dilithium Press

32 BASIC PROGRAMS FOR THE ATARI® COMPUTER

TOM RUGG, PHIL FELDMAN, TIMOTHY BARRY

PLACE
STAMP
HERE

dilithium Software
P.O. Box 606
Beaverton, OR 97075

32 BASIC Programs for the ATARI® Computer

32 BASIC Programs for the ATARI® Computer

Tom Rugg, Phil Feldman,
and Tim Barry

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 ATARI computer.

Bibliography: p.
Includes index.
1. Atari 400 (Computer)—Programming. 2. Atari 800 (Computer)—Programming. 3. Atari 1200XL (Computer)—Programming. 4. Basic (Computer program language)
I. Feldman, Phil. II. Barry, Tim. III. Title
IV. Title: Thirty-two BASIC programs for the ATARI computer.
QA76.8.A8R84 1984 001.64'2 83-23164
ISBN 0-88056-084-3

Cover: Bruce Fleming

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, John Craig, Merl and Patti Miller, Nancy Morrice, and Sally Bowman. Special thanks to Ray Alen.

AN IMPORTANT NOTE

Preface

You have bought yourself an ATARI 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?

An ATARI computer 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 ATARI (or each other) at some fascinating games. The possibilities go on and on.

All these things can be done with an ATARI computer, 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 the ATARI itself.
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 the ATARI'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 ATARI. 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 ATARI, 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 ATARI 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 an ATARI 400, 800, 600XL, 800XL, or 1200XL computer with ATARI BASIC. See Appendix 1 for details of the amount of memory required. 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 ATARI computer, 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 your ATARI computer to learn the fundamentals of communication with the computer. This will teach you how to turn the computer on, get into 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 ATARI's memory. Using Section 4 of the chapter ("Program Listing"), carefully enter the program into the ATARI. Be particularly careful to get all the punctuation characters right (i.e., commas, semicolons, colons, quotation marks, etc.). Certain typing errors will give you an immediate ERROR response. When this happens, reenter the line with corrections.
5. *After the entire program is entered into the ATARI'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 CSAVE or 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, severe typing errors, 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 ERROR in a line, LIST that line and look at it closely. Something is not right.

   Referring to the error code in the error message, consult the ATARI 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. Usually the error will be in another line that deals with the same variable names that are used in the line with the error. 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 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, CSAVE or 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, CSAVE or 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.

A NOTE ON THE PROGRAM LISTINGS

A line on the screen of an ATARI computer is 38 characters wide, unless reset by a program to 40. 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 48 characters. This combination of facts might cause you a little confusion when you are entering the programs into your ATARI. Here's the way it works.

Wherever there is a line in a program that is longer than 48 characters, it has been divided into two or more lines that are each no more than 48 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 ATARI as a single line.

Don't be fooled by the fact that the cursor on your ATARI jumps down to the next line after you enter the 38th or 40th character — it's just one long line until you press **RETURN.**

IMPORTANT NOTE

You can avoid all the problems of entering and debugging the programs in this book. How? All programs are available ready to run on both cassette and disk. You simply load the program and you're ready to go. The order card in the front of the book has complete information on how to order these timesaving pre-entered programs.

Contents

Section 1

Applications Programs

INTRODUCTION TO APPLICATIONS PROGRAMS

Good practical applications are certainly a prime use of personal computers. There are a myriad of ways an ATARI computer 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)? CHCKBOOK 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 ATARI is not limited to numerical applications. STOP turns your computer into a sophisticated stopwatch with a variety of uses.

Often we are faced with difficult decisions. DECIDE transforms the ATARI 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 BIORHYTH 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 ATARI and run this program.

BIORHYTH

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 ATARI computer.

HOW TO USE IT

The program first asks for the birth day 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.

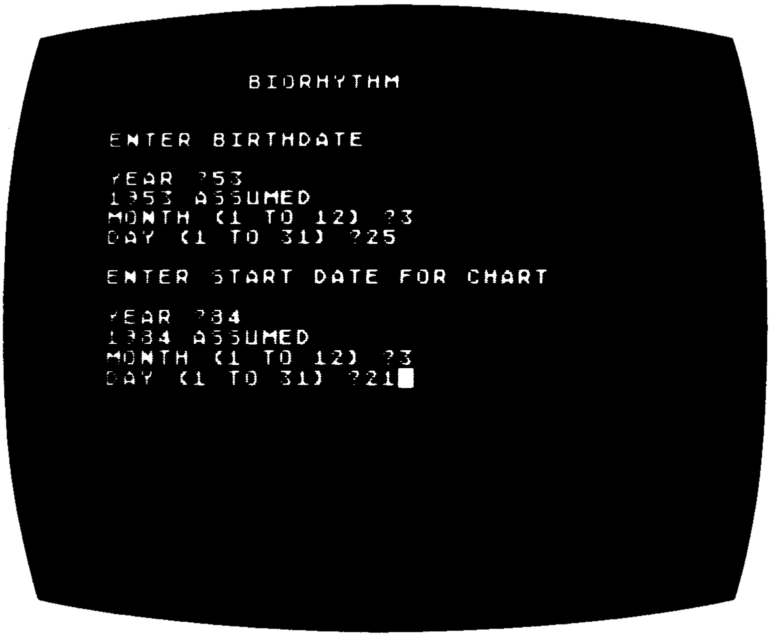
The left half of the chart is the "down" (negative) side of each cycle. The right half is the "up" (positive) side. The center line shows the critical days when you are at a zero point (neither positive nor negative).

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 program ends. If you press the **SPACE** key (or almost any other key except **BREAK** or **SHIFT**), the program clears the screen and displays the next eighteen days of the chart.

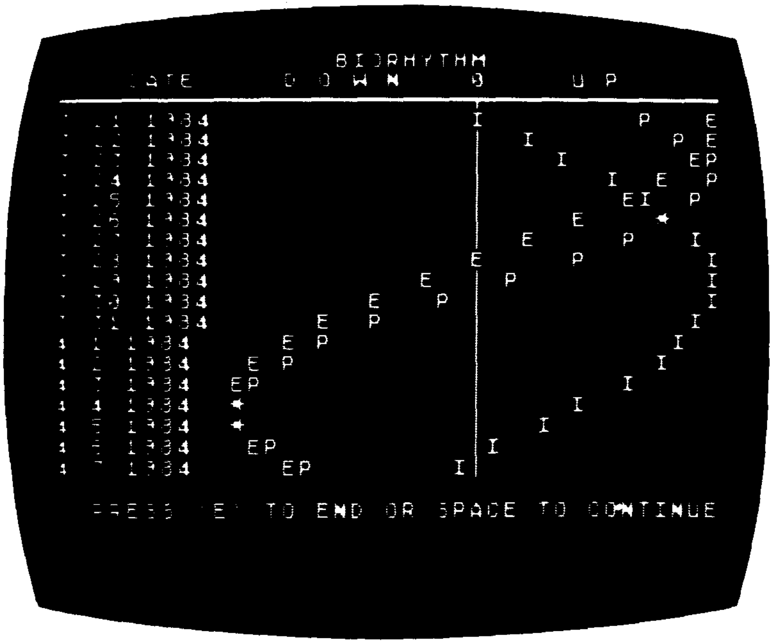
The program will allow you to enter dates from the year 100 A.D. and on. We make no guarantees about any extreme future dates, however, such as entering a year greater than 3000.

SAMPLE RUN



```
            BIORHYTHM

ENTER BIRTHDATE

YEAR ?53
1953 ASSUMED
MONTH (1 TO 12) ?3
DAY (1 TO 31) ?25

ENTER START DATE FOR CHART

YEAR ?84
1984 ASSUMED
MONTH (1 TO 12) ?3
DAY (1 TO 31) ?21█
```

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



```
                 BIORHYTHM
    DATE      D O W N    0     U P
----------------------------------------
7 21 1984                I          P   E
7 22 1984                |  I         P E
7 23 1984                |    I        EP
7 24 1984                |        I  E  P
7 25 1984                |         EI  P
7 26 1984                |     E     *
7 27 1984                |  E      P   I
7 28 1984                E     P        I
7 29 1984            E  | P            I
7 30 1984         E   P |              I
7 31 1984      E  P     |             I
8 1 1984      E P        |            I
8 2 1984    E P          |           I
8 3 1984   EP            |         I
8 4 1984   *             |     I
8 5 1984   *             |   I
8 6 1984    EP           |I
8 7 1984      EP        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

```
10 REM BIORHYTH
15 REM COPYRIGHT 1984 DILITHIUM PRESS
20 POKE 82,0
30 DIM U$(40),X(12),C$(40),L$(40),R$(40),A$(40),
X1$(40),X2$(40),X3$(40),W$(40)
40 FOR J=1 TO 12:READ A:X(J)=A:NEXT J
50 DATA 31,28,31,30,31,30,31,31,30,31,30,31
60 L=0:Z=0.99999:T=14:P=3.14159265
70 FOR J=1 TO 29:U$(J,J)=CHR$(18):W$(J,J)=CHR$(3
2):NEXT J
80 U$(15,15)=CHR$(23):W$(15,15)=CHR$(124)
90 GRAPHICS 0:PRINT "            BIORHYTHM":PRINT
:PRINT
100 PRINT "  ENTER BIRTHDATE"
110 GOSUB 240:JB=JD
120 PRINT :PRINT "  ENTER START DATE FOR CHART"
130 GOSUB 240:JC=JD:IF JC>=JB THEN 150
140 PRINT "  CHART DATE CAN'T BE EARLIER THAN BI
RTHDATE - TRY AGAIN"
145 GOTO 100
150 POKE 752,1:FOR J=1 TO 1000:NEXT J
160 GOSUB 330
```

```
170 N=JC-JB:V=23:GOSUB 350:V=28:GOSUB 350:V=33:G
OSUB 350
180 GOSUB 490:L=L+1:POKE 752,1:PRINT C$:POSITION
 10,L+3:PRINT L$:JC=JC+1
190 IF L<18 THEN 170
200 POKE 764,255:PRINT :PRINT "  PRESS 'E' TO EN
D OR SPACE TO CONTINUE";
210 A=PEEK(764):IF A=255 THEN 210
220 IF A=42 THEN POKE 764,255:POKE 752,0:POKE 82
,2:END
230 L=0:GOTO 160
240 X(2)=28:PRINT :PRINT "  YEAR ";:INPUT Y:Y=IN
T(Y)
245 IF Y<0 THEN GOSUB 540:GOTO 240
250 IF (Y/100)-(INT(Y/100))<>0 THEN 270
260 IF (Y/400)-(INT(Y/400))<>0 THEN 280
270 IF (Y/4)-(INT(Y/4))=0 THEN X(2)=29
275 IF Y<=99 THEN Y=Y+1900:PRINT "  ";Y;" ASSUME
D"
280 PRINT "  MONTH (1 TO 12) ";:INPUT M:M=INT(M)
:IF M<1 OR M>12 THEN GOSUB 540:GOTO 280
290 PRINT "  DAY (1 TO ";X(M);") ";:INPUT D:IF D
<1 OR D>X(M) THEN GOSUB 540:GOTO 290
300 W=INT((M-14)/12+Z):JD=INT(1461*(Y+4800+W)/4)
:B=367*(M-2-W*12)/12
310 IF B<0 THEN B=B+Z
320 B=INT(B):JD=JD+B:B=INT(INT(3*(Y+4900+W)/100)
/4):JD=JD+D-32075-B:RETURN
330 GRAPHICS 0:POSITION 16,1:POKE 752,1:PRINT "B
IORHYTHM":POSITION 4,2:PRINT "DATE"
340 POSITION 13,2:PRINT "D O W N    O     U P":F
OR J=1 TO 10:PRINT CHR$(18);:NEXT J:POSITION 10,
3:PRINT U$:RETURN
350 W=INT(N/V):R=N-W*V
360 IF V<>23 THEN 390
370 L$=W$
380 IF V=23 THEN C$="P"
390 IF V=28 THEN C$="E"
400 IF V=33 THEN C$="I"
410 W=R/V:W=W*2*P:W=T*SIN(W):W=W+T+1.5:W=INT(W):
A$=L$(W,W)
420 IF A$="P" OR A$="E" OR A$="*" THEN C$="*"
430 IF W=1 THEN 470
440 IF W=T+T+1 THEN 480
450 X1$=L$(1,W-1):X2$=L$(LEN(L$)-(T+T-W),LEN(L$)
):C=LEN(C$):X1=LEN(X1$):X2=LEN(X2$)
460 L$(1,X1)=X1$:L$(X1+1,X1+C)=C$:L$(X1+C+1,X1+C
+X2)=X2$:RETURN
470 X1$=L$(LEN(L$)-(T+T-1),LEN(L$)):C=LEN(C$):X1
=LEN(X1$):L$(1,C)=C$:L$(C+1,C+X1)=X1$:RETURN
480 X1$=L$(1,T+T):X1=LEN(X1$):C=LEN(C$):L$(1,X1)
=X1$:L$(X1+1,X1+C)=C$:RETURN
```

```
490 W=JC+68569:R=INT(4*W/146097):W=W-INT((146097
*R+3)/4):Y=INT(4000*(W+1)/1461001):W=W-INT(1461*
Y/4)+31
500 M=INT(80*W/2447):D=W-INT(2447*M/80):W=INT(M/
11):M=M+2-12*W:Y=100*(R-49)+Y+W
510 X1$=STR$(M):X2$=STR$(D):X3$=STR$(Y):X1=LEN(X
1$):X2=LEN(X2$):X3=LEN(X3$)
520 C$(1,X1)=X1$:C$(X1+1,X1+1)="/":C$(X1+2,X1+X2
+1)=X2$:C$(X1+2+X2,X1+2+X2)="/"
530 C$(X1+3+X2,X1+2+X2+X3)=X3$:RETURN
540 PRINT :PRINT "  ** ILLEGAL ENTRY. TRY AGAIN.
 **":PRINT :RETURN
```

EASY CHANGES

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

   175 PRINT "DAYS=";N: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 shown on each screen, alter the 18 in line 190.

MAIN ROUTINES

10– 90	Initializes variables. Displays titles.
100–110	Asks for birth date and converts to Julian date format (i.e., the number of days since January 1, 4713 B.C.).
120–145	Asks for start date for chart and converts to Julian date format and checks that chart date is not sooner than birth date.
150	Delays about one second before displaying chart.
160	Displays heading at top of screen.
170	Determines number of days between birth date and current chart date and plots points in L$ string for each of the three cycles. Converts Julian date back into month/day/year format.
180	Displays one line on the chart and adds one to chart date.
190–230	Checks to see if the screen is full.
240–320	Subroutine to ask operator for month, day, and year. Edits replies.
330–340	Subroutine to clear screen and display headings.

350–480	Subroutine to convert month, day, and year into Julian date format.
490–530	Subroutine to convert Julian data JC back into month/day/year format.
540	Subroutine to print error message.

MAIN VARIABLES

L	Counter of number of lines on screen.
T	Number of characters on one side of the center 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	Loop and work variable.
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.
L$	String with one line of the biorhythm chart.
R$	Reply from operator after screen fills up.
M	Month (1–12).
D	Day (1–31).
Y	Year (100 or greater).
W, B	Work variables.
R	Remainder of N/V (number of days into cycle).
A$	Work variable.
U$, W$	Formatting strings.

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?

CHCKBOOK

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 ATARI 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 press the **RETURN** key.

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.

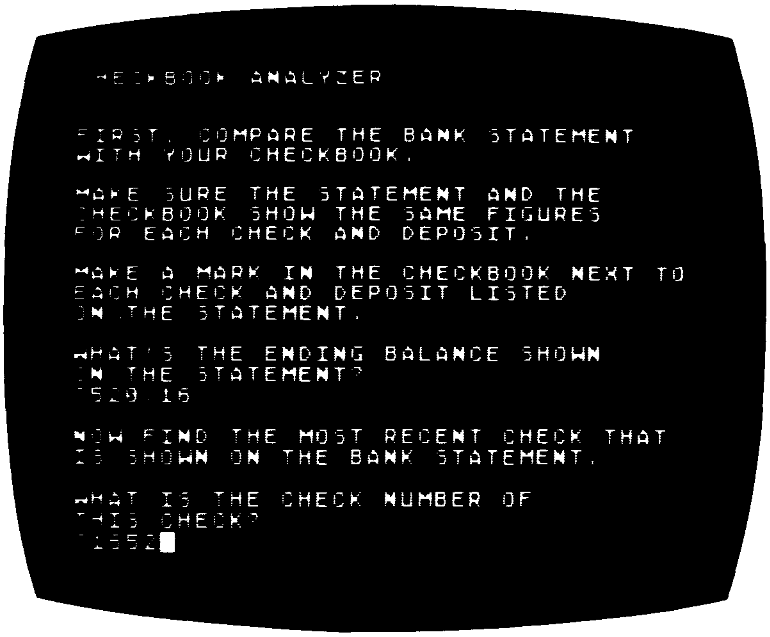
Another common error is accidentally adding the amount of a check in your checkbook instead of subtracting it. If you did this, your error will be twice the amount of the check (which makes it easy to find).

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.

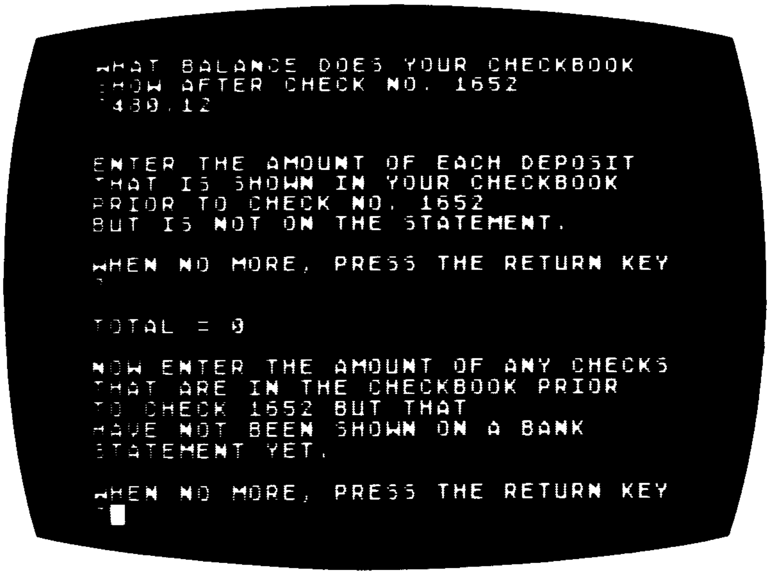
NOTE: SEE DISCLAIMER IN FRONT PART OF BOOK.

SAMPLE RUN



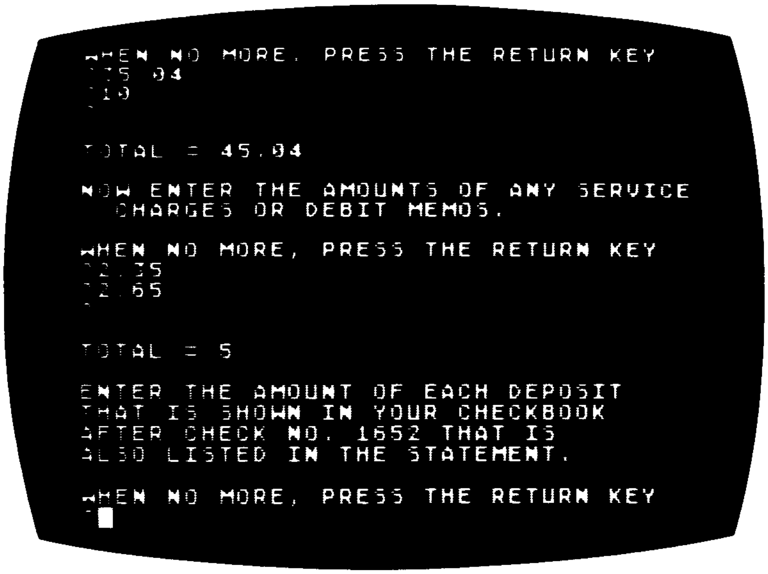
```
  CHECKBOOK ANALYZER

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
IN 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?
?1352█
```

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



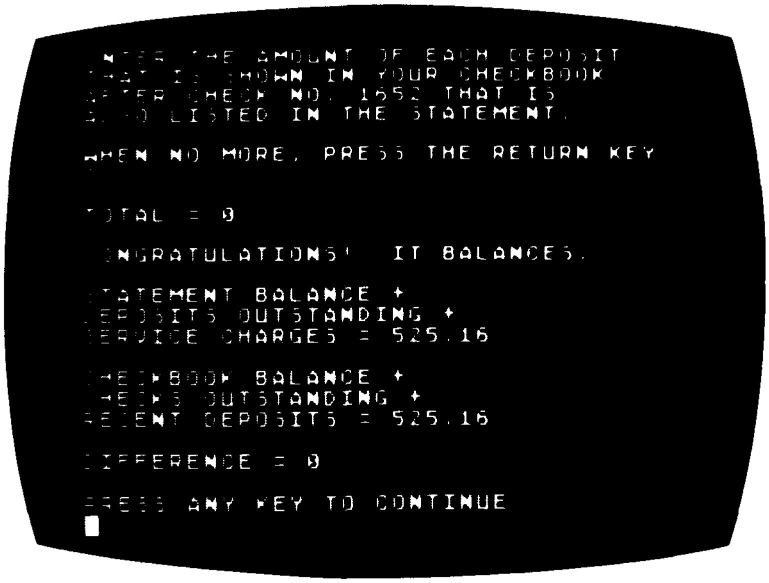
```
WHAT BALANCE DOES YOUR CHECKBOOK
SHOW AFTER CHECK NO. 1652
?430.12


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, PRESS THE RETURN KEY
?

TOTAL = 0

NOW ENTER THE AMOUNT OF ANY CHECKS
THAT ARE IN THE CHECKBOOK PRIOR
TO CHECK 1652 BUT THAT
HAVE NOT BEEN SHOWN ON A BANK
STATEMENT YET.

WHEN NO MORE, PRESS THE RETURN KEY
?█
```

The operator continues by entering the checkbook balance, and then presses **RETURN** to indicate no outstanding deposits.



```
WHEN NO MORE, PRESS THE RETURN KEY
?35.04
?10
?

TOTAL = 45.04

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

WHEN NO MORE, PRESS THE RETURN KEY
?2.35
?2.65
?

TOTAL = 5

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

WHEN NO MORE, PRESS THE RETURN KEY
?█
```

The operator enters the outstanding checks and the service charges.



```
ENTER THE AMOUNT OF EACH DEPOSIT
THAT IS SHOWN IN YOUR CHECKBOOK
AFTER CHECK NO. 1552 THAT IS
NOT LISTED IN THE STATEMENT.

WHEN NO MORE, PRESS THE RETURN KEY

TOTAL = 0

 CONGRATULATIONS!  IT BALANCES.

STATEMENT BALANCE +
DEPOSITS OUTSTANDING +
SERVICE CHARGES = 525.16

CHECKBOOK BALANCE +
CHECKS OUTSTANDING +
RECENT DEPOSITS = 525.16

DIFFERENCE = 0

PRESS ANY KEY TO CONTINUE
█
```

After the service charges are entered, the operator indicates no late deposits by pushing **RETURN**. The program displays balancing information and waits for a key to be pressed.

PROGRAM LISTING

```
10 REM CHCKBOOK
15 REM COPYRIGHT 1984 DILITHIUM PRESS
20 DIM L$(20),E$(30),A$(40),R$(40)
30 GRAPHICS 0
40 PRINT "CHECKBOOK ANALYZER"
50 PRINT :PRINT
60 MC=20:MD=10:MS=10:MR=10
70 L$="NO MORE ROOM"
80 DIM C(MC),D(MD),S(MS),R(MR)
90 TC=0:TD=TC:TS=TC:TR=TC:ND=TC:NC=TC:NS=TC:NR=T
C
100 E$="ERROR. RE-ENTER, PLEASE."
110 PRINT "FIRST, COMPARE THE BANK STATEMENT"
120 PRINT "WITH YOUR CHECKBOOK.":PRINT
130 PRINT "MAKE SURE THE STATEMENT AND THE "
140 PRINT "CHECKBOOK SHOW THE SAME FIGURES"
150 PRINT "FOR EACH CHECK AND DEPOSIT."
160 PRINT :PRINT "MAKE A MARK IN THE CHECKBOOK N
EXT TO "
170 PRINT "EACH CHECK AND DEPOSIT LISTED"
180 PRINT "ON THE STATEMENT."
190 PRINT :PRINT "WHAT'S THE ENDING BALANCE SHOW
N"
```

```
200 PRINT "ON THE STATEMENT? ":INPUT SB
210 PRINT :PRINT "NOW FIND THE MOST RECENT CHECK
 THAT"
220 PRINT "IS SHOWN ON THE BANK STATEMENT."
230 PRINT
240 PRINT "WHAT IS THE CHECK NUMBER OF"
250 PRINT "THIS CHECK? ":INPUT LC
260 IF LC=INT(LC) THEN 290
270 PRINT "NO, NOT THE AMOUNT OF THE CHECK."
280 GOTO 210
290 PRINT
300 PRINT "WHAT BALANCE DOES YOUR CHECKBOOK"
310 PRINT "SHOW AFTER CHECK NO. ";LC
320 INPUT CB
330 PRINT :PRINT
340 PRINT "ENTER THE AMOUNT OF EACH DEPOSIT"
350 PRINT "THAT IS SHOWN IN YOUR CHECKBOOK"
360 PRINT "PRIOR TO CHECK NO. ";LC
370 PRINT "BUT IS NOT ON THE STATEMENT."
380 A$="WHEN NO MORE, PRESS THE RETURN KEY":PRIN
T :PRINT A$
390 INPUT R$
400 IF LEN(R$)=0 THEN 460
410 IF VAL(R$)>0 THEN 430
420 PRINT :PRINT E$:GOTO 380
430 ND=ND+1:D(ND)=VAL(R$):TD=TD+D(ND)
440 IF ND<MD THEN 390
450 PRINT :PRINT L$
460 PRINT :PRINT "TOTAL = ";TD:PRINT
470 PRINT "NOW ENTER THE AMOUNT OF ANY CHECKS"
480 PRINT "THAT ARE IN THE CHECKBOOK PRIOR "
490 PRINT "TO CHECK ";LC;" BUT THAT"
500 PRINT "HAVE NOT BEEN SHOWN ON A BANK "
510 PRINT "STATEMENT YET."
520 PRINT :PRINT A$
530 INPUT R$
540 IF LEN(R$)=0 THEN 610
550 IF VAL(R$)>0 THEN 580
560 PRINT :PRINT E$
570 GOTO 520
580 NC=NC+1:C(NC)=VAL(R$):TC=TC+C(NC)
590 IF NC<MC THEN 530
600 PRINT :PRINT L$
610 PRINT :PRINT "TOTAL = ";TC:PRINT
620 PRINT "NOW ENTER THE AMOUNTS OF ANY SERVICE
   CHARGES OR DEBIT MEMOS."
630 PRINT :PRINT A$
640 INPUT R$
650 IF LEN(R$)=0 THEN 710
660 IF VAL(R$)>0 THEN 680
670 PRINT :PRINT E$:GOTO 630
680 NS=NS+1:S(NS)=VAL(R$):TS=TS+S(NS)
```

```
690 IF NS<MS THEN 640
700 PRINT :PRINT L$
710 PRINT :PRINT "TOTAL = ";TS:PRINT
720 GOSUB 1210
730 W=SB+TD+TS-CB-TC-TR:W=ABS(W)
740 IF W<1.0E-03 THEN W=0
750 IF W<>0 THEN 780
760 PRINT "CONGRATULATIONS!  IT BALANCES."
770 GOTO 790
780 PRINT :PRINT "SORRY, IT'S OUT OF BALANCE."
790 PRINT
800 PRINT "STATEMENT BALANCE +"
810 PRINT "DEPOSITS OUTSTANDING +"
815 PRINT "SERVICE CHARGES = ";SB+TD+TS
820 PRINT
830 PRINT "CHECKBOOK BALANCE +"
835 PRINT "CHECKS OUTSTANDING +"
840 PRINT "RECENT DEPOSITS = ";CB+TC+TR
850 PRINT
860 PRINT "DIFFERENCE = ";W
870 PRINT :PRINT "PRESS ANY KEY TO CONTINUE":CLO
SE #1:OPEN #1,4,0,"K:"
880 GET #1,R
890 PRINT
900 PRINT "NEXT ACTION:"
910 PRINT "1 - LIST CHECKS OUTSTANDING"
920 PRINT "2 - LIST DEPOSITS OUTSTANDING"
930 PRINT "3 - LIST SERVICE CHARGES"
940 PRINT "4 - START OVER"
950 PRINT "5 - END PROGRAM"
960 PRINT "6 - DISPLAY BALANCING INFO"
970 PRINT "7 - LIST DEPOSITS AFTER LAST CHECK"
980 GET #1,R:IF R<49 OR R>56 THEN 980
990 R=VAL(CHR$(R))
1000 IF R<1 OR R>7 THEN 1020
1010 ON R GOTO 1030,1070,1110,1150,1160,790,1170
1020 PRINT :PRINT E$:GOTO 890
1030 PRINT :PRINT "CHECKS OUTSTANDING"
1040 FOR J=1 TO NC
1050 PRINT C(J):NEXT J
1060 GOTO 870
1070 PRINT :PRINT "DEPOSITS OUTSTANDING"
1080 FOR J=1 TO ND
1090 PRINT D(J):NEXT J
1100 GOTO 870
1110 PRINT :PRINT "SERVICE CHARGES"
1120 FOR J=1 TO NS
1130 PRINT S(J):NEXT J
1140 GOTO 870
1150 CLR :GOTO 20
1160 END
1170 PRINT :PRINT "RECENT DEPOSITS"
```

```
1180 FOR J=1 TO NR
1190 PRINT R(J):NEXT J
1200 GOTO 870
1210 PRINT "ENTER THE AMOUNT OF EACH DEPOSIT"
1220 PRINT "THAT IS SHOWN IN YOUR CHECKBOOK "
1230 PRINT "AFTER CHECK NO. ";LC;" THAT IS"
1240 PRINT "ALSO LISTED IN THE STATEMENT."
1250 PRINT :PRINT A$
1260 INPUT R$
1270 IF LEN(R$)=0 THEN 1330
1280 IF VAL(R$)>0 THEN 1300
1290 PRINT :PRINT E$:GOTO 1250
1300 NR=NR+1:R(NR)=VAL(R$):TR=TR+R(NR)
1310 IF NR<MR THEN 1260
1320 PRINT :PRINT L$
1330 PRINT :PRINT "TOTAL = ";TR:PRINT
1340 RETURN
```

EASY CHANGES

Change the limitations of how many entries you can make in each category. Line 60 establishes these limits. If you have more than 20 checks outstanding at some time, change the value of MC to 50, 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), the number of service charges and credit memos (MS), and the number of recent deposits and credit memos (MR). You'll need sufficient memory free to make these changes. (See Appendix 1.)

MAIN ROUTINES

10– 180	Initializes variables and displays first instructions.
190– 280	Gets most recent check number.
290– 330	Gets checkbook balance after most recent check number.
340– 460	Gets outstanding deposits.
470– 610	Gets outstanding checks.
620– 710	Gets service charges and debit memos.
720	Gets recent deposits and credit memos.
730– 890	Does balancing calculation. Displays it.
900–1020	Asks for next action. Goes to appropriate subroutine.
1030–1060	Subroutine to display checks outstanding.
1070–1100	Subroutine to display deposits outstanding.
1110–1140	Subroutine to display service charges and debit memos.

1150	Restarts program.
1160	Ends the program.
1170–1200	Subroutine to display recent deposits.
1210–1340	Subroutine to get recent deposits.

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.
D	Array for deposits outstanding.
S	Array for service charges and debit memos.
R	Array for recent deposits and credit memos.
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.
ND	Number of deposits outstanding.
NS	Number of service charges and debit memos.
NR	Number of recent deposits and credit memos.
E$	Error message.
SB	Statement balance.
LC	Number of last check on statement.
CB	Checkbook balance after last check on statement.
R$	Reply from operator.
W	Amount by which checkbook is out of balance.
R	Numeric value of reply for next action.
A$	Message showing how to indicate no more data.
L$	Message indicating no more room for data.
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 ATARI 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 then press **RETURN** 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.

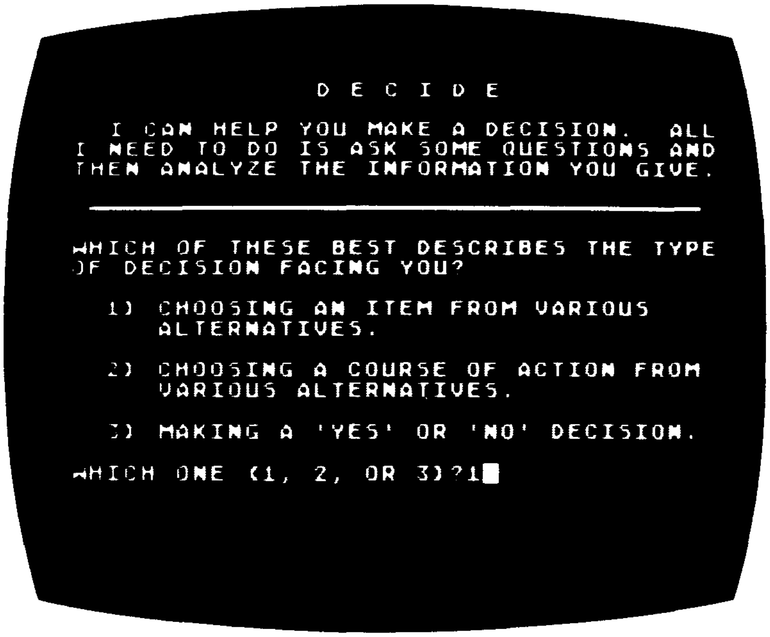
Now you must rate the decision possibilities with respect to each of the importance factors. Each importance factor will be treated separately. Considering *only* that importance factor, you must rate how each decision possibility stacks up. The program first assigns a value of 10 to one of the decision 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.

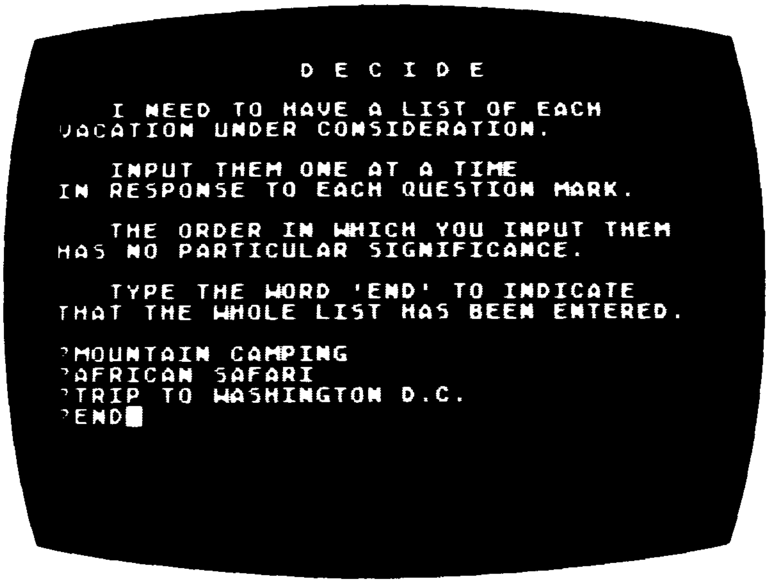
Of course, DECIDE should not be used as a substitute for good, clear thinking. However, it can often provide valuable insights. You might find one alternative coming out surprisingly low or high. A trend may become obvious when the program is re-run with improved data. At least, it may help you think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

SAMPLE RUN



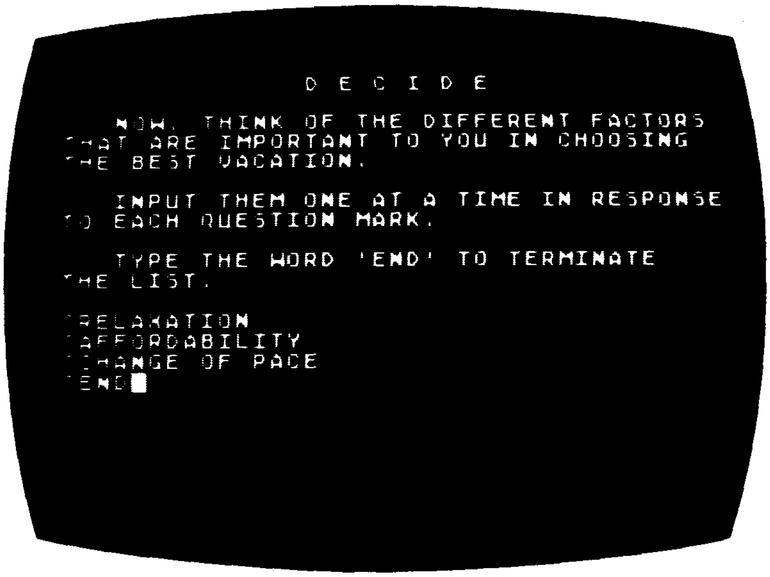
```
                D E C I D E

  I CAN HELP YOU MAKE A DECISION.  ALL
I NEED TO DO IS ASK SOME QUESTIONS AND
THEN 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) MAKING A 'YES' OR 'NO' DECISION.

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.



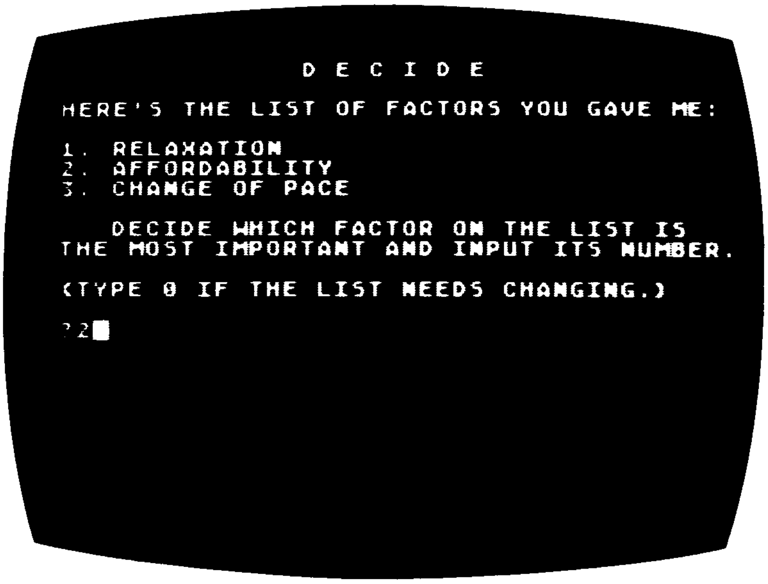
```
                  D E C I D E

   I NEED TO HAVE 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 PARTICULAR SIGNIFICANCE.

   TYPE THE WORD 'END' TO INDICATE
THAT THE WHOLE LIST HAS BEEN ENTERED.

?MOUNTAIN CAMPING
?AFRICAN SAFARI
?TRIP TO WASHINGTON D.C.
?END
```

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



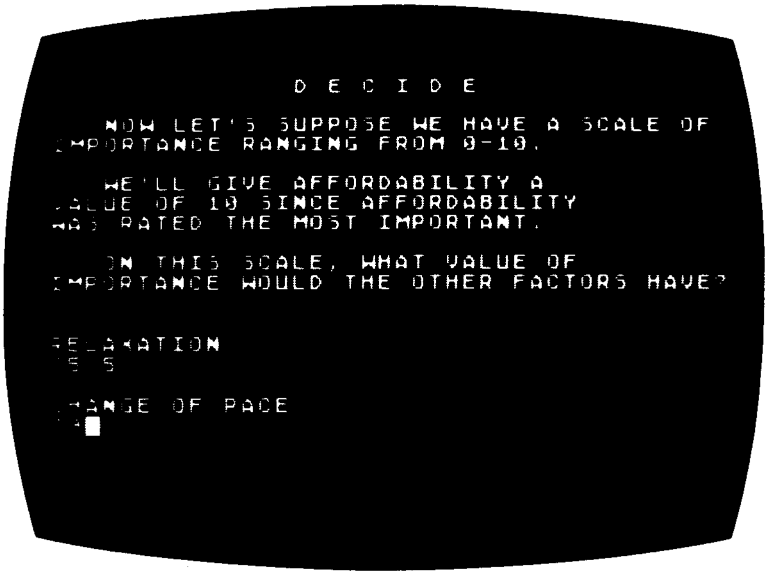
```
                  D E C I D E

   NOW, THINK OF THE DIFFERENT FACTORS
THAT ARE IMPORTANT TO YOU IN CHOOSING
THE BEST VACATION.

   INPUT 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.



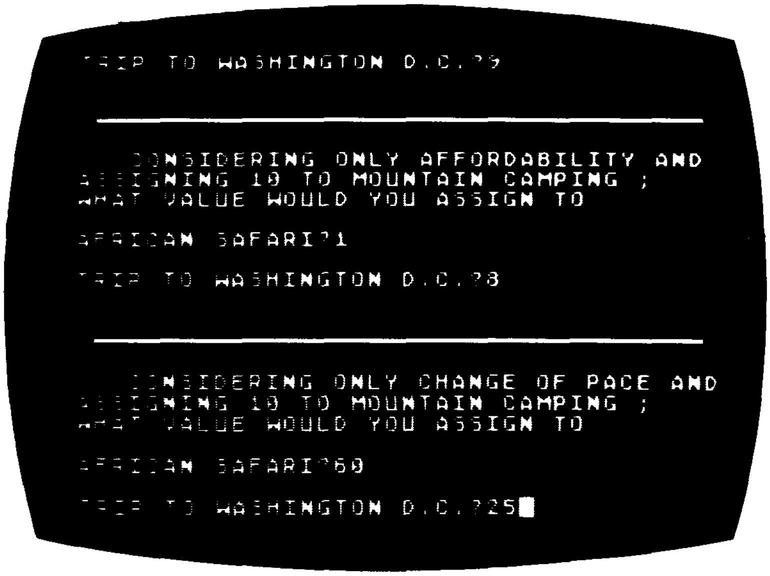
```
              D E C I D E

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.



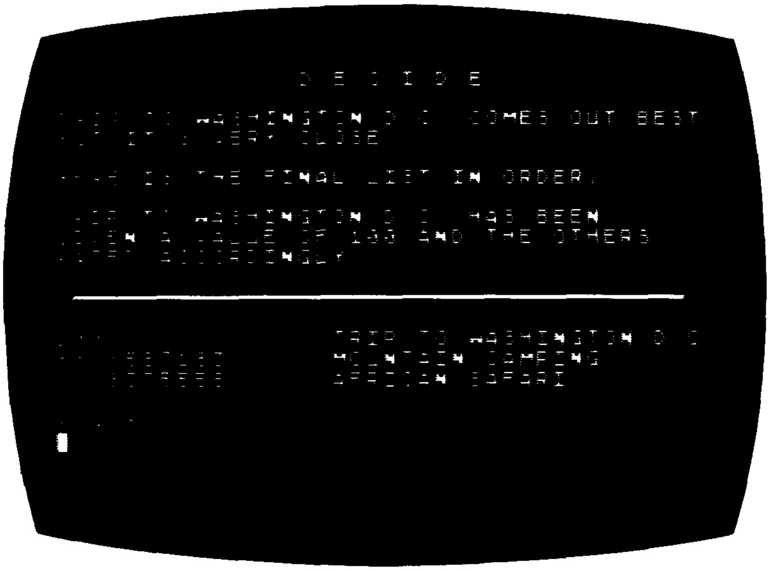
```
              D E C I D E

   NOW LET'S SUPPOSE WE HAVE A SCALE OF
IMPORTANCE RANGING FROM 0-10.

   WE'LL GIVE AFFORDABILITY A
VALUE OF 10 SINCE AFFORDABILITY
WAS RATED THE MOST IMPORTANT.

   ON THIS SCALE, WHAT VALUE OF
IMPORTANCE WOULD THE OTHER FACTORS HAVE?


RELAXATION
?5 5

CHANGE OF PACE
?4█
```

The operator selects numerical values for the other two factors.



```
TRIP TO WASHINGTON D.C.?9

 ____________________________________

  CONSIDERING ONLY AFFORDABILITY AND
ASSIGNING 10 TO MOUNTAIN CAMPING ;
WHAT VALUE WOULD YOU ASSIGN TO

AFRICAN SAFARI?1

TRIP TO WASHINGTON D.C.?8

 ____________________________________

  CONSIDERING ONLY CHANGE OF PACE AND
ASSIGNING 10 TO MOUNTAIN CAMPING ;
WHAT VALUE WOULD YOU ASSIGN TO

AFRICAN SAFARI?60

TRIP TO WASHINGTON D.C.?25█
```

After being given instructions on how to rate each vacation with respect to each factor, and entering 3 for the African safari, the operator provides the additional requested data.



```
              D E C I D E

TRIP TO WASHINGTON D.C. COMES OUT BEST
BUT IT IS VERY CLOSE

HERE IS THE FINAL LIST IN ORDER.

TRIP TO WASHINGTON D.C. HAS BEEN
GIVEN A VALUE OF 100 AND THE OTHERS
RATED ACCORDINGLY

 ____________________________________

[illegible]          TRIP TO WASHINGTON D.C.
[illegible]          MOUNTAIN CAMPING
[illegible]          AFRICAN SAFARI

[illegible]
█
```

The program displays the choices in order of desirability.

PROGRAM LISTING

```
10 REM DECIDE
15 REM COPYRIGHT 1984 DILITHIUM PRESS
150 GRAPHICS 0:POKE 82,0:PRINT CHR$(125)
160 DIM L$(500),F$(500),V(10),C(10,10),D(10),Z(1
0),T$(40),X$(40),E$(10),R$(40),LX(10),FX(10)
170 FOR J=1 TO 500:L$(J)=CHR$(32):F$(J)=CHR$(32)
:NEXT J
180 E$="END"
200 GOSUB 2000
210 PRINT "  I CAN HELP YOU MAKE A DECISION.  AL
L"
220 PRINT "I NEED TO DO IS ASK SOME QUESTIONS AN
D"
230 PRINT "THEN ANALYZE THE INFORMATION YOU GIVE
.":PRINT
240 GOSUB 2050:PRINT
250 PRINT "WHICH OF THESE BEST DESCRIBES THE TYP
E"
260 PRINT "OF DECISION FACING YOU?":PRINT
270 PRINT "  1) CHOOSING AN ITEM FROM VARIOUS"
280 PRINT "     ALTERNATIVES.":PRINT
290 PRINT "  2) CHOOSING A COURSE OF ACTION FROM
"
300 PRINT "     VARIOUS ALTERNATIVES.":PRINT
310 PRINT "  3) MAKING A 'YES' OR 'NO' DECISION.
":PRINT
320 PRINT "WHICH ONE (1, 2, OR 3)";
330 INPUT R$
335 IF R$="" THEN 200
340 IF R$<>"1" AND R$<>"2" AND R$<>"3" THEN ? :?
 "** ILLEGAL ENTRY **":? :GOTO 320
345 T=VAL(R$)
350 GOSUB 2000
400 ON T GOTO 410,440,470
410 PRINT "WHAT TYPE OF ITEM MUST YOU DECIDE UPO
N"
420 INPUT T$:IF T$="" THEN 410
430 GOTO 500
440 T$="COURSE OF ACTION":GOTO 500
470 T$="'YES' OR 'NO'"
480 NI=2:L$(1,12)="DECIDING YES":L$(51,61)="DECI
DING NO":LX(1)=12:LX(2)=11
490 GOTO 750
500 GOSUB 2000:NI=0
510 PRINT "   I NEED TO HAVE A LIST OF EACH"
520 PRINT T$;" UNDER CONSIDERATION.":PRINT
530 PRINT "   INPUT THEM ONE AT A TIME"
540 PRINT "IN RESPONSE TO EACH QUESTION MARK.":P
RINT
550 PRINT "   THE ORDER IN WHICH YOU INPUT THEM"
```

```
560 PRINT "HAS NO PARTICULAR SIGNIFICANCE.":PRIN
T
570 PRINT "   TYPE THE WORD '";E$;"' TO INDICATE
"
580 PRINT "THAT THE WHOLE LIST HAS BEEN ENTERED.
":PRINT
590 NI=NI+1:INPUT X$:IF X$=E$ THEN 610
592 IF X$="" THEN NI=NI-1:PRINT :PRINT "** ILLEG
AL ENTRY **":PRINT :GOTO 590
594 IF LEN(X$)>35 THEN PRINT "* YOUR ITEM IS TOO
 LONG. RE-ENTER IT. *":FOR K=1 TO 1000:NEXT K:NI
=NI-1:GOTO 590
595 L$((NI-1)*50+1,(NI-1)*50+LEN(X$))=X$:LX(NI)=
LEN(X$)
600 GOTO 590
610 NI=NI-1
620 IF NI>=2 THEN 650
630 PRINT :PRINT "* YOU MUST HAVE AT LEAST 2 CHO
ICES *":PRINT
640 PRINT "** TRY AGAIN **":GOSUB 2100:GOTO 500
650 GOSUB 2000:PRINT "HERE'S THE LIST YOU'VE GIV
EN ME":PRINT
660 FOR J=1 TO NI:PRINT J;". ";L$((J-1)*50+1,(J-
1)*50+LX(J)):NEXT J:PRINT
670 PRINT "IS THIS CORRECT (Y OR N) ";
680 INPUT R$
685 IF R$="" THEN PRINT :PRINT "** ILLEGAL ENTRY
 **":PRINT :GOTO 680
690 IF R$(1,1)="Y" THEN 750
695 IF R$(1,1)<>"N" THEN 670
700 PRINT "** THE LIST MUST BE RE-ENTERED **"
710 GOSUB 2100:GOTO 500
750 GOSUB 2000
760 PRINT "   NOW, THINK OF THE DIFFERENT FACTOR
S"
770 IF T<3 THEN PRINT "THAT ARE IMPORTANT TO YOU
 IN CHOOSING"
780 IF T<3 THEN PRINT "THE BEST ";T$;"."
790 IF T=3 THEN PRINT "THAT ARE IMPORTANT TO YOU
 IN DECIDING":PRINT "YES OR NO"
800 PRINT :PRINT "   INPUT THEM ONE AT A TIME IN
 RESPONSE"
810 PRINT "TO EACH QUESTION MARK.":PRINT
820 PRINT "   TYPE THE WORD '";E$;"' TO TERMINAT
E"
830 PRINT "THE LIST.":PRINT :NF=0
840 NF=NF+1:INPUT X$:IF X$=E$ THEN 860
842 IF X$="" THEN PRINT "** ILLEGAL ENTRY **":NF
=NF-1:GOTO 840
845 IF LEN(X$)>35 THEN PRINT :PRINT "* YOUR ITEM
 IS TOO LONG. RE-ENTER IT.":PRINT :FOR K=1 TO 10
00:NEXT K:NF=NF-1
```

```
850 F$((NF-1)*50+1,(NF-1)*50+LEN(X$))=X$:FX(NF)=
LEN(X$)
855 GOTO 840
860 NF=NF-1:PRINT
870 IF NF<1 THEN PRINT "YOU MUST HAVE AT LEAST O
NE! - REDO IT"
880 IF NF<1 THEN GOSUB 2100:GOTO 750
890 GOSUB 2000:PRINT "HERE'S THE LIST OF FACTORS
 YOU GAVE ME:":PRINT
900 FOR J=1 TO NF:PRINT J;". ";F$((J-1)*50+1,(J-
1)*50+FX(J)):NEXT J:PRINT
910 PRINT "   DECIDE WHICH FACTOR ON THE LIST IS
"
920 PRINT "THE MOST IMPORTANT AND INPUT ITS NUMB
ER."
930 PRINT "(TYPE 0 IF THE LIST NEEDS CHANGING.)"
:PRINT
940 INPUT A:A=INT(A):IF A=0 THEN 750
950 IF A<0 OR A>NF THEN 890
1000 GOSUB 2000:IF NF=1 THEN 1200
1010 PRINT "   NOW LET'S SUPPOSE WE HAVE A SCALE
 OF"
1020 PRINT "IMPORTANCE RANGING FROM 0-10.":PRINT

1030 PRINT "   WE'LL GIVE ";F$((A-1)*50+1,(A-1)*
50+FX(A));" A"
1040 PRINT "VALUE OF 10 SINCE ";F$((A-1)*50+1,(A
-1)*50+FX(A))
1050 PRINT "WAS RATED THE MOST IMPORTANT.":PRINT

1060 PRINT "   ON THIS SCALE, WHAT VALUE OF"
1070 PRINT "IMPORTANCE WOULD THE OTHER FACTORS H
AVE?"
1080 FOR J=1 TO NF:Q=A:IF J=Q THEN 1110
1090 PRINT :PRINT F$((J-1)*50+1,(J-1)*50+FX(J)):
INPUT XX:V(J)=XX
1100 IF V(J)<0 OR V(J)>10 THEN PRINT :PRINT "**
IMPOSSIBLE VALUE - TRY AGAIN **":PRINT :GOTO 109
0
1110 NEXT J
1200 V(A)=10:Q=0:FOR J=1 TO NF:Q=Q+V(J):NEXT J:F
OR J=1 TO NF
1210 V(J)=V(J)/Q:NEXT J:GOSUB 2000
1220 IF T<>3 THEN PRINT "   EACH ";T$
1230 IF T=3 THEN PRINT "   DECIDING YES OR NO"
1240 PRINT "MUST NOW BE COMPARED WITH RESPECT TO
"
1250 PRINT "EACH IMPORTANCE FACTOR."
1260 PRINT "   WE'LL CONSIDER EACH FACTOR"
1270 PRINT "SEPARATELY AND THEN RATE"
1280 IF T<>3 THEN PRINT "EACH ";T$;" IN TERMS"
1290 IF T=3 THEN PRINT "DECIDING YES OR DECIDING
 NO IN TERMS"
```

```
1300 PRINT "OF THAT FACTOR ONLY."
1310 PRINT "   LET'S GIVE ";L$(1,LX(1))
1320 PRINT "A VALUE OF 10 ON EVERY SCALE.":PRINT
1330 IF T<>3 THEN PRINT "   THEN EVERY OTHER ";T
$
1340 IF T=3 THEN PRINT "   THEN DECIDING NO"
1350 PRINT "WILL BE ASSIGNED A VALUE HIGHER OR"
1360 PRINT "LOWER THAN 10.  THIS VALUE DEPENDS O
N"
1370 PRINT "HOW MUCH YOU THINK IT IS BETTER OR"
1380 PRINT "WORSE THAN ";L$(1,LX(1));".":PRINT
1390 FOR J=1 TO NF
1400 GOSUB 2050:PRINT
1410 PRINT "   CONSIDERING ONLY ";F$((J-1)*50+1,
(J-1)*50+FX(J));" AND"
1420 PRINT "ASSIGNING 10 TO ";L$(1,LX(1));" ;"
1430 PRINT "WHAT VALUE WOULD YOU ASSIGN TO"
1440 PRINT :FOR K=2 TO NI
1450 PRINT L$((K-1)*50+1,(K-1)*50+LX(K));:INPUT
XX:C(K,J)=XX:PRINT :IF C(K,J)>=0 THEN 1470
1460 PRINT " - NEGATIVE VALUES NOT LEGAL":GOTO 1
450
1470 NEXT K:PRINT :C(1,J)=10:NEXT J
1500 FOR J=1 TO NF:Q=0:FOR K=1 TO NI
1510 Q=Q+C(K,J):NEXT K:FOR K=1 TO NI
1520 C(K,J)=C(K,J)/Q:NEXT K:NEXT J
1530 FOR K=1 TO NI:D(K)=0:FOR J=1 TO NF
1540 D(K)=D(K)+C(K,J)*V(J):NEXT J:NEXT K
1550 MX=0:FOR K=1 TO NI
1560 IF D(K)>MX THEN MX=D(K)
1570 NEXT K:FOR K=1 TO NI:D(K)=D(K)*100/MX:NEXT
K
1600 FOR K=1 TO NI:Z(K)=K:NEXT K:NM=NI-1
1610 FOR K=1 TO NI:FOR J=1 TO NM:N1=Z(J):N2=Z(J+
1):IF D(N1)>D(N2) THEN 1630
1620 Z(J+1)=N1:Z(J)=N2
1630 NEXT J:NEXT K:J1=Z(1):J2=Z(2):DF=D(J1)-D(J2
):GOSUB 2000
1700 PRINT L$((J1-1)*50+1,(J1-1)*50+LX(J1));
1710 PRINT " COMES OUT BEST"
1720 IF DF<5 THEN PRINT "BUT IT'S VERY CLOSE.":G
OTO 1800
1730 IF DF<10 THEN PRINT "BUT IT'S FAIRLY CLOSE.
":GOTO 1800
1740 IF DF<20 THEN PRINT "BY A FAIR AMOUNT.":GOT
O 1800
1750 PRINT "QUITE DECISIVELY."
1800 PRINT :PRINT "HERE IS THE FINAL LIST IN ORD
ER.":PRINT
1810 PRINT L$((J1-1)*50+1,(J1-1)*50+LX(J1));" HA
S BEEN"
1820 PRINT "GIVEN A VALUE OF 100 AND THE OTHERS"
```

```
1830 PRINT "RATED ACCORDINGLY.":PRINT
1840 GOSUB 2050:PRINT
1845 X$="                                     ":
REM 37 BLANK SPACES
1850 FOR J=1 TO NI:Q=Z(J):PRINT D(Q);:POSITION 1
6,PEEK(84)
1855 PRINT L$((Q-1)*50+1,(Q-1)*50+LX(Q)):NEXT J
1860 END
2000 FOR J=1 TO 50:NEXT J
2010 PRINT CHR$(125):POSITION 14,2:PRINT "D E C
I D E":PRINT :RETURN
2050 PRINT CHR$(32);:FOR K=1 TO 36:PRINT CHR$(18
);:NEXT K:PRINT :RETURN
2100 FOR J=1 TO 300:NEXT J: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 180. For example, to use the word "DONE," change line 180 to

   180 E$="DONE"

2. Line 2100 contains a timing delay used regularly in the program. If things seem to change too fast, you can make the number 300 larger. Try

   2100 FOR J=1 TO 500:NEXT J:RETURN

3. The program can currently accept up to nine decision alternatives and/or nine importance factors. If you need more, increase the dimensioning in line 160. Each numeric dimension value is one more than the number the program will actually allow. Thus, to use 14 values, line 160 should be

   160 DIM L$(750),F$(750),V(15),C(15,15),Z(15),D(15),
   T$(40),X$(40),E$(15),R$(40),LX(15),FX(15)

MAIN ROUTINES

150- 180	Initializes and dimensions variables.
200- 350	Determines category of decision.
400- 490	Gets or sets T$.
500- 710	Gets list of possible alternatives from user.

750- 950	Gets list of importance factors from user.
1000-1110	User rates each importance factor.
1200-1470	User rates the decision alternatives with respect to each importance factor.
1500-1570	Evaluates the various alternatives.
1600-1630	Sorts alternatives into their relative ranking.
1700-1860	Displays results.
2000-2010	Subroutine to clear screen and display header.
2050	Formatting subroutine.
2100	Delay 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).
X$	User input string.
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, J1, J2, NM, XX	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.
LX, FX	Work arrays

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 one million dollars or more are rejected (you can afford to hire an investment counselor if you want to borrow that much). Then 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 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. Sixteen 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 sixteen 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.

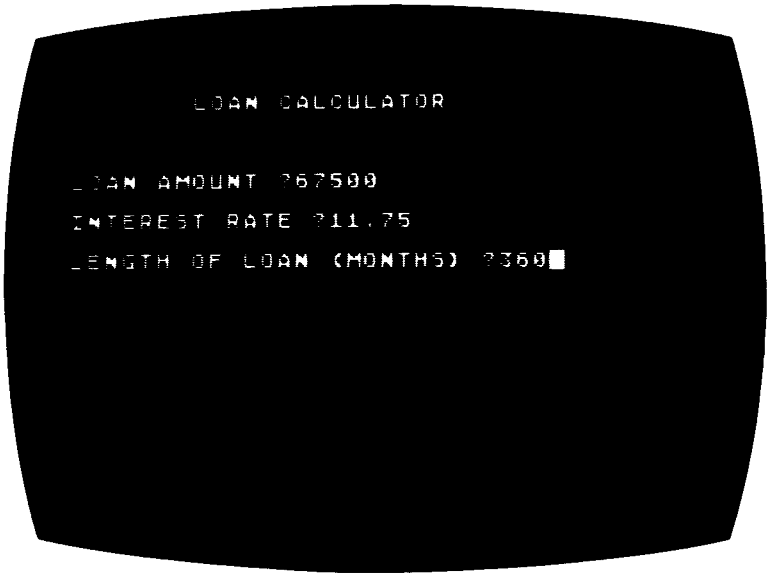
The third option is to simply start over. You will generally use this option if you are just comparing what the different monthly payments would be for different loan possibilities.

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.

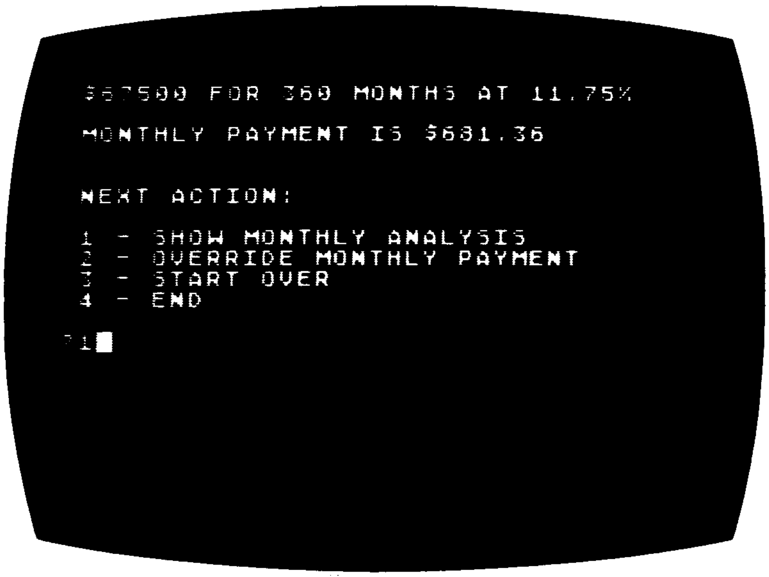
NOTE: SEE DISCLAIMER IN FRONT PART OF BOOK.

SAMPLE RUN



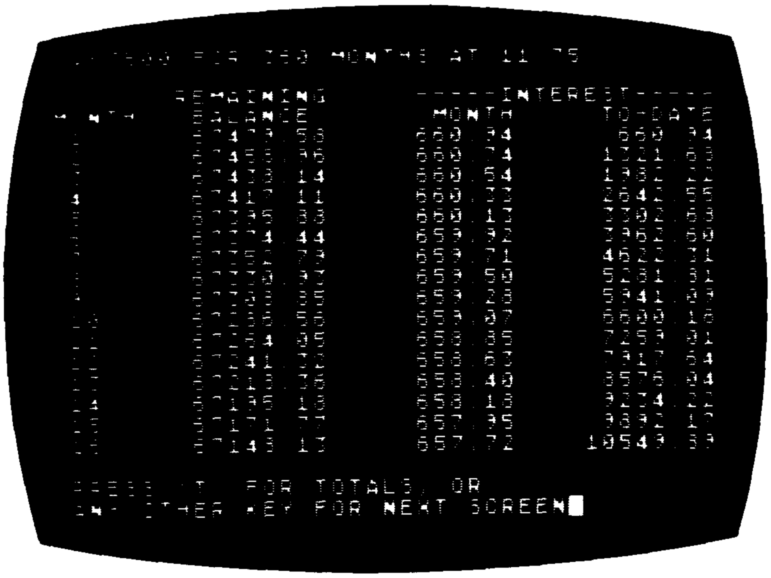
```
          LOAN CALCULATOR


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.



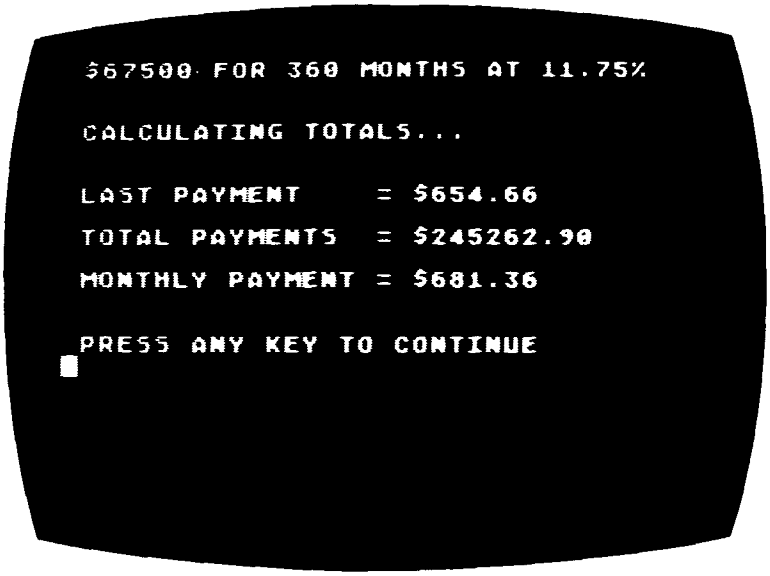
```
$67500 FOR 360 MONTHS AT 11.75%

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 asks for the monthly analysis.



```
$67500 FOR 360 MONTHS AT 11.75%

        REMAINING         -----INTEREST-----
MONTH    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
 10      67286.56         659.07      6600.16
 11      67264.05         658.85      7259.01
 12      67241.32         658.63      7917.64
 13      67218.36         658.40      8576.04
 14      67195.18         658.18      9234.22
 15      67171.77         657.95      9892.17
 16      67148.13         657.72     10549.89

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

The program responds with information about the first 16 months of the loan, then waits.



```
$67500 FOR 360 MONTHS AT 11.75%

CALCULATING TOTALS...

LAST PAYMENT     = $654.66
TOTAL PAYMENTS   = $245262.90
MONTHLY PAYMENT  = $681.36

PRESS ANY KEY TO CONTINUE
```

The operator presses "T", and after a few seconds the program displays totalling information about the loan.

PROGRAM LISTING

```
10 REM LOAN
15 REM COPYRIGHT 1984 DILITHIUM PRESS
20 OPEN #1,4,0,"K:":POKE 82,0
30 DIM BL$(12),R$(12),B$(12),T$(12),TT$(12),S$(1
2),D$(12),X$(12)
40 GRAPHICS 0
45 BL$="            ":REM 12 SPACES
50 PRINT BL$(1,8);"LOAN CALCULATOR"
60 PRINT :PRINT :PRINT
70 PRINT " LOAN AMOUNT ";:INPUT A
80 GOSUB 750:IF A=0 THEN 70
90 PRINT :PRINT " INTEREST RATE ";:INPUT R
100 PRINT :PRINT " LENGTH OF LOAN (MONTHS) ";:IN
PUT N
110 R=ABS(R):N=INT(N):M=R/1200:PRINT
120 GOSUB 630
130 W=(1+M)^N
140 P=A*M*W/(W-1)
150 P=INT(P*100+0.99):P=P/100
160 W=P*100:GOSUB 900:PRINT " MONTHLY PAYMENT IS
 $";S$
170 R$="":FP=P:PRINT :PRINT
180 PRINT " NEXT ACTION:"
190 PRINT
200 PRINT " 1 - SHOW MONTHLY ANALYSIS"
210 PRINT " 2 - OVERRIDE MONTHLY PAYMENT"
220 PRINT " 3 - START OVER"
230 PRINT " 4 - END"
240 PRINT :INPUT C
250 IF C<1 OR C>4 THEN 270
260 ON C GOTO 320,300,40,290
270 PRINT " CHOICES ARE 1, 2, 3, AND 4"
280 GOTO 180
290 POKE 82,2:END
300 PRINT :PRINT " MONTHLY PAYMENT ";:INPUT P
310 GOTO 170
320 GOSUB 330:GOTO 370
330 GOSUB 630:IF R$="T" THEN RETURN
340 POSITION 7,PEEK(84):PRINT "REMAINING";
345 POSITION 21,PEEK(84):PRINT "------INTEREST---
---"
350 PRINT "MONTH   BALANCE";
355 POSITION 22,PEEK(84):PRINT "MONTH     TO-DAT
E"
360 RETURN
370 B=A*100:TT=0:TP=TT:L=TT:P=P*100:R$=""
380 FOR J=1 TO N
390 T=M*B
400 T=INT(T+0.5)
410 IF J=N THEN P=B+T
```

```
420 TP=TP+P:B=B-P+T:TT=TT+T
430 IF B<0 THEN GOSUB 790
440 IF R$="T" THEN 560
450 W=B:GOSUB 670:B$=S$
460 W=T:GOSUB 670:T$=S$
470 W=TT:GOSUB 670:TT$=S$
480 PRINT " ";J;:POSITION 6,PEEK(84)
482 PRINT B$;:POSITION 17,PEEK(84)
484 PRINT T$;:POSITION 29,PEEK(84)
486 PRINT TT$
490 IF B=0 THEN J=N:GOTO 510
500 L=L+1:IF L<16 THEN 560
510 PRINT :PRINT " PRESS 'T' FOR TOTALS, OR"
520 PRINT " ANY OTHER KEY FOR NEXT SCREEN";
530 GET #1,XX
540 IF XX=84 THEN R$="T"
550 L=0:GOSUB 330:IF R$="T" THEN PRINT :PRINT "
CALCULATING TOTALS..."
560 NEXT J
570 W=P:GOSUB 900:PRINT :PRINT :PRINT " LAST PAY
MENT     = $";S$
580 W=TP:GOSUB 900:PRINT :PRINT " TOTAL PAYMENTS
  = $";S$
590 W=FP*100:GOSUB 900:PRINT :PRINT " MONTHLY PA
YMENT = $";S$:PRINT
600 PRINT :PRINT " PRESS ANY KEY TO CONTINUE"
610 GET #1,ZX
620 P=FP:GOTO 170
630 PRINT CHR$(125)
640 W=R*100:GOSUB 900:PRINT " $";A;" FOR ";N;" M
ONTHS AT ";S$;"%"
650 PRINT
660 RETURN
670 W=INT(W)
680 X$=S$:S$="":S$=STR$(W/100):IF (INT(W/100))*1
00=W THEN S$(LEN(S$)+1)=".00":GOTO 690
685 IF (INT(W/10))*10=W THEN S$(LEN(S$)+1)="0"
690 K=LEN(S$):IF K>9 THEN RETURN
700 X$=S$:S$="":S$(1,10-K)=BL$(1,10-K):S$(LEN(S$
)+1,10)=X$:RETURN
710 D$(1,1)=".":D$(2,3)=S$(K-2,K)
720 X$=S$:S$(1,LEN(X$(1,K-2)))=X$(1,K-2):S$(LEN(
X$(1,K-2))+1,LEN(X$(1,K-2))+LEN(D$))=D$
730 X$=S$:S$(1,10)=BL$(1,10):S$(10,9+LEN(X$))=X$
740 RETURN
750 A=ABS(A):A=INT(A)
760 IF A<1000000 THEN RETURN
770 PRINT "**** TOO LARGE ****"
780 A=0:RETURN
790 P=P+B:TP=TP+B:B=0
800 RETURN
```

```
900 W=INT(W)
910 S$=STR$(W/100):IF (INT(W/100))*100=W THEN S$
(LEN(S$)+1)=".00":RETURN
920 IF (INT(W/10))*10=W THEN S$(LEN(S$)+1)="0"
930 RETURN
```

EASY CHANGES

1. The number of lines of data that are displayed on each screen when getting a monthly analysis can be changed by altering the constant 16 in statement 500.
2. To include the monthly payment in the heading at the top of each screen of the monthly analysis, insert the following line:

645 IF FP< >0 THEN PRINT "MONTHLY PAYMENT = ";FP

MAIN ROUTINES

10–100	Defines arrays. Displays title. Gets loan information.
110–160	Calculates and displays monthly payment.
170–280	Asks for next action. Goes to corresponding routine.
300–310	Gets override for monthly payment.
320–620	Calculates and displays monthly analysis.
630–660	Subroutine to clear screen and display data about the loan at the top.
670–740	Subroutine to convert integer amount to fixed-length string with aligned decimal point.
750–780	Edits loan amount (size and whole dollar).
790–800	Subroutine to handle early payoff of loan.
900–930	Subroutine to convert integer amount to a string with two decimal places.

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$, X	Work strings.
D$, BL$	Work strings.
K, XX, ZX	Work variables.

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 operating 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 1000. 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 1000 and incrementing each new line by five or ten. This allows later insertion to correct mistakes or to add previously missing data. 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, colons, 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.)

If you do not know part of the information for a particular DATA line you can do one of two things: make your best guess for the unknown item(s) or leave the entry for the unknown item(s) null. You can leave a null entry as follows: for an unknown date, place a comma as the first nonblank character after DATA; for an unknown odometer reading, or gallons value, simply enter zero. The program will recognize these special input forms. However, *in all cases*, each DATA statement that you enter must contain *precisely* two commas.

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 mode. The operator branches to one of three available subroutines. When a subroutine completes execution, control returns to the command mode 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 9999). 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. 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), the program will display an error message and terminate. If this happens, check all your DATA statements to be sure they are in the correct format. 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 mode.

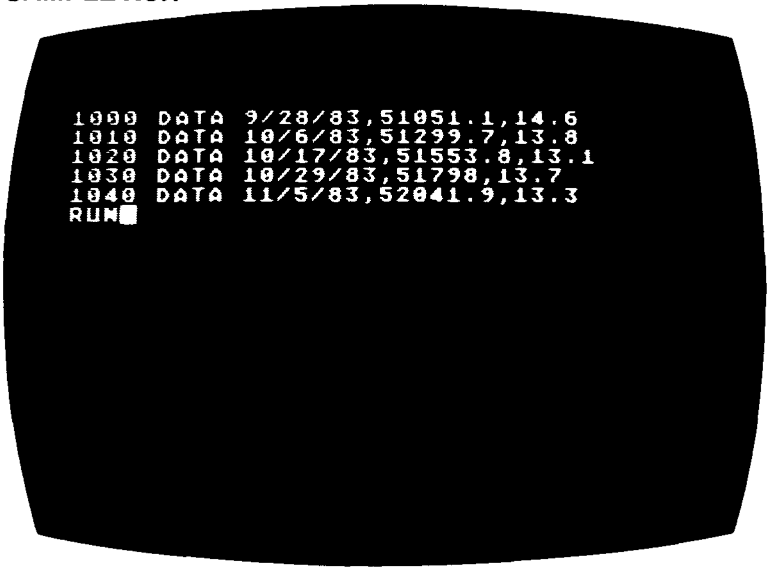
Display Mileage Information

This subroutine computes mileage (miles per gallon) from the available data. It formats all information and displays it in tabular form. Numbers are rounded to the nearest tenth 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

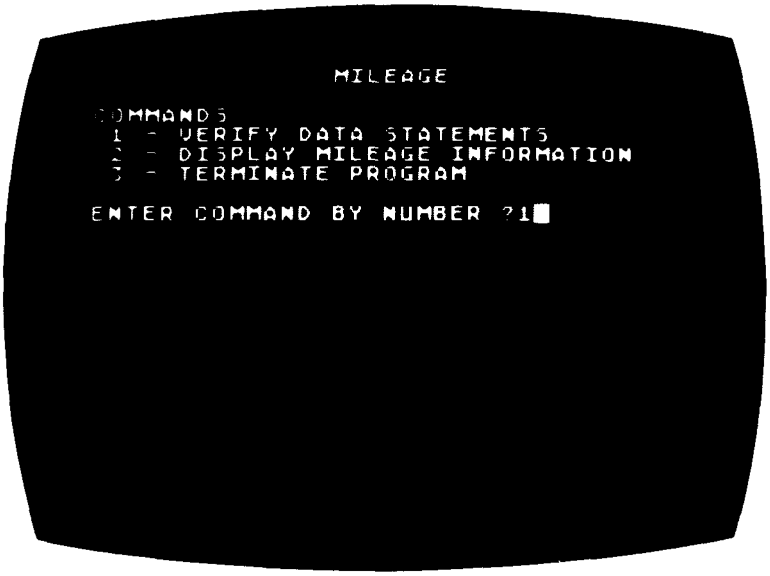
This subroutine ends program execution and returns the computer to direct BASIC.

SAMPLE RUN



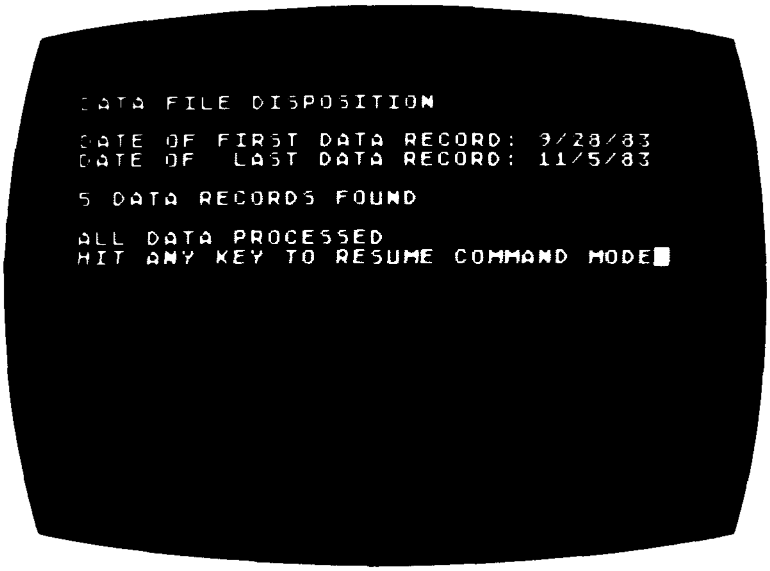
```
1000 DATA 9/28/83,51051.1,14.6
1010 DATA 10/6/83,51299.7,13.8
1020 DATA 10/17/83,51553.8,13.1
1030 DATA 10/29/83,51798,13.7
1040 DATA 11/5/83,52041.9,13.3
RUN█
```

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



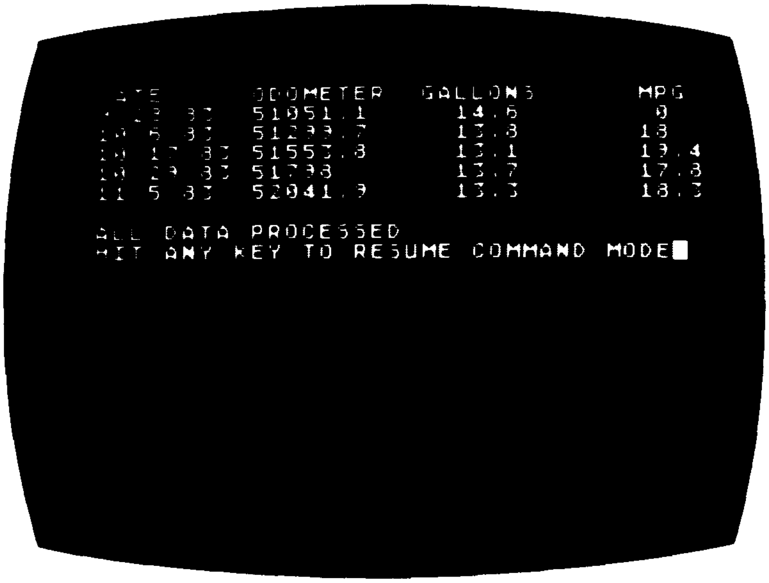
```
                    MILEAGE

COMMANDS
  1 - VERIFY DATA STATEMENTS
  2 - DISPLAY MILEAGE INFORMATION
  3 - TERMINATE PROGRAM

ENTER COMMAND BY NUMBER ?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.



```
DATA FILE DISPOSITION

DATE OF FIRST DATA RECORD: 9/28/83
DATE OF  LAST DATA RECORD: 11/5/83

5 DATA RECORDS FOUND

ALL DATA PROCESSED
HIT ANY KEY TO RESUME COMMAND MODE█
```

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



```
DATE          ODOMETER   GALLONS        MPG
9  28  83     51051.1      14.6          0
10  5  83     51299.7      13.8         18
10  17  83    51553.8      13.1         19.4
10  29  83    51798        13.7         17.8
11  5  83     52041.9      13.3         18.3

ALL DATA PROCESSED
HIT ANY KEY TO RESUME COMMAND MODE
```

Later, 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

```
10 REM MILEAGE
15 REM COPYRIGHT 1984 DILITHIUM PRESS
110 DIM A$(20),C$(1)
200 GRAPHICS 0:POKE 82,2:TRAP 500
210 PRINT :POSITION 16,PEEK(84):PRINT "MILEAGE"
220 PRINT :PRINT "COMMANDS"
230 PRINT " 1 - VERIFY DATA STATEMENTS"
240 PRINT " 2 - DISPLAY MILEAGE INFORMATION"
250 PRINT " 3 - TERMINATE PROGRAM"
260 PRINT :PRINT "ENTER COMMAND BY NUMBER ";:INP
UT C$
270 IF C$<>"1" AND C$<>"2" AND C$<>"3" THEN 200
280 C=VAL(C$):ON C GOTO 300,700,290
290 POKE 764,255:POKE 752,0:END
300 GRAPHICS 0:RESTORE
310 PRINT "DATA FILE DISPOSITION":PRINT
320 N=0:DD=-1.0E-03
330 TRAP 500
340 READ A$,D,G
350 N=N+1
360 IF N=1 THEN PRINT "DATE OF FIRST DATA RECORD
: ";A$
```

```
370 IF D>=0 AND D<=999999 THEN 400
380 PRINT "--BAD ODOMETER VALUE OF ";D
390 PRINT "  AT DATE: ";A$
400 IF D>DD THEN 440
410 PRINT "--INCONSISTENT ODOMETER VALUES"
420 PRINT "  ODO READS ";D;" AT DATE: ";A$
430 PRINT "  YET READS ";DD;" AT PREVIOUS DATE"
440 IF G>=0 AND G<=9999 THEN 470
450 PRINT "--BAD GALLONS VALUE OF ";G
460 PRINT "  DETECTED AT DATE: ";A$
470 DD=D:GOTO 330
500 LN=PEEK(187)*256+PEEK(186)
510 EC=PEEK(195)
520 IF EC=3 THEN 560
530 IF EC=6 THEN 590
540 PRINT :PRINT "--ERROR # ";EC;" HAS BEEN DETE
CTED"
550 PRINT "  AT LINE ";LN
555 PRINT :GOTO 580
560 PRINT :PRINT "--BAD DATA RECORD DETECTED"
570 PRINT "  AT OR BEFORE LINE ";LN
580 PRINT "  PROGRAM ABORTED":GOTO 290
590 IF N>0 THEN 620
600 PRINT :PRINT "--NO DATA FOUND"
610 GOTO 580
620 IF C=2 THEN 650
630 PRINT "DATE OF  LAST DATA RECORD: ";A$
640 PRINT :PRINT N;" DATA RECORDS FOUND"
650 PRINT :PRINT "ALL DATA PROCESSED"
660 PRINT "HIT ANY KEY TO RESUME COMMAND MODE";:
POKE 764,255
665 IF PEEK(764)=255 THEN 665
670 POKE 764,255:TRAP 500:GOTO 200
700 GOSUB 820:RESTORE :DD=-1:N=0
710 TRAP 500
720 READ A$,D,G:N=N+1:IF LEN(A$)>8 THEN A$=A$(1,
8)
730 R=D:GOSUB 860:D=R:LD=18-L:IF INT(D)=D THEN L
D=LD-2
740 R=G:GOSUB 910:G=R:LG=27-L:IF INT(G)=G THEN L
G=LG-2
750 IF DD<0 OR G=0 THEN M=0:GOTO 770
760 M=(D-DD)/G
770 R=M:GOSUB 960:M=R:LM=38-L:IF INT(M)=M THEN L
M=LM-2
780 DD=D:GOSUB 790:GOTO 710
790 PK=PEEK(84):PRINT A$;:POSITION LD,PK:PRINT D
;:POSITION LG,PK
795 PRINT G;:POSITION LM,PK:PRINT M
800 K=K+1:IF K<20 THEN RETURN
810 PRINT :PRINT "HIT ANY KEY TO CONTINUE";:POKE
 764,255
```

```
815 IF PEEK(764)=255 THEN 815
820 POKE 764,0:GRAPHICS 0
830 PRINT "DATE      ODOMETER  GALLONS      MPG"
840 K=0
850 RETURN
860 IF R>=0 AND R<=999999 THEN 890
870 PRINT :PRINT "ERROR IN GALLONS DATA AT DATE:
 ";A$
880 GOTO 290
890 R=R*10+0.5:R=INT(R)/10
900 L=LEN(STR$(R)):RETURN
910 IF R>=0 AND R<=9999 THEN 940
920 PRINT :PRINT "ERROR IN GALLONS DATA AT DATE:
 ";A$
930 GOTO 290
940 R=R*10+0.5:R=INT(R)/10
950 L=LEN(STR$(R)):RETURN
960 IF R<0 OR R>9999 THEN R=0
970 R=R*10+0.5:R=INT(R)/10
980 L=LEN(STR$(R)):RETURN
```

EASY CHANGES

1. If you would like to give a name to the data file and have that name print out with the command mode, change lines 110 and 210 and add line 205 as follows:

```
110 DIM A$(20),B$(20),C$(1)
205 LET B$="VOLVO 1983"
210 PRINT "MILEAGE FOR:";B$
```

Just set B$ in line 205 to whatever file name you wish to use.

2. This program uses ATARI BASIC TRAP statements to detect certain expected errors. Should an unexpected one occur, the program may abort after printing out a message like:

```
- - ERROR #8 HAS BEEN DETECTED AT LINE 340
```

These error numbers and their meanings can be found in your ATARI manual with the explanation of the TRAP statement. Should you get one and not understand what has happened, re-run the program after making the following changes in order to get normal BASIC error messages:

```
200 GRAPHICS 0:POKE 82,2
330 REM
710 REM
```

MAIN ROUTINES

110	Dimension arrays.
200- 280	Command mode. Displays available subroutines and branches to the operator's choice.
290	Subroutine to terminate execution.
300- 470	Subroutine to verify DATA statements.
500- 670	Processes errors in reading DATA statements.
700- 780	Subroutine to display mileage information.
790- 850	Subroutine to print results.
860- 900	Subroutine to round odometer values.
910- 950	Subroutine to round gallons values.
960- 980	Subroutine to round mileage values.
1000-	User-created DATA statements.

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.
C$	User reply string.
DD	Previous odometer value
D	Current odometer value.
G	Current gallons value.
M	Current MPG value.
PK	Current print row.
LD	Print position for odometer value.
LG	Print position for gallons value.
LM	Print position for mileage value.
L	Length of string.
R	Pre-rounded numbers.
K	Number of lines printed since screen cleared
EC	Error code.
LN	Line number of detected error.

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 purchased. 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.

STOP

PURPOSE AND DISCUSSION

If you are only using your ATARI for making calculations or other "normal" work, you are missing out on something. The ATARI has a very accurate internal timer, which can be very useful. This program uses it in a very obvious way—as a stopwatch. Using a computer as a stopwatch gives you the advantage of leaving the last timing on the screen for reference while you are making the next timing. Of course, the computer is "smart" enough to allow you to get "lap" times as well as the final time.

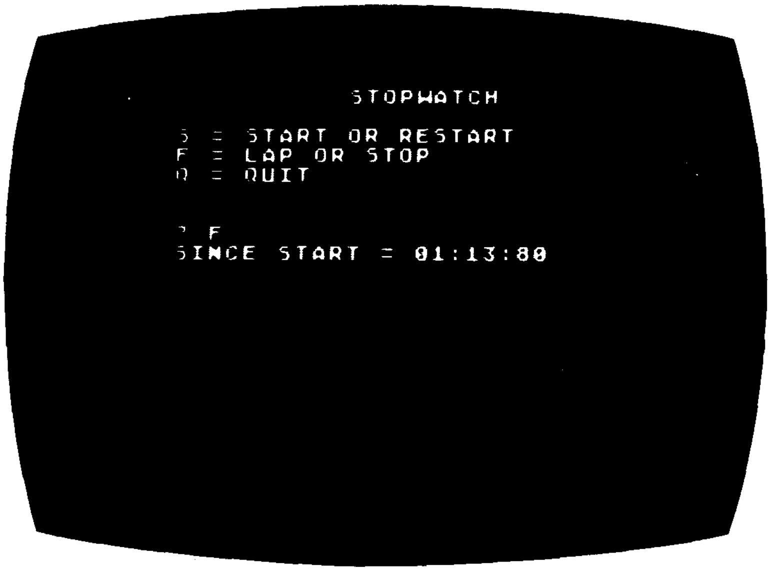
HOW TO USE IT

The opening messages from the program show you your three options. Pressing the **S** key causes the stopwatch to start (or restart, if you already started it). Pressing the **F** key causes the stopwatch program to show you the time since the **S** key was pressed. Pressing the **F** key a second time causes the program to show you both the time since the start and the time since the last **F** was pressed. This lets you see interim or "lap" times. Pressing the **Q** key terminates the program.

For example, suppose you want to time a one-mile race that is run as four laps around a quarter-mile track. Before the race begins, start the program running with the RUN command. When the starting gun is fired to start the race, press the **S** key. At the end of the first lap around the track, press the **F** key. This causes the program to show the time since the race started. When the second lap is completed, press the **F** key again. The program will show the time since the start and the time of the second lap. Press the **F** key again when the third and fourth laps are finished to get the time since start and lap times. Of course, the time since start at the end of the fourth lap is the final time of the race, even though the stopwatch keeps running. At that time, you can either press **S** to restart the stopwatch (when the next race begins) or press the **BREAK** or **Q** key to stop the program.

The internal timer of the ATARI is accurate to one sixtieth (1/60) of a second. The program displays the time as though it is accurate to 0.01 second. As a result, the second decimal place is not precise. The actual time is within plus or minus 0.02 second of the time that is shown. Also, because of the computation time required to display each timing, wait at least 0.5 second between consecutive **F** suppressions.

SAMPLE RUN



```
                STOPWATCH

S = START OR RESTART
F = LAP OR STOP
Q = QUIT

? F
SINCE START = 01:13:80
```

First the operator presses "S" to start the stopwatch. Then, after 1 minute, 13.80 seconds, he or she presses "F".

PROGRAM LISTING

```
10 REM STOP
15 REM COPYRIGHT 1984 DILITHIUM PRESS
100 REM A STOPWATCH PROGRAM
110 GRAPHICS 0
120 POKE 752,1
130 POSITION 12,2:PRINT "STOPWATCH":PRINT
140 PRINT "S = START OR RESTART"
150 PRINT "F = LAP OR STOP"
160 PRINT "Q = QUIT"
170 PRINT
180 POSITION 2,9:PRINT "? ";:POKE 764,255
190 K=PEEK(764):IF K=255 THEN 190
200 A=PEEK(20):B=PEEK(19):C=PEEK(18)
210 IF K<>62 THEN 260
220 S1=0:POKE 18,S1:POKE 19,S1:POKE 20,S1:PRINT
"S"
230 POSITION 16,10:PRINT "00:00:00"
240 POSITION 2,11:PRINT "                      "
:REM 22 SPACES
250 GOTO 180
260 IF K=56 THEN PRINT "F":GOSUB 290:GOTO 180
270 IF K=47 THEN POSITION 2,12:POKE 764,255:POKE
 752,0:END
280 GOTO 180
290 S=A+(B*256)+(C*4096)
300 POSITION 2,10:PRINT "SINCE START = ";
310 T=S:GOSUB 370
320 IF S1=0 THEN 350
330 POSITION 2,11:PRINT "SINCE  LAST = ";
340 T=S-S1:GOSUB 370
350 S1=S
360 RETURN
370 M=INT(T/3600)
380 N=INT(T/60)-60*M
390 P=INT(T/0.6)-100*(N+60*M)
400 K=M:GOSUB 440:PRINT ":";
410 K=N:GOSUB 440:PRINT ":";
420 K=P:GOSUB 440
430 RETURN
440 IF K<10 THEN PRINT CHR$(48);
450 PRINT K;:RETURN
```

EASY CHANGES

1. To allow *any* key to act like an **F** (except **S** or **Q**), change line 260 to read

```
260 IF K< >47 THEN GOSUB 290:GOTO 180
```

2. To display the time to the nearest tenth of a second (instead of the nearest hundredth), change lines 390 and 420 to:

```
390 P=INT (T/6)-10*(N+60*M)
420 PRINT P;
```

MAIN ROUTINES

10–170	Displays the title and three options.
180–190	Displays question mark and waits for a key to be pressed.
200–210	Saves time of key suppression. Checks which key was pressed.
220–280	Saves starting time and displays start clock.
290–360	Displays total and lap times.
370–450	Subroutines to convert system ticks into minutes, seconds, and hundredths and display them.

MAIN VARIABLES

S1	Time that last **S** or **F** was pressed.
S	Current time value.
K	Key pressed by operator; work variable.
A, B, C	System real time clock counter readings.
T	Time mark variable.
M, N, P	Minutes, seconds, and hundredths work variables.

SUGGESTED PROJECTS

1. Instead of displaying only the last lap time, display the last two or three lap times. Or, display *all* lap times since the start by saving each time in an array. Allow for at least twenty entries.

Section 2

Educational Programs

INTRODUCTION TO 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 ATARI 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. MATH 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 ATARI 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 ATARI 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? FLASH allows you to create your own "computer flashcards." Then you can drill yourself until you get it right.

MATH

PURPOSE

MATH 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 five 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 **RETURN** 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.

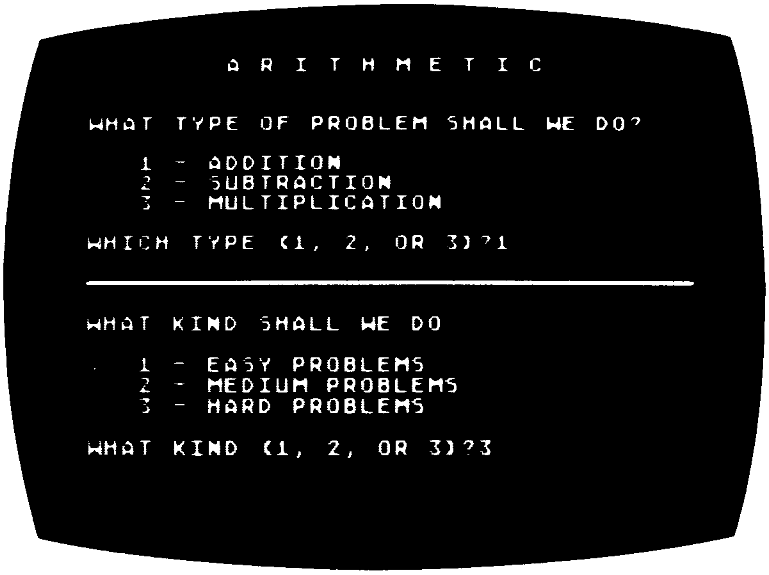
If the **RETURN** key is pressed to begin a problem, an answer of zero is assumed intended. No problems created by this program have answers of more than three digits. If a four-digit answer is given, the program will accept the answer, but then go immediately to the next problem. Answers to the problems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

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

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

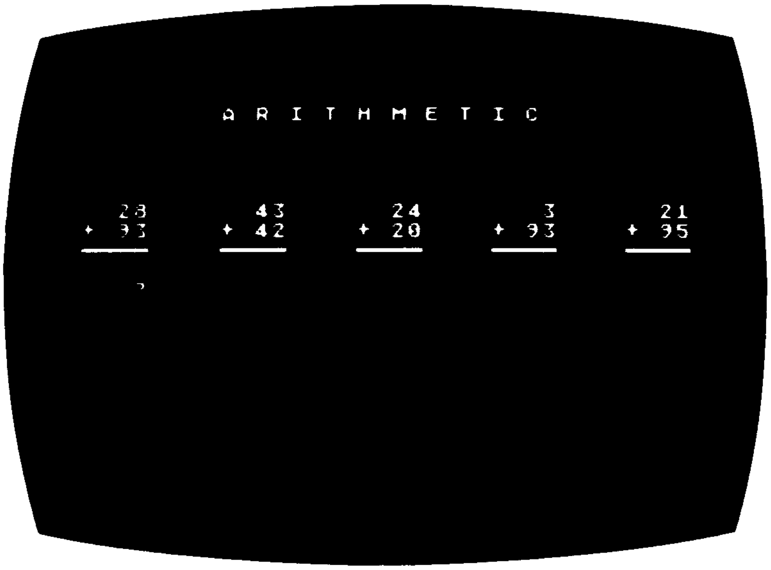
This continues until twenty problems have been worked. Before ending, the program shows what the student's performance has been. This is expressed as the number of problems solved correctly and also as the percentage of problems solved correctly.

SAMPLE RUN



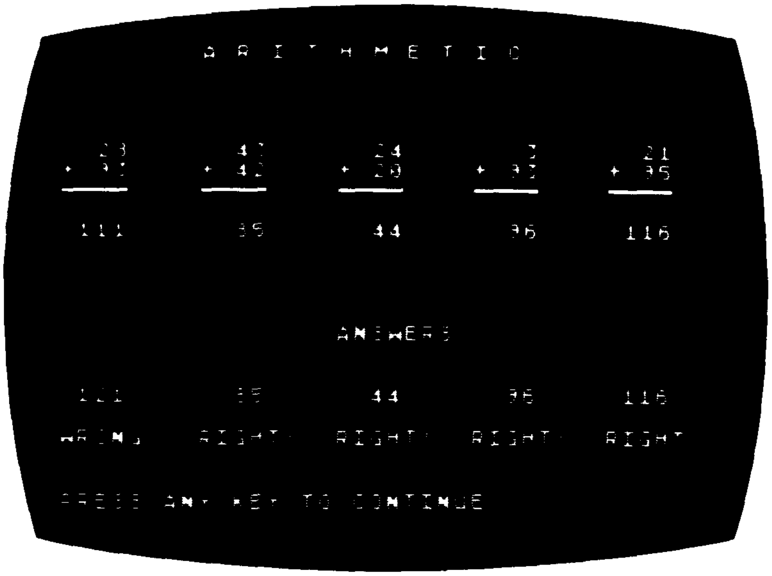
```
            A R I T H M E T I C

WHAT TYPE OF PROBLEM 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

WHAT KIND (1, 2, OR 3)?3
```

The operator chooses to do hard addition problems.



```
            A R I T H M E T I C


    28        43        24         3        21
  + 93      + 42      + 20      + 93      + 95
  ----      ----      ----      ----      ----

     ?
```

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



```
          A R I T H M E T I C



   [illegible]   [illegible]       24   [illegible]       21
 + [illegible] + [illegible]  +    20 + [illegible]  +    95
  _____          _____          _____       _____       _____

   111          85            44          96         116




                         ANSWERS

   111          85            44          96         116
 WRONG        RIGHT         RIGHT       RIGHT       RIGHT

 PRESS ANY KEY TO CONTINUE
```

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

PROGRAM LISTING

```
10 REM MATH
15 REM COPYRIGHT 1984 DILITHIUM PRESS
20 GRAPHICS 0:POKE 752,1:OPEN #1,4,0,"K:"
30 ND=0
40 DIM A(5),B(5),C(5),G(5),R$(10),P$(10)
50 NP=20
60 GOSUB 540
70 PRINT :PRINT :PRINT "WHAT TYPE OF PROBLEM SHA
LL WE DO?":PRINT
80 PRINT "   1 - ADDITION"
90 PRINT "   2 - SUBTRACTION"
100 PRINT "   3 - MULTIPLICATION"
110 PRINT :PRINT "WHICH TYPE (1, 2, OR 3)";:INPU
T R$
120 IF R$="1" OR R$="2" OR R$="3" THEN 130
125 GOTO 110
130 T=VAL(R$):PRINT :FOR J=1 TO 36:PRINT CHR$(18
);:NEXT J
140 PRINT :PRINT :PRINT "WHAT KIND SHALL WE DO"
150 PRINT :PRINT "   1 - EASY PROBLEMS"
160 PRINT "   2 - MEDIUM PROBLEMS"
170 PRINT "   3 - HARD PROBLEMS"
```

```
180 PRINT :PRINT "WHAT KIND (1, 2, OR 3)";:INPUT
 R$
190 IF R$="1" OR R$="2" OR R$="3" THEN 200
195 GOTO 180
200 D=VAL(R$)
210 ON D GOTO 220,230,250
220 GOSUB 570:GOSUB 550:GOSUB 560:GOTO 260
230 GOSUB 570:GOSUB 560:IF T=3 THEN GOSUB 590:GO
SUB 550:GOTO 260
240 IF T<>3 THEN GOSUB 580:GOSUB 550:GOTO 260
250 GOSUB 580:GOSUB 550:GOSUB 560:IF T=3 THEN GO
SUB 570:GOSUB 560
260 IF T<>2 THEN 300
270 FOR J=1 TO 5
280 IF B(J)>C(J) THEN R=C(J):C(J)=B(J):B(J)=R
290 NEXT J
300 GOSUB 600:GOSUB 540
310 FOR J=1 TO 5:X=(J-1)*8+2:Y=6:GOSUB 640:NEXT
J
320 FOR K=1 TO 5:X=(K-1)*8+5:Y=10:GOSUB 440:G(K)
=N:NEXT K
330 X=17:Y=15:GOSUB 720:PRINT " ANSWERS"
340 FOR J=1 TO 5:X=(J-1)*8+2:Y=18:GOSUB 760:NEXT
 J
350 Y=20:FOR J=1 TO 5:X=(J-1)*8+2:GOSUB 720
360 IF A(J)<>G(J) THEN PRINT "WRONG!":GOTO 380
370 PRINT "RIGHT!":NR=NR+1
380 NEXT J
390 PRINT :PRINT "PRESS ANY KEY TO CONTINUE";:CL
OSE #1:OPEN #1,4,0,"K:":GET #1,RX
400 ND=ND+5
410 IF ND<NP THEN GOSUB 540:GOTO 210
420 GOSUB 800
430 POKE 752,0:END
440 N=0:M=1
450 P$="?":GOSUB 530
460 GET #1,A
470 IF A=155 AND M=1 THEN P$="0":GOSUB 530:RETUR
N
480 IF A=155 THEN P$=CHR$(32):GOSUB 530:RETURN
490 IF A<48 OR A>57 THEN 460
500 V=A-48:P$=CHR$(A):GOSUB 530:N=N+M*V:M=M*10
510 IF M>1000 THEN RETURN
520 X=X-1:GOTO 450
530 POSITION X,Y:PRINT P$;:POSITION ZZ,Y:RETURN
540 PRINT CHR$(125):POSITION 10,PEEK(84):PRINT "
A R I T H M E T I C":RETURN
550 FOR K=1 TO 5:C(K)=INT(RND(1)*(H-L+1))+L:NEXT
 K:RETURN
560 FOR K=1 TO 5:B(K)=INT(RND(1)*(H-L+1))+L:NEXT
 K:RETURN
```

```
570 H=9:L=0:RETURN
580 H=99:L=0:RETURN
590 H=25:L=1:RETURN
600 ON T GOTO 610,620,630
610 FOR J=1 TO 5:A(J)=B(J)+C(J):NEXT J:RETURN
620 FOR J=1 TO 5:A(J)=C(J)-B(J):NEXT J:RETURN
630 FOR J=1 TO 5:A(J)=C(J)*B(J):NEXT J:RETURN
640 GOSUB 720:Y=Y+1:PRINT CHR$(32);CHR$(32);
650 IF C(J)<10 THEN PRINT CHR$(32);
660 PRINT C(J):GOSUB 720:Y=Y+1:IF T=1 THEN PRINT
 "+";
670 IF T=2 THEN PRINT "-";
680 IF T=3 THEN PRINT "X";
690 PRINT CHR$(32);:IF B(J)<10 THEN PRINT CHR$(3
2);
700 PRINT B(J):GOSUB 720:FOR K=1 TO 4:PRINT CHR$
(18);
710 NEXT K:RETURN
720 POSITION X,Y
730 RETURN
760 GOSUB 720:IF A(J)<10 THEN PRINT CHR$(32);CHR
$(32);CHR$(32);:GOTO 790
770 IF A(J)<100 THEN PRINT CHR$(32);CHR$(32);:GO
TO 790
780 IF A(J)<1000 THEN PRINT CHR$(32);
790 PRINT A(J):RETURN
800 GOSUB 540:PRINT :PRINT
810 PRINT "YOU GOT ";NR;" RIGHT"
820 PRINT "OUT OF ";NP;" PROBLEMS"
830 P=NR/NP*100
840 PRINT :PRINT "THAT'S ";P;" PERCENT CORRECT":
RETURN
```

EASY CHANGES

1. The program currently does twenty problems per session. You can change this number by altering the variable NP in line 50. For example,

   50 NP=10

   will cause the program to do only ten problems per session. The value of NP should be kept a positive multiple of five.
2. Zero is currently allowed as a possible problem operand. If you do not wish to allow this, change lines 570 and 580 to read as follows:

   570 H=9:L=1:RETURN
   580 H=99:L=1:RETURN

MAIN ROUTINES

20- 60	Initializes constants, displays header.
70–195	Asks operator for type of problems desired.
200–300	Sets A, B, C arrays, clears screen.
310–430	Mainline routine–displays problems, gets operator's answers, displays correct answers and user's performance.
440–520	Subroutine to get and display user's answers.
530	Character-printing subroutine.
540	Subroutine to clear screen and display title.
550–560	Subroutine to set B, C arrays.
570–590	Subroutine to set L, H.
600–630	Subroutine to calculate A array from B, C arrays.
640–710	Subroutine to display problems.
720–730	Subroutine to move cursor to screen position X, Y.
760–790	Subroutine to display the correct answers.
800–840	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.
M	Answer column being worked on.
R$	Operator's input character.
V	Value of A.
A	Input character.
X, Y	Horizontal, vertical screen position of cursor.
J, K	Loop indices and work variables.
P	Percentage of correct answers.
P$	Character to be printed.

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 program 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 200–300 and 570–590. 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.

FLASH

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 on cassette tape or disk for later review.

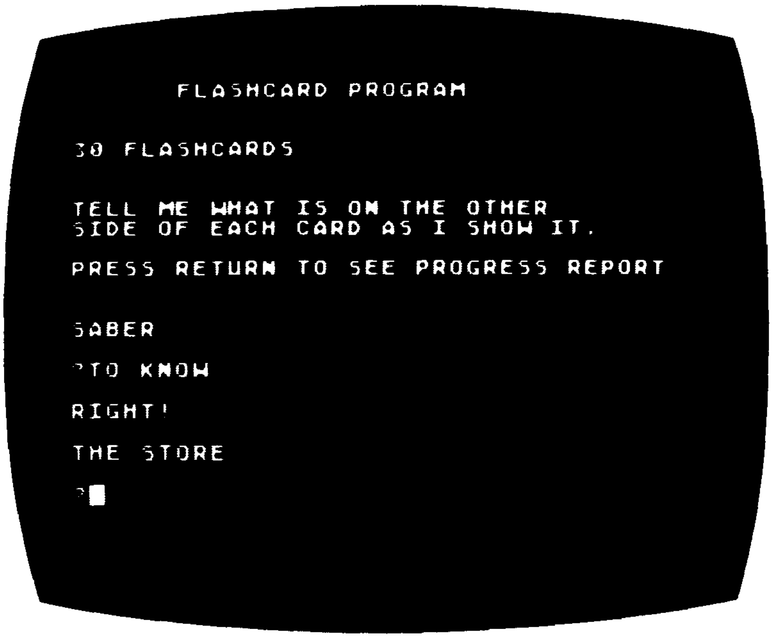
HOW TO USE IT

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 **RETURN** 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.

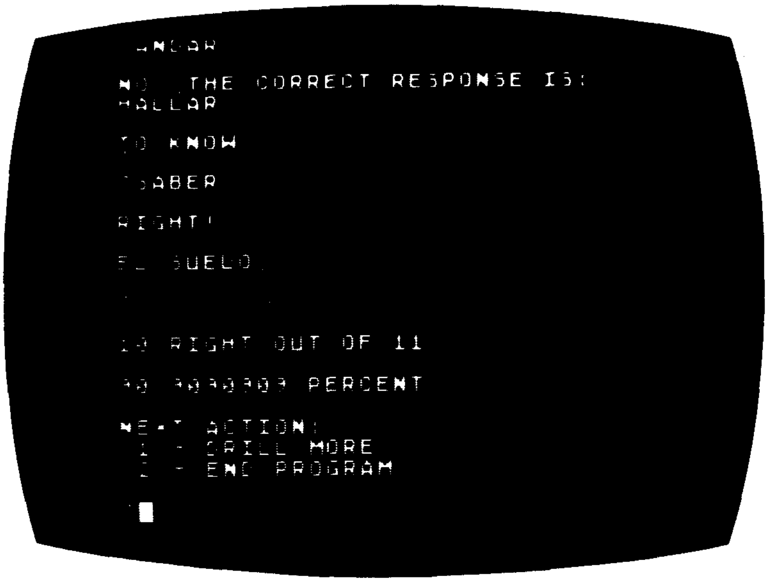
During the drill, the program will not repeat a card that was used in the previous four questions.

SAMPLE RUN



```
         FLASHCARD PROGRAM

30 FLASHCARDS

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

PRESS RETURN TO SEE PROGRESS REPORT

SABER

?TO KNOW

RIGHT!

THE STORE

?
```

In its present form, the program begins by offering 30 flashcards. The operator enters a response and pushes RETURN. This continues until the operator requests a progress report.



```
ANDAR

NO, THE CORRECT RESPONSE IS:
HALLAR

TO KNOW

SABER

RIGHT!

EL SUELO



10 RIGHT OUT OF 11

90.9090909 PERCENT

NEXT ACTION:
 1 - DRILL MORE
 2 - END PROGRAM

?
```

Later the operator ends the drill and the program shows the number and percentage of correctly answered questions. The options for continuing are displayed and the program waits for the operator's choice.

PROGRAM LISTING

```
10 REM FLASH
15 REM COPYRIGHT 1984 DILITHIUM PRESS
20 GRAPHICS 0:POKE 82,2
30 L=50:M=5
40 DIM F$(40),B$(40),P(M-1),X(L),C$(40),P$(40),R
$(40)
50 FOR J=0 TO M-1:P(J)=0:NEXT J:POSITION 8,2:PRI
NT "FLASHCARD PROGRAM"
60 PRINT :PRINT
70 K=1:C=0:W=0
80 IF K>L THEN 120
90 READ Y:IF Y=-1 THEN 130
100 X(K)=Y
110 K=K+1:GOTO 80
120 PRINT "** FLASHCARD ARRAY FULL **":PRINT
130 K=K-1
140 IF K<M THEN PRINT "** NOT ENOUGH FLASHCARDS
**":END
150 PRINT K;" FLASHCARDS"
160 PRINT :PRINT
180 PRINT "TELL ME WHAT IS ON THE OTHER"
190 PRINT "SIDE OF EACH CARD AS I SHOW IT."
195 PRINT :PRINT "PRESS RETURN TO SEE PROGRESS R
EPORT"
```

```
200 PRINT :PRINT
210 R=INT(K*RND(1))+1
220 FOR J=0 TO M-2
230 IF P(J)=R THEN 210
240 NEXT J
250 RESTORE X(R):READ F$:READ B$
260 J=RND(1):IF J>0.5 THEN 290
270 PRINT F$:C$=B$
280 GOTO 300
290 PRINT B$:C$=F$
300 PRINT :INPUT R$
310 IF LEN(R$)=0 THEN 440
320 PRINT
330 IF R$=C$ THEN 370
340 PRINT "NO, THE CORRECT RESPONSE IS:"
350 PRINT C$
360 W=W+1:GOTO 390
370 PRINT "RIGHT!"
380 C=C+1
390 FOR J=1 TO M-2
400 P(J-1)=P(J):NEXT J
410 P(M-2)=R
420 PRINT
430 GOTO 210
440 PRINT
450 IF C+W=0 THEN 500
460 PRINT C;" RIGHT OUT OF ";C+W
470 PRINT
480 PRINT C*100/(C+W);" PERCENT"
490 PRINT
500 PRINT "NEXT ACTION:"
510 PRINT " 1 - DRILL MORE"
520 PRINT " 2 - END PROGRAM"
530 PRINT
540 INPUT R$
550 IF R$="2" THEN END
560 IF R$="1" THEN 160
570 PRINT "** ENTER 1 OR 2 PLEASE **"
580 PRINT :GOTO 500
590 DATA 610,620,630,640,650,660,670,680,690,700
,710,720,730,740,750,760,770,780,790,800,810,820
,830,840
600 DATA 850,860,870,880,890,900,-1
610 DATA THE PEN,LA PLUMA
620 DATA THE DOOR,LA PUERTA
630 DATA THE SCHOOL,LA ESCUELA
640 DATA THE FLOOR,EL SUELO
650 DATA THE STORE,LA TIENDA
660 DATA THE HAND,LA MANO
670 DATA THE HOUSE,LA CASA
680 DATA THE FRIEND,EL AMIGO
690 DATA THE DINNER,LA COMIDA
700 DATA THE CHAIR,LA SILLA
```

```
710 DATA TO ARRIVE,LLEGAR
720 DATA TO ASK,PREGUNTAR
730 DATA TO BUY,COMPRAR
740 DATA TO BRING,LLEVAR
750 DATA TO COME,VENIR
760 DATA TO EAT,COMER
770 DATA TO FIND,HALLAR
780 DATA TO GO,ANDAR
790 DATA TO HAVE,TENER
800 DATA TO KNOW,SABER
810 DATA BLUE,AZUL
820 DATA GREEN,VERDE
830 DATA RED,ROJO
840 DATA WHITE,BLANCO
850 DATA YELLOW,AMARILLO
860 DATA ENOUGH,BASTANTE
870 DATA FAR,LEJOS
880 DATA FEW,POCOS
890 DATA MANY,MUCHOS
900 DATA NEAR,CERCA
```

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. The DATA statements are to be entered as lines 610 through 900 (or higher, if necessary). The earlier DATA statements (590 and 600) must contain a list of all the later DATA statements, with a value of negative one as the last number. If necessary, more than these two lines can be used, but be sure the line numbers used are all in the range of 590 to 609.
2. Change the limits of the number of flashcards that can be entered by altering line 30. L is the upper limit and M is the minimum. The current upper limit of 50 will fit in an ATARI with 4000 bytes free (see Appendix 1) if each side of each flashcard averages no more than about twelve to fifteen characters in length. If you have 16,000 bytes free, 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. Make sure each side of each flashcard is no more than 40 characters long.
3. To cause the program to always display side one of the flashcards (and ask you to respond with side two), change this line:

```
260 REM
```

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

```
260 GOTO 290
```

MAIN ROUTINES

10- 70	Initializes variables. Creates arrays. Displays title.
80–150	Reads flashcards from DATA statements.
160–490	Drills operator on flashcards in memory.
500–580	Displays options and analyzes response. Branches to appropriate routine.
590–600	DATA statements defining DATA statements containing flashcards.
610–900	DATA statements of the flashcards.

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.
X	Array containing line number of flashcards.
F$	Array containing front side of flashcard (side 1).
B$	Array containing back side of flashcard (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.
Y	Flashcard input index.

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:

```
310 REM
385 IF C=20 THEN 440
495 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.

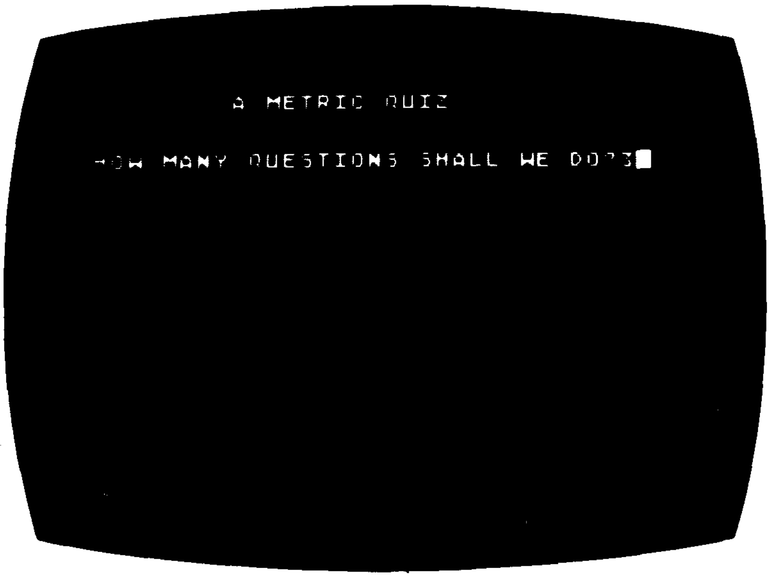
The sample run shows how each question is formulated. A quantity in English units is compared with one in metric units. Either one may appear first in the question. Each quantity will have an integral value. The relating word ("longer," "hotter," "heavier," etc.) indicates what type of quantities are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

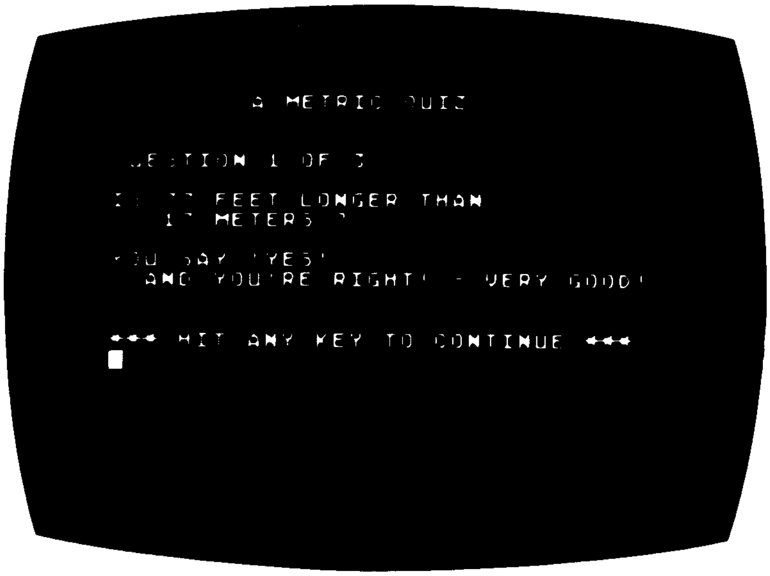
The program will continue generating the requested number of questions. Before ending, it will show you how many correct answers you gave and your percentage correct.

SAMPLE RUN



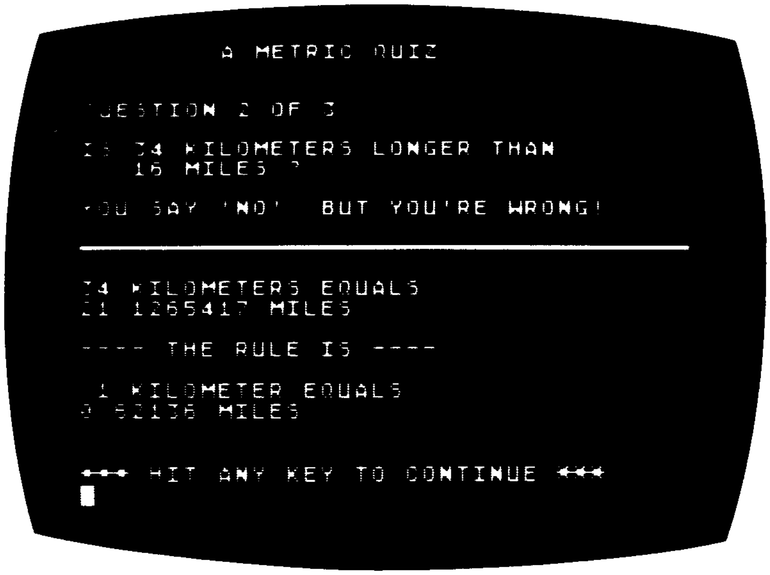
```
          A METRIC QUIZ

HOW MANY QUESTIONS SHALL WE DO? 3
```

The operator requests a three question quiz.



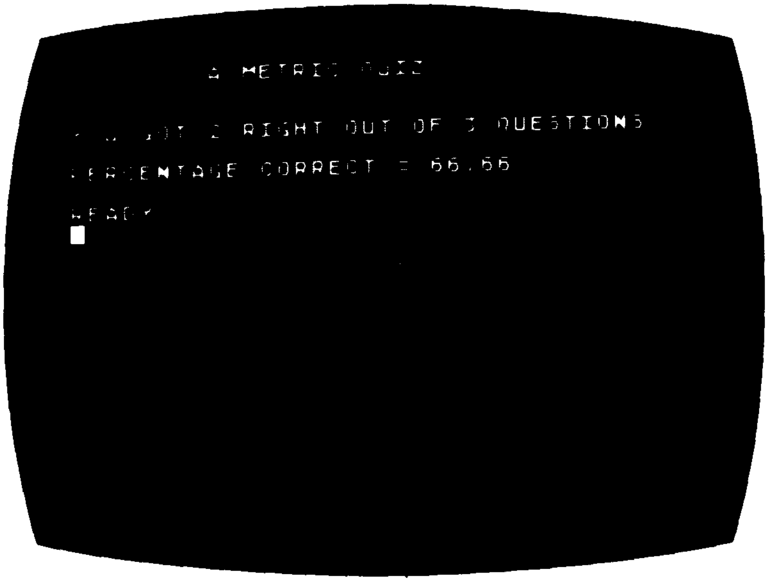
```
          A METRIC QUIZ

  QUESTION 1 OF 3

IS 77 FEET LONGER THAN
   17 METERS ?

YOU SAY 'YES'
  AND YOU'RE RIGHT! - VERY GOOD!

*** HIT ANY KEY TO CONTINUE ***
```

The first question is correctly answered "yes". The program waits for a key to be hit before continuing the quiz.



```
          A METRIC QUIZ

QUESTION 2 OF 3

IS 34 KILOMETERS LONGER THAN
   16 MILES ?

YOU SAY 'NO'  BUT YOU'RE WRONG!

34 KILOMETERS EQUALS
21.1265417 MILES

---- THE RULE IS ----

 1 KILOMETER EQUALS
0.62136 MILES

*** HIT ANY KEY TO CONTINUE ***
```

The second question is incorrectly answered "no". The correct conversion and governing rule are then displayed.



```
        A METRIC QUIZ

YOU GOT 2 RIGHT OUT OF 3 QUESTIONS

PERCENTAGE CORRECT = 66.66

READY
```

When all three questions have been answered, the program shows the number and percentage of correctly answered questions.

PROGRAM LISTING

```
10 REM METRIC
15 REM COPYRIGHT 1984 DILITHIUM PRESS
130 GRAPHICS 0:POKE 82,2
150 DIM ES$(30),MS$(30),R$(30),EP$(30),MP$(30),X
(30),B$(10)
160 B$=CHR$(32)
165 TRAP 880
170 CLOSE #1:OPEN #1,4,0,"K:"
200 GOSUB 400:GOSUB 450
210 PRINT "HOW MANY QUESTIONS SHALL WE DO";:INPU
T NQ:NQ=INT(NQ):IF NQ<1 THEN 210
220 FOR J=1 TO NQ:GOSUB 600:GOSUB 900:NEXT J
230 GOSUB 450:PRINT "YOU GOT ";NR;" RIGHT OUT OF
 ";NQ;" QUESTIONS":PRINT
240 P=(INT(10000*NR/NQ))/100:PRINT "PERCENTAGE C
ORRECT = ";P
250 END
400 RESTORE :ND=0
410 ND=ND+1:READ ZZ
420 IF ZZ=-1 THEN ND=ND-1:RETURN
430 X(ND)=ZZ:GOTO 410
450 PRINT CHR$(125):POSITION 10,PEEK(84):PRINT "
A METRIC QUIZ"
460 PRINT :PRINT :RETURN
```

```
600 N=INT(ND*RND(1))+1
605 RESTORE X(N):READ ES$,MS$,R$,C,EP$,MP$
610 F=0:IF RND(1)>0.5 THEN F=1
620 V1=INT(RND(1)*99)+2:V3=V1*C:IF F=1 THEN V3=V
1/C
630 IF N=1 THEN V3=(V1-32)/1.8:IF F=1 THEN V3=(V
1*1.8)+32
640 V2=V3*(0.5+RND(1)):V2=INT(V2+0.5):T=0:IF V2<
V3 THEN T=1
650 GOSUB 450:PRINT "QUESTION ";J;" OF ";NQ:PRIN
T
660 IF F=0 THEN PRINT "IS ";V1;B$;EP$;B$;R$;" TH
AN":PRINT B$;B$;B$;V2;B$;MP$;B$;"?";
670 IF F=1 THEN PRINT "IS ";V1;B$;MP$;B$;R$;" TH
AN":PRINT B$;B$;B$;V2;B$;EP$;B$;"?";
680 GET #1,QQ
700 IF QQ=89 THEN PRINT :PRINT :PRINT "YOU SAY '
YES' ";:R=1:GOTO 730
710 IF QQ=78 THEN PRINT :PRINT :PRINT "YOU SAY '
NO' ";:R=0:GOTO 730
720 PRINT :PRINT :PRINT "YOU HAVE NO IDEA":R=2
730 X=T-R:IF R=2 THEN GOSUB 800:GOTO 760
740 IF X=0 THEN PRINT :PRINT "  AND YOU'RE RIGHT
! - VERY GOOD!":NR=NR+1:GOTO 760
750 PRINT " BUT YOU'RE WRONG!":GOSUB 800
760 RETURN
800 PRINT :FOR K=1 TO 36:PRINT CHR$(18);:NEXT K:
PRINT :PRINT
810 IF F=0 THEN PRINT V1;B$;EP$;" EQUALS":PRINT
V3;B$;MP$
820 IF F=1 THEN PRINT V1;B$;MP$;" EQUALS":PRINT
V3;B$;EP$
830 PRINT :PRINT "---- THE RULE IS ----":PRINT
840 IF N=1 AND F=0 THEN PRINT " DEG.C = (DEG.F -
 32)/1.8":RETURN
850 IF N=1 AND F=1 THEN PRINT " DEG.F = (DEG.C *
 1.8) + 32":RETURN
860 IF F=0 THEN PRINT " 1 ";ES$;" EQUALS":PRINT
C;B$;MP$:RETURN
870 Q=INT(100000/C)/100000:PRINT " 1 ";MS$;" EQU
ALS":PRINT Q;B$;EP$:RETURN
880 IF PEEK(195)=8 THEN PRINT :PRINT "** ILLEGAL
 ENTRY **":PRINT :TRAP 880:GOTO 210
890 END
900 PRINT :PRINT :PRINT "*** HIT ANY KEY TO CONT
INUE ***"
910 GET #1,QQ
920 RETURN
950 DATA 1000,1020,1040,1060,1080,1100,1120,-1
1000 DATA DEGREE FAHRENHEIT,DEGREE CENTIGRADE,HO
TTER,0.5
1010 DATA DEGREES FAHRENHEIT,DEGREES CENTIGRADE
```

```
1020 DATA MILE PER HOUR,KILOMETER PER HOUR,FASTE
R,1.60935
1030 DATA MILES PER HOUR,KILOMETERS PER HOUR
1040 DATA FOOT,METER,LONGER,0.3048
1050 DATA FEET,METERS
1060 DATA MILE,KILOMETER,LONGER,1.60935
1070 DATA MILES,KILOMETERS
1080 DATA INCH,CENTIMETER,LONGER,2.54
1090 DATA INCHES,CENTIMETERS
1100 DATA GALLON,LITRE,MORE,3.78533
1110 DATA GALLONS,LITRES
1120 DATA POUND,KILOGRAM,HEAVIER,0.45359
1130 DATA POUNDS,KILOGRAMS
```

EASY CHANGES

1. To have the program always ask a fixed number of questions, change line 210 to set NQ to the desired value. For example:

   210 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	liters
7	weight	pounds	kilograms

   If you wish to be quizzed on only one type of question, set N to this value in line 600. Thus,

   600 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

   610 F=0

   while to go from metric to English use

   610 F=1

4. You might want the converted value and governing rule to be displayed even when the correct answer is given. This is accomplished by changing line 740 and adding line 745 as follows:

```
740 IF X=0 THEN PRINT:PRINT "AND YOU'RE
    RIGHT-VERY GOOD"
745 IF X=0 THEN NR=NR+1:GOSUB 800:GOTO 760
```

MAIN ROUTINES

130– 170	Dimensions and initializes variables.
200– 250	Mainline routine, drives other routines.
400– 430	Reads and initializes data.
450– 460	Displays header.
600– 760	Forms and asks questions. Processes user's reply.
800– 870	Displays exact conversion and governing rule.
880– 890	Error trap routine for input errors.
900– 920	Waits for user to hit any key.
950	DATA statement defining line numbers of other DATA statements.
1000–1130	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	Conversion factor.
Q, QQ, ZZ	Work variables.
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).

K	Loop index.
R	User reply flag (0=no; 1=yes; 2=no idea).
X	User's result (0 if correct answer was given).

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 variables in this format. 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.) If new DATA statements are added, the DATA statement at line 950 must be modified to include the new line numbers.
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 of 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

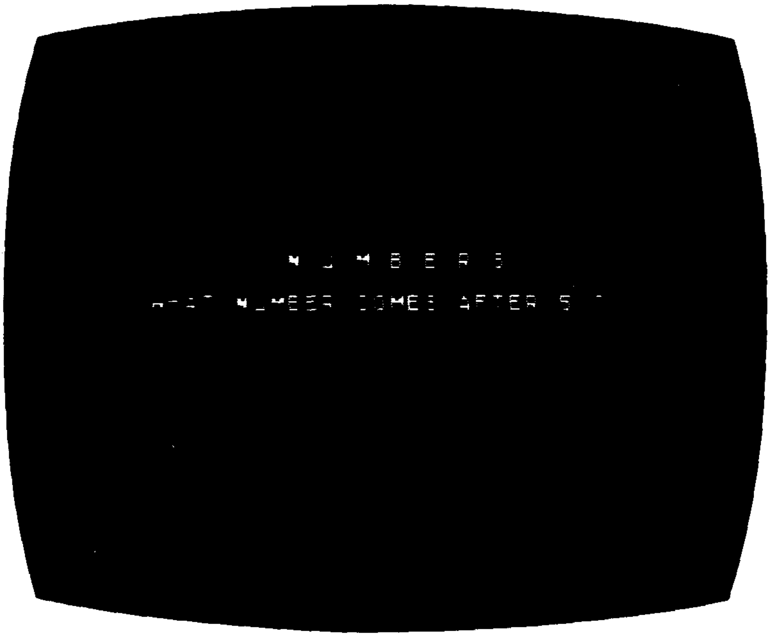
We know a child who learned how to type CLOAD and RUN to get this program started before she turned three, but you'll probably have to help your child with this for a while. The program asks the question, "WHAT NUMBER COMES AFTER n?", where n is a number from one to nine. Even if the child can't read 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 **RETURN** 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 square is drawn. In the center, a triangle is drawn. Then an asterisk (or a snowflake, perhaps?) is drawn in the lower right portion of the screen. After about a five-second delay, the program clears the screen and asks another question. The same number is never asked twice in a row. The size of the three figures is chosen at random each time.

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

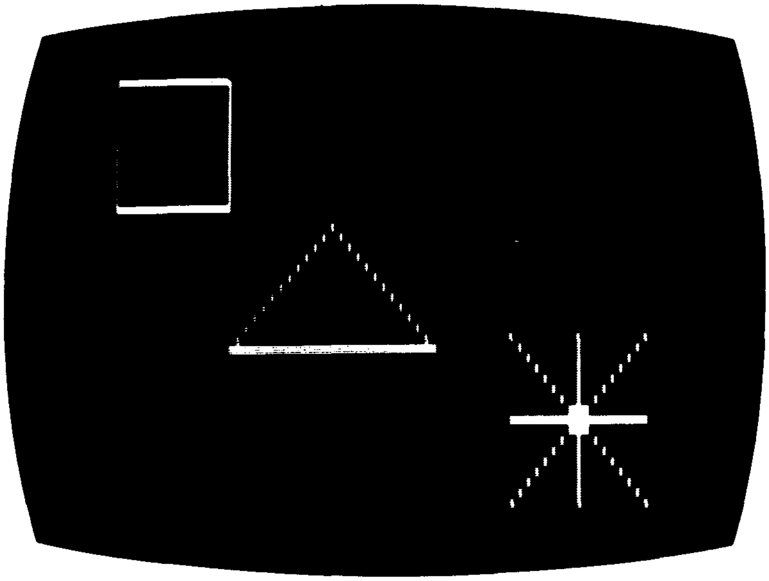
The program keeps on going until you enter "E" (for end) instead of a number. Remember that most children have a pretty short attention span, so please do not force your child to continue after his or her interest diminishes. Keep each session short and fun. This way, it will always be a treat to "play"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

SAMPLE RUN



```
              N U M B E R S

      WHAT NUMBER COMES AFTER 5 ?
```

The program asks what number comes after 5 and waits for a response.



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

PROGRAM LISTING

```
10 REM NUMBERS
15 REM COPYRIGHT 1984 DILITHIUM PRESS
20 DIM R$(3)
30 GRAPHICS 3:SETCOLOR 1,12,6:SETCOLOR 2,0,0
40 M=9:TX=25:TY=15:A=55:B=35
50 POKE 752,1:PRINT CHR$(125)
60 PRINT
70 PRINT "          N U M B E R S"
80 R=INT(M*RND(1))+1:IF R=P THEN 80
90 PRINT
100 PRINT "WHAT NUMBER COMES AFTER ";R;" ";
110 INPUT R$
115 IF LEN(R$)=0 THEN 100
120 PRINT :IF R$="E" THEN GRAPHICS 0:END
125 IF ASC(R$)<48 OR ASC(R$)>57 THEN 140
130 IF VAL(R$)=R+1 THEN 160
140 PRINT "NO, THAT'S NOT IT. TRY AGAIN."
150 GOTO 90
160 PRINT "THAT'S RIGHT!"
170 FOR X=1 TO 1000:NEXT X
180 P=R:PRINT CHR$(125)
190 C1=INT(16*RND(0)):C2=INT(16*RND(0))
200 C3=INT(16*RND(0)):IF C1=C2 THEN C2=INT((C2+4
)/4)
```

```
205 IF C1=C3 THEN C3=INT((C3+2)/2)
207 IF C2=C3 THEN C2=INT((C2+3)/3)
210 GRAPHICS 5+16
220 SETCOLOR 0,C1,10
230 SETCOLOR 1,C2,10
240 SETCOLOR 2,C3,10
245 E=INT(10*RND(0))+5
250 COLOR 1:PLOT 1,1:DRAWTO E,1:DRAWTO E,E:DRAWT
O 1,E:DRAWTO 1,1
255 E=INT(10*RND(0))+5
260 COLOR 2:FOR J=1 TO E
270 Y=TY+J:X=TX+J:PLOT X,Y:NEXT J
280 FOR J=1 TO E:Y=TY+J:X=TX-J+2:PLOT X,Y:NEXT J
290 Y=TY+E+1:FOR X=TX-E+1 TO TX+E+1:PLOT X,Y:NEX
T X
300 COLOR 3
305 E=INT(8*RND(0))+5
310 FOR J=1 TO E
320 X=A+J:Y=B+J:PLOT X,Y
330 Y=B-J:PLOT X,Y
340 Y=B:PLOT X,Y
350 X=A:PLOT X,Y
360 Y=B+J:PLOT X,Y
370 Y=B-J:PLOT X,Y
380 X=A-J:PLOT X,Y
390 Y=B:PLOT X,Y
400 Y=B+J:PLOT X,Y
410 NEXT J
420 FOR J=1 TO 1500:NEXT J
430 GRAPHICS 3:SETCOLOR 1,12,6:SETCOLOR 2,0,0
440 GOTO 80
```

EASY CHANGES

1. Change the range of numbers that the program asks by altering the value of M in line 40. 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 1000 in statement 170. Double it to double the time delay, etc. The same can be done with the 1500 in line 420 to alter the delay after the figures are drawn.
3. To avoid randomness in the size of the figures that are drawn, replace line 245 with

   ```
   245 E=11
   ```

   Instead of 11, you can use any integer from 3 to 12. Also change lines 255 and 305 to

   ```
   255 REM
   305 REM
   ```

MAIN ROUTINES

10– 70	Initializes variables. Clears screen.
80	Picks random integer from 1 to M.
90–160	Asks question. Gets answer. Determines if right or wrong.
170	Delays about 3 seconds.
180–250	Determines colors and draws a square.
255–290	Draws a triangle.
300–410	Draws an asterisk.
420	Delays about 5 seconds.
430– 440	Clears screen. Goes back to ask next question.

MAIN VARIABLES

C1, C2, C3	Color control 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; also work variables.
TX, TY	Triangle's reference locations.
A, B	X,Y coordinate values.
J	Loop index.

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 **BREAK** key or shift keys, of course). If you press the **ESCAPE** key, the program ends. After you press a key, the screen is blanked out except for two horizontal dash lines in the upper left-hand corner. After approximately two seconds, a phrase is flashed on the screen between the two lines. 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).

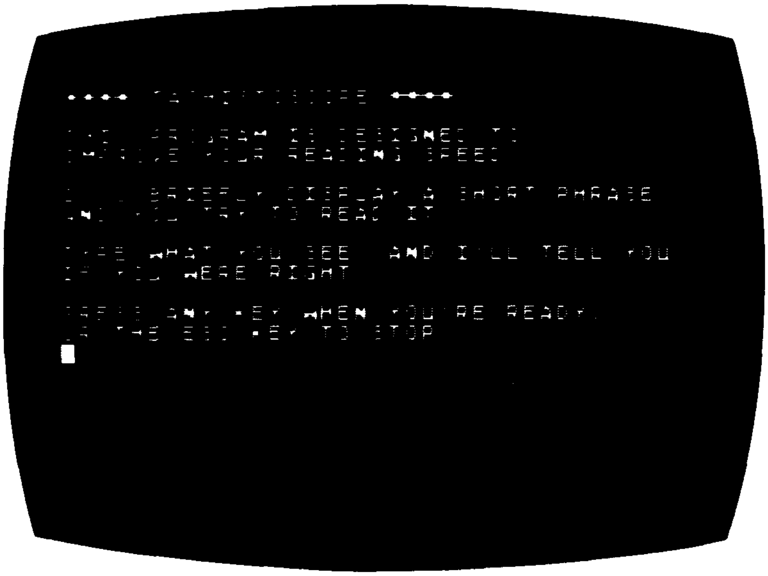
The fastest the computer can display a phrase and erase it is about 0.02 second (one-fiftieth). See if you can reach the top speed and still continue to read the phrases correctly.

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.

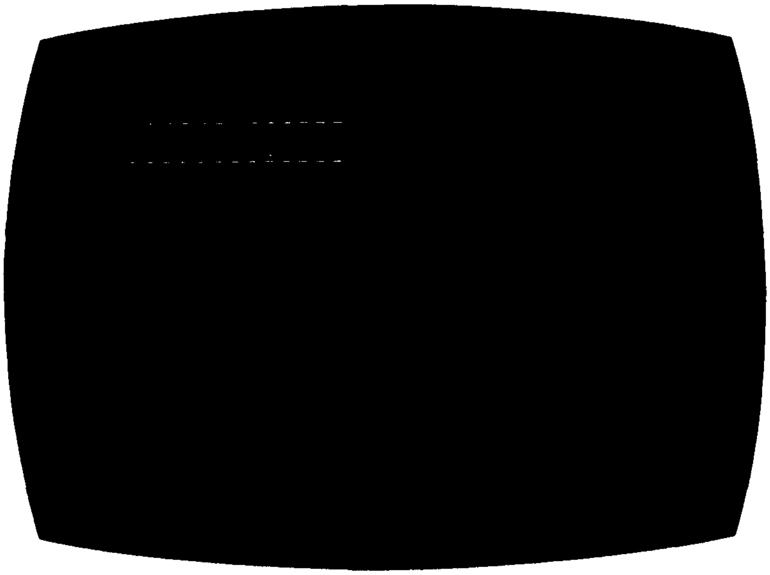
If you can incorporate this technique into your reading and increase the width of the phrases you absorb, your reading speed can increase dramatically.

SAMPLE RUN

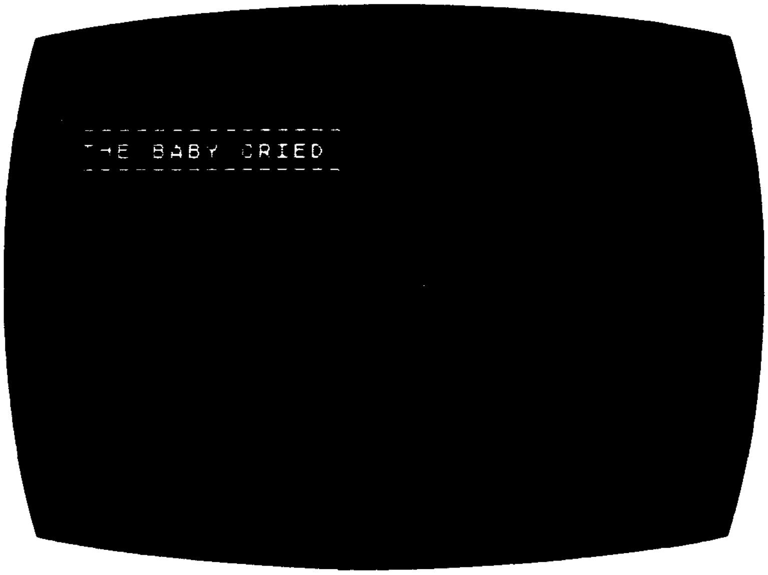


```
**** TACHISTOSCOPE ****

THIS PROGRAM IS DESIGNED TO
IMPROVE YOUR READING SPEED

I'LL BRIEFLY DISPLAY A SHORT PHRASE
AND YOU TRY TO READ IT

TYPE WHAT YOU SEE  AND I'LL TELL YOU
IF YOU WERE RIGHT

PRESS ANY KEY WHEN YOU'RE READY.
OR THE ESC KEY TO STOP
█
```

The program displays an introduction, then waits.

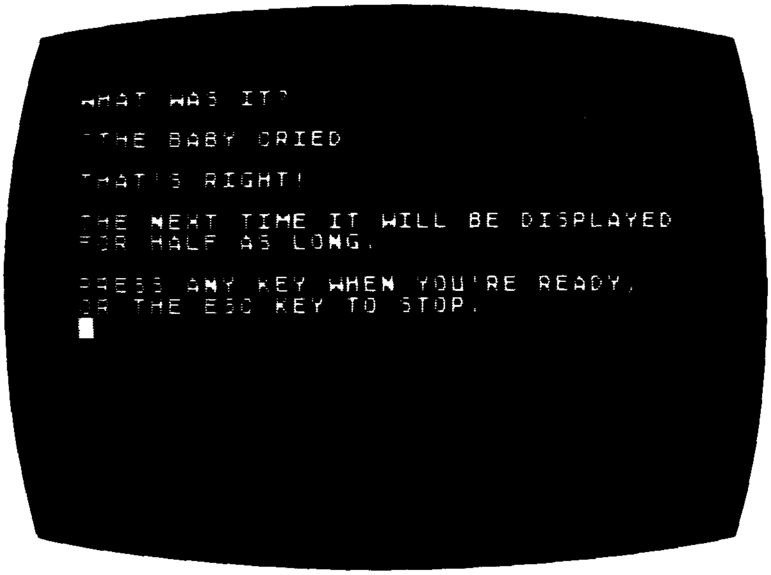


The program clears the screen and displays two parallel lines in the upper left of the screen for a couple of seconds.



```
---------------
THE BABY CRIED
---------------
```

The program flashes a short phrase (chosen at random) between the two lines for a fraction of a second, then clears the screen.



```
WHAT WAS IT?

THE BABY CRIED

THAT'S RIGHT!

THE NEXT TIME IT WILL BE DISPLAYED
FOR HALF AS LONG.

PRESS ANY KEY WHEN YOU'RE READY,
OR THE ESC KEY TO STOP.
█
```

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

```
10 REM TACHIST
15 REM COPYRIGHT 1984 DILITHIUM PRESS
130 T=256:CLOSE #1:OPEN #1,4,0,"K:"
140 L=50
150 DIM T$(40),X(L),R$(40)
160 C=0:POKE 82,2
170 READ R
180 IF R=-1 THEN 250
190 C=C+1:IF C>L THEN PRINT "** TOO MANY DATA ST
ATEMENTS **":END
200 X(C)=R
210 GOTO 170
250 GRAPHICS 0
260 PRINT
270 PRINT "**** TACHISTOSCOPE ****"
280 PRINT
290 PRINT "THIS PROGRAM IS DESIGNED TO"
300 PRINT "IMPROVE YOUR READING SPEED."
310 PRINT
320 PRINT "I'LL BRIEFLY DISPLAY A SHORT PHRASE,"
330 PRINT "AND YOU TRY TO READ IT."
340 PRINT
350 PRINT "TYPE WHAT YOU SEE, AND I'LL TELL YOU"
360 PRINT "IF YOU WERE RIGHT."
```

```
370 PRINT
410 PRINT "PRESS ANY KEY WHEN YOU'RE READY,"
415 PRINT "OR THE ESC KEY TO STOP."
420 GET #1,R:IF R=27 THEN END
430 R=INT(C*RND(1))+1
440 IF R=P1 OR R=P2 OR R=P3 OR R=P4 OR R=P5 THEN
 430
460 GOSUB 840:FOR K=1 TO 300:NEXT K
470 GOSUB 750:POSITION 2,2:PRINT T$
480 FOR J=1 TO T:NEXT J
500 PRINT CHR$(125):FOR K=1 TO 200:NEXT K
510 PRINT :PRINT :PRINT :PRINT
520 PRINT "WHAT WAS IT?":POKE 752,0
530 PRINT :INPUT R$
540 IF R$<>T$ THEN 700
550 PRINT :PRINT "THAT'S RIGHT!"
560 T=T/2
590 R$="FOR HALF AS LONG."
600 P1=P2:P2=P3:P3=P4:P4=P5:P5=R
610 PRINT
620 IF T<8 THEN T=8:R$="AT MAXIMUM SPEED."
630 PRINT "THE NEXT TIME IT WILL BE DISPLAYED":P
RINT R$
640 PRINT :GOTO 410
700 PRINT "NO, THAT'S NOT IT. IT WAS"
710 PRINT :PRINT CHR$(39);T$;CHR$(39)
720 T=T*2
730 IF T>1024 THEN T=1024:R$="AT THE SAME SPEED.
":GOTO 600
740 R$="FOR TWICE AS LONG.":GOTO 600
750 RESTORE X(R):READ T$:RETURN
840 POKE 752,1:PRINT CHR$(125):FOR J=1 TO 15:PRI
NT CHR$(45);
850 NEXT J:PRINT :PRINT :FOR J=1 TO 15
860 PRINT CHR$(45);:NEXT J
870 RETURN
900 DATA 910,920,930,940,950,960,970,980,990,100
0,1010,1020,1030,1040,1050,1060,1070,1080,1090,1
100,-1
910 DATA AT THE TIME
920 DATA THE BROWN COW
930 DATA LOOK AT THAT
940 DATA IN THE HOUSE
950 DATA THIS IS MINE
960 DATA SHE SAID SO
970 DATA THE BABY CRIED
980 DATA TO THE STORE
990 DATA READING IS FUN
1000 DATA IN ALL THINGS
1010 DATA HE GOES FAST
1020 DATA GREEN GRASS
1030 DATA TWO BIRDS FLY
```

```
1040 DATA LATE LAST NIGHT
1050 DATA THEY ARE HOME
1060 DATA THROUGH A DOOR
1070 DATA WE CAN TRY
1080 DATA MY FOOT HURTS
1090 DATA HAPPY NEW YEAR
1100 DATA ON THE PHONE
```

EASY CHANGES

1. Change the phrases that are displayed by changing the DATA statements that start at line 910. Add more and/or replace those shown with your own phrases or words. Line 140 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 line 140 to say

```
140 L=100
```

   Be sure to enter your DATA statements in the same form as shown in the program listing. The line numbers of all these DATA statements must be listed on earlier DATA statements, followed by a value of negative one, as shown in line 900. More than one such statement can be used if necessary, with a negative one at the end of the last one only. To begin with, you may want to start off with shorter phrases or single words. Later, try longer phrases. If you have 4000 bytes free (see Appendix 1), you have room for about 80 phrases of the approximate size shown in the program listing.
2. To change the length of time the first phrase is displayed, change the value of T in line 130. Double it to double the length of time, etc. Don't make it less than 8.
3. To cause all phrases to be displayed for the same length of time, remove lines 560 and 720, and insert these lines:

```
595 R$="AT THE SAME SPEED"
725 R$="AT THE SAME SPEED":GOTO 600
```

4. If you want to change the waiting period before the phrase is flashed on the screen, change the 300 in line 460. To make the delay five seconds, change it to 1000. To make it one second, change it to 150.
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:

```
515 GOTO 710
595 R$="AT THE SAME SPEED"
715 GOTO 590
```

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

130–150	Initializes variables.
160–210	Reads line numbers of DATA statements into X array.
250–370	Displays introduction.
410–420	Waits for operator to press a key.
430–440	Picks random phrase. Ensures no duplication from previous five phrases.
460	Clears screen and displays horizontal lines
470–500	Displays phrase for appropriate length of time.
510–530	Asks what phrase was.
540	Determines if typed phrase matches the phrase displayed.
550–640	Shortens time for next phrase if reply was correct. Saves subscript to avoid repetition. Goes back to wait for key to be pressed.
700–740	Shows what phrase was. Lengthens time for next phrase. Ensures that time period does not exceed maximum.
750	Subroutine to place phrase into T$.
840–870	Subroutine to display horizontal dash lines.
900–909	DATA statements with line numbers of other DATA statements.
910–1100	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.
X	Array of DATA statement line numbers.
T$	Selected phrase (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.
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 440 and 600.
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. It 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.

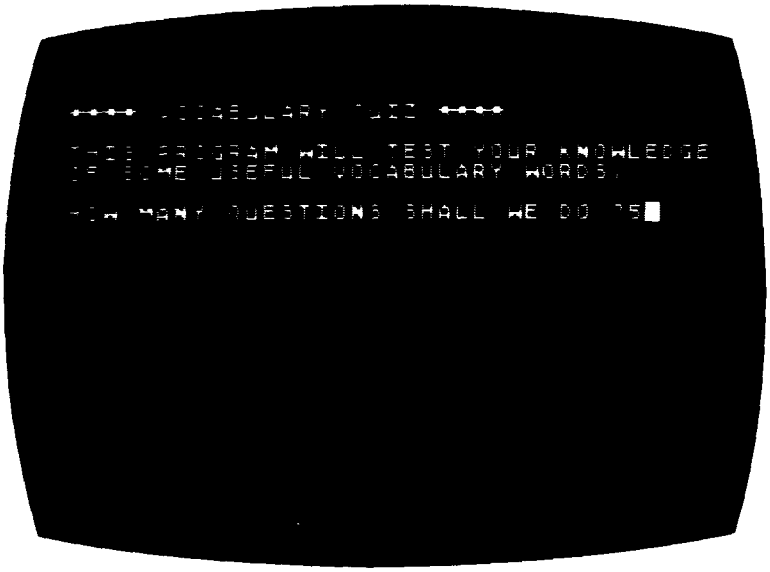
HOW TO USE IT

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.

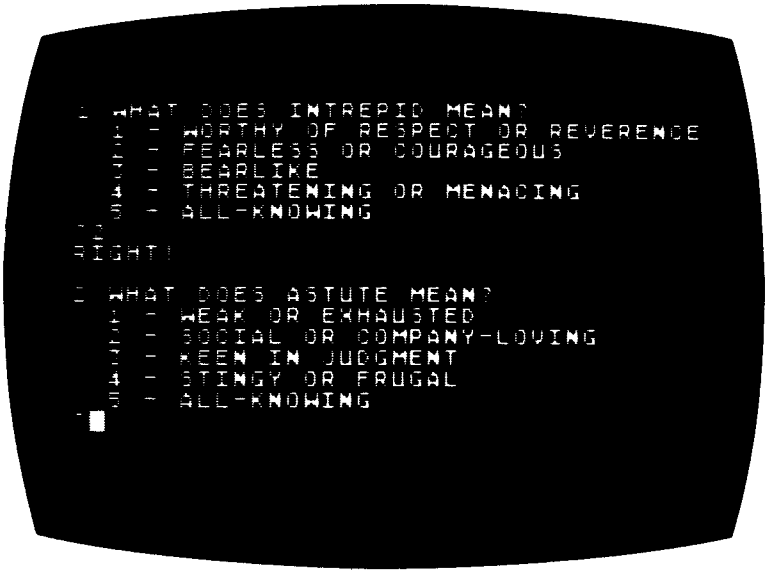
Finally, after the last question, you are shown the percentage you got correct, along with a comment on your performance. Then you have the option of going back for another round of questions or stopping.

SAMPLE RUN



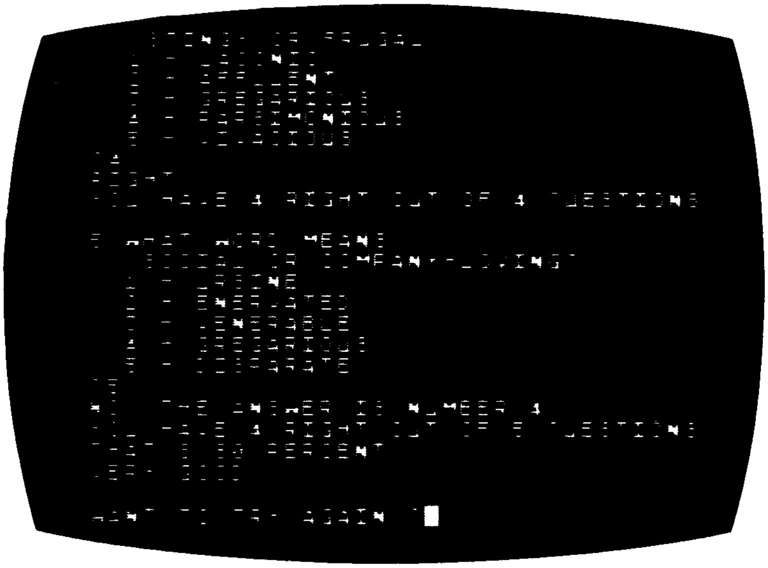
```
**** VOCABULARY QUIZ ****

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

HOW MANY QUESTIONS SHALL WE DO ?5█
```

The program displays an introduction and asks how many questions to display. The operator selects 5.



```
1 WHAT DOES INTREPID MEAN?
   1 - WORTHY OF RESPECT OR REVERENCE
   2 - FEARLESS OR COURAGEOUS
   3 - BEARLIKE
   4 - THREATENING OR MENACING
   5 - ALL-KNOWING
?2
RIGHT!

2 WHAT DOES ASTUTE MEAN?
   1 - WEAK OR EXHAUSTED
   2 - SOCIAL OR COMPANY-LOVING
   3 - KEEN IN JUDGMENT
   4 - STINGY OR FRUGAL
   5 - ALL-KNOWING
?
```

The program responds that the first answer was correct, and asks the next question.



```
    STINGY OR FRUGAL
   1 - [illegible]
   2 - [illegible]
   3 - [illegible]
   4 - PARSIMONIOUS
   5 - [illegible]
?4
RIGHT
YOU HAVE 4 RIGHT OUT OF 4 QUESTIONS

5 WHAT WORD MEANS
    SOCIAL OR COMPANY-LOVING?
   1 - [illegible]
   2 - ENERVATED
   3 - VENERABLE
   4 - GREGARIOUS
   5 - [illegible]
?[illegible]
NO, THE ANSWER IS NUMBER 4
YOU HAVE 4 RIGHT OUT OF 5 QUESTIONS
THAT IS 80 PERCENT
VERY GOOD

WANT TO TRY AGAIN ?
```

At the end of 5 questions, the program gives a final score and asks about trying again.

PROGRAM LISTING

```
10 REM VOCAB
15 REM COPYRIGHT 1984 DILITHIUM PRESS
100 REM VOCABULARY PROGRAM
200 TRAP 6500
300 GOSUB 1000
400 GOSUB 2000
500 GOSUB 3000
600 GOSUB 4000
700 GOSUB 5000
800 GOSUB 6000
900 IF E=0 THEN 500
910 GOTO 300
1000 IF E<>0 THEN 1060
1010 GRAPHICS 0:PRINT CHR$(125):POKE 82,2
1020 PRINT "**** VOCABULARY QUIZ ****"
1030 PRINT
1040 PRINT "THIS PROGRAM WILL TEST YOUR KNOWLEDG
E"
1050 PRINT "OF SOME USEFUL VOCABULARY WORDS."
1060 PRINT
1110 ZX=2:PRINT "HOW MANY QUESTIONS SHALL WE DO
";:INPUT L
1120 IF L>4 THEN 1140
1130 PRINT :PRINT "THAT'S NOT ENOUGH. LET'S DO 5
":FOR L=1 TO 1000:NEXT L:L=5
1140 IF E<>0 THEN 1200
1150 PRINT
1200 PRINT CHR$(125):RETURN
2000 IF E<>0 THEN 2200
2010 C=5
2020 D=26
2030 DIM D$(40),E$(40),P(C),X(D),R$(40),T$(40)
2050 J=1:RESTORE
2060 READ Z:IF Z=-1 THEN 2140
2070 X(J)=Z
2100 J=J+1
2110 IF J<=D THEN 2060
2120 PRINT "** TOO MANY DATA STATEMENTS. **"
2130 PRINT "** ONLY FIRST ";D;" ARE USED. **"
2140 D=J-1
2200 Q=1
2210 E=0
2220 Q1=0
2300 RETURN
3000 FOR J=1 TO C
3010 P(J)=0
3020 NEXT J
3030 FOR J=1 TO C
3040 P=INT(D*RND(1))+1
3045 IF P=P1 OR P=P2 OR P=P3 THEN 3040
```

```
3050 FOR K=1 TO J
3060 IF P(K)=P THEN 3040
3070 NEXT K
3080 P(J)=P
3090 NEXT J
3110 A=INT(C*RND(1))+1
3200 RETURN
4000 PRINT
4010 M=RND(1)
4020 RESTORE X(P(A)):READ D$:READ E$
4025 IF M>0.5 THEN 4100
4030 PRINT Q;" WHAT WORD MEANS"
4035 PRINT "    ";E$;"?"
4040 FOR J=1 TO C
4045 RESTORE X(P(J)):READ R$:READ T$
4050 PRINT "  ";J;" - ";R$
4060 NEXT J
4070 GOTO 4210
4100 PRINT Q;" WHAT DOES ";D$;" MEAN?"
4110 FOR J=1 TO C
4115 RESTORE X(P(J)):READ R$:READ T$
4120 PRINT "  ";J;" - ";T$
4130 NEXT J
4210 RETURN
5000 ZX=1:INPUT R
5010 IF R>=1 AND R<=C THEN 5050
5020 PRINT "I NEED A NUMBER FROM 1 TO ";C
5030 GOTO 5000
5050 IF R=A THEN 5100
5060 PRINT "NO, THE ANSWER IS NUMBER ";A
5070 GOTO 5210
5100 PRINT "RIGHT!"
5110 Q1=Q1+1
5210 IF Q=1 THEN 5300
5220 PRINT "YOU HAVE ";Q1;" RIGHT OUT OF ";Q;" Q
UESTIONS"
5300 P3=P2
5310 P2=P1
5320 P1=P(A)
5330 RETURN
6000 Q=Q+1
6010 IF Q<=L THEN RETURN
6020 E=1
6030 Q=Q1*100/(Q-1)
6040 IF Q>0 THEN 6070
6050 PRINT "WELL, THAT'S A 'PERFECT' SCORE..."
6060 GOTO 6200
6070 PRINT "THAT'S ";Q;" PERCENT."
6080 IF Q>25 THEN 6110
6090 PRINT "CONGRATULATIONS ON AVOIDING A SHUTOU
T"
6100 GOTO 6200
```

```
6110 IF Q>50 THEN 6140
6120 PRINT "YOU CAN USE SOME MORE PRACTICE"
6130 GOTO 6200
6140 IF Q>75 THEN 6170
6150 PRINT "NOT BAD, BUT ROOM FOR IMPROVEMENT"
6160 GOTO 6200
6170 PRINT "VERY GOOD!"
6180 IF Q>95 THEN PRINT "YOU'RE ALMOST AS SMART
AS I AM!"
6200 PRINT
6210 PRINT "WANT TO TRY AGAIN ";:INPUT R$
6215 IF LEN(R$)=0 THEN 6210
6220 IF R$(1,1)<>"N" THEN 6240
6230 PRINT :PRINT "CHECK YOU LATER":END
6240 IF R$(1,1)<>"Y" THEN 6200
6250 RETURN
6500 IF PEEK(195)=8 THEN PRINT :PRINT "** ILLEGA
L ENTRY **":PRINT :TRAP 6500:ON ZX GOTO 5000,111
0
6510 END
7000 DATA 7020,7030,7040,7050,7060,7070,7080,709
0,7100,7110,7120,7130,7140,7150,-1
7020 DATA ANONYMOUS,OF UNKNOWN OR HIDDEN ORIGIN
7030 DATA OMINOUS,THREATENING OR MENACING
7040 DATA AFFLUENT,WEALTHY
7050 DATA LACONIC,TERSE
7060 DATA INTREPID,FEARLESS OR COURAGEOUS
7070 DATA GREGARIOUS,SOCIAL OR COMPANY-LOVING
7080 DATA ENERVATED,WEAK OR EXHAUSTED
7090 DATA VENERABLE,WORTHY OF RESPECT OR REVEREN
CE
7100 DATA DISPARATE,DIFFERENT AND DISTINCT
7110 DATA VIVACIOUS,LIVELY OR SPIRITED
7120 DATA ASTUTE,KEEN IN JUDGMENT
7130 DATA URSINE,BEARLIKE
7140 DATA PARSIMONIOUS,STINGY OR FRUGAL
7150 DATA OMNISCIENT,ALL-KNOWING
```

EASY CHANGES

1. Add more DATA statements after line 7150, or replace the DATA statements between lines 7020 and 7150 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 value of D in line 2020

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 40 of them if you have 4000 bytes free (see Appendix 1), or over 200 with 10,000 bytes free. Be sure to add the line numbers of added DATA statements to the list in line 7000. This list is terminated with –1 to signal the end of the data.

2. To get something other than five choices for each question, change the value of C in line 2010. You might want only three or four choices per question.
3. If you do not want to be given a choice of how many questions are going to be asked, remove lines 1110 through 1130 and insert the following lines:

```
1110 PRINT "WE'LL DO TEN QUESTIONS."
1120 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

200	Sets error trap.
300– 910	Mainline routine. Calls major subroutines.
1000–1200	Displays introduction. Determines number of questions to be asked.
2000–2300	Reads line numbers of vocabulary words and definitions into arrays. Performs housekeeping.
3000–3200	Selects choices for answers and determines which will be the correct one.
4000–4210	Determines in which format the question will be asked. Asks it.
5000–5330	Accepts answer from operator. Determines if right or wrong. Keeps score. Saves subscripts of last three correct answers.
6000–6250	Gives final score. Asks about doing it again.
6500	Error trap for illegal inputs. Displays error message and tries again.
7000–7150	Variable 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. Used to DIM arrays.
D$	Selected word.
E$	Selected definition.
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.
P	Work variable.
P1, P2, P3	Last three correct answers.
T$	Temporary string.
K, ZX	Working variables.
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.
X	Array of DATA line numbers.

SUGGESTED PROJECTS

1. Modify lines 6030 through 6200 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 3045 and 5300 through 5320.)
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

INTRODUCTION TO GAME PROGRAMS

Almost everyone likes to play games. Computer games are a fun and entertaining use of your ATARI 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 ATARI 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 ATARI 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.

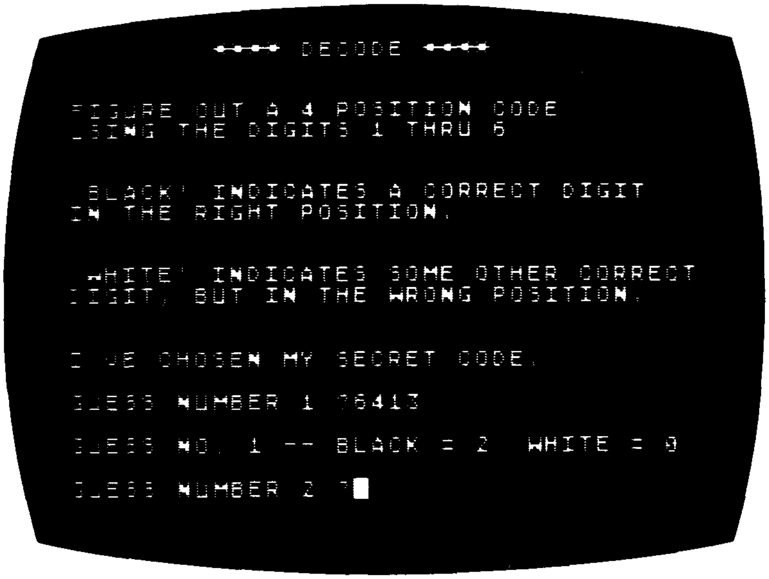
Each of the white numbers indicates a digit in your guess that is correct, but which is in the wrong position. For example, if the secret code is 6153 and your guess is 1434, you will be told that white is 2. The 1 and 3 are correct, but in wrong positions.

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.

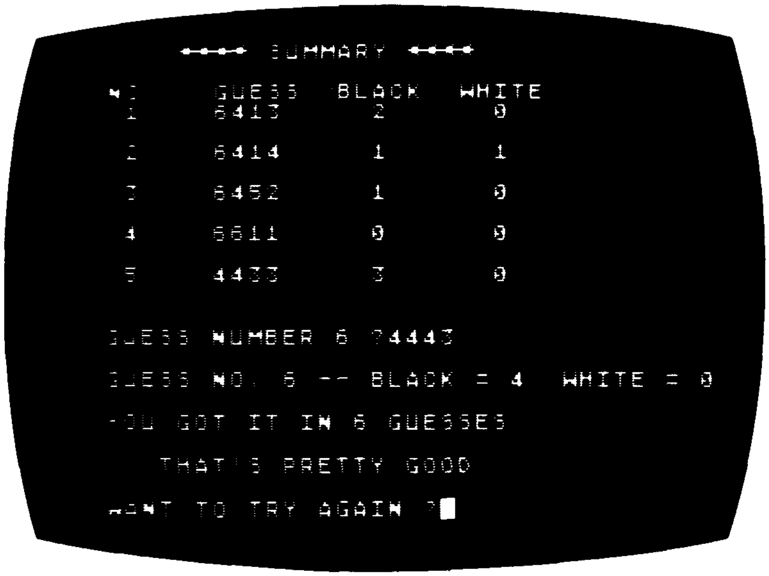
At any time during the game, you can ask for a "SUMMARY" by entering an S instead of a guess. This causes the program to clear the screen and display each guess (with the corresponding result) that has occurred so far.

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.



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

FIGURE OUT A 4 POSITION CODE
USING THE DIGITS 1 THRU 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. After the operator makes a guess, the program responds with a "black" and a "white" number, and asks for the second guess.



```
         **** SUMMARY ****

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. The program acknowledges that the guess is correct and asks about trying another game.

PROGRAM LISTING

```
10 REM DECODE
15 REM COPYRIGHT 1984 DILITHIUM PRESS
30 D=6:P=4:L=12
40 DIM GG(P,L),G(L),C(P),B(L),W(L),B$(40),A$(40)
,X(L)
50 GOSUB 650
60 GOSUB 220:GOSUB 240
65 TRAP 1000
70 PRINT "GUESS NUMBER ";G;" ";:INPUT A$
100 IF A$="S" THEN 300
110 IF A$="Q" THEN 380
120 IF LEN(A$)<>P THEN PRINT "ILLEGAL GUESS":GOT
O 70
130 FOR J=1 TO P
140 GG(J,G)=VAL(A$(J,J))
150 NEXT J
160 TRAP 40000
170 GOSUB 470
180 GOSUB 610
190 IF B(G)=P THEN 800
200 G=G+1:IF G>L THEN 940
210 GOTO 65
220 G=1:FOR J=1 TO P:C(J)=0:NEXT J
230 RETURN
240 FOR J=1 TO P
250 R=INT(D*RND(1))+1
260 C(J)=R
270 NEXT J
280 PRINT "I'VE CHOSEN MY SECRET CODE."
290 PRINT :RETURN
300 IF G=1 THEN PRINT "NO GUESS YET.":GOTO 65
310 PRINT CHR$(125):PRINT "     **** SUMMARY ****
":PRINT
320 PRINT "NO.   GUESS  BLACK  WHITE"
330 FOR J=1 TO G-1:IF J<10 THEN PRINT CHR$(32);
340 PRINT J;"    ";
345 FOR K=1 TO P:PRINT GG(K,J);:NEXT K
347 PRINT "      ";B(J);"       ";W(J)
350 IF G<10 THEN PRINT
360 NEXT J:PRINT
370 GOTO 65
380 PRINT
390 PRINT "CAN'T TAKE IT, HUH?"
400 PRINT :PRINT "WELL, MY CODE WAS?";
410 FOR J=1 TO 4
420 PRINT ".";
430 FOR K=1 TO 300:NEXT K
440 NEXT J
450 FOR J=1 TO P:PRINT C(J);:NEXT J:PRINT
460 GOTO 880
```

```
470 B=0:W=0
480 FOR J=1 TO P
490 G(J)=GG(J,G)
500 X(J)=C(J)
510 IF X(J)=G(J) THEN B=B+1:G(J)=0:X(J)=0
520 NEXT J
530 FOR J=1 TO P
540 H=0:FOR K=1 TO P
550 IF X(J)=0 THEN 580
560 IF C(J)<>G(K) THEN 580
570 H=1:G(K)=0:X(J)=0
580 NEXT K:W=W+H
590 NEXT J
600 RETURN
610 B(G)=B:W(G)=W:PRINT
620 PRINT "GUESS NO. ";G;
630 PRINT " -- BLACK = ";B;"  WHITE = ";W
640 PRINT :RETURN
650 GRAPHICS 0:PRINT CHR$(125):POKE 82,2
660 PRINT "          **** DECODE ****"
670 PRINT :PRINT
680 PRINT "FIGURE OUT A ";P;" POSITION CODE"
700 PRINT "USING THE DIGITS 1 THRU ";D
710 PRINT :PRINT
720 PRINT "'BLACK' INDICATES A CORRECT DIGIT"
730 PRINT "IN THE RIGHT POSITION."
740 PRINT
750 PRINT
760 PRINT "'WHITE' INDICATES SOME OTHER CORRECT"
770 PRINT "DIGIT, BUT IN THE WRONG POSITION."
780 PRINT :PRINT
790 RETURN
800 PRINT "YOU GOT IT IN ";G;" GUESSES"
810 IF G<5 THEN B$="OUTSTANDING!"
820 IF G=5 OR G=6 THEN B$="PRETTY GOOD"
830 IF G=7 THEN B$="NOT BAD"
840 IF G=8 THEN B$="NOT TOO GREAT"
850 IF G>8 THEN B$="PRETTY BAD"
860 PRINT :PRINT "...THAT'S ";B$
870 PRINT
880 PRINT "WANT TO TRY AGAIN ";:INPUT A$
890 IF LEN(A$)=0 THEN 880
900 IF A$(1,1)="Y" THEN 50
910 IF A$(1,1)<>"N" THEN 880
920 PRINT :PRINT "COWARD.":PRINT
930 END
940 PRINT
950 PRINT "THAT'S YOUR LIMIT OF GUESSES"
960 PRINT
970 PRINT "MY CODE WAS ";:FOR J=1 TO P:PRINT C(J
);:NEXT J
980 GOTO 870
```

```
1000 IF PEEK(195)<>18 THEN 1030
1010 PRINT "ILLEGAL GUESS"
1020 GOTO 65
1030 PRINT "ERROR TYPE ";PEEK(195);
1040 PRINT " IN LINE ";PEEK(187)*256+PEEK(186)
1050 END
```

EASY CHANGES

1. Modify line 30 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:

   30 D=8:P=5:L=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.
2. To change the program so that it will always display the "Summary" information after each guess automatically, replace line 210 with:

   210 GOTO 300

MAIN ROUTINES

10- 60	Initializes variables. Displays introduction. Chooses secret code.
65- 180	Gets a guess from operator. Analyzes reply. Displays result.
190	Determines if operator guessed correctly.
200- 210	Saves guess. Adds one to guess-counter. Determines if limit on number of guesses was exceeded.
220- 230	Subroutine to initialize variables.
240- 290	Subroutine to choose secret code and inform operator.
300- 370	Subroutine to display summary of guesses so far.
380- 460	Subroutine to slowly display secret code when operator quits.
470- 600	Subroutine to determine number of black and white responses for the guess.
610- 640	Subroutine to display number of black and white responses for the guess.
650- 790	Subroutine to display title and introduction.

800– 930	Subroutine to analyze operator's performance after correct answer is guessed and ask about playing again.
940– 980	Subroutine to display secret code after operator exceeds limit of number of guesses.
1000–1050	Error trap routine.

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.
GG	Array in which guesses are saved.
G, X	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	Work variables.
G	Counter of the number of guesses made.
A$	Reply by the operator.
C	Secret code chosen by the program (array).
J, K	Loop variables.
B, W	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 810 through 850 to determine the message to be displayed based on the number of possibilities in the code as well as G.
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 70.

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 ATARI 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 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 ATARI indicates who is rolling.

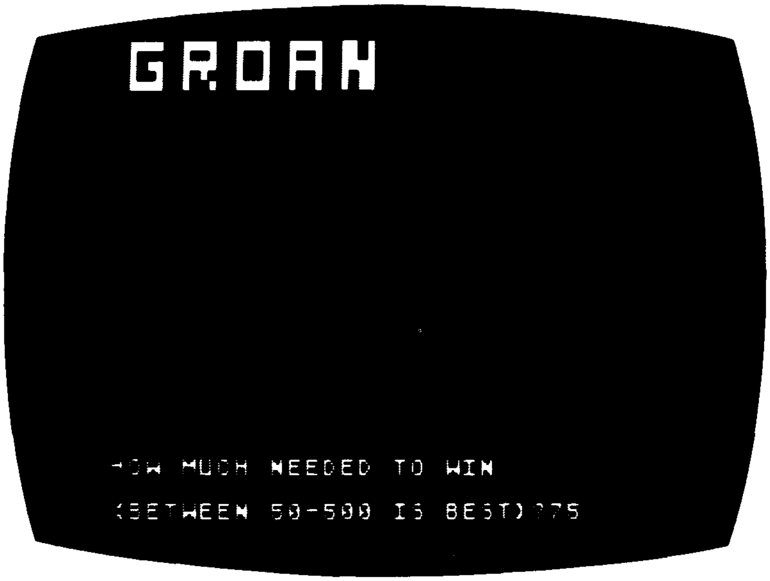
Each roll is followed by a display of the scoreboard. This scoreboard gives all relevant information: score needed to win, both players' scores before the current series of rolls, and the total score for the current series.

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 ATARI. 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.

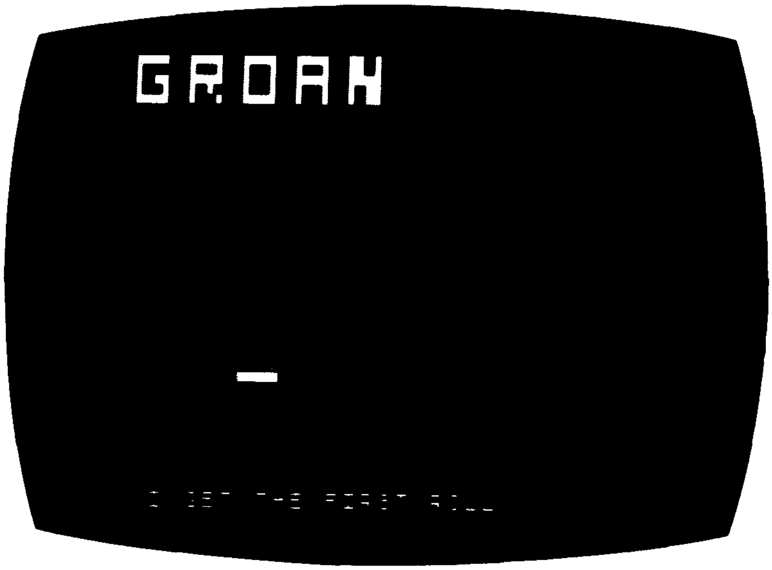
The first player to pass the dice with a score greater than or equal to the winning score is the victor. This will surely cause his opponent to GROAN. The computer will acknowledge the winner and then request another game.

SAMPLE RUN



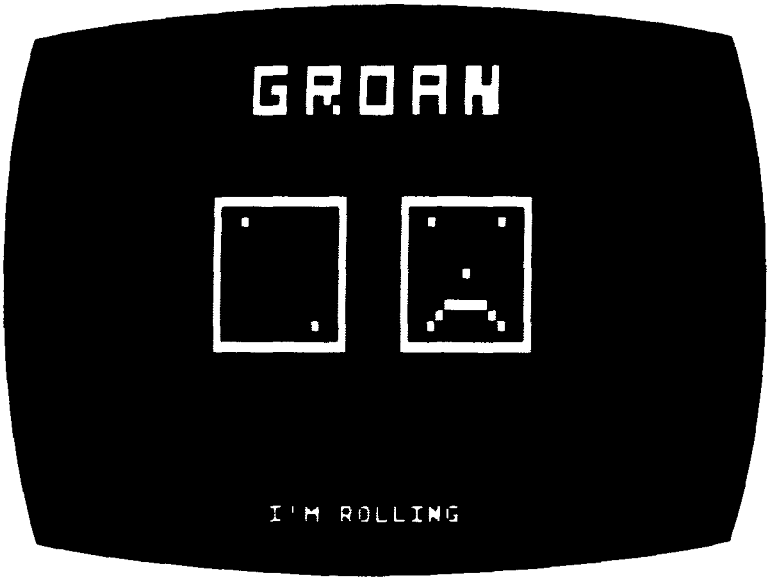
```
GROAN

HOW MUCH NEEDED TO WIN
(BETWEEN 50-500 IS BEST)?75
```

The operator has decided to challenge the ATARI to a 75-point game of GROAN.

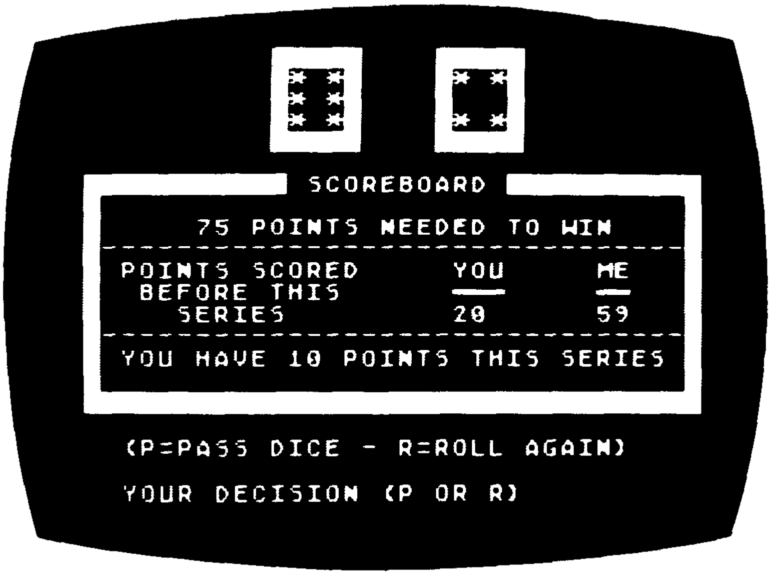


```
GROAN

I GET THE FIRST ROLL
```

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



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



```
              SCOREBOARD
      75 POINTS NEEDED TO WIN
 ----------------------------------
 POINTS SCORED      YOU      ME
  BEFORE THIS       ---      --
    SERIES          20       59
 ----------------------------------
 YOU HAVE 10 POINTS THIS SERIES

(P=PASS DICE - R=ROLL AGAIN)

YOUR DECISION (P OR R)
```

Later in the same game, the operator rolls a 10 to start a series of rolls. The score was ATARI-59, operator-20 before the roll. The operator must now decide whether to pass the dice or risk rolling again.

PROGRAM LISTING

```
10 REM GROAN
15 REM COPYRIGHT 1984 DILITHIUM PRESS
110 DIM B$(40),C$(40),Q$(40),P$(40)
120 OPEN #1,4,0,"K:"
150 A=0:H=A:T=A:B$=CHR$(160)
160 C$="*":BZ=253
200 GRAPHICS 5:SETCOLOR 2,0,0:POKE 752,1:POKE 82
,2:PRINT CHR$(125):GOSUB 2080:COLOR C
210 XS=6:YS=2
220 GOSUB 2200:GOSUB 2100
230 PRINT CHR$(125):PRINT "HOW MUCH NEEDED TO WI
N":PRINT
235 TRAP 6000
240 PRINT "(BETWEEN 50-500 IS BEST)";:INPUT W
250 W=INT(W):IF W<=0 THEN 230
260 GOSUB 2080:GOSUB 2500
270 IF Q>0.5 THEN 500
300 T=0
310 GOSUB 2700:T=T+R1+R2:IF F>0 THEN T=0
320 IF F=2 THEN H=0
330 GRAPHICS 5:SETCOLOR 2,0,0:POKE 752,1:PRINT C
HR$(125)
335 PRINT :PRINT " YOU'RE ROLLING"
340 P$="YOU":GOSUB 3200
350 IF F>0 THEN P$="ME":GOSUB 1560:DL=400:GOSUB
2050:GOTO 500
360 GOSUB 1300:IF Q$="R" THEN 310
370 H=H+T:IF H>=W THEN 900
500 T=0
510 GRAPHICS 5:SETCOLOR 2,0,0:POKE 752,1:PRINT C
HR$(125):GOSUB 2700
520 T=T+R1+R2:IF F>0 THEN T=0
530 IF F=2 THEN A=0
540 PRINT :PRINT "  I'M ROLLING"
550 P$="I":GOSUB 3200
560 IF F>0 THEN P$="YOU":GOSUB 1560:DL=400:GOSUB
 2050:GOTO 300
570 POSITION XS,YS:GOSUB 5000
580 IF AD=1 THEN PRINT "I'LL ROLL AGAIN":DL=250:
GOSUB 2050:GOTO 510
590 PRINT "I'LL STOP WITH THIS":A=A+T
600 IF A>=W THEN DL=350:GOSUB 2050:GOTO 900
610 YS=YS+2:P$="YOU":GOSUB 1560:DL=350:GOSUB 205
0:GOTO 300
900 XS=2:YS=2:PRINT CHR$(125):P$="WE":T=0:GOSUB
1010
910 POSITION 3,16:IF A>=W THEN PRINT "I WIN -- S
KILL TRIUMPHS AGAIN"
920 IF H>=W THEN PRINT "YOU WIN -- IT WAS SHEER
LUCK"
```

```
930 IF A>=W THEN PRINT CHR$(253)
940 IF H>=W THEN BL=250:GOSUB 2000
945 POSITION 3,18:PRINT "HOW ABOUT ANOTHER GAME
(Y OR N)";:INPUT Q$:IF Q$="Y" THEN 150
950 GRAPHICS 0:END
1000 GRAPHICS 0:SETCOLOR 2,0,0:POKE 752,1:PRINT
CHR$(125):XS=2:YS=9
1010 POSITION XS,YS
1020 FOR J=1 TO 11:PRINT B$;:NEXT J
1030 PRINT " SCOREBOARD ";
1040 FOR J=1 TO 11:PRINT B$;:NEXT J:PRINT
1050 FOR J=1 TO 9:PRINT B$;:POSITION 35,PEEK(84)
:PRINT B$:NEXT J
1060 FOR J=1 TO 34:PRINT B$;:NEXT J
1080 POSITION XS+6,YS+2
1090 PRINT W;" POINTS NEEDED TO WIN";
1100 POSITION XS+1,YS+3:FOR J=1 TO 32
1110 PRINT "-";:NEXT J
1120 POSITION XS+2,YS+4
1130 PRINT "POINTS SCORED";:Q=PEEK(84):POSITION
22,Q
1135 PRINT "YOU";:POSITION 30,Q:PRINT "ME"
1140 POSITION XS+3,YS+5
1150 PRINT "BEFORE THIS";:Q=PEEK(84):POSITION 22
,Q
1155 PRINT CHR$(18);CHR$(18);CHR$(18);:POSITION
30,Q:PRINT CHR$(18);CHR$(18)
1160 POSITION XS+5,YS+6
1170 PRINT "SERIES";:POSITION XS+20,YS+6:PRINT H
;
1180 POSITION XS+28,YS+6:PRINT A;
1190 POSITION XS+1,YS+7:FOR J=1 TO 32
1200 PRINT "-";:NEXT J:POSITION XS+2,YS+8
1210 PRINT P$;" HAVE ";T;" POINTS THIS SERIES";
1220 RETURN
1300 POSITION XS-2,YS
1310 PRINT "(P=PASS DICE - R=ROLL AGAIN)"
1320 POSITION XS-2,YS+2
1330 PRINT "YOUR DECISION (P OR R)";
1340 GET #1,Q:Q$=CHR$(Q)
1350 IF Q$="R" OR Q$="P" THEN RETURN
1370 GOTO 1340
1400 FOR Y=YS-2 TO YS+2 STEP 4
1410 FOR X=XS-2 TO XS+2:GOSUB 1500:NEXT X
1420 FOR X=XS+7 TO XS+11:GOSUB 1500:NEXT X:NEXT
Y
1430 FOR X=XS-2 TO XS+2 STEP 4
1440 FOR Y=YS-2 TO YS+2:GOSUB 1500:NEXT Y:NEXT X
1460 FOR X=XS+7 TO XS+11 STEP 4:FOR Y=YS-2 TO YS
+2
1470 GOSUB 1500:NEXT Y:NEXT X:RETURN
1500 POSITION X,Y:PRINT B$:RETURN
```

```
1520 POSITION X,Y:PRINT "G R O A N";:RETURN
1540 POSITION X,Y:PRINT "D E S P A I R !";:RETUR
N
1560 POSITION XS,YS:PRINT "DICE PASS TO ";P$:RET
URN
1600 POSITION XS,YS:PRINT C$;:RETURN
1610 POSITION XS-1,YS-1:PRINT C$;
1620 POSITION XS+1,YS+1:PRINT C$;:RETURN
1630 GOSUB 1600:GOSUB 1610:RETURN
1640 POSITION XS+1,YS-1:PRINT C$;
1650 POSITION XS-1,YS+1:PRINT C$;:GOSUB 1610:RET
URN
1660 GOSUB 1600:GOSUB 1640:RETURN
1670 POSITION XS-1,YS:PRINT C$;:POSITION XS+1,YS
1680 PRINT C$;:GOSUB 1640:RETURN
1700 POSITION XS-1,YS-1:PRINT CHR$(9);CHR$(32);C
HR$(15);
1710 POSITION XS,YS:PRINT CHR$(96);
1720 POSITION XS-1,YS+1
1730 PRINT CHR$(17);CHR$(18);CHR$(5);:RETURN
1800 LX=XS-L:RX=XS+L:UY=YS-L:LY=YS+L
1810 PLOT LX,UY:DRAWTO RX,UY:PLOT LX,LY:DRAWTO R
X,LY
1820 PLOT LX,UY:DRAWTO LX,LY:PLOT RX,UY:DRAWTO R
X,LY:RETURN
1900 PLOT XS,YS:RETURN
1910 PLOT XS-4,YS-5:PLOT XS+4,YS+5:RETURN
1920 GOSUB 1900:GOSUB 1910:RETURN
1930 PLOT XS-4,YS+5:PLOT XS+4,YS-5:GOSUB 1910:RE
TURN
1940 GOSUB 1900:GOSUB 1930:RETURN
1950 PLOT XS-4,YS:PLOT XS+4,YS:GOSUB 1930:RETURN
1960 GOSUB 1940:FOR J=1 TO 3
1970 PLOT XS-J-1,YS+J+2:PLOT XS+J+1,YS+J+2
1980 NEXT J:PLOT XS-1,YS+3:PLOT XS,YS+3:PLOT XS+
1,YS+3
1990 RETURN
2000 SOUND 0,230,14,14:FOR J=1 TO BL:NEXT J:SOUN
D 0,0,0,0:RETURN
2050 FOR J=1 TO DL:NEXT J:RETURN
2080 C=2:RETURN
2100 BL=300:GOSUB 2000
2110 RETURN
2150 BL=200:GOSUB 2000:RETURN
2200 PLOT XS,YS:DRAWTO XS+3,YS:PLOT XS,YS:DRAWTO
 XS,YS+4
2210 PLOT XS,YS+4:DRAWTO XS+3,YS+4:PLOT XS+3,YS+
2:DRAWTO XS+3,YS+4
2220 PLOT XS+2,YS+2:PLOT XS+6,YS:DRAWTO XS+6,YS+
4
2230 PLOT XS+6,YS:DRAWTO XS+9,YS:PLOT XS+6,YS+2:
DRAWTO XS+9,YS+2
```

```
2240 PLOT XS+9,YS+1:PLOT XS+8,YS+3
2250 PLOT XS+9,YS+4:PLOT XS+12,YS:DRAWTO XS+15,Y
S
2260 PLOT XS+12,YS+4:DRAWTO XS+15,YS+4:PLOT XS+1
2,YS:DRAWTO XS+12,YS+4
2270 PLOT XS+15,YS:DRAWTO XS+15,YS+4:PLOT XS+18,
YS:DRAWTO XS+18,YS+4
2280 PLOT XS+21,YS:DRAWTO XS+21,YS+4:PLOT XS+18,
YS:DRAWTO XS+21,YS
2290 PLOT XS+18,YS+2:DRAWTO XS+21,YS+2:PLOT XS+2
4,YS:DRAWTO XS+24,YS+4
2300 PLOT XS+27,YS:DRAWTO XS+27,YS+4:PLOT XS+25,
YS:DRAWTO XS+25,YS+2
2310 PLOT XS+26,YS+2:DRAWTO XS+26,YS+4:RETURN
2500 DL=20:X=20:Y=33
2510 PRINT CHR$(125);"NOW A COIN TOSS FOR THE FI
RST ROLL"
2520 PRINT :PRINT "THE COIN IS IN THE AIR AND...
."
2530 Y=Y-2:COLOR C:GOSUB 2650:GOSUB 2050
2540 COLOR 0:GOSUB 2650:COLOR C:GOSUB 2660:GOSUB
 2050
2550 COLOR 0:GOSUB 2660:IF Y>9 THEN 2530
2560 Y=Y+2:COLOR C:GOSUB 2650:GOSUB 2050
2570 COLOR 0:GOSUB 2650:COLOR C
2580 COLOR 0:GOSUB 2660:IF Y<33 THEN 2560
2590 COLOR C:Y=33:GOSUB 2650:BL=50:GOSUB 2000
2600 Q=RND(1):Q$="YOU":IF Q>0.5 THEN Q$="I"
2610 PRINT CHR$(125):PRINT
2620 PRINT CHR$(32);CHR$(32);Q$;" GET THE FIRST
ROLL"
2630 DL=300:GOSUB 2050:RETURN
2650 PLOT X-2,Y:DRAWTO X+2,Y:RETURN
2660 PLOT X,Y-2:DRAWTO X,Y+2:RETURN
2700 R1=INT(RND(1)*6)+1:R2=INT(RND(1)*6)+1
2710 F=0:IF R1=1 THEN F=1
2720 IF R2=1 THEN F=F+1
2730 RETURN
2800 FOR J=0 TO 14:L=J/2
2810 XS=9:YS=8+J:COLOR C:GOSUB 1800
2820 XS=30:GOSUB 1800:COLOR 0:XS=9:GOSUB 1800
2830 XS=30:GOSUB 1800:NEXT J
2840 COLOR C:XS=9:GOSUB 1800
2850 XS=30:GOSUB 1800:RETURN
3000 XS=14:YS=5:GOSUB 1400:Q=R1:GOSUB 3100
3010 XS=23:Q=R2:GOSUB 3100
3020 IF R1=1 THEN X=2:Y=YS:GOSUB 1520:GOSUB 2150
:DL=100:GOSUB 2050
3030 IF R2=1 THEN X=27:Y=YS:GOSUB 1520:GOSUB 215
0
3050 IF F<2 THEN RETURN
3060 DL=200:GOSUB 2050
3070 X=12:Y=1:GOSUB 1540:BL=300
```

```
3080 SOUND 0,230,12,14:FOR J=1 TO BL:NEXT J:SOUN
D 0,0,0,0:RETURN
3100 ON Q GOSUB 1700,1610,1630,1640,1660,1670:RE
TURN
3200 GOSUB 2080:GOSUB 2800:XS=9:YS=22:Q=R1:GOSUB
 3250
3210 XS=30:Q=R2:GOSUB 3250:GOSUB 2080:COLOR C:XS
=6
3220 YS=2:IF F>0 THEN GOSUB 2200:GOSUB 2150:GOTO
 3240
3230 DL=400:GOSUB 2050
3240 GOSUB 1000:GOSUB 3000:XS=6:YS=21:RETURN
3250 ON Q GOSUB 1960,1910,1920,1930,1940,1950:RE
TURN
5000 V=A+T:IF V>=W THEN 5100
5010 IF W-H<10 THEN 5110
5020 IF A>=H THEN CT=T/25:GOTO 5050
5030 IF V<H THEN CT=T/35:GOTO 5050
5040 CT=T/30
5050 IF RND(1)>CT THEN 5110
5100 AD=0:RETURN
5110 AD=1:RETURN
6000 IF PEEK(195)=8 THEN PRINT "ILLEGAL ENTRY":T
RAP 6000:GOTO 240
```

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 230 and deleting line 240. Simply set W to the winning score desired. For example:

   ```
   230 W=100:PRINT CHR$(125)
   ```

   would make the winning score 100. (Don't forget to delete line 240.)
2. The rolling dice graphics display before each roll can be eliminated by deleting lines 2810, 2820, 2830, 2840 and changing lines 2800 and 2850 as follows:

   ```
   2800 YS=22:L=7:COLOR  C
   2850 XS=9:GOSUB 1800:XS=30:
        GOSUB 1800:RETURN
   ```

   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:

```
2050 DL=DL/2:FOR J=1 TO DL:NEXT J
2055 DL=2*DL:RETURN
```

To slow down the pacing, try:

```
2050 DL=2*DL:FOR J=1 TO DL:NEXT J
2055 DL=DL/2:RETURN
```

4. The color of the graphic displays can be altered by modifying line 2080. Try:

```
2080 C=1:RETURN
```

to see the effect.

5. As currently written, the program should fit in a 16K ATARI 400 (see Appendix 1). If it doesn't, however, it can be made to squeeze into this system configuration by: implementing Easy Change #2 above; deleting lines 2210 through 2310; deleting lines 2540 through 2590; and adding

```
2200 RETURN
2500 REM
2530 DL=2000:GOSUB 2050
```

These changes remove some of the program's fancy effects but do not materially change the performance of the game.

MAIN ROUTINES

110- 160	Initializes constants.
200- 270	Initial displays. Gets winning score.
300- 370	Human rolls.
500- 610	ATARI rolls.
900- 950	Ending messages.
1000-1220	Displays scoreboard
1300-1370	Asks user for re-roll decision.
1400-1470	Draws scoreboard dice outline.
1500-1560	Subroutines to print various messages.
1600-1730	Draws scoreboard dice faces.
1800-1820	Draws graphics dice outline.
1900-1990	Draws graphics dice faces.
2000-2150	Delay loops and buzzer tone control loops.
2200-2310	Draws graphics "groan."
2500-2660	Performs coin toss.
2700-2730	Determines dice roll.
2800-2850	Controls graphics dice rolling.

3000–3100	Controls scoreboard display.
3200–3250	Subroutine to control scoreboard and dice faces.
5000–5110	ATARI's strategy. Sets AD=0 to stop rolling or AD=1 to continue rolling.
6000	Input error trap. Display error message and try again.

MAIN VARIABLES

A	Previous score of ATARI.
H	Previous score of human.
T	Score of current series of rolls.
B$	String of one blank character.
C$	String of one asterisk.
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	Buzzer length (length of tone generation).
P$	String name of current roller.
Q$	Work string variable.
AD	ATARI strategy flag (0=stop rolling, 1=roll again).
J, K	Loop indices.
X, Y	Horizontal, vertical print position.
L	Half length of a die side.
LX, RX	Left, right position of a die face.
UY, LY	Upper, lower position of a die face.
V	Score ATARI would have if it passed the dice.
CT	Cutoff threshold used in ATARI's built-in strategy.

SUGGESTED PROJECTS

1. The computer's built-in strategy is contained from line 5000 on. Remember, after a no-frown roll, the ATARI 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 ATARI passes the dice or AD=1 to mean the ATARI will roll again.

2. Ask the operator for his/her 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 ATARI will play against you. But be careful! You will find your computer quite a formidable opponent.

The rules of JOT are fairly 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 ATARI alternating guesses at each other's secret word.

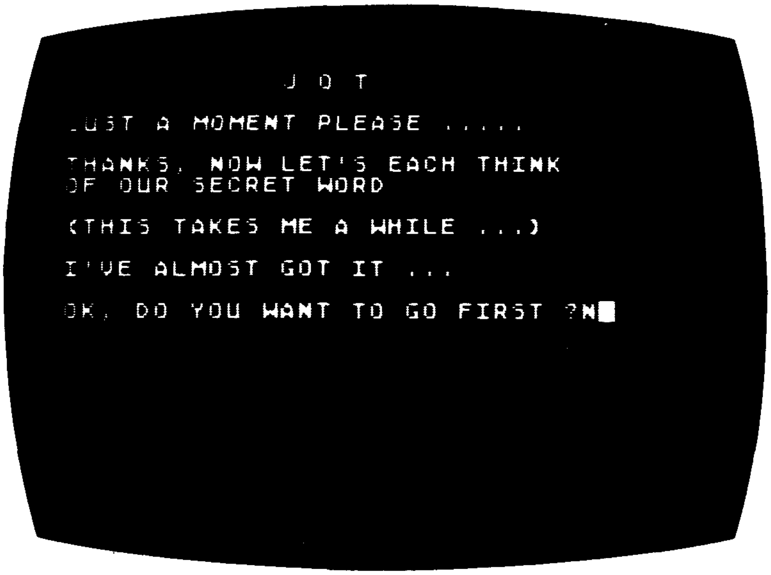
After the computer guesses, it will immediately ask you how it did. Possible replies are **0, 1, 2, 3,** or **R**. The response of **R** (for right) means the ATARI has just guessed your word correctly—a truly humbling experience. The numerical replies indicate that the word guessed by the ATARI had that number of hits in your secret word. A response of 3 means that all the letters were correct, but they need to be rearranged 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 ATARI 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.

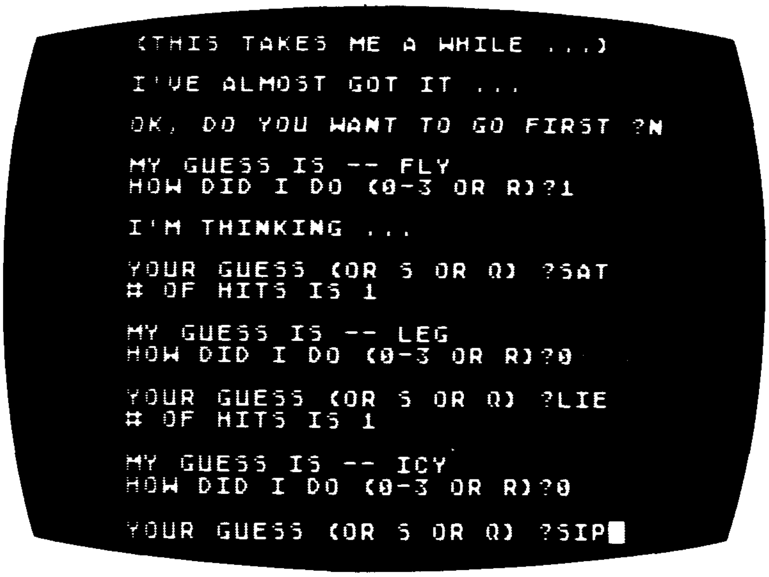
When not making one of these special replies, you will input a guess at the computer's secret word. This will be, of course, a three-letter word. If the word used is not legal, the computer will so inform you. After a legal guess, you will be told how many hits your guess had. If you correctly guess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

SAMPLE RUN



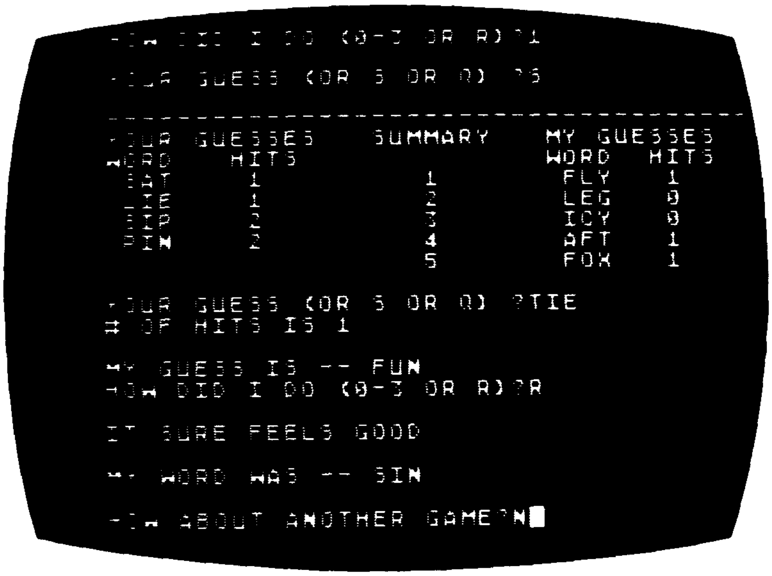
```
                  J O T

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.



```
(THIS TAKES ME A WHILE ...)

I'VE ALMOST GOT IT ...

OK, DO YOU WANT TO GO FIRST ?N

MY GUESS IS -- FLY
HOW DID I DO (0-3 OR R)?1

I'M THINKING ...

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

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

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

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

YOUR GUESS (OR S OR Q) ?SIP█
```

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



```
HOW DID I DO (0-3 OR R)?1

YOUR GUESS (OR S OR Q) ?S

----------------------------------------
YOUR GUESSES    SUMMARY    MY GUESSES
WORD    HITS               WORD   HITS
 SAT     1          1       FLY    1
 LIE     1          2       LEG    0
 SIP     2          3       ICY    0
 PIN     2          4       AFT    1
                    5       FOX    1

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

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

IT SURE FEELS GOOD

MY WORD WAS -- SIN

HOW ABOUT ANOTHER GAME?N
```

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

```
10 REM JOT
15 REM COPYRIGHT 1984 DILITHIUM PRESS
140 M=25:N=406
150 DIM A$(4*(N+4)),G1$(4*M),G2$(5*M),H1(M),H2(M
),XX$(40),Q$(40),M$(3),M1$(1),M2$(1),M3$(1),P$(5
)
200 G1=0:G2=0
210 L=N
250 PRINT CHR$(125):POSITION 14,PEEK(84):PRINT "
J O T":PRINT
260 PRINT "JUST A MOMENT PLEASE .....":GOSUB 300
0:Q=INT(RND(1)*N)+1:PRINT
270 PRINT "THANKS, NOW LET'S EACH THINK":PRINT "
OF OUR SECRET WORD"
280 PRINT :PRINT "(THIS TAKES ME A WHILE ...)"
290 GOSUB 2200:M$=A$(Q*3,Q*3+2):PRINT :PRINT "OK
, ";
300 PRINT "DO YOU WANT TO GO FIRST ";:INPUT Q$
305 IF Q$="" THEN 330
310 IF Q$(1,1)="N" THEN 600
320 IF Q$(1,1)="Y" THEN 500
330 PRINT :PRINT "** YES OR NO PLEASE **":PRINT
:GOTO 300
500 PRINT :PRINT "YOUR GUESS (OR S OR Q) ";:INPU
T P$:IF P$="S" THEN GOSUB 1000:GOTO 500
```

```
510 IF P$="Q" THEN 3210
520 IF P$=M$ THEN G1=G1+1:G1$(G1*3,G1*3+2)=P$:H1
(G1)=9:GOTO 3400
530 GOSUB 1800:IF F=0 THEN PRINT "* THAT'S NOT A
 LEGAL WORD - RETRY *":GOTO 500
540 Q$=M$:GOSUB 2600:Q$=P$:GOSUB 1500
550 PRINT "# OF HITS IS ";Q
560 G1=G1+1:G1$(G1*3,G1*3+2)=Q$:H1(G1)=Q
570 IF G1=M THEN 3600
600 Q$=A$(L*3,L*3+2):G2=G2+1:G2$(G2*3,G2*3+2)=Q$
610 PRINT :PRINT "MY GUESS IS -- ";Q$
620 PRINT "HOW DID I DO (0-3 OR R)";:INPUT P$
625 IF P$="" THEN 610
630 P$=P$(1,1)
640 IF P$="R" THEN H2(G2)=9:GOTO 3200
645 IF P$="0" OR P$="1" OR P$="2" OR P$="3" THEN
 660
650 PRINT "** BAD ANSWER **":GOTO 610
660 P=VAL(P$):IF L>100 THEN PRINT :PRINT "I'M TH
INKING ..."
670 H2(G2)=P:GOSUB 800
680 GOTO 500
800 Q$=G2$(G2*3,G2*3+2):H=H2(G2):J=0:GOSUB 2600:
L=L-1:IF L<1 THEN 900
810 J=J+1:IF J>L THEN 1500
820 Q$=A$(J*3,J*3+2):GOSUB 1500
830 IF Q=H THEN 810
840 A=J:B=L:GOSUB 2400:L=L-1
850 IF L<1 THEN 900
860 IF L>=J THEN 820
870 RETURN
900 PRINT :PRINT "** SOMETHING'S WRONG **"
910 PRINT :PRINT "** WHAT'S YOUR SECRET WORD";:I
NPUT P$:GOSUB 1800
920 IF F=0 THEN PRINT :PRINT "* ILLEGAL WORD --
I HAD NO CHANCE *":GOTO 1200
930 PRINT :PRINT "** YOU GAVE A BAD ANSWER SOMEW
HERE **"
940 PRINT "** CHECK THE SUMMARY **":GOSUB 1000
950 GOTO 1200
1000 PRINT :Q=G1:IF G2>G1 THEN Q=G2
1010 IF Q=0 THEN PRINT "** NO GUESSES YET **":RE
TURN
1020 FOR J=1 TO 36:PRINT "-";:NEXT J:PRINT "-"
1030 PK=PEEK(84):PRINT "YOUR GUESSES";
1040 POSITION 17,PK:PRINT "SUMMARY";
1050 POSITION 27,PK:PRINT "MY GUESSES":PK=PEEK(8
4)
1060 PRINT "WORD";:POSITION 9,PK:PRINT "HITS";:P
OSITION 27,PK:PRINT "WORD  HITS"
1070 FOR J=1 TO Q:PK=PEEK(84):K=1:IF J>9 THEN K=
0
1080 IF J>G1 THEN POSITION 19+K,PK:PRINT J;:POSI
TION 28,PK
```

```
1085 IF J>G1 THEN PRINT G2$(J*3,J*3+2);:POSITION
 34,PK:PRINT H2(J):GOTO 1110
1090 IF J>G2 THEN PRINT CHR$(32);G1$(J*3,J*3+2);
:POSITION 10,PK
1095 IF J>G2 THEN PRINT H1(J);:POSITION 19+K,PK:
PRINT J:GOTO 1110
1100 PRINT CHR$(32);G1$(J*3,J*3+2);:POSITION 10,
PK:PRINT H1(J);:POSITION 19+K,PK
1105 PRINT J;:POSITION 28,PK:PRINT G2$(J*3,J*3+2
);:POSITION 34,PK:PRINT H2(J)
1110 NEXT J:RETURN
1200 PRINT :PRINT "HOW ABOUT ANOTHER GAME";:INPU
T Q$
1205 IF Q$="" THEN 1230
1210 IF Q$(1,1)="Y" THEN 200
1220 IF Q$(1,1)="N" THEN GRAPHICS 0:END
1230 PRINT :PRINT "** YES OR NO PLEASE **":GOTO
1200
1500 P$=Q$(1,1):Q=0:GOSUB 1600
1510 P$=Q$(2,2):GOSUB 1600
1520 P$=Q$(3,3):GOSUB 1600:RETURN
1600 IF P$=M1$ OR P$=M2$ OR P$=M3$ THEN Q=Q+1
1610 RETURN
1800 F=0
1810 FOR J=1 TO N
1820 IF A$(J*3,J*3+2)=P$ THEN F=1:RETURN
1830 NEXT J:RETURN
2200 FOR A=N TO 100 STEP -1:B=INT(RND(1)*A)+1
2210 GOSUB 2400:NEXT A
2220 PRINT :PRINT "I'VE ALMOST GOT IT ..."
2230 FOR A=99 TO 2 STEP -1:B=INT(RND(1)*A)+1
2240 GOSUB 2400:NEXT A:RETURN
2400 Q$=A$(B*3,B*3+2):A$(B*3,B*3+2)=A$(A*3,A*3+2
):A$(A*3,A*3+2)=Q$:RETURN
2600 M1$=Q$(1,1):M2$=Q$(2,2)
2610 M3$=Q$(3,3):RETURN
3000 RESTORE :FOR P=1 TO N:PX=P*3:READ XX$:A$(PX
,PX+2)=XX$:NEXT P:RETURN
3200 PRINT :PRINT "IT SURE FEELS GOOD"
3210 PRINT :PRINT "MY WORD WAS -- ";M$
3220 GOTO 1200
3400 PRINT :PRINT "CONGRATULATIONS - THAT WAS IT
":PRINT
3410 PRINT "WHAT WAS YOUR WORD";:INPUT P$:GOSUB
1800:J=1
3420 IF F=0 THEN PRINT :PRINT "** ILLEGAL WORD -
 I HAD NO CHANCE **":GOTO 1200
3430 IF A$(J*3,J*3+2)=P$ THEN PRINT :PRINT "NICE
 WORD":GOTO 1200
3440 J=J+1:IF J<=L THEN 3430
3450 PRINT :PRINT "** YOU MADE AN ERROR SOMEWHER
E **":PRINT "** CHECK THE SUMMARY **"
3460 GOSUB 1000:GOTO 1200
```

```
3600 PRINT :PRINT "** SORRY, I'M OUT OF MEMORY *
*":PRINT
3610 PRINT "MY WORD WAS - ";M$:GOTO 1200
5000 DATA ACE,ACT,ADE,ADO,ADS,AFT,AGE
5010 DATA AGO,AID,AIL,AIM,AIR,ALE,ALP
5020 DATA AND,ANT,ANY,APE,APT,ARC,ARE
5030 DATA ARK,ARM,ART,ASH,ASK,ASP,ATE
5040 DATA AWE,AWL,AXE,AYE,BAD,BAG,BAN
5050 DATA BAR,BAT,BAY,BED,BEG,BET,BID
5060 DATA BIG,BIN,BIT,BOA,BOG,BOW,BOX
5070 DATA BOY,BUD,BUG,BUM,BUN,BUS,BUT
5080 DATA BUY,BYE,CAB,CAD,CAM,CAN,CAP
5090 DATA CAR,CAT,COB,COD,COG,CON,COP
5100 DATA COT,COW,COY,CRY,CUB,CUD,CUE
5110 DATA CUP,CUR,CUT,DAB,DAM,DAY,DEN
5120 DATA DEW,DIE,DIG,DIM,DIN,DIP,DOE
5130 DATA DOG,DON,DOT,DRY,DUB,DUE,DUG
5140 DATA DYE,DUO,EAR,EAT,EGO,ELK,ELM
5150 DATA END,ELF,ERA,FAD,FAG,FAN,FAR
5160 DATA FAT,FED,FEW,FIG,FIN,FIR,FIT
5170 DATA FIX,FLY,FOE,FOG,FOR,FOX,FRY
5180 DATA FUN,FUR,GAP,GAS,GAY,GEM,GET
5190 DATA GIN,GNU,GOB,GOD,GOT,GUM,GUN
5200 DATA GUT,GUY,GYP,HAD,HAG,HAM,HAS
5210 DATA HAT,HAY,HEN,HEX,HID,HIM,HIP
5220 DATA HIS,HIT,HER,HEM,HOE,HOG,HOP
5230 DATA HOT,HOW,HUB,HUE,HUG,HUM,HUT
5240 DATA ICE,ICY,ILK,INK,IMP,ION,IRE
5250 DATA IRK,ITS,IVY,JAB,JAR,JAW,JAY
5260 DATA JOB,JOG,JOT,JOY,JUG,JAG,JAM
5270 DATA JET,JIB,JIG,JUT,KEG,KEY,KID
5280 DATA KIN,KIT,LAB,LAD,LAG,LAP,LAW
5290 DATA LAY,LAX,LED,LEG,LET,LID,LIE
5300 DATA LIP,LIT,LOB,LOG,LOP,LOT,LOW
5310 DATA LYE,MAD,MAN,MAP,MAR,MAT,MAY
5320 DATA MEN,MET,MID,MOB,MOP,MOW,MUD
5330 DATA MIX,MUG,NAB,NAG,NAP,NAY,NET
5340 DATA NEW,NIL,NIP,NOD,NOT,NOR,NOW
5350 DATA NUT,OAF,OAK,OAR,OAT,ODE,OIL
5360 DATA OLD,ONE,OPT,ORE,OUR,OUT,OVA
5370 DATA OWE,OWL,OWN,PAD,PAL,PAN,PAR
5380 DATA PAT,PAW,PAY,PEA,PEG,PEN,PET
5390 DATA PEW,PIE,PIG,PIT,PLY,POD,POT
5400 DATA POX,PER,PIN,PRO,PRY,PUB,PUN
5410 DATA PUS,PUT,RAG,RAM,RAN,RAP,RAT
5420 DATA RAW,RAY,RED,RIB,RID,REV,RIG
5430 DATA RIM,RIP,ROB,ROD,ROE,ROT,ROW
5440 DATA RUB,RUE,RUG,RUM,RUN,RUT,RYE
5450 DATA SAD,SAG,SAP,SAT,SAW,SAY,SET
5460 DATA SEW,SEX,SHY,SEA,SIN,SHE,SIP
5470 DATA SIR,SIT,SIX,SKI,SKY,SLY,SOB
5480 DATA SOD,SON,SOW,SOY,SPA,SPY,STY
5490 DATA SUE,SUM,SUN,TAB,TAD,TAG,TAN
```

```
5500 DATA TAP,TAX,TAR,TEA,TEN,THE,THY
5510 DATA TIC,TIE,TIN,TIP,TOE,TON,TOP
5520 DATA TOW,TOY,TRY,TUB,TUG,TWO,URN
5530 DATA USE,UPS,VAN,VAT,VEX,VIA,VIE
5540 DATA VIM,VOW,YAK,YAM,YEN,YES,YET
5550 DATA YOU,WAD,WAG,WAN,WAR,WAS,WAX
5560 DATA WAY,WEB,WED,WET,WHO,WHY,WIG
5570 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 line 500 to read:

    ```
    500 GOSUB 1000:PRINT:PRINT
        "YOUR GUESS (OR Q)";:INPUT P$
    ```

2. The maximum number of guesses allowed, M, can be changed in line 140. 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 M to 15, change line 140 to read

    ```
    140 M=15:N=406
    ```

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 140, N is set to be the total number of words in the data list. The data list itself is from line 5000 on.

    To add word(s), do the following. Enter them in data statements after the current data (use line numbers larger than 5570). Then redefine the value of N to be 406 plus the number of new words added. For example, to add the words "ohm" and "yaw" into the list, change line 140 to read

    ```
    140 M=25:N=408
    ```

and add a new line

```
5580 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

140– 150	Defines variables, dimensions arrays.
200– 330	Initializes new game.
500– 570	Human guesses at the computer's word.
600– 680	ATARI guesses.
800– 870	Evaluates human's possible secret words. Moves them to the front of A$ array.
900– 950	Processes inconsistency in given information.
1000–1110	Displays the current summary table.
1200–1230	Inquires about another game.
1500–1610	Compares a guess with a key word.
1800–1830	Checks if input word is legal.
2200–2240	Shuffles A$ array randomly.
2400	Swaps elements A and B in the A$ array.
2600–2610	Breaks word Q$ into separate letters.
3000	Fill A$ array from data.
3200–3220	Post-mortem after ATARI wins.
3400–3460	Post-mortem after human wins.
3600–3610	Error routine—too many guesses.
5000–5570	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$, Q$, XX$	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, PX	Temporaries; array and loop indices.
K	Formatting variable for the summary display.
PK	Current row for printing.

SUGGESTED PROJECTS

1. Additional messages during the course of the game can personify the program even more. After the ATARI 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

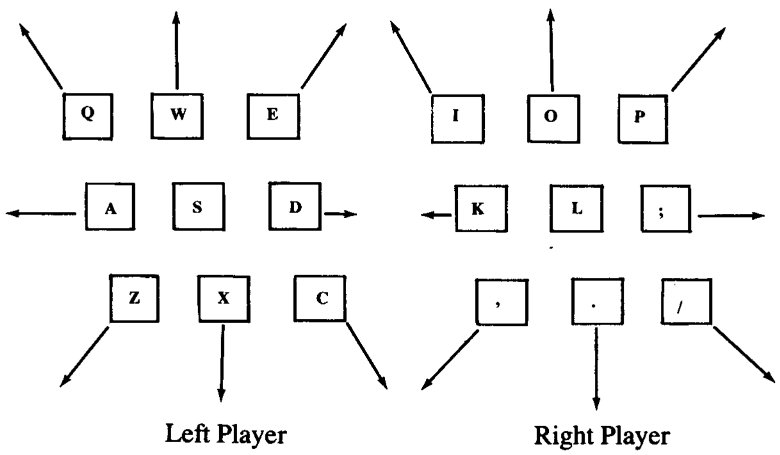
PURPOSE

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. The names can be up to eight characters long. Then it displays the playing field, shows the starting point for each player, and tells you to press any key to start. Pressing the **ESCAPE** or **Q** key ends the game.

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 players can change direction in one of eight directions as shown in the following Figure:



Arrows Indicate Direction of Motion

Find these keys on the ATARI keyboard and you will see the logic behind these choices.

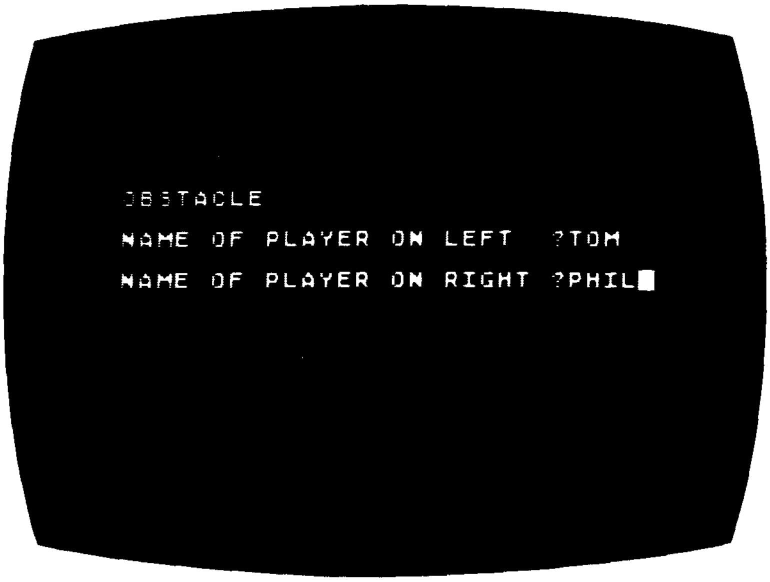
The first time either player bumps into the wall surrounding the playing field or the obstacle wall built by either player, he loses. When this happens the program indicates the point of impact for a few seconds and displays the name of the winner. Then the game starts over.

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.

Be aware that if you plan a move properly, there are ways to cross over certain obstacles. It's not hard to cross a diagonal obstacle when moving diagonally, for example.

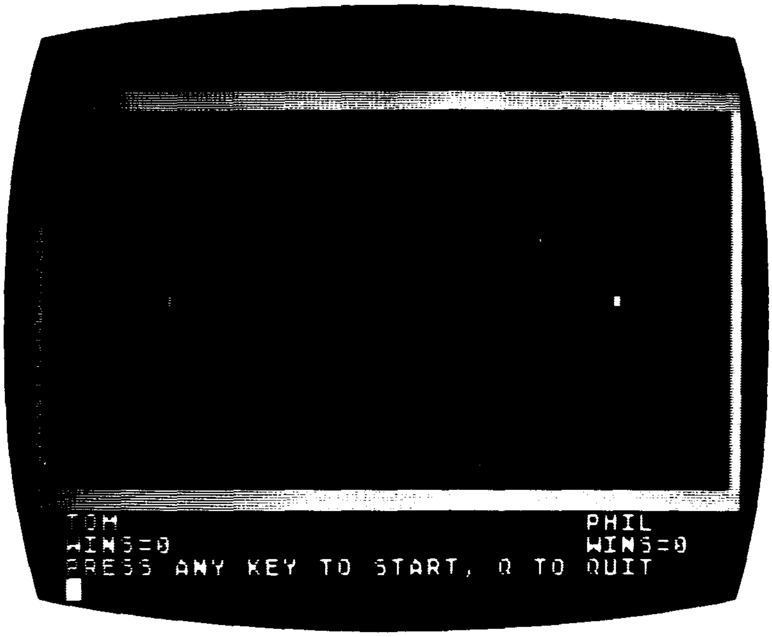
When pressing a key to change direction, be sure to press it quickly and release it. *Do not* hold a key down – you might inhibit the computer from recognizing a move your opponent is trying to make. Once in a while, only one key will be recognized when two are hit at once.

SAMPLE RUN



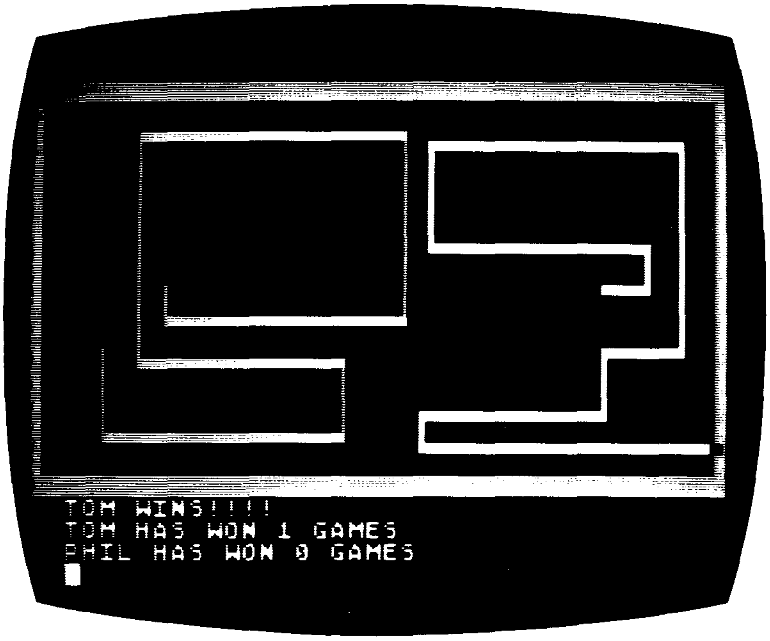
```
OBSTACLE
NAME OF PLAYER ON LEFT  ?TOM
NAME OF PLAYER ON RIGHT ?PHIL
```

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



```
TOM                                 PHIL
WINS=0                              WINS=0
PRESS ANY KEY TO START, Q TO QUIT
```

The program draws the playing field and waits for a key to be pressed.



```
TOM WINS!!!!
TOM HAS WON 1 GAMES
PHIL HAS WON 0 GAMES
```

The program starts both players moving in random directions. Phil (on the right) doesn't change direction soon enough and crashes into the wall, making Tom the winner.

PROGRAM LISTING

```
10 REM OBSTACLE
15 REM COPYRIGHT 1984 DILITHIUM PRESS
20 DIM A(70),AN$(8),BN$(8),R$(10),XX$(10)
30 GOSUB 1300
120 GOSUB 600
125 GRAPHICS 5:SOUND 0,0,0,0
130 IF LEN(AN$)<5 THEN PRINT AN$;CHR$(127);CHR$(
127);CHR$(127);CHR$(127);BN$:GOTO 140
135 PRINT AN$;CHR$(127);CHR$(127);CHR$(127);BN$
140 PRINT "WINS=";L;CHR$(127);CHR$(127);CHR$(127
);"WINS=";M
150 PRINT "PRESS ANY KEY TO START, Q TO QUIT"
155 Z=1:F=764:S=255:T=40
160 AX=15:AY=20:BX=65:BY=20:A=1:B=2
165 E=1:AD=INT(4*RND(1))+1:BD=INT(4*RND(1))+1
170 GOSUB 950:GOSUB 900
175 POKE F,S
180 C=PEEK(F):IF C>128 THEN 180
190 POKE F,S
```

```
195 IF C=28 OR C=47 THEN GRAPHICS 0:END
200 GOSUB 900
210 X=AX:Y=AY:D=AD:GOSUB 1000
215 IF AD>8 THEN AD=AD-8
220 AR=R:AX=X:AY=Y
230 X=BX:Y=BY:D=BD:GOSUB 1000
235 IF BD>8 THEN BD=BD-8
240 BR=R:BX=X:BY=Y
245 IF AR=1 OR BR=1 THEN 400
250 GOSUB 900
260 C=PEEK(F):POKE F,S
270 IF C=255 OR C>70 THEN 210
280 JJ=A(C)
290 IF JJ=0 THEN 210
300 IF JJ=17 THEN BD=BD+8:GOTO 210
310 IF JJ=18 THEN AD=AD+8:GOTO 210
320 IF JJ<9 THEN AD=JJ:GOTO 210
330 IF JJ>8 THEN BD=JJ-8:GOTO 210
400 GOSUB 700:X=AX:Y=AY
410 IF AR=1 THEN 420
415 X=BX:Y=BY
420 FOR J=1 TO 100
430 COLOR Z:PLOT X,Y
440 FOR K=1 TO 10:NEXT K
450 COLOR 0:PLOT X,Y
460 FOR K=1 TO 10:NEXT K
470 NEXT J
490 GOTO 125
600 GRAPHICS 0:PRINT "OBSTACLE":PRINT
610 PRINT "NAME OF PLAYER ON LEFT  ";:INPUT AN$
620 PRINT
630 PRINT "NAME OF PLAYER ON RIGHT ";:INPUT BN$
650 RETURN
700 PRINT
710 SOUND 0,100,10,10:FOR J=1 TO 200:NEXT J:SOUN
D 0,0,0,0
720 IF AR=0 OR BR=0 THEN 740
730 PRINT "YOU BOTH LOSE!!!!!":RETURN
740 R$=AN$:IF AR=1 THEN R$=BN$
750 IF R$=AN$ THEN L=L+1
760 IF R$=BN$ THEN M=M+1
770 PRINT R$;" WINS!!!!"
780 PRINT AN$;" HAS WON ";L;" GAMES":PRINT BN$;"
 HAS WON ";M;" GAMES"
790 RETURN
900 COLOR A:PLOT AX,AY
910 COLOR B:PLOT BX,BY
915 FOR J=1 TO 100:NEXT J
920 RETURN
940 GRAPHICS 5
950 COLOR E
```

```
960 PLOT 0,0:DRAWTO 79,0:DRAWTO 79,39:DRAWTO 0,3
9:DRAWTO 0,1:DRAWTO 78,1
980 DRAWTO 78,38:DRAWTO 1,38:DRAWTO 1,1
990 RETURN
1000 ON D GOTO 1010,1020,1030,1040,1050,1060,107
0,1080,1090,1100,1110,1120,1130,1140,1150,1160
1010 Y=Y-1:GOTO 1200
1020 Y=Y+1:GOTO 1200
1030 X=X-1:GOTO 1200
1040 X=X+1:GOTO 1200
1050 Y=Y-1:X=X-1:GOTO 1200
1060 Y=Y-1:X=X+1:GOTO 1200
1070 Y=Y+1:X=X+1:GOTO 1200
1080 Y=Y+1:X=X-1:GOTO 1200
1090 Y=Y-2:GOTO 1200
1100 Y=Y+2:GOTO 1200
1110 X=X-2:GOTO 1200
1120 X=X+2:GOTO 1200
1130 Y=Y-2:X=X-2:GOTO 1200
1140 Y=Y-2:X=X+2:GOTO 1200
1150 Y=Y+2:X=X+2:GOTO 1200
1160 Y=Y+2:X=X-2:GOTO 1200
1200 LOCATE X,Y,W
1210 R=0:IF W<>0 THEN R=1
1220 RETURN
1300 FOR J=0 TO 70:A(J)=0:NEXT J
1310 A(47)=5:A(46)=1:A(42)=6
1320 A(63)=3:A(58)=4:A(23)=8
1330 A(22)=2:A(18)=7:A(10)=14
1340 A(2)=12:A(38)=15:A(34)=10
1350 A(32)=16:A(5)=11:A(13)=13
1360 A(8)=9
1370 RETURN
```

EASY CHANGES

1. To speed the game up, change the 100 in line 915 to a 50 or so. To slow it down, make it 150 or 200.
2. To make both players always start moving upward at the beginning of each game (instead of in a random direction), insert the following statement:

   168 AD=1:BD=1

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

   168 AD=4:BD=3

3. To change the length of time that the final messages are displayed after each game, modify line 420. Change the 100 to 50 (or so) to shorten it, or to 150 to lengthen it.

MAIN ROUTINES

10– 170	Initializes variables. Gets players' names. Displays titles, playing field.
175– 200	Waits for key to be pressed to start game. Redisplays playing field.
210– 250	Makes move for player A (on left side) and B (on right). Saves results.
260– 330	Accepts moves from keyboard and translates direction.
400– 490	Displays winner's name. Goes back to start next game.
600– 650	Subroutine that gets each player's name.
700– 790	Subroutine that displays winner's name.
900– 920	Subroutine that displays each graphics character of each player's obstacle on the screen.
950– 990	Subroutine that displays playing field.
1000–1220	Subroutine that moves marker and determines if space moved to is empty.
1300–1370	Subroutine that defines key directions.

MAIN VARIABLES

A	Array of key direction values.
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.
F	Peek address for reading keyboard.
S	Poke address for resetting keyboard.
T	Length of edge of playing field.
AD, BD	Current direction in which A and B are going (1=up, 2=down, 3=left, 4=right, etc.).
E	Graphics color for edge of playing field.
C	Character being read from keyboard.
X, Y	Temporary position on screen.
D	Temporary direction.
R	Result of move (0=okay, 1=loser).
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.
XX$	Work string.
L, M	Player A and B's number of games won.
W	Numeric value of LOCATE graphics character.
JJ	Direction value from A array.

SUGGESTED PROJECTS

1. 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.
2. 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.
3. Modify the program to use the ATARI joystick controllers instead of keyboard keys to control player motions.

ROADRACE

PURPOSE

Imagine yourself at the wheel of a high-speed race car winding your way along a treacherous course. The road curves unpredictably. To stay on course, you must steer accurately or risk collision. How far can you go in one day? How many days will it take you to race cross-country? 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

The program begins with a short color graphics display. It then asks you for two inputs – road width and visibility. The road width (in characters) can be anywhere between 4 and 12. 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 "terrible" to "good." When visibility is good, the car appears low on the screen. This allows a good view of the twisting road ahead. When visibility is poor, the car appears high on the screen allowing only a brief look at the upcoming road.

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. The road moves continually down the screen. Its twists and turns are controlled randomly. You must steer the car accurately to keep it on track.

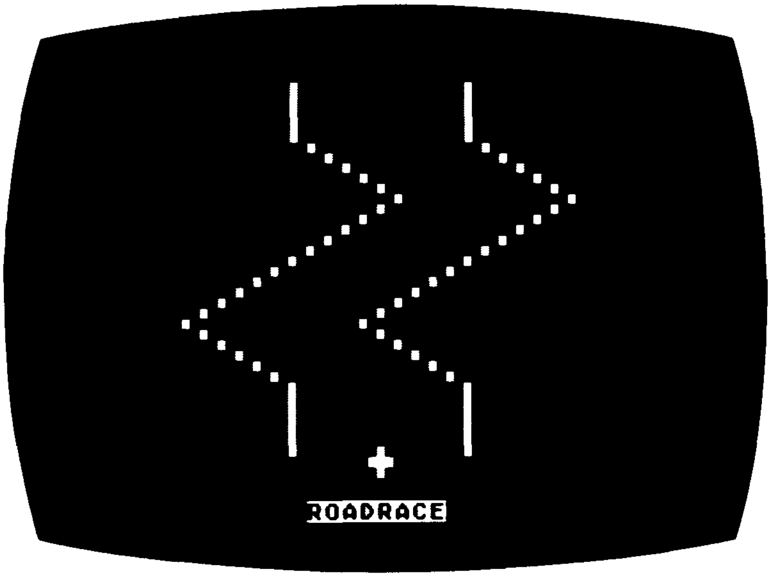
The car is controlled with the use of two keys near the right of the keyboard. Pressing the plus key (+) will cause the car to move to the left while pressing the asterisk key (*) will cause the car to move to the right. These keys were chosen because they have the left and right arrows on them. Doing neither will cause the car to continue straight up.

The race proceeds until the car goes "off the road." Turn up the sound on your TV and you'll hear the crash. 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 and the average miles per day.

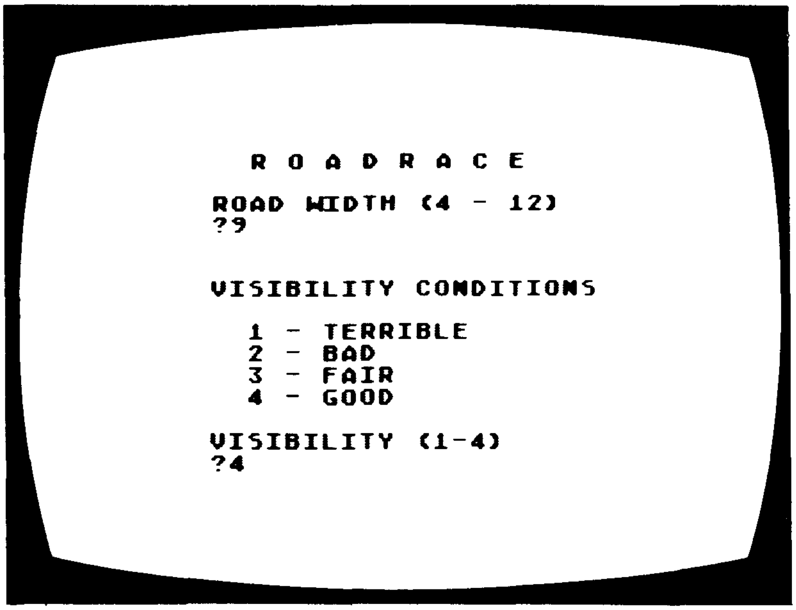
After each collision, you can proceed by pressing either **C**, **R**, or **Q**. Selecting **C** will continue the race for another day with the same road conditions. Cumulative totals will be retained. **R** will restart the race. This allows changing the road conditions and initializing back to day one. **Q** simply quits the race and returns the ATARI back to direct BASIC.

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 3000 miles for a cross-country trip. 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

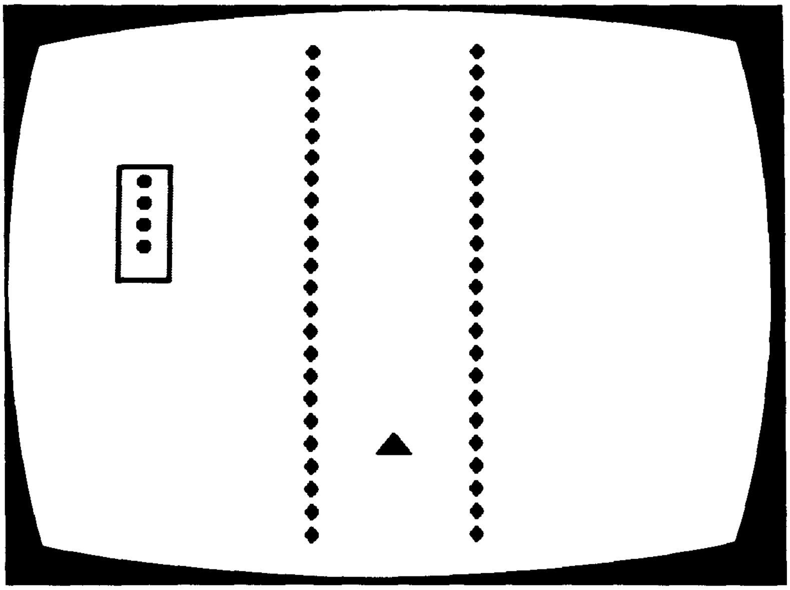


The program displays its logo.

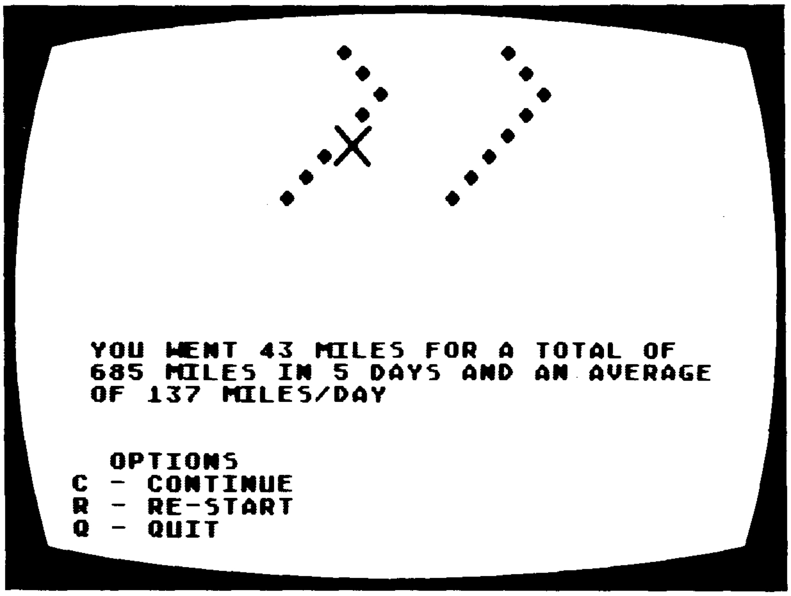


```
  R O A D R A C E

ROAD WIDTH (4 - 12)
?9

VISIBILITY CONDITIONS

  1 - TERRIBLE
  2 - BAD
  3 - FAIR
  4 - GOOD

VISIBILITY (1-4)
?4
```

The operator selects good visibility and a course-width of nine characters. Now the race is ready to begin.



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 43 MILES FOR A TOTAL OF
685 MILES IN 5 DAYS AND AN AVERAGE
OF 137 MILES/DAY


  OPTIONS
C - CONTINUE
R - RE-START
Q - QUIT
```

The operator, steering the car from the keyboard, finally crashes. A distance of 43 miles is obtained on this leg for a total of 685 miles in 5 days (legs). The options for continuing are displayed while the program waits for the operator's choice.

PROGRAM LISTING

```
10 REM ROADRACE
15 REM COPYRIGHT 1984 DILITHIUM PRESS
110 DIM A$(40),B$(40),K$(2),R$(10),N$(1)
120 GRAPHICS 5:SETCOLOR 2,0,0:POKE 82,0:POKE 752
,1:POKE 764,255
125 LC=0.4
130 X=15:P=20:W=10:V=8:RC=1-LC:LM=4:RM=36
140 MR=7:ML=6
150 K$(1)=CHR$(8):K$(2)=CHR$(10)
160 N$=CHR$(32):B$=CHR$(32):B$(40)=B$:B$(2)=B$
170 T=0:D=0:GOSUB 670
180 A$=B$
190 Y=RND(0)
200 IF Y<LC THEN X=X-1:GOTO 220
210 IF Y>RC THEN X=X+1
220 IF X<LM THEN X=4:GOTO 240
230 IF X>RM-W THEN X=RM-W
240 A$(X)=CHR$(96):A$(X+W)=CHR$(96)
250 POSITION 0,0:PRINT CHR$(157);A$
260 POSITION P,V+1:PRINT N$;N$
270 S=PEEK(764):IF S=255 THEN 310
280 IF S=ML THEN P=P-1:GOTO 300
290 IF S=MR THEN P=P+1
300 POKE 764,255
310 LOCATE P,V,A:LOCATE P+1,V,Z
320 IF A<>32 OR Z<>32 THEN 360
330 POSITION P,V:PRINT K$
340 M=M+1:T=T+1
350 GOTO 180
360 D=D+1:POSITION P,V:PRINT CHR$(7);CHR$(6)
370 POSITION P,V+1:PRINT CHR$(6);CHR$(7)
380 FOR J=-14 TO 0 STEP 2
390 SOUND 0,200,4,14-ABS(J)
400 SOUND 1,255,4,14-ABS(J)
410 SOUND 2,255,4,14-ABS(J)
420 SOUND 3,150,4,14-ABS(J)
430 FOR K=1 TO 15:NEXT K
440 NEXT J
450 FOR J=0 TO 14
460 SOUND 0,200,4,14-J
470 SOUND 1,255,4,14-J
480 SOUND 2,255,4,14-J
490 SOUND 3,150,4,14-J
500 FOR K=1 TO 50:NEXT K
510 NEXT J
520 FOR J=0 TO 3:SOUND J,0,0,0:NEXT J
530 POSITION 0,22:PRINT B$:PRINT B$:PRINT B$:POK
E 764,255
540 R$=" DAYS":IF D=1 THEN R$=" DAY"
```

```
550 PRINT N$;N$;"YOU WENT";N$;M;N$;"MILES FOR A
TOTAL OF";N$
555 PRINT N$;N$;T;N$;"MILES IN";N$;D;R$;N$;"AND
AN AVERAGE"
560 PRINT N$;N$;"OF";N$;INT((T/D)*100)/100;N$;"M
ILES/DAY"
570 PRINT B$:PRINT N$;N$;N$;"OPTIONS"
580 PRINT " C - CONTINUE"
590 PRINT " R - RE-START"
600 PRINT " Q - QUIT"
610 POKE 764,255
620 S=PEEK(764):IF S=255 THEN 620
630 POKE 764,255:IF S=47 THEN 660
640 IF S=40 THEN M=0:T=0:D=0:GOSUB 770:GOTO 180
645 IF S<>18 THEN 610
650 PRINT CHR$(125):M=0:GOSUB 940:GOTO 180
660 GRAPHICS 0:POKE 82,2:END
670 COLOR 1
680 X=28:Y=1
690 FOR J=1 TO 6:Y=Y+1:PLOT X,Y:PLOT X+W*2,Y:NEX
T J
700 FOR J=1 TO 6:X=X+2:Y=Y+1:PLOT X,Y:PLOT X+W*2
,Y:NEXT J
710 FOR J=1 TO 12:X=X-2:Y=Y+1:PLOT X,Y:PLOT X+W*
2,Y:NEXT J
720 FOR J=1 TO 6:X=X+2:Y=Y+1:PLOT X,Y:PLOT X+W*2
,Y:NEXT J
730 FOR J=1 TO 6:Y=Y+1:PLOT X,Y:PLOT X+W*2,Y:NEX
T J
740 X=X+W:PLOT X,Y:Y=Y+1:PLOT X-1,Y:PLOT X,Y:PLO
T X+1,Y:PLOT X,Y+1
750 PRINT :FOR J=1 TO 15:PRINT N$;:NEXT J:RESTOR
E :FOR J=1 TO 8:READ K:PRINT CHR$(K);:NEXT J
755 DATA 210,207,193,196,210,193,195,197
760 FOR J=1 TO 1000:NEXT J
770 GRAPHICS 0:SETCOLOR 2,1,12:SETCOLOR 1,0,0:PO
KE 82,2:POKE 752,1
780 PRINT :PRINT :PRINT
790 PRINT N$;N$;"R O A D R A C E"
800 PRINT
810 PRINT "ROAD WIDTH (4 - 12) ":INPUT W:W=INT(W
)
820 IF W<4 OR W>12 THEN PRINT :PRINT "** ILLEGAL
 ENTRY **":GOTO 800
830 PRINT :PRINT :PRINT "VISIBILITY CONDITIONS"
840 PRINT
850 PRINT "  1 - TERRIBLE"
860 PRINT "  2 - BAD"
870 PRINT "  3 - FAIR"
```

```
880 PRINT "  4 - GOOD"
890 PRINT
900 PRINT "VISIBILITY (1-4) ":INPUT V:V=INT(V)
910 IF V<1 OR V>4 THEN PRINT :PRINT "** ILLEGAL
ENTRY **":GOTO 890
920 V=V*4+2
930 POKE 82,0
940 X=14:P=X+W/2-2
950 A$=B$:A$(X)=CHR$(96):A$(X+W)=CHR$(96)
960 PRINT CHR$(125):FOR J=1 TO 23:PRINT A$:NEXT
J
970 POSITION P,V:PRINT K$
980 POSITION 3,5:PRINT CHR$(14);CHR$(14);CHR$(14
)
990 POSITION 3,6:PRINT CHR$(22);N$;CHR$(2)
1000 POSITION 3,7:PRINT CHR$(22);N$;CHR$(2)
1010 POSITION 3,8:PRINT CHR$(22);N$;CHR$(2)
1020 POSITION 3,9:PRINT CHR$(22);N$;CHR$(2)
1030 POSITION 3,10:PRINT CHR$(22);N$;CHR$(2)
1040 POSITION 3,11:PRINT CHR$(13);CHR$(13);CHR$(
13)
1050 FOR K=1 TO 5
1060 FOR J=1 TO 200:NEXT J
1070 POSITION 4,5+K:PRINT CHR$(20)
1080 NEXT K
1090 RETURN
```

EASY CHANGES

1. The amount of windiness in the road can be adjusted by changing the value of LC in line 125. Maximum windiness is achieved with a value of 0.5 for LC. To get a straighter road, make LC smaller. A value of 0 will produce a completely straight road. LC should lie between 0 and 0.5. To get a somewhat more winding road, you might change line 125 to read

   ```
   125 LC=0.45
   ```

2. Experiment with a different color background by changing the value in line 670 and the SETCOLOR argument in line 770. Try

   ```
   670 COLOR 2
   770 GRAPHICS 0:SETCOLOR 2,6,12:SETCOLOR
       1,0,0:POKE 82,2:POKE 752,1
   ```

3. The keys which cause the car to move left and right can be easily changed. You may wish to do this if you are left handed or find that two widely separated keys would be more convenient. The changes are to be made in line 140. The constants that are assigned to the variables ML and MR control which keys will cause the car to move left and right respectively. To find the correct constants, first type in this short program:

   ```
   10 Q=PEEK (764):IF Q=255 THEN 10
   20 POKE 764,255:PRINT Q
   ```

   Run the program, then press the key you wish to use to make the car move left. A number will appear on the screen. This number will be assigned to the variable ML. Run the program again and hit the key you want to cause the car to move right. Assign the number that now appears to the variable MR. If, for example, you want 1 to cause a left move and 9 to cause a right move, line 140 will become

   ```
   140 ML=31:MR=48
   ```

4. Instead of the keyboard, a joystick can be used to control the car's movements. We'll assume stick 0 will be used. Now make the following changes to the main program:

   ```
   140 ML=11:MR=7
   270 S=STICK(0)
   280 IF S=ML THEN P=P-1
   290 IF S=MR THEN P=P+1
   ```

The stick will now control the car. When the stick is neutral the car will continue straight up the road. If the stick is moved to the left, the car will *continue* to move left. If the stick is moved to the right, the car will *continue* to move right.

MAIN ROUTINES

110- 170	Variable initialization.
180- 260	Draws next road segment.
270- 350	Updates the car position.
360- 660	Processes end of race day.
670- 790	Routine to do initial graphics display.
800- 930	Gets road conditions from user.
940- 970	Initializes road.
980-1090	Routine to display starting light.

MAIN VARIABLES

P	Current horizontal position of car.
W	Road width.
V	Visibility, vertical position of car.
M	Miles driven on current day.
D	Number of days of the race.
T	Total miles driven for whole race.
ML, MR	Peek constants to move road left, right.
LC, RC	Random value cutoff to move road left, right.
LM, RM	Leftmost, rightmost allowable road position.
R$	Work string.
X, Y	Horizontal, vertical position of road.
J, K	Loop indices and work variables.
A$, B$	Screen lines.
A, Z	Numeric values of LOCATE graphic characters.
S	Peek argument.
K$	String representation of car.
N$	String of one blank character.

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. The coding can be done between lines 560 and 570.
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. This can be done with appropriate statements after line 240. The program will recognize a collision if the car moves into any non-blank square.

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.

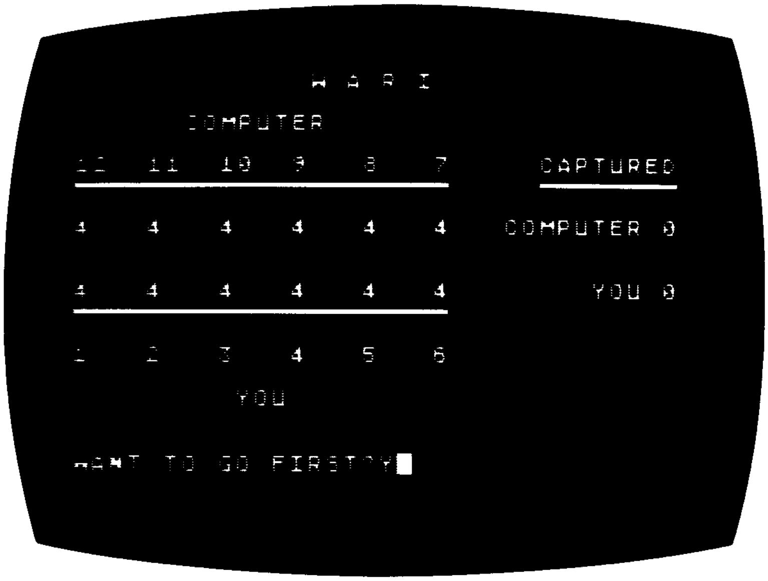
There are a few special rules to cover some situations that can come up in the game. It is not legal to capture all the stones on the opponent's side of the board, since this would leave the opponent with no moves on his next turn. By the same token, when your opponent has no stones on his side (because 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 forty-five 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).

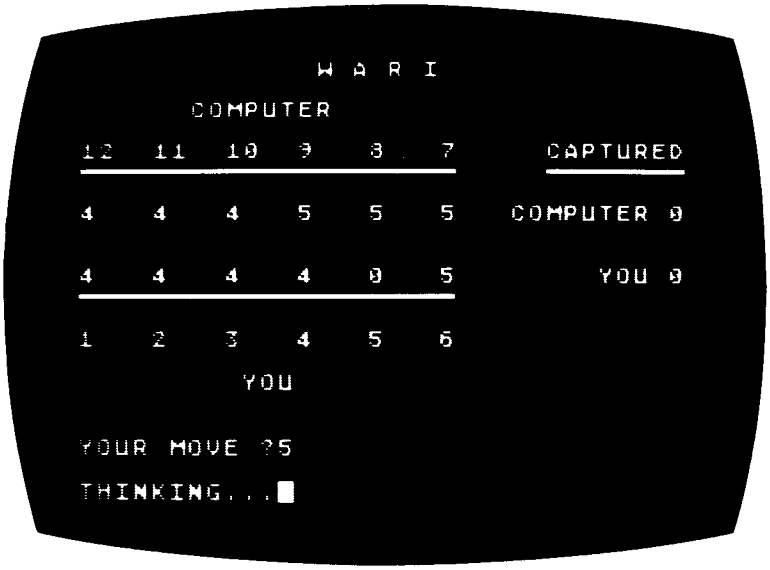
Entering the word "END" instead of a square number ends the game.

SAMPLE RUN



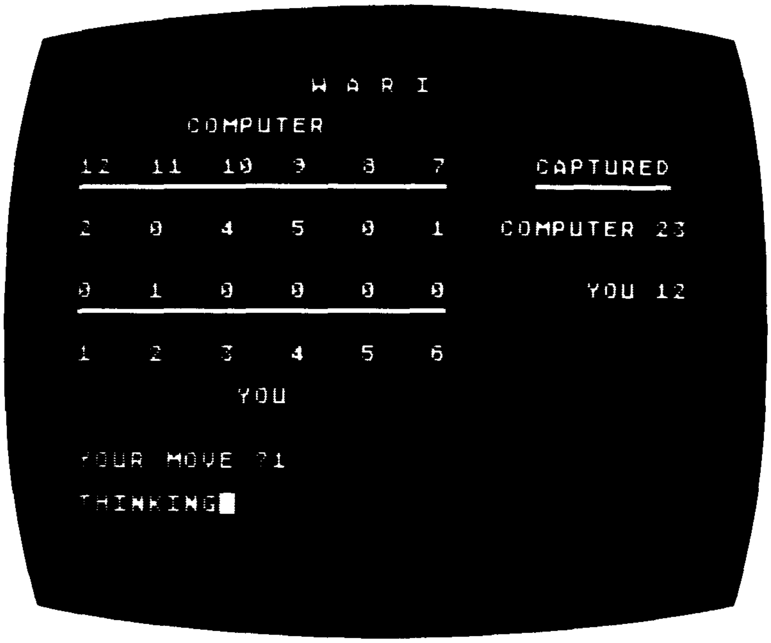
```
                 W A R I
       COMPUTER
12   11   10   9    8    7      CAPTURED
______________________________  ________
4    4    4    4    4    4      COMPUTER 0

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.



```
                 W A R I
       COMPUTER
12   11   10   9    8    7      CAPTURED
______________________________  ________
4    4    4    5    5    5      COMPUTER 0

4    4    4    4    0    5           YOU 0
______________________________
1    2    3    4    5    6
             YOU

YOUR MOVE ?5
THINKING...
```

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



```
                W A R I
         COMPUTER
12   11   10   9    8    7      CAPTURED

2    0    4    5    0    1    COMPUTER 23

0    1    0    0    0    0          YOU 12

1    2    3    4    5    6
             YOU

YOUR MOVE ?1

THINKING
```

Later in the same game, the computer is about to move square 9, which will capture 2 or more stones and win the game.

PROGRAM LISTING

```
10 REM WARI
15 REM COPYRIGHT 1984 DILITHIUM PRESS
120 J=1:K=1:Q=14:P=13:F=50:D=12
130 DIM T(Q),Y(Q),W(Q),V(6),E(6),B(Q)
135 DIM R$(10),C$(24),D$(40)
140 ZB=RND(1):ZB=ZB/Q:ZA=0.25+ZB:ZB=0.25-ZB:I=1:
GOSUB 750
150 FOR J=1 TO D:B(J)=4:NEXT J:B(P)=0:B(Q)=0:MN=
0:GOSUB 1200:GOSUB 900
160 GOSUB 990:PRINT "WANT TO GO FIRST";:INPUT R$
165 IF LEN(R$)=0 THEN 160
170 GOSUB 990:PRINT D$:IF R$(1,1)="Y" THEN 250
180 IF R$(1,1)<>"N" THEN 160
190 GOSUB 1050:PRINT D$;D$;D$;:GOSUB 1050:PRINT
"THINKING";:GOSUB 510
195 IF M<1 THEN 2000
200 GOSUB 1050:PRINT D$;:GOSUB 1050:PRINT "MY MO
VE IS ";M
210 FOR J=1 TO Q:T(J)=B(J):NEXT J:GOSUB 350
220 FOR J=1 TO Q:B(J)=T(J):NEXT J:GOSUB 900
```

```
230 IF B(Q)<24 THEN 250
240 POSITION 3,22:PRINT "** I WIN **":GOTO 810
250 GOSUB 990:PRINT D$;D$:GOSUB 990:PRINT "YOUR
MOVE ";:INPUT R$
255 IF R$="END" THEN 840
260 R$=R$(1,1):IF R$<"1" OR R$>"6" THEN 330
265 M=VAL(R$)
270 FOR J=1 TO Q:T(J)=B(J):NEXT J
280 GOSUB 350:IF M<0 THEN 330
290 FOR J=1 TO Q:B(J)=T(J):NEXT J
300 MN=MN+1:GOSUB 900
310 IF B(P)<24 THEN 190
320 GOSUB 1050:PRINT "** YOU WIN **";D$:GOTO 810
330 POSITION 19,19:PRINT " ILLEGAL ";:FOR J=1 TO
 1000:NEXT J:GOTO 250
350 IF T(M)=0 THEN M=-1:RETURN
360 R$="H":IF M>6 THEN R$="C":GOTO 380
370 FOR J=1 TO Q:Y(J)=T(J):NEXT J:GOTO 400
380 FOR J=1 TO 6:Y(J)=T(J+6):Y(J+6)=T(J):NEXT J
390 Y(P)=T(Q):Y(Q)=T(P):M=M-6
400 C=M:N=Y(C):FOR J=1 TO N:C=C+1
410 IF C=P THEN C=1
420 IF C=M THEN C=C+1:GOTO 410
430 Y(C)=Y(C)+1:NEXT J:Y(M)=0:L=C
440 IF L<7 OR Y(L)>3 OR Y(L)<2 THEN 460
450 Y(P)=Y(P)+Y(L):Y(L)=0:L=L-1:GOTO 440
460 S=0:FOR J=7 TO D:S=S+Y(J):NEXT J
470 IF S=0 THEN M=-2:RETURN
480 IF R$="H" THEN FOR J=1 TO Q:T(J)=Y(J):NEXT J
:RETURN
490 FOR J=1 TO 6:T(J)=Y(J+6):T(J+6)=Y(J):NEXT J
500 T(Q)=Y(P):T(P)=Y(Q):RETURN
510 FOR A=1 TO 6:M=A+6:IF B(M)=0 THEN E(A)=-F:GO
TO 690
530 FOR J=1 TO Q:T(J)=B(J):NEXT J:GOSUB 350
540 IF M<0 THEN E(A)=-F:GOTO 690
550 IF T(Q)>23 THEN M=A+6:RETURN
560 FOR J=1 TO Q:W(J)=T(J):NEXT J:FOR K=1 TO 6
570 IF T(K)=0 THEN V(K)=F:GOTO 670
580 FOR J=1 TO Q:T(J)=W(J):NEXT J:M=K:GOSUB 350
590 IF M<0 THEN V(K)=F:GOTO 670
600 FA=0:FB=0.05:FC=0:FD=0:FOR J=7 TO D
610 FB=FB+T(J):IF T(J)>0 THEN FA=FA+1
620 IF T(J)<3 THEN FC=FC+1
630 IF T(J)>FD THEN FD=T(J)
640 NEXT J:FE=FB:FOR J=1 TO 6:FE=FE+T(J):NEXT J
650 FA=FA/6:FD=1-FD/FB:FC=1-FC/6:FB=FB/FE
660 V(K)=ZA*(FA+FB)+ZB*(FC+FD)+T(Q)+B(P)-B(Q)-T(
P)
670 NEXT K:E(A)=F:FOR J=1 TO 6:IF V(J)<E(A) THEN
 E(A)=V(J)
```

```
680 NEXT J
690 PRINT ".";:NEXT A:M=0:FA=-F:FOR J=1 TO 6
700 IF E(J)>FA THEN FA=E(J):M=J+6
710 NEXT J:RETURN
750 C$=CHR$(18):FOR J=1 TO 21:C$(J,J)=CHR$(18):N
EXT J
790 D$=CHR$(32):FOR J=1 TO 5:XX=LEN(D$):D$(XX+1,
XX+XX)=D$:NEXT J
800 RETURN
810 PRINT "GOOD GAME"
840 PRINT :PRINT "DO YOU WANT TO PLAY AGAIN";:IN
PUT R$
845 IF LEN(R$)=0 THEN 840
850 IF R$(1,1)="Y" THEN 140
860 IF R$(1,1)<>"N" THEN 840
870 PRINT "SEE YOU LATER"
880 PRINT :END
900 POSITION 2,8
920 FOR J=0 TO 5:POSITION 4*J+3,8:PRINT B(12-J);
:IF B(12-J)=0 THEN GOSUB 1100
930 NEXT J:POSITION 27,8:PRINT "COMPUTER ";B(Q)
940 FOR J=0 TO 5
950 POSITION 4*J+3,11:PRINT B(J+1);:IF B(J+1)=0
THEN GOSUB 1100
960 NEXT J:POSITION 32,11:PRINT "YOU ";B(P)
970 RETURN
990 POSITION 3,19:RETURN
1050 POSITION 3,21:RETURN
1100 PRINT CHR$(32);:RETURN
1200 GRAPHICS 0:POSITION 16,1:PRINT "W A R I"
1210 PRINT
1220 POSITION 9,3:PRINT "COMPUTER":PRINT
1230 FOR J=0 TO 5:POSITION 4*J+3,5:PRINT 12-J;:N
EXT J
1240 POSITION 29,5:PRINT "CAPTURED":PRINT CHR$(3
2);C$;:POSITION 29,6:PRINT C$(1,8)
1250 PRINT :PRINT :PRINT :PRINT :PRINT :PRINT CH
R$(32);C$:PRINT
1260 FOR J=0 TO 5:POSITION 4*J+3,14:PRINT J+1;:N
EXT J
1270 POSITION 12,16:PRINT "YOU"
1280 PRINT :RETURN
2000 PRINT "NO LEGAL MOVES."
2010 PRINT "GAME IS A DRAW."
2020 GOTO 840
```

EASY CHANGES

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

```
555 GOTO 600
665 E(A)=V(K):GOTO 690
```

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. Change line 690 so it looks like this:

```
690 PRINT E(A);"/";:NEXT A:M=0:FA=-F:
    FOR J=1 TO 6
```

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- 150	Initializes variables. Displays board.
160- 180	Asks who goes first. Evaluates answer.
190- 220	Determines computer's move. Displays new board position.
230- 240	Determines if computer's move resulted in a win. Displays a message if so.
250- 300	Gets operator's move. Checks for legality. Displays new board position.
310- 320	Determines if operator's move resulted in a win.
330	Displays message if illegal move attempted.
350- 500	Subroutine to make move M in T array.
360- 390	Copies T array into Y array (inverts if computer is making the move).
400- 430	Makes move in Y array.
440- 450	Checks for captures. Removes stones. Checks previous square.

460- 470	Sees if opponent is left with a legal move.
480- 500	Copies Y array back into T array.
510- 710	Subroutine to determine computer's move.
750- 800	Subroutine to create graphics strings for board display.
810- 880	Displays ending message. Asks about playing again.
900- 970	Subroutine to display stones on board and captured.
990	Subroutine to move cursor to "YOUR MOVE" position on screen.
1050	Subroutine to move cursor to "MY MOVE" position on screen.
1100	Subroutine to display one blank character.
1200-1280	Subroutine to display Wari board (without stones).
2000-2020	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.
J, K	Loop indices.
N, XX	Work variables.
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.
C$	String of graphics characters used to display the Wari board.
D$	String of 32 blanks.

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 300 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 600 through 660 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

INTRODUCTION TO GRAPHICS DISPLAY PROGRAMS

The ATARI computers are amazing machines. They have very useful graphics capabilities in addition to their 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 ATARI'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 ATARI. This program will keep you entertained with an example of computer animation.

KALEIDO

PURPOSE

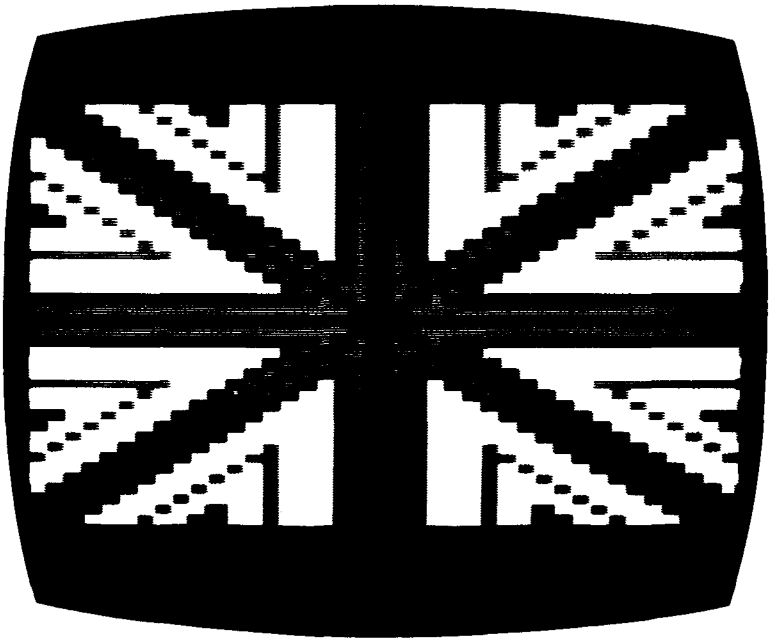
If you have ever played with a kaleidoscope, you were probably fascinated by the endless symmetrical patterns you saw displayed. This program creates a series of kaleidoscope-like designs, with each one overlaying the previous one.

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 ATARIs over (all your friends *do* have ATARIs, 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

```
10 REM KALEIDO
15 REM COPYRIGHT 1984 DILITHIUM PRESS
20 GOTO 100
50 COLOR R(0):PLOT X*2,Y*4:DRAWTO X*2,Y*4+3:PLOT
 X*2+1,Y*4+3:DRAWTO X*2+1,Y*4
52 IF J=1 THEN RETURN
55 GOTO 900
60 COLOR R(N):PLOT X2*2,Y2*4:DRAWTO X2*2,Y2*4+3:
PLOT X2*2+1,Y2*4+3:DRAWTO X2*2+1,Y2*4
65 RETURN
100 DIM R(16),ZY(8)
105 FOR J=1 TO 8:ZY(J)=0:R(J)=0:NEXT J
110 GRAPHICS 10:POKE 704,0
120 GOSUB 3000
125 P=19:A=19:B=19:D=-1
130 M=8
```

```
150 FOR J=1 TO 8
160 R(J)=INT(RND(1)*8)+1
164 ZA=0
165 FOR T=1 TO 8:IF T=J THEN 167
166 IF R(J)=R(T) THEN ZA=1
167 NEXT T
168 IF ZA=1 THEN 160
170 NEXT J
180 D=-D:K=1:L=19:IF D>0 THEN 200
190 K=19:L=1
200 FOR J=K TO L STEP D
210 X=A+J:Y=B:GOSUB 50
220 X=A-J:GOSUB 50
230 X=A:Y=B+J:GOSUB 50
240 Y=B-J:GOSUB 50
250 X=A+J:Y=B+J:GOSUB 50
260 X=A-J:Y=B-J:GOSUB 50
270 Y=B+J:GOSUB 50
280 X=A+J:Y=B-J:GOSUB 50
700 NEXT J
750 FOR J=1 TO 500:NEXT J
760 FOR J=1 TO 8:R(J)=0:NEXT J
800 GOTO 150
900 W=INT(J/2):T=J-W-1
930 FOR N=1 TO W
940 IF X<>A THEN 950
945 Y2=Y:X2=X+N:GOSUB 60:X2=X-N:GOSUB 60:NEXT N:
RETURN
950 IF Y<>B THEN 970
960 X2=X:Y2=Y+N:GOSUB 60:Y2=Y-N:GOSUB 60:NEXT N:
RETURN
970 Y2=Y:IF X>=A THEN 980
975 X2=X+N:GOSUB 60:GOTO 990
980 X2=X-N:GOSUB 60
990 X2=X:IF Y>=B THEN 1000
995 Y2=Y+N:GOSUB 60:GOTO 1010
1000 Y2=Y-N:GOSUB 60
1010 NEXT N:RETURN
2000 COLOR R(N):PLOT X2,Y2
2010 RETURN
3000 ZX=0:FOR Q=705 TO 712
3001 ZX=ZX+1
3005 Z=INT(RND(1)*15)+1
3009 ZA=0
3010 FOR J=1 TO 8:IF Z=ZY(J) THEN ZA=1:J=8
3011 NEXT J
3012 IF ZA=1 THEN 3005
3015 POKE Q,Z*16+ZX+6:ZY(ZX)=Z
3020 NEXT Q:RETURN
```

EASY CHANGES

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

```
175 IF RND(1) <=.2 THEN GRAPHICS 10:POKE
    704,0
```

   For 50%, use .5 instead of .2, etc.

2. To cause only the outward patterns to be displayed, insert line 178 to say

```
178 D=-1
```

   To cause only inward patterns, change it to say

```
178 D=1
```

3. To alter the number of graphics colors used in the patterns, alter the value of M in line 130. Be sure it is an integer from 2 to 15.
4. To lengthen the delay after each pattern is drawn, change this line:

```
750 FOR J=1 TO 1500:NEXT J
```

   Or, eliminate line 750 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

50– 65	Subroutines to do color plotting.
100– 130	Housekeeping. Initializes variables.
150– 170	Picks 8 random graphics colors.
180– 190	Reverses direction of display (inward-outward).
200– 700	Displays a full screen of the pattern.
750	Delay loop.
760	Zeroes out R array.
800	Goes back to create next pattern.
900–1010	Plots points on axes of design.
2000–2010	Subroutine to plot points between axes.
3000–3020	Initialization subroutine.

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 random graphics colors.
T, 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, Q, Z, ZA, ZX	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.

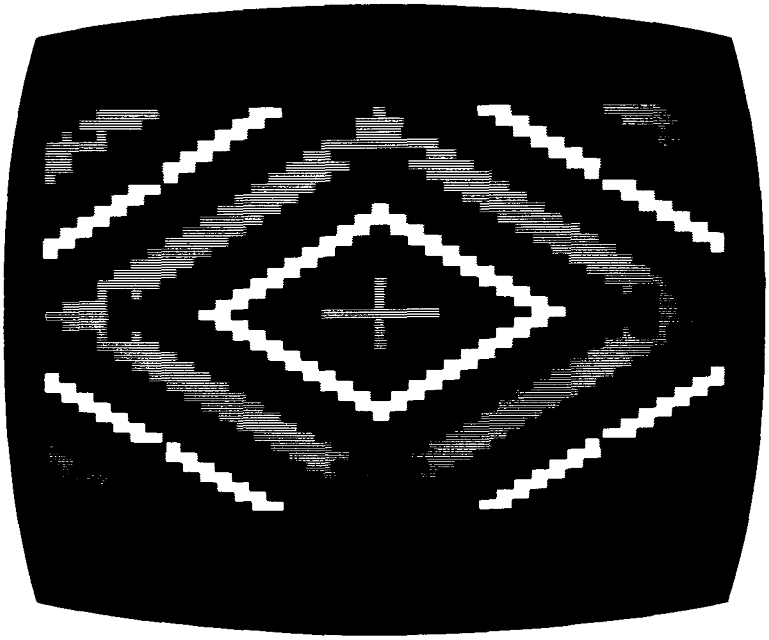
The second type of pattern starts about two seconds after the first has finished. It is a series of "sweeps" across the screen—left to right and top to bottom. Each sweep uses a random graphics color that is spaced equally across the screen. The spacing distance is chosen at random for each sweep. Also, 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 ATARI then sit back and watch the results of your labor.

SAMPLE RUN



One of the patterns generated by the SPARKLE program.

PROGRAM LISTING

```
10 REM SPARKLE
15 REM COPYRIGHT 1984 DILITHIUM PRESS
20 GOTO 100
50 H=XX*2:Z=YY*4
55 PLOT H,Z:DRAWTO H,Z+3:PLOT H+1,Z+3:DRAWTO H+1
,Z
60 RETURN
100 GRAPHICS 10
110 S=19
120 DIM A(S),B(S):X=S:Y=S
130 GOSUB 1000
140 T=INT(8*RND(1)+1)
150 FOR J=0 TO S:A(J)=J:B(J)=J:NEXT J
160 FOR J=0 TO S:R=INT((S+1)*RND(1))
170 W=A(J):A(J)=A(R):A(R)=W:NEXT J
180 FOR J=0 TO S:R=INT((S+1)*RND(1))
190 W=B(J):B(J)=B(R):B(R)=W:NEXT J
200 FOR J=0 TO S:FOR K=0 TO S
210 R=A(J):W=B(K):C=R+W+T
220 COLOR C
240 XX=X+R:YY=Y+W:GOSUB 50
```

```
250 XX=X+R:YY=Y-W:GOSUB 50
260 XX=X-R:YY=Y-W:GOSUB 50
270 XX=X-R:YY=Y+W:GOSUB 50
280 XX=X+W:YY=Y+R:GOSUB 50
290 XX=X+W:YY=Y-R:GOSUB 50
300 XX=X-W:YY=Y-R:GOSUB 50
310 XX=X-W:YY=Y+R:GOSUB 50
320 NEXT K:NEXT J
350 FOR J=1 TO 20:NEXT J
400 M=15
405 N=INT(21*RND(1))+10
410 FOR J=1 TO N
420 R=INT(22*RND(1))+1:W=INT(M*RND(1))
430 COLOR W
450 FOR XX=Y-S TO Y+S STEP INT(R/4)+1
460 FOR YY=X-S TO X+S STEP R
470 GOSUB 50
480 NEXT YY:NEXT XX:NEXT J
490 GOTO 140
1000 POKE 704,0
1010 FOR J=705 TO 712
1020 H=INT(RND(1)*16)
1030 L=INT(RND(1)*10)+4
1040 Z=(H*16)+L
1050 POKE J,Z
1060 NEXT J
1070 RETURN
```

EASY CHANGES

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

   135 GOTO 400

   Or, eliminate the first type of pattern by inserting:

   145 GOTO 400

   Or, eliminate the second type of pattern by inserting:

   360 GOTO 140

2. Increase the delay after the first type of pattern by increasing the 20 in line 350 to, say, 500. Remove line 350 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 405 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 405 to 5 or 10.
4. Watch the effect on the second type of pattern if you change the 22 in line 420 into various integer values between 2 and 39.

5. Change the value of M in line 400 to alter the graphics color used in the second type of pattern. For example, try

```
400 M=4
```

Be sure that M is an integer from 2 to 15.

MAIN ROUTINES

50- 60	Color plotting subroutine.
120- 130	Initializes variables. Clears screen.
140- 320	Displays diamond pattern in center of screen.
150- 190	Shuffles the numbers 0 through 19 in the A and B arrays.
200- 320	Displays graphics colors on the screen.
350	Delay routine.
400- 480	Overlays the entire screen with a random graphics color spaced at a fixed interval chosen at random.
1000-1070	Initialize screen and colors.

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 (19 across, 19 down).
XX, YY	Graphic coordinate work variables.
T	Integer from 0 to 8, used in creating random graphics colors.
J, K, L	Work and loop variables.
H, W, Z	Work variables.
C	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

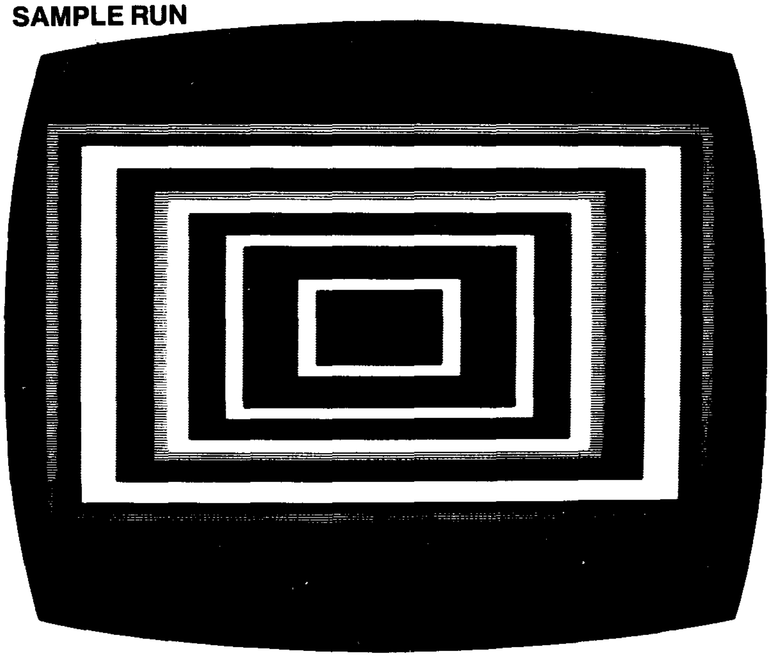
PURPOSE

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. They are actually rectangles, not squares, but let's not be nit-pickers.

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

```
10 REM SQUARES
15 REM COPYRIGHT 1984 DILITHIUM PRESS
20 GOTO 100
50 K=XX*2:L=YY*4
60 PLOT K,L:DRAWTO K,L+3:DRAWTO K+1,L+3:DRAWTO K
+1,L
70 RETURN
100 C1=1:C2=15
110 DIM ZY(8)
115 FOR J=1 TO 8:ZY(J)=0:NEXT J
120 GRAPHICS 10
125 GOSUB 500
130 X=20:Y=20:N=1
140 C=INT(RND(1)*8):IF C=C1 OR C=C2 OR C=C3 THEN
 GOTO 140
150 C3=C2:C2=C1:C1=C
170 COLOR C:FOR J=0 TO N:XX=X+J:YY=Y:GOSUB 50:NE
XT J
```

```
180 X=X+N:N=N+1
190 FOR J=0 TO N:XX=X:YY=Y-J:GOSUB 50:NEXT J
200 Y=Y-N:FOR J=0 TO N:XX=X-J:YY=Y:GOSUB 50:NEXT
 J
210 X=X-N
220 FOR J=0 TO N:XX=X:YY=Y+J:GOSUB 50:NEXT J
230 N=N+1:Y=Y+N
240 IF N<38 THEN 140
250 FOR J=1 TO 10:NEXT J
260 GOTO 130
500 POKE 704,0
510 FOR K=705 TO 712
515 ZX=ZX+1
520 Q=INT(RND(1)*16):ZA=0
525 FOR J=1 TO 8:IF Q=ZY(J) THEN ZA=1:J=8
530 NEXT J
535 IF ZA=1 THEN 520
540 L=(Q*16)+ZX
550 POKE K,L
560 NEXT K:RETURN
```

EASY CHANGES

1. Change the delay after each set of patterns by changing the 10 in line 250. A bigger number causes a longer delay.
2. To occasionally blank out the screen (about 20% of the time), insert this:

   255 IF RND(1)<.2 THEN GRAPHICS 10:POKE 704,0

MAIN ROUTINES

50– 70	Color plotting subroutine.
100–125	Housekeeping. Clears screen.
130	Initializes counters for each pattern. Points to the center of the screen.
140–150	Picks a graphics color.
170–180	Draws the bottom side of the square.
190	Draws the right side.
200–210	Draws the top side.
220–230	Draws the left side.
240	Tests if the outermost square has been drawn.
250	Delay loop.
260	Goes back to start at the center again.
500–560	Initialize screen and colors.

MAIN VARIABLES

X, Y	Coordinates of points being plotted.
XX, YY	Plotting work variables.
N	Length of the side currently being drawn.
C	Numeric equivalent of the random graphics color chosen.
J	Loop variable.
K, L, Q, ZA, C1, C2, C3	Work variables.

WALLOONS

PURPOSE

The ATARI 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 ATARI Arena.

The ATARI Arena 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.

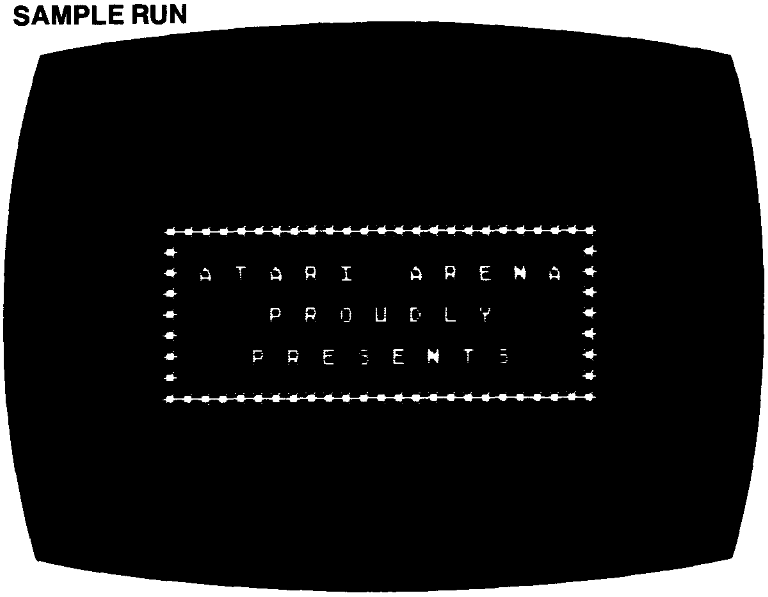
HOW TO USE IT

Just sit back, relax, and get ready to enjoy the show. 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 finale for their act, but they haven't quite perfected it yet.

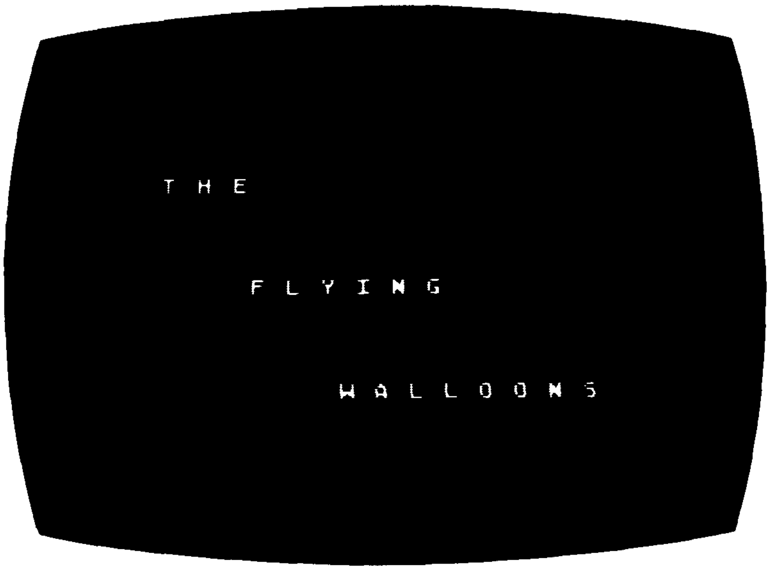
Note: This program POKES into machine memory and may clobber the program if not entered correctly. So, after typing it into your machine, be sure to save it on cassette or disk *before* running it. Then you can reload it in case of trouble.

SAMPLE RUN



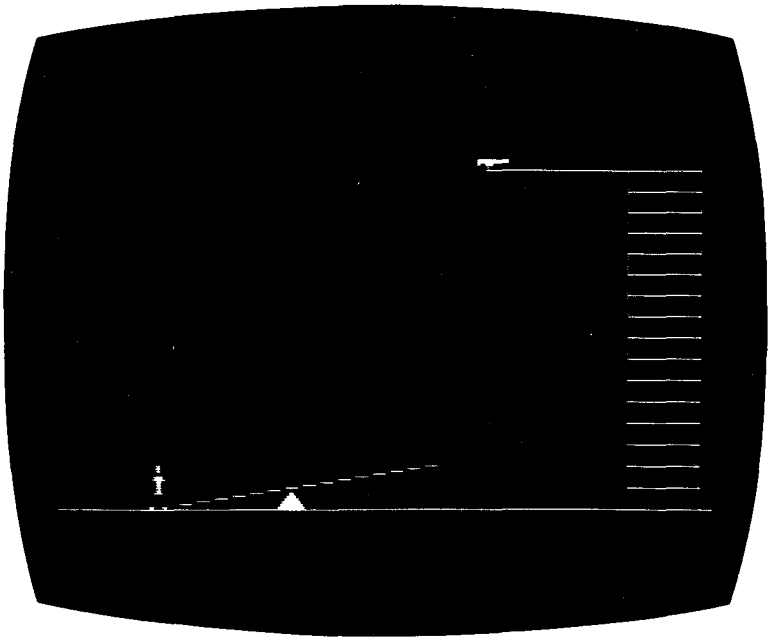
```
* * * * * * * * * * * * * * * * * * * * * * * * *
*                                               *
*  A  T  A  R  I     A  R  E  N  A              *
*                                               *
*        P  R  O  U  D  L  Y                    *
*                                               *
*      P  R  E  S  E  N  T  S                   *
*                                               *
* * * * * * * * * * * * * * * * * * * * * * * * *
```

The billboard announces a new presentation of the (in)famous ATARI Arena.



```
T  H  E


        F  L  Y  I  N  G


                W  A  L  L  O  O  N  S
```

"The Flying Walloons" are to perform!


The Walloons attempt a dangerous trick from their repertoire.

PROGRAM LISTING

```
10 REM WALLOONS
15 REM COPYRIGHT 1984 DILITHIUM PRESS
100 DIM B$(1):B$=CHR$(32)
120 GOTO 2000
150 POSITION AX,J-1:PRINT B$;B$;:POSITION AX,J:P
RINT B$;B$;:RETURN
160 POSITION AX,J+1:PRINT B$;B$;:POSITION AX,J+2
:PRINT B$;B$;:RETURN
200 POSITION AX,J:PRINT CHR$(100);CHR$(102);:POS
ITION AX,J+1:PRINT CHR$(101);CHR$(104);:RETURN
300 POSITION AX,J:PRINT CHR$(109);CHR$(111);:POS
ITION AX,J+1:PRINT CHR$(110);CHR$(112);:RETURN
400 POSITION AX,J:PRINT CHR$(107);CHR$(105);:POS
ITION AX,J+1:PRINT CHR$(108);CHR$(106);:RETURN
500 POSITION AX,J:PRINT CHR$(113);CHR$(115);:POS
ITION AX,J+1:PRINT CHR$(114);CHR$(116);:RETURN
800 IF AX=21 THEN 900
810 AX=21:POSITION 6,18:PRINT B$;B$;:POSITION 6,
19:PRINT CHR$(100);CHR$(102);
815 POSITION 6,20:PRINT CHR$(101);CHR$(104);
820 POSITION 6,19:PRINT B$;B$;:POSITION 6,20:PRI
```

```
NT CHR$(100);CHR$(102);
825 POSITION 21,20:PRINT CHR$(100);CHR$(102);:PO
SITION 6,21
830 PRINT CHR$(101);CHR$(104);B$;:FOR K=1 TO 12:
PRINT B$;:NEXT K:PRINT CHR$(101);CHR$(104);
835 POSITION 6,22
840 FOR K=1 TO 7:PRINT CHR$(0);:NEXT K:PRINT CHR
$(16);CHR$(11);CHR$(17);:FOR K=1 TO 7:PRINT CHR$
(0);:NEXT K
850 POSITION 21,19:PRINT CHR$(100);CHR$(102);:PO
SITION 6,20
855 PRINT B$;B$;:POSITION 21,20:PRINT CHR$(101);
CHR$(104);
860 POSITION 6,21:PRINT CHR$(100);CHR$(102);:FOR
 K=1 TO 7:PRINT B$;:NEXT K:FOR K=7 TO 0 STEP -1
862 PRINT CHR$(K);:NEXT K
865 POSITION 6,22:PRINT CHR$(101);CHR$(104);:FOR
 K=6 TO 2 STEP -1:PRINT CHR$(K);:NEXT K:PRINT CH
R$(14);CHR$(11);
870 PRINT CHR$(13);:FOR K=1 TO 7:PRINT B$;:NEXT
K
880 POSITION 21,19:PRINT B$;B$;:POSITION 21,20:P
RINT B$;B$;
890 RETURN
900 AX=6:POSITION 21,18:PRINT B$;B$;:POSITION 21
,19:PRINT CHR$(100);CHR$(102);:POSITION 21,20:PR
INT CHR$(101);
905 PRINT CHR$(104);
910 POSITION 21,19:PRINT B$;B$;:POSITION 6,20:PR
INT CHR$(100);CHR$(102);
915 POSITION 21,20:PRINT CHR$(100);CHR$(102);:PO
SITION 6,21
920 PRINT CHR$(101);CHR$(104);B$;:FOR K=1 TO 12:
PRINT B$;:NEXT K:PRINT CHR$(101);CHR$(104);
925 POSITION 6,22
930 FOR K=1 TO 7:PRINT CHR$(0);:NEXT K:PRINT CHR
$(16);CHR$(11);CHR$(17);:FOR K=1 TO 7:PRINT CHR$
(0);:NEXT K
940 POSITION 6,19:PRINT CHR$(100);CHR$(102);:POS
ITION 21,20:PRINT B$;B$;:POSITION 6,20:PRINT CHR
$(101);CHR$(104);
950 POSITION 6,21:FOR K=0 TO 7:PRINT CHR$(K);:NE
XT K
955 FOR K=1 TO 8:PRINT B$;:NEXT K:POSITION 21,22
:PRINT CHR$(101);CHR$(104);
960 POSITION 21,21:PRINT CHR$(100);CHR$(102);:PO
SITION 6,22:FOR K=1 TO 7:PRINT B$;:NEXT K
965 PRINT CHR$(12);CHR$(11);CHR$(15);:FOR K=2 TO
 6:PRINT CHR$(K);:NEXT K
980 POSITION 6,19:PRINT B$;B$;:POSITION 6,20:PRI
NT B$;B$;
990 RETURN
```

```
1000 POKE 752,1:PRINT CHR$(125):FOR I=1 TO 38:PO
SITION I,23:PRINT CHR$(0);:NEXT I
1010 FOR I=3 TO 22:POSITION 38,I:PRINT CHR$(9);:
NEXT I
1020 FOR I=7 TO 22:POSITION 33,I:PRINT CHR$(8);:
NEXT I
1030 FOR I=7 TO 22:POSITION 34,I:PRINT CHR$(0);C
HR$(0);CHR$(0);CHR$(0);:NEXT I
1040 POSITION 33,7:PRINT CHR$(10);
1050 FOR I=32 TO 26 STEP -1:POSITION I,7:PRINT C
HR$(0);:NEXT I:POSITION 25,7:PRINT CHR$(103);
1060 FOR I=0 TO 7:POSITION 22-I,21:PRINT CHR$(I)
;:NEXT I
1070 FOR I=1 TO 7:POSITION 14-I,22:PRINT CHR$(I)
;:NEXT I
1080 POSITION 13,22:PRINT CHR$(14);CHR$(11);CHR$
(13);
1090 GOSUB 2100
1100 POSITION 24,4:PRINT CHR$(100);CHR$(102);:PO
SITION 24,5:PRINT CHR$(101);CHR$(104);
1105 FOR I=1 TO 100:NEXT I
1110 POSITION 23,3:PRINT CHR$(107);CHR$(105);:PO
SITION 23,4:PRINT CHR$(108);CHR$(106);B$
1115 POSITION 24,5:PRINT B$;B$:GOSUB 2900
1120 POSITION 22,3:PRINT CHR$(107);CHR$(105);B$;
:POSITION 22,4:PRINT CHR$(108);CHR$(106);B$
1125 GOSUB 2900
1130 POSITION 21,4:PRINT CHR$(113);CHR$(115);B$;
:POSITION 21,5:PRINT CHR$(114);CHR$(116);B$
1135 POSITION 22,3:PRINT B$;B$;:GOSUB 2900
1140 AX=21:Z=4:FOR I=4 TO 15
1150 FOR J=I TO 18:GOSUB 150:Z=Z+1:IF Z=5 THEN Z
=1
1160 IF Z=2 AND 18-J<4 THEN Z=1
1170 ON Z GOSUB 200,300,400,500
1180 GOSUB 2900
1200 NEXT J
1210 GOSUB 800
1220 FOR J=18 TO I STEP -1:GOSUB 160:Z=Z+1:IF Z=
5 THEN Z=1
1230 ON Z GOSUB 200,300,400,500
1240 GOSUB 2900
1250 NEXT J
1255 J=J+1:GOSUB 150
1260 NEXT I
1270 FOR J=16 TO 21:GOSUB 150:Z=Z+1:IF Z=5 THEN
Z=1
1280 ON Z GOSUB 200,300,400,500
1290 NEXT J
1300 FOR I=1 TO 500:NEXT I:GRAPHICS 0:END
2000 RAM=PEEK(106):POKE 106,RAM-4:GRAPHICS 0:POK
E 752,1:GOSUB 2500:CH=RAM-4
```

```
2010 AD=CH*256:FOR X=1 TO 500:POKE AD+X,PEEK(573
44+X):NEXT X:GOSUB 2600
2020 FOR X=501 TO 1023:POKE AD+X,PEEK(57344+X):N
EXT X
2030 RESTORE 3000:FOR I=1 TO 46:READ M:XX=AD+M
2040 FOR J=0 TO 7:READ N:POKE XX+J,N:NEXT J:NEXT
 I
2050 GRAPHICS 0:POKE 756,CH+2:POKE 752,1:GOTO 10
00
2100 XX=20:D=1:FOR I=1 TO 100:NEXT I
2110 FOR I=1 TO 6:POSITION I-1,22:PRINT B$;:POSI
TION I-1,21:PRINT B$;:POSITION I,21:PRINT CHR$(1
8);
2120 D=-D:XX=XX-D:POSITION I,22:PRINT CHR$(XX);:
FOR J=1 TO 50:NEXT J:NEXT I
2130 GOSUB 2900:POSITION 6,21:PRINT CHR$(100);CH
R$(102);:POSITION 6,22:PRINT CHR$(101);CHR$(104)
;
2140 GOSUB 2700
2150 FOR I=21 TO 5 STEP -1:GOSUB 2950:POSITION 3
5,I+2:PRINT CHR$(0);CHR$(0);:GOSUB 2900:NEXT I
2160 XX=22:D=1:A=19:FOR I=35 TO 25 STEP -1:GOSUB
 2450:GOSUB 2400:GOSUB 2900:NEXT I
2170 POSITION 25,6:PRINT CHR$(22);:FOR I=1 TO 17
5:NEXT I
2180 XX=20:D=1:A=18:FOR I=25 TO 35:GOSUB 2450:GO
SUB 2350:GOSUB 2900:NEXT I:FOR I=1 TO 100:NEXT I
2190 POSITION 35,5:PRINT CHR$(19);:POSITION 35,6
:PRINT CHR$(22);:FOR I=1 TO 200:NEXT I
2200 XX=22:D=1:A=19:FOR I=35 TO 25 STEP -1:GOSUB
 2450:GOSUB 2400:GOSUB 2900:NEXT I
2210 FOR I=1 TO 100:NEXT I:POSITION 25,5:PRINT B
$;:POSITION 25,6:PRINT CHR$(24);CHR$(25);
2220 FOR I=1 TO 200:NEXT I:POSITION 25,5:PRINT C
HR$(19);:POSITION 25,6:PRINT CHR$(22);B$;
2230 FOR I=1 TO 100:NEXT I:POSITION 25,5:PRINT C
HR$(18);:POSITION 25,6:PRINT CHR$(20);
2240 XX=20:D=1:A=18:FOR I=25 TO 35:GOSUB 2450:GO
SUB 2350:GOSUB 2900:NEXT I:FOR I=1 TO 100:NEXT I
2250 POSITION 35,5:PRINT CHR$(19);:POSITION 35,6
:PRINT CHR$(22);:FOR I=1 TO 200:NEXT I
2260 XX=22:D=1:A=19:FOR I=35 TO 25 STEP -1:GOSUB
 2450:GOSUB 2400:GOSUB 2900:NEXT I
2270 FOR I=1 TO 3:POSITION 25,4:PRINT CHR$(19);:
POSITION 25,5:PRINT CHR$(22);:POSITION 25,6:PRIN
T B$;
2280 FOR K=1 TO 100:NEXT K:POSITION 25,4:PRINT B
$;:POSITION 25,5:PRINT CHR$(19);:POSITION 25,6
2290 PRINT CHR$(22);:FOR K=1 TO 100:NEXT K:NEXT
I:POSITION 25,4:PRINT CHR$(19);:POSITION 25,5:PR
INT CHR$(22);
```

```
2300 POSITION 25,6:PRINT B$;:RETURN
2350 POSITION I-1,5:PRINT B$;:POSITION I-1,6:PRI
NT B$;:RETURN
2400 POSITION I+1,5:PRINT B$;:POSITION I+1,6:PRI
NT B$;:RETURN
2450 D=-D:XX=XX-D:POSITION I,5:PRINT CHR$(A);:PO
SITION I,6:PRINT CHR$(XX);:RETURN
2500 PRINT CHR$(125):POSITION 11,9:PRINT "A T A
R I   A R E N A";
2510 POSITION 15,11:PRINT "P R O U D L Y";
2520 POSITION 14,13:PRINT "P R E S E N T S";
2530 POSITION 9,7:FOR J=1 TO 25:PRINT "*";:NEXT
J
2540 FOR J=1 TO 7:POSITION 9,7+J:PRINT "*";:POSI
TION 33,7+J:PRINT "*";:NEXT J
2550 POSITION 9,15:FOR J=1 TO 25:PRINT "*";:NEXT
2560 RETURN
2600 PRINT CHR$(125):POSITION 5,5:PRINT "T H E";
2610 POSITION 10,10:PRINT "F L Y I N G";
2620 POSITION 15,15:PRINT "W A L L O O N S";
2630 RETURN
2700 XX=22:X=39:GOSUB 2850:GOSUB 2900
2710 GOSUB 2850:POSITION 38,21:PRINT CHR$(9);:PO
SITION 38,22:PRINT CHR$(9);:GOSUB 2900
2720 FOR I=1 TO 4:GOSUB 2850:POSITION X+1,21:PRI
NT CHR$(0);:POSITION X+1,22:PRINT CHR$(0);:GOSUB
 2900:NEXT I
2730 GOSUB 2850:POSITION 33,21:PRINT CHR$(8);:PO
SITION 33,22:PRINT CHR$(8);:GOSUB 2900
2740 FOR I=1 TO 5:GOSUB 2850:POSITION X+1,21:PRI
NT B$;:POSITION X+1,22:PRINT B$;:GOSUB 2900:NEXT
 I
2750 FOR I=1 TO 75:NEXT I:POSITION X,21:PRINT CH
R$(100);CHR$(102);:POSITION X,22:PRINT CHR$(101)
;CHR$(104);
2755 FOR I=1 TO 300:NEXT I
2760 XX=20:FOR I=6 TO 1 STEP -1:GOSUB 2800:POSIT
ION X-1,21:PRINT B$;:POSITION X-1,22
2770 PRINT B$;:GOSUB 2900:NEXT I:GOSUB 2800:POSI
TION X-1,21:PRINT CHR$(8);:POSITION X-1,22:PRINT
 CHR$(8);
2780 GOSUB 2900:GOSUB 2800:POSITION X-1,21:PRINT
 CHR$(0);:POSITION X-1,22:PRINT CHR$(0);:GOSUB 2
900
2790 POSITION X,21:PRINT CHR$(100);CHR$(102);:PO
SITION X,22:PRINT CHR$(101);CHR$(104);:RETURN
2800 X=X+1:D=-D:XX=XX-D:POSITION X,21:PRINT CHR$
(18);:POSITION X,22:PRINT CHR$(XX);:RETURN
2850 X=X-1:D=-D:XX=XX-D:POSITION X,21:PRINT CHR$
(19);:POSITION X,22:PRINT CHR$(XX);:RETURN
2900 FOR K=1 TO 10:NEXT K:RETURN
```

```
2950 POSITION 35,I:PRINT CHR$(100);CHR$(102);:PO
SITION 35,I+1:PRINT CHR$(101);CHR$(104);:RETURN
3000 DATA 512,255,0,0,0,0,0,0,0,520,0,255,0,0,0,
0,0,0
3005 DATA 528,0,0,255,0,0,0,0,0,536,0,0,0,255,0,
0,0,0
3010 DATA 544,0,0,0,0,255,0,0,0,552,0,0,0,0,0,25
5,0,0
3015 DATA 560,0,0,0,0,0,0,255,0,568,0,0,0,0,0,0,
0,255
3020 DATA 576,3,3,3,3,3,3,3,3,584,192,192,192,19
2,192,192,192,192
3025 DATA 592,255,3,3,3,3,3,3,3,600,255,24,60,12
6,255,255,255,255
3030 DATA 608,0,0,0,0,0,1,3,7,616,0,0,0,0,0,128,
192,224
3035 DATA 624,0,255,0,0,0,1,3,7,632,0,255,0,0,0,
128,192,224
3040 DATA 640,255,0,0,0,0,1,3,7,648,255,0,0,0,0,
128,192,224
3045 DATA 656,224,224,224,192,224,240,248,236,66
4,7,7,7,3,7,15,31,55
3050 DATA 672,224,224,224,224,224,224,224,248,68
0,224,224,240,216,204,196,192,240
3055 DATA 688,7,7,7,7,7,7,7,31,696,7,7,15,27,51,
227,3,15
3060 DATA 704,0,0,0,0,255,255,239,6,712,0,0,0,0,
255,255,131,3
3090 DATA 800,3,3,3,1,7,15,27,51,808,3,3,7,6,6,6
,6,30
3100 DATA 816,192,192,192,128,224,240,216,204,82
4,15,0,0,0,0,0,0,0
3110 DATA 832,192,192,224,96,96,96,96,120,840,12
0,96,96,96,96,224,192,192
3120 DATA 848,204,216,240,224,128,192,192,192,85
6,30,6,6,6,6,7,3,3
3130 DATA 864,51,27,15,7,1,3,3,3,872,0,1,1,3,6,1
2,239,255
3140 DATA 880,255,239,12,6,3,1,1,0,888,0,0,0,3,3
,63,255,224
3150 DATA 896,224,255,63,3,3,0,0,0,904,0,0,0,192
,192,252,255,7
3160 DATA 912,7,255,252,192,192,0,0,0,920,0,0,0,
192,96,48,247,255
3170 DATA 928,255,247,48,96,192,0,0,0,936,7,7,7,
3,7,15,31,55
3180 DATA 944,103,7,15,27,51,227,3,15,952,103,7,
7,7,7,7,7,31
```

EASY CHANGES

1. If you wish to have the Walloons perform more (or less) jumps during their performance, change the loop bound value of 3 in line 2270 accordingly. To get more jumps, try

```
2270 FOR I=1 TO 8: <rest of line>
```

2. The title placard is currently bordered by asterisks. To get another character, change the asterisk character strings in lines 2530, 2540, and 2550 to another character.
3. You might want to personalize the title placard and make yourself the presenter of the Walloons. This can be done by altering the string literal, "ATARI ARENA", in line 2500 to something else. However, you cannot use a string with a length of more than 22 characters or it will be clipped by the end of the placard. To say, for example, that Mr. Simon Q. Fenster presents the Walloons, change line 2500 to read:

```
2500 PRINT CHR$(125):POSITION 11,9:
      PRINT"MR. SIMON Q. FENSTER";
```

MAIN ROUTINES

150-1290	Subroutines to draw (and erase) Walloons and their performing apparatus.
1300	Time-delay subroutine.
2000-2050	Initializes memory locations.
2100-2450	Drives text and graphics displays.
2500-2560	Subroutine to display placard.
2600-2630	The performers are announced.
2700-2950	The Walloons make their entrance and perform.
3000-3180	Graphics data.

MAIN VARIABLES

X, Y	Current X,Y location of Walloon; also "bit" table data values and locations.
XX, AX	Reference X,Y locations for Walloons.
I, J, K	Loop indices.

Z	Work variable.
RAM	Highest available memory page.
CH, AD	Reference memory pages and locations.
M, N, D	Work variables used in reading data.

SUGGESTED PROJECTS

1. Add some alternate tricks or endings to the act; 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.
2. Scour the world yourself for other acts to include in the ATARI Arena. 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 ATARI's graphic powers to draw the graph of a function $Y=f(X)$. The function is supplied by you. INTEGRAL 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 about a minute. Fifty data pairs and a fifth degree fit will take over three minutes.

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.

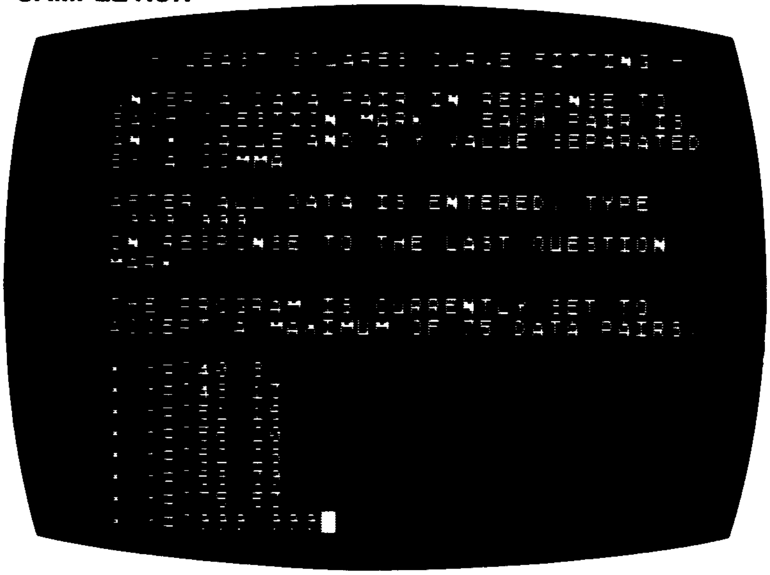
The painting is going to be sold at auction in 1984. What price should the investor expect to have to pay to purchase the painting? If he resold it in 1988, how much profit should he expect to make?

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.

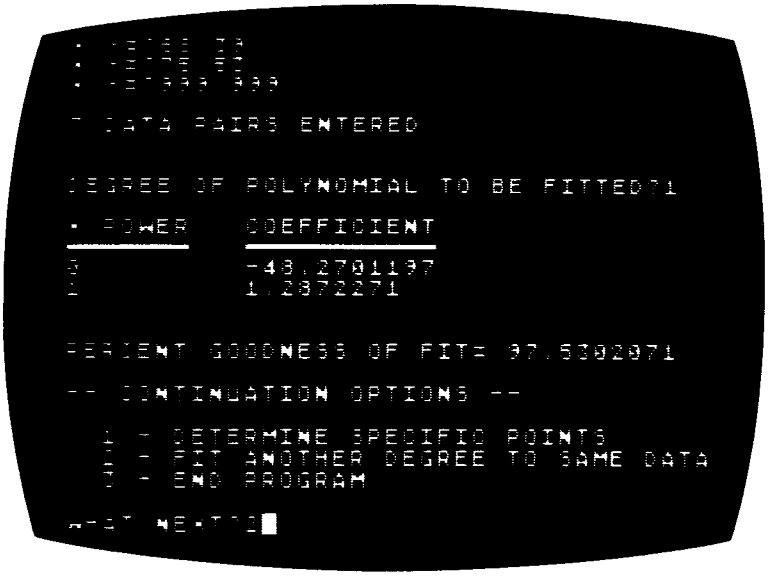
Of course, the investor did not make his decision solely on the basis of this program. He used it only as one guide to his decision. There is never any guarantee that financial data will perform in the future as it has done in the past. Though CURVE is probably as good a way as any, extrapolation of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

SAMPLE RUN



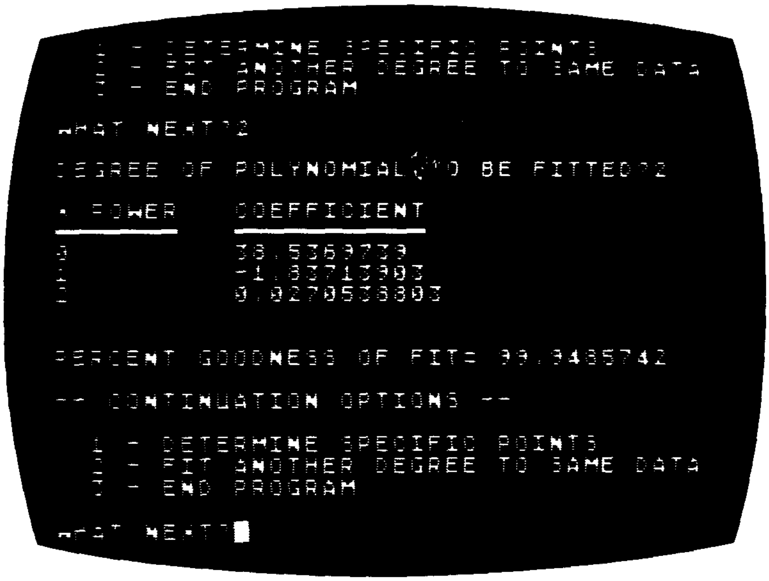
```
      - LEAST SQUARES CURVE FITTING -

   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=? [illegible]
   X,Y=? [illegible]
   X,Y=? [illegible]
   X,Y=? [illegible]
   X,Y=? [illegible]
   X,Y=? [illegible]
   X,Y=? [illegible]
   X,Y=? 999,999
```

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



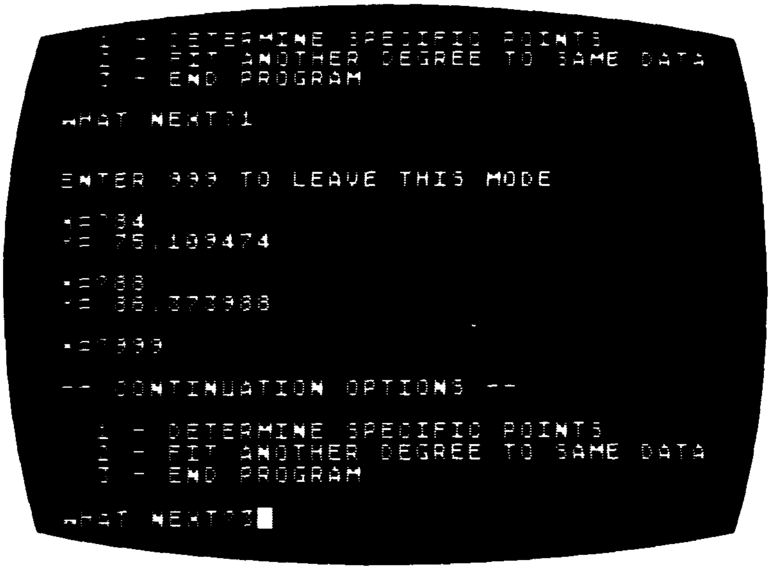
```
X,Y=? [illegible],73
X,Y=? [illegible],87
X,Y=? 999,999

7 DATA PAIRS ENTERED

DEGREE OF POLYNOMIAL TO BE FITTED?1

X POWER        COEFFICIENT
________       _____________
0              -48.2701197
1              1.2872271

PERCENT GOODNESS OF FIT= 97.5302071

-- CONTINUATION OPTIONS --

  1 - DETERMINE SPECIFIC POINTS
  2 - FIT ANOTHER DEGREE TO SAME DATA
  3 - END PROGRAM

WHAT NEXT?2
```

The operator selects a first degree fit to the data.



```
  1 - DETERMINE SPECIFIC POINTS
  2 - FIT ANOTHER DEGREE TO SAME DATA
  3 - END PROGRAM

WHAT NEXT?2

DEGREE OF POLYNOMIAL TO BE FITTED?2

X POWER     COEFFICIENT
-------     -----------
0           38.5369739
1           -1.83713903
2           0.0270533803


PERCENT GOODNESS OF FIT= 99.9485742

-- CONTINUATION OPTIONS --

  1 - DETERMINE SPECIFIC POINTS
  2 - FIT ANOTHER DEGREE TO SAME DATA
  3 - END PROGRAM

WHAT NEXT?█
```

A second degree fit is attempted for the same data.



```
  1 - DETERMINE SPECIFIC POINTS
  2 - FIT ANOTHER DEGREE TO SAME DATA
  3 - END PROGRAM

WHAT NEXT?1


ENTER 999 TO LEAVE THIS MODE

X=?84
Y= 75.109474

X=?88
Y= 88.373988

X=?999

-- CONTINUATION OPTIONS --

  1 - DETERMINE SPECIFIC POINTS
  2 - FIT ANOTHER DEGREE TO SAME DATA
  3 - END PROGRAM

WHAT NEXT?3█
```

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

PROGRAM LISTING

```
10 REM CURVE
15 REM COPYRIGHT 1984 DILITHIUM PRESS
130 GRAPHICS 0:PRINT CHR$(125)
150 MX=75
160 EF=999
170 MD=7
200 DIM X(MX),Y(MX)
210 Q=MD+1:DIM A(Q,Q),R(Q),V(Q)
220 Q=MD*2:DIM P(Q)
300 PRINT "  - LEAST SQUARES CURVE FITTING -":PR
INT
310 PRINT "ENTER A DATA PAIR IN RESPONSE TO"
320 PRINT "EACH QUESTION MARK.  EACH PAIR IS"
330 PRINT "AN X VALUE AND A Y VALUE SEPARATED":P
RINT "BY A COMMA."
340 PRINT :PRINT "AFTER ALL DATA IS ENTERED, TYP
E"
350 PRINT CHR$(32);EF;",";EF
360 PRINT "IN RESPONSE TO THE LAST QUESTION":PRI
NT "MARK."
370 PRINT :PRINT "THE PROGRAM IS CURRENTLY SET T
O"
380 PRINT "ACCEPT A MAXIMUM OF ";MX;" DATA PAIRS
."
400 PRINT :J=0
405 TRAP 3000
410 XZ=1:J=J+1:PRINT "X,Y=";:INPUT X,Y:X(J)=X:Y(
J)=Y
420 IF X(J)=EF AND Y(J)=EF THEN J=J-1:GOTO 450
430 IF J=MX THEN PRINT :PRINT "NO MORE DATA ALLO
WED":GOTO 450
440 GOTO 410
450 NP=J:PRINT
460 IF NP=0 THEN GOSUB 1600:PRINT "NO DATA ENTER
ED":END
470 PRINT NP;" DATA PAIRS ENTERED":PRINT
500 XZ=2:PRINT :PRINT "DEGREE OF POLYNOMIAL TO B
E FITTED";:INPUT D:PRINT
510 IF D<0 THEN GOSUB 1500:PRINT "DEGREE MUST BE
 >= 0":GOTO 500
520 D=INT(D):IF D<NP THEN 540
530 GOSUB 1500:PRINT "NOT ENOUGH DATA":GOTO 500
540 D2=2*D:IF D>MD THEN GOSUB 1500:PRINT "DEGREE
 TOO HIGH":GOTO 500
550 N=D+1
600 FOR J=1 TO D2:P(J)=0:FOR K=1 TO NP
610 P(J)=P(J)+X(K)^J:NEXT K:NEXT J:P(0)=NP
620 R(1)=0:FOR J=1 TOR)=R(1)+Y(J)
630 NEXT J:IF N=1 THEN 660
640 FOR J=2 TO N:R(J)=0:FOR K=1 TO NP
```

```
650 R(J)=R(J)+Y(K)*X(K)^(J-1):NEXT K:NEXT J
660 FOR J=1 TO N:FOR K=1 TO N:A(J,K)=P(J+K-2):NE
XT K:NEXT J
670 GOSUB 2000
700 PRINT "X POWER","COEFFICIENT"
710 FOR J=1 TO 7:PRINT CHR$(18);:NEXT J:PRINT CH
R$(32);CHR$(32);CHR$(32);
720 FOR J=1 TO 11:PRINT CHR$(18);:NEXT J:PRINT
730 FOR J=1 TO N:PRINT J-1,V(J):NEXT J:PRINT :PR
INT
740 Q=0:FOR J=1 TO NP:Q=Q+Y(J):NEXT J:M=Q/NP:T=0
:G=0:FOR J=1 TO NP
750 Q=0:FOR K=1 TO N:Q=Q+V(K)*X(J)^(K-1):NEXT K:
T=T+(Y(J)-Q)^2
760 G=G+(Y(J)-M)^2:NEXT J:IF G=0 THEN T=100:GOTO
 780
770 T=100*SQR(1-T/G)
780 PRINT "PERCENT GOODNESS OF FIT= ";T
800 PRINT :PRINT "-- CONTINUATION OPTIONS --":PR
INT
810 PRINT "  1 - DETERMINE SPECIFIC POINTS"
820 PRINT "  2 - FIT ANOTHER DEGREE TO SAME DATA
"
830 PRINT "  3 - END PROGRAM":PRINT
840 XZ=3:PRINT "WHAT NEXT";:INPUT Q:Q=INT(Q):IF
Q=3 THEN END
850 IF Q=2 THEN 500
860 IF Q<>1 THEN 800
900 PRINT :PRINT :PRINT "ENTER ";EF;" TO LEAVE T
HIS MODE"
910 XZ=4:PRINT :PRINT "X=";:INPUT XV:IF XV=EF TH
EN 800
920 YV=0:FOR K=1 TO N
930 YV=YV+V(K)*XV^(K-1):NEXT K:PRINT "Y= ";YV
940 GOTO 910
1500 PRINT "** ERROR! ** -- ";:RETURN
1600 PRINT "** FATAL ERROR! ** -- ";:RETURN
2000 IF N=1 THEN V(1)=R(1)/A(1,1):RETURN
2010 FOR K=1 TO N-1
2020 I=K+1
2030 L=K
2040 IF ABS(A(I,K))>ABS(A(L,K)) THEN L=I
2050 IF I<N THEN I=I+1:GOTO 2040
2060 IF L=K THEN 2100
2070 FOR J=K TO N:Q=A(K,J):A(K,J)=A(L,J)
2080 A(L,J)=Q:NEXT J
2090 Q=R(K):R(K)=R(L):R(L)=Q
2100 I=K+1
2110 Q=A(I,K)/A(K,K):A(I,K)=0
2120 FOR J=K+1 TO N:A(I,J)=A(I,J)-Q*A(K,J):NEXT
J
```

```
2130 R(I)=R(I)-Q*R(K):IF I<N THEN I=I+1:GOTO 211
0
2140 NEXT K
2150 V(N)=R(N)/A(N,N):FOR I=N-1 TO 1 STEP -1
2160 Q=0:FOR J=I+1 TO N:Q=Q+A(I,J)*V(J)
2170 V(I)=(R(I)-Q)/A(I,I):NEXT J:NEXT I
2180 RETURN
3000 IF PEEK(195)<>8 THEN 3050
3005 PRINT "ILLEGAL ENTRY"
3010 TRAP 3000:IF XZ=1 THEN J=J-1:GOTO 410
3020 IF XZ=2 THEN 500
3030 IF XZ=3 THEN 840
3040 IF XZ=4 THEN 910
3050 END
```

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 160 to any value not needed in your data. To use 10101, for example, make this change:

   160 EF=10101

2. Currently a maximum value of 75 data pairs is allowed. If you need more, change the value of MX in line 150 to the number required. For example, to allow up to 200 data pairs, use

   150 MX=200

3. To allow fits of higher degrees than seven, set MD in line 170 to the maximum degree desired. To achieve up to tenth degree fits, set the value of MD appropriately:

   170 MD=10

   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, an out-of-memory or illegal-quantity error will result after execution begins. If this occurs, decrease MX and/or MD to enable the program to run.

MAIN ROUTINES

130- 170	Initializes constants.
200- 220	Dimensions arrays.
300- 380	Displays introductory messages.
400- 470	Gets X-Y input data from the user.
500- 550	Gets degree of polynomial from the user, determines if it is acceptable.
600- 670	Sets up equations for the simultaneous equation solver and calls it.
700- 780	Calculates percent goodness of fit, displays all results.
800- 860	Gets user's continuation option and branches to it.
900- 940	Determines Y value corresponding to any X value.
1500-1600	Subroutines to print error messages.
2000-2180	Subroutine to solve simultaneous linear algebraic equations.
3000-3050	Trap to catch illegal entries.

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, XZ	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.

The program allows two forms of output. You can have the answers tabulated in columns or plotted graphically.

HOW TO USE IT

The first thing you must do is enter the differential equation into the program. This must be done at line 200. 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 200. The form of line 200 should be:

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

D represents dY/dX. GOSUBs are made to line 200 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 200 and 899 to complete the definition of D. Line 899 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 200. 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.

You now have a choice between two types of output. Enter a **T** for tabular output or a **G** for graphical output. The tabular form is simply a two-column display of the corresponding values of X and Y.

The graphical output plots the values of Y along a horizontal axis as each corresponding X value is displayed on successive lines of the screen. This graphical display requires you to input the minimum and maximum values of Y that will be used on the Y axis. You will be prompted for them if this output form is chosen. An asterisk (*) is used to plot the value of Y. If, however, the value of Y is "off-scale," an inverse asterisk is plotted at the appropriate edge of the graph.

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. It is applicable to both the tabular and graphical forms of output.

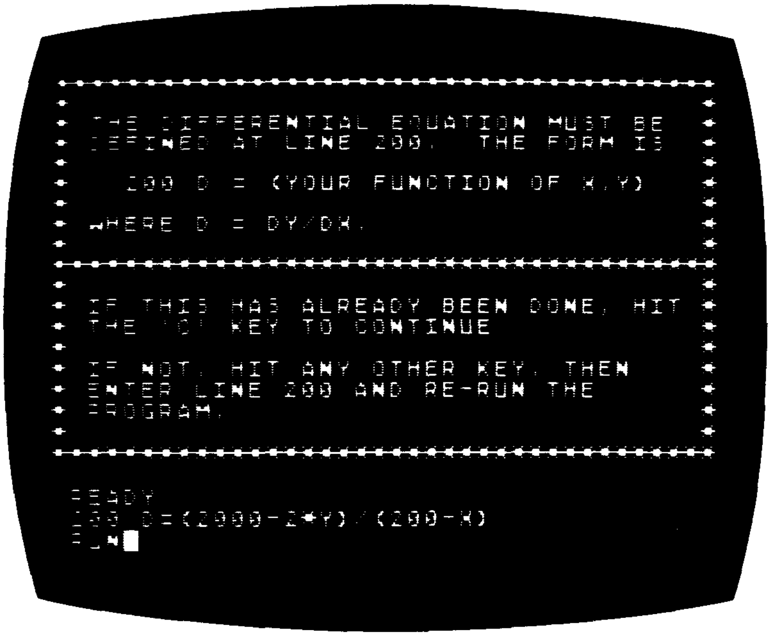
SAMPLE PROBLEM AND RUN

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:

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

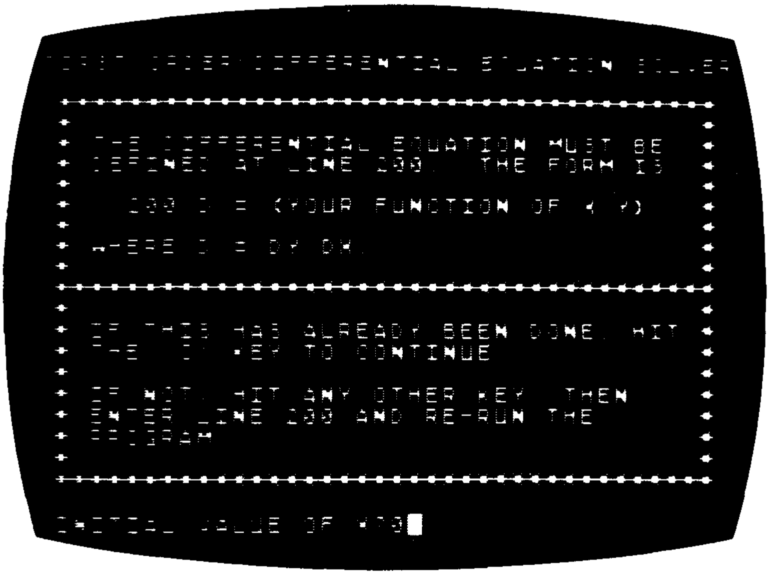
Find the velocity of the body every ten seconds up through two and one-half minutes. Also, plot this velocity as a function of time.

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



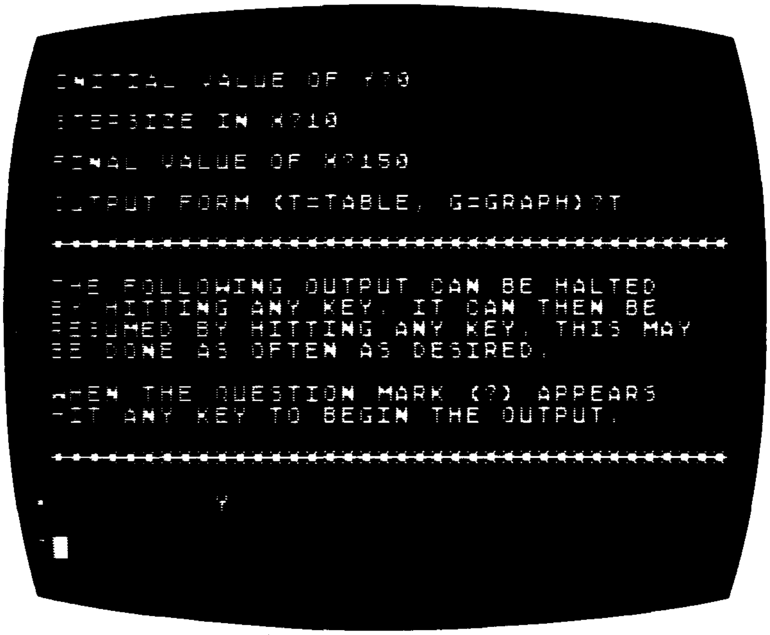
```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*                                         *
*  THE DIFFERENTIAL EQUATION MUST BE      *
*  DEFINED AT LINE 200.  THE FORM IS      *
*                                         *
*    200 D = (YOUR FUNCTION OF X,Y)       *
*                                         *
*  WHERE D = DY/DX.                       *
*                                         *
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*                                         *
*  IF THIS HAS ALREADY BEEN DONE, HIT     *
*  THE 'C' KEY TO CONTINUE                *
*                                         *
*  IF NOT, HIT ANY OTHER KEY, THEN        *
*  ENTER LINE 200 AND RE-RUN THE          *
*  PROGRAM.                               *
*                                         *
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

READY
200 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 200. He types RUN to restart the program.



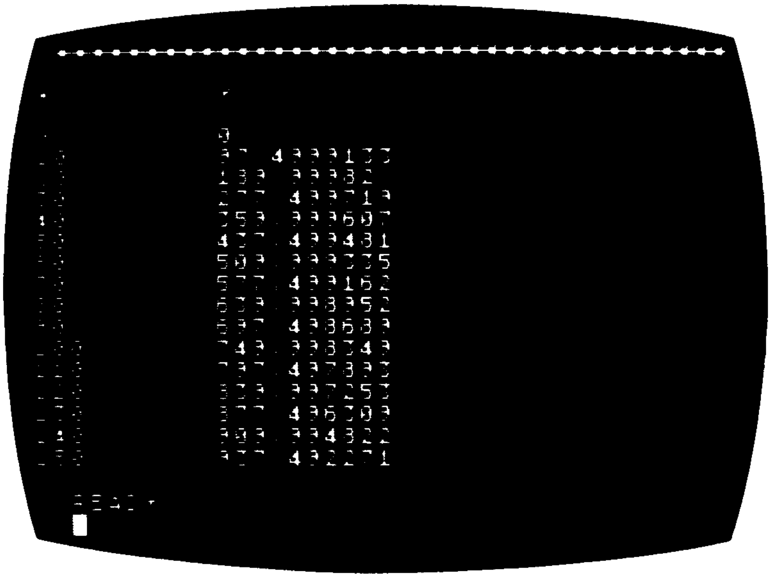
```
FIRST ORDER DIFFERENTIAL EQUATION SOLVER
+++++++++++++++++++++++++++++++++++++++
+                                     +
+ THE DIFFERENTIAL EQUATION MUST BE   +
+ DEFINED AT LINE 200.  THE FORM IS   +
+                                     +
+    200 D = (YOUR FUNCTION OF X,Y)   +
+                                     +
+ WHERE D = DY/DX.                    +
+                                     +
+++++++++++++++++++++++++++++++++++++++
+                                     +
+ IF THIS HAS ALREADY BEEN DONE, HIT  +
+ THE 'C' KEY TO CONTINUE             +
+                                     +
+ IF NOT, HIT ANY OTHER KEY.  THEN    +
+ ENTER LINE 200 AND RE-RUN THE       +
+ PROGRAM                             +
+                                     +
+++++++++++++++++++++++++++++++++++++++

INITIAL VALUE OF X?0
```

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



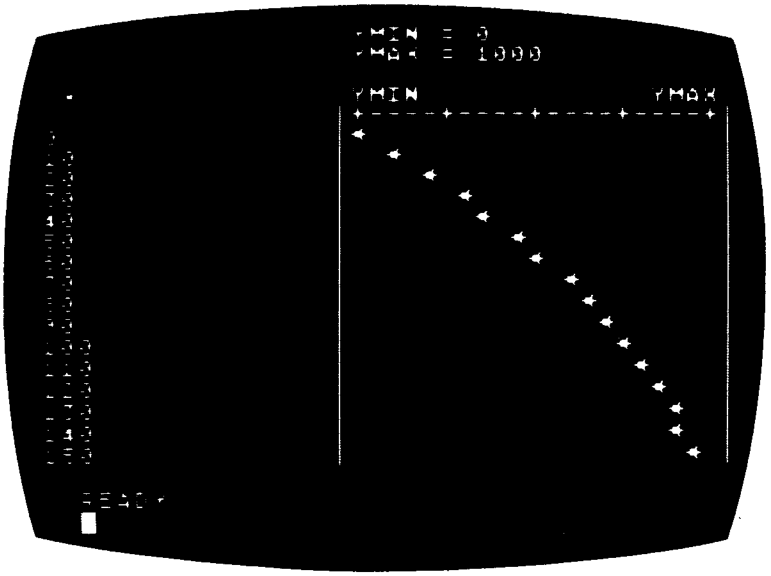
```
INITIAL VALUE OF Y?0

STEPSIZE IN X?10

FINAL VALUE OF X?150

OUTPUT FORM (T=TABLE, G=GRAPH)?T

+++++++++++++++++++++++++++++++++++++++

THE FOLLOWING OUTPUT CAN BE HALTED
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.

+++++++++++++++++++++++++++++++++++++++

            ?
```

The operator has completed the input and requested tabular output. 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.



```
[illegible]
READY
```

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



```
                    YMIN  = 0
                    YMAX  = 1000

                    YMIN                              YMAX
                    +-----+-----+-----+-----+
[illegible]
READY
```

The program is rerun requesting graphical output. Before this photo, the program requested a minimum and maximum value of Y to use on the Y axis. Values of 0 and 1000 respectively were entered. The program displays the desired graph.

PROGRAM LISTING

```
10 REM DIFFEQN
15 REM COPYRIGHT 1984 DILITHIUM PRESS
120 DIM F$(10)
130 GRAPHICS 0:PRINT CHR$(125)
150 GOTO 1200
200 GOTO 3000:REM REDEFINE THIS LINE TO BE D=(YO
UR FUNCTION OF X,Y)
899 RETURN
900 REM ****************************
910 REM DEFINE THE DIFFERENTIAL
920 REM EQUATION BETWEEN LINES
930 REM 200 AND 899
940 REM
950 REM LINE 200 MUST BE
960 REM OVERWRITTEN, MAKING IT
970 REM THE FIRST LINE OF THE
980 REM EQUATION
990 REM ****************************
1000 IF F$(1,1)="T" THEN PRINT XX,YY:GOSUB 1160:
RETURN
1010 F=(YY-YL)/(YH-YL):V=INT(17+20*F+0.5)
1020 C=42:IF YY<YL THEN V=17:C=170
1030 IF YY>YH THEN V=37:C=170
1040 PRINT XX;:POSITION 16,23:PRINT CHR$(124);:P
OSITION V,23
1050 PRINT CHR$(C);:POSITION 38,23:PRINT CHR$(12
4)
1060 GOSUB 1160:RETURN
1100 PRINT CHR$(32);:FOR Q=1 TO 38:PRINT "*";:NE
XT Q:PRINT :RETURN
1110 PRINT CHR$(32);"*";:POSITION 38,PEEK(84):PR
INT "*":RETURN
1120 POSITION 38,PEEK(84):PRINT "*":RETURN
1160 Q=PEEK(764):IF Q=255 THEN RETURN
1170 POKE 764,255
1180 Q=PEEK(764):IF Q=255 THEN 1180
1190 POKE 764,255:RETURN
1200 POKE 82,0:PRINT CHR$(125)
1210 PRINT "FIRST ORDER DIFFERENTIAL EQUATION SO
LVER"
1220 GOSUB 1100:GOSUB 1110
1230 PRINT " * THE DIFFERENTIAL EQUATION MUST BE
";:GOSUB 1120
1240 PRINT " * DEFINED AT LINE 200.  THE FORM IS
";:GOSUB 1120
1250 GOSUB 1110:PRINT " *    200 D = (YOUR FUNCTI
ON OF X,Y)";:GOSUB 1120
1260 GOSUB 1110:PRINT " * WHERE D = DY/DX.";:GOS
UB 1120
```

```
1270 GOSUB 1110:GOSUB 1100:GOSUB 1110
1280 PRINT " * IF THIS HAS ALREADY BEEN DONE, HI
T *"
1290 PRINT " * THE 'C' KEY TO CONTINUE";:GOSUB 1
120
1300 GOSUB 1110:PRINT " * IF NOT, HIT ANY OTHER
KEY. THEN";:GOSUB 1120
1310 PRINT " * ENTER LINE 200 AND RE-RUN THE";:G
OSUB 1120
1320 PRINT " * PROGRAM.";:GOSUB 1120
1330 GOSUB 1110:GOSUB 1100
1400 TRAP 4000:R=PEEK(764):IF R=255 THEN 1400
1405 POKE 764,255:IF R<>18 THEN POKE 82,2:END
1410 XZ=1:PRINT :PRINT " INITIAL VALUE OF X";:IN
PUT XX
1420 XZ=2:PRINT :PRINT " INITIAL VALUE OF Y";:IN
PUT YY:Y=YY:X=XX:GOSUB 200
1430 XZ=3:PRINT :PRINT " STEPSIZE IN X";:INPUT D
X
1440 XZ=4:PRINT :PRINT " FINAL VALUE OF X";:INPU
T XF
1450 PRINT :PRINT " OUTPUT FORM (T=TABLE, G=GRAP
H)";:INPUT F$
1455 IF F$="" THEN 1450
1460 IF F$(1,1)<>"T" AND F$(1,1)<>"G" THEN 1450
1470 IF F$(1,1)="T" THEN 1600
1480 XZ=5:PRINT :PRINT " MINIMUM Y FOR THE GRAPH
 AXIS";:INPUT YL
1490 XZ=6:PRINT :PRINT " MAXIMUM Y FOR THE GRAPH
 AXIS";:INPUT YH
1500 IF YH>YL THEN 1600
1510 PRINT
1520 PRINT " ** ERROR -- MAX Y MUST BE > MIN Y"
1530 PRINT CHR$(253):GOTO 1480
1600 PRINT :GOSUB 1100:PRINT
1610 PRINT " THE FOLLOWING OUTPUT CAN BE HALTED"
1620 PRINT " BY HITTING ANY KEY. IT CAN THEN BE"
1630 PRINT " RESUMED BY HITTING ANY KEY. THIS MA
Y"
1640 PRINT " BE DONE AS OFTEN AS DESIRED.":PRINT

1650 PRINT " WHEN THE QUESTION MARK (?) APPEARS"
1660 PRINT " HIT ANY KEY TO BEGIN THE OUTPUT."
1670 PRINT :GOSUB 1100:PRINT :POKE 764,255
1700 IF F$(1,1)="T" THEN PRINT "X","Y":GOTO 1800
1710 POSITION 17,23:PRINT "YMIN = ";YL
1720 POSITION 17,23:PRINT "YMAX = ";YH
1730 PRINT :PRINT " X";:POSITION 17,23:PRINT "YM
IN";:POSITION 34,23:PRINT "YMAX"
1740 POSITION 16,23
1750 PRINT CHR$(124);"+----+----+----+----+";CHR
$(124);
```

```
1800 PRINT :PRINT "?";
1810 IF PEEK(764)=255 THEN 1810
1815 POKE 764,255
1820 PRINT CHR$(30);CHR$(32);CHR$(30);
1830 GOSUB 1000
1900 Q=XX+DX:IF Q<XF+1.0E-05 THEN 1910
1905 POKE 82,2:END
1910 X=XX:Y=YY:GOSUB 200:K0=D:X=XX+DX/2:Y=YY+K0*
DX/2
1920 GOSUB 200:K1=D:Y=YY+K1*DX/2:GOSUB 200:K2=D
1930 X=XX+DX:Y=YY+K2*DX:GOSUB 200:K3=D
1940 DY=DX*(K0+2*K1+2*K2+K3)/6
1950 YY=YY+DY:XX=XX+DX:GOSUB 1000
1960 GOTO 1900
3000 PRINT :PRINT " ** ERROR! ** - YOU HAVE NOT
DEFINED"
3010 PRINT " THE DIFFERENTIAL EQUATION IN LINE 2
00"
3020 POKE 82,2:PRINT CHR$(253):END
4000 IF PEEK(195)<>8 THEN 4100
4005 TRAP 4000
4010 PRINT "  **ILLEGAL ENTRY**"
4020 ON XZ GOTO 1410,1420,1430,1440,1480,1490
4100 POKE 82,2:END
```

EASY CHANGES

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

   1215 PRINT:GOTO 1410

   This will immediately begin the input dialog.
2. If you wish to use negative stepsizes, line 1900 must be changed to:

   1900 Q=XX+DX:IF Q>XF−1.0E−05 THEN 1910

MAIN ROUTINES

120– 150	Initializes, begins execution.
200– 899	User-supplied subroutine to define D.
1000–1060	Displays output.
1100–1120	Subroutines to format messages.
1160–1190	Subroutine to stop and start output.
1200–1330	Displays initial messages.
1400–1530	Gets user's inputs.

1600–1670	Displays additional messages.
1700–1750	Initializes output display.
1800–1830	Waits for user to hit a key to start the output.
1900–1960	Computes each step.
3000–3020	Error message.
4000–4100	Error trap for illegal conditions.

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.
F$	Output flag string (T=table, G=graph).
YL, YH	Minimum, maximum values of Y plot axis.
F	Fractional distance of graphical point along Y axis.
V	Tab position for graphical output.
C	CHR$ argument for graphical output.
K0, K1, K2, K3	Runge-Kutta coefficients.
Q, XZ, R	Work variables.
J	Loop index.

SUGGESTED PROJECTS

1. Modify the program to display the tabular output followed by the graphical output. During the 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 200. 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 999 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 200. 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 ATARI uses to express each value must be no more than eight.

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.

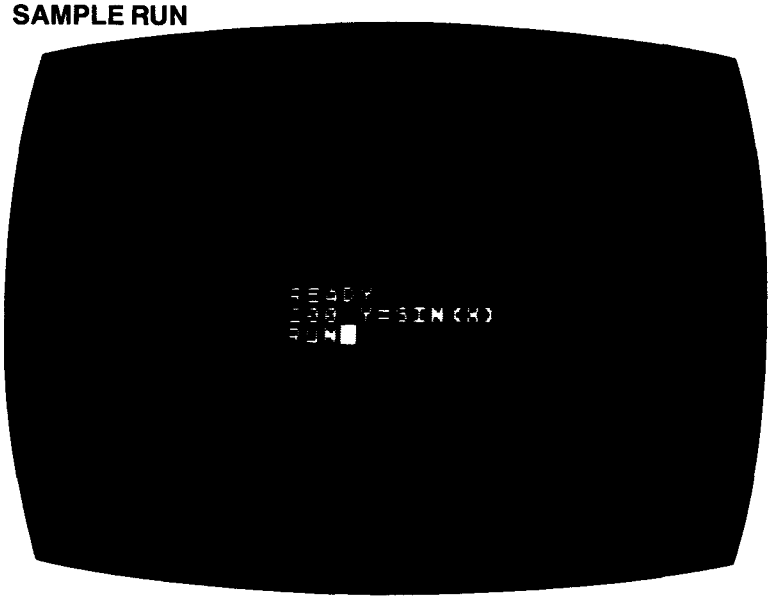
The program will now request that you hit any key to display the plot of your function. Each axis is 20 tick-marks long, with the origin defined as the minimum scale values of both X and Y. The minimum, middle, and maximum values on each scale are displayed appropriately. (Note: in certain cases, the program will not display the middle scaling value for one or both axes. However, the highest and lowest scaling values for each axis will always be present.)

The actual plot is drawn with eight times the resolution shown by the axes' tick-marks. That is, 160 values of X and Y are plotted. This is accomplished by taking advantage of the ATARI's high-resolution graphics capabilities.

If a value for Y should be off-scale, a special "enlarged line" 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 or if its value is exactly the minimum Y value.

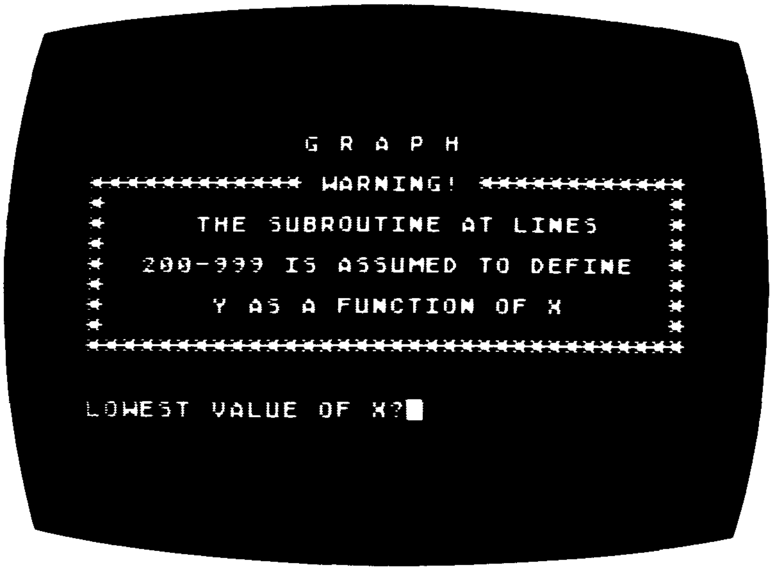
After the plot is drawn, you can exit the program and return to text mode by hitting any key.

SAMPLE RUN



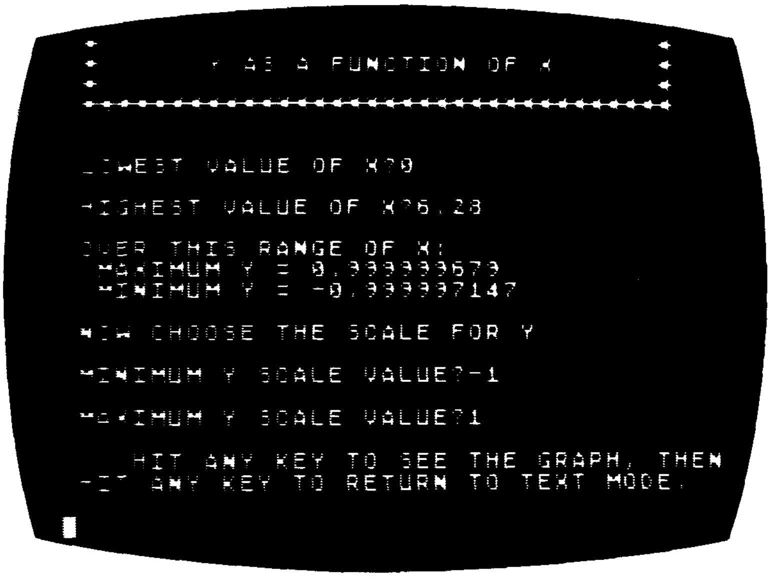
```
READY
200 Y=SIN(X)
RUN█
```

After loading the program, the operator enters line 200 to request the graph Y = SIN(X). RUN is typed to begin the program.



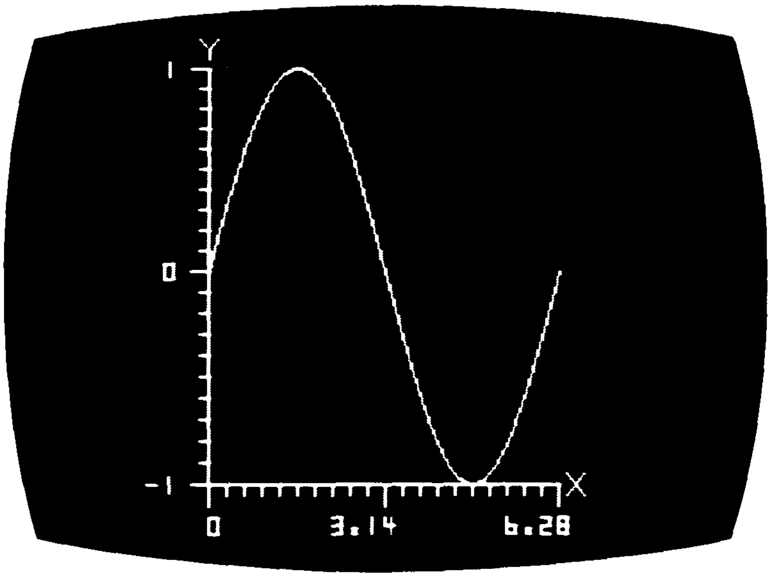
```
          G R A P H

*********** WARNING! ***********
*                              *
*   THE SUBROUTINE AT LINES    *
*                              *
* 200-999 IS ASSUMED TO DEFINE *
*                              *
*     Y AS A FUNCTION OF X     *
*                              *
********************************

LOWEST VALUE OF X?█
```

The program initiates the input dialog.



```
*                                          *
*         Y AS A FUNCTION OF X             *
*                                          *
********************************************

LOWEST VALUE OF X?0

HIGHEST VALUE OF X?6.28

OVER THIS RANGE OF X:
 MAXIMUM Y = 0.999999679
 MINIMUM Y = -0.999997147

NOW CHOOSE THE SCALE FOR Y

MINIMUM Y SCALE VALUE?-1

MAXIMUM Y SCALE VALUE?1

   HIT ANY KEY TO SEE THE GRAPH, THEN
HIT ANY KEY TO RETURN TO TEXT MODE.
```

The input dialog transpires. The operator asks that the domain of X be 0–6.28. The program responds by showing the maximum and minimum values of Y over this domain. The operator chooses an appropriate scale for the Y axis.



The graph is displayed as requested. The program waits for the operator to hit any key to return to text mode.

PROGRAM LISTING

```
10 REM GRAPH
15 REM COPYRIGHT 1984 DILITHIUM PRESS
110 POKE 82,0
120 DIM XL$(10),XU$(10),XM$(10),YL$(10),YU$(10),
YM$(10),D$(10),A$(10)
150 GOTO 1200
170 REM
180 REM ** ENTER SUBROUTINE AT 200 **
190 REM
200 REM *** Y=F(X) GOES HERE ***
999 RETURN
1200 GRAPHICS 0:PRINT CHR$(125):TRAP 6000
1210 POSITION 13,1:PRINT "G R A P H":PRINT
1220 PRINT " ************ WARNING! ************"
1230 GOSUB 1290
1240 PRINT " *      THE SUBROUTINE AT LINES";:GOS
UB 1295:GOSUB 1290
1250 PRINT " *  200-999 IS ASSUMED TO DEFINE";:G
OSUB 1295:GOSUB 1290
1260 PRINT " *       Y AS A FUNCTION OF X";:GOSUB
 1295:GOSUB 1290
1270 PRINT " ********************************"
1280 PRINT :GOTO 1300
1290 PRINT CHR$(32);"*";:POSITION 34,PEEK(84):PR
INT "*":RETURN
1295 POSITION 34,PEEK(84):PRINT "*":RETURN
1300 XZ=1:PRINT :PRINT " LOWEST VALUE OF X";:INP
UT XL
1310 XL$=STR$(XL):L=LEN(XL$)
1320 IF L>8 THEN PRINT :PRINT " *** TOO MANY DIG
ITS - PLEASE REDUCE":GOTO 1300
1330 XZ=2:PRINT :PRINT " HIGHEST VALUE OF X";:IN
PUT XU
1340 IF XU<=XL THEN PRINT :PRINT " *** BAD X RAN
GE ***":GOTO 1300
1350 XU$=STR$(XU):L=LEN(XU$)
1360 IF L>8 THEN PRINT :PRINT " *** TOO MANY DIG
ITS - PLEASE REDUCE:GOTO 1330"
1370 XM=(XL+XU)/2:XM$=STR$(XM)
1380 IF LEN(XM$)>9 THEN XM$=CHR$(32)
1400 DX=(XU-XL)/160:X=XL:GOSUB 200:MN=Y:MX=Y
1410 FOR J=1 TO 160:X=XL+J*DX:GOSUB 200
1420 IF Y>MX THEN MX=Y
1430 IF Y<MN THEN MN=Y
1440 NEXT J
1450 PRINT :PRINT " OVER THIS RANGE OF X:"
1460 PRINT "   MAXIMUM Y = ";MX
1470 PRINT "   MINIMUM Y = ";MN:PRINT
1480 PRINT " NOW CHOOSE THE SCALE FOR Y":PRINT
```

```
1490 XZ=3:PRINT " MINIMUM Y SCALE VALUE";:INPUT
YL:PRINT
1500 YL$=STR$(YL):L=LEN(YL$)
1510 IF L>8 THEN PRINT " *** TOO MANY DIGITS - P
LEASE REDUCE":PRINT :GOTO 1490
1520 XZ=4:PRINT " MAXIMUM Y SCALE VALUE";:INPUT
YU:PRINT
1530 IF YU<=YL THEN PRINT " *** BAD Y RANGE ***"
:GOTO 1450
1540 YU$=STR$(YU):L=LEN(YU$)
1550 IF L>8 THEN PRINT " *** TOO MANY DIGITS - P
LEASE REDUCE":PRINT :GOTO 1520
1560 YM=(YU+YL)/2:YM$=STR$(YM)
1570 IF LEN(YM$)>9 THEN YM$=CHR$(32)
1600 PRINT "    HIT ANY KEY TO SEE THE GRAPH, TH
EN"
1610 PRINT " HIT ANY KEY TO RETURN TO TEXT MODE.
"
1620 POKE 764,255:PRINT
1630 IF PEEK(764)=255 THEN 1630
1640 POKE 764,255
2000 GRAPHICS 8+16:SETCOLOR 2,0,0:SETCOLOR 1,15,
14:COLOR 1
2100 HP=107:VP=172
2110 PLOT HP,VP-160:DRAWTO HP,VP:DRAWTO HP+160,V
P
2120 FOR J=VP-160 TO VP STEP 8
2130 PLOT HP-4,J:DRAWTO HP,J:NEXT J
2140 FOR J=VP-160 TO VP STEP 80
2150 PLOT HP-8,J:DRAWTO HP,J:NEXT J
2160 FOR J=HP TO HP+160 STEP 8
2170 PLOT J,VP:DRAWTO J,VP+4:NEXT J
2180 FOR J=HP TO HP+160 STEP 80
2190 PLOT J,VP:DRAWTO J,VP+8:NEXT J
2200 PLOT HP+164,VP-4:DRAWTO HP+172,VP+4
2210 PLOT HP+164,VP+4:DRAWTO HP+172,VP-4
2220 PLOT HP-4,VP-172:DRAWTO HP,VP-168
2230 DRAWTO HP+4,VP-172
2240 PLOT HP,VP-168:DRAWTO HP,VP-164
2300 H=HP-20:V=VP-163:A$=YU$:GOSUB 5000
2310 H=HP-20:V=VP-83:A$=YM$:GOSUB 5000
2320 H=HP-20:V=VP-3:A$=YL$:GOSUB 5000
2340 H=HP:V=VP+12:A$=XL$:GOSUB 5000
2350 H=HP+80:V=VP+12:A$=XM$:GOSUB 5000
2360 H=HP+160:V=VP+12:A$=XU$:GOSUB 5000
2400 DX=(XU-XL)/160:DY=(YU-YL)/160
2410 AF=0:X=XL:GOSUB 200
2420 IF Y>YU THEN Y=YU:AF=1
2430 IF Y<YL THEN Y=YL:AF=1
2440 YY=(Y-YL)/DY:H=HP:V=VP-YY
2450 PLOT H,V
```

```
2460 IF AF=1 THEN GOSUB 3500
2500 FOR J=1 TO 160:X=XL+J*DX:GOSUB 200
2510 IF Y>YU THEN Y=YU:AF=1
2520 IF Y<=YL THEN Y=YL:AF=1
2530 YY=(Y-YL)/DY:H=HP+J:V=VP-YY
2540 DRAWTO H,V
2550 IF AF=1 THEN GOSUB 3500
2560 NEXT J
2565 IF PEEK(764)=255 THEN 2565
2570 GRAPHICS 0:POKE 764,255:POKE 82,2:END
3000 PLOT H,V:DRAWTO H4,V:DRAWTO H4,V6
3010 DRAWTO H,V6:DRAWTO H,V
3020 RETURN
3100 PLOT H4,V:DRAWTO H,V:DRAWTO H,V6
3110 DRAWTO H4,V6:PLOT H,V3:DRAWTO H4,V3
3120 RETURN
3200 PLOT H,V3:DRAWTO H4,V3:PLOT H+2,V+1:DRAWTO
H+2,V+5
3210 RETURN
3300 PLOT H,V3:DRAWTO H4,V3:RETURN
3400 PLOT H+1,V+4:DRAWTO H+3,V+4
3410 DRAWTO H+3,V6:DRAWTO H+1,V6
3420 DRAWTO H+1,V+4:RETURN
3500 DRAWTO H-1,V-1:DRAWTO H-1,V+1
3510 DRAWTO H+1,V+1:DRAWTO H+1,V-1
3520 DRAWTO H-1,V-1:DRAWTO H,V
3530 AF=0:RETURN
4100 PLOT H+2,V:DRAWTO H+2,V6
4110 RETURN
4200 PLOT H,V:DRAWTO H4,V
4210 DRAWTO H4,V3:DRAWTO H,V3
4220 DRAWTO H,V6:DRAWTO H4,V6:RETURN
4300 PLOT H,V:DRAWTO H4,V
4310 DRAWTO H4,V6:DRAWTO H,V6
4320 PLOT H,V3:DRAWTO H4,V3:RETURN
4400 PLOT H,V:DRAWTO H,V3:DRAWTO H4,V3
4410 PLOT H+3,V:DRAWTO H+3,V6:RETURN
4500 PLOT H4,V:DRAWTO H,V
4510 DRAWTO H,V3:DRAWTO H4,V3
4520 DRAWTO H4,V6:DRAWTO H,V6:RETURN
4600 PLOT H,V:DRAWTO H,V6:DRAWTO H4,V6
4610 DRAWTO H4,V3:DRAWTO H,V3:RETURN
4700 PLOT H,V:DRAWTO H4,V:DRAWTO H4,V+2
4710 DRAWTO H,V6:RETURN
4800 PLOT H,V:DRAWTO H4,V:DRAWTO H4,V6
4810 DRAWTO H,V6:DRAWTO H,V
4820 PLOT H,V3:DRAWTO H4,V3:RETURN
4900 PLOT H4,V6:DRAWTO H4,V:DRAWTO H,V
4910 DRAWTO H,V3:DRAWTO H4,V3:RETURN
5000 FOR J=LEN(A$) TO 1 STEP -1:H4=H+4:V3=V+3:V6
=V+6
```

```
5010 D$=A$(J,J):IF D$>"9" THEN 5040
5015 IF D$<"0" THEN 5040
5016 D=VAL(D$)
5020 IF D=0 THEN 5040
5030 ON D GOSUB 4100,4200,4300,4400,4500,4600,47
00,4800,4900:GOTO 5090
5040 IF D$="0" THEN GOSUB 3000:GOTO 5090
5050 IF D$="E" THEN GOSUB 3100:GOTO 5090
5060 IF D$="+" THEN GOSUB 3200:GOTO 5090
5070 IF D$="-" THEN GOSUB 3300:GOTO 5090
5080 IF D$="." THEN GOSUB 3400
5090 H=H-8:NEXT J:RETURN
6000 IF PEEK(195)<>8 THEN 6050
6010 TRAP 6000:PRINT :PRINT " ** ILLEGAL ENTRY *
*":PRINT
6020 ON XZ GOTO 1300,1330,1490,1520
6050 END
```

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:

```
1443 IF LEN(STR$(MX)) > 8 OR LEN(STR$(MN)) > 8
     THEN 1450
1444 IF MX < =MN THEN 1450
1445 YU=MX:YL=MN:PRINT
1447 YU$=STR$(YU):YL$=STR$(YL):GOTO 1560
```

(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. Do you sometimes forget to enter the subroutine at line 200 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 200 to read:

```
200 PRINT:PRINT “DEFINE SUBROUTINE AT LINE
    200”:END
```

Now, if you don't enter the actual subroutine desired, the program will stop after printing the warning message to define the subroutine.

MAIN ROUTINES

110– 150	Initializes variables, dimensions arrays, begins execution.
200– 999	User-supplied subroutine to evaluate Y as a function of X.
1200–1295	Displays introductory warning.
1300–1380	Gets X scaling from user.
1400–1570	Determines the minimum, maximum Y values; gets Y scale from user.
1600–1640	Waits for user to hit any key.
2000–2240	Draws plot axes.
2300–2360	Labels each axis.
2400–2570	Draws plot and terminates program.
3000–3420	Subroutine to draw characters O, E, +, – in high resolution.
3500–3530	Subroutine to draw off-axis line at H,V.
4100–4910	Subroutines to draw digits 1–9 in high resolution.
5000–5090	Subroutine to draw scaling values.
6000–6050	Error trap for program errors.

MAIN VARIABLES

XL, XM, XU	Lower, middle, upper scale values of X.
YL, YM, YU	Lower, middle, upper scale values of Y.
DX, DY	Scale increments of X, Y.
X, Y	Current values of X, Y.
XL$, XM$, XU$	String representation of XL, XM, XU.
YL$, YM$, YU$	String representation of YL, YM, YU.

D$, A$	Temporary string variables.
D	Numeric value of D$.
H, V	Horizontal, vertical position in hi-res units.
J	Loop index.
MN, MX	Minimum, maximum values of Y.
XZ, L	Work variables.
HP, VP	Horizontal, vertical position of axes origin.
YY	Y direction offset in hi-res units.
AF	Y position flag (0=in bounds, 1=out of bounds).
H4, V3, V6	H, V offsets.

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.

INTEGRAL

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

$$\int_{x=\ell}^{x=u} f(x)\,dx \cong \frac{\Delta}{3}\Bigg\{ f(\ell)+f(u) + 4\sum_{j=1}^{N/2} f[\ell+\Delta(2j-1)] + 2\sum_{j=1}^{(N-2)/2} f[\ell+2\Delta j]\Bigg\}$$

Here N is the number of segments into which the total interval is divided. N is 4 in the diagram.

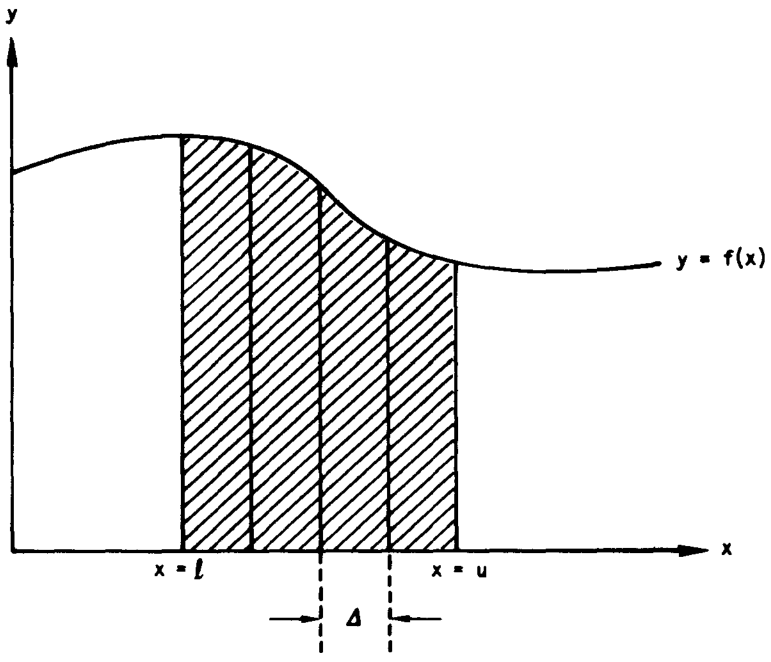


Figure 1. The Integral of f(x)

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.

HOW TO USE IT

The program begins with a warning! This is remind you that you should have already entered the subroutine to evaluate Y as a function of X. This subroutine must start at line 200. 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 **BREAK** 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 ATARI hours, days, and weeks to compute steps. Eventually, round-off errors begin degrading the results, causing a nice, constant, converged solution to change.

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 200. 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 200.

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
```

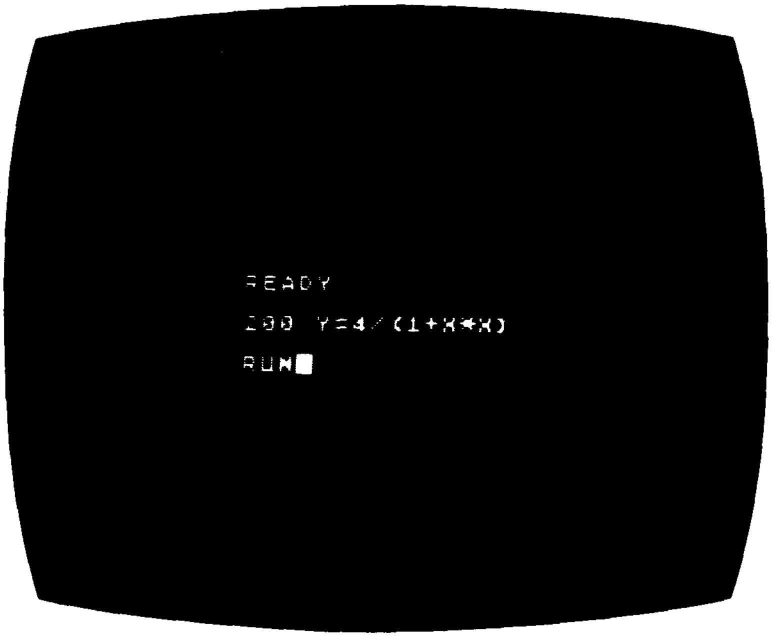
where A, B, C, and D are constants, the program will calculate the integral exactly even with only two segments.

SAMPLE RUN

The sample run illustrates the following integration:

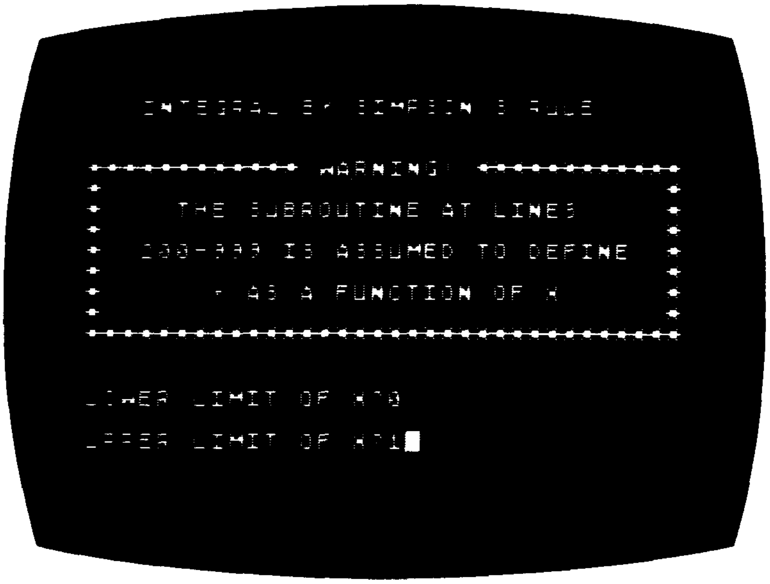
$$\int_{x=0}^{x=1} \frac{4}{1+x^2}\,dx$$

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 the line 200. Note how the function converges and then diverges as errors build up in the calculations.



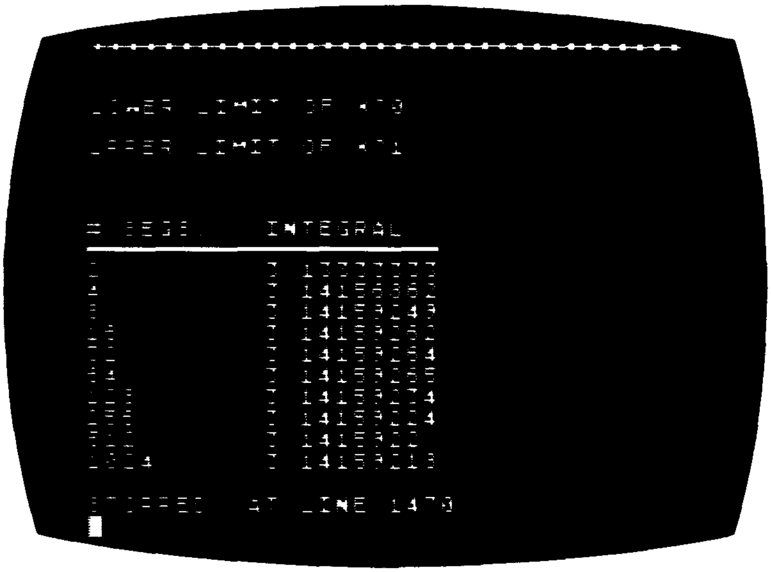
```
READY
200 Y=4/(1+X*X)
RUN
```

The operator enters the integrand function at line 200 and types RUN to start the program.



```
      INTEGRAL BY SIMPSON'S RULE

*************  WARNING!  *************
*                                    *
*     THE SUBROUTINE AT LINES        *
*                                    *
*  100-999 IS ASSUMED TO DEFINE      *
*                                    *
*       Y AS A FUNCTION OF X         *
*                                    *
**************************************

LOWER LIMIT OF X?0

UPPER LIMIT OF X?1
```

The upper and lower bounds of the integration are entered as requested.



```
**************************************

LOWER LIMIT OF X?0

UPPER LIMIT OF X?1

# SEGS.      INTEGRAL
--------------------------
2            3.13333333
4            3.14156862
8            3.14159249
16           3.14159261
32           3.14159264
64           3.14159265
128          3.14159274
256          3.14159214
512          3.1415921
1024         3.14159213

STOPPED  AT LINE 1470
```

The results are computed up to 1024 segments. Then **BREAK** is pressed to terminate the calculation.

PROGRAM LISTING

```
10 REM INTEGRAL
15 REM COPYRIGHT 1984 DILITHIUM PRESS
130 GRAPHICS 0:PRINT CHR$(125)
150 N=2:GOTO 1200
170 REM
180 REM ** ENTER SUBROUTINE AT 200 **
190 REM
200 REM **** Y=F(X) GOES HERE
999 RETURN
1200 PRINT "   INTEGRAL BY SIMPSON'S RULE":PRINT
 :PRINT
1210 PRINT "************ WARNING! ************"
1220 GOSUB 1290
1230 PRINT "*    THE SUBROUTINE AT LINES";:GOSUB
 1285
1240 GOSUB 1290
1250 PRINT "*  200-999 IS ASSUMED TO DEFINE";:GO
SUB 1285:GOSUB 1290
1260 PRINT "*       Y AS A FUNCTION OF X";:GOSUB
1285:GOSUB 1290
1270 PRINT "**********************************"
1280 PRINT :GOTO 1295
1285 POSITION 35,PEEK(84):PRINT "*":RETURN
1290 PRINT "*";:POSITION 35,PEEK(84):PRINT "*":R
ETURN
1295 TRAP 2000
1300 XZ=1:PRINT :PRINT "LOWER LIMIT OF X";:INPUT
 L
1310 XZ=2:PRINT :PRINT "UPPER LIMIT OF X";:INPUT
 U
1360 PRINT :PRINT :PRINT
1370 PRINT "# SEGS.","INTEGRAL"
1380 FOR J=1 TO 20:PRINT CHR$(18);:NEXT J:PRINT
1400 DX=(U-L)/N:T=0
1410 X=L:GOSUB 200:T=T+Y
1420 X=U:GOSUB 200:T=T+Y
1450 M=N/2:Z=0
1460 FOR J=1 TO M
1470 X=L+DX*(2*J-1):GOSUB 200:Z=Z+Y
1480 NEXT J:T=T+4*Z
1500 M=M-1:IF M=0 THEN 1600
1510 Z=0:FOR J=1 TO M
1520 X=L+DX*2*J:GOSUB 200:Z=Z+Y
1530 NEXT J:T=T+2*Z
1600 A=DX*T/3
1610 PRINT N,A
1620 N=N*2
1630 GOTO 1400
2000 IF PEEK(195)<>8 THEN 2050
```

```
2010 TRAP 2000:PRINT :PRINT "** ILLEGAL ENTRY **
"
2020 IF XZ=1 THEN 1300
2030 IF XZ=2 THEN 1310
2050 END
```

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 line will cause the program to stop after the integral is evaluated for a number of segments greater than or equal to 100.

   1615 IF N> =100 THEN 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 1620. To increase the rate of change, try

   1620 N=N*4

   To change it at a constant (and slower) rate, try

   1620 N=N+50

   Be sure, however, that the value of N is always even.

MAIN ROUTINES

130– 150	Initializes variables and goes to mainline routine.
200– 999	User-supplied subroutine to evaluate f(X).
1200–1295	Displays introductory messages and warning.
1300–1310	Gets integration limits from operator.
1360–1380	Displays column headings.
1400–1420	Computes integral contribution from end points.
1450–1480	Adds contribution from one summation.
1500–1530	Adds contribution from other summation.
1600–1630	Completes integral calculation and displays it. Increases number of segments and restarts calculation.
2000–2050	Error trap for input errors.

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 $y=f(x)$.
XZ	Error trap flag.

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.
2. Modify the program to compute answers to "double precision" or greater; i.e., at least fifteen significant digits. Try the function used in the Sample Run to see if you can calculate pi to this high degree of precision.

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_i 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?

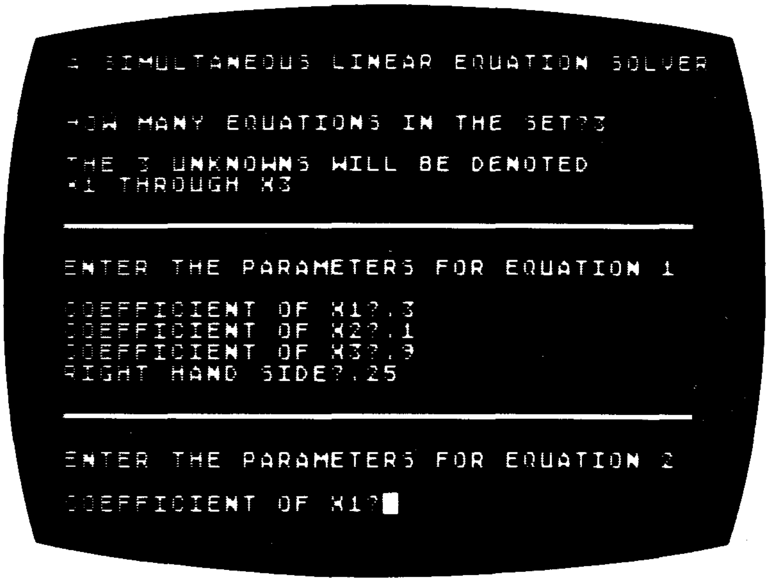
Solution: Let X_1=percent of dark green to use,
X_2=percent of light green to use,
X_3=percent of pure blue to use.

The problem leads to these three simultaneous equations to solve:

$$\begin{aligned} 0.3\,X_1 + 0.1\;\;X_2 + 0.9\,X_3 &= 0.25 \\ 0.2\,X_1 + 0.35\,X_2 \qquad\quad &= 0.25 \\ X_1 + \qquad X_2 + \quad X_3 &= 1.0 \end{aligned}$$

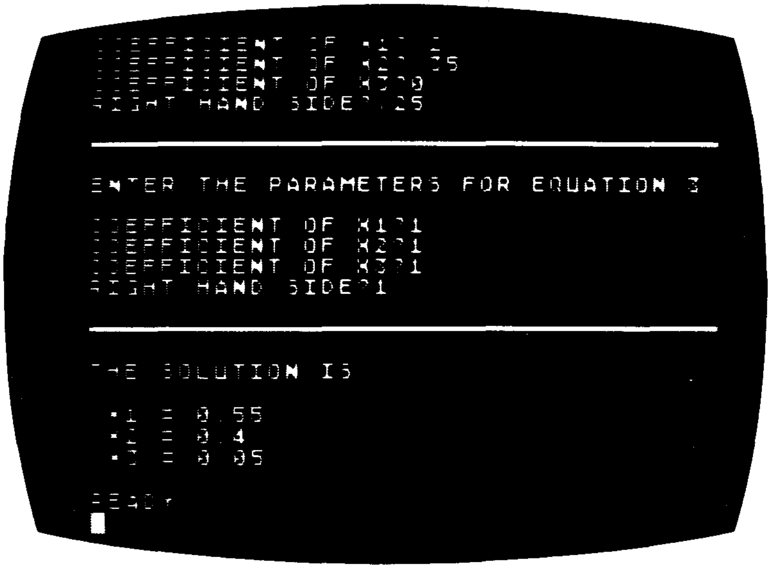
The first equation expresses the amount of blue pigment in the mixture. The second equation is for the yellow pigment. The third equation states that the mixture is composed entirely of the three given paints. (Note that all the percentages are expressed as numbers from 0–1.) The problem leads to the following use of SIMEQN.

SAMPLE RUN



```
* SIMULTANEOUS LINEAR EQUATION SOLVER

HOW MANY EQUATIONS IN THE SET?3

THE 3 UNKNOWNS WILL BE DENOTED
X1 THROUGH X3
________________________________________

ENTER THE PARAMETERS FOR EQUATION 1

COEFFICIENT OF X1?.3
COEFFICIENT OF X2?.1
COEFFICIENT OF X3?.9
RIGHT HAND SIDE?.25
________________________________________

ENTER THE PARAMETERS FOR EQUATION 2

COEFFICIENT OF X1?█
```

The operator chooses to solve a set of three simultaneous equations and then enters the coefficients for the first equations.



```
COEFFICIENT OF X1?.2
COEFFICIENT OF X2?.35
COEFFICIENT OF X3?0
RIGHT HAND SIDE?.25
________________________________________

ENTER THE PARAMETERS FOR EQUATION 3

COEFFICIENT OF X1?1
COEFFICIENT OF X2?1
COEFFICIENT OF X3?1
RIGHT HAND SIDE?1
________________________________________

THE SOLUTION IS

 X1 = 0.55
 X2 = 0.4
 X3 = 0.05

READY
█
```

The coefficients for the remaining two equations 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

```
10 REM SIMEQN
15 REM COPYRIGHT 1984 DILITHIUM PRESS
130 GRAPHICS 0
150 M=20
160 TRAP 3000
200 PRINT CHR$(125)
210 PRINT "A SIMULTANEOUS LINEAR EQUATION SOLVER
"
220 PRINT :PRINT
300 XZ=1:PRINT "HOW MANY EQUATIONS IN THE SET";:
INPUT N
310 PRINT :N=INT(N):IF N>0 AND N<=M THEN 330
320 PRINT " ** ERROR ** IT MUST BE BETWEEN 1-";M
:PRINT :GOTO 300
330 DIM A(N,N),R(N),V(N)
340 PRINT "THE ";N;" UNKNOWNS WILL BE DENOTED"
350 PRINT "X1 THROUGH X";N
360 GOSUB 900:FOR J=1 TO N
370 PRINT "ENTER THE PARAMETERS FOR EQUATION ";J
380 PRINT :FOR K=1 TO N
390 PRINT "COEFFICIENT OF X";K;
400 XZ=3:INPUT XX:A(J,K)=XX:NEXT K
410 XZ=2:PRINT "RIGHT HAND SIDE";:INPUT XX:R(J)=
XX
420 GOSUB 900:NEXT J
430 GOSUB 2000
500 PRINT "THE SOLUTION IS":PRINT
510 FOR J=1 TO N
520 PRINT " X";J;" = ";V(J)
530 NEXT J:END
900 PRINT :FOR L=1 TO 36:PRINT CHR$(18);:NEXT L
910 PRINT :PRINT :RETURN
2000 IF N=1 THEN V(1)=R(1)/A(1,1):RETURN
2010 FOR K=1 TO N-1
2020 I=K+1
2030 L=K
2040 IF ABS(A(I,K))>ABS(A(L,K)) THEN L=I
2050 IF I<N THEN I=I+1:GOTO 2040
2060 IF L=K THEN 2100
2070 FOR J=K TO N:Q=A(K,J):A(K,J)=A(L,J)
2080 A(L,J)=Q:NEXT J
2090 Q=R(K):R(K)=R(L):R(L)=Q
2100 I=K+1
2110 Q=A(I,K)/A(K,K):A(I,K)=0
2120 FOR J=K+1 TO N:A(I,J)=A(I,J)-Q*A(K,J):NEXT
J
2130 R(I)=R(I)-Q*R(K):IF I<N THEN I=I+1:GOTO 211
0
2140 NEXT K
```

```
2150 V(N)=R(N)/A(N,N):FOR I=N-1 TO 1 STEP -1
2160 Q=0:FOR J=I+1 TO N:Q=Q+A(I,J)*V(J)
2170 V(I)=(R(I)-Q)/A(I,I):NEXT J:NEXT I
2180 RETURN
3000 IF PEEK(195)<>8 THEN 3050
3010 TRAP 3000:PRINT :PRINT "** ILLEGAL ENTRY **
":PRINT
3020 IF XZ=1 THEN 300
3030 IF XZ=2 THEN 410
3040 IF XZ=3 THEN 390
3050 END
```

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 150 to the maximum number of equations needed. For example, to allow up to 50 equations in 50 unknowns, change line 150 to read

   150 M=50

2. You may be surprised sometime to see the program terminate without displaying a solution. This probably 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 this line:

   2145 IF A(N,N)=0 THEN PRINT
   "ILL-CONDITIONED INPUT":STOP

MAIN ROUTINES

130– 220	Initializes variables, clears screen and displays program title.
300– 430	Gets input from user and calculates the solution.
500– 530	Displays the solution.
900– 910	Subroutine to space and separate the output.
2000–2180	Subroutine to calculate the solution; consisting of the following parts:
2000	Forms solution if N=1.
2010–2140	Gaussian elimination.

2030–2100	Interchanges rows to achieve pivotal condensation.
2150–2180	Back substitution.
3000–3050	Error trap for illegal input.

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, XX, XZ	Work variables.
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 **RETURN** 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. You do not need to hit the **RETURN** 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. Identifiers can be any length from one to ten characters. Anything in excess of ten characters will be ignored.

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.

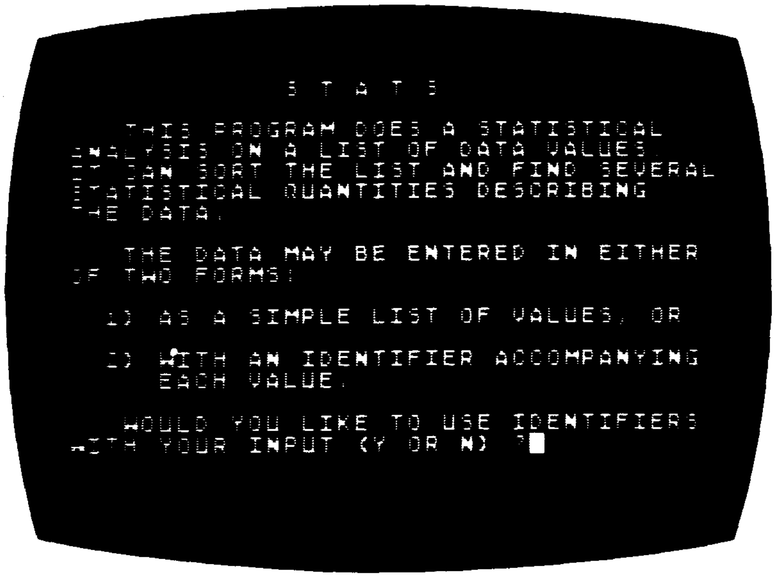
These measures of location give information about the average value of the data. However, they give no idea of how the data is dispersed or spread out around this "average." For that we need "measures of dispersion" or as they are sometimes called, "measures of variation." The simplest of these is the *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.

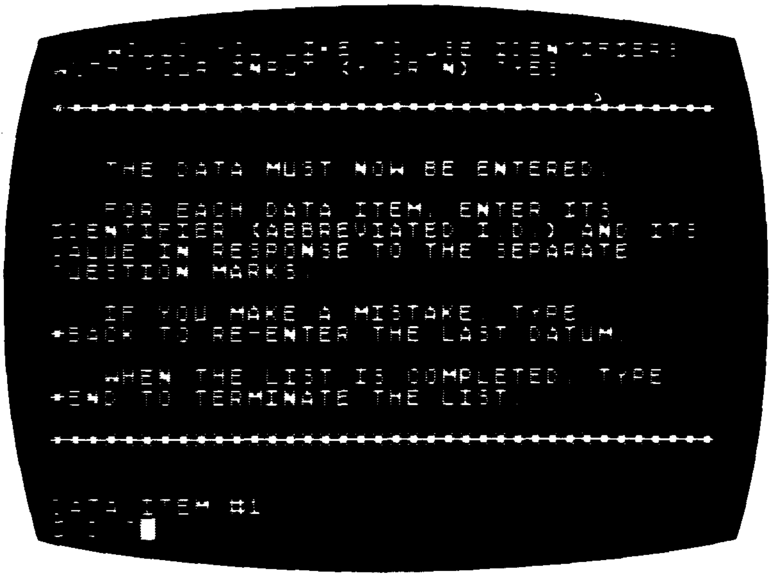
One word of warning—the first time option 2 or 3 is selected, the program must take some time to sort the data into numerical order. The time this requires depends upon how many items are on the list and how badly they are out of sequence. Average times are about thirty seconds for 25 items, ninety seconds for 50 items, and over six minutes for 100 items. The ATARI 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



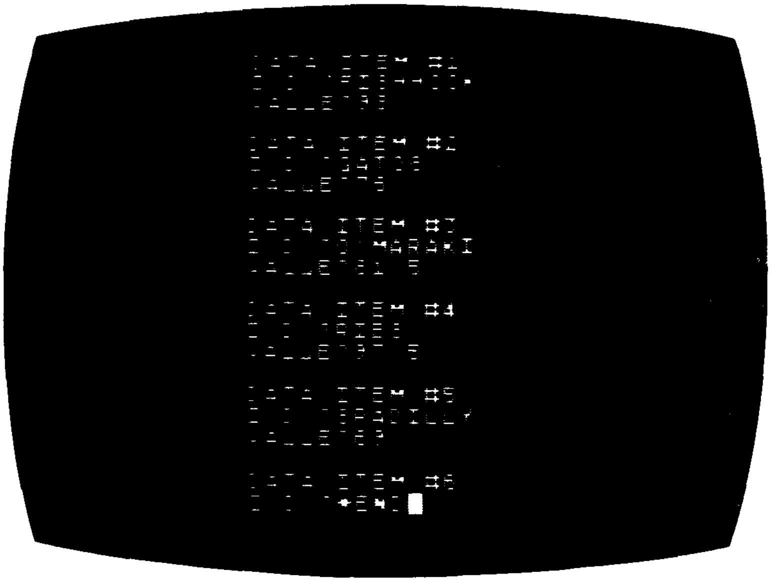
```
              S T A T S

   THIS PROGRAM DOES A STATISTICAL
ANALYSIS ON A LIST OF DATA VALUES.
IT CAN SORT 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) ?
```

The program describes its wares. It asks whether or not the operator wishes to use identifiers with his or her input data.



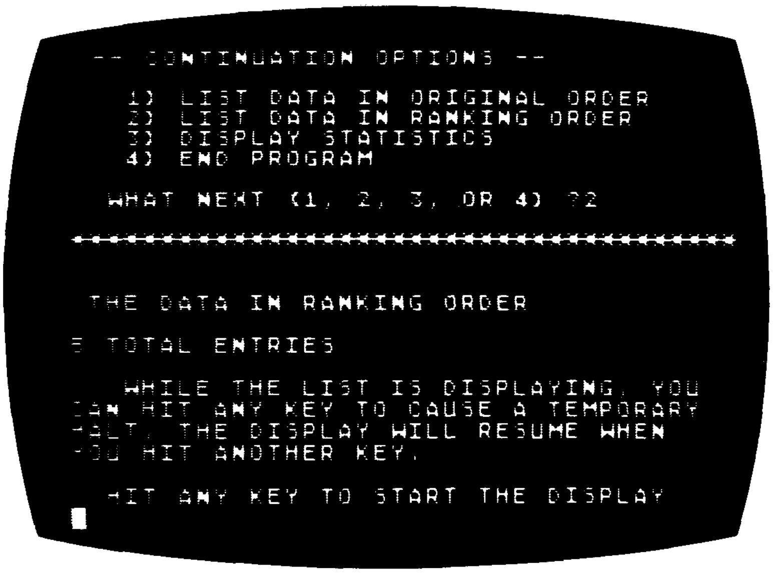
```
   WOULD YOU LIKE TO USE IDENTIFIERS
WITH YOUR INPUT (Y OR N) ?YES
**************************************

   THE DATA MUST NOW BE ENTERED.

   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 COMPLETED, TYPE
*END TO TERMINATE THE LIST.

**************************************

DATA ITEM #1
I.D. ?
```

The operator wishes to use identifiers. The program explains how data is to be entered; it is ready to receive the operator's input.



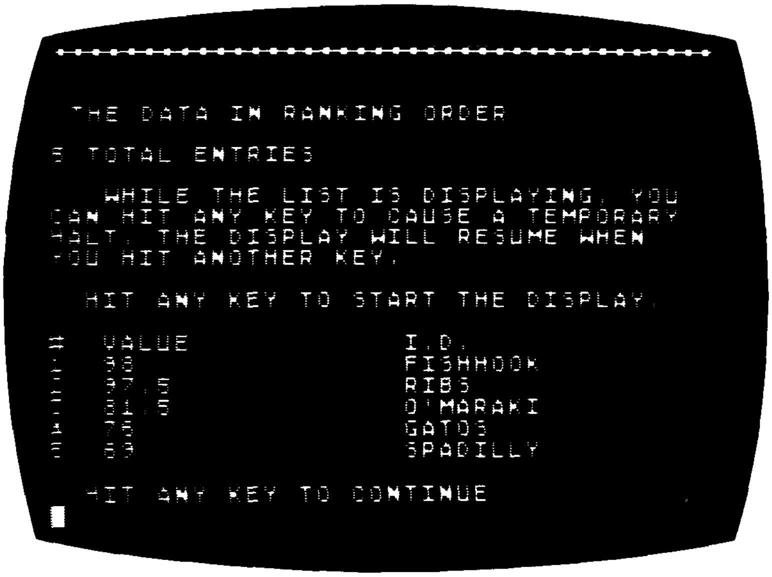
```
DATA ITEM #1
ID ? [illegible]
VALUE ? [illegible]

DATA ITEM #2
ID ? [illegible]
VALUE ? [illegible]

DATA ITEM #3
ID ? [illegible]
VALUE ? [illegible]

DATA ITEM #4
ID ? [illegible]
VALUE ? [illegible]

DATA ITEM #5
ID ? [illegible]
VALUE ? [illegible]

DATA ITEM #6
ID ? *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.



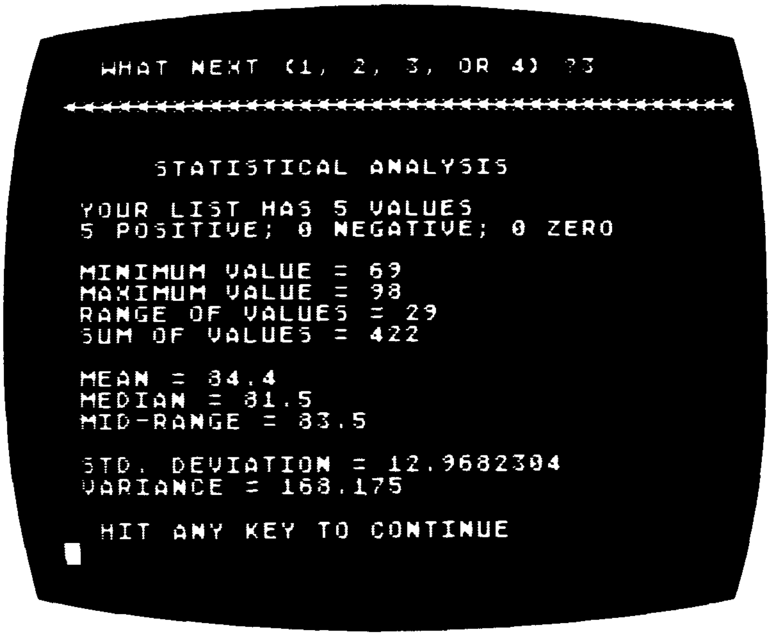
```
-- CONTINUATION OPTIONS --

 1) LIST DATA IN ORIGINAL ORDER
 2) LIST DATA IN RANKING ORDER
 3) DISPLAY STATISTICS
 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
```

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  87.5                  RIBS
3  81.5                  O'MARAKI
4  75                    GATOS
5  69                    SPADILLY

  HIT ANY KEY TO CONTINUE
```

The operator hits a key and is shown the data list in ranking order. The program waits for the pressing of a key to continue.



```
  WHAT NEXT (1, 2, 3, OR 4) ?3

      STATISTICAL ANALYSIS

 YOUR LIST HAS 5 VALUES
 5 POSITIVE; 0 NEGATIVE; 0 ZERO

 MINIMUM VALUE = 69
 MAXIMUM VALUE = 98
 RANGE OF VALUES = 29
 SUM OF VALUES = 422

 MEAN = 84.4
 MEDIAN = 81.5
 MID-RANGE = 83.5

 STD. DEVIATION = 12.9682304
 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

```
10 REM STATS
15 REM COPYRIGHT 1984 DILITHIUM PRESS
140 DIM B$(8),E$(8),N$(40),R$(40)
150 B$="*BACK":E$="*END"
160 MX=100
170 DIM D$(MX*40),V(MX),Z(MX)
180 Z(0)=0:N$=CHR$(32):N$(40)=N$:N$(2)=N$
200 GRAPHICS 0:POKE 82,2:PRINT CHR$(125)
210 POSITION 14,2:PRINT "S T A T S":PRINT
220 PRINT "   THIS PROGRAM DOES A STATISTICAL"
230 PRINT "ANALYSIS ON A LIST OF DATA VALUES."
240 PRINT "IT CAN SORT THE LIST AND FIND SEVERAL
"
250 PRINT "STATISTICAL QUANTITIES DESCRIBING"
260 PRINT "THE DATA.":PRINT
270 PRINT "   THE DATA MAY BE ENTERED IN EITHER"
280 PRINT "OF TWO FORMS:":PRINT
290 PRINT "  1) AS A SIMPLE LIST OF VALUES, OR":
PRINT
300 PRINT "  2) WITH AN IDENTIFIER ACCOMPANYING"
310 PRINT "     EACH VALUE.":PRINT
320 PRINT "   WOULD YOU LIKE TO USE IDENTIFIERS"
330 PRINT "WITH YOUR INPUT (Y OR N) ?";
335 POKE 764,255
340 R=PEEK(764):IF R=255 THEN 340
350 IF R=43 THEN PRINT "YES":F=1:GOTO 400
360 IF R=35 THEN PRINT "NO":F=0:GOTO 400
370 GOTO 340
400 POKE 764,255:GOSUB 2100:PRINT
410 PRINT "   THE DATA MUST NOW BE ENTERED."
420 PRINT :IF F=1 THEN 460
430 PRINT "   ENTER EACH VALUE SEPARATELY IN"
440 PRINT "RESPONSE TO THE QUESTION MARK."
450 GOSUB 2000:GOTO 500
460 PRINT "   FOR EACH DATA ITEM, ENTER ITS"
470 PRINT "IDENTIFIER (ABBREVIATED I.D.) AND ITS
"
480 PRINT "VALUE IN RESPONSE TO THE SEPARATE"
490 PRINT "QUESTION MARKS.":GOSUB 2000
500 GOSUB 2100:POKE 764,255:N=1
510 IF N<1 THEN N=1
520 PRINT :PRINT "DATA ITEM #";N
530 IF F=0 THEN D$((N-1)*40+1,(N-1)*40+10)=N$:GO
TO 570
540 PRINT "I.D.";:INPUT R$:IF R$=E$ THEN 700
545 IF R$="" THEN 540
550 IF R$=B$ THEN N=N-1:GOTO 510
552 IF LEN(R$)>=10 THEN 560
555 R$(LEN(R$)+1,10)=N$(1,10-LEN(R$))
```

```
560 D$((N-1)*40+1,(N-1)*40+10)=R$(1,10)
570 PRINT "VALUE";:INPUT R$:IF R$=E$ THEN 700
580 IF R$=B$ AND F=1 THEN 520
590 IF R$=B$ THEN N=N-1:GOTO 510
592 TRAP 595:V(N)=VAL(R$):GOTO 600
595 PRINT " ** ILLEGAL ENTRY **":TRAP 40000:GOTO
 570
600 TRAP 40000
610 IF N=MX THEN PRINT :PRINT " ** NO MORE DATA
ALLOWED! **":N=N+1:GOTO 700
620 N=N+1:GOTO 510
700 N=N-1:IF N=0 THEN PRINT
710 IF N=0 THEN PRINT " ** NO DATA ENTERED -- RU
N ABORTED **":END
720 GOSUB 2100
730 PRINT :PRINT " -- CONTINUATION OPTIONS --":P
RINT
740 PRINT "   1) LIST DATA IN ORIGINAL ORDER"
750 PRINT "   2) LIST DATA IN RANKING ORDER"
760 PRINT "   3) DISPLAY STATISTICS"
770 PRINT "   4) END PROGRAM"
775 TRAP 3000
780 PRINT :PRINT "  WHAT NEXT (1, 2, 3, OR 4) ";
:INPUT R
790 R=INT(R):IF R<1 OR R>4 THEN 730
800 IF R=4 THEN END
810 ON R GOSUB 1000,1200,1500
820 GOTO 720
1000 GOSUB 2100:PRINT
1010 PRINT " THE ORIGINAL DATA ORDER":PRINT
1020 PRINT N;" TOTAL ENTRIES":GOSUB 2300
1030 PRINT :PRINT "#";:POSITION 5,PEEK(84):PRINT
 "VALUE";
1040 IF F=0 THEN PRINT
1050 IF F=1 THEN POSITION 22,PEEK(84):PRINT "I.D
."
1060 FOR J=1 TO N
1070 PRINT J;:POSITION 5,PEEK(84):PRINT V(J);:PO
SITION 22,PEEK(84)
1075 PRINT D$((J-1)*40+1,(J-1)*40+10)
1080 GOSUB 2500
1090 NEXT J:GOSUB 2900:RETURN
1200 GOSUB 2100:PRINT
1210 PRINT " THE DATA IN RANKING ORDER":PRINT
1220 PRINT N;" TOTAL ENTRIES"
1230 GOSUB 2700
1280 GOSUB 2300:PRINT :PRINT "#";:POSITION 5,PEE
K(84):PRINT "VALUE";
1290 IF F=0 THEN PRINT
1300 IF F=1 THEN POSITION 22,PEEK(84):PRINT "I.D
."
1310 FOR J=1 TO N
```

```
1320 PRINT J;:POSITION 5,PEEK(84):PRINT V(Z(J));
:POSITION 22,PEEK(84)
1325 PRINT D$((Z(J)-1)*40+1,(Z(J)-1)*40+10)
1330 GOSUB 2500
1340 NEXT J:GOSUB 2900:RETURN
1500 GOSUB 2100:PRINT :TRAP 40000
1510 PRINT "      STATISTICAL ANALYSIS":PRINT
1520 PRINT " YOUR LIST HAS ";N;" VALUES"
1530 NP=0:NN=NP:NZ=NP:SQ=NP:W=NP
1540 FOR J=1 TO N:W=W+V(J):SQ=SQ+V(J)*V(J)
1550 IF V(J)>0 THEN NP=NP+1
1560 IF V(J)<0 THEN NN=NN+1
1570 IF V(J)=0 THEN NZ=NZ+1
1590 NEXT J:M=W/N:VA=0:IF N=1 THEN 1610
1600 VA=(SQ-N*M*M)/(N-1)
1610 SD=SQR(VA)
1620 PRINT CHR$(32);NP;" POSITIVE; ";NN;" NEGATI
VE; ";NZ;" ZERO":PRINT
1630 GOSUB 2700:PRINT " MINIMUM VALUE = ";V(Z(N)
)
1640 PRINT " MAXIMUM VALUE = ";V(Z(1))
1650 PRINT " RANGE OF VALUES = ";V(Z(1))-V(Z(N))
1660 PRINT " SUM OF VALUES = ";W:PRINT
1670 PRINT " MEAN = ";M
1680 Q=INT(N/2)+1:MD=V(Z(Q)):IF N/2>INT(N/2) THE
N 1700
1690 MD=(V(Z(Q))+V(Z(Q-1)))/2
1700 PRINT " MEDIAN = ";MD
1710 PRINT " MID-RANGE = ";(V(Z(1))+V(Z(N)))/2
1715 PRINT :PRINT " STD. DEVIATION = ";SD
1720 PRINT " VARIANCE = ";VA
1740 GOSUB 2900:RETURN
2000 PRINT :PRINT "   IF YOU MAKE A MISTAKE, TYP
E"
2010 PRINT B$;" TO RE-ENTER THE LAST DATUM."
2020 PRINT :PRINT "   WHEN THE LIST IS COMPLETED
, TYPE"
2030 PRINT E$;" TO TERMINATE THE LIST.":RETURN
2100 PRINT :FOR J=1 TO 38:PRINT "*";:NEXT J
2110 PRINT :RETURN
2300 PRINT :PRINT "   WHILE THE LIST IS DISPLAYI
NG, YOU"
2310 PRINT "CAN HIT ANY KEY TO CAUSE A TEMPORARY
"
2320 PRINT "HALT. THE DISPLAY WILL RESUME WHEN"
2330 PRINT "YOU HIT ANOTHER KEY."
2340 PRINT :PRINT "  HIT ANY KEY TO START THE DI
SPLAY."
2345 POKE 764,255
2350 IF PEEK(764)=255 THEN 2350
2360 POKE 764,255:RETURN
2500 IF PEEK(764)=255 THEN RETURN
```

```
2510 POKE 764,255
2520 IF PEEK(764)=255 THEN 2520
2530 POKE 764,255:RETURN
2700 IF Z(0)=1 THEN RETURN
2710 FOR J=1 TO N:Z(J)=J:NEXT J:IF N=1 THEN RETU
RN
2720 NM=N-1:FOR K=1 TO N:FOR J=1 TO NM:N1=Z(J)
2730 N2=Z(J+1):IF V(N1)>V(N2) THEN 2750
2740 Z(J+1)=N1:Z(J)=N2
2750 NEXT J:NEXT K:Z(0)=1:RETURN
2900 PRINT :PRINT "  HIT ANY KEY TO CONTINUE"
2905 POKE 764,255
2910 IF PEEK(764)=255 THEN 2910
2920 POKE 764,255:RETURN
3000 IF PEEK(195)<>8 THEN 3050
3010 GOTO 775
3050 END
```

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 upon the maximum dimension parameter, MX. Should your application require more than 100 data values, you will have to increase the value of MX in line 160 accordingly. To accommodate up to 300 data items, make this change

   160 MX=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 line 150. Each string can be up to eight characters long. If you wish to terminate the data with /DONE/ and to back up with /LAST/, for example, line 150 should be:

   150 B$="/LAST/":E$="/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 2730 to a "less than" sign (<). Thus:

   2730 N2=Z(J+1):IF V(N1)<V(N2) THEN 2750

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:

```
1630 GOSUB 2700:PRINT
    "MINIMUM VALUE=";V(Z(1))
1640 PRINT "MAXIMUM VALUE=";V(Z(N))
1650 PRINT "RANGE=";V(Z(N))-V(Z(1))
```

MAIN ROUTINES

140– 180	Initializes constants and dimensioning.
200– 370	Displays messages, determines if identifiers will be used.
400– 620	Gets data from the user.
700– 710	Checks that input contains at least one value.
720– 820	Command mode—gets user's next option and does a GOSUB to it.
1000–1090	Subroutine to list data in the original order.
1200–1340	Subroutine to list data in ranking order.
1500–1740	Subroutine to calculate and display statistics.
2000–2360	Subroutines to display various messages.
2500–2530	Subroutine to allow user to temporarily start and stop display listing.
2700–2750	Subroutine to sort the list in ranking order.
2900–2920	Subroutine to detect if user has hit a key to continue.
3000–3050	Trap routine for program errors.

MAIN VARIABLES

MX	Maximum number of data values allowed.
D$	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 for a null identifier.

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.

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:

```
2720 Q=0:FOR J=1 TO N-1:N1=Z(J)
2730 N2=Z(J+1):IF V(N1)>=V(N2) THEN 2750
2745 Q=1
2750 NEXT J:IF Q=1 THEN 2720
2760 Z(0)=1:RETURN
```

   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. 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 ATARI with the proper "know-how." POWERS will calculate the values of integers raised to various powers, not to the ATARI's "normal" nine-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.

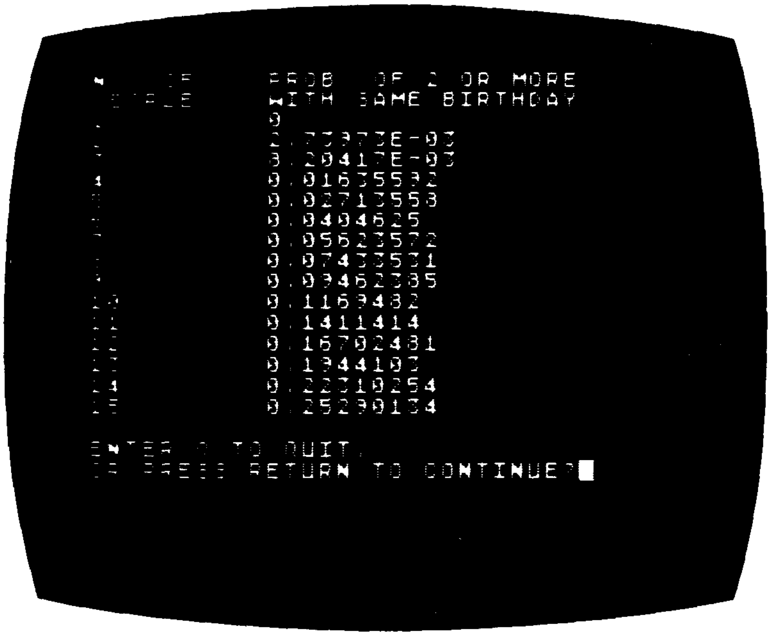
The probability for four people is

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 you enter "Q" to stop it. The program stops every 15 or 16 people and waits for you to press the **RETURN** key. This allows you to examine values before they scroll off the screen.

SAMPLE RUN



```
N    OF         PROB  OF 2 OR MORE
PEOPLE          WITH SAME BIRTHDAY
1               0
2               2.73973E-03
3               8.20417E-03
4               0.01635592
5               0.02713558
6               0.0404625
7               0.05623572
8               0.07433531
9               0.09462385
10              0.1169482
11              0.1411414
12              0.16702481
13              0.1944103
14              0.22310254
15              0.25290134

ENTER Q TO QUIT,
OR PRESS RETURN TO CONTINUE?█
```

After the probability of 15 people with duplicate birthdays is shown, the option to quit or continue is offered.

PROGRAM LISTING

```
10 REM BIRTHDAY
15 REM COPYRIGHT 1984 DILITHIUM PRESS
110 DIM X$(1):N=1:Q=1
120 GRAPHICS 0:POKE 82,2
130 PRINT CHR$(125):PRINT "NO. OF    PROB. OF 2
OR MORE"
140 PRINT "PEOPLE    WITH SAME BIRTHDAY"
150 PRINT N,1-Q
160 Q=Q*(365-N)/365
170 N=N+1:IF ((N/16)-INT(N/16))*16<>0 THEN 150
180 PRINT :PRINT "ENTER Q TO QUIT,":PRINT "OR PR
ESS RETURN TO CONTINUE";:INPUT X$
190 IF X$<>"Q" THEN 130
200 END
```

EASY CHANGES

1. To make the program stop after a specific number of people (such as 50), make these changes:

```
170 N=N+1: IF N< =50 THEN 150
175 END
```

MAIN ROUTINES

110–120	Initializes variables and clears screen.
130–140	Displays headings.
150–160	Calculates probability of no duplication, then displays probability of duplication.
170–200	Adds a person. Determines if screen is full. Waits for key to be entered.

MAIN VARIABLES

N	Number of people in the room.
Q	Probability of no duplication of birthdays.
X$	Input string.

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.

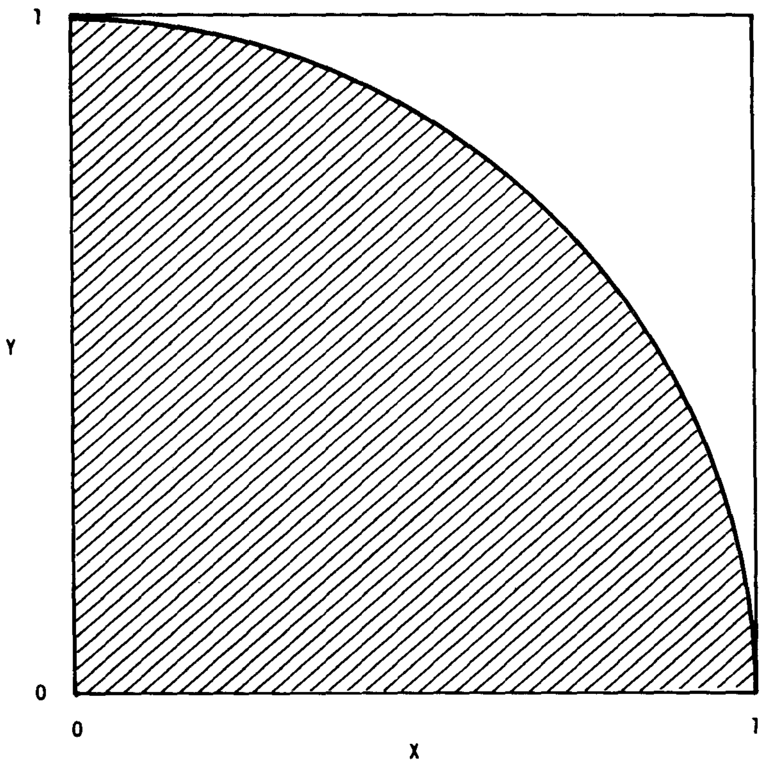


Figure 1. The PI Dartboard

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.

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

$$\text{RATIO} = \frac{\#\text{ hits}}{\#\text{ thrown}} = \frac{\text{shaded area}}{\text{entire area}} = \frac{\pi/4}{1} = \frac{\pi}{4}$$

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 ATARI 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.

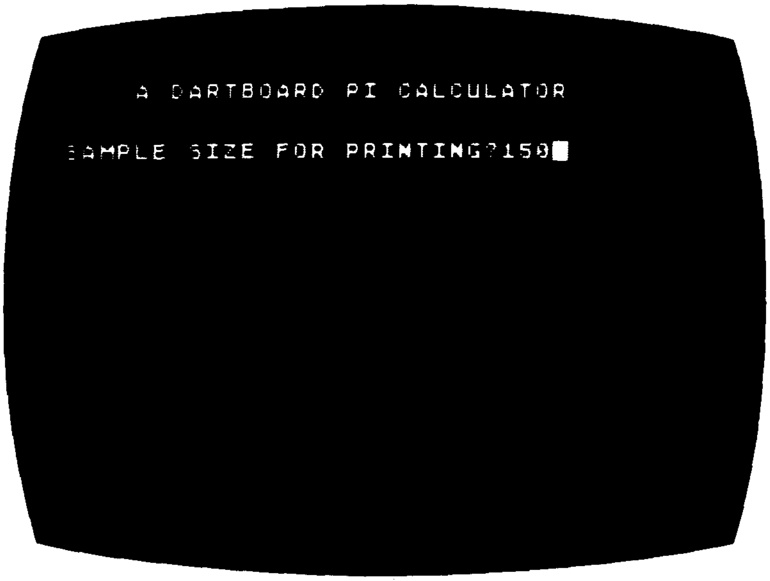
HOW TO USE IT

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.

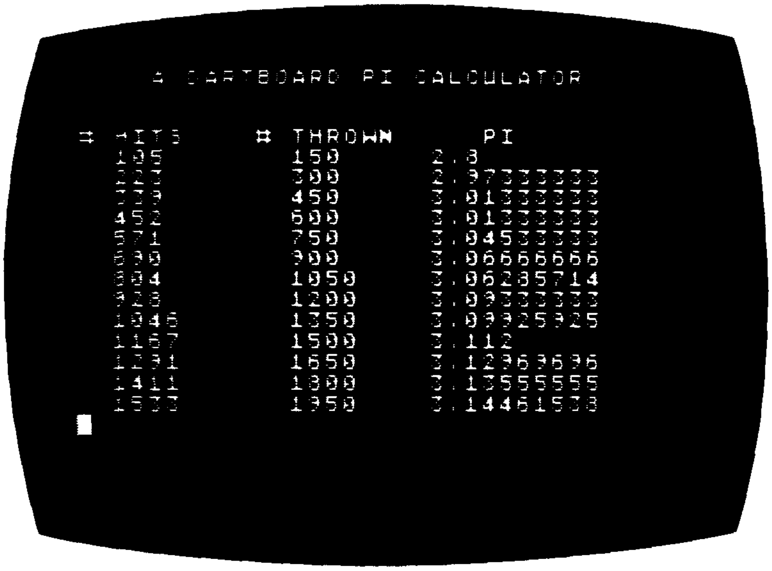
This will continue until you press **BREAK**. When you are satisfied with the total number of darts thrown, press **BREAK** to terminate the program execution.

SAMPLE RUN



```
      A DARTBOARD PI CALCULATOR

SAMPLE SIZE FOR PRINTING?150
```

The operator selects 150 for the printing sample size.



```
       A DARTBOARD PI CALCULATOR

# HITS      # THROWN      PI
  105         150        2.8
  223         300        2.97333333
  339         450        3.01333333
  452         600        3.01333333
  571         750        3.04533333
  690         900        3.06666666
  804         1050       3.06285714
  928         1200       3.09333333
  1046        1350       3.09925925
  1167        1500       3.112
  1291        1650       3.12969696
  1411        1800       3.13555555
  1533        1950       3.14461538
```

1950 darts are "thrown". The program will continue until the BREAK key is pressed.

PROGRAM LISTING

```
10 REM PI
15 REM COPYRIGHT 1984 DILITHIUM PRESS
160 T=0:TH=T
200 TRAP 1000
300 GOSUB 600
310 PRINT "SAMPLE SIZE FOR PRINTING";:INPUT NP
320 NP=INT(NP):IF NP<1 THEN 300
330 GOSUB 600
340 PRINT "# HITS","# THROWN",CHR$(32);CHR$(32);
CHR$(32);"PI"
400 GOSUB 500:TH=TH+NH:T=T+NP:P=4*TH/T
410 PRINT CHR$(32);CHR$(32);TH,CHR$(32);CHR$(32)
;T,P
420 GOTO 400
500 NH=0:FOR J=1 TO NP
510 X=RND(1):Y=RND(1)
520 IF (X*X+Y*Y)<1 THEN NH=NH+1
530 NEXT J:RETURN
600 PRINT CHR$(125):POKE 82,2:POSITION 6,PEEK(84
)
610 PRINT "A DARTBOARD PI CALCULATOR"
620 PRINT :PRINT :RETURN
1000 IF PEEK(195)=8 THEN TRAP 1000:GOTO 300
```

EASY CHANGES

1. If you want the program to always use a fixed sample size, change line 310 to read

   ```
   310 NP=150
   ```

   Of course, the value of 150 given here may be changed to whatever you wish. With this change, line 320 is 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 two lines:

   ```
   315 PRINT "TOTAL # DARTS TO THROW":INPUT
       ND
   415 IF T> =ND THEN 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

160–200	Initializes.
300–340	Gets operator input, displays column headings.
400–420	Calculates and displays results.
500–530	Throws NP darts and records number of "hits."
600–620	Clears screen and displays program title.
1000	Error trap for illegal inputs.

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 PI, which is set to the value of pi. Then the percentage error, PE, can be calculated as:

   PE=100*ABS(P−PI)/PI

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 line 510 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 ATARI keeps track of nine significant digits when dealing with numbers. For integers less than one billion (1,000,000,000), the ATARI can retain the precision value of the number. But for larger integers the ATARI only keeps track of the most significant (leftmost) nine digits, plus the exponent. This means, of course, that there is no way you can use the ATARI to deal with precise integers greater than one 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

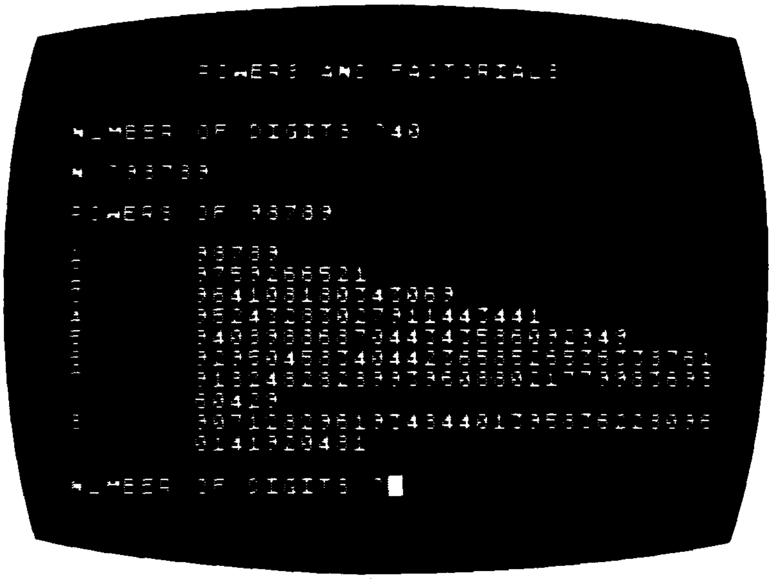
The program first asks you how many digits long you want the largest number to be. This can be any integer from 1 to 250. So, for example, if you enter 40, you will get answers up to 40 digits long.

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      9408988687044343586092949
6      929504583404423658526536338761
7      918248282899396088021779983698
       60429
8      907128296193484401395836228096
       0141920481

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

```
10 REM POWERS
15 REM COPYRIGHT 1984 DILITHIUM PRESS
110 DIM N$(10)
120 GRAPHICS 0:POKE 82,2:PRINT CHR$(125)
125 TRAP 1000
130 POSITION 9,1:PRINT "POWERS AND FACTORIALS"
```

```
140 PRINT :PRINT
150 DIM N(255)
160 FOR J=1 TO 255:N(J)=0:NEXT J
170 XZ=1:PRINT "NUMBER OF DIGITS ";:INPUT M
175 IF M=0 THEN END
180 M=INT(M):IF M>250 OR M<1 THEN 170
190 XZ=2:PRINT :PRINT "N ";:INPUT N
200 N=INT(N)
210 IF N<1 OR N>100000 THEN 190
220 PRINT
230 F=0:IF N=1 THEN F=1:PRINT "FACTORIALS"
240 IF F=0 THEN PRINT "POWERS OF ";N:PRINT
250 T=10:K=1:N(0)=N
260 FOR J=0 TO M
270 IF N(J)<T THEN 300
280 Q=INT(N(J)/T):W=N(J)-Q*T
290 N(J)=W:N(J+1)=N(J+1)+Q
300 NEXT J
310 J=M+1
320 IF N(J)=0 THEN J=J-1:GOTO 320
330 IF J>=M THEN 500
340 D=0:PRINT K,CHR$(30);CHR$(30);CHR$(30);
350 N$=STR$(N(J))
360 D=D+1:IF D>30 THEN D=1:PRINT :POSITION 9,PEE
K(84)
370 PRINT N$;:J=J-1:IF J>=0 THEN 350
380 N=N+F
390 K=K+1:PRINT
400 FOR J=0 TO M:N(J)=N(J)*N:NEXT J
410 GOTO 260
500 M=0:PRINT
510 GOTO 160
1000 IF PEEK(195)<>8 THEN 1050
1010 TRAP 1000:PRINT "** ILLEGAL ENTRY **"
1020 IF XZ=1 THEN 170
1030 IF XZ=2 THEN 190
1050 END
```

EASY CHANGES

1. To change the program so that it always uses, say, 50-digit numbers, remove line 170, and insert this line:

   170 M=50

2. To clear the screen before the output begins being displayed, change line 220 to say:

   220 GRAPHICS 0

3. If 250 digits isn't enough for you, you can go higher. For 500 digits, make these changes:
   a. In line 150 and 160, change the 255 to 505.
   b. In line 180, change the 250 to 500.

MAIN ROUTINES

110– 160	Displays title. Sets up array for calculations.
170– 240	Asks for number of digits and N. Checks validity of responses. Displays heading.
250	Initializes variables for calculations.
260– 300	Performs "carrying" in N array so each element has a value no larger than nine.
310– 320	Scans backwards through N array for first non-zero element.
330	Checks to see if this value would be larger than the number of digits requested.
340– 370	Displays counter and number. Goes to second line if necessary.
380– 390	Prepares to multiply by N to get next number.
400– 410	Multiplies each digit in N array by N. Goes back to line 260.
500– 510	Prepares for next request. Goes back to 160.
1000–1050	Error trap for illegal inputs.

MAIN VARIABLES

N	Array in which calculations are made.
M	Number of digits of precision requested by operator.
N	Starting value. If 1, factorials. If greater than 1, powers of N.
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 30).
XZ	Work variable for error trap.
N$	String variable used to convert each digit into displayable format.

SUGGESTED PROJECTS

1. Determine the largest N that could be used without errors entering into the calculation (because of intermediate results exceeding one billion), then modify line 210 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 16 sets of values on each screen, and then waits for you to press any key (except **BREAK** or **E**) before it continues with the next 16. 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:

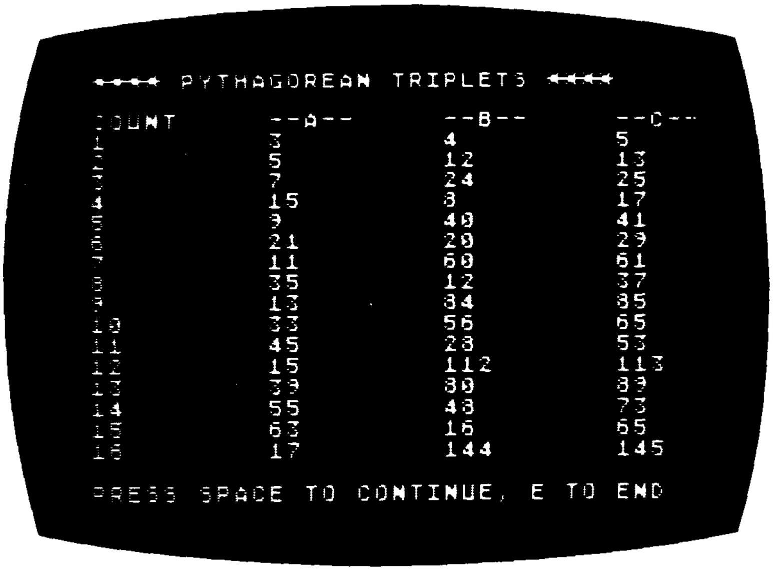
1. R is greater than S,
2. R and S are of opposite parity (one is odd and the other is even), and
3. R and S are relatively prime (they have no common integer divisors except 1),

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 ◆◆◆◆

COUNT      --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

PRESS SPACE TO CONTINUE, E TO END
```

The program generates a screen full of Pythagorean triplets, then waits for the operator to press a key to continue.

PROGRAM LISTING

```
10 REM PYTHAG
15 REM COPYRIGHT 1984 DILITHIUM PRESS
130 R=2:K=1:D=0
150 GOSUB 350
180 S=R-1
190 A=R*R-S*S
200 B=2*R*S
210 C=R*R+S*S
220 PRINT K,A,B,C
230 K=K+1:D=D+1:GOTO 400
240 S=S-2:IF S<=0 THEN R=R+1:GOTO 180
250 S1=S:B1=R
260 N=INT(B1/S1)
270 R1=B1-(S1*N)
280 IF R1<>0 THEN B1=S1:S1=R1:GOTO 260
300 IF S1<>1 THEN 240
320 GOTO 190
350 POKE 752,1:PRINT CHR$(125)
360 PRINT "**** PYTHAGOREAN TRIPLETS ****"
370 PRINT
380 PRINT "COUNT","--A--","--B--","--C--"
390 RETURN
400 IF D<16 THEN 240
420 PRINT :PRINT "PRESS SPACE TO CONTINUE, E TO
END"
425 POKE 764,255
430 X=PEEK(764):IF X=255 THEN 430
440 IF X=42 THEN POKE 752,0:POKE 764,255:END
450 GOSUB 350:D=0
460 GOTO 240
```

EASY CHANGES

1. Alter the starting value of R in line 130. 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 16 in line 400 to a 10, for example. You probably won't want to try a value greater than 20, 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:

```
405 GOTO 450
```

or

```
405 GOTO 460
```

The first will display headings for each screen. The second will only display the headings at the beginning of the run.

MAIN ROUTINES

130	Initializes variables.
150	Displays the title and column headings.
180	Calculates the first value of S for current R value.
190–210	Calculates A, B, and C.
220–230	Displays one line of values. Adds to counters.
240	Calculates next S value. If no more, calculates next R value.
250–300	Determines if R and S are relatively prime.
350–390	Subroutine to display title and column headings.
400–460	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, R1, N	Used in determining if R and S are relatively prime.
X	Key pressed to continue.

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 ATARI's nine-digit integer capacity and produce incorrect results. Can you determine when this will be? Modify the program to stop when this occurs.

Appendix 1
Memory Usage

The programs in this book were written on an ATARI 800 computer with 48K of RAM. They were also tested on ATARI 400, ATARI 600XL, ATARI 800XL, and ATARI 1200XL computers. All the programs should run unmodified on any of these machines, except for some of the longer programs (WALLOONS, GROAN) which will not run in a standard ATARI 400 or 16K ATARI 800 due to memory limitations. These programs can be modified to run by deleting some of the features in the program.

Atari has recently introduced several new computers, including the 1400XL and 1450XL. These machines are purported to be software compatible with the earlier computers, but the programs in this book have not been tested on them.

Several of the chapters in this book mention the limitations of the programs based on the number of "bytes free" in your system. This refers to the memory space available when Atari BASIC has been started, but no BASIC program has yet been entered. To find out how many bytes free your system has, enter the command PRINT FRE(0) after Atari BASIC is started. If this number is 10,000 or more, all the programs in this book will easily fit your system's available RAM. If this number is 4000 or more, all but the longest programs will fit.

Appendix 2

Microsoft BASIC for the Atari

The programs in this book are written in standard Atari BASIC as available in the standard Atari BASIC CXL4002 plug-in ROM cartridge (or built into the computer for the XL series) and defined in the Atari BASIC Reference Manual. Since this BASIC is available for all Atari computers there is no real need to adapt these programs to any other dialect of BASIC. On the other hand, there are a great many programs available written for the many versions of Microsoft BASIC, and it may be desirable to translate these programs into Atari BASIC.

For those interested in experimenting, Atari Microsoft BASIC, ATARI P/N CX8126, is available on diskette. The two BASICs are similar, and programs can be adapted from one to the other with a bit of effort. The principal difference between the two BASICs is their handling of string variables, differences in some of the graphics commands, and differences in file operations. Appendices F and J of the Atari Microsoft BASIC Instruction Manual contain a summary of the differences between the languages. This summary can be used to translate programs from one BASIC format to the other.

USING A DISKETTE OR CASSETTE

If you purchased this book as a sofware package you will need loading instructions to get the programs into your computer. The loading instructions 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.

Bibliography

BOOKS

Bell, R. C., *Board and Table Games From Many Civilizations*, Oxford University Press, London, 1969. (WARI)

Brown, Jerald R., *Instant BASIC*, dilithium Press, Beaverton, Ore., 1977. (Self-teaching text on the BASIC language)

Cohen, Daniel, *Biorhythms in Your Life*, Fawcett Publications, Greenwich, Connecticut, 1976. (BIORHYTH)

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, INTEGRAL, SIMEQN)

Kuo, S. S., *Computer Applications of Numerical Methods*, Addison-Wesley, Reading, Massachusetts, 1972. (CURVE, DIFFEQN, INTEGRAL, SIMEQN)

McCracken, Daniel D., and Dorn, W. S., *Numerical Methods and FORTRAN Programming*, John Wiley and Sons, New York, 1964. (CURVE, DIFFEQN, INTEGRAL, 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. (BIORHYTH)

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 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.

Software Support
Errata – 32 Programs for the ATARI
dilithium Press
8285 S.W. Nimbus
Suite 151
Beaverton, OR 97005

32 BASIC Programs for the ATARI Computer

Loading Instructions

Call our toll free number 800-547-1842 and ask for Software Support with any questions.

What You Need To Run 32 BASIC Programs for the ATARI Computer

ATARI 400, 800, 600XL, 800XL, 1200XL

16K of memory

One disk drive

Standard Monitor or TV with transfer switch for display

BASIC cartridge (unless using an XL model).

ATARI DOS

Preliminary Steps

Before you do anything else, make a backup copy of your *32 BASIC PROGRAMS* disk (see your user's manual for instructions on how to do this). There are two sides of your program disk, (side A and side B), each containing part of the *32 programs for the ATARI Computer*. You may want to copy DOS on to each side of your backup disk. This will eliminate the need for a separate DOS disk.

Follow these simple steps to get the *32 BASIC Programs* up and running on your ATARI computer:

1. With the label side up, insert your DOS diskette into drive 1, and close the door.

2. Turn on the monitor or TV.

3. Turn on the disk drive and then the computer, (the switch is on the right side of the computer). The following message appears:

READY

4. To get a list of the programs on the diskette, type

DOS

and press the RETURN key. Your ATARI DOS SYSTEM menu will appear on the screen. Select Letter A and press the RETURN key. Insert the 32 Basic Programs diskette into Drive 1 with side A facing up, close the door, and press RETURN. A list of the programs on side A of the disk will appear on the screen.

5. Choose a program from the list, for example, GROAN. To run this program, type B and press RETURN. (This will clear the screen and allow you to enter further instructions.) Type

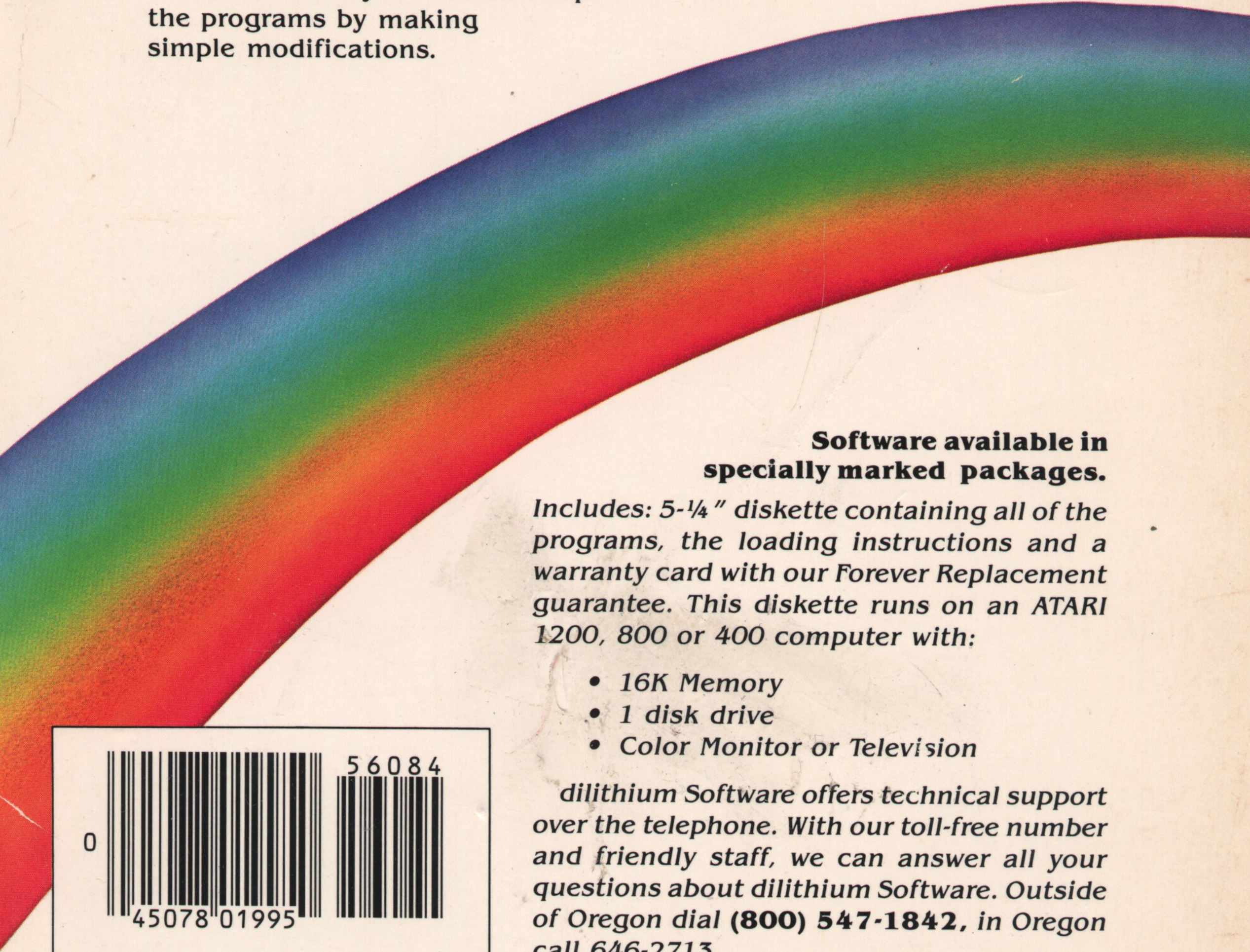
RUN "D1:GROAN"

and press RETURN. Be certain to spell the name exactly as it appears in the directory listing. If you make a mistake, press RETURN (a message telling you there is an error will appear), and retype the instruction.

If at any time you want to stop a program, press the break button (upper right corner). If you want to run another program, go back to Step 4. It is not necessary to get a directory each time you want to run a program. After the diskette is inserted and the computer is turned on, you can type RUN and the name of the program as shown above (remember to press RETURN). The directory is necessary only if you want to view the names of the programs.

©dilithium Press 1983
8285 S.W. Nimbus
Suite 151
Beaverton, OR 97005
(503) 646-2713 or
toll free (800) 547-1842

This book is chock full of completely-tested ready to run programs designed specifically for your Atari 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: 5-¼″ diskette containing all of the programs, the loading instructions and a warranty card with our Forever Replacement guarantee. This diskette runs on an ATARI 1200, 800 or 400 computer with:

- *16K Memory*
- *1 disk drive*
- *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.*

56084
0
45078 01995
ISBN 0-88056-084-3
>>$19.95

dilithium Press
SOFTWARE