There is no element of luck involved, as would be the case with backgammon, for example. When you lose, it's because you were outplayed.

HOW TO USE IT

When you start the program, the first thing it does is display the Wari board and ask you if you want to go first. The board is made up of twelve "squares" in two rows of six. Your side is the bottom side, numbered one through six from left to right. The computer's side is on the top, numbered seven through twelve from right to left.

At the start of the game, each square has four "stones" in it. There is no way to differentiate between your stones and the computer's. They all look alike and will move from one side to the other during the course of play.

The first player "picks up" all the stones in one of the squares on his side of the board and drops them, one to a square, starting with the next highest numbered square. The stones continue to be dropped consecutively in each square, continuing over onto the opponent's side if necessary (after square number 12 comes square number 1 again).

If the last stone is dropped onto the opponent's side *and* leaves a total of either two or three stones in that square, these stones are captured by the player who moved, and removed from the board. Also, if the next-to-last square in which a stone was dropped meets the same conditions (on the opponent's side and now with two or three stones), its stones are also captured. This continues backwards until the string of consecutive squares of two or three on the opponent's side is broken.

Regardless of whether any captures are made, play alternates back and forth between the two players.

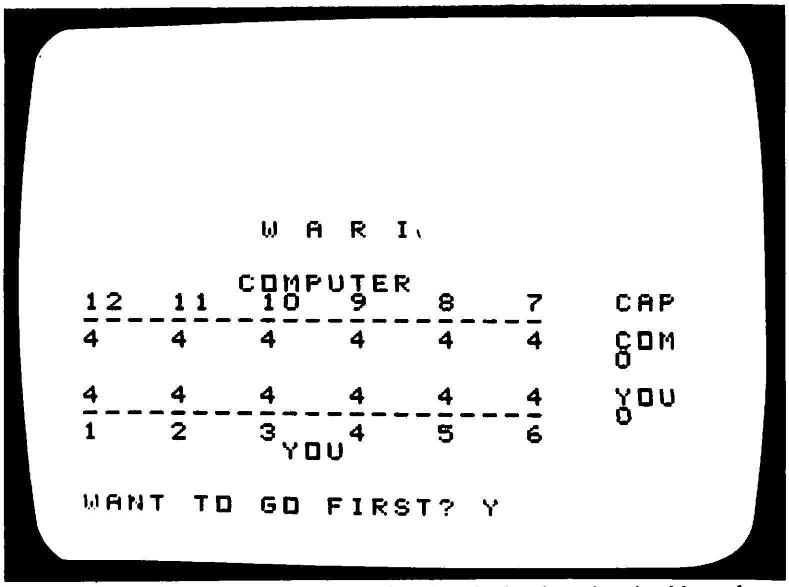
The object of the game is to be the first player to capture 24 or more stones. That's half of the 48 stones that are on the board at the beginning of the game.

There are a few special rules to cover some situations that can come up in the game. It is not legal to capture all the stones on the opponent's side of the board, since this would leave the opponent with no moves on this next turn. By the same token, when your opponent has no stones on his side (because he had to move his last one to your side on his turn), you have to make a move that gives him at least one stone to move on his next turn, if possible. If you cannot make such a move, the game is over and counted as a draw.

During the course of the game, it's possible for a square to accumulate 12 or more stones in it. Moving from such a square causes stones to be distributed all the way around the board. When this happens, the square from which the move was made is skipped over. So, the square moved from is always left empty.

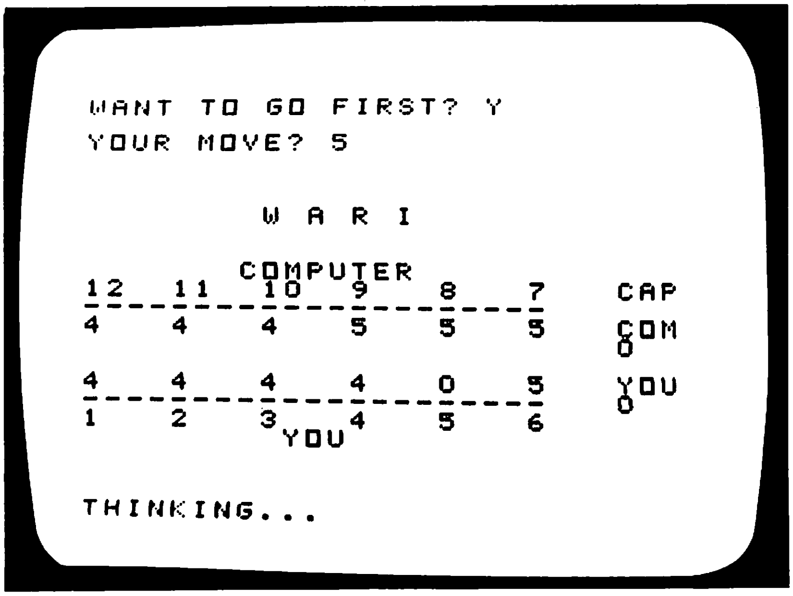
It takes the computer anywhere from fifteen seconds to about ninety seconds to make a move, depending on the complexity of the board position. The word THINKING is displayed during this time, and a period is added to it as each possible move is evaluated in sequence (seven through twelve).

SAMPLE RUN



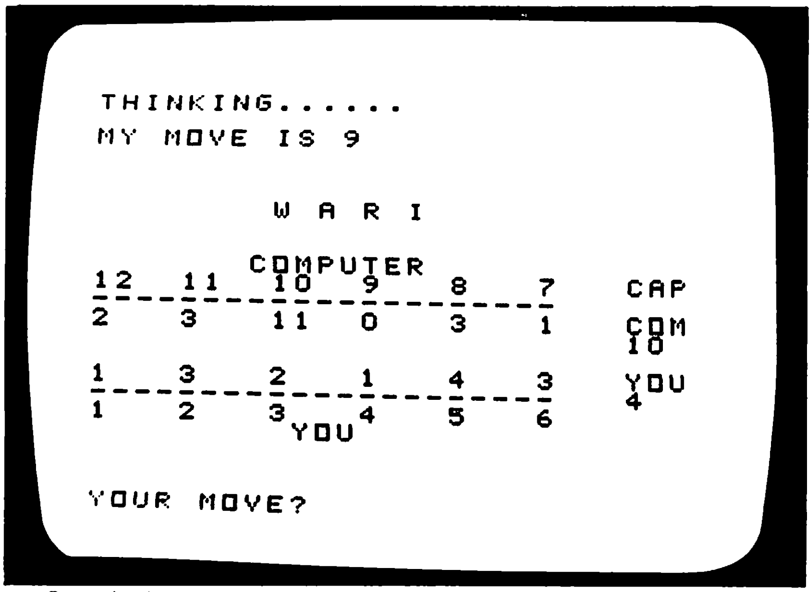
```
           W A R I

           COMPUTER
12    11    10    9     8     7       CAP
-----------------------------------
4     4     4     4     4     4       COM
                                      0
4     4     4     4     4     4       YOU
-----------------------------------   0
1     2     3     4     5     6
            YOU

WANT TO GO FIRST? Y
```

The program starts off by drawing the playing board and asking who should move first. The operator decides to go first.



```
WANT TO GO FIRST? Y
YOUR MOVE? 5

           W A R I

           COMPUTER
12    11    10    9     8     7       CAP
-----------------------------------
4     4     4     5     5     5       COM
                                      0
4     4     4     4     0     5       YOU
-----------------------------------   0
1     2     3     4     5     6
            YOU

THINKING...
```

The program asks for the operator's move. He or she decides to move square number 5. The program alters the board accordingly, and begins "thinking" about what move to make.



```
THINKING......
MY MOVE IS 9

           W  A  R  I

            COMPUTER
12   11    10    9    8    7      CAP
--------------------------------
2    3     11    0    3    1      COM
                                  10
1    3     2     1    4    3      YOU
--------------------------------  4
1    2     3     4    5    6
             YOU

YOUR MOVE?
```

Later in the same game, the computer chooses 9 and the operator is prompted to pick a move. This will continue until the operator or computer captures half the stones and wins the game.

PROGRAM LISTING

```
100 REM WARI
110 REM (C) 1984 DILITHIUM PRESS
120 J=1
130 K=1
140 Q=14
150 P=13
160 F=50
170 D=12
180 I=1
190 DIM T(14),Y(14),W(14),V(6),E(6),B(14)
200 RANDOMIZE
210 ZB=RND/Q
220 ZA=.25+ZB
230 ZB=.25-ZB
240 CALL CLEAR
250 FOR J=1 TO D
260 B(J)=4
270 NEXT J
280 B(P)=0
```

```
290 B(Q)=0
300 MN=0
310 GOSUB 2120
320 INPUT "WANT TO GO FIRST? ":R$
330 R$=SEG$(R$,1,1)
340 IF R$="Y" THEN 550
350 IF R$<>"N" THEN 320
360 PRINT
370 PRINT "THINKING";
380 GOSUB 1360
390 IF M<1 THEN 2430
400 PRINT
410 PRINT
420 PRINT "MY MOVE IS";M
430 FOR J=1 TO Q
440 T(J)=B(J)
450 NEXT J
460 GOSUB 790
470 FOR J=1 TO Q
480 B(J)=T(J)
490 NEXT J
500 GOSUB 2120
510 IF B(Q)<24 THEN 550
520 PRINT
530 PRINT "I WIN!"
540 GOTO 2040
550 PRINT
560 INPUT "YOUR MOVE? ":R$
570 IF LEN(R$)=0 THEN 760
580 R$=SEG$(R$,1,1)
590 IF R$<"1" THEN 760
600 IF R$>"6" THEN 760
610 M=VAL(R$)
620 FOR J=1 TO Q
630 T(J)=B(J)
640 NEXT J
650 GOSUB 790
660 IF M<0 THEN 760
670 FOR J=1 TO Q
680 B(J)=T(J)
690 NEXT J
700 MN=MN+1
```

```
710 GOSUB 2120
720 IF B(P)<24 THEN 360
730 PRINT
740 PRINT "YOU WIN!"
750 GOTO 2040
760 PRINT "ILLEGAL"
770 CALL SOUND(200,440,15)
780 GOTO 550
790 IF T(M)<>0 THEN 820
800 M=-1
810 RETURN
820 R$="H"
830 IF M<=6 THEN 860
840 R$="C"
850 GOTO 900
860 FOR J=1 TO Q
870 Y(J)=T(J)
880 NEXT J
890 GOTO 970
900 FOR J=1 TO 6
910 Y(J)=T(J+6)
920 Y(J+6)=T(J)
930 NEXT J
940 Y(P)=T(Q)
950 Y(Q)=T(P)
960 M=M-6
970 C=M
980 N=Y(C)
990 FOR J=1 TO N
1000 C=C+1
1010 IF C<>P THEN 1030
1020 C=1
1030 IF C<>M THEN 1060
1040 C=C+1
1050 GOTO 1010
1060 Y(C)=Y(C)+1
1070 NEXT J
1080 Y(M)=0
1090 L=C
1100 IF L<7 THEN 1170
1110 IF Y(L)>3 THEN 1170
1120 IF Y(L)<2 THEN 1170
```

```
1130 Y(P)=Y(P)+Y(L)
1140 Y(L)=0
1150 L=L-1
1160 GOTO 1100
1170 S=0
1180 FOR J=7 TO D
1190 S=S+Y(J)
1200 NEXT J
1210 IF S<>0 THEN 1240
1220 M=-2
1230 RETURN
1240 IF R$<>"H" THEN 1290
1250 FOR J=1 TO Q
1260 T(J)=Y(J)
1270 NEXT J
1280 RETURN
1290 FOR J=1 TO 6
1300 T(J)=Y(J+6)
1310 T(J+6)=Y(J)
1320 NEXT J
1330 T(Q)=Y(P)
1340 T(P)=Y(Q)
1350 RETURN
1360 FOR A=1 TO 6
1370 M=A+6
1380 IF B(M)<>0 THEN 1410
1390 E(A)=-F
1400 GOTO 1940
1410 FOR J=1 TO Q
1420 T(J)=B(J)
1430 NEXT J
1440 GOSUB 790
1450 IF M>=0 THEN 1480
1460 E(A)=-F
1470 GOTO 1940
1480 IF T(Q)<24 THEN 1510
1490 M=A+6
1500 RETURN
1510 FOR J=1 TO Q
1520 W(J)=T(J)
1530 NEXT J
1540 FOR K=1 TO 6
```

```
1550 IF T(K)<>0 THEN 1580
1560 V(K)=F
1570 GOTO 1880
1580 FOR J=1 TO Q
1590 T(J)=W(J)
1600 NEXT J
1610 N=K
1620 GOSUB 790
1630 IF N>=0 THEN 1660
1640 V(K)=F
1650 GOTO 1880
1660 FA=0
1670 FB=.05
1680 FC=0
1690 FD=0
1700 FOR J=7 TO D
1710 FB=FB+T(J)
1720 IF T(J)<1 THEN 1740
1730 FA=FA+1
1740 IF T(J)>2 THEN 1760
1750 FC=FC+1
1760 IF T(J)<=FD THEN 1780
1770 FD=T(J)
1780 NEXT J
1790 FE=FB
1800 FOR J=1 TO 6
1810 FE=FE+T(J)
1820 NEXT J
1830 FA=FA/6
1840 FD=1-FD/FB
1850 FC=1-FC/6
1860 FB=FB/FE
1870 V(K)=ZA*(FA+FB)+ZB*(FC+FD)+T(Q)+B(P)-B(Q)
-T(P)
1880 NEXT K
1890 E(A)=F
1900 FOR J=1 TO 6
1910 IF V(J)>=E(A)THEN 1930
1920 E(A)=V(J)
1930 NEXT J
1940 PRINT ".";
1950 NEXT A
```

```
1960 M=0
1970 FA=-F
1980 FOR J=1 TO 6
1990 IF E(J)<=FA THEN 2020
2000 FA=E(J)
2010 M=J+6
2020 NEXT J
2030 RETURN
2040 PRINT "GOOD GAME!"
2050 INPUT "WANT TO PLAY AGAIN? ":R$
2060 IF LEN(R$)=0 THEN 2050
2070 R$=SEG$(R$,1,1)
2080 IF R$="Y" THEN 200
2090 IF R$<>"N" THEN 2050
2100 PRINT "SEE YOU LATER."
2110 END
2120 PRINT
2130 PRINT
2140 PRINT
2150 PRINT TAB(9);"W A R I"
2160 PRINT
2170 PRINT
2180 PRINT TAB(8);"COMPUTER"
2190 FOR J=0 TO 5
2200 PRINT TAB(4*J+1);STR$(12-J);
2210 NEXT J
2220 PRINT TAB(25);"CAP"
2230 PRINT "--------------------"
2240 FOR J=0 TO 5
2250 PRINT TAB(4*J+1);STR$(B(12-J));
2260 NEXT J
2270 PRINT TAB(25);"COM"
2280 PRINT TAB(24);B(Q)
2290 PRINT
2300 FOR J=0 TO 5
2310 PRINT TAB(4*J+1);STR$(B(J+1));
2320 NEXT J
2330 PRINT TAB(25);"YOU"
2340 PRINT "--------------------";
2350 PRINT TAB(24);B(P)
2360 FOR J=0 TO 5
2370 PRINT TAB(4*J+1);STR$(J+1);
```

```
2380 NEXT J
2390 PRINT TAB(10);"YOU"
2400 PRINT
2410 PRINT
2420 RETURN
2430 PRINT "NO LEGAL MOVES."
2440 PRINT "GAME IS A DRAW."
2450 GOTO 2050
```

EASY CHANGES

1. Want a faster-moving game against an opponent who isn't quite such a good player? Insert the following lines:

```
1510 GOTO 1660
1530 REM
1875 E(A)=V(K)
1876 GOTO 1940
```

   In the standard version of the game, the computer looks at each of its possible moves and each of your possible replies when evaluating which move to make. This change causes the computer to only look at each of its moves, without bothering to look at any of your possible replies. As a result, the computer does not play as well, but it takes only a few seconds to make each move.
2. If you are curious about what the computer thinks are the relative merits of each of its possible moves, you can make this change to find out:

```
1940 PRINT E(A);
```

   This will cause the program to display its evaluation number for each of its moves in turn (starting with square seven). It will select the largest number of the six. A negative value means that it will lose stones if that move is made, assuming that you make the best reply you can. A value of negative 50 indicates an illegal move. A positive value greater than one means that a capture can be made by the computer, and it will come out ahead after your best reply.

MAIN ROUTINES

120- 310	Initializes variables. Displays board.
320- 350	Asks who goes first. Evaluates answer.
360- 500	Determines computer's move. Displays new board position.

510– 540	Determines if computer's move resulted in a win. Displays a message if so.
550– 710	Gets operator's move. Checks for legality. Displays new board position.
720– 750	Determines if operator's move resulted in a win.
760– 780	Displays message if illegal move attempted.
790–1350	Subroutine to make move M in T array.
820– 960	Copies T array into Y array (inverts if computer is making the move).
970–1080	Makes move in Y array.
1090–1160	Checks for captures. Removes stones. Checks previous square.
1170–1230	Sees if opponent is left with a legal move.
1240–1350	Copies Y array back into T array.
1360–2030	Subroutine to determine computer's move.
2040–2110	Displays ending message. Asks about playing again.
2120–2420	Subroutine to display Wari board and score.
2430–2450	Displays message when computer has no legal move.

MAIN VARIABLES

Q, P, F, D	Constant values of 14, 13, 50 and 12, respectively.
T, Y, W	Arrays with temporary copies of the Wari board.
V	Array with evaluation values of operator's six possible replies to computer's move being considered.
E	Array with evaluation values of computer's six possible moves.
B	Array containing Wari board. Thirteenth element has stones captured by operator. Fourteenth has computer's.
ZA, ZB	Weighting factors for evaluation function.
MN	Move number.
R$	Operator's reply. Also used as switch to indicate whose move it is (C for computer, H for human).
M	Move being made (1–6 for operator, 7–12 for computer). Set negative if illegal.
C	Subscript used in dropping stones around board.
L	Last square in which a stone was dropped.
S	Stones on opponent's side of the board after a move.
A	Subscript used to indicate which of the six possible computer moves is currently being evaluated.

FA	First evaluation factor used in determining favorability of board position after a move (indicates computer's number of occupied squares).
FB	Second evaluation factor (total stones on computer's side of the board).
FC	Third evaluation factor (number of squares with two or less stones).
FD	Fourth evaluation factor (number of stones in most populous squares on computer's side).
FE	Total stones on board.
J, K	Loop variables.

SUGGESTED PROJECTS

1. Modify the program to declare the game a draw if neither player has made a capture in the past 30 moves. Line 700 adds one to the counter of the number of moves made. To make the change, keep track of the move number of the last capture, and compare the difference between it and the current move number with 30.
2. Modify the evaluation function used by the computer strategy to see if you can improve the quality of its play. Lines 1660 through 1870 examine the position of the board after the move that is being considered. Experiment with the factors and/or the weighting values, or add a new factor of your own.
3. Change the program so it can allow two people to play against each other, instead of just a person against the computer.

Section 4

Graphics Display Programs

The TI-99/4A is an amazing machine. It has very useful graphics capabilities in addition to its other capacities. Programs in the other sections of this book take advantage of these graphics to facilitate and "spice up" their various output. Here we explore the use of the TI-99/4A's graphic capabilities for sheer fun, amusement, and diversion.

Ever look through a kaleidoscope and enjoy the symmetrical changing patterns produced? KALEIDO will create such effects with full eight-point symmetry.

Two other programs produce ever-changing patterns but with much different effects. SPARKLE will fascinate you with a changing shimmering collage. SQUARES uses geometric shapes to obtain its pleasing displays.

WALLOONS demonstrates a totally different aspect of the TI-99/4A. This program will keep you entertained with an example of computer animation.

KALEIDO

PURPOSE

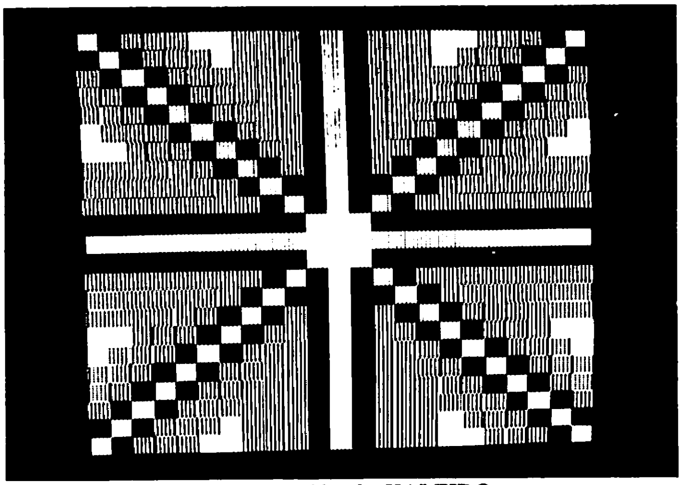
If you have ever played with a kaleidoscope, you were probably fascinated by the endless symmetrical patterns you saw displayed. This program creates a series of kaleidoscope-like designs, with each one overlaying the previous one. Some of the designs are symmetrical about eight axes (others about four)—can you find them?

HOW TO USE IT

There is not much to say about how to use this one. Just type RUN, then sit back and watch. Turning down the lights and playing a little music is a good way to add to the effect.

Have a few friends bring their TI-99/4As over (all your friends *do* have TI-99/4As, don't they?), and get them all going with KALEIDO at once. Let us know if you think you have set a new world's record. Please note that we will not be responsible for any hypnotic trances induced this way.

SAMPLE RUN


One of the patterns generated by the KALEIDO program.

PROGRAM LISTING

```
100 REM KALIEDO
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 CALL SCREEN(2)
140 FOR J=1 TO 14
150 CALL COLOR(J,J+1,J+1)
160 NEXT J
170 P=11
180 RANDOMIZE
190 A=16
200 B=13
210 D=-1
220 M=14
230 DIM R(11)
240 FOR J=0 TO P
250 R(J)=(INT(M*RND)+4)*8
260 NEXT J
270 D=-D
280 K=1
290 L=P
300 IF D>0 THEN 330
310 K=P
```

```
320 L=1
330 FOR J=K TO L STEP D
340 X=A+J
350 Y=B
360 GOSUB 590
370 X=A-J
380 GOSUB 590
390 X=A
400 Y=B+J
410 GOSUB 590
420 Y=B-J
430 GOSUB 590
440 X=A+J
450 Y=B+J
460 GOSUB 590
470 X=A-J
480 Y=B-J
490 GOSUB 590
500 Y=B+J
510 GOSUB 590
520 X=A+J
530 Y=B-J
540 GOSUB 590
550 NEXT J
560 FOR J=1 TO 1000
570 NEXT J
580 GOTO 240
590 CALL HCHAR(Y,X,R(0))
600 IF J<>1 THEN 630
610 CALL HCHAR(B,A,R(0))
620 GOTO 950
630 W=INT(J/2)
640 T=J-W-1
650 FOR N=1 TO W
660 IF X<>A THEN 730
670 Y2=Y
680 X2=X+N
690 GOSUB 960
700 X2=X-N
710 GOSUB 960
720 GOTO 940
730 IF Y<>B THEN 800
```

```
740 X2=X
750 Y2=Y+N
760 GOSUB 960
770 Y2=Y-N
780 GOSUB 960
790 GOTO 940
800 Y2=Y
810 IF X>=A THEN 850
820 X2=X+N
830 GOSUB 960
840 GOTO 870
850 X2=X-N
860 GOSUB 960
870 X2=X
880 IF Y>=B THEN 920
890 Y2=Y+N
900 GOSUB 960
910 GOTO 940
920 Y2=Y-N
930 GOSUB 960
940 NEXT N
950 RETURN
960 CALL HCHAR(Y2,X2,R(N))
970 RETURN
```

EASY CHANGES

1. To clear the screen before the next pattern about 20% of the time (chosen at random), insert this:

```
265 IF RND >.2 THEN 270
266 CALL CLEAR
```

   For 50%, use .5 instead of .2, etc.
2. To randomly change the size of the patterns, insert:

```
262 P=INT(7*RND)+5
```

3. To cause only the outward patterns to be displayed, insert line 263 to

```
263 D=-1
```

   To cause only inward patterns, change it to

```
263 D=1
```

4. To alter the number of graphics colors used in the patterns, alter the value of M in line 220. Be sure it is an integer from 4 to 14.
5. To lengthen the delay after each pattern is drawn, change this line:

```
560 FOR J=1 TO 5000
```

Or, eliminate lines 560 and 570 to eliminate the delay.

Note: These changes add a lot to the appeal of the designs. Experiment! Each change can be done by itself or in combination with other changes.

MAIN ROUTINES

120–230	Housekeeping. Initializes variables.
240–260	Picks 11 random graphics colors.
270	Reverses direction of display (inward-outward).
280–550	Displays a full screen of the pattern.
580	Goes back to create next pattern.
590–950	Plots points on and between axes of design.
960–970	Subroutine to plot points between axes.

MAIN VARIABLES

P	Distance from center to edge of design.
A, B	Pointer to center of design.
D	Direction in which design is drawn (1 = outward, –1 = inward).
M	Multiplier used to determine the range of random graphics colors.
R	Array for the 11 random graphics colors.
J, K, L	Subscript variables.
X, Y	Coordinates of point to be plotted on axes.
X2, Y2	Coordinates of point to be plotted between axes.
N, W	Work variables.

SPARKLE

PURPOSE

This graphics display program provides a continuous series of hypnotic patterns, some of which seem to sparkle at you while they are created. Two types of patterns are used. The first is a set of concentric diamond shapes in the center of the screen. Although the pattern is regular, the sequence in which it is created is random, which results in the "sparkle" effect.

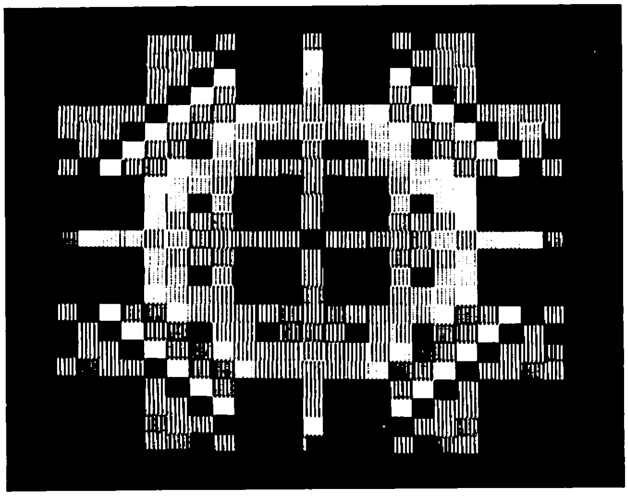
The second type of pattern starts about two seconds after the first has finished. It is a series of "sweeps" across the screen – left to right and top to bottom. Each sweep uses a random graphic color that is spaced equally across the screen. The spacing distance is chosen at random for each sweep. Also, the number of sweeps to be made is chosen at random each time in the range from 10 to 30.

After the second type of pattern is complete, the program goes back to the first type, which begins to overlay the center of the second type.

HOW TO USE IT

Confused by what you just read? Never mind. You have to see it to appreciate it. Just enter the program into your TI-99/4A, then sit back and watch the results of your labor.

SAMPLE RUN


One of the patterns generated by the SPARKLE program.

PROGRAM LISTING

```
100 REM SPARKLE
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 CALL SCREEN(2)
140 RANDOMIZE
150 S=11
160 DIM A(12),B(12)
170 X=16
180 Y=12
190 FOR J=1 TO 14
200 CALL COLOR(J,J+1,J+1)
210 NEXT J
220 T=INT(RND*16)+1
230 FOR J=0 TO S
240 A(J)=J
250 B(J)=J
260 NEXT J
270 FOR J=0 TO S
280 R=INT((S+1)*RND)+1
290 W=A(J)
300 A(J)=A(R)
310 A(R)=W
320 NEXT J
```

```
330 FOR J=0 TO S
340 R=INT((S+1)*RND)+1
350 W=B(J)
360 B(J)=B(R)
370 B(R)=W
380 NEXT J
390 FOR J=0 TO S
400 FOR K=0 TO S
410 R=A(J)
420 W=B(K)
430 C=R+W+T
440 IF C<=14 THEN 470
450 C=C-14
460 GOTO 440
470 CC=(C+3)*8
480 CALL HCHAR(Y+W,X+R,CC)
490 CALL HCHAR(Y-W,X+R,CC)
500 CALL HCHAR(Y-W,X-R,CC)
510 CALL HCHAR(Y+W,X-R,CC)
520 CALL HCHAR(Y+R,X+W,CC)
530 CALL HCHAR(Y+R,X-W,CC)
540 CALL HCHAR(Y-R,X-W,CC)
550 CALL HCHAR(Y-R,X+W,CC)
560 NEXT K
570 NEXT J
580 FOR J=1 TO 1000
590 NEXT J
600 M=14
610 N=INT(21*RND)+10
620 FOR J=1 TO N
630 R=INT(23*RND)+1
640 W=INT(M*RND)
650 CW=(W+4)*8
660 FOR L=Y-S TO Y+S STEP INT(R/4)+1
670 FOR K=X-S TO X+S STEP R
680 CALL HCHAR(L,K,CW)
690 NEXT K
700 NEXT L
710 NEXT J
720 GOTO 220
```

EASY CHANGES

1. Make the second type of pattern appear first by inserting this line:

```
215 GOTO 600
```

   Or, eliminate the first type of pattern by inserting:

```
225 GOTO 600
```

   Or, eliminate the second type of pattern by inserting:

```
595 GOTO 220
```

2. Increase the delay after the first type of pattern by increasing the 1000 in line 580 to, say, 5000. Remove lines 580 and 590 to eliminate the delay.
3. Increase the number of sweeps across the screen of the second type of pattern by changing the 10 at the right end of line 610 into a 30 or a 50, for example. Decrease the number of sweeps by changing the 10 to a 1, and also changing the 21 in line 610 to 5 or 10.
4. Watch the effect on the second type of pattern if you change the 23 in line 630 into various integer values between 2 and 26.
5. Change the value of M in line 600 to alter the graphics color used in the second type of pattern. For example, try

```
600 M=4
```

   Be sure that M is an integer from 2 to 14.

MAIN ROUTINES

120–210	Initializes variables. Clears screen.
220–570	Displays diamond pattern in center of screen.
230–380	Shuffles the numbers 0 through 11 in the A and B arrays.
390–570	Displays graphics colors on the screen.
580–590	Delays for about two seconds.
600–720	Overlays the entire screen with a random graphics color spaced at a fixed interval chosen at random.

MAIN VARIABLES

S	Size of first type of pattern.
R	Random integer. Also, work variable.
A, B	Arrays in which shuffled integers from 0 to S are stored for use in making first type of pattern.

X, Y	Coordinates of center for screen (16 across, 12 down).
T	Integer from 0 to 15, used in creating random graphics colors.
J, K, L	Work and loop variables.
W, CW, C	Work variables.
CC	Graphics color to be displayed on screen in first pattern.
N	Number of repetitions of second type of pattern.
M	Multiplier used in getting a random color for second type of pattern.

SUGGESTED PROJECTS

Make the second type of pattern alternate between "falling from the top" (as it does now) and rising from the bottom of the screen.

SQUARES

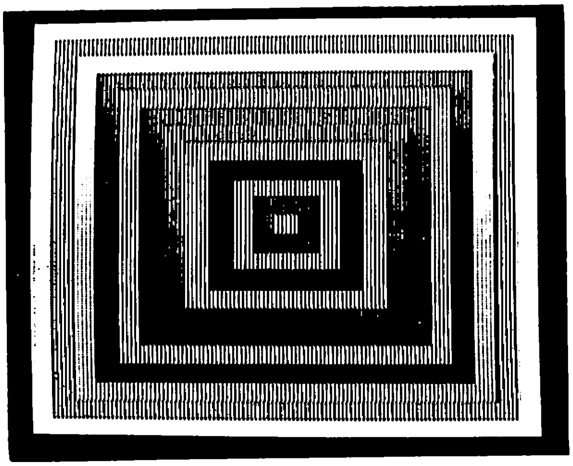
PURPOSE

This is another graphics-display program. It draws a series of concentric squares with the graphics color used for each one chosen at random. After a full set of concentric squares is drawn, the next set starts again at the center and overlays the previous one.

HOW TO USE IT

As with most of the other graphics display programs, you just sit back and enjoy watching this one once you get it started.

SAMPLE RUN


One of the patterns generated by the SQUARES program.

PROGRAM LISTING

```
100 REM SQUARES
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 CALL SCREEN(2)
140 RANDOMIZE
150 FOR J=1 TO 14
160 CALL COLOR(J,J+1,J+1)
170 NEXT J
180 X=16
190 Y=13
200 N=1
210 CALL HCHAR(Y-1,X,(INT(RND*14)+4)*8)
220 C=(INT(RND*14)+4)*8
230 FOR J=0 TO N
240 CALL HCHAR(Y,X+J,C)
250 NEXT J
260 X=X+N
270 N=N+1
280 FOR J=0 TO N
290 CALL HCHAR(Y-J,X,C)
300 NEXT J
310 Y=Y-N
320 FOR J=0 TO N
```

```
330 CALL HCHAR(Y,X-J,C)
340 NEXT J
350 X=X-N
360 FOR J=0 TO N
370 CALL HCHAR(Y+J,X,C)
380 NEXT J
390 N=N+1
400 Y=Y+N
410 IF N<22 THEN 220
420 FOR J=1 TO 1000
430 NEXT J
440 GOTO 180
```

EASY CHANGES

1. Change the delay after each set of patterns by changing the 1000 in line 420. A bigger number causes a longer delay.
2. To occasionally blank out the screen (about 20% of the time), insert this:

```
435 IF RND>.2 THEN 180
436 CALL CLEAR
```

MAIN ROUTINES

120–170	Housekeeping. Clears screen.
180–200	Initializes counters for each pattern. Points to the center of the screen.
210	Plots random color at center.
220	Picks a graphics color.
230–250	Draws the bottom side of the square.
280–300	Draws the right side.
320–340	Draws the top side.
360–380	Draws the left side.
410	Tests if the outermost square has been drawn.
420–430	Delays about three seconds.
440	Goes back to start at the center again.

MAIN VARIABLES

X, Y	Coordinates of points being plotted.
N	Length of the side currently being drawn.
C	Numeric equivalent of the random graphics color chosen.
J	Loop variable.

WALLOONS

PURPOSE

The TI-99/4A is quite a versatile machine. This program takes advantage of its powerful graphics capability to produce computer animation. That's right, animation! WALLOONS will entertain you with a presentation from the T.I. Theatre.

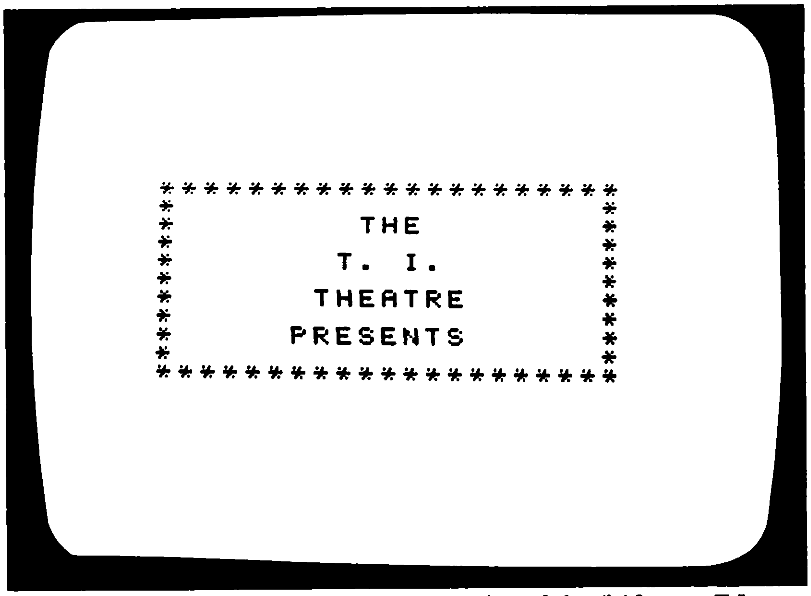
The Theatre searches the world over to bring you the best in circus acts and other performing artists. Today, direct from their performance before the uncrowned heads of Europe, the Arena brings you the Flying Walloons.

HOW TO USE IT

Just sit back, relax, and get ready to enjoy the show. Remember, this program requires the use of TI Extended BASIC. Type RUN and the Flying Walloons will be ready to perform. You have a front-row-center seat and the curtain is about to go up.

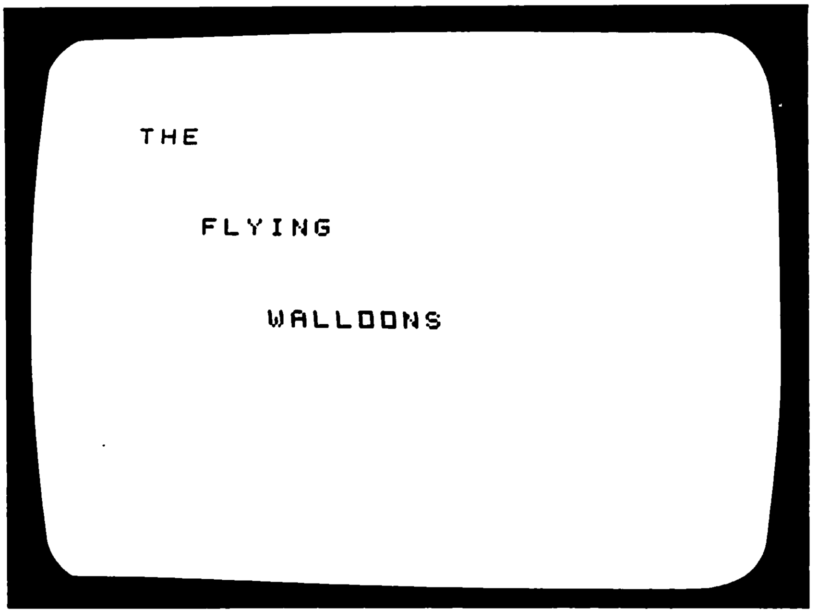
Applause might be appropriate if you enjoy their performance. Please note that the Walloons have been working on a big new finish to their act which they haven't quite yet perfected.

SAMPLE RUN



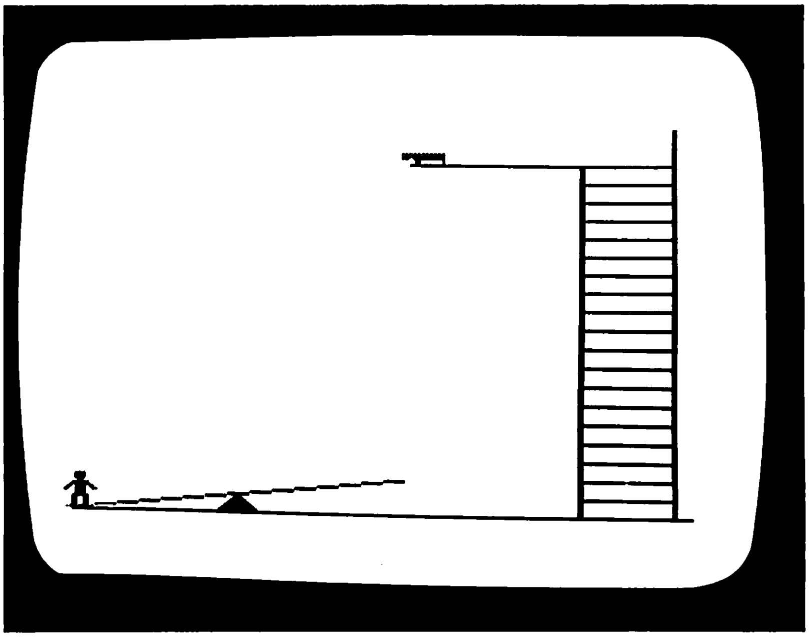
```
* * * * * * * * * * * * * * * * * * * * *
*                                       *
*                THE                    *
*                                       *
*              T.  I.                   *
*                                       *
*            THEATRE                    *
*                                       *
*           PRESENTS                    *
*                                       *
* * * * * * * * * * * * * * * * * * * * *
```

The billboard announces a new presentation of the (in)famous T.I. Theatre.



```
THE

   FLYING

      WALLOONS
```

The Flying Walloons are to perform!


The Walloons attempt a dangerous trick from their repertoire.

PROGRAM LISTING

```
100 REM WALLOONS
110 REM (C) 1984 DILITHIUM PRESS
120 DIM C1(15),C2(15),D1(15),D2(15)
130 FOR I=1 TO 15 :: READ A :: C1$=C1$&CHR$(A)
:: NEXT I
140 FOR I=1 TO 15 :: READ A :: D1$=D1$&CHR$(A)
:: NEXT I
150 FOR I=1 TO 15 :: READ A :: C2$=C2$&CHR$(A)
:: NEXT I
160 FOR I=1 TO 15 :: READ A :: D2$=D2$&CHR$(A)
:: NEXT I
170 DATA 32,32,32,32,32,32,32,32,46,45,44,43,4
2,41,40
180 DATA 46,45,44,43,42,41,48,136,49,32,32,32,
32,32,32
190 DATA 40,41,42,43,44,45,46,32,32,32,32,32,3
2,32,32
```

```
200 DATA 32,32,32,32,32,32,47,136,50,41,42,43,
44,45,46
210 CALL CLEAR :: GOSUB 740
220 FOR I=96 TO 138 :: READ A$ :: CALL CHAR(I,
A$):: NEXT I :: GOSUB 820 :: FOR I
=39 TO 50 :: READ A$ :: CALL CHAR(I,A$):: NEXT
 I
230 FOR I=1 TO 1000 :: NEXT I
240 CALL CLEAR :: FOR I=1 TO 28 :: DISPLAY AT(
24,I):CHR$(39):: NEXT I
250 FOR I=3 TO 23 :: DISPLAY AT(I,28)SIZE(1):C
HR$(134):: NEXT I
260 FOR I=5 TO 23 :: DISPLAY AT(I,23)SIZE(1):C
HR$(133):: NEXT I
270 FOR I=5 TO 23 :: DISPLAY AT(I,24)SIZE(4):C
HR$(39);CHR$(39);CHR$(39);CHR$(39)
:: NEXT I
280 DISPLAY AT(5,23)SIZE(1):CHR$(135)
290 FOR I=22 TO 16 STEP -1 :: DISPLAY AT(5,I)S
IZE(1):CHR$(39):: NEXT I
300 FOR I=1 TO 7 :: DISPLAY AT(22,16-I)SIZE(1)
:CHR$(39+I):: NEXT I
310 FOR I=0 TO 7 :: DISPLAY AT(23,8-I)SIZE(1):
CHR$(39+I):: NEXT I
320 DISPLAY AT(23,7)SIZE(3):CHR$(48);CHR$(136)
;CHR$(49)
330 D=-2 :: CALL MAGNIFY(3):: FOR I=1 TO 500 :
: NEXT I
340 CALL SPRITE(#1,112,5,169,245,0,-8):: FOR I
=1 TO 48 :: FOR J=1 TO 30 :: NEXT
J :: D=-D :: CALL PATTERN(#1,114+D):: NEXT I
350 CALL MOTION(#1,0,0)
360 GOSUB 960 :: CALL LOCATE(#1,169,12):: CALL
 PATTERN(#1,96)
370 GOSUB 870
380 FOR I=1 TO 300 :: NEXT I
390 CALL PATTERN(#2,112):: FOR I=1 TO 100 :: N
EXT I
400 D=-2 :: CALL MOTION(#2,0,-8):: FOR I=1 TO
15 :: FOR J=1 TO 30 :: NEXT J :: D
=-D :: CALL PATTERN(#2,114+D):: NEXT I
```

```
410 CALL MOTION(#2,0,0)
420 CALL LOCATE(#2,17,134)
430 FOR I=1 TO 200 :: NEXT I
440 CALL PATTERN(#2,128):: FOR I=1 TO 150 :: N
EXT I :: CALL PATTERN(#2,112)
450 FOR I=1 TO 100 :: NEXT I :: CALL PATTERN(#
2,120):: CALL MOTION(#2,0,8):: D=-
2 :: FOR I=1 TO 15 :: FOR J=1 TO 30 :: NEXT J
460 D=-D :: CALL PATTERN(#2,122+D):: NEXT I ::
 CALL MOTION(#2,0,0):: GOSUB 960 :
: CALL PATTERN(#2,112)
470 FOR I=1 TO 500 :: NEXT I
480 CALL MOTION(#2,0,-8):: D=-2 :: FOR I=1 TO
15 :: D=-D :: FOR J=1 TO 30 :: NEX
T J :: CALL PATTERN(#2,114+D):: NEXT I
490 CALL MOTION(#2,0,0)
500 FOR I=1 TO 200 :: NEXT I :: CALL PATTERN(#
2,128):: FOR I=1 TO 1000 :: NEXT I
 :: CALL PATTERN(#2,112)
510 FOR I=1 TO 5 :: CALL MOTION(#2,-7,0):: FOR
 J=1 TO 60 :: NEXT J :: CALL MOTIO
N(#2,7,0):: FOR J=1 TO 60 :: NEXT J :: NEXT I
520 CALL MOTION(#2,-7,-4):: FOR J=1 TO 10 :: N
EXT J :: CALL PATTERN(#2,128):: FO
R J=1 TO 45 :: NEXT J
530 CALL PATTERN(#2,100):: FOR J=1 TO 45 :: NE
XT J :: CALL PATTERN(#2,108):: FOR
 J=1 TO 45 :: NEXT J :: CALL PATTERN(#2,96)
540 CALL MOTION(#2,10,-4)
550 FOR K=1 TO 5
560 FOR J=1 TO 30 :: NEXT J :: CALL MOTION(#2,
10,0)
570 CALL POSITION(#2,Y2,X2):: IF Y2<155 THEN 5
70
580 CALL MOTION(#2,0,0):: CALL LOCATE(#2,169,X
2)
590 CALL MOTION(#1,-10,0)
600 DISPLAY AT(22,1)SIZE(15):C2$ :: DISPLAY AT
(23,1)SIZE(15):D2$
610 FOR I=1 TO 400 :: NEXT I :: CALL MOTION(#1
,10,0)
```

```
620 CALL POSITION(#1,Y1,X1):: IF Y1<155 THEN 6
20
630 CALL MOTION(#1,0,0):: CALL LOCATE(#1,169,X
1)
640 IF K=5 THEN GOSUB 930 :: GOTO 680
650 CALL MOTION(#2,-10,0)
660 DISPLAY AT(22,1)SIZE(15):C1$ :: DISPLAY AT
(23,1)SIZE(15):D1$
670 FOR I=1 TO 400 :: NEXT I
680 NEXT K
690 DISPLAY AT(22,1)SIZE(15):C1$ :: DISPLAY AT
(23,1)SIZE(15):D1$ :: GOSUB 940
700 CALL MOTION(#2,10,3):: CALL PATTERN(#2,100
)
710 CALL POSITION(#2,Y2,X2):: IF Y2<169 THEN 7
10
720 CALL MOTION(#2,0,0)
730 FOR I=1 TO 2000 :: NEXT I :: CALL CLEAR ::
 END
740 DISPLAY AT(7,4):"**********************"
750 FOR I=8 TO 16 :: DISPLAY AT(I,4):"*    .
              *" :: NEXT I
760 DISPLAY AT(17,4):"**********************"
770 DISPLAY AT(9,13)SIZE(3):"THE"
780 DISPLAY AT(11,12)SIZE(5):"T. I."
790 DISPLAY AT(13,11)SIZE(7):"THEATRE"
800 DISPLAY AT(15,10)SIZE(8):"PRESENTS"
810 RETURN
820 CALL CLEAR
830 DISPLAY AT(3,3):"THE"
840 DISPLAY AT(8,6):"FLYING"
850 DISPLAY AT(13,9):"WALLOONS"
860 RETURN
870 D=-2 :: CALL SPRITE(#2,120,13,169,5,0,8)::
 FOR I=1 TO 41 :: D=-D :: FOR J=1
TO 30 :: NEXT J :: CALL PATTERN(#2,122+D):: NE
XT I
880 CALL MOTION(#2,0,0):: GOSUB 960 :: CALL PA
TTERN(#2,96)
890 CALL LOCATE(#2,169,210)
900 FOR I=1 TO 5 :: GOSUB 960 :: NEXT I
910 FOR I=19 TO 1 STEP -1 :: CALL LOCATE(#2,(I
*8)+17,210):: FOR J=1 TO 9 :: NEXT
 J :: GOSUB 960 :: GOSUB 960 :: NEXT I
920 CALL SPRITE(#2,96,13,17,210):: RETURN
930 CALL MOTION(#2,-10,3):: RETURN
940 FOR I=1 TO 150 :: NEXT I
950 RETURN
960 FOR K=1 TO 30 :: NEXT K :: RETURN
970 DATA 03030301070F1B33
980 DATA 030307060606061F
990 DATA C0C0C080E0F0D8CC
1000 DATA C0C0E06060606078
1010 DATA 1F06060606070363
1020 DATA 331B0F0701030303
1030 DATA 7860606060E0C0C0
1040 DATA CCD8F0E080C0C0C0
1050 DATA 00000103060CEFFF
1060 DATA FFEF0C0603010000
1070 DATA 00000003033FFFE0
1080 DATA E0FF3F0303000000
1090 DATA 000000C0C0FCFF07
1100 DATA 07FFFCC0C0000000
1110 DATA 000080C06030F7FF
1120 DATA FFF73060C0800000
1130 DATA 07070703070F1F37
1140 DATA 070707070707071F
1150 DATA 0000000000000000
1160 DATA 0000000000000000
1170 DATA 07070703070F1F37
1180 DATA 07070F1B3363030F
1190 DATA 0000000000000000
1200 DATA 0000000000000000
1210 DATA E0E0E0C0E0F0F8EC
1220 DATA E0E0E0E0E0E0E0F8
1230 DATA 0000000000000000
1240 DATA 0000000000000000
1250 DATA E0E0E0C0E0F0F8EC
1260 DATA E0E0F0D8CCC6C0F0
1270 DATA 0000000000000000
1280 DATA 0000000000000000
1290 DATA 0000000000000000
1300 DATA 000000FFFFEF0606
1310 DATA 0000000000000000
```

```
1320 DATA 000000FFFFFF0101
1330 DATA 0F00000000000000
1340 DATA 0303030303030303
1350 DATA C0C0C0C0C0C0C0C0
1360 DATA FF03030303030303
1370 DATA FF183C7EFFFFFFFF
1380 DATA FF000000000080C0E0
1390 DATA FF00000000010307
1400 DATA FF00000000000000
1410 DATA 00FF000000000000
1420 DATA 0000FF0000000000
1430 DATA 000000FF00000000
1440 DATA 00000000FF000000
1450 DATA 0000000000FF0000
1460 DATA 000000000000FF00
1470 DATA 00000000000000FF
1480 DATA 0000000000010307
1490 DATA 00FF000000010307
1500 DATA 0000000000080C0E0
1510 DATA 00FF00000080C0E0
```

EASY CHANGES

1. If you wish to have the Walloons perform more (or less) jumps during their performance, change the loop bound value of 5 in lines 550 and 640 accordingly. To get two jumps, use

   ```
   550 FOR K=1 TO 2
   640 IF K=2 THEN GOSUB 930::GOTO 680
   ```

2. You might want to personalize the title placard and make yourself the presenter of the Walloons. This can be done by altering the string literals in lines 770–790 to something else. However, you cannot use a string with a length of more than 19 characters or it will be clipped by the end of the placard. To say, for example, that Simon Q. Fenster presents the Walloons, delete lines 770 and 790, and change line 780 to read:

   ```
   780 DISPLAY AT (11,6) SIZE (16):"SIMON Q. FENSTER"
   ```

MAIN ROUTINES

120– 200	Initialize variables.
210– 230	Drive placard and announcement. Load shapes.
240– 330	Draws performing apparatus.
340– 370	Walloons enter.
380– 680	Walloons perform.
690– 730	Final act.
740– 810	Subroutine to display placard.
820– 860	Subroutine to display announcement.
870– 920	Subroutine to enter Walloon.
930– 950	Time delay for final act.
960	Time delay.
970–1510	Data to store shapes.

MAIN VARIABLES

X1, Y1	Current location for Walloons.
X2, Y2	
I, J, K	Loop indices.
C1, C2	Arrays containing shape of numbers.
D1, D2	
A$	Work variable.
A, D	Work variables.

SUGGESTED PROJECTS

1. There are many possibilities for "spicing up" the Walloons' act with extra tricks or improved ones. Perhaps you would like to change their finish.
2. If you add some alternate tricks or endings as suggested in the previous project, try randomizing if and when they will be done. Thus, the Walloons' performance will be different each time the program is run. At least their ending may be variable.
3. Scour the world yourself for other acts to include in the T.I. Theatre. Maybe someday we will have a complete software library of performing artists.

Section 5

Mathematics Programs

INTRODUCTION TO MATHEMATICS PROGRAMS

Since their invention, computers have been used to solve mathematical problems. Their great speed and reliability render solvable many otherwise difficult (or impossible) calculations. Several different numerical techniques lend themselves naturally to computer solution. The following programs explore some of them. They will be of interest mainly to engineers, students, mathematicians, statisticians, and others who encounter such problems in their work.

GRAPH takes advantage of the TI-99/4A's graphic powers to draw the graph of a function $Y=f(X)$. The function is supplied by you. INTEGRATE calculates the integral, or "area under the curve," for any such function.

Experimental scientific work frequently results in data at discrete values of X and Y. CURVE finds a polynomial algebraic expression to express this data with a formula.

Theoretical scientists (and algebra students) often must find the solution to a set of simultaneous linear algebraic equations. SIMEQN does the trick.

Much modern engineering work requires the solution of differential equations. DIFFEQN will solve any first-order ordinary differential equation that you provide.

STATS will take a list of data and derive standard statistical information describing it. In addition, it will sort the data list into ranking numerical order.

CURVE

PURPOSE AND DISCUSSION

CURVE fits a polynomial function to a set of data. The data must be in the form of pairs of X-Y points. This type of data occurs frequently as the result of some experiment, or perhaps from sampling tabular data in a reference book.

There are many reasons why you might want an analytic formula to express the functional relationship inherent in the data. Often you will have experimental errors in the Y values. A good formula expression tends to smooth out these fluctuations. Perhaps you want to know the value of Y at some X not obtained exactly in the experiment. This may be a point between known X values (interpolation) or one outside the experimental range (extrapolation). If you wish to use the data in a computer program, a good formula is a convenient and efficient way to do it.

This program fits a curve of the form

$$Y=C_0+C_1X^1+C_2X^2+\ldots+C_DX^D$$

to your data. You may select D, the degree (or power) of the highest term, to be as large as 7. The constant coefficients, C_0-C_D, are the main output of the program. Also calculated is the goodness of fit, a guide to the accuracy of the fit. You may fit different degree polynomials to the same data and also ask to have Y calculated for specific values of X.

The numerical technique involved in the computation is known as least squares curve fitting. It minimizes the sum of the squares of the errors. The least squares method reduces the problem to a set of simultaneous algebraic equations. Thus these equations could be solved by the algorithm used in SIMEQN. In fact, once the proper equations are set up, CURVE uses the identical subroutine found in SIMEQN to solve the equations. For more information, the bibliography contains references to descriptions of the numerical technique.

HOW TO USE IT

The first thing you must do, of course, is enter the data into the program. This consists of typing in pairs of numbers. Each pair represents an X value and its corresponding Y value. The two numbers (of each pair) are separated by a comma. A question mark will prompt you for each data pair. After you have entered them all, type

999,999

to signal the end of the data. When you do this, the program will respond by indicating how many data pairs have been entered. A maximum of 75 data pairs is allowed.

Next, you must input the degree of the polynomial to be fitted. This can be any non-negative integer subject to certain constraints. The maximum allowed is 7. Also, D must be less than the number of data pairs.

A few notes regarding the selection of D may be of interest. If $D=0$, the program will output the mean value of Y as the coefficient C_0. If $D=1$, the program will be calculating the best straight line through the data. This special case is known as "linear regression." If D is one less than the number of data pairs, the program will find an exact fit to the data (barring round-off and other numerical errors). This is a solution which passes exactly through each data point.

Once you have entered the desired degree, the program will begin calculating the results. There will be a pause while this calculation is performed. The time involved depends on the number of data pairs and the degree selected. For 25 data pairs and a third degree fit, the pause will be less than half a minute. Fifty data pairs and a fifth degree fit will take about ninety seconds.

The results are displayed in a table. It gives the values of the coefficients for each power of X from 0 to D. That is, the values of C_0-C_D are output. Also shown is the percent goodness of fit. This is a measure of how accurately the program was able to fit the given case. A value of 100 percent means perfect fit, lesser values indicate correspondingly poorer fits. It is hard to say what value denotes *satisfactory* fit since much depends on the accuracy of data and the purpose at hand. But as a rule of thumb, anything in the high nineties is quite good. For those interested, the formula to calculate the percent goodness of fit is

$$\text{P.G.F.} = 100 * \sqrt{1 - \frac{\sum_i (Y_i - \hat{Y}_i)^2}{\sum_i (Y_i - \bar{Y})^2}}$$

where Y_i are the actual Y data values, $\hat{Y}_i$ are the calculated Y values (through the polynomial expression), and $\bar{Y}$ is the mean value of Y.

Next, you are presented with three options for continuing the run. These are 1) determining specific points, 2) fitting another degree, 3) ending the program. Simply type **1**, **2**, or **3** to make your selection. A description of each choice now follows.

Option 1 allows you to see the value of Y that the current fit will produce for a given value of X. In this mode you are continually prompted to supply any value of X. The program then shows what the polynomial expression produces as the value for Y. Input 999 for an X value to leave this mode.

Option 2 allows you to fit another degree polynomial to the same data. Frequently, you will want to try successively higher values of D to improve the goodness of fit. Unless round-off errors occur, this will cause the percent of goodness of fit to increase.

Option 3 simply terminates the program and with that we will terminate this explanation of how to use CURVE.

SAMPLE PROBLEM AND RUN

Problem: An art investor is considering the purchase of Primo's masterpiece, "Frosted Fantasy." Since 1940, the painting has been for sale at auction seven times. Here is the painting's sales record from these auctions.

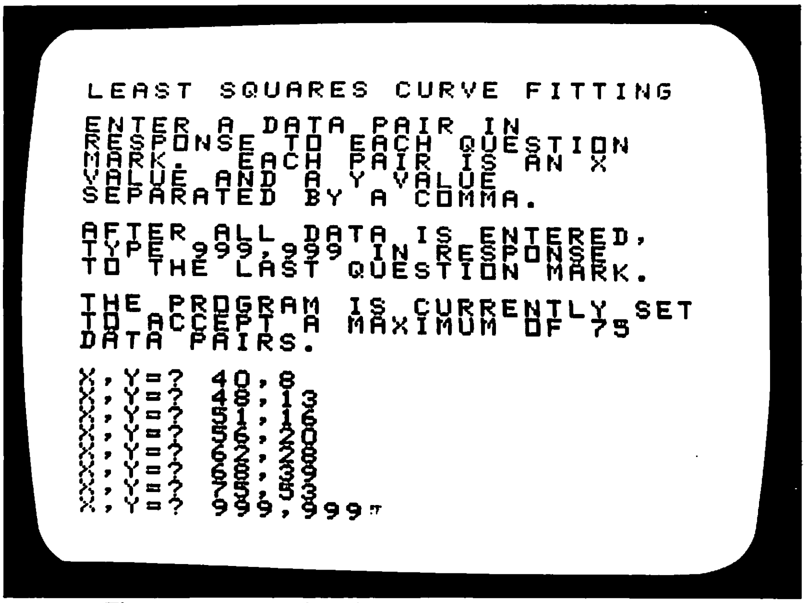
Year	*Price*
1940	$ 8000.
1948	$13000.
1951	$16000.
1956	$20000.
1962	$28000.
1968	$39000.
1975	$53000.

The painting is going to be sold at auction in 1984. What price should the investor expect to have to pay to purchase the painting? If he resold it in 1988, how much profit should he expect to make?

Solution: The investor will try to get a polynomial function that expresses the value of the painting as a function of the year. This is suitable for CURVE. The year will be represented by the variable X, and the price is shown by the variable Y. To keep the magnitude of the numbers small, the years will be expressed as elapsed years since 1900, and the price will be in units of $1000. (Thus a year of 40 represents 1940, a price of 8 represents $8000.)

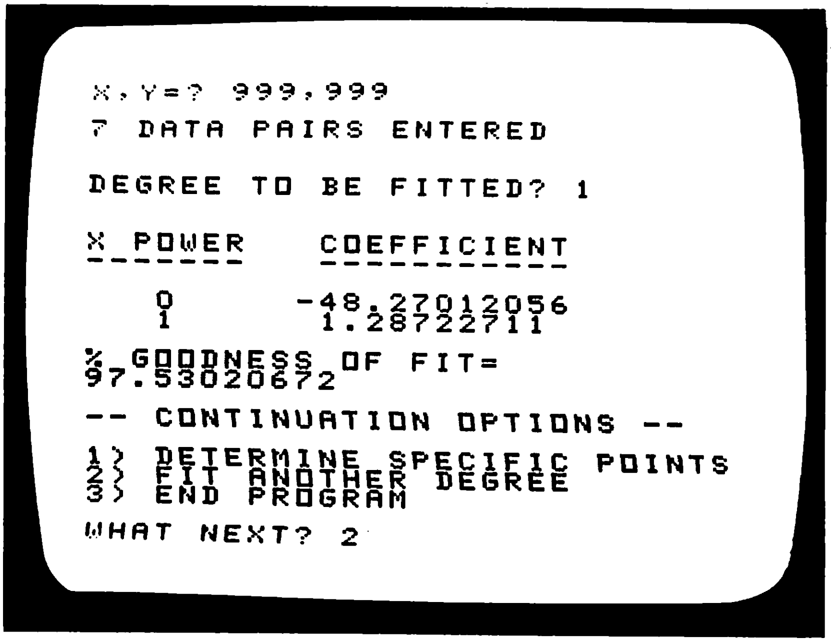
Initially, a first degree fit was tried and a goodness of fit of about 97.5% was obtained. The investor wanted to do better, so he tried a second degree fit next. This had a very high goodness of fit. He then asked for the extrapolation of his data to the years 1984 and 1988. He found that he should expect to pay about $75000 to buy the painting in 1984. Around a $11000 profit could be expected upon resale in 1988.

Of course, the investor did not make his decision solely on the basis of this program. He used it only as one guide to his decision. There is never any guarantee that financial data will perform in the future as it has done in the past. Though CURVE is probably as good a way as any, extrapolation of data can never be a totally reliable process.



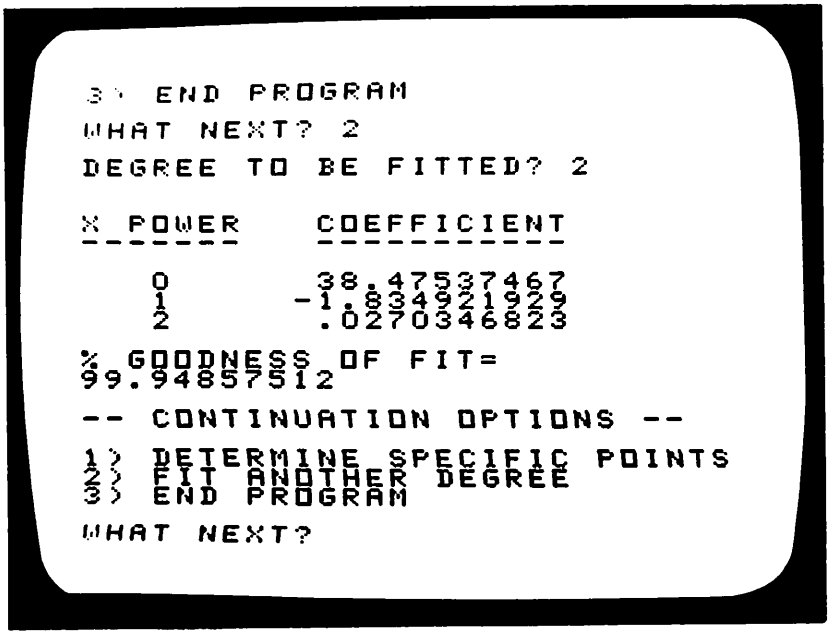
```
LEAST SQUARES CURVE FITTING

ENTER A DATA PAIR IN
RESPONSE TO EACH QUESTION
MARK.  EACH PAIR IS AN X
VALUE AND A Y VALUE
SEPARATED BY A COMMA.

AFTER ALL DATA IS ENTERED,
TYPE 999,999 IN RESPONSE
TO THE LAST QUESTION MARK.

THE PROGRAM IS CURRENTLY SET
TO ACCEPT A MAXIMUM OF 75
DATA PAIRS.

X,Y=? 40,8
X,Y=? 48,13
X,Y=? 51,16
X,Y=? 56,20
X,Y=? 62,28
X,Y=? 68,39
X,Y=? 75,53
X,Y=? 999,999
```

The operator enters the painting's previous auction sales records.



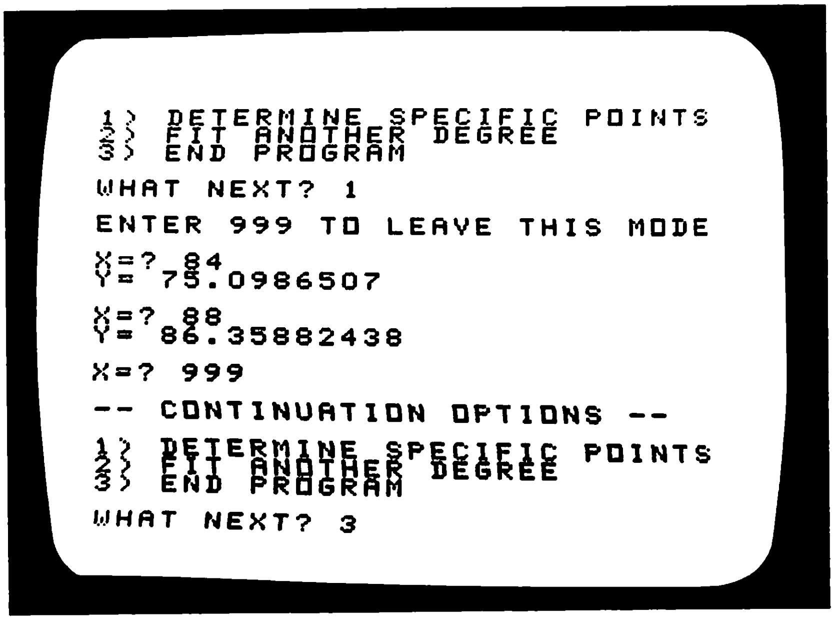
```
X,Y=? 999,999

7 DATA PAIRS ENTERED

DEGREE TO BE FITTED? 1

X POWER     COEFFICIENT
-------     -----------

   0         -48.27012056
   1           1.28722711

% GOODNESS OF FIT=
97.5302067 2

-- CONTINUATION OPTIONS --

1) DETERMINE SPECIFIC POINTS
2) FIT ANOTHER DEGREE
3) END PROGRAM

WHAT NEXT? 2
```

The operator selects a first degree fit to the data.



```
3 > END PROGRAM
WHAT NEXT? 2
DEGREE TO BE FITTED? 2

X POWER    COEFFICIENT
-------    -----------

   0        38.47537467
   1       -1.834921929
   2        .0270346823

% GOODNESS OF FIT=
99.94857512
-- CONTINUATION OPTIONS --
1 > DETERMINE SPECIFIC POINTS
2 > FIT ANOTHER DEGREE
3 > END PROGRAM
WHAT NEXT?
```

A second degree fit is attempted for the same data.



```
1 > DETERMINE SPECIFIC POINTS
2 > FIT ANOTHER DEGREE
3 > END PROGRAM
WHAT NEXT? 1
ENTER 999 TO LEAVE THIS MODE
X=? 84
Y= 75.0986507
X=? 88
Y= 86.35882438
X=? 999
-- CONTINUATION OPTIONS --
1 > DETERMINE SPECIFIC POINTS
2 > FIT ANOTHER DEGREE
3 > END PROGRAM
WHAT NEXT? 3
```

The operator asks to see the predicted sales for 1984 and 1988, and then exits from the program.

PROGRAM LISTING

```
100 REM CURVE
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 MX=75
140 EF=999
150 MD=7
160 DIM X(75),Y(75)
170 DIM A(8,8),R(8),V(8)
180 DIM P(14)
190 PRINT "LEAST SQUARES CURVE FITTING"
200 PRINT
210 PRINT "ENTER A DATA PAIR IN"
220 PRINT "RESPONSE TO EACH QUESTION"
230 PRINT "MARK.  EACH PAIR IS AN X"
240 PRINT "VALUE AND A Y VALUE"
250 PRINT "SEPARATED BY A COMMA."
260 PRINT
270 PRINT "AFTER ALL DATA IS ENTERED,"
280 PRINT "TYPE ";STR$(EF);",";STR$(EF);" IN R
ESPONSE"
290 PRINT "TO THE LAST QUESTION MARK."
300 PRINT
310 PRINT "THE PROGRAM IS CURRENTLY SET"
320 PRINT "TO ACCEPT A MAXIMUM OF";MX
330 PRINT "DATA PAIRS."
340 PRINT
350 J=0
360 J=J+1
370 INPUT "X,Y=? ":X(J),Y(J)
380 IF X(J)<>EF THEN 420
390 IF Y(J)<>EF THEN 420
400 J=J-1
410 GOTO 460
420 IF J=MX THEN 440
430 GOTO 360
440 PRINT "NO MORE DATA ALLOWED"
450 GOTO 460
460 NP=J
470 PRINT
480 IF NP>0 THEN 520
```

```
490 GOSUB 1520
500 PRINT "NO DATA ENTERED"
510 STOP
520 PRINT STR$(NP);" DATA PAIRS ENTERED"
530 PRINT
540 PRINT
550 INPUT "DEGREE TO BE FITTED? ":D
560 PRINT
570 IF D>=0 THEN 610
580 GOSUB 1490
590 PRINT "DEGREE MUST BE >= 0"
600 GOTO 540
610 D=INT(D)
620 IF D<NP THEN 660
630 GOSUB 1490
640 PRINT "NOT ENOUGH DATA"
650 GOTO 540
660 D2=2*D
670 IF D<=MD THEN 710
680 GOSUB 1490
690 PRINT "DEGREE TOO HIGH"
700 GOTO 540
710 N=D+1
720 FOR J=1 TO D2
730 P(J)=0
740 FOR K=1 TO NP
750 P(J)=P(J)+X(K)^J
760 NEXT K
770 NEXT J
780 P(0)=NP
790 R(1)=0
800 FOR J=1 TO NP
810 R(1)=R(1)+Y(J)
820 NEXT J
830 IF N=1 THEN 900
840 FOR J=2 TO N
850 R(J)=0
860 FOR K=1 TO NP
870 R(J)=R(J)+Y(K)*X(K)^(J-1)
880 NEXT K
890 NEXT J
900 FOR J=1 TO N
```

```
910 FOR K=1 TO N
920 A(J,K)=P(J+K-2)
930 NEXT K
940 NEXT J
950 GOSUB 1550
960 PRINT
970 PRINT "X POWER   COEFFICIENT"
980 PRINT "-------   -----------"
990 PRINT
1000 FOR J=1 TO N
1010 PRINT TAB(3);J-1;TAB(10);V(J)
1020 NEXT J
1030 PRINT
1040 Q=0
1050 FOR J=1 TO NP
1060 Q=Q+Y(J)
1070 NEXT J
1080 M=Q/NP
1090 T=0
1100 G=0
1110 FOR J=1 TO NP
1120 Q=0
1130 FOR K=1 TO N
1140 Q=Q+V(K)*X(J)^(K-1)
1150 NEXT K
1160 T=T+(Y(J)-Q)^2
1170 G=G+(Y(J)-M)^2
1180 NEXT J
1190 IF G<>0 THEN 1220
1200 T=100
1210 GOTO 1230
1220 T=100*SQR(1-T/G)
1230 PRINT "% GOODNESS OF FIT= ";STR$(T)
1240 PRINT
1250 PRINT "-- CONTINUATION OPTIONS --"
1260 PRINT
1270 PRINT "1) DETERMINE SPECIFIC POINTS"
1280 PRINT "2) FIT ANOTHER DEGREE"
1290 PRINT "3) END PROGRAM"
1300 PRINT
1310 INPUT "WHAT NEXT? ";Q
1320 Q=INT(Q)
```

```
1330 IF Q<>3 THEN 1350
1340 STOP
1350 IF Q=2 THEN 540
1360 IF Q<>1 THEN 1240
1370 PRINT
1380 PRINT "ENTER";EF;"TO LEAVE THIS MODE"
1390 PRINT
1400 INPUT "X=? ":XV
1410 IF XV<>EF THEN 1430
1420 GOTO 1240
1430 YV=0
1440 FOR K=1 TO N
1450 YV=YV+V(K)*XV^(K-1)
1460 NEXT K
1470 PRINT "Y=";YV
1480 GOTO 1390
1490 PRINT "ERROR"
1500 CALL SOUND(300,200,10)
1510 RETURN
1520 PRINT "FATAL ERROR"
1530 CALL SOUND(300,200,10)
1540 RETURN
1550 IF N<>1 THEN 1580
1560 V(1)=R(1)/A(1,1)
1570 RETURN
1580 FOR K=1 TO N-1
1590 I=K+1
1600 L=K
1610 IF ABS(A(I,K))<=ABS(A(L,K))THEN 1630
1620 L=I
1630 IF I>=N THEN 1660
1640 I=I+1
1650 GOTO 1610
1660 IF L=K THEN 1750
1670 FOR J=K TO N
1680 Q=A(K,J)
1690 A(K,J)=A(L,J)
1700 A(L,J)=Q
1710 NEXT J
1720 Q=R(K)
1730 R(K)=R(L)
1740 R(L)=Q
```

```
1750 I=K+1
1760 Q=A(I,K)/A(K,K)
1770 A(I,K)=0
1780 FOR J=K+1 TO N
1790 A(I,J)=A(I,J)-Q*A(K,J)
1800 NEXT J
1810 R(I)=R(I)-Q*R(K)
1820 IF I>=N THEN 1850
1830 I=I+1
1840 GOTO 1760
1850 NEXT K
1860 V(N)=R(N)/A(N,N)
1870 FOR I=N-1 TO 1 STEP -1
1880 Q=0
1890 FOR J=I+1 TO N
1900 Q=Q+A(I,J)*V(J)
1910 V(I)=(R(I)-Q)/A(I,I)
1920 NEXT J
1930 NEXT I
1940 RETURN
```

EASY CHANGES

1. The program uses 999 as the flag number to terminate various input modes. This may cause a problem if your data include 999. You can easily change the flag number by modifying the value of EF in line 140 to any value not needed in your data. To use 10101, for example, make this change:

   ```
   140 EF=10101
   ```

2. Currently a maximum value of 75 data pairs is allowed. If you need more, change the value of MX in line 130 to the number required and change the values in the arrays X and Y at line 160. For example, to allow up to 200 data pairs, use

   ```
   130 MX=200
   160 DIM X(200),Y(200)
   ```

3. To allow fits of higher degrees than seven, set MD in line 150 to the maximum degree desired, set the arguments of the arrays A, R, and V in line 170 to the maximum number plus one, and set the argument of the array P in line 180 to twice the maximum. To achieve up to tenth degree fits, set the value of MD appropriately:

```
150 MD=10
170 DIM A(11,11), R(11), V(11)
180 DIM P(20)
```

However, it must be stressed that it can be unreliable to attempt high degree fits. Unless your data is well behaved (X and Y values close to 1), the program will often not produce accurate results if D is greater than five or so. This is because sums of powers of X and Y are calculated up to powers of 2*D. These various sums are several orders of magnitude different from each other. Errors result because of the numerous truncation and round-off operations involved in doing arithmetic with them. A practical limit for MD is seven.

4. The demand on available RAM memory is increased if you raise the values of MX or MD as described in the above two Easy Changes. Should MX or MD be set too large for your available memory, a memory full error will result after execution begins.

MAIN ROUTINES

120– 150	Initializes constants.
160– 180	Dimensions arrays.
190– 340	Displays introductory messages.
350– 530	Gets X-Y input data from the user.
540– 700	Gets degree of polynomial from the user, determines if it is acceptable.
710– 950	Sets up equations for the simultaneous equation solver and calls it.
960–1230	Calculates percent goodness of fit, displays all results.
1240–1360	Gets user's continuation option and branches to it.
1370–1480	Determines Y value corresponding to any X value.
1490–1540	Subroutines to print error messages.
1550–1940	Subroutine to solve simultaneous linear algebraic equations.

MAIN VARIABLES

MX	Maximum number of data pairs allowed.
MD	Maximum degree allowed to fit.
EF	Ending flag value for data input and X point mode.
X, Y	Arrays of X and Y data points.
NP	Number of data pairs entered.

D	Degree of polynomial to fit.
D2	2*D, the maximum power sum to compute.
N	D+1, number of simultaneous equations to solve.
A, R, V	Arrays for simultaneous linear equation solver.
P	Array for holding sums of various powers of X.
I, J, K, L	Loop indices.
Q, G	Work variables.
M	Mean value of Y.
T	Percent goodness of fit.
XV	Specific X point for which to calculate Y.
YV	Y value corresponding to XV.

SUGGESTED PROJECTS

1. No provision for modifying the data is incorporated into the program. Often it would be nice to add, subtract, or modify parts of the data after some results are seen. Build in a capability to do this.
2. You may desire other forms of output. A useful table for many applications might include the actual X values, calculated Y values, and/or percentage errors in Y.
3. Sometimes certain points (or certain regions of points) are known to be more accurate than others. Then you would like to weight these points as being more important than others to be fit correctly. The least squares method can be modified to include such a weighting parameter with each data pair. Research this technique and incorporate it into the program. (Note: you can achieve some weighting with the current program by entering important points two or more times. There is a certain danger to this, however. You must only ask for a solution with D less than the number of *unique* data points. A division by zero error may result otherwise.)
4. Often you wish to try successively higher degree polynomials until a certain minimum percent goodness of fit is obtained. Modify the program to accept a minimally satisfactory percent goodness of fit from the user. Then have the program automatically try various polynomial fits until it finds the lowest degree fit, if any, with a satisfactory goodness of fit.

DIFFEQN

PURPOSE

Differential equations express functions by giving the rate of change of one variable with respect to another. This type of relation occurs regularly in almost all the physical sciences. The solution of these equations is necessary in many practical engineering problems.

For many such equations, a closed form (or exact analytical expression) solution can be obtained. However, for just as many, no such "simple" solution exists. The equation must then be solved numerically, usually by a computer program such as this.

There are many types and classes of differential equations. This program solves those of a simple type; namely, first order, ordinary differential equations. This means that the equation to be solved can be written in the form

$$\frac{dY}{dX} = \text{(any function of X,Y)}$$

Here, X is the independent variable and Y is the dependent variable. The equation expresses the derivative (or rate of change) of Y with respect to X. The right-hand side is an expression which may involve X and/or Y.

To use the program, you must supply it with the differential equation to be solved. The procedure used to do this is explained in the "How To Use It" section.

A technique known as the "fourth-order, Runge-Kutta" method is used to solve the equation. Space limitations prevent any detailed explanation of it here. However, it is discussed well in the numerical analysis books referenced in the bibliography.

HOW TO USE IT

The first thing you must do is enter the differential equation into the program. This must be done at line 1200. Currently this line contains a GOTO statement. This GOTO will cause an error message to be displayed if the program is run before you have changed line 1200. The form of line 1200 should be:

```
1200 D=(your function of X,Y)
```

D represents dY/dX. GOSUBs are made to line 1200 with X and Y set to their current values. Thus, when each RETURN is made, D will be set to the appropriate value of dY/dX for that given X and Y. If necessary, you may use the lines between 1200 and 2000 to complete the definition of D. Line 2000 already contains a RETURN statement so you do not need to add another one.

The program begins by warning you that you should have already entered the equation at line 1200. You acknowledge that this has been done by hitting the C key to continue.

Now the various initial conditions are input. You are prompted for them one at a time. They consist of: the initial values of X and Y, the stepsize interval in X at which to display the output, and the final value of X.

With the input phase completed, the program initializes things to begin the output. A question mark will be displayed in the lower left of the screen, telling you the program is waiting for you to hit any key to begin the output.

The output is displayed at each interval of the stepsize until the final value of X is reached. Output may temporarily be halted at any time by simply hitting any key. This will stop the display until you hit any key to resume the output. The output may be started and stopped as often as desired, thus enabling you to leisurely view intermediate results before they scroll off the screen.

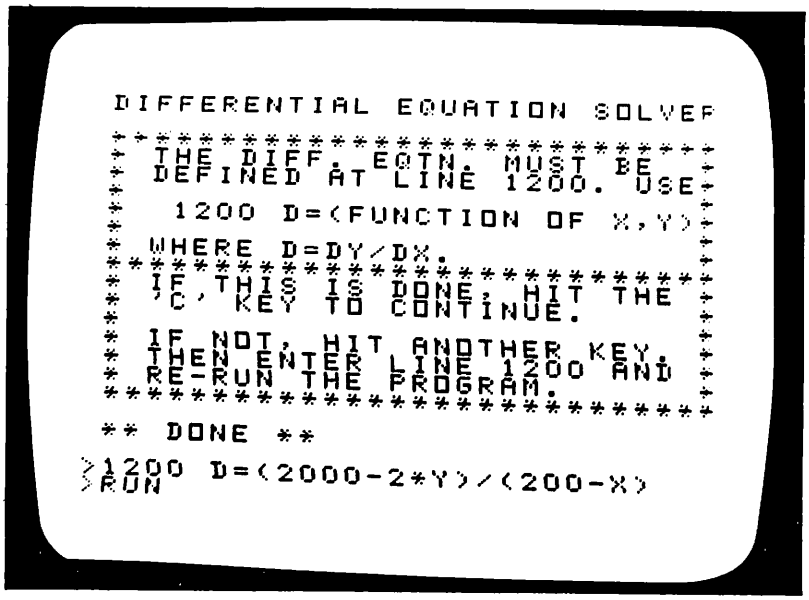
SAMPLE PROBLEM AND RUN

Problem: A body, originally at rest, is subjected to a force of 2000 dynes. Its initial mass is 200 grams. However, while it moves, it loses mass at the rate of 1 gram/sec. There is also an air resistance equal to twice its velocity retarding its movement. The differential equation expressing this motion is:

$$\frac{dY}{dX}=\frac{(2000-2Y)}{(200-X)} \qquad \text{where } Y=\text{velocity (cm./sec.)} \quad X=\text{time (sec.)}$$

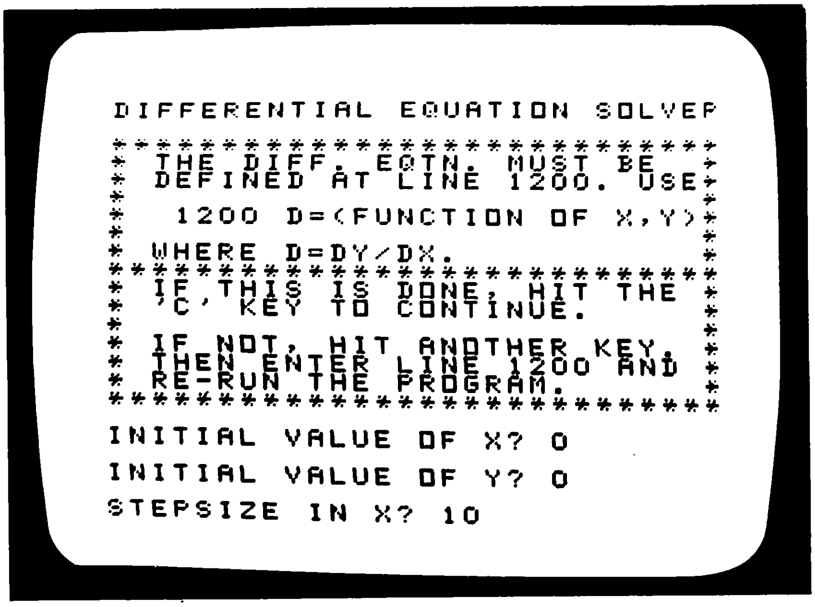
Find the velocity of the body every ten seconds up through two and one-half minutes.

Solution and Sample Run: The solution and sample run are illustrated in the accompanying photographs.



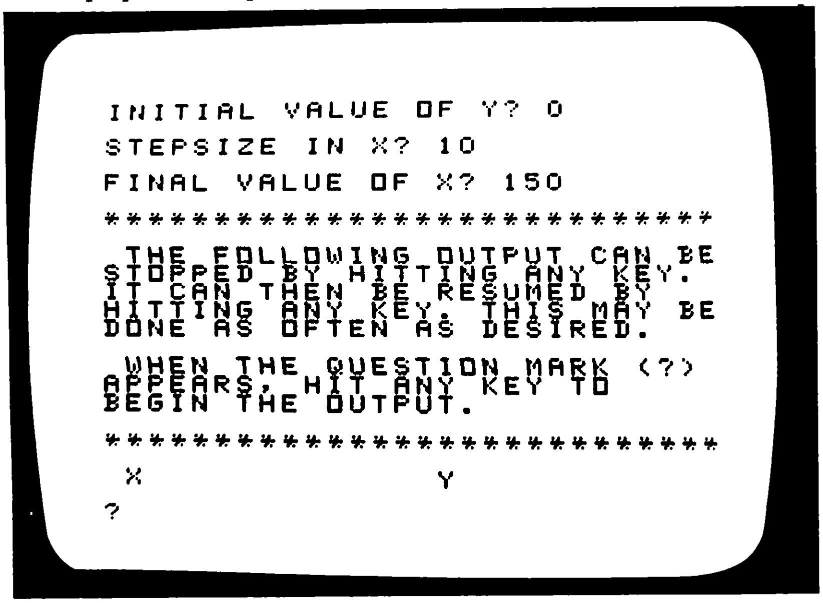
```
DIFFERENTIAL EQUATION SOLVER
****************************
* THE DIFF. EQTN. MUST BE  *
* DEFINED AT LINE 1200. USE*
*   1200 D=(FUNCTION OF X,Y)*
* WHERE D=DY/DX.           *
****************************
* IF THIS IS DONE, HIT THE *
* 'C' KEY TO CONTINUE.     *
*                          *
* IF NOT, HIT ANOTHER KEY. *
* THEN ENTER LINE 1200 AND *
* RE-RUN THE PROGRAM.      *
****************************
** DONE **
>1200 D=(2000-2*Y)/(200-X)
>RUN
```

The operator hits a key to exit from the program. Then he enters the differential equation into line 1200. He types RUN to restart the program.



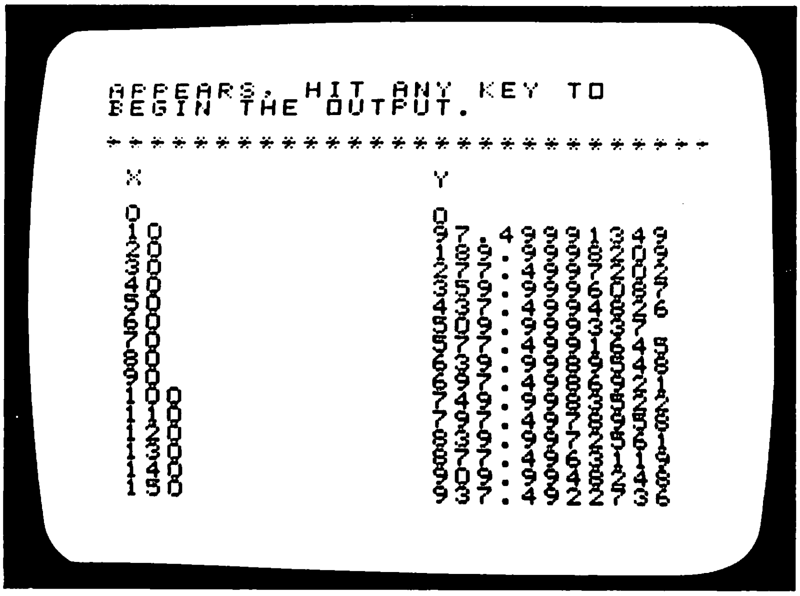
```
DIFFERENTIAL EQUATION SOLVER
******************************
* THE DIFF. EQTN. MUST BE    *
* DEFINED AT LINE 1200. USE*
*                             *
*  1200 D=(FUNCTION OF X,Y)*
*                             *
* WHERE D=DY/DX.              *
******************************
* IF THIS IS DONE, HIT THE   *
* 'C' KEY TO CONTINUE.        *
*                             *
* IF NOT, HIT ANOTHER KEY.   *
* THEN ENTER LINE 1200 AND   *
* RE-RUN THE PROGRAM.        *
******************************
INITIAL VALUE OF X? 0
INITIAL VALUE OF Y? 0
STEPSIZE IN X? 10
```

The operator has hit the C key. The program responds by beginning the input phase. The operator has responded to the first three requests.



```
INITIAL VALUE OF Y? 0
STEPSIZE IN X? 10
FINAL VALUE OF X? 150
****************************
 THE FOLLOWING OUTPUT CAN BE
STOPPED BY HITTING ANY KEY.
IT CAN THEN BE RESUMED BY
HITTING ANY KEY. THIS MAY BE
DONE AS OFTEN AS DESIRED.
 WHEN THE QUESTION MARK (?)
APPEARS, HIT ANY KEY TO
BEGIN THE OUTPUT.
****************************
 X              Y
?
```

The operator has completed the input. The program signals with a question mark that it is waiting for him to hit any key. It will not continue the run until he does so.



```
APPEARS, HIT ANY KEY TO
BEGIN THE OUTPUT.
****************************

 X                      Y

 0                      0
 10                     97.49991349
 20                     189.9998309
 30                     277.4997202
 40                     359.9996087
 50                     437.4994856
 60                     509.999357
 70                     577.4991645
 80                     639.9989548
 90                     697.4986521
 100                    749.9985522
 110                    797.4975556
 120                    839.9970561
 130                    877.4962119
 140                    909.9945240
 150                    937.4922736
```

The operator hits a key and the program responds with the tabulated output. X is time in seconds and Y is velocity in cm/sec.

PROGRAM LISTING

```
100 REM DIFFEQN
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 PRINT "DIFFERENTIAL EQUATION SOLVER"
140 PRINT
150 PRINT
160 PRINT TAB(3);"THE DIFF. EQTN. MUST BE"
170 PRINT TAB(3);"DEFINED AT LINE 1200. USE"
180 PRINT
190 PRINT TAB(4);"1200 D=(FUNCTION OF X,Y)"
200 PRINT
210 PRINT TAB(3);"WHERE D=DY/DX."
220 PRINT
230 PRINT TAB(3);"IF THIS IS DONE, HIT THE"
240 PRINT TAB(3);"'C' KEY TO CONTINUE."
250 PRINT
260 PRINT TAB(3);"IF NOT, HIT ANOTHER KEY."
270 PRINT TAB(3);"THEN ENTER LINE 1200 AND"
280 PRINT TAB(3);"RE-RUN THE PROGRAM."
```

```
290 CALL HCHAR(10,3,42,28)
300 CALL HCHAR(17,3,42,28)
310 CALL HCHAR(24,3,42,28)
320 CALL VCHAR(10,3,42,15)
330 CALL VCHAR(10,30,42,15)
340 PRINT
350 CALL KEY(0,KK,SS)
360 IF SS=0 THEN 350
370 IF KK=67 THEN 390
380 END
390 PRINT
400 INPUT "INITIAL VALUE OF X? ":XX
410 PRINT
420 INPUT "INITIAL VALUE OF Y? ":YY
430 X=XX
440 Y=YY
450 GOSUB 1200
460 PRINT
470 INPUT "STEPSIZE IN X? ":DX
480 PRINT
490 INPUT "FINAL VALUE OF X? ":XF
500 PRINT
510 CALL HCHAR(24,3,42,28)
520 PRINT
530 PRINT
540 PRINT " THE FOLLOWING OUTPUT CAN BE"
550 PRINT "STOPPED BY HITTING ANY KEY."
560 PRINT "IT CAN THEN BE RESUMED BY"
570 PRINT "HITTING ANY KEY. THIS MAY BE"
580 PRINT "DONE AS OFTEN AS DESIRED."
590 PRINT
600 PRINT " WHEN THE QUESTION MARK (?)"
610 PRINT "APPEARS, HIT ANY KEY TO"
620 PRINT "BEGIN THE OUTPUT."
630 PRINT
640 CALL HCHAR(24,3,42,28)
650 PRINT
660 PRINT
670 PRINT " X"," Y"
680 PRINT
690 CALL HCHAR(24,3,63)
700 CALL KEY(0,KK,SS)
```

```
710 IF SS=0 THEN 700
720 CALL HCHAR(24,3,32)
730 PRINT XX,YY
740 GOSUB 960
750 Q=XX+DX
760 IF Q>XF+1.E-5 THEN 380
770 X=XX
780 Y=YY
790 GOSUB 1200
800 K0=D
810 X=XX+DX/2
820 Y=YY+K0*DX/2
830 GOSUB 1200
840 K1=D
850 Y=YY+K1*DX/2
860 GOSUB 1200
870 K2=D
880 X=XX+DX
890 Y=YY+K2*DX
900 GOSUB 1200
910 K3=D
920 DY=DX*(K0+2*K1+2*K2+K3)/6
930 YY=YY+DY
940 XX=XX+DX
950 GOTO 730
960 CALL KEY(0,KK,SS)
970 IF SS<>0 THEN 990
980 RETURN
990 FOR J=1 TO 100
1000 NEXT J
1010 CALL KEY(0,KK,SS)
1020 IF SS=0 THEN 1010
1030 RETURN
1040 PRINT
1050 CALL HCHAR(24,3,43,28)
1060 PRINT
1070 PRINT
1080 PRINT "ERROR! -- YOU HAVE NOT"
1090 PRINT "DEFINED THE DIFFERENTIAL"
1100 PRINT "EQUATION IN LINE 1200."
1110 CALL SOUND(100,300,10)
1120 END
```

```
1130 REM ******************
1140 REM DEFINE THE D.E.
1150 REM AT LINES 1200-2000
1160 REM
1170 REM LINE 1200 MUST BE
1180 REM THE FIRST LINE.
1190 REM ******************
1200 GOTO 1040
2000 RETURN
```

EASY CHANGES

1. If you have already entered the differential equation and wish to skip the introductory output, add this line:

```
145 GOTO 390
```

   This will immediately begin the input dialog.
2. If you wish to use negative stepsizes, line 760 must be changed to:

```
760 IF Q<XF-1.E-5 THEN 380
```

MAIN ROUTINES

120- 340	Displays initial messages.
350- 500	Gets user's input.
510- 620	Displays additional messages.
630- 690	Initializes output display.
700- 720	Waits for user to hit a key to start the output.
730- 740	Displays output.
750- 950	Computes each step.
960-1030	Subroutine to stop and start output.
1040-1120	Error message.
1200-2000	User-supplied subroutine to define D.

MAIN VARIABLES

D	Value of dY/dX.
X, Y	Values of X, Y on current step.
XX, YY	Values of X, Y on last step.
DX	Stepsize in X.
XF	Final value of X.

K0, K1, K2, K3	Runge-Kutta coefficients.
Q	Work variable.
J	Loop index.
KK, SS	User input.

SUGGESTED PROJECTS

1. Add the capability to display the output graphically. During the current tabular phase, the minimum and maximum values of Y can be saved and automatically used as the plot limits for the graphical output.
2. The value of dY/dX as a function of X is often a useful quantity to know. Modify the program to add it to the columnar display and/or the graphical display.
3. The inherent error in the calculation depends on the stepsize chosen. Most cases should be run with different stepsizes to insure that the errors are not large. If the answers do not change much, you can be reasonably certain that your solutions are accurate. Better yet, techniques exist to vary the stepsize during the calculation to insure that the error is sufficiently small during each step. Research these methods and incorporate them into the program.
4. The program can be easily broadened to solve a set of coupled, first order, differential equations simultaneously. This would greatly increase the types of problems that could be solved. Research this procedure and expand the program to handle it.

GRAPH

PURPOSE

Is a picture worth a thousand words? In the case of mathematical functions, the answer is often "yes." A picture, i.e., a graph, enables you to see the important behavior of a function quickly and accurately. Trends, minima, maxima, etc. become easy and convenient to determine.

GRAPH produces a two-dimensional plot of a function that you supply. The function must be in the form Y=(any function of X). The independent variable X will be plotted along the abscissa (horizontal axis). The dependent variable Y will be plotted along the ordinate (vertical axis). You have complete control over the scaling that is used on the X and Y axes.

HOW TO USE IT

Before running the program, you must enter into it the function to be plotted. This is done as a subroutine beginning at line 2100. It must define Y as a function of X. The subroutine will be called with X set to various values. It must then set the variable Y to the correct corresponding value. The subroutine may be as simple or complex as necessary to define the function. It can take one line or several hundred lines. Line 2999 is already set as a RETURN statement, so you need not add another one.

Having entered this subroutine, you are ready to run the program. The program begins by warning you that it assumes the function has already been entered at line 2100. It will then ask you for the domain of X, i.e., the lowest and highest values of X that you wish to have plotted. Values can be positive or negative as long as two conditions are met: the highest value must be algebraically larger than the lowest value and the number of characters the TI-99/4A uses to express each value must be no more than eleven.

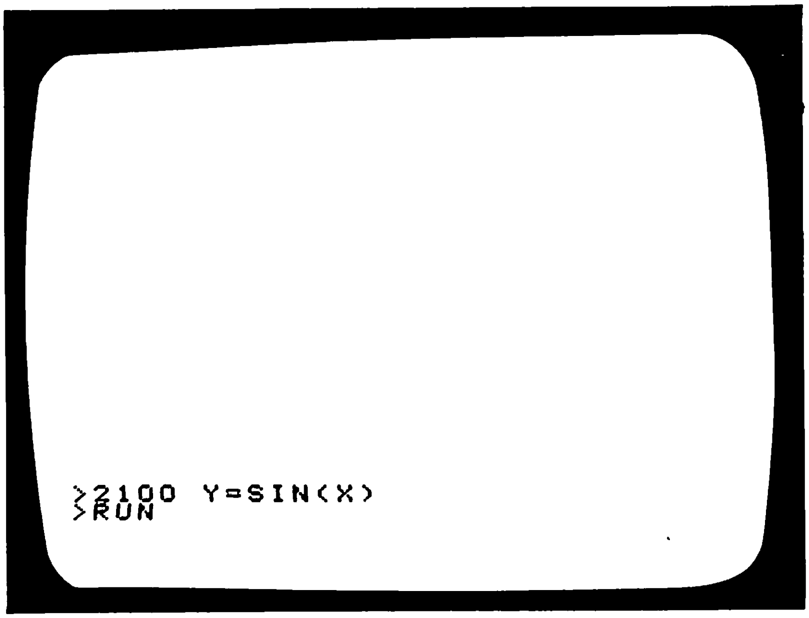
Now you must choose the scale for Y. To do this intelligently, you probably need to know the minimum and maximum values of Y over the domain of X selected. The program finds these values and displays them for you. You must then choose the minimum and maximum values you wish to have on the Y scale. Again, any two values are acceptable as long as they satisfy the two conditions given above, but here the number of characters is limited to seven. (Note: certain numbers entered in scientific or exponential notation can be input with less than seven characters but expanded by the computer to more than seven. If this occurs, when the computer asks you to "reduce the number of digits," simply re-enter the original number and the computer will accept it the second time.)

Now the program will display the plot of your function. Each axis is 20 tick-marks long with the origin defined as the minimum scale values of both X and Y. The minimum and maximum values on each scale are displayed appropriately. If the scaling values for the Y axis are too long to fit on the screen, the program will use an abbreviation for one or both of the values: "YL" for the lower or minimum Y and "YU" for the upper or maximum Y.

The plot is presented with a special "double-precision" blue line. If a value for Y should be off-scale, a special "enlarged red square" will be displayed at the appropriate value of X. If the actual value of Y is too large, it will be plotted at the maximum Y value. Similarly, it will be drawn at the minimum Y value if it is too low.

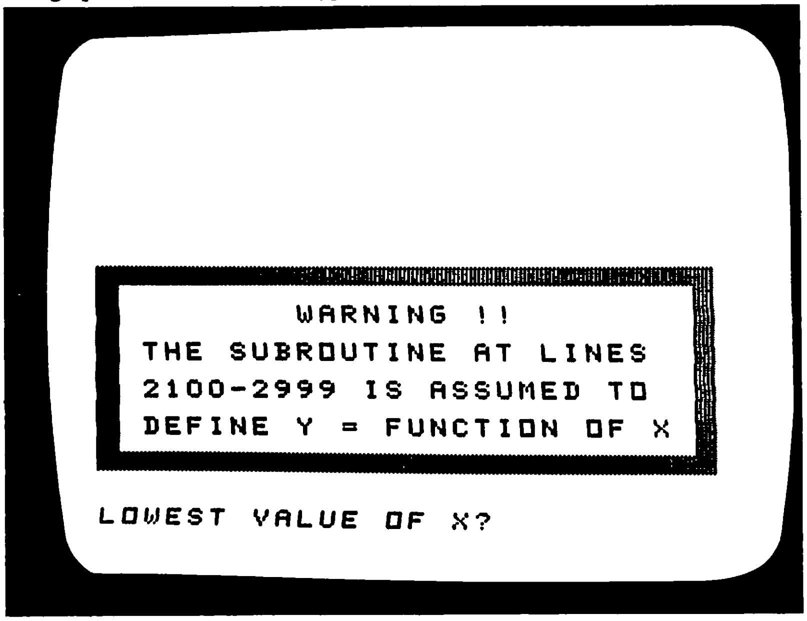
After the plot is drawn, the program will tell you to hit any key to erase the graph and end the program.

SAMPLE RUN



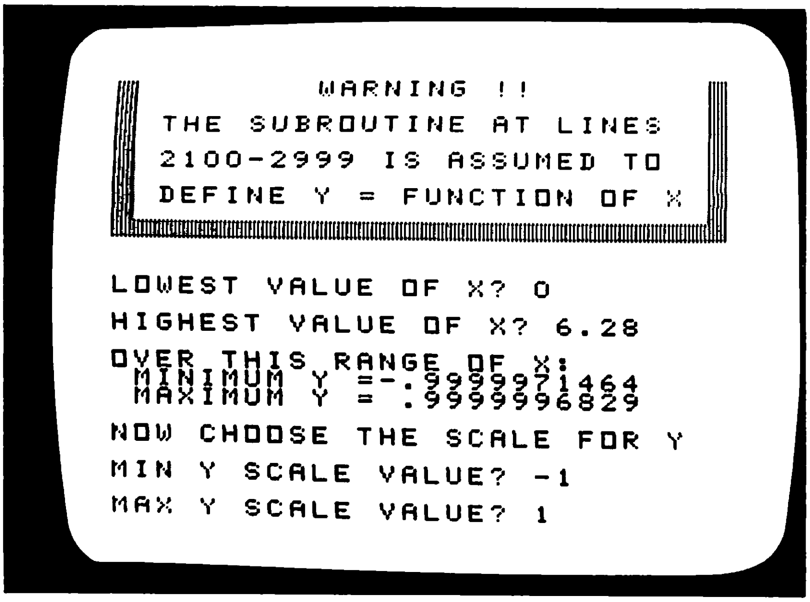
```
>2100 Y=SIN(X)
>RUN
```

After loading the program, the operator enters line 2100 to request the graph Y=SIN(X). RUN is typed to begin the program.



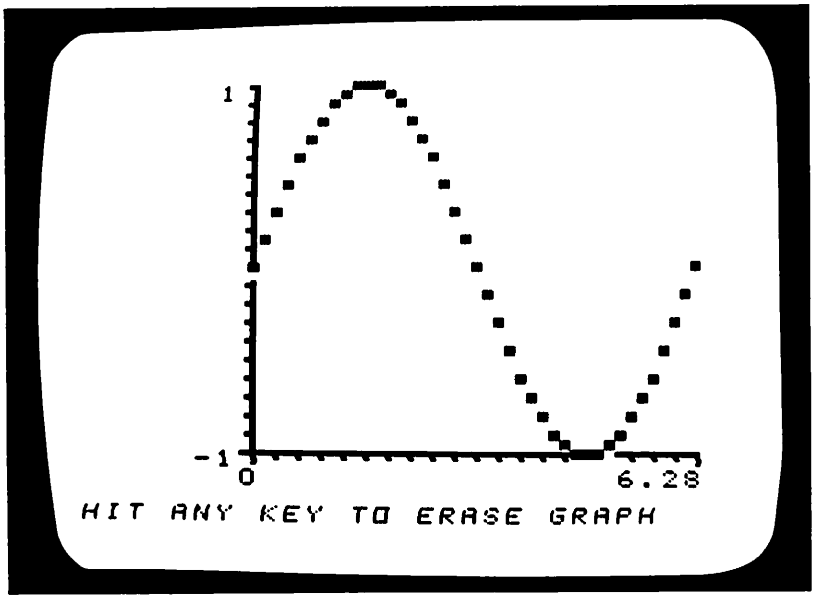
```
           WARNING !!
THE SUBROUTINE AT LINES
2100-2999 IS ASSUMED TO
DEFINE Y = FUNCTION OF X

LOWEST VALUE OF X?
```

After printing a warning, the program initiates the input dialog.



```
        WARNING !!
 THE SUBROUTINE AT LINES
 2100-2999 IS ASSUMED TO
 DEFINE Y = FUNCTION OF X

LOWEST VALUE OF X? 0
HIGHEST VALUE OF X? 6.28
OVER THIS RANGE OF X:
 MINIMUM Y =-.9999971464
 MAXIMUM Y = .9999996829
NOW CHOOSE THE SCALE FOR Y
MIN Y SCALE VALUE? -1
MAX Y SCALE VALUE? 1
```

The input dialog transpires. The operator asks that the domain of X be 0–6.28. The program responds by showing the minimum and maximum values of Y over this domain. The operator chooses an appropriate scale for the Y axis.



```
HIT ANY KEY TO ERASE GRAPH
```

The graph is displayed as requested. The program waits for the operator to hit any key to clear the screen and end the program.

PROGRAM LISTING

```
100 REM GRAPH
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 C=5
140 CA=9
150 HP=10
160 VP=21
170 PRINT TAB(10);"WARNING !!"
180 PRINT
190 PRINT TAB(3);"THE SUBROUTINE AT LINES"
200 PRINT
210 PRINT TAB(3);"2100-2999 IS ASSUMED TO"
220 PRINT
230 PRINT TAB(3);"DEFINE Y = FUNCTION OF X"
240 PRINT
250 PRINT
260 PRINT
270 CALL CHAR(112,"FFFFFFFFFFFFFFFF")
280 CALL COLOR(11,CA,1)
290 CALL HCHAR(12,3,112,28)
300 CALL VCHAR(13,3,112,9)
310 CALL VCHAR(13,30,112,9)
320 CALL HCHAR(22,3,112,28)
330 PRINT
340 INPUT "LOWEST VALUE OF X? ":XL
350 PRINT
360 XL$=STR$(XL)
370 L=LEN(XL$)
380 IF L<12 THEN 410
390 GOSUB 1010
400 GOTO 330
410 INPUT "HIGHEST VALUE OF X? ":XU
420 IF XU>XL THEN 460
430 PRINT "*** BAD X RANGE ***"
440 CALL SOUND(300,200,3)
450 GOTO 330
460 XU$=STR$(XU)
470 L=LEN(XU$)
480 IF L<12 THEN 520
490 GOSUB 1010
500 PRINT
```

```
510 GOTO 410
520 DX=(XU-XL)/40
530 X=XL
540 GOSUB 2100
550 MN=Y
560 MX=Y
570 FOR J=1 TO 40
580 X=XL+J*DX
590 GOSUB 2100
600 IF Y<=MX THEN 620
610 MX=Y
620 IF Y>=MN THEN 640
630 MN=Y
640 NEXT J
650 PRINT
660 PRINT "OVER THIS RANGE OF X:"
670 PRINT " MINIMUM Y =";MN
680 PRINT " MAXIMUM Y =";MX
690 PRINT
700 PRINT "NOW CHOOSE THE SCALE FOR Y"
710 T$="TEST"
720 PRINT
730 INPUT "MIN Y SCALE VALUE? ":YL
740 PRINT
750 YL$=STR$(YL)
760 L=LEN(YL$)
770 IF L<8 THEN 840
780 IF T$<>"YL" THEN 810
790 YL$=T$
800 GOTO 840
810 T$="YL"
820 GOSUB 1010
830 GOTO 720
840 T$="TEST"
850 INPUT "MAX Y SCALE VALUE? ":YU
860 PRINT
870 IF YU>YL THEN 910
880 PRINT "*** BAD Y RANGE ***"
890 CALL SOUND(300,200,3)
900 GOTO 650
910 YU$=STR$(YU)
920 L=LEN(YU$)
```

```
930 IF L<8 THEN 1040
940 IF T$<>"YU" THEN 970
950 YU$="YU"
960 GOTO 1040
970 T$="YU"
980 GOSUB 1010
990 PRINT
1000 GOTO 850
1010 PRINT "PLEASE REDUCE # OF DIGITS"
1020 CALL SOUND(300,200,3)
1030 RETURN
1040 CALL CLEAR
1050 CALL CHAR(129,"061E1E0606060606")
1060 CALL CHAR(130,"00FFFF0606")
1070 CALL CHAR(131,"067F7F06060606")
1080 CALL CHAR(132,"001E1E0606060606")
1090 CALL CHAR(133,"00FEFE06060606")
1100 CALL VCHAR(VP-20,HP,132)
1110 CALL VCHAR(VP-19,HP,129,19)
1120 CALL VCHAR(VP,HP,131)
1130 CALL HCHAR(VP,HP+1,130,19)
1140 CALL HCHAR(VP,HP+20,133)
1150 CALL CHAR(136,"00000000F0F0F0F0")
1160 CALL CHAR(137,"F0F0F0F0")
1170 CALL CHAR(138,"0F0F0F0F")
1180 CALL CHAR(139,"000000000F0F0F0F")
1190 CALL CHAR(140,"00000000FFFFFFFF")
1200 CALL CHAR(141,"0F0F0F0FF0F0F0F0")
1210 CALL CHAR(142,"FFFFFFFF")
1220 CALL CHAR(143,"F0F0F0F00F0F0F0F")
1230 CALL COLOR(14,C,1)
1240 L=LEN(YU$)
1250 H=HP-L-1
1260 FOR J=1 TO L
1270 CALL HCHAR(VP-20,H+J,ASC(SEG$(YU$,J,1)))
1280 NEXT J
1290 L=LEN(YL$)
1300 H=HP-L-1
1310 FOR J=1 TO L
1320 CALL HCHAR(VP,H+J,ASC(SEG$(YL$,J,1)))
1330 NEXT J
1340 L=LEN(XL$)
```

```
1350 V=VP+1
1360 H=HP-INT(L/2)-1
1370 FOR J=1 TO L
1380 CALL HCHAR(V,H+J,ASC(SEG$(XL$,J,1)))
1390 NEXT J
1400 L=LEN(XU$)
1410 H=HP+20-L
1420 FOR J=1 TO L
1430 CALL HCHAR(V,H+J,ASC(SEG$(XU$,J,1)))
1440 NEXT J
1450 DX=(XU-XL)/20
1460 DY=(YU-YL)/20
1470 X=XL
1480 GOSUB 1910
1490 CC=138
1500 IF TF=1 THEN 1540
1510 CC=139
1520 IF TF=0 THEN 1540
1530 CC=112
1540 CALL HCHAR(V,HP,CC)
1550 FOR J=1 TO 20
1560 X=XL+DX*(J-0.5)
1570 GOSUB 1910
1580 CC=137
1590 IF TF=1 THEN 1630
1600 CC=136
1610 IF TF=0 THEN 1630
1620 CC=112
1630 CALL HCHAR(V,HP+J,CC)
1640 IF TF=2 THEN 1850
1650 X=X+DX/2
1660 GOSUB 1910
1670 IF TF<>2 THEN 1700
1680 CC=112
1690 GOTO 1840
1700 CALL GCHAR(V,HP+J,CD)
1710 IF CD=136 THEN 1770
1720 IF CD=137 THEN 1810
1730 CC=138
1740 IF TF=1 THEN 1840
1750 CC=139
1760 GOTO 1840
```

```
1770 CC=141
1780 IF TF=1 THEN 1840
1790 CC=140
1800 GOTO 1840
1810 CC=142
1820 IF TF=1 THEN 1840
1830 CC=143
1840 CALL HCHAR(V,HP+J,CC)
1850 NEXT J
1860 PRINT "HIT ANY KEY TO ERASE GRAPH";
1870 CALL KEY(0,KK,SS)
1880 IF SS=0 THEN 1870
1890 CALL CLEAR
1900 END
1910 GOSUB 2100
1920 YY=(Y-YL)/DY
1930 V=VP-INT(YY+0.75)
1940 T=YY-INT(YY)
1950 TF=1
1960 IF T<0.25 THEN 1990
1970 IF T>0.75 THEN 1990
1980 TF=0
1990 IF V<=VP THEN 2030
2000 V=VP
2010 TF=2
2020 GOTO 2060
2030 IF V>=(VP-20)THEN 2060
2040 V=VP-20
2050 TF=2
2060 RETURN
2070 REM ******************
2080 REM SET Y=F(X) AT 2100
2090 REM ******************
2100 REM Y=F(X) GOES HERE
2999 RETURN
```

EASY CHANGES

1. You may want the program to self-scale the Y axis for you. That is, you want it to use the minimum and maximum Y values that it finds as the limits on the Y axis. This can be accomplished by adding the following lines:

```
 645 GOTO 3000
3000 YL=MN
3010 YU=MX
3020 IF YL> =YU THEN 650
3030 YL$="YL"
3040 IF LEN(STR$(YL))>7 THEN 3060
3050 YL$=STR$(YL)
3060 YU$="YU"
3070 IF LEN(STR$(YU))>7 THEN 1040
3080 YU$=STR$(YU)
3090 GOTO 1040
```

(Note: on rare occasions, the program will not be able to perform the desired self-scaling. In these cases, the program will revert to requesting the Y scaling from you.)

2. The colors of the normal plot character and the special off-axis character can be easily changed. They are controlled by the variables C and CA respectively; currently they are set to dark blue and medium red in lines 130 and 140. Should you desire a magenta plot with white off-axis characters, make these changes

```
130 C=14
140 CA=16
```

You may set these variables to any number from 2–16. Consult your BASIC manual for the resulting colors.

3. Do you sometimes forget to enter the subroutine at line 2100 despite the introductory warning? As is, the program will plot the straight line Y=0 if you do this. If you want a more drastic reaction to prevent this, change line 2100 to read:

```
2100 Y=SQR(-1)
```

Now, if you don't enter the actual subroutine desired, the program will stop and print the following message after you enter the X scaling values:

```
* BAD ARGUMENT IN 2100
```

MAIN ROUTINES

120– 160	Clears screen and initializes variables.
170– 320	Displays introductory warning.
330– 510	Gets X-scaling from user.

520–1000	Determines minimum, maximum Y values; gets Y scale from user.
1010–1030	Subroutine to request user to shorten input.
1040–1140	Draws plot axes.
1150–1230	Defines special graphics character codes.
1240–1440	Labels plot axes.
1450–1850	Draws plot.
1860–1900	Waits for user to hit any key to end program.
1910–2060	Subroutine to determine Y location of a point.
2100–2999	User-supplied subroutine to evaluate Y as a function of X.

MAIN VARIABLES

XL, XU	Lower, upper scale values of X.
YL, YU	Lower, upper scale values of Y.
DX, DY	Scale increments of X, Y.
X, Y	Current values of X, Y.
XL$, XU$	String representation of XL, XU.
YL$, YU$	String representation of YL, YU.
H, V	Horizontal, vertical position in plotting units.
HP, VP	Horizontal, vertical position of axes origin.
YY	Y direction offset.
TF	Y position flag (0=low, 1=high, 2=out of bounds).
KK, SS	User input.
MN, MX	Minimum, maximum values of Y.
J	Loop variable.
C	Color of "regular" point.
CA	Color of "off-axis" point.
L	Length of a string.
T$	Temporary string.
CC, CD	Character code value.
T	Y position inside one plot unit.

SUGGESTED PROJECTS

1. Determine and display the values of X at which the minimum and maximum values of Y occur.
2. After the graph is plotted, allow the user to obtain the exact value of Y for any given X.

INTEGRATE

PURPOSE AND DEFINITION

The need to evaluate integrals occurs frequently in much scientific and mathematical work. This program will numerically integrate a function that you supply using a technique known as Simpson's rule. It will continue to grind out successive approximations of the integral until you are satisfied with the accuracy of the solution.

Mathematical integration will probably be a familiar term to those who have studied some higher mathematics. It is a fundamental subject of second-year calculus. The integral of a function between the limits $x = \ell$ (lower limit) and $x = u$ (upper limit) represents the area under its curve; i.e., the shaded area in Figure 1.

We may approximate the integral by first dividing up the area into rectangular strips or segments. We can get a good estimate of the total integral by summing the areas of these segments by using a parabolic fit across the top. For those who understand some mathematical theory, Simpson's rule may be expressed as

$$\int_{x=\ell}^{x=u} f(x)\,dx \cong \frac{\Delta}{3}\left\{ f(\ell) + f(u) \right.$$
$$\left. +4 \sum_{j=1}^{N/2} f[\ell + \Delta(2j-1)] + 2 \sum_{j=1}^{(N-2)/2} f[\ell + 2\Delta j] \right\}$$

Here N is the number of segments into which the total interval is divided. N is 4 in the diagram.

For a good discussion of the numerical evaluation of integrals see: McCracken, Dorn, *Numerical Methods and FORTRAN Programming*, New York, Wiley, 1964, p. 160. Don't let the word "FORTRAN" scare you away. The discussions in the book are independent of programming language with only some program examples written in FORTRAN.

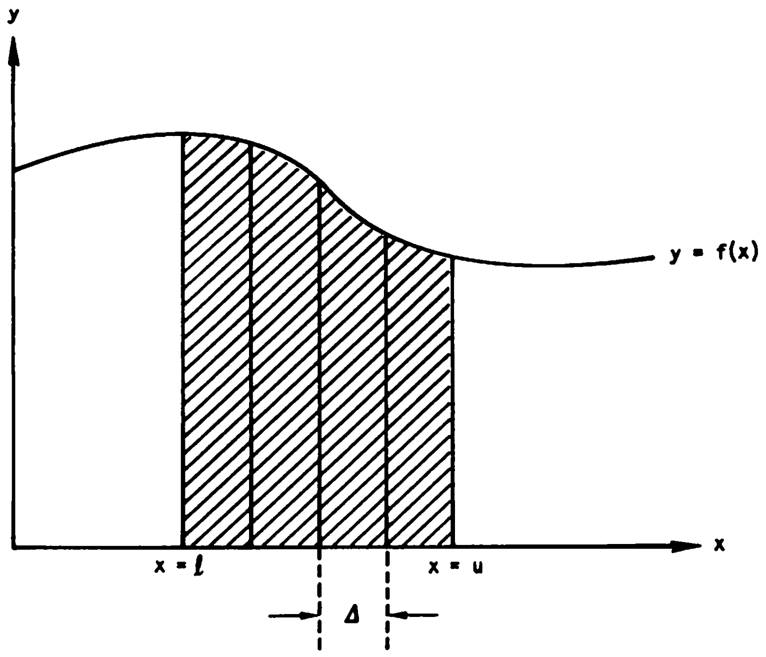


Figure 1. The Integral of f(x)

HOW TO USE IT

The program begins with a warning! This is to remind you that you should have already entered the subroutine to evaluate Y as a function of X. This subroutine must start at line 700. More about it shortly.

You will then be asked to provide the lower and upper limits of the integration domain. Any numerical values are acceptable. It is not even necessary that the lower limit of X be smaller than the upper one.

The program will now begin displaying its numerical evaluations of the integral. The number of segments used in the calculation continually doubles. This causes the accuracy of the integral to increase at the expense of additional computation time. For most functions, you should see the value of the integral converging quickly to a constant (or near constant) value. This, of course, will be the best numerical evaluation of the integral at hand.

When you are satisfied with the accuracy of the solution, you must hit **FCTN-CLEAR** to terminate the program. If not, the program will run forever (assuming you can pay the electric bills). The amount of computation is approximately doubled each step. This means it will take the computer about the same amount of time to compute the next step that it took to compute *all* the previous steps. Thus, it will soon be taking the TI-99/4A hours, days, and weeks to compute steps. Eventually, round-off errors begin degrading the results, causing a nice, constant, converged solution to change. However, the high precision of the computer's floating point arithmetic will postpone this for quite a while. You will probably lose patience before seeing it.

The function to be integrated can be as simple or as complicated as you desire. It may take one line or a few hundred lines of code. In any case, the subroutine to express it must start at line 700. Line 999 already contains a **RETURN** statement so you need not add another one. This subroutine will be continually called with the variable X set. When it returns, it should have set the variable Y to the corresponding value of the function for the given X. The subroutine must be able to evaluate the function at any value of X between the lower and upper bounds of the integration domain.

If your function consists of experimental data at discrete values of X, you must do something to enable the subroutine to evaluate the function at intermediate values of X. We recommend one of two approaches. First, you could write the subroutine to linearly interpolate the value of Y between the appropriate values of X. This will involve searching your data table for the pair of experimental X values that bound the value of X where the function is to be evaluated. Secondly, the program CURVE presented elsewhere in this section can produce an approximate polynomial expression to fit your experimental data. This expression can then be easily entered as the subroutine at line 700.

By the way, Simpson's rule is *exact* for any polynomial of degree three or less. This means that if the function can be written in the form

```
Y=A*X↑3+B*X↑2+C*X+D
```

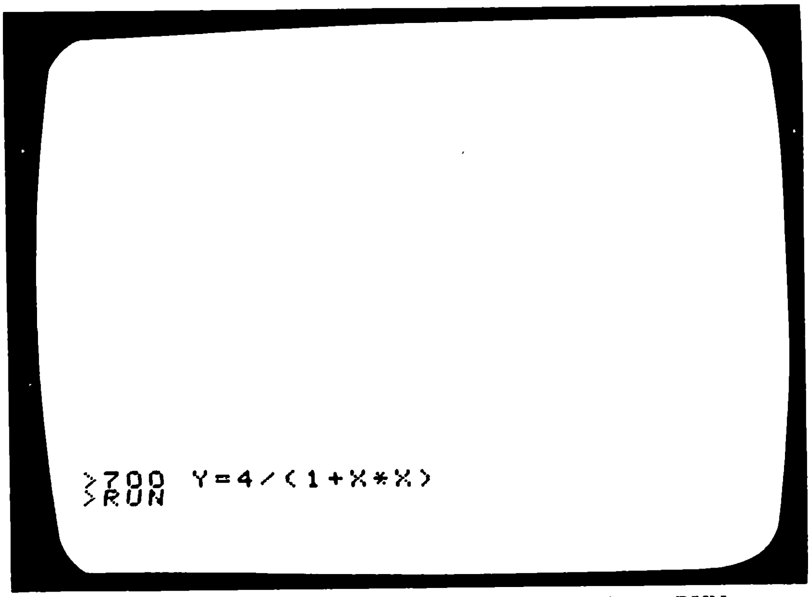
where A, B, C, and D are constants, the program will calculate the integral exactly even with only two segments.

The sample run illustrates the following integration:

$$\int_{x=0}^{x=1} \frac{4}{1+x^2}\,dx$$

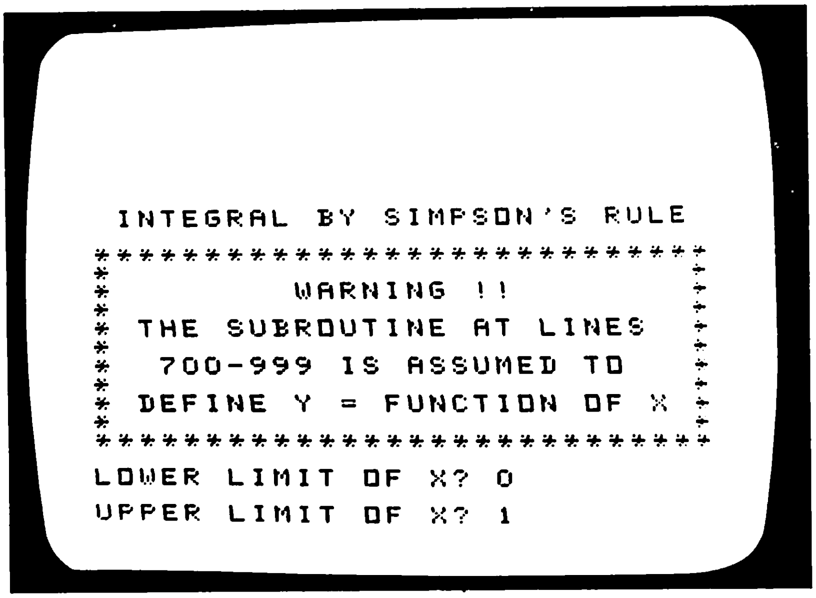
This integral has the theoretical value of π (pi) as the correct answer! Pi, as you may know, has the value 3.1415926535. . . .Before the run is started, the above function is entered at line 700.

SAMPLE RUN



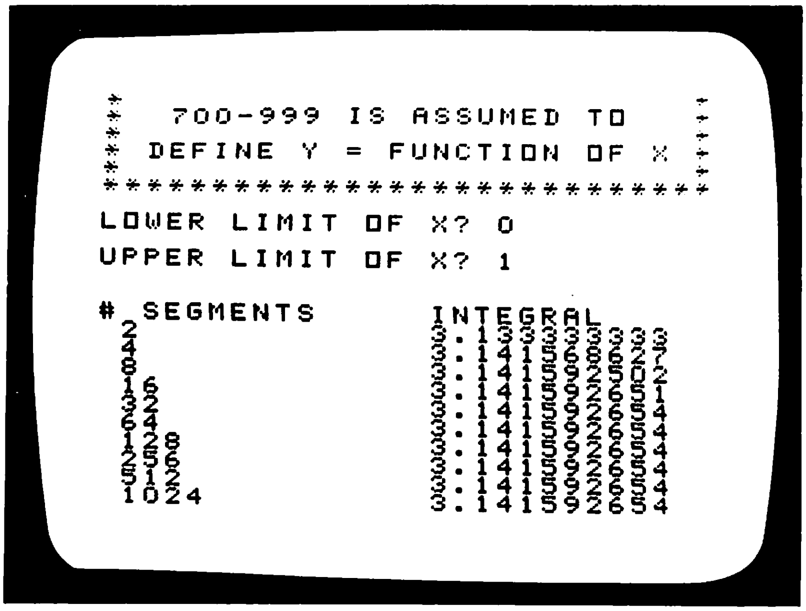
```
>700  Y=4/(1+X*X)
>RUN
```

The operator enters the integrand function at line 700 and types RUN to start the program.



```
  INTEGRAL BY SIMPSON'S RULE
****************************
*                          *
*          WARNING !!      *
*                          *
* THE SUBROUTINE AT LINES  *
*                          *
*  700-999 IS ASSUMED TO   *
*                          *
* DEFINE Y = FUNCTION OF X *
*                          *
****************************
LOWER LIMIT OF X? 0
UPPER LIMIT OF X? 1
```

The lower and upper bounds of the integration of X are entered as requested.



```
*
*   700-999 IS ASSUMED TO    *
*                            *
*  DEFINE Y = FUNCTION OF X  *
*                            *
******************************
LOWER LIMIT OF X? 0
UPPER LIMIT OF X? 1

# SEGMENTS           INTEGRAL
 2                   3.133333333
 4                   3.141568627
 8                   3.141592502
 16                  3.141592651
 32                  3.141592654
 64                  3.141592654
 128                 3.141592654
 256                 3.141592654
 512                 3.141592654
 1024                3.141592654
```

The results are computed up to 1024 segments. The **FCTN-CLEAR** keys must be pressed to end the calculation.

PROGRAM LISTING

```
100 REM INTEGRATE
110 REM (C) 1984 DILITHIUM PRESS
120 N=2
130 CALL CLEAR
140 PRINT TAB(2);"INTEGRAL BY SIMPSON'S RULE"
150 PRINT
160 PRINT
170 PRINT
180 PRINT TAB(10);"WARNING !!"
190 PRINT
200 PRINT TAB(3);"THE SUBROUTINE AT LINES"
210 PRINT
220 PRINT TAB(4);"700-999 IS ASSUMED TO"
230 PRINT
240 PRINT TAB(3);"DEFINE Y = FUNCTION OF X"
250 PRINT
260 CALL HCHAR(14,3,42,28)
270 CALL HCHAR(24,3,42,28)
280 CALL VCHAR(15,3,42,9)
```

```
290 CALL VCHAR(15,30,42,9)
300 PRINT
310 PRINT
320 INPUT "LOWER LIMIT OF X? ":L
330 PRINT
340 INPUT "UPPER LIMIT OF X? ":U
350 PRINT
360 PRINT
370 PRINT "# SEGMENTS";TAB(16);"INTEGRAL"
380 DX=(U-L)/N
390 T=0
400 X=L
410 GOSUB 700
420 T=T+Y
430 X=U
440 GOSUB 700
450 T=T+Y
460 M=N/2
470 Z=0
480 FOR J=1 TO M
490 X=L+DX*(2*J-1)
500 GOSUB 700
510 Z=Z+Y
520 NEXT J
530 T=T+4*Z
540 M=M-1
550 IF M=0 THEN 630
560 Z=0
570 FOR J=1 TO M
580 X=L+DX*2*J
590 GOSUB 700
600 Z=Z+Y
610 NEXT J
620 T=T+2*Z
630 A=DX*T/3
640 PRINT N,A
650 N=N*2
660 GOTO 380
670 REM *****************
680 REM SET Y=F(X) AT 700
690 REM *****************
700 REM Y=F(X) GOES HERE
999 RETURN
```

EASY CHANGES

1. You might want the program to stop calculation after the integral has been evaluated for a given number of segments. Adding the following lines will cause the program to stop after the integral is evaluated for a number of segments greater than or equal to 100.

```
642 IF N<100 THEN 650
644 END
```

   Of course, you may use any value you wish instead of 100.

2. Perhaps you would like to see the number of segments change at a different rate during the course of the calculation. This can be done by modifying line 650. To increase the rate of change, try

```
650 N=N*4
```

   To change it at a constant (and slower) rate, try

```
650 N=N+50
```

   Be sure, however, that the value of N is always even.

MAIN ROUTINES

120	Initializes N.
130–310	Displays introductory messages and warning.
320–340	Gets integration limits from operator.
350–370	Displays column headings.
380–450	Computes integral contribution from end points.
460–530	Adds contribution from one summation.
540–620	Adds contribution from other summation.
630–660	Completes integral calculation and displays it. Increases number of segments and restarts calculation.
700–999	User-supplied subroutine to evaluate f(X).

MAIN VARIABLES

N	Number of segments.
J	Loop index.
L, U	Lower, upper integration limit of x.
DX	Width of one segment.
T	Partial result of integral.
M	Number of summations.
Z	Subtotal of summations.

A	Value of integral.
X	Current value of x.
Y	Current value of the function y=f(x).

SUGGESTED PROJECTS

1. Research other similar techniques for numerical integration such as the simpler trapezoid rule. Then add a column of output computing the integral with this new method. Compare how the two methods converge toward the (hopefully) correct answer.

SIMEQN

PURPOSE

This program solves a set of simultaneous linear algebraic equations. This type of problem often arises in scientific and numerical work. Algebra students encounter them regularly—many "word" problems can be solved by constructing the proper set of simultaneous equations.

The program can handle up to 20 equations in 20 unknowns. This should prove more than sufficient for any practical application.

The equations to be solved can be written mathematically as follows:

$$\begin{array}{l} A_{11}X_1+A_{12}X_2+\ldots+A_{1N}X_N=R_1 \\ A_{21}X_1+A_{22}X_2+\ldots+A_{2N}X_N=R_2 \\ \vdots \qquad \vdots \qquad\qquad \vdots \qquad \vdots \\ A_{N1}X_1+A_{N2}X_2+\ldots+A_{NN}X_N=R_N \end{array}$$

N is the number of equations and thus the number of unknowns also. The unknowns are denoted X_1 through X_N.

Each equation contains a coefficient multiplier for each unknown and a right-hand-side term. These coefficients (the A matrix) and the right-hand-sides (R_1 through R_N) must be constants—positive, negative, or zero. The A matrix is denoted with double subscripts. The first subscript is the equation number and the second one is the unknown that the coefficient multiplies.

HOW TO USE IT

The program will prompt you for all necessary inputs. First, it asks how many equations (and thus how many unknowns) comprise your set. This number must be at least 1 and no more than 20.

Next, you must enter the coefficients and right-hand-sides for each equation. The program will request these one at a time, continually indicating which term it is expecting next.

Once it has all your inputs, the program begins calculating the solution. This may take a little while if the value of N is high. The program ends by displaying the answers. These, of course, are the values of each of the unknowns, X_1 through X_N.

If you are interested, the numerical technique used to solve the equations is known as Gaussian elimination. Row interchange to achieve pivotal condensation is employed. (This keeps maximum significance in the numbers.) Then back substitution is used to arrive at the final results. This technique is much simpler than it sounds and is described well in the numerical analysis books referenced in the bibiliography.

SAMPLE PROBLEM AND RUN

Problem: A painter has a large supply of three different colors of paint: dark green, light green, and pure blue. The dark green is 30% blue pigment, 20% yellow pigment, and the rest base. The light green is 10% blue pigment, 35% yellow pigment, and the rest base. The pure blue is 90% blue pigment, no yellow pigment, and the rest base. The painter, however, needs a medium green to be composed of 25% blue pigment, 25% yellow pigment, and the rest base. In what percentages should he mix his paints to achieve this mixture?

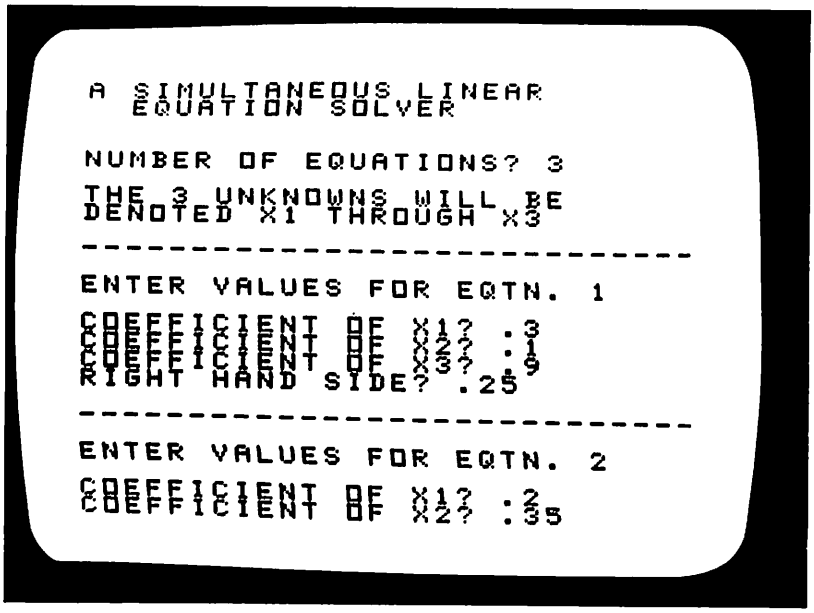
Solution: Let X_1 = percent of dark green to use,
X_2 = percent of light green to use,
X_3 = percent of pure blue to use.

The problem leads to these three simultaneous equations to solve:

$$\begin{aligned} 0.3X_1 + 0.1X_2 + 0.9X_3 &= 0.25 \\ 0.2X_1 + 0.35X_2 \qquad &= 0.25 \\ X_1 + X_2 + X_3 &= 1.0 \end{aligned}$$

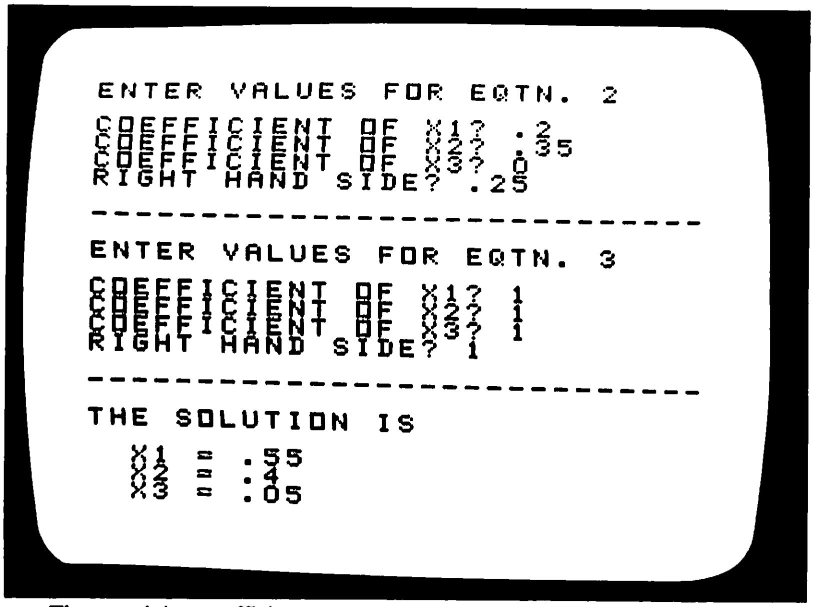
The first equation expresses the amount of blue pigment in the mixture. The second equation is for the yellow pigment. The third equation states that the mixture is composed entirely of the three given paints. (Note that all the percentages are expressed as numbers from 0–1.) The problem leads to the following use of SIMEQN.

SAMPLE RUN



```
A SIMULTANEOUS LINEAR
  EQUATION SOLVER

NUMBER OF EQUATIONS? 3

THE 3 UNKNOWNS WILL BE
DENOTED X1 THROUGH X3
----------------------------

ENTER VALUES FOR EQTN. 1

COEFFICIENT OF X1? .3
COEFFICIENT OF X2? .1
COEFFICIENT OF X3? .9
RIGHT HAND SIDE? .25
----------------------------

ENTER VALUES FOR EQTN. 2

COEFFICIENT OF X1? .2
COEFFICIENT OF X2? .35
```

The operator chooses to solve a set of three simultaneous equations and then begins entering coefficients.



```
ENTER VALUES FOR EQTN. 2

COEFFICIENT OF X1? .2
COEFFICIENT OF X2? .35
COEFFICIENT OF X3? 0
RIGHT HAND SIDE? .25
----------------------------

ENTER VALUES FOR EQTN. 3

COEFFICIENT OF X1? 1
COEFFICIENT OF X2? 1
COEFFICIENT OF X3? 1
RIGHT HAND SIDE? 1
----------------------------

THE SOLUTION IS

   X1 = .55
   X2 = .4
   X3 = .05
```

The remaining coefficients are entered and the computer provides the solution. The painter should use a mixture of 55% dark green, 40% light green, and 5% pure blue.

PROGRAM LISTING

```
100 REM SIMEQN
110 REM (C) 1984 DILITHIUM PRESS
120 M=20
130 DIM A(20,20),R(20),V(20)
140 CALL CLEAR
150 PRINT "A SIMULTANEOUS LINEAR"
160 PRINT TAB(3);"EQUATION SOLVER"
170 PRINT
180 PRINT
190 INPUT "NUMBER OF EQUATIONS? ":N
200 N=INT(N)
210 IF N<1 THEN 240
220 IF N>M THEN 240
230 GOTO 290
240 PRINT
250 PRINT "ERROR - IT MUST BE BETWEEN"
260 PRINT "1 AND";M
270 CALL SOUND(300,200,3)
280 GOTO 180
290 PRINT
300 PRINT "THE";N;"UNKNOWNS WILL BE"
310 PRINT "DENOTED X1 THROUGH X";STR$(N)
320 PRINT
330 CALL HCHAR(24,3,45,28)
340 PRINT
350 FOR J=1 TO N
360 PRINT
370 PRINT "ENTER VALUES FOR EQTN.";J
380 PRINT
390 FOR K=1 TO N
400 PRINT "COEFFICIENT OF X";STR$(K);
410 INPUT A(J,K)
420 NEXT K
430 INPUT "RIGHT HAND SIDE? ":R(J)
440 PRINT
450 CALL HCHAR(24,3,45,28)
460 PRINT
470 NEXT J
480 GOSUB 560
490 PRINT
```

```
500 PRINT "THE SOLUTION IS"
510 PRINT
520 FOR J=1 TO N
530 PRINT TAB(3);"X";STR$(J);" = ";STR$(V(J))
540 NEXT J
550 END
560 IF N<>1 THEN 590
570 V(1)=R(1)/A(1,1)
580 RETURN
590 FOR K=1 TO N-1
600 I=K+1
610 L=K
620 IF ABS(A(I,K))<=ABS(A(L,K))THEN 640
630 L=I
640 IF I>=N THEN 670
650 I=I+1
660 GOTO 620
670 IF L=K THEN 760
680 FOR J=K TO N
690 Q=A(K,J)
700 A(K,J)=A(L,J)
710 A(L,J)=Q
720 NEXT J
730 Q=R(K)
740 R(K)=R(L)
750 R(L)=Q
760 I=K+1
770 Q=A(I,K)/A(K,K)
780 A(I,K)=0
790 FOR J=K+1 TO N
800 A(I,J)=A(I,J)-Q*A(K,J)
810 NEXT J
820 R(I)=R(I)-Q*R(K)
830 IF I>=N THEN 860
840 I=I+1
850 GOTO 770
860 NEXT K
870 V(N)=R(N)/A(N,N)
880 FOR I=N-1 TO 1 STEP -1
890 Q=0
900 FOR J=I+1 TO N
```

```
910 Q=Q+A(I,J)*V(J)
920 V(I)=(R(I)-Q)/A(I,I)
930 NEXT J
940 NEXT I
950 RETURN
```

EASY CHANGES

1. The program is currently set to allow a maximum of 20 equations. If you should somehow need more, and your system has the available memory to handle it, change the value of M in line 120 to the maximum number of equations needed. For example, to allow up to 50 equations in 50 unknowns, change line 120 to read

```
120 M=50
```

   If you use a number greater than 20 you must change the array arguments in line 130 to this number.
2. You may be surprised sometime to see the program fail completely and display a message similar to this:

```
* WARNING
  NUMBER IS TOO BIG
```

   This means your input coefficients (the A array) were ill-conditioned and no solution was possible. This can arise from a variety of causes; e.g., if one equation is an exact multiple of another, or if *every* coefficient of one particular unknown is zero. If you would like the program to print a diagnostic message in these cases, add these lines:

```
862 IF A(N,N)< >0 THEN 870
864 PRINT
866 PRINT "ILL-CONDITIONED INPUT"
868 STOP
```

MAIN ROUTINES

120–130	Initializes variables.
140–180	Clears screen and displays program title.
190–480	Gets input from user and calculates the solution.
490–550	Displays the solution.
560–950	Subroutine to calculate the solution; consisting of the following parts:

560–580	Forms solution if N = 1.
590–860	Gaussian elimination.
610–760	Interchanges rows to achieve pivotal condensation.
870–950	Back substitution.

MAIN VARIABLES

I, J, K, L	Loop indices and subscripts.
N	Number of equations (thus number of unknowns also).
A	Doubly dimensioned array of the coefficients.
R	Array of right-hand-sides.
V	Array of the solution.
Q	Work variable.
M	Maximum allowable number of equations.

SUGGESTED PROJECTS

1. The program modifies the A and R arrays while computing the answer. This means that the original input cannot be displayed after it is input. Modify the program to save the information and enable the user to retrieve it after the solution is given.
2. Currently, a mistake in typing input cannot be corrected once the **ENTER** key is pressed after typing a number. Modify the program to allow correcting previous input.

STATS

PURPOSE

Ever think of yourself as a statistic? Many times we lament at how we have become just numbers in various computer memories, or we simply moan at our insurance premiums. To most people, the word "statistics" carries a negative connotation. To invoke statistics is almost to be deceitful, or at least dehumanizing. But really, we all use statistical ideas regularly. When we speak of things like "she was of average height" or the "hottest weather in years," we are making observations in statistical terms. It is difficult not to encounter statistics in our lives, and this book is no exception.

Of course, when used properly, statistics can be a powerful analytical tool. STATS analyzes a set of numerical data that you provide. It will compile your list, order it sequentially, and/or determine several statistical parameters which describe it.

This should prove useful in a wide variety of applications. Teachers might determine grades by analyzing a set of test scores. A businessman might determine marketing strategy by studying a list of sales to clients. Little leaguers always like to pore over the current batting and pitching averages. You can probably think of many other applications.

HOW TO USE IT

Before entering the data, the program will ask whether or not you wish to use identifiers with the data values. These identifiers can be anything associated with the data: e.g., names accompanying test scores, cities accompanying population values, corporations accompanying sales figures, etc. Hit the **Y** or **N** key to indicate yes or no regarding the use of identifiers. Then hit the **ENTER** key.

Next, your data list must be entered. The program will prompt you for each value with a question mark. If identifiers are being used, you will be prompted for them before being asked for the associated data value. You may use any length character strings you desire for identifiers. However, if you limit them to a maximum of 12 characters, the formatting of later output will be "cleaner."

Two special inputs, *END and *BACK, may be used at any time during this data input phase. They are applicable whether or not identifiers are being used. To signal the end of data, input the four-character string, *END, in response to the (last) question mark. You must, of course, enter at least one data value.

If you discover that you have made a mistake, the five-character string, *BACK, can be used to back up the input process. This will cause the program to re-prompt you for the previous entry. By successive uses of *BACK you can return to any previous position.

With the input completed, the program enters a command mode. You have four options to continue the run:

1) List the data in the order input
2) List the data in ranking order
3) Display statistical parameters
4) End the program

Simply input the number **1**, **2**, **3**, or **4** to indicate your choice. If one of the first three is selected, the program will perform the selected function and return to this command mode to allow another choice. This will continue until you choose **4** to terminate the run. A description of the various options now follows.

Options 1 and 2 provide lists of the data. Option 1 does it in the original input order while option 2 sorts the data from highest value to lowest. In either case the identifiers, if used, will be shown alongside their associated values.

The lists are started by hitting any key when told to do so. Either list may be temporarily halted by hitting any key while the list is being displayed. This allows you to leisurely view data that might otherwise start scrolling off the screen. Simply hit any key to resume the display. This starting and stopping can be repeated as often as desired. When the display is completed, you must again hit a key to re-enter the command mode.

Option 3 produces a statistical analysis of your data. Various statistical parameters are calculated and displayed. The following is an explanation of some that may not be familiar to you.

Three measures of location, or central tendency, are provided. These are indicators of an "average" value. The *mean* is the sum of the values divided by the number of values. If the values are arranged in order from highest to lowest, the *median* is the middle value if the number of values is odd. If it is even, the median is the number halfway between the two middle values. The *midrange* is the number halfway between the largest and smallest values.

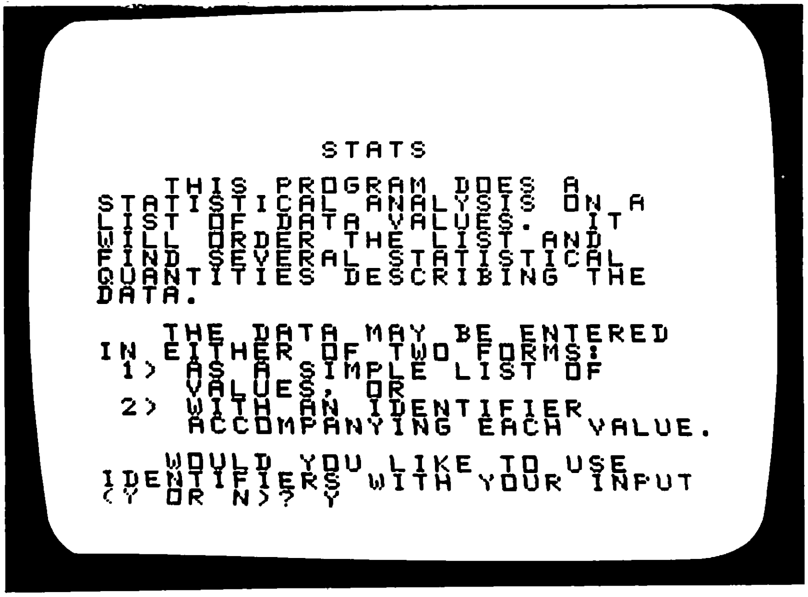
These measures of location give information about the average value of the data. However, they give no idea of how the data is dispersed or spread out around this "average." For that we need "measures of dispersion" or as they are sometimes called, "measures of variation." The simplest of these is the *range* which is just the difference between the highest and lowest data values. Two other closely related measures of dispersion are given: the *variance* and the *standard deviation*. The variance is defined as:

$$VA = \frac{\sum_{i=1}^{N} (V_i - M)^2}{N-1}$$

Here N is the number of values, V_i is the value i, and M is the mean value. The standard deviation is simply the square root of the variance. We do not have space to detail a lengthy discussion of their theoretical use. For this refer to the bibliography. Basically, however, the smaller the standard deviation, the more all the data tends to be clustered close to the mean value.

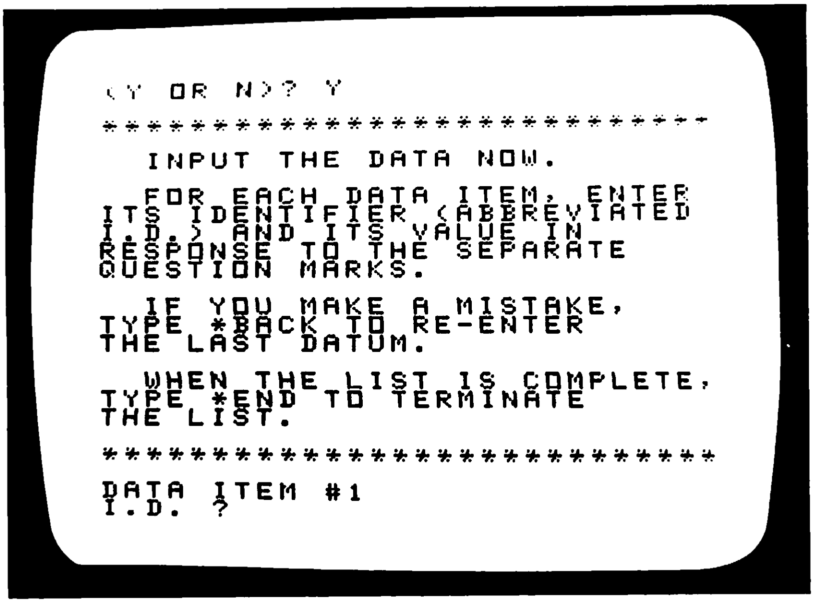
One word of warning—the first time option 2 or 3 is selected, the program must take some time to sort the data into numerical order. The time this requires depends upon how many items are on the list and how badly they are out of sequence. Average times are 30 seconds for 25 items, two minutes for 50 items, and about eight minutes for 100 items. The TI-99/4A will pause while this is occurring, so don't think it has hung up or fallen asleep! If you have several items on your list, this is the perfect chance to rob your refrigerator, make a quick phone call, or whatever.

SAMPLE RUN



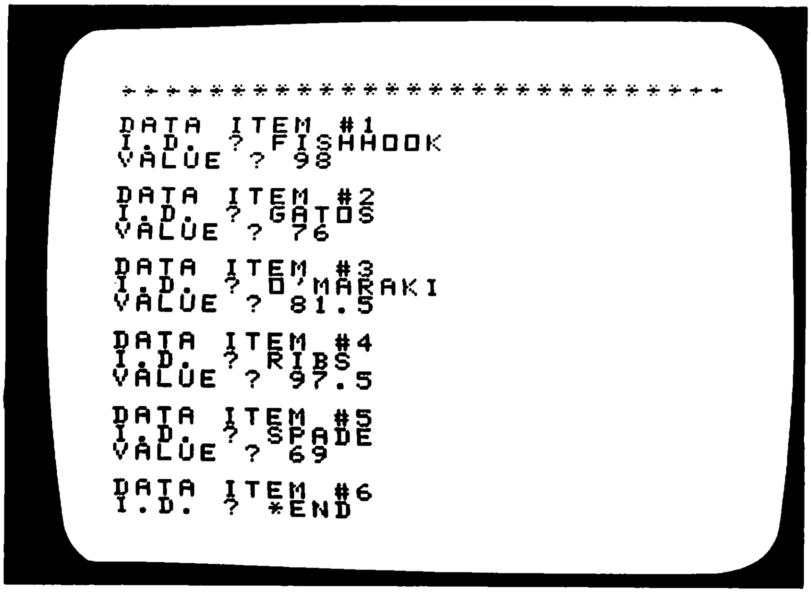
```
              STATS
   THIS PROGRAM DOES A
STATISTICAL ANALYSIS ON A
LIST OF DATA VALUES. IT
WILL ORDER THE LIST AND
FIND SEVERAL STATISTICAL
QUANTITIES DESCRIBING THE
DATA.
   THE DATA MAY BE ENTERED
IN EITHER OF TWO FORMS:
 1) AS A SIMPLE LIST OF
    VALUES, OR
 2) WITH AN IDENTIFIER
    ACCOMPANYING EACH VALUE.
   WOULD YOU LIKE TO USE
IDENTIFIERS WITH YOUR INPUT
(Y OR N)? Y
```

The program beings with an explanation and asks the operator if he or she would like to use identifiers. A yes response is given.



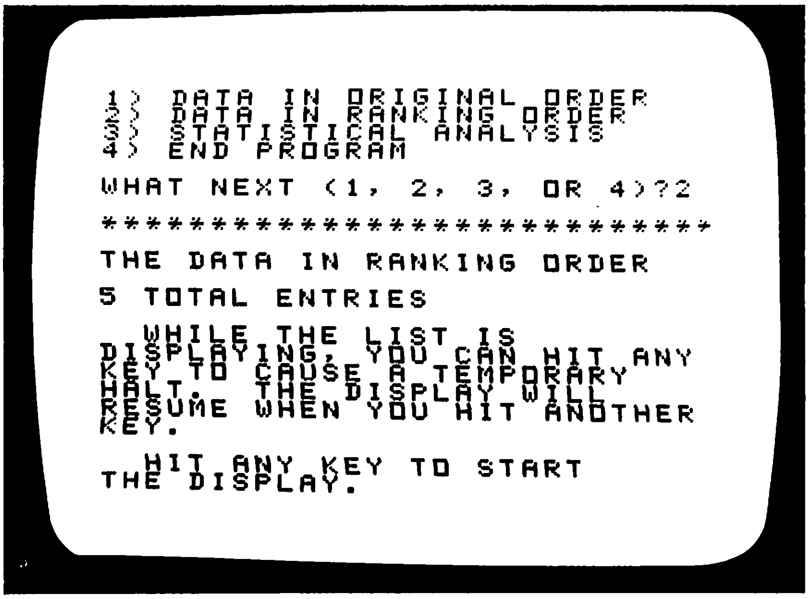
```
(Y OR N)? Y
****************************
  INPUT THE DATA NOW.
   FOR EACH DATA ITEM, ENTER
ITS IDENTIFIER (ABBREVIATED
I.D.) AND ITS VALUE IN
RESPONSE TO THE SEPARATE
QUESTION MARKS.
   IF YOU MAKE A MISTAKE,
TYPE *BACK TO RE-ENTER
THE LAST DATUM.
   WHEN THE LIST IS COMPLETE,
TYPE *END TO TERMINATE
THE LIST.
****************************
DATA ITEM #1
I.D. ?
```

The program explains how data is to be entered; it is ready to receive the operator's input.



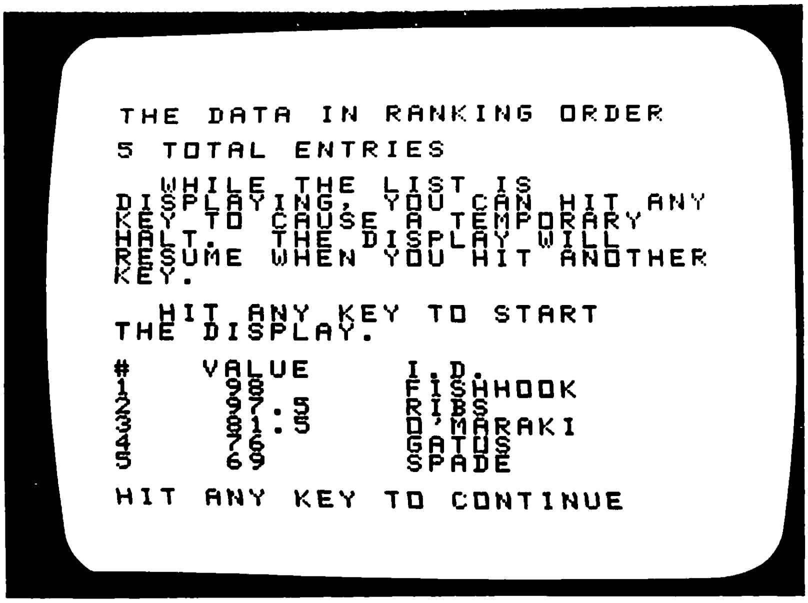
```
******************************
DATA ITEM #1
I.D. ? FISHHOOK
VALUE ? 98
DATA ITEM #2
I.D. ? GATOS
VALUE ? 76
DATA ITEM #3
I.D. ? O'MARAKI
VALUE ? 81.5
DATA ITEM #4
I.D. ? RIBS
VALUE ? 97.5
DATA ITEM #5
I.D. ? SPADE
VALUE ? 69
DATA ITEM #6
I.D. ? *END
```

The operator enters the names and scores of those who took a programming aptitude test. The actual test was given to many people, but for demonstration purposes, only five names are used here. The special string, * END, is used to signal the end of the data.



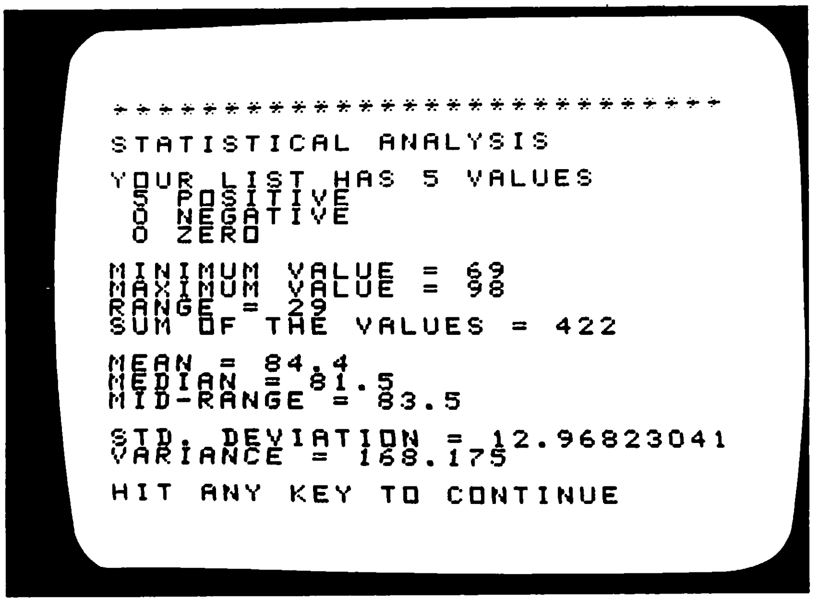
```
1) DATA IN ORIGINAL ORDER
2) DATA IN RANKING ORDER
3) STATISTICAL ANALYSIS
4) END PROGRAM
WHAT NEXT (1, 2, 3, OR 4)?2
******************************
THE DATA IN RANKING ORDER
5 TOTAL ENTRIES
   WHILE THE LIST IS
DISPLAYING, YOU CAN HIT ANY
KEY TO CAUSE A TEMPORARY
HALT. THE DISPLAY WILL
RESUME WHEN YOU HIT ANOTHER
KEY.
   HIT ANY KEY TO START
THE DISPLAY.
```

After being shown the list of continuation options, the operator requests that the list be sorted into numerical order. The program waits for a key to be pressed to continue the run.



```
THE DATA IN RANKING ORDER
5 TOTAL ENTRIES
   WHILE THE LIST IS
DISPLAYING, YOU CAN HIT ANY
KEY TO CAUSE A TEMPORARY
HALT.  THE DISPLAY WILL
RESUME WHEN YOU HIT ANOTHER
KEY.
   HIT ANY KEY TO START
THE DISPLAY.
#    VALUE        I.D.
1     98          FISHHOOK
2     97.5        RIBS
3     81.5        O'MARAKI
4     76          GATOS
5     69          SPADE
HIT ANY KEY TO CONTINUE
```

The operator hits a key to start the display and is shown the data list in ranking order. The program waits for a key to be pressed to continue.



```
****************************
STATISTICAL ANALYSIS
YOUR LIST HAS 5 VALUES
 5 POSITIVE
 0 NEGATIVE
 0 ZERO
MINIMUM VALUE = 69
MAXIMUM VALUE = 98
RANGE = 29
SUM OF THE VALUES = 422
MEAN = 84.4
MEDIAN = 81.5
MID-RANGE = 83.5
STD. DEVIATION = 12.96823041
VARIANCE = 168.175
HIT ANY KEY TO CONTINUE
```

Later in the run, the operator selects continuation option 3. This calculates and displays the various statistical quantities.

PROGRAM LISTING

```
100 REM STATS
110 REM (C) 1984 DILITHIUM PRESS
120 B$="*BACK"
130 E$="*END"
140 MX=100
150 DIM D$(100),V(100),Z(100)
160 Z(0)=0
170 N$=CHR$(32)
180 CALL CLEAR
190 PRINT TAB(11);"STATS"
200 PRINT
210 PRINT "   THIS PROGRAM DOES A"
220 PRINT "STATISTICAL ANALYSIS ON A"
230 PRINT "LIST OF DATA VALUES.  IT"
240 PRINT "WILL ORDER THE LIST AND"
250 PRINT "FIND SEVERAL STATISTICAL"
260 PRINT "QUANTITIES DESCRIBING THE"
270 PRINT "DATA."
280 PRINT
290 PRINT "   THE DATA MAY BE ENTERED"
300 PRINT "IN EITHER OF TWO FORMS:"
310 PRINT " 1) AS A SIMPLE LIST OF"
320 PRINT TAB(5);"VALUES, OR"
330 PRINT " 2) WITH AN IDENTIFIER"
340 PRINT TAB(5);"ACCOMPANYING EACH VALUE."
350 PRINT
360 PRINT "   WOULD YOU LIKE TO USE"
370 PRINT "IDENTIFIERS WITH YOUR INPUT"
380 INPUT "(Y OR N)? ":R$
390 R$=SEG$(R$,1,1)
400 IF R$="Y" THEN 430
410 IF R$="N" THEN 450
420 GOTO 350
430 F=1
440 GOTO 460
450 F=0
460 GOSUB 2110
470 PRINT
480 PRINT TAB(3);"INPUT THE DATA NOW."
```

```
490 PRINT
500 IF F=1 THEN 560
510 PRINT TAB(3);"ENTER EACH VALUE"
520 PRINT "SEPARATELY IN RESPONSE TO"
530 PRINT "THE QUESTION MARK."
540 GOSUB 2020
550 GOTO 620
560 PRINT TAB(3);"FOR EACH DATA ITEM, ENTER"
570 PRINT "ITS IDENTIFIER (ABBREVIATED"
580 PRINT "I.D.) AND ITS VALUE IN"
590 PRINT "RESPONSE TO THE SEPARATE"
600 PRINT "QUESTION MARKS."
610 GOSUB 2020
620 GOSUB 2110
630 N=1
640 IF N>=1 THEN 660
650 N=1
660 PRINT
670 PRINT "DATA ITEM #";STR$(N)
680 IF F=1 THEN 710
690 D$(N)=N$
700 GOTO 770
710 INPUT "I.D. ? ":R$
720 IF R$=E$ THEN 940
730 IF R$<>B$ THEN 760
740 N=N-1
750 GOTO 640
760 D$(N)=R$
770 INPUT "VALUE ? ":R$
780 IF R$=E$ THEN 940
790 IF R$<>B$ THEN 820
800 IF F=0 THEN 820
810 GOTO 660
820 IF R$<>B$ THEN 850
830 N=N-1
840 GOTO 640
850 V(N)=VAL(R$)
860 IF N<MX THEN 920
870 PRINT
880 PRINT "NO MORE DATA ALLOWED !!"
890 CALL SOUND(300,200,3)
```

```
900 N=N+1
910 GOTO 940
920 N=N+1
930 GOTO 640
940 N=N-1
950 IF N>0 THEN 1000
960 PRINT
970 PRINT "NO DATA - RUN ABORTED"
980 CALL SOUND(300,200,3)
990 END
1000 GOSUB 2110
1010 PRINT
1020 PRINT "-- CONTINUATION OPTIONS --"
1030 PRINT
1040 PRINT "1) DATA IN ORIGINAL ORDER"
1050 PRINT "2) DATA IN RANKING ORDER"
1060 PRINT "3) STATISTICAL ANALYSIS"
1070 PRINT "4) END PROGRAM"
1080 PRINT
1090 INPUT "WHAT NEXT (1, 2, 3, OR 4)?":R
1100 R=INT(R)
1110 IF R<1 THEN 1010
1120 IF R>4 THEN 1010
1130 ON R GOSUB 1170,1350,1540,1150
1140 GOTO 1000
1150 CALL CLEAR
1160 END
1170 GOSUB 2110
1180 PRINT
1190 PRINT "THE ORIGINAL DATA ORDER "
1200 PRINT
1210 PRINT STR$(N);N$;"TOTAL ENTRIES"
1220 GOSUB 2150
1230 PRINT
1240 PRINT "#";TAB(5);"VALUE";
1250 IF F=1 THEN 1280
1260 PRINT
1270 GOTO 1290
1280 PRINT TAB(14);"I.D."
1290 FOR J=1 TO N
1300 PRINT STR$(J);TAB(5);V(J);TAB(14);SEG$(D$
(J),1,12)
1310 GOSUB 2280
1320 NEXT J
1330 GOSUB 2510
1340 RETURN
1350 GOSUB 2110
1360 PRINT
1370 PRINT "THE DATA IN RANKING ORDER"
1380 PRINT
1390 PRINT STR$(N);" TOTAL ENTRIES"
1400 GOSUB 2340
1410 GOSUB 2150
1420 PRINT
1430 PRINT "#";TAB(5);"VALUE";
1440 IF F=1 THEN 1470
1450 PRINT
1460 GOTO 1480
1470 PRINT TAB(14);"I.D."
1480 FOR J=1 TO N
1490 PRINT STR$(J);TAB(5);V(Z(J));TAB(14);SEG$
(D$(Z(J)),1,12)
1500 GOSUB 2280
1510 NEXT J
1520 GOSUB 2510
1530 RETURN
1540 GOSUB 2110
1550 PRINT
1560 PRINT "STATISTICAL ANALYSIS"
1570 PRINT
1580 PRINT "YOUR LIST HAS";N;"VALUES"
1590 NP=0
1600 NN=0
1610 NZ=0
1620 SQ=0
1630 W=0
1640 FOR J=1 TO N
1650 W=W+V(J)
1660 SQ=SQ+V(J)*V(J)
1670 IF V(J)>0 THEN 1710
1680 IF V(J)<0 THEN 1730
1690 NZ=NZ+1
```

```
1700 GOTO 1740
1710 NP=NP+1
1720 GOTO 1740
1730 NN=NN+1
1740 NEXT J
1750 M=W/N
1760 VA=0
1770 IF N=1 THEN 1790
1780 VA=(SQ-N*M*M)/(N-1)
1790 SD=SQR(VA)
1800 PRINT NP;"POSITIVE"
1810 PRINT NN;"NEGATIVE"
1820 PRINT NZ;"ZERO"
1830 PRINT
1840 GOSUB 2340
1850 PRINT "MINIMUM VALUE =";V(Z(N))
1860 PRINT "MAXIMUM VALUE =";V(Z(1))
1870 PRINT "RANGE =";V(Z(1))-V(Z(N))
1880 PRINT "SUM OF THE VALUES =";W
1890 PRINT
1900 PRINT "MEAN =";M
1910 Q=INT(N/2)+1
1920 MD=V(Z(Q))
1930 IF (N/2)>INT(N/2)THEN 1950
1940 MD=(V(Z(Q))+V(Z(Q-1)))/2
1950 PRINT "MEDIAN =";MD
1960 PRINT "MID-RANGE =";(V(Z(1))+V(Z(N)))/2
1970 PRINT
1980 PRINT "STD. DEVIATION =";SD
1990 PRINT "VARIANCE =";VA
2000 GOSUB 2510
2010 RETURN
2020 PRINT
2030 PRINT TAB(3);"IF YOU MAKE A MISTAKE "
2040 PRINT "TYPE";N$;B$;N$;"TO RE-ENTER"
2050 PRINT "THE LAST DATUM."
2060 PRINT
2070 PRINT TAB(3);"WHEN THE LIST IS COMPLETE,"
2080 PRINT "TYPE";N$;E$;N$;"TO TERMINATE"
2090 PRINT "THE LIST."
2100 RETURN
```

```
2110 PRINT
2120 CALL HCHAR(24,3,42,28)
2130 PRINT
2140 RETURN
2150 PRINT
2160 PRINT TAB(3);"WHILE THE LIST IS"
2170 PRINT "DISPLAYING, YOU CAN HIT ANY"
2180 PRINT "KEY TO CAUSE A TEMPORARY"
2190 PRINT "HALT.  THE DISPLAY WILL"
2200 PRINT "RESUME WHEN YOU HIT ANOTHER"
2210 PRINT "KEY."
2220 PRINT
2230 PRINT TAB(3);"HIT ANY KEY TO START"
2240 PRINT "THE DISPLAY."
2250 CALL KEY(0,KK,SS)
2260 IF SS=0 THEN 2250
2270 RETURN
2280 CALL KEY(0,KK,SS)
2290 IF SS<>0 THEN 2310
2300 RETURN
2310 CALL KEY(0,KK,SS)
2320 IF SS=0 THEN 2310
2330 RETURN
2340 IF Z(0)=1 THEN 2500
2350 FOR J=1 TO N
2360 Z(J)=J
2370 NEXT J
2380 IF N=1 THEN 2500
2390 NM=N-1
2400 FOR K=1 TO N
2410 FOR J=1 TO NM
2420 N1=Z(J)
2430 N2=Z(J+1)
2440 IF V(N1)>V(N2)THEN 2470
2450 Z(J+1)=N1
2460 Z(J)=N2
2470 NEXT J
2480 NEXT K
2490 Z(0)=1
2500 RETURN
2510 PRINT
```

```
2520 PRINT "HIT ANY KEY TO CONTINUE"
2530 CALL KEY(0,KK,SS)
2540 IF SS=0 THEN 2530
2550 RETURN
```

EASY CHANGES

1. The program arrays are currently dimensioned to allow a maximum of 100 data items. The total storage required for the program depends on the maximum dimension parameter, MX, whether or not identifiers are being used, and if so, on the length of a typical identifier. Should your application require more than 100 data values, you will have to increase the value of MX in line 140 and the array arguments in line 150 accordingly. To accommodate up to 300 data items, make these changes:

   ```
   140 MX=300
   150 DIM DX(300),V(300),Z(300)
   ```

   Of course, you will have to have enough RAM memory to enable this.
2. Because of possible conflicts with identifiers in your list, you may wish to change the special strings that signal termination of data input and/or the backing up of data input. These are controlled by the variables E$ and B$, respectively. They are set in lines 120 and 130. If you wish to terminate the data with /DONE/ and to back up with /LAST/, for example, change these lines as follows:

   ```
   120 B$="/LAST/"
   130 E$="/DONE/"
   ```

3. You may wish to see your lists sorted from smallest value to largest value instead of the other way around, as done now. This can be accomplished by changing the "greater than" sign (>) in line 2440 to a "less than" sign (<). Thus:

   ```
   2440 IF V(N1)<V(N2) THEN 2470
   ```

   This will, however, cause a few funny things to happen to the statistics. The real minimum value will be displayed under the heading "maximum" and vice versa. Also, the range will have its correct magnitude but with an erroneous minus sign in front. To cure these afflictions, make these changes also:

```
1850 PRINT "MINIMUM VALUE=";V(Z(1))
1860 PRINT "MAXIMUM VALUE=";V(Z(N))
1870 PRINT "RANGE=";V(Z(N))-V(Z(1))
```

MAIN ROUTINES

120– 170	Initializes constants and dimensioning.
180– 450	Displays messages, determines if identifiers will be used.
460– 940	Gets data from the user.
950– 990	Checks that input contains at least one value.
1000–1140	Command mode—gets user's next option and does a GOSUB to it.
1150–1160	Subroutine to end program.
1170–1340	Subroutine to list data in the original order.
1350–1530	Subroutine to list data in ranking order.
1540–2010	Subroutine to calculate and display statistics.
2020–2270	Subroutines to display various messages.
2280–2330	Subroutine to allow user to temporarily start and stop display listing.
2340–2500	Subroutine to sort the list in ranking order.
2510–2550	Subroutine to detect if user has hit a key to continue.

MAIN VARIABLES

MX	Maximum number of data values allowed.
D$(MX)	String array of identifiers.
V(MX)	Array of the data values.
Z(MX)	Array of the sorting order.
N	Number of data values in current application.
F	Flag on identifier usage (1=yes, 0=no).
B$	Flag string to back up the input.
E$	Flag string to signal end of the input.
N$	String of one blank character.
R$	User input string.
NM	N−1.
R	Continuation option.
NP	Number of positive values.
NN	Number of negative values.
NZ	Number of zero values.
W	Sum of the values.
SQ	Sum of the squares of the values.

M	Mean value.
MD	Median of the values.
VA	Variance.
SD	Standard deviation.
J, K	Loop indices.
N1, N2	Possible data locations to interchange during sorting.
Q	Work variable.
KK, SS	User input.

SUGGESTED PROJECTS

1. The sorting algorithm used in the program is efficient only when the number of list items is fairly small—less than 25 or so. This is because it does not do checking along the way to see when the list becomes fully sorted. If your lists tend to be longer than 25 items, you might wish to use another sorting algorithm more appropriate for longer lists. Try researching other sorts and incorporating them into the program. To get you started, try these changes:

   ```
   2400 Q=0
   2440 IF V(N1)> =V(N2) THEN 2470
   2465 Q=1
   2480 IF Q=1 THEN 2400
   ```

   If your lists are short, this routine will probably be a little slower than the current one. However, for longer lists it will save proportionately more and more time.
2. Because the INPUT statement is used when entering identifiers, commas cannot be used inside identifier names. This can be circumvented if you use quotes around the identifier name, but you may forget to do this. By modifying the input routine to use a series of **CALL KEY** commands, you can build up the identifier strings piecemeal and allow embedded commas. Modify the appropriate routine to do this.
3. Many other statistical parameters exist to describe this kind of data. Research them and add some that might be useful to you. One such idea is classifying the data. This consists of dividing the range into a number of equal classes and then counting how many values fall into each class.

Section 6

Miscellaneous Programs

INTRODUCTION TO MISCELLANEOUS PROGRAMS

These programs show how simple programs can do interesting things. All of them have a mathematical flavor. They are short and, as such, would be useful for study for those just learning BASIC in particular or programming in general.

Monte Carlo simulation involves programming the computer to conduct an experiment. (It doesn't involve high-stakes gambling!) PI shows how this technique can be used to calculate an approximation to the famous mathematical constant pi.

PYTHAG will find all right triangles with integral side lengths. A clever algorithm is utilized to do this.

Have you ever looked around your classroom or club meeting and wondered if any two people had the same birthdate? BIRTHDAY will show you what the surprising odds are.

Very high precision arithmetic can be done on the TI-99/4A with the proper "know-how." POWERS will calculate the values of integers raised to various powers, not to the TI-99/4A's "normal" ten-digit precision, but up to 250 full digits of precision.

BIRTHDAY

PURPOSE

Suppose you are in a room full of people. What is the probability that two or more of these people have the same birthday? How many people have to be in the room before the probability becomes greater than 50 percent? We are talking only about the month and day of birth, not the year.

This is a fairly simple problem to solve, even without a computer. With a computer to help with the calculations, it becomes very easy. What makes the problem interesting is that the correct answer is nowhere near what most people immediately guess. Before reading further, what do you think? How many people have to be in a room before there is better than a 50-50 chance of birthday duplication? 50? 100? 200?

HOW TO USE IT

When you RUN the program, it starts by displaying headings over two columns of numbers that will be shown. The left column is the number of people in the room, starting with one. The right column is the probability of birthday duplication.

For one person, of course, the probability is zero, since there is no one else with a possible duplicate birthday. For two people, the probability is simply the decimal equivalent of 1/365 (note that we assume a 365-day year, and an equal likelihood that each person could have been born on any day of the year).

What is the probability of duplication when there are three people in the room? No, not just 2/365. It's actually

1 – (364/365 times 363/365)

This is simply one minus the probability of *no* duplicate birthdays.

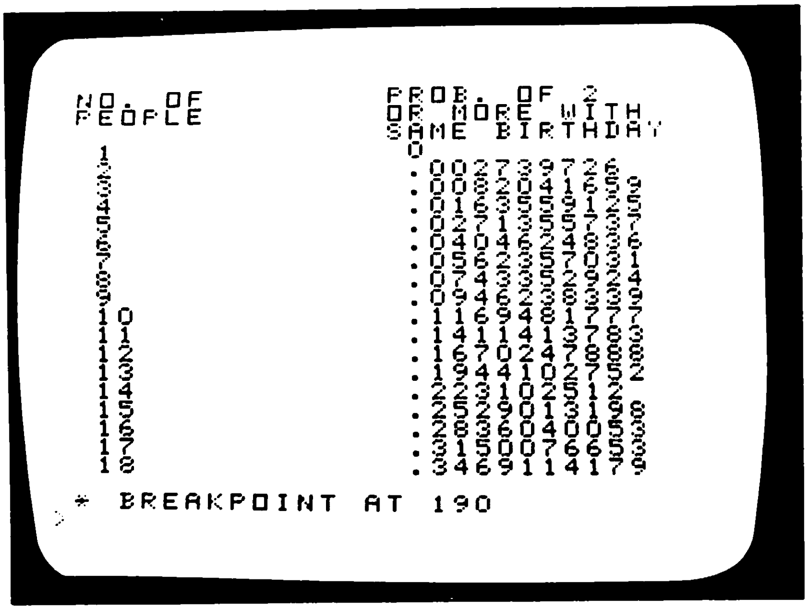
The probability for four people is

1 – (364/365 times 363/365 times 362/365)

The calculation continues like this, adding a new term for each additional person in the room. You will find that the result (probability of duplication) exceeds .50 surprisingly fast.

The program continues with the calculation until there are 60 people in the room. You will have to stop the program long before that to see the point where the probability first exceeds 50 percent.

SAMPLE RUN



```
NO. OF            PROB. OF 2
PEOPLE            OR MORE WITH
                  SAME BIRTHDAY
 1                 0
 2                 .002739726
 3                 .0082041659
 4                 .0163559125
 5                 .0271355737
 6                 .0404624836
 7                 .0562357031
 8                 .0743352924
 9                 .0946238339
 10                .1169481777
 11                .1411413783
 12                .1670247888
 13                .1944102752
 14                .223102512
 15                .2529013198
 16                .2836040053
 17                .3150076653
 18                .3469114179
* BREAKPOINT AT 190
>
```

After the probability of 18 people with the same birthday is shown, the **FCTN-CLEAR** key is pressed to terminate the run.

PROGRAM LISTING

```
100 REM BIRTHDAY
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 PRINT "NO. OF","PROB. OF 2"
140 PRINT "PEOPLE","OR MORE WITH"
150 PRINT ,"SAME BIRTHDAY"
160 Q=1
170 FOR N=1 TO 60
180 PRINT N,1-Q
190 Q=Q*(365-N)/365
200 NEXT N
210 END
```

EASY CHANGE

1. Change the constant value of 60 in line 170 to alter the range of the number of people in the calculation. For example, change it to 100 and watch how fast the probability approaches 1. Or, change it to 20 to end the program before the output rolls off the screen.

MAIN ROUTINES

120–150	Displays headings.
160	Initializes Q to 1.
170–200	Calculates probability of no duplication, then displays probability of duplication.

MAIN VARIABLES

N	Number of people in the room.
Q	Probability of no duplication of birthdays.

SUGGESTED PROJECTS

Modify the program to allow for leap years in the calculation, instead of assuming 365 days per year.

PI

PURPOSE AND DISCUSSION

The Greek letter pi, π, represents probably the most famous constant in mathematical history. It occurs regularly in many different areas of mathematics. It is best known as the constant appearing in several geometric relationships involving the circle. The circumference of a circle of radius r is $2\pi r$, while the area enclosed by the circle is πr^2.

Being a transcendental number, pi cannot be expressed exactly by any number of decimal digits. To nine significant digits, its value is 3.14159265. Over many centuries, man has devised many different methods to calculate pi.

This program uses a valuable, modern technique known as computer simulation. The name "simulation" is rather self-explanatory; the computer performs an experiment for us. This is often desirable for many different reasons. The experiment may be cheaper, less dangerous, or more accurate to run on a computer. It may even be impossible to do in "real life." Usually, however, the reason is that the speed of the computer allows the simulation to be performed many times faster than actually conducting the real experiment.

This program simulates the results of throwing darts at a specially constructed dartboard. Consider Figure 1 which shows the peculiar square dartboard involved. The curved arc, outlining the shaded area, is that of a circle with the center in the lower left hand corner. The sides of the square, and thus the radius of the circle, are considered to have a length of 1.

Suppose we were able to throw darts at this square target in such a way that each dart had an equal chance of landing anywhere within the square. A certain percentage of darts would result in "hits," i.e., land in the shaded area. The expected value of this percentage is simply the area of the shaded part divided by the area of the entire square.

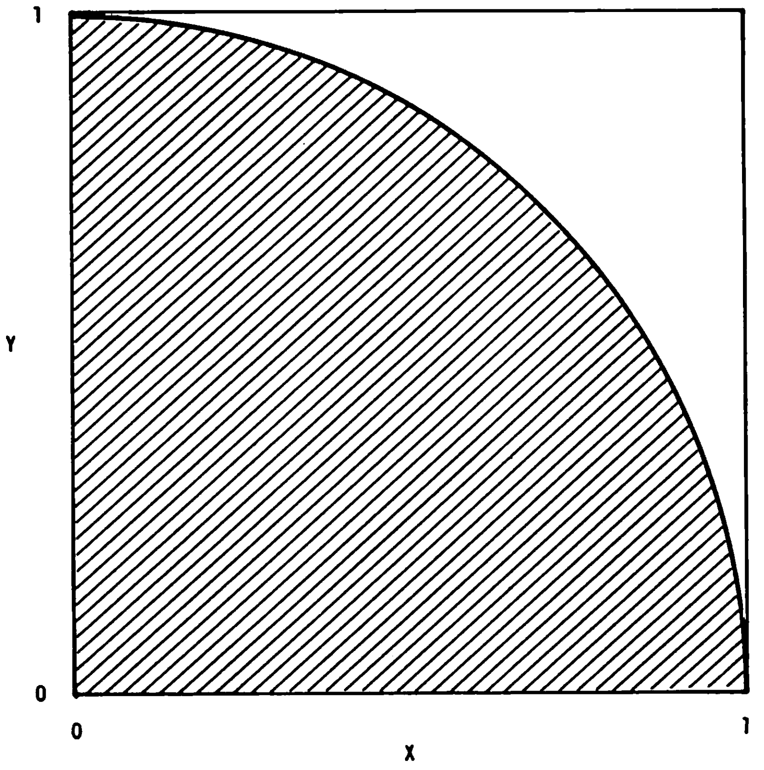


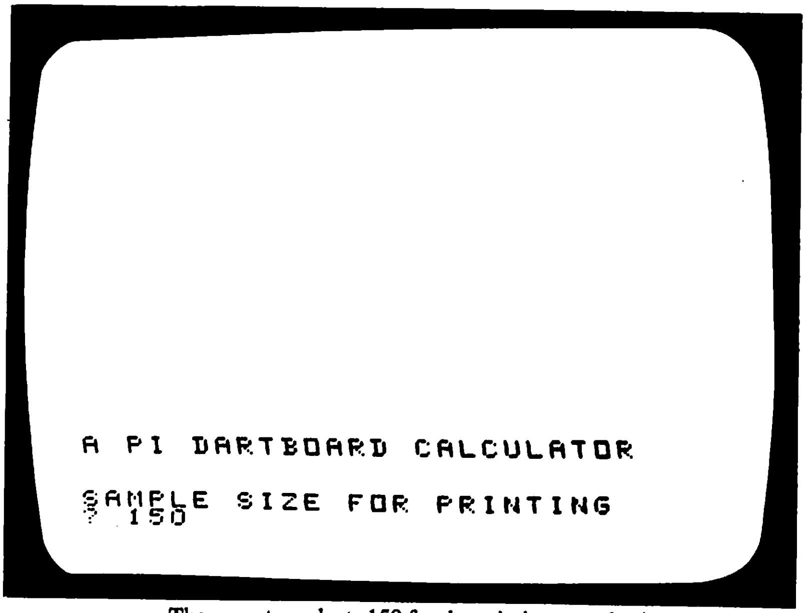
Figure 1. The PI Dartboard

The area of the shaded part is one-fourth of the area that the entire circle would enclose if the arc were continued to completely form the circle. Recall that the area of a circle is πr^2 where r is the radius. In our case, $r=1$, and the area of the entire circle would simply be π. The shaded area of the dartboard is one-fourth of this entire circle and thus has an area of $\pi/4$. The area of the square is s^2, where s is the length of one side. On our dartboard, $s=1$, and the area of the whole dartboard is 1.

Now, the expected ratio of "hits" to darts thrown can be expressed as

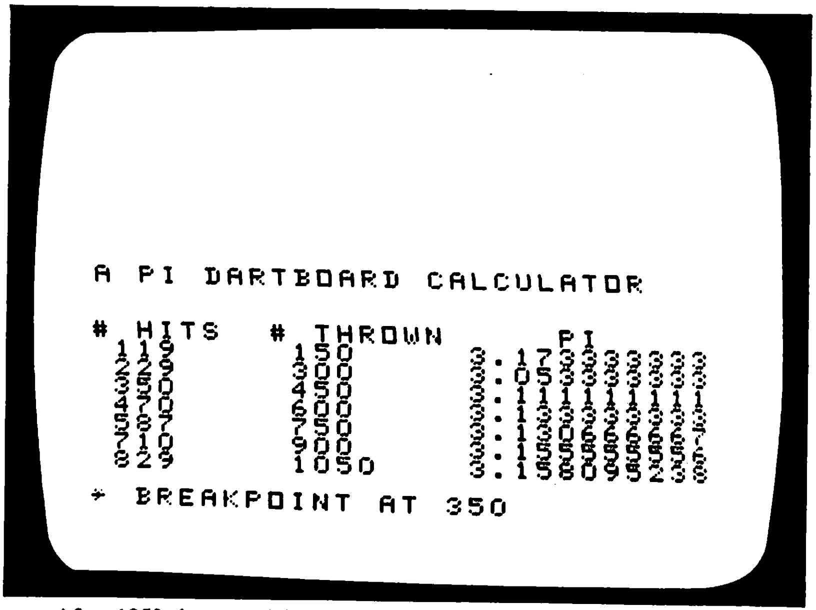
$$\text{RATIO} = \frac{\#\text{ hits}}{\#\text{ thrown}} = \frac{\text{shaded area}}{\text{entire area}} = \frac{\pi/4}{1} = \frac{\pi}{4}$$

SAMPLE RUN



```
A PI DARTBOARD CALCULATOR

SAMPLE SIZE FOR PRINTING
? 150
```

The operator selects 150 for the printing sample size.



```
A PI DARTBOARD CALCULATOR

# HITS     # THROWN          PI
  119        150       3.173333333
  229        300       3.053333333
  350        450       3.111111111
  470        600       3.133333333
  587        750       3.130666667
  710        900       3.155555556
  829        1050      3.158095238
* BREAKPOINT AT 350
```

After 1050 darts are "thrown," the user presses the **FCTN-CLEAR** key to terminate the run.

So we now have an experimental way to approximate the value of π. We perform the experiment and compute the ratio of "hits" observed. We then multiply this number by four and we have calculated π experimentally.

But instead of actually constructing the required dartboard and throwing real darts, we will let the TI-99/4A do the job. The program "throws" each dart by selecting a separate random number between 0 and 1 for the X and Y coordinates of each dart. This is accomplished by using the built-in RND function of BASIC. A "dart" is in the shaded area if $X^2+Y^2<1$ for it.

So the program grinds away, continually throwing darts and determining the ratio of "hits." This ratio is multiplied by four to arrive at an empirical approximation to π.

HOW TO USE IT

The program requires only one input from you. This is the "sample size for printing," i.e., how many darts it should throw before printing its current results. Any value of one or higher is acceptable.

After you input this number, the program will commence the simulation and display its results. A cumulative total of "hits," darts thrown, and the current approximation to π will be displayed for each multiple of the sample size.

This will continue until you press FCTN-CLEAR. When you are satisfied with the total number of darts thrown, press FCTN-CLEAR to terminate the program execution.

PROGRAM LISTING

```
100 REM PI
110 REM (C) 1984 DILITHIUM PRESS
120 T=0
130 TH=0
140 RANDOMIZE
150 CALL CLEAR
160 GOSUB 380
170 PRINT "SAMPLE SIZE FOR PRINTING"
180 INPUT NP
190 NP=INT(NP)
200 IF NP<1 THEN 150
210 CALL CLEAR
220 GOSUB 380
```

```
230 PRINT "# HITS";TAB(9);"# THROWN";TAB(22);"
PI"
240 GOSUB 300
250 TH=TH+NH
260 T=T+NP
270 P=4*TH/T
280 PRINT TH;TAB(9);T;TAB(17);P
290 GOTO 240
300 NH=0
310 FOR J=1 TO NP
320 X=RND
330 Y=RND
340 IF (X*X+Y*Y)>1 THEN 360
350 NH=NH+1
360 NEXT J
370 RETURN
380 PRINT "A PI DARTBOARD CALCULATOR"
390 PRINT
400 PRINT
410 RETURN
```

EASY CHANGES

1. If you want the program to always use a fixed sample size, change lines 170 and 180 to read

```
170 NP=150
180 REM
```

Of course, the value of 150 used here may be changed to whatever you wish. With this change, lines 190, 200, 210, and 220 are not needed and may be deleted.

2. If you want the program to stop by itself after a certain number of darts have been thrown, add the following lines:

```
202 PRINT
204 PRINT "TOTAL # DARTS TO THROW":
206 INPUT ND
282 IF T<ND THEN 290
284 END
```

This will ask the operator how many total darts should be thrown, and then terminate the program when they have been thrown.

MAIN ROUTINES

120–140	Initializes constants.
150–230	Gets operator input, displays column headings.
240–290	Calculates and displays results.
300–370	Throws NP darts and records number of "hits."
380–410	Displays program title.

MAIN VARIABLES

T	Total darts thrown.
TH	Total "hits."
NP	Sample size for printing.
NH	Number of hits in one group of NP darts.
P	Calculated value of pi.
X, Y	Random-valued coordinates of a dart.
J	Loop index.

SUGGESTED PROJECTS

1. Calculate the percentage error in the program's calculation of pi and display it with the other results. You will need to define a variable, say PY, which is set to the value of pi. Then the percentage error, PE, can be calculated as:

   PE=100*ABS(P−PY)/PY

2. The accuracy of this simulation is highly dependent on the quality of the computer's random number generator. Try researching different algorithms for pseudo random number generation. Then try incorporating them into the program. Change lines 320 and 330 to use the new algorithm(s). This can actually be used as a test of the various random number generators. Gruenberger's book, referenced in the bibliography, contains good material on various pseudo random number generators.

POWERS

PURPOSE

By now you have probably learned that the TI-99/4A displays ten significant digits when dealing with numbers. For integers less than ten billion (10,000,000,000), the TI-99/4A can display the precise value of the number. But for larger integers the TI-99/4A only displays the most significant (leftmost) ten digits, plus the exponent. This means, of course, that there is no way you can use the TI-99/4A to deal with precise integers greater than ten billion, right?

Wrong.

This program calculates either factorials or successive powers of an integer, and can display precise results that are up to 250 digits long. By using a "multiple-precision arithmetic" technique, this program can tell you *exactly* what 973 to the 47th power is, for example.

HOW TO USE IT

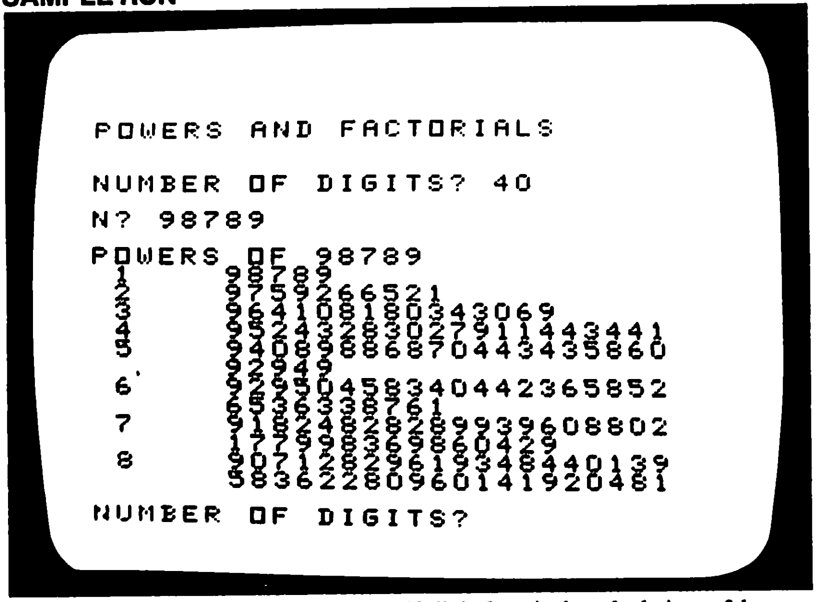
The program first asks you how many digits long you want the largest number to be. This can be any integer from 1 to 250. So, for example, if you enter 40, you will get answers up to 40 digits long.

Next you are asked for the value of N. If you respond with a value of one, you are requesting to be shown all the factorials that will fit in the number of digits you specified. First you will get one factorial, then two factorial, and so on. In case you have forgotten, three factorial is three times two times one, or six. Four factorial is four times three times two times one, or twenty-four.

If you enter an N in the range from 2 through 100,000, you are requesting the successive powers of that number up to the limit of digits you specified. So, if you provide an N of 23, you will get 23 to the first power, then 23 squared, then 23 cubed, and so on.

Finally, after it has displayed the largest number that will fit within the number of digits you entered, the program starts over. The larger the number of digits you ask for, the longer it will take the program to calculate each number. If you ask for zero digits, the program ends.

SAMPLE RUN



```
POWERS AND FACTORIALS

NUMBER OF DIGITS? 40
N? 98789
POWERS OF 98789
 1      98789
 2      9759266521
 3      964108180343069
 4      95243283027911443441
 5      940899886870443435860
        92949
 6      92950458340442365852
        6536338761
 7      91824828289939608802
        177998369860429
 8      90712829619348440139
        58362280960141920481
NUMBER OF DIGITS?
```

The operator wants answers up to 40 digits long in the calculations of the powers of 98789. The program calculates numbers up to 98789^8 and then asks for the number of digits again (in preparation for the next calculation the operator requests).

PROGRAM LISTING

```
100 REM POWERS
110 REM (C) 1984 DILITHIUM PRESS
120 CALL CLEAR
130 PRINT "POWERS AND FACTORIALS"
140 PRINT
150 PRINT
160 DIM N(255)
170 INPUT "NUMBER OF DIGITS? ":M
180 IF M<>0 THEN 210
```

```
190 CALL CLEAR
200 END
210 M=INT(M)
220 IF M>250 THEN 170
230 IF M<1 THEN 170
240 PRINT
250 INPUT "N? ":NN
260 NN=INT(NN)
270 IF NN<1 THEN 240
280 IF NN>100000 THEN 240
290 PRINT
300 F=0
310 IF NN<>1 THEN 350
320 F=1
330 PRINT "FACTORIALS"
340 GOTO 360
350 PRINT "POWERS OF";NN
360 T=10
370 K=1
380 N(0)=NN
390 FOR J=0 TO M
400 IF N(J)<T THEN 450
410 Q=INT(N(J)/T)
420 W=N(J)-Q*T
430 N(J)=W
440 N(J+1)=N(J+1)+Q
450 NEXT J
460 J=M+1
470 IF N(J)<>0 THEN 500
480 J=J-1
490 GOTO 470
500 IF J>=M THEN 690
510 D=0
520 PRINT K;TAB(7);
530 N$=STR$(N(J))
540 D=D+1
550 IF D<=20 THEN 590
560 D=1
570 PRINT
580 PRINT TAB(7);
590 PRINT N$;
```

```
600 J=J-1
610 IF J>=0 THEN 530
620 NN=NN+F
630 K=K+1
640 PRINT
650 FOR J=0 TO M
660 N(J)=N(J)*NN
670 NEXT J
680 GOTO 390
690 FOR J=1 TO 255
700 N(J)=0
710 NEXT J
720 M=0
730 NN=0
740 PRINT
750 GOTO 170
```

EASY CHANGES

1. To change the program so that it always uses, say, 50-digit numbers, remove lines 170 through 200, and insert this line:

   170 M=50

2. To clear the screen before the output begins being displayed, change line 290 to say:

   290 CALL CLEAR

3. If 250 digits isn't enough for you, you can go higher. For 500 digits, make these changes:

   a. In line 160, change the 255 to 505.
   b. In line 220, change the 250 to 500.
   c. In line 690, change the 255 to 505.

MAIN ROUTINES

120–160	Displays title. Sets up array for calculations.
170–350	Asks for number of digits and N. Checks validity of responses. Displays heading.
360–380	Initializes variables for calculations.
390–450	Performs "carrying" in N array so each element has a value no larger than nine.

460–490	Scans backwards through N array for first non-zero element.
500	Checks to see if this value would be larger than the number of digits requested.
510–610	Displays counter and number. Goes to second line if necessary.
620–640	Prepares to multiply by NN to get next number.
650–680	Multiplies each digit in N array by NN. Goes back to line 390.
690–750	Zeroes out N array in preparation for next request. Goes back to 170.

MAIN VARIABLES

N	Array in which calculations are made.
M	Number of digits of precision requested by operator.
NN	Starting value. If 1, factorials. If greater than 1, powers of NN.
F	Set to 0 if powers, 1 if factorials.
T	Constant value of 10.
K	Counter of current power or factorial.
J	Subscript variable.
Q, W	Temporary variables used in reducing each integer position in the N array to a value from 0 to 9.
D	Number of digits displayed so far on the current line (maximum is 20).
N$	String variable used to convert each digit into displayable format.

SUGGESTED PROJECTS

1. Determine the largest NN that could be used without errors entering into the calculation (because of intermediate results exceeding ten billion), then modify line 280 to permit values that large to be entered.
2. Create a series of subroutines that can add, subtract, multiply, divide, and exchange numbers in two arrays, using a technique like the one used here. Then you can perform high-precision calculations by means of a series of GOSUB statements.

PYTHAG

PURPOSE

Remember the Pythagorean Theorem? It says that the sum of the squares of the two legs of a right triangle is equal to the square of the hypotenuse. Expressed as a formula, it is $a^2+b^2=c^2$. The most commonly remembered example of this is the 3-4-5 right triangle ($3^2+4^2=5^2$). Of course, there are an infinite number of other right triangles.

This program displays integer values of a, b, and c that result in right triangles.

HOW TO USE IT

To use this program, all you need to do is RUN it and watch the "Pythagorean triplets" (sets of values for a, b, and c) come out. The program displays 18 sets of values on each screen, and then waits for you to press any key (except **E**) before it continues with the next 18. It will go on indefinitely until you press the **E** key (for "end").

The left-hand column shows the count of the number of sets of triplets produced, and the other three columns are the values of a, b, and c.

The sequence in which the triplets are produced is not too obvious, so we will explain how the numbers are generated.

It has been shown that the following technique will generate all *primitive* Pythagorean triplets. ("Primitive" means that no set is an exact multiple of another.) If you have two positive integers called R and S such that:

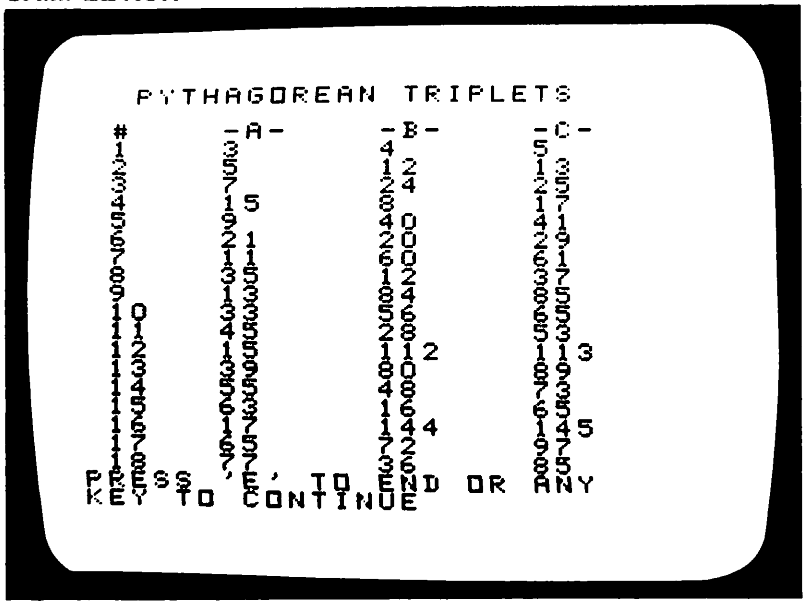
1. R is greater than S,
2. R and S are of opposite parity (one is odd and the other is even), and
3. R and S are relatively prime (they have no common integer divisors except 1),

then a, b, and c can be found as follows:

$a=R^2-S^2$
$b=2RS$
$c=r^2+S^2$

The program starts with a value of two for R. It generates all possible S values for that R (starting at R − 1 and then decreasing) and then adds one to R and continues. So, the first set of triplets is created when R is two and S is one, the second set when R is three and S is two, and so on.

SAMPLE RUN



```
  PYTHAGOREAN  TRIPLETS
#        -A-       -B-       -C-
1        3         4         5
2        5         12        13
3        7         24        25
4        15        8         17
5        9         40        41
6        21        20        29
7        11        60        61
8        35        12        37
9        13        84        85
10       33        56        65
11       45        28        53
12       15        112       113
13       39        80        89
14       55        48        73
15       63        16        65
16       17        144       145
17       65        72        97
18       77        36        85
PRESS 'E' TO END OR ANY
KEY TO CONTINUE
```

The program shows the first screen of Pythagorean triplets.

PROGRAM LISTING

```
100 REM PYTHAG
110 REM (C) 1984 DILITHIUM PRESS
120 R=2
130 K=1
140 D=0
150 GOSUB 380
160 S=R-1
170 A=R*R-S*S
180 B=2*R*S
190 C=R*R+S*S
200 PRINT K;TAB(6);A;TAB(13);B;TAB(20);C
210 K=K+1
220 D=D+1
230 GOTO 430
240 S=S-2
250 IF S>0 THEN 280
260 R=R+1
270 GOTO 160
280 S1=S
290 B1=R
300 N=INT(B1/S1)
310 R1=B1-(S1*N)
320 IF R1=0 THEN 360
330 B1=S1
340 S1=R1
350 GOTO 300
360 IF S1<>1 THEN 240
370 GOTO 170
380 CALL CLEAR
390 PRINT "  PYTHAGOREAN TRIPLETS"
400 PRINT
410 PRINT " #";TAB(7);"-A-";TAB(14);"-B-";TAB(
21);"-C-"
420 RETURN
430 IF D<18 THEN 240
440 PRINT "PRESS 'E' TO END OR ANY"
450 PRINT "KEY TO CONTINUE"
460 CALL KEY(0,KK,SS)
470 IF SS=0 THEN 460
480 IF KK<>69 THEN 510
```

```
490 CALL CLEAR
500 END
510 GOSUB 380
520 D=0
530 GOTO 240
```

EASY CHANGES

1. Alter the starting value of R in line 120. Instead of 2, try 50 or 100.
2. If you want, you can change the number of sets of triplets displayed on each screen. Change the 18 in line 430 to a 10, for example. You probably won't want to try a value greater than 18, since that would cause the column headings to roll off the screen.
3. To make the program continue without requiring you to press a key for the next screen of values, insert either of these lines:

   ```
   435 GOTO 510
   ```

   or

   ```
   435 GOTO 240
   ```

   The first will display headings for each screen. The second will only display the headings at the beginning of the run.

MAIN ROUTINES

120–140	Initializes variables.
150	Displays the title and column headings.
160	Calculates the first value of S for current R value.
170–190	Calculates A, B, and C.
200–230	Displays one line of values. Adds to counters.
240–270	Calculates next S value. If no more, calculates next R value.
280–370	Determines if R and S are relatively prime.
380–420	Subroutine to display title and column headings.
430–530	Checks if screen is full yet. If so, waits for key to be pressed.

MAIN VARIABLES

R, S	See explanation in "How To Use It."
K	Count of total number of sets displayed.
D	Count of number of sets displayed on one screen.

A, B, C	Lengths of the three sides of the triangle.
S1, B1,	Used in determining if R and S are relatively prime.
R1, N	Key pressed by operator.
KK	Status from CALL KEY.
SS	

SUGGESTED PROJECTS

1. In addition to displaying K, A, B, and C on each line, display R and S. You will have to squeeze the columns closer together.
2. Because this program uses integer values that get increasingly larger, eventually some will exceed the TI-99/4A's integer capacity and produce incorrect results. Can you determine when this will be? Modify the program to stop when this occurs.

Bibliography

BOOKS

Bell, R. C., *Board and Table Games From Many Civilizations*, Oxford University Press, London, 1969. (WARI)

Brown, Jerald R., *Instant BASIC*, Dymax, Menlo Park, Calif., 1977. (Self-teaching text on the BASIC language)

Cohen, Daniel, *Biorhythms in Your Life*, Fawcett Publications, Greenwich, Connecticut, 1976. (BIORHYTHM)

Crow, E. L., David, F. A., and Maxfield, M. W., *Statistics Manual*, Dover Publications, New York, 1960. (STATS)

Croxton, F. E., Crowden, D. J., and Klein, S., *Applied General Statistics* (Third Edition), Prentice-Hall, Englewood Cliffs, N.J., 1967. (STATS)

Gruenberger, Fred J., and Jaffray, George, *Problems for Computer Solution*, John Wiley and Sons, New York, 1965. (BIRTHDAY, PI)

Gruenberger, Fred J., and McCracken, Daniel D., *Introduction to Electronic Computers,* John Wiley and Sons, New York, 1961. (MILEAGE, PI, PYTHAG, WARI as Oware)

Hildebrand, F. B., *Introduction to Numerical Analysis*, McGraw-Hill, New York, 1956. (CURVE, DIFFEQN, INTEGRATE, SIMEQN)

Kuo, S. S., *Computer Applications of Numerical Methods*, Addison-Wesley, Reading, Massachusetts, 1972. (CURVE, DIFFEQN, INTEGRATE, SIMEQN)

McCracken, Daniel D., and Dorn, W. S., *Numerical Methods and FORTRAN Programming*, John Wiley and Sons, New York, 1964. (CURVE, DIFFEQN, INTEGRATE, SIMEQN)

Shefter, Harry, *Faster Reading Self-Taught*, Washington Square Press, New York, 1960. (TACHIST)

PERIODICALS

Feldman, Phil, and Rugg, Tom, "Pass the Buck," *Kilobaud*, July 1977, pp. 90–96. (DECIDE)

Fliegel, H. F., and Van Flandern, T. C., "A Machine Algorithm for Processing Calendar Dates," *Communications of the ACM*, October, 1968, p. 657. (BIORHYTHM)

ERRATA OFFER

All of the programs in this book have been tested carefully and are working correctly to the best of our knowledge. However, we take no responsibility for any losses which may be suffered as a result of errors or misuse. You must bear the responsibility of verifying each program's accuracy and applicability for your purposes.

If you want to get a copy of an errata sheet that lists corrections for any errors or ambiguities we have found to date, send one dollar ($1.00) and a self-addressed stamped envelope (SASE) to the address below. Ask for errata for this book (by name).

If you think you've found an error, please let us know. If you want an answer, include a SASE.

Please keep in mind that the most likely cause of a program working incorrectly is a typing error. Check your typing *very* carefully before you send us an irate note about an error in one of the programs. Reread the "How To Use This Book" section, too.

Attention Software
dilithium Press
Errata—32 Programs for the TI-99/4A
8285 S.W. Nimbus
Suite 151
Beaverton, OR 97005

32 BASIC PROGRAMS FOR THE TI-99/4A COMPUTER

LOADING INSTRUCTIONS FOR DISKETTE

What You Need to Run 32 BASIC Programs for the TI-99/4A

TI-99/4A Personal Home Computer
TI Extended BASIC Cartridge (for ROADRACE and WALLOONS programs only)
Monitor or TV
Peripheral Expansion System
32K Memory Expansion Card
Disk Controller Card
Disk Drive
Disk Manager 2 Cartridge

Preliminary Steps

The first thing you should do when you receive your 32 BASIC Programs disk is to make a backup copy. To make a backup copy of a disk, follow the instructions in your Disk Memory System manual. After you have made a backup copy you are ready to load the programs.

Loading the Programs

Turn on the Expansion System, the computer (using the switch on the front right-hand side), and then the monitor or TV. A message will appear on the screen displaying the manufacturer's logo. Follow the instructions to continue. Another message will appear on the screen giving you the following options:

1 TI BASIC

2 TI Extended BASIC (if you have it)

Choose option 1 to use TI BASIC.

You will need to choose option 2, TI Extended BASIC, to load and run Walloons and Roadrace. You will then see a READY prompt (>) to indicate that the computer is waiting for your command response.

Now you are ready to select a program from the list below:

The following programs can be found on Side A:

Biorhythm	Checkbook	Decide
Loan	Mileage	Quest
Arith	Flashcard	Metric
Numbers	Tachist	Vocab
Decode	Groan	Jot
Obstacle	Roadrace	Wari

The following programs can be found on Side B:

Kaleido	Sparkle	Squares
Walloons	Curve	Diffeqn
Graph	Integrate	Simeqn
Stats	Birthday	Pi
Powers	Pythag	

After selecting a program, type

RUN "DSK1.(Program name)" and press the ENTER key.

If the program doesn't load, check to be sure you spelled the name of the program correctly, and try again. Also, be sure you have everything hooked up correctly, and that you have selected the correct BASIC to operate the programs.

Special consideration or instruction for using each of the programs can be found in the *32 BASIC Programs* book.

32 BASIC PROGRAMS FOR THE TI-99/4A COMPUTER

LOADING INSTRUCTIONS FOR CASSETTE

What You Need to Run 32 BASIC Programs for the TI-99/4A Computer

TI-99/4A Personal Home Computer
16K Memory
TI Extended BASIC Cartridge (needed for ROADRACE and
WALLOONS programs only)
Monitor or TV
Cassette Player (preferably with a tape counter)
32 BASIC Programs Cassette

Preliminary Steps

The first thing you should do when you receive your 32 BASIC Programs cassette is to make a backup copy. To make a backup copy you need a blank cassette. Load all of the programs one at a time, and save them on the blank cassette. When saving programs, be sure that you note the counter number at the beginning of each program. These will be of assistance to you when you wish to load programs individually. When you have a backup copy of the cassette, you are ready to use the programs.

Loading the Programs

Turn on the computer using the switch on the front right-hand side, and then the monitor or television. A message will appear on the screen displaying the manufacturer's logo. Follow the instructions to continue. Another message will appear, giving you the following options:

```
1 TI BASIC
2 TI EXTENDED BASIC (if you have it)
```

Choose option 1 to use TI BASIC. You will need to choose option 2, TI Extended BASIC, to run Walloons and Roadrace. You will then see a READY prompt (>) to indicate that the computer is waiting for your command response. Then type

```
OLD CS1
```

and press ENTER.

Follow the instructions on the screen to load the first program. The message *READING* indicates the tape is being read into memory. This process takes a minute or two. When the program is loaded, a READY prompt (>) will appear. Type

```
RUN
```

and press ENTER.

To load additional programs, fast forward or rewind to the tape position desired. Then type

```
OLD CS1
```

and press ENTER.

Follow the instructions on the screen, except the instruction to rewind the cassette, and when you get a READY prompt (>) on the screen, your program should be loaded. Now type

```
RUN
```

and press ENTER.

Loading a Program Using Tape Counter Positions

If you do not have a cassette with tape counter, you can still load the programs from the cassette, although it will be a more difficult and lengthy process.

To use Tape Counter Positions, simply load each program initially, recording tape counter positions as you load each one. By completing this process one time, you can then load programs by the tape counter positions you've written down. (This can be done when you back up your cassette.)

After you have chosen the desired option (as directed above), instead of loading the first program, you can load any program on the tape by fast forwarding to the appropriate tape counter position and typing

```
OLD CS1
```

followed by ENTER. The programs should load the same as if you were loading them sequentially.

If you have problems this way, try loading each program sequentially as they are found on the tape to ensure proper loading.

The following programs can be found on Side A:

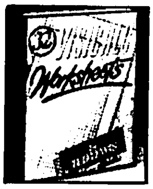
Biorhythm	Checkbook	Decide
Loan	Mileage	Quest
Arith	Flashcard	Metric
Numbers	Tachist	Vocab
Decode	Groan	Jot
Obstacle	Roadrace	Wari

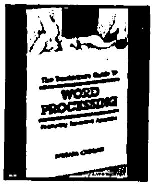
The following programs can be found on Side B:

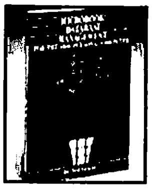
Kaleido	Sparkle	Squares
Walloons	Curve	Diffeqn
Graph	Integrate	Simeqn
Stats	Birthday	Pi
Powers	Pythag	

Special instructions or considerations for each program may be found in *32 BASIC Programs for the TI-99/4A Computer.*

There is no warranty or representation by dilithium Press or the authors that these programs will enable the user to achieve any particular result.







This book is chock-full of completely-tested, ready to run programs designed specifically for your TI-99/4A computer. It includes games, applications, educational programs, graphics, and mathematics. Each chapter fully documents a program by providing a complete source listing of the program, its purpose, and how to use it. Plus the authors tell you how to adapt the programs by making simple modifications.

Software available in specially marked packages. Includes: 5¼" disk or cassette containing all of the programs, the loading instructions and a warranty card with our Forever Replacement guarantee. The software runs on a TI-99/4A Personal Home Computer with:

- *16K Memory (plus 32 K Memory Expansion Card for disk)*
- *TI Extended BASIC Cartridge*
- *Peripheral Expansion System (disk only)*
- *Disk Controller Card (disk only)*
- *Disk Manager 2 Cartridge (disk only)*
- *1 disk drive or cassette recorder*
- *Color Monitor or Television*

dilithium Software offers technical support over the telephone. With our toll-free number and friendly staff, we can answer all your questions about dilithium Software. Outside of Oregon dial (800) 547-1842, in Oregon call 646-2713.

56136
0 45078 01995
ISBN 0-88056-136-X
>>$19.95

dilithium Press
SOFTWARE