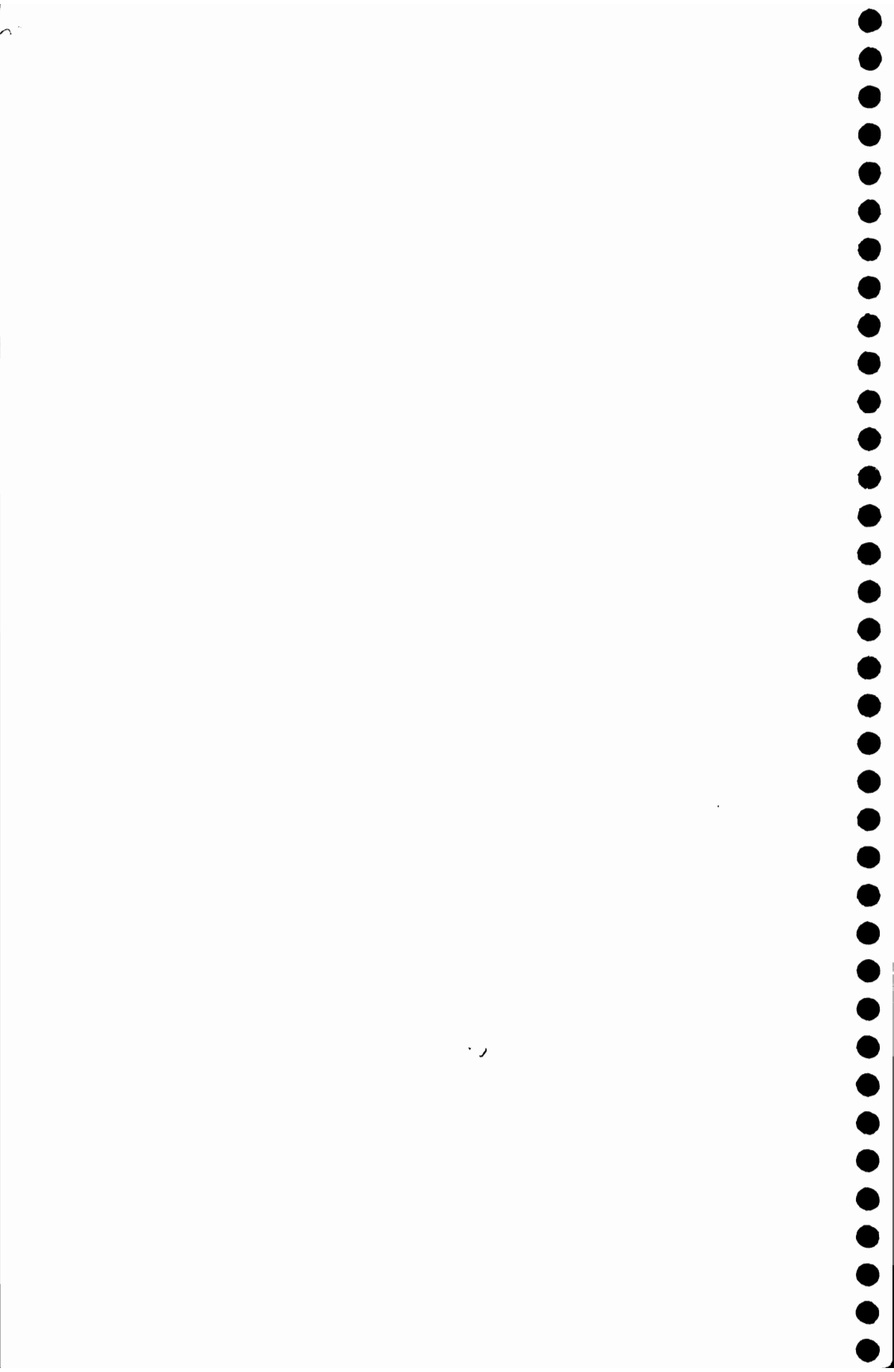


TRAINING YOUR COMMODORE 64

*Master the Commands
that Control Your Computer*

















LEVEL 2





REFERENCE CHART





Frequently Used Keys and how they will look in a program listing.

KEYBOARD	COLOR	DISPLAY	KEYBOARD	COLOR	DISPLAY
CTRL 1	BLACK		G 1	ORANGE	
CTRL 2	WHITE		G 2	BROWN	
CTRL 3	RED		G 3	LT. RED	
CTRL 4	CYAN		G 4	GRAY 1	
CTRL 5	PURPLE		G 5	GRAY 2	
CTRL 6	GREEN		G 6	LT. GREEN	
CTRL 7	BLUE		G 7	LT. BLUE	
CTRL 8	YELLOW		G 8	GRAY 3	

CURSOR MOVEMENT



GRAPHIC SYMBOL FOR THAT CURSOR WHEN YOU LIST THE PROGRAM

-  REVERSED Q
-  REVERSED CIRCLE
-  BRACKET
-  STRAIGHT LINE

MEMORY LOCATIONS FOR GRAPHICS AND MUSIC

GRAPHICS

To change the screen color
POKE 53281

To change the border color
POKE 53280

MUSIC

MEMORY LOCATION

	VOICE 1	VOICE 2
LOW FREQUENCY	54272	54279
HIGH FREQUENCY	54273	54280
WAVEFORM	54276	54283
ATTACK/DECAY	54277	54284
SUSTAIN/RELEASE	54278	54285
VOLUME	54296	54296

COLOR CODE FOR GRAPHIC MEMORY LOCATIONS

0 BLACK	8 ORANGE
1 WHITE	9 BROWN
2 RED	10 LIGHT RED
3 CYAN	11 GRAY 1
4 PURPLE	12 GRAY 2
5 GREEN	13 LIGHT GREEN
6 BLUE	14 LIGHT BLUE
7 YELLOW	15 GRAY 3



TRAINING YOUR COMMODORE 64

*Master the Commands
that control your computer*

LEVEL 2

PUBLISHED BY:
WOODSTOCK SOFTWARE COMPANY
Woodstock Square — Box 886
Woodstock, Illinois 60098

Copyright © Creative Programming Inc. 1984
Under Universal Copyright Convention,
International Copyright Convention,
and Pan-American Copyright Conventions

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical including photocopying, recording or by any information storage and retrieval system, without permission in advance in writing from Creative Programming Inc. and the exclusive publisher, Woodstock Software Company.

Printing 1 2 3 4 5 6 7 8 9 10

ISBN 0-912079-50-9
Printed in the United States of America

TABLE OF CONTENTS

INTRODUCTION

LESSON 13

? and Editing of Programs	1
RND	2
INT	3
ABS	9
CLR	10
Trying Your Hand at Programming	13

LESSON 14

Arrays	14
DIM	15
Trying Your Hand at Programming	27

REVIEW 4

28

LESSON 15

DATA-READ	29
Trying Your Hand at Programming	39

LESSON 16

RESTORE	40
Trying Your Hand at Programming	44

LESSON 17

Two-dimensional Arrays	45
Trying Your Hand at Programming	50

LESSON 18

ON GOSUB	51
ON GOTO	52
Trying Your Hand at Programming	54

LESSON 19

POKE and Graphics	55
Trying Your Hand at Programming	72

LESSON 20

Animated Graphics	73
PEEK	77
Trying Your Hand at Programming	79

LESSON 8	
Keyboard Graphics	67
Animating Keyboard Graphics	73
Color Graphics	76
Trying Your Hand at Programming	80
LESSON 9	
GOSUB-RETURN	81
Trying Your Hand at Programming	84
LESSON 10	
String variables	85
INPUT	86
Trying Your Hand at Programming	94
LESSON 11	
IF-THEN	95
AND, OR	100
<, >, <=, >=, <>	103
Trying Your Hand at Programming	104
REVIEW	105
LESSON 12	
Overview of software available for use in the home	106
APPENDIX	
APPLICATION — Checkbook Balancer/Inventory	121
Answer to Reviews	132
Solutions to Trying Your Hand at Programming	133
Chart of Error Messages	149
Glossary	154
Index	159
COMPUTER COMMAND CARD	INSIDE BACK COVER

INTRODUCTION

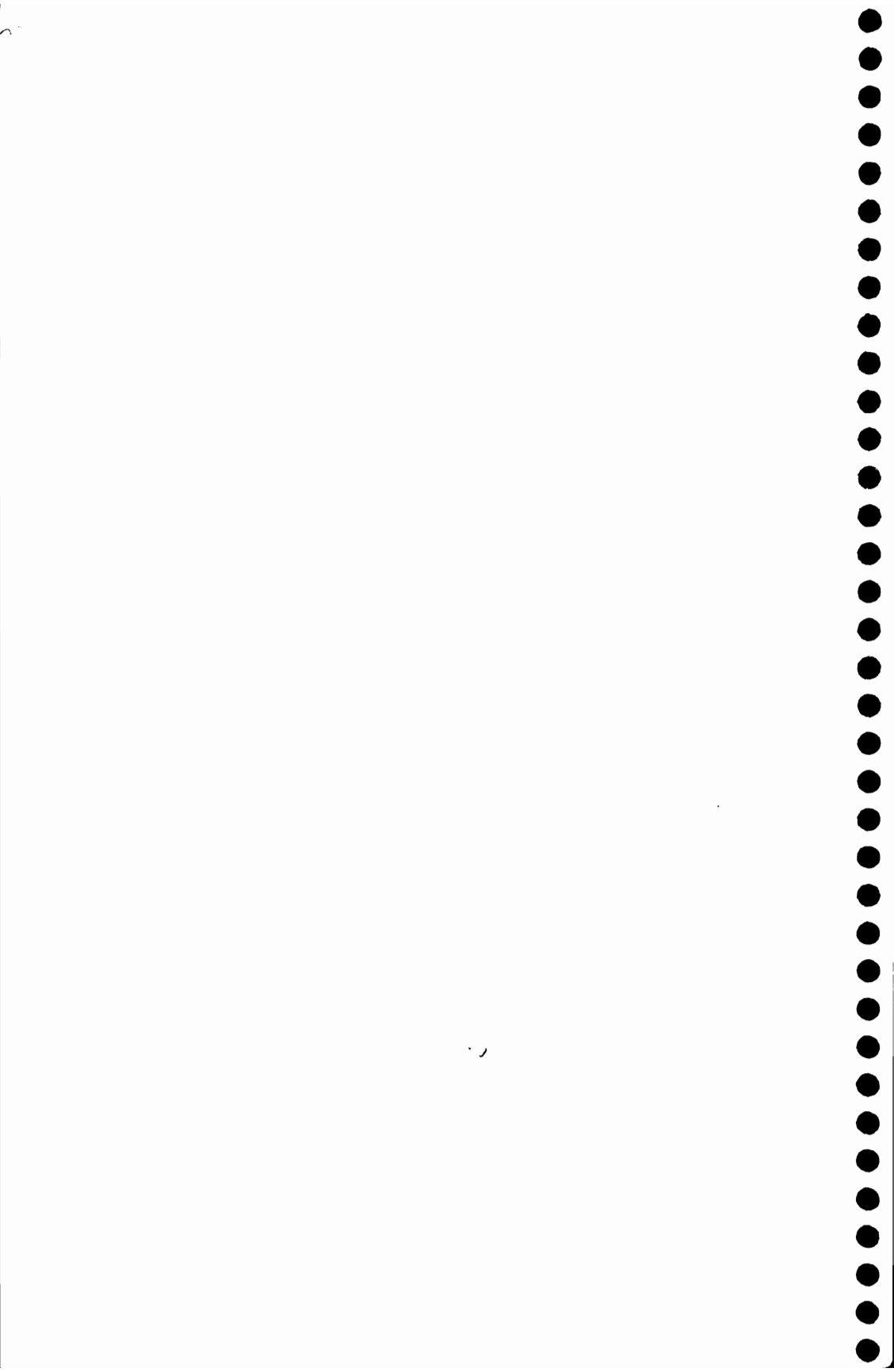
In level 1 of **TRAINING YOUR COMMODORE 64**, you were introduced to elementary **BASIC** concepts so that you could begin to find your way in and around your Commodore. But programming is like working with building blocks. One **BASIC** command fits with another. In level 1, you built the “basement” of your programs. With the new commands and concepts in level 2, you will start with that foundation and construct more interesting and powerful programs.

In each lesson, you will work through hands-on examples and exercises so that you can create your own programs with the skills you have mastered. The projects at the end of each lesson give you a chance to try your hand at programming and in case you need some help, you will find suggested program solutions in the Appendix.

After you have finished the lessons in this book, you may wish to remove the computer command card located on the inside back cover. Keep this card close to your computer as a quick reminder of the often used commands.

One final thought — be patient with yourself and your computer. It takes time to learn new concepts. With practice, you will be able to use these new commands in your own original programs.

TRAINING YOUR COMMODORE 64 gives you and each member of your family the programming ideas and practice that you need to master the commands that control your computer.



LESSON 13 RND, INT, ABS, CLR

INTRODUCTION

*In this lesson, you will be working with four new BASIC functions, **INT**, **RND**, **ABS** and **CLR**. These are used a lot in developing game programs as well as in financial and statistical applications.*

Before you begin this lesson, here are two programming time-savers.

#1 Have you found that you use the **PRINT** command a lot? You can type a **?** rather than the word **PRINT** when you are programming. Try this.

Type —

```
10 ? "[CLR]" ←  
20 ? "FANTASTIC!!"  
30 ? "ISN'T THIS GOING TO SAVE TIME?"
```

(Remember that [CLR] means to hold down the SHIFT key and press the CLR/HOME key.)

RUN

the program.

LIST

the program.

When the computer **LISTs** the program, it replaces your question mark with the word **PRINT**.

#2 Have you ever been in the middle of editing a line and decide that you liked it the way it was? The way to leave the current line unchanged is to hold down the SHIFT key and press RETURN. Try this.

LIST

line 20 from the last program.

Say you have decided to change the word FANTASTIC to the word WONDERFUL.

Move the cursor up to line 20 and over to the word FANTASTIC.

Now type in the word WONDERFUL but DO NOT PRESS RETURN.

Oops, you decide you like the word FANTASTIC better. Hold down the **SHIFT** key and press **RETURN**.

RUN

the program.

The computer ignored your editing and printed the word FANTASTIC. SHIFTed RETURN might be considered the "FORGET WHAT I JUST DID" key. It can get you out of a lot of messy situations.

Now, on with the lesson.

Enter this program into your computer —

```
10 PRINT RND(1)
20 GOTO 10
```

RUN

the program. Use the CTRL key to reduce the speed of the scrolling or press the STOP key so you can get a better look at the numbers.

RND(1) tells the computer to choose a random number between 0 and 1. See how all of the numbers are between 0 and 1?

RUN

your program several times and see that the numbers are always between 0 and 1.

RND is a function which tells the computer to pick a random number. PRINT RND(1) tells the computer to choose a random number between 0 and 1 and PRINT that number on the monitor screen.

Now you know how to get a random number between 0 and 1. How about getting random numbers between 0 and 5?

Enter this program into your computer —

```
10 PRINT RND(1)*6
20 GOTO 10
```

- Line 10 means —
- 1) pick a random number between 0 and 1
 - 2) multiply that number by 6
 - 3) **PRINT** the number

RUN

the program.

What numbers did you get? Are they between 0 and 6? Can you guess how you would get a random number between 0 and 9?

You could have written a line like this:

```
10 PRINT RND(1)*10
```

(This line will produce a random number between 0 and 9.)

Write a line to get a random number between 0 and 99.

You could have written a line like this:

```
10 PRINT RND(1)*100
```

Enter this program into your computer —

```
10 PRINT RND(1)*25  
20 GOTO 10
```

RUN

the program.

Take a look at your numbers. They are all between 0 and 25. This program will never produce the number 0 or the number 25. Notice that there is no pattern to the numbers being chosen by the computer? It's unpredictable — that's what **RND** (random) means.

RANDOM WHOLE NUMBERS

RND(1) can be useful. But first we need to change the random numbers into random whole numbers or **INTEGERS**. An integer is a number without any numbers after the decimal point.

2.22 is NOT an integer.

.111 is NOT an integer.

-6.2 is NOT an integer.

2 is an integer.

0 is an integer.

-100 is an integer.

Enter this program into your computer —

```
10 PRINT INT(2.22)
```

RUN

the program.

INT is a function which tells the computer to take away any numbers to the right of the decimal point. INT rounds off the number and the result will always be less than or equal to the number given.

Enter this program into your computer —

```
10 PRINT INT(7)
```

RUN

the program.

The computer should have printed a 7. Do you see why?

Enter this program into your computer —

```
10 PRINT INT(5)
20 PRINT INT(5.5)
30 PRINT INT(100.2)
```

What do you think the computer will **PRINT**? **RUN** the program and see if you were right.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 INPUT "WHAT IS YOUR NUMBER" ;N
30 PRINT "THE INT OF "N" IS "INT(N)
40 PRINT: GOTO 20
```

RUN

the program and try several numbers. Make sure you try some figures with numbers to the right of the decimal point (e.g. 100.567).

Now, use **INT** and **RND** together. Enter this program into your computer —

```
10 PRINT INT(RND(1)*10)
20 GOTO 10
```

RUN

the program and you'll get a random integer between 0 and 9. Using this statement, the computer will **PRINT** a 0 but will never **PRINT** the number 10.

Change the program so that it will give random whole numbers between 0 and 99.

You could have written a line like this:

```
10 PRINT INT(RND(1)*100)
```

What if you want random integers between 1 (not 0) and 10? All you have to do is write a line for random integers between 0 and 10 and then add 1 to them.

Enter this program into your computer —

```
10 PRINT INT(RND(1)*10)+1 ← (This line gives a random number
                             between 1 and 10.)
20 PRINT INT(RND(1)*100)+1 ← (This line gives a random number
                              between 1 and 100.)
```

RUN

the program.

Write a program line for each of the following:

1. Random integers between 0 and 49.
2. Random integers between 0 and 999.
3. Random numbers (not just integers) between 0 and 1.
4. Random integers between 1 and 99.

You could have written lines like this:

```
10 PRINT INT(RND(1)*50)
20 PRINT INT(RND(1)*1000)
30 PRINT (RND(1))
40 PRINT INT(RND(1)*99)+1
```

NUMBER GUESSING GAME

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 X=INT(RND(1)*25)
30 PRINT "I HAVE CHOSEN A NUMBER BETWEEN 0 AND 24."
40 INPUT "WHAT DO YOU THINK IT IS";N
50 IF N>X THEN PRINT "TOO BIG.": PRINT: GOTO 40
60 IF N<X THEN PRINT "TOO SMALL.": PRINT: GOTO 40
70 PRINT "GOOD GUESS! THE NUMBER WAS"X"."
80 PRINT
90 PRINT "DO YOU WANT TO 1) PLAY AGAIN OR 2) QUIT?"
100 GET A$
110 IF A$="1" THEN GOTO 10
120 IF A$="2" THEN PRINT "O.K. THANKS FOR PLAYING."
130 IF A$<>"1" OR A$<>"2" THEN GOTO 100
    
```

RUN

the program.

Did it take you many guesses?

Here is a program which is like the rolling of dice. Enter this program into your computer —

```

10 PRINT "[CLR]"
20 PRINT "I WILL ROLL ONE DIE.": GOSUB 300
30 PRINT "THEN I WILL ROLL A DIE FOR YOU.": GOSUB 300
40 PRINT "WHICHEVER NUMBER IS CLOSEST TO 6 WINS!!":
GOSUB 300
50 PRINT "HERE IS MY ROLL OF THE DIE.": GOSUB 300
60 X=INT(RND(1)*6)+1: REM NUMBER BETWEEN 1 AND 6
70 PRINT TAB(30) X: GOSUB 300
80 PRINT "HERE IS YOUR ROLL OF THE DIE.": GOSUB 300
90 Y=INT(RND(1)*6)+1: REM NUMBER BETWEEN 1 AND 6
100 PRINT TAB(30) Y: GOSUB 300
110 PRINT "YOU ROLLED A"Y
120 PRINT "I ROLLED A"X
130 PRINT
140 IF X<Y THEN PRINT "YOU WIN.": GOSUB 300
150 IF X>Y THEN PRINT "I WIN.": GOSUB 300
160 IF X=Y THEN PRINT "WE ROLLED THE SAME NUMBER!!!":
GOSUB 300
170 PRINT "DO YOU WANT TO PLAY AGAIN?"
    
```



```

180 INPUT "(1 FOR YES, 2 FOR NO)";A
190 IF A=1 THEN GOTO 10
200 IF A=2 THEN END
210 IF A<>1 OR A<>2 THEN GOTO 180
300 REM DELAY AND BLANK LINE
310 FOR D=1 TO 2000: NEXT D
320 PRINT
330 RETURN

```

RUN

the program.

FINDING EVEN AND/OR ODD INTEGERS

A number is even if when you divide it by 2 the answer is a whole number. A number is odd if when you divide it by 2 the answer has numbers after the decimal point. When you divide 10 by 2, the answer is 5. Five is a whole number so 10 is even. When you divide 11 by 2, the answer is 5.5. That answer has numbers after the decimal point, so 11 is odd.

When X is even, $X/2 = \text{INT}(X/2)$.

When X is odd, $X/2 <> \text{INT}(X/2)$.

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 INPUT "WHAT IS YOUR NUMBER";N
30 PRINT
40 IF N/2=INT(N/2) THEN GOTO 60
50 PRINT "YOUR NUMBER IS ODD.": PRINT: GOTO 20
60 PRINT "YOUR NUMBER IS EVEN.": PRINT: GOTO 20

```

RUN

the program six times and try each of these numbers:

333 102 11 -2 7777770 2221

IMPROVING THE ADDITION QUIZ

In level 1 of this series, there is an addition quiz using **IF-THEN** statements. You can improve the program using **RND** and **INT**. If you **SAVED** the program, **LOAD** it into your computer now.

If you have **LOAD**ed the program from level 1, delete lines 240 through 320 and enter lines 150 through 230 from the following program.

If you did not save the ADDITION QUIZ from level 1, enter this program into your computer (**SAVE** it for use throughout this lesson.) —

```

10 PRINT "[CLR]"
20 PRINT TAB(5) "THIS IS A SHORT ADDITION QUIZ."
30 PRINT: PRINT "YOU WILL SEE A MATH PROBLEM."
40 PRINT: PRINT "THERE WILL BE A ? AFTER THE PROBLEM."
50 PRINT: PRINT "TYPE YOUR ANSWER AND THEN"
60 PRINT "PRESS THE RETURN KEY."
70 PRINT
80 INPUT "ARE YOU READY TO BEGIN (YES OR NO)";R$
90 IF R$="YES" THEN GOTO 150
100 IF R$<>"NO" THEN GOTO 80
110 IF R$="NO" THEN PRINT "PRESS THE LETTER 'R' WHEN
YOU ARE READY!"
120 GET N$
130 IF N$="R" THEN GOTO 150
140 IF N$<>"R" THEN GOTO 120
150 FOR P=1 TO 5
160 A=INT(RND(1)*20+1): REM NUMBERS BETWEEN 1 AND 20
170 B=INT(RND(1)*10): REM NUMBERS BETWEEN 0 AND 9
180 PRINT: PRINT A "+" B "=": INPUT N
190 IF N=A+B THEN GOSUB 2000
200 IF N<>A+B THEN PRINT "THAT IS INCORRECT! THE
ANSWER IS "A+B"." : GOSUB 3000
210 NEXT P
220 GOSUB 4000
230 END
1000 REM SUBROUTINE FOR DELAY AND CLEAR SCREEN
1010 FOR T=1 TO 3000: NEXT T
1020 PRINT "[CLR]": RETURN
2000 REM CORRECT ANSWER SUBROUTINE
2010 PRINT
2020 FOR X=0 TO 20 STEP 2
2030 PRINT TAB(X) "THAT'S CORRECT!"
2040 NEXT X
2050 REM C STANDS FOR NUMBER CORRECT
2060 LET C=C+1
3000 REM Q STANDS FOR QUESTIONS SO FAR
3010 LET Q=Q+1
3020 PRINT: PRINT "YOU NOW HAVE" C "OUT OF " Q
"CORRECT."
3030 GOSUB 1000
3040 RETURN

```

```

4000 REM SUBROUTINE FOR TEST SCORE EVALUATION
4010 PRINT "THAT'S THE END OF THE QUIZ."
4020 PRINT
4030 IFC<=2 THEN PRINT"THAT'S NOT VERY GOOD. BETTER
PRACTICE MORE OFTEN.":RETURN
4040 IF C=3 THEN PRINT "YOU COULD SPEND SOME MORE
TIME WITH ME.":RETURN
4050 IF C=4 THEN PRINT "HOW DID YOU MISS ONE?":
RETURN
4060 PRINT "YOU REALLY KNOW THOSE ADDITION
FACTS!!!": RETURN

```

RUN

the program.

SAVE this program. You will revise it later in this lesson.

FINDING POSITIVE NUMBERS

ABS is a math function and stands for ABSOLUTE VALUE. **ABS** return the absolute value of a number, which is the number without its sign (+ or -).

If a number is positive, the ABS of that number is that number (e.g. ABS(3)=3).

If a number is negative, the ABS of that number is the positive form of that number (e.g. ABS(-3)=3).

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 PRINT "PLEASE INPUT ANY NUMBER. IT CAN BE"
30 PRINT "EITHER POSITIVE OR NEGATIVE."
40 INPUT N
50 IF ABS(N)=N THEN GOTO 80
60 PRINT "YOUR NUMBER,"N", IS NEGATIVE."
70 FOR D=1 TO 1500: NEXT D: GOTO 10
80 PRINT "YOUR NUMBER,"N", IS POSITIVE."
90 FOR D=1 TO 1500: NEXT D: GOTO 10

```

RUN

the program and try several different numbers, using both positive and negative numbers.

IMPROVING THE ADDITION QUIZ

LOAD

the ADDITION QUIZ program.

Delete lines 4030, 4040, 4050 and 4060.

Change line 4020 so it reads —

```
4020 PRINT: RETURN
```

Add these lines to the program —

```
150 PRINT: INPUT "HOW MANY PROBLEMS WOULD YOU  
LIKE";H  
152 IF H<>ABS(H) OR H=0 THEN PRINT: PRINT "TRY AGAIN.":  
GOTO 150  
155 FOR P=1 TO H
```

RUN

the program and when asked how many problems you want, **INPUT** a negative number.

SAVE the program again. There will be one more revision!

SETTING VARIABLES TO ZERO OR NULL

Enter this program into your computer —

```
10 PRINT "[CLR]"  
20 INPUT X  
30 PRINT X  
40 CLR  
50 PRINT X
```

RUN

the program. Type any number and then press **RETURN**.

The computer will **INPUT** your number as X. Then the computer will **PRINT** your number and will then **PRINT** a 0.

CLR stands for clear and sets the value of all variables back to zero. It also sets all string variables back to null.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 INPUT W$
30 PRINT W$
40 CLR: PRINT "THE COMPUTER WILL NOW CLEAR YOUR
INPUT."
50 PRINT W$
```

RUN

the program.

The computer will **INPUT** and **PRINT** whatever you enter as W\$. It will then **CLear** the variable and **PRINT** a blank line because W\$ is null.

Use **CLR** when you want to set all variables to 0 within a program.

IMPROVING THE ADDITION QUIZ

LOAD

the ADDITION QUIZ program.

You can give the user a chance to continue the program, and also give them a choice of beginning with a new score or continuing to add to the old score.

Add these lines to the program —

```
230 REM CHOICE OF CONTINUING PROGRAM
240 PRINT: PRINT: PRINT: PRINT
250 PRINT TAB(6) "PRESS THE LETTER 'M' IF YOU WOULD
LIKE MORE PROBLEMS."
260 PRINT: PRINT: PRINT
270 PRINT TAB(6) "PRESS THE LETTER 'S' IF YOU WOULD
LIKE TO STOP."
280 GET P$
290 IF P$="M" THEN GOSUB 500
300 IF P$="S" THEN END
310 IF P$<>"M" OR P$<>"S" THEN GOTO 280
500 REM ASK IF SCORE SHOULD BEGIN AT ZERO
510 PRINT "[CLR]"
520 PRINT: PRINT TAB(3) "PRESS THE LETTER 'N' IF YOU
WANT"
530 PRINT "TO BEGIN WITH A NEW SCORE."
```

```
540 PRINT: PRINT TAB(3) "PRESS THE LETTER 'C' IF YOU  
WANT"  
550 PRINT "TO CONTINUE WITH THE SAME SCORE."  
560 GET C$  
570 IF C$="N" THEN CLR: GOTO 150  
580 IF C$="C" THEN GOTO 150  
590 IF C$<>"N" OR C$<>"C" THEN GOTO 560  
600 END
```

RUN

the program.

TRYING YOUR HAND AT PROGRAMMING

13-1 Change the addition quiz so it will choose numbers between 1 and 100, inclusive.

Change the addition quiz so that it is a multiplication quiz using random numbers between 1 and 12.

13-2 Write a program that will ask the user to **INPUT** 10 numbers. They can be positives and negatives. Then have the computer give you the sum of the positive numbers and the sum of the negative numbers. Improve the program by adding 2 **GOSUB** statements which will keep track of the number of positive numbers and the number of negative numbers entered.

Hint: Use a **FOR-NEXT** loop to allow 10 **INPUT**s and include **IF-THEN** statements with **ABS** to check to see whether the numbers entered are positive or negative.

13-3 Write your own version of a game of "21" or a number guessing game.

LESSON 14 ARRAYS

INTRODUCTION

Arrays are very useful in programming because with an array you can group related ideas together in one variable. For example, say you are designing a program to keep an inventory of what's in the freezer. You might have four meat products and ninety-nine different flavors of ice cream. Instead of using 103 variables for all the items, you can use one array for the meat group and one for the ice cream.

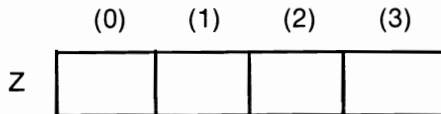
This lesson will show you how to store information in an array. After you learn how to store information in an array, you will use an array in your programming.

First, what is an array?

Think of an array as a row of boxes.



The array must have a name. This array will be called Z. Each box gets a number.



You decide the number of boxes you will need to store information. Then you tell the computer how many memory spaces it will need for your array.

Enter this program line into your computer —

```
10 DIM Z(3)
```

RUN
the program.

You cannot see anything happening but the computer has set aside four memory spaces for the array named Z. Memory spaces begin with the number 0 and the **DIM** statement tells the computer what the highest box number will be. Therefore, **DIM Z(3)** reserves spaces 0, 1, 2 and 3 for an array called Z.

DIM is a command that tells the computer how big you want your array to be (how many boxes or memory spaces). DIM stands for DIMension which means size.

Now you can put information into each of the boxes of the array.

Add these program lines —

```
20 Z(0)=10
30 Z(1)=7
40 Z(2)=4
50 Z(3)=2
```

RUN
the program.

Still not much happens, right?

Wrong! When you ran the program, the computer set up an array with the information in each "box" as shown. This program puts the information into the boxes. So now the array called Z looks like this:

	(0)	(1)	(2)	(3)
Z	10	7	4	2

Now add these lines to the program —

```
60 PRINT "[CLR]"
70 PRINT "HERE IS THE INFORMATION STORED IN EACH"
80 PRINT "BOX OF YOUR ARRAY."
90 PRINT
100 PRINT "BOX NUMBER, INFORMATION STORED THERE"
110 PRINT "Z(0)", Z(0)
120 PRINT "Z(1)", Z(1)
130 PRINT "Z(2)", Z(2)
140 PRINT "Z(3)", Z(3)
```

RUN
the program.

Array is not a command in BASIC. To make an array of stored information you need:

A DIM statement

A line number

The name of the array

The box number in parentheses

An equals sign

The information you want stored in that box.

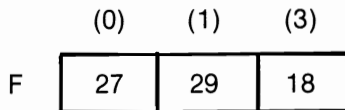
Here are a few examples of program lines using arrays:

```

10 Z(0)=3 ←———— (This line puts the number 3 into the Z(0) box.)
20 Z(1)=25 ←———— (This line puts the number 25 into the Z(1) box.)
30 PRINT "Z(0)" ←———— (This line PRINTs Z(0).)
40 PRINT Z(0) ←———— (This line PRINTs the information stored in
                        box Z(0).)
    
```

An array is like a row of boxes. Each box has a name (e.g. Z(3)). You can store information in each of the boxes.

Write a program to make an array called F. Your array is to have 3 boxes with the following information:



Then have the computer **PRINT** "HERE IS THE INFORMATION STORED IN EACH BOX OF YOUR ARRAY" just like in the previous program with the array Z.

You could have written a program similar to this:

```

10 PRINT "[CLR]"
20 DIM F(2) ←———— (In this program, this line is not necessary
30 F(0)=27             because the computer automatically DIMs
40 F(1)=29             an array to 11 boxes and you are only using 3.)
50 F(2)=18
    
```

```

60 PRINT "HERE IS THE INFORMATION STORED IN EACH"
70 PRINT "BOX OF YOUR ARRAY."
80 PRINT
90 PRINT "BOX NUMBER, INFORMATION STORED THERE"
100 PRINT "F(0)", F(0)
110 PRINT "F(1)", F(1)
120 PRINT "F(2)", F(2)

```

NOTE: The computer will **DIM** an array to 11 boxes without a **DIM** statement. A **DIM** statement is only needed if you want to store information in more than 11 memory spaces. However, it is a good idea to **DIM** every array. You will get a BAD SUBSCRIPT ERROR message if you do not **DIM** enough memory spaces for your array.

The information stored in the rows of an array does not have to be a number.

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 DIM A$(2)
30 A$(0)="ARRAYS"
40 A$(1)="ARE"
50 A$(2)="EASY!"

```

RUN
the program.

All you will see is your screen being erased.

But in your computer's memory, you have set up an array called A\$ that looks like this:

	(0)	(1)	(2)
A\$	ARRAYS	ARE	EASY!

Add these program lines —

```

60 PRINT A$(0)
70 PRINT A$(1)
80 PRINT A$(2)

```

RUN
the program.

An array is a set of memory spaces where you can store information. You can store numbers (e.g. A(1)=37) or you can store words (e.g. A\$(1)="HI").

Now, you write a program to set up an array that looks like this:

	(0)	(1)	(2)
P\$	ARE	ROSES	RED.

Then have the computer **PRINT** the information like this:

```
ROSES
ARE
RED.
```

You could have written a program similar to this:

```
10 PRINT "[CLR]"
20 DIM P$(2)
30 P$(0)="ARE"
40 P$(1)="ROSES"
50 P$(2)="RED."
60 PRINT P$(1) : PRINT P$(0) : PRINT P$(2)
```

Now, change your program so the computer will **PRINT** the information on one line like this:

```
ROSES ARE RED.
```

You could have changed line 60 so it reads:

```
60 PRINT P$(1) " "; P$(0) " "; P$(2)
```

You can use **INPUT** and **FOR-NEXT** to speed things up.

What if you wanted to store the names of the "TOP 5 RADIO STATIONS FOR THIS WEEK" in an array?

The array is going to look like this:

	(1)	(2)	(3)	(4)	(5)
RS\$	WGN	WCLR	KYW	WLS	WWBC

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 DIM RS$(5)
30 FOR N=1 TO 5
40 INPUT RS$(N) ← (By using FOR-NEXT, this line will become
50 NEXT N
                    INPUT RS$(1)
                    INPUT RS$(2)
                    INPUT RS$(3)
                    INPUT RS$(4)
                    INPUT RS$(5)

```

To get the computer to display the information you have stored in the array RS\$, add these lines to the program —

```

60 PRINT "HERE ARE YOUR TOP 5 RADIO STATIONS THIS
WEEK:"
70 PRINT
80 FOR N=1 TO 5
90 PRINT "#" N; RS$(N)
100 NEXT N

```

RUN

the program. When you see the question mark, **INPUT** the names from the boxes above, one at a time and in the proper order. When you finish, you will have created an array called RS\$.

Look at the **DIM** statement in line 20. Because the **FOR-NEXT** loops in lines 30 and 80 go from number 1 (rather than 0) to number 5, the array needs to be able to use those same numbers. Therefore, the highest box number in the array needs to be a 5.

RUN

the program again and **INPUT** the names of five of the top radio stations in your area.

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 DIM F$(4)
30 PRINT "PLEASE INPUT YOUR FOUR FAVORITE FOODS."
40 FOR N=1 TO 4
50 INPUT F$(N)
60 NEXT N
70 PRINT "HERE ARE YOUR 4 FAVORITE FOODS:"
80 PRINT

```

```

90 FOR N=1 TO 4
100 PRINT "#"N;F$(N)
110 NEXT N
    
```

RUN

the program and **INPUT** four of your favorite foods.

You can change the information stored in the boxes of an array.

Add these lines to the program —

```

120 PRINT
130 PRINT "DO YOU WANT TO CHANGE ONE"
140 INPUT "(YES OR NO)";A$
150 IF A$="YES" THEN GOTO 180
160 IF A$="NO" THEN END
170 IF A$<>"YES" OR A$<>"NO" THEN GOTO 140
180 INPUT "WHICH NUMBER DO YOU WANT TO CHANGE";C
190 PRINT "WHAT WOULD YOU LIKE TO CHANGE IT TO"
200 INPUT C$
210 F$(C)=C$ ← (This line puts the new information into the
220 PRINT: GOTO 70 box of the array you wanted to change.)
    
```

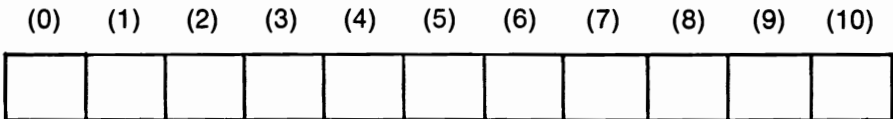
RUN

the program again. **INPUT** four of your favorite foods and then change one or several of them.

So far you have been thinking of an array as a row of boxes that you can store information in.

You can also think of an array as a row of parking spaces. At each space you can "park" information.

Each parking space gets a number. You use a **DIM** statement to tell the computer how many spaces you will need. For this example, you would **DIM** 11 memory spaces.



The parking lot (the array) must have a name. Use a variable to store numbers in the spaces. This array is called **L**.

	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
L	27	62	11	12	13	9	8	260	101	40	376.1

To store any character from the keyboard (letters, symbols, or numbers) in the spaces, use a string variable. This array is called **L string**.

	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
L\$	WENDY	CHRIS	JO	TRISH	BUZZ	MUFFY	SHELLY	DAVID	JEFF	KELLY	BEN

To "park" the "cars" in their spaces (to put information into the boxes of the array) you use an equation. Here are two examples:

10 L(4)=13 (This line puts 13 into L(4).)
20 L\$(5)="MUFFY" (This line puts MUFFY into L\$(5).)

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 DIM L$(10)
30 PRINT "PARKING LOT PROBLEM"
40 PRINT: PRINT
50 INPUT "WHAT CAR" ;C$
60 INPUT "WHAT SPACE DO YOU WANT IT IN";S
70 L$(S)=C$ ←(This line "parks" the name of the car in its space in the array.)
80 PRINT
90 PRINT "WOULD YOU LIKE TO 1)PARK ANOTHER CAR,"
100 PRINT "OR 2) SEE THE LOT?"
110 GET Q
120 IF Q=1 THEN GOTO 40
130 IF Q=2 THEN GOTO 150
140 IF Q<>1 OR Q<>2 THEN GOTO 110
150 PRINT: PRINT "SPACE", "CAR"
160 FOR S=1 TO 10
170 PRINT S, L$(S) ←(This line PRINTs the number of each
180 NEXT S "parking space" in the array and the
190 GOTO 40 information stored there.)

```

Here is a list of vehicles to park using this program:

TEN-SPEED	STATION WAGON	BUG
JAG	WHITE HOG	SPORTS CAR
MOPED	VISITOR	CESSNA
VISITOR		

Park the cars so that they meet the following qualifications:

1. The WHITE HOG must be on one end — it's big!!!
2. Park the MOPED next to the TEN-SPEED so it will have some company.
3. Park the CESSNA next to either the MOPED or the TEN-SPEED because of the wingspan.
4. Leave at least 3 spaces between the 2 VISITOR spots.
5. The SPORTS CAR and the STATION WAGON always park next to each other.
6. Park the JAG within 3 spaces of the WHITE HOG.

RUN

the program and park the cars so they meet the qualifications listed above. To change a space you have already filled, just type the space number and the new car you want there.

There are several ways you can park the cars. Here is one way:

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
L\$	WHITE HOG	VISITOR	JAG	SPORTS CAR	STATION WAGON	TEN-SPEED	MOPED	CESSNA	VISITOR	BUG

A REVIEW COMPARING VARIABLES AND ARRAYS

Say you wanted a program that would record your scores on 10 tests and then give you the average. You could use 10 variables, or you could use an array.

Enter this program using variables into your computer —

```
10 PRINT "[CLR]"
20 PRINT "INPUT YOUR SCORES"
30 INPUT A
40 INPUT B
50 INPUT C
60 INPUT D
70 INPUT E
80 INPUT F
90 INPUT G
100 INPUT H
110 INPUT J
120 INPUT K
130 PRINT "YOUR AVERAGE IS" (A+B+C+D+E+F+G+H+I+J+K)/10
```

RUN

the program. Make up 10 test scores to **INPUT**.

Now, change your program using arrays.

Add this line to the program —

```
15 DIM T(10)
```

Change these lines so they read —

```
30 FOR N=1 TO 10
40 INPUT T(N)
50 LET S=S+T(N)
60 NEXT N
70 PRINT "YOUR AVERAGE IS" S/10
```

Delete lines 80 through 130.

RUN

the program and use the same test scores or make up some new ones.

Compare the two programs. See how using an array can make your programs shorter?

Look at the following program:

```
10 PRINT "[CLR]"
20 PRINT "SUPPOSE YOU JUST WON A MILLION DOLLARS."
30 PRINT "WHAT WOULD YOU BUY?"
40 INPUT P1$
50 INPUT P2$
60 INPUT P3$
70 INPUT P4$
80 INPUT P5$
90 INPUT P6$
100 INPUT P7$
110 INPUT P8$
120 INPUT P9$
130 INPUT P10$
140 PRINT "HERE'S WHAT YOU BOUGHT:"
150 PRINT P1$, P2$, P3$, P4$, P5$, P6$, P7$, P8$, P9$, P10$
```

Now, compare it with this program:

```
10 PRINT "[CLR]"
20 PRINT "SUPPOSE YOU JUST WON A MILLION DOLLARS."
30 PRINT "WHAT WOULD YOU BUY?"
40 DIM P$(10)
50 FOR N=1 TO 10
60 INPUT P$(N)
70 NEXT N
80 PRINT "HERE'S WHAT YOU BOUGHT:"
90 FOR N=1 TO 10
100 PRINT P$(N),
110 NEXT N
```

Enter both programs if you wish, but even without running them, you can see that using arrays rather than many variables makes it a lot easier to keep track of what the program is doing and saves a lot of time.

THE MATCH GAME

Arrays are a useful way to store information. Use an array for a computerized version of a game show.

Enter this program into your computer (Look at lines 120, 170 and 190. Type them as they appear here with no spaces between commands.) —

```

10 PRINT "[CLR]"
20 DIM M$(10) ← (This line reserves memory spaces
30 FOR S=1 TO 10 ← for the array called M$.)
40 INPUT M$(S) ← (These lines let you INPUT
50 NEXT S ← the prizes into the array.)
60 PRINT "[CLR]"
70 FOR N=1 TO 10
80 PRINT N
90 NEXT N
110 INPUT "WHAT IS YOUR FIRST CHOICE";A
120 IFM$(A)=" THENPRINT"THAT'S ALREADY GONE!":FOR
T=1TO1500:NEXTT:GOTO100
130 PRINT TAB(12) M$(A)
140 PRINT
150 INPUT "WHAT IS YOUR SECOND CHOICE";B
160 IF A=B THEN GOTO 140
170 IFM$(B)=" THENPRINT"THAT'S ALREADY GONE!":FOR
T=1TO1500:NEXTT:GOTO140
180 PRINT TAB(12) M$(B)
190 IFM$(A)=M$(B)THENPRINT"THAT'S A MATCH!":M$(A)=
"":M$(B)=" ":GOTO 210
200 PRINT "NOT A MATCH!"
210 FOR D=1 TO 1500: NEXT D: GOTO 60

```

RUN

the program. The computer will ask you to **INPUT** information into the 10 boxes of the array M\$. **INPUT** five prizes TWICE at any spots in the array you want. After you have each prize in two spots in the array, ask a friend to play the Match Game. See if he/she can match up all 5 prizes.

LIST

line 190.

In line 190, why do you program the computer to set M\$(A) and M\$(B) equal to " "?

After a match has been made, the " " (empty quotation marks) will tell lines 120 and 170 if a number has already been matched.

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 PRINT "THIS PROGRAM HELPS YOU MAKE YOUR"
30 PRINT "SHOPPING LIST."

```

```
40 PRINT: PRINT "PLEASE INPUT WHAT YOU NEED TO BUY
TODAY. (UP TO 15 ITEMS)"
50 DIM G$(14)
60 FOR N=0 TO 14
70 INPUT G$(N)
80 NEXT N
90 PRINT "HERE'S YOUR LIST": PRINT
100 FOR L=0 TO 14
110 PRINT G$(L)
120 NEXT L
130 END
```

RUN

the program and make your shopping list!

Change these lines so they read —

```
40 PRINT: PRINT "PLEASE INPUT WHAT YOU NEED TO BUY
TODAY."
50 DIM G$(X)
60 FOR N=0 TO X
100 FOR L=0 TO X
```

Now, add these lines to the program to **DIM**ension your array to the exact size you need —

```
35 PRINT: PRINT "HOW MANY DIFFERENT ITEMS ARE YOU
GOING TO BUY"
37 INPUT A
39 X=A-1
```

RUN

the program

A NOTE ON READING ARRAYS

When you see:

Say to yourself:

A(4)=96
A\$(4)="SUGARLIPS"

A sub four equals ninety-six
A string sub four equals sugarlips

The name for the boxes of an array is elements. Instead of saying an array may have up to 11 boxes, you can say an array may have up to 11 elements.

The number of each box (element) is called its subscript.

TRYING YOUR HAND AT PROGRAMMING

14-1 Write a program that will ask someone to **INPUT** 8 favorite television shows. Have the computer store the answers in an array called TV\$ and then **PRINT** the 8 names of the television shows. Then allow the person to change the names of his/her favorite shows.

14-2 Write a program which uses 2 arrays — one to store a person's name and the second to store their phone number. Then display that information on the monitor.

14-3 Write a program that asks the user to **INPUT** the number of guests they are going to invite to a party. Have the program **DIM** the array with enough boxes to store the names. Then have the user **INPUT** each name into a space in the array. Have the computer **PRINT** their guest list on the bottom half of the screen.

14-4 Say it's summertime and you're in business for yourself — (e.g. cutting grass, babysitting, programming computers). Write a program that will allow you to **INPUT** your earnings for each week. Have the computer store the information for each week in an array. Then have the computer display an account of your earnings (the information from the array), give the average weekly earnings and the total earnings.

REVIEW 4

Choose a word which will match each definition.

ABS

array

CLR

DIM

element

INT

PRINT INT(RND)(1)*24)+1

PRINT RND(1)

RND

subscript

1. _____ is a command which, when used with a number, will have the computer choose a number by chance.
2. _____ is a command used to round a number to the closest whole number and take away any numbers to the right of the decimal point.
3. _____ would print random numbers between 0 and 1.
4. _____ would print random integers between 1 and 24.
5. _____ is a command which will always give the positive form of a number.
6. _____ sets the value of all variables to zero and/or all string variables to null.
7. An _____ is a "row of boxes" in which the computer stores information.
8. _____ is the term for the "boxes" in an array.
9. _____ is the name for the number of the element in an array, as in A(3).
10. _____ tells the computer how many memory spaces to set aside for an array.

LESSON 15 DATA-READ

INTRODUCTION

*With the commands, **DATA** and **READ**, you have a very useful way of storing information within a program. Up to now, you could **INPUT** information into a program but as soon as you turned off the computer the data was gone. With **DATA** you can now keep a permanent record of the data you want read during the program.*

Using arrays is one way to store information. You have used two different methods to assign strings and numeric data to variables. You have used **INPUT** which allowed you to write a program without knowing what the data in the variable would be:

```
10 INPUT D
20 PRINT D
```

You have also assigned the value in a program line:

```
10 D=5
20 PRINT D
30 A$="HELLO"
40 PRINT A$
```

DATA and **READ** statements are an important way to get information into a program.

DATA and READ allow you to store information in a program. The information will be assigned to a variable while the program is actually running.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 READ A
30 PRINT A
100 DATA 10,15,20,25,30
```

(The commas separate one piece of data from the next.) ←

To read all of the data, the number of items in the DATA statement should equal the number of variables in the READ statement.

You can **READ** the data in one order and **PRINT** in another. Change the program so it will **PRINT** the numbers in this order — 30, 25, 20, 15, 10.

You could have written lines like this:

```
30 PRINT E
32 PRINT D
34 PRINT C
36 PRINT B
38 PRINT A
```

READ and **DATA** can also be used with string variables.

Enter this program into your computer —

```
10 DATA USING, DATA, AND READ, IS EASY
20 READ A$, B$, C$, D$
30 PRINT A$
40 PRINT B$
50 PRINT C$
60 PRINT D$
```

(Quotation marks are not needed in a **DATA** statement.)

RUN

the program.

Change the program so that E\$ will equal "AND FUN" and the computer will **PRINT** E\$.

You could have written lines like this:

```
15 DATA AND FUN
25 READ E$
70 PRINT E$
```

When using DATA and READ, put the values on the DATA line with commas between values.

**(e.g. 10 DATA 34.6,11.22,1300,666
10 DATA I'M,BAD!)**

Put the variables on the READ line with commas between them.

**(e.g. 20 READ A,B,C,D
20 READ D\$, B\$)**

Notice that in the first program the DATA statement was at the end of the program. In this program, the DATA statements are at the beginning.

If you put all of your DATA statements at the very beginning or at the very end of your program, the values will be easier to change if needed.

A TELEPHONE DIRECTORY USING DATA-READ

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 FOR X=1 TO 10
30 READ MT$
40 PRINT MT$
50 NEXT X
60 PRINT: PRINT
70 PRINT TAB(5) "LISTED ABOVE ARE MOVIE THEATRES IN
THIS AREA."
80 GOTO 80 ← (This line creates an endless loop
100 DATA PLAZA which keeps READY and the blinking
110 DATA LIDO cursor off the screen.)
120 DATA ROXY
130 DATA SANDS
140 DATA BIJOU
150 DATA RITZ
160 DATA CRITERION
170 DATA MANN
180 DATA EDWARDS,MISSION
```

RUN

the program.

Remember that you need to press the RUN/STOP key to BREAK an endless loop.

LIST

the program.

Notice the **FOR-NEXT** loop in lines 20 through 50. It executes the **READ** and **PRINT** statements 10 times so that all of the entries in the **DATA** statement are used.

When the computer sees a READ statement, it uses the first value in the first DATA statement.

For the second value from a READ statement, the computer goes to the next DATA value and so on.

The computer always READs the values from the first DATA statement in a program before going on to further DATA statements.

USING A FLAG WITH DATA-READ

So far, you have counted carefully and have used **READ** as many times as you have had entries in the **DATA** statements. What happens if there is not enough **DATA** for the computer to **READ**?

Change line 20 in the MOVIE THEATRE program so it reads —

```
20 FOR X=1 TO 30
```

RUN

the program.

The computer **PRINTs** ?OUT OF DATA ERROR IN 30 and then stops the program. An **OUT OF DATA ERROR** means that all of the **DATA** statements in the program have already been read. There are at least two ways to fix the program.

Counting the number of **DATA** entries is one way to make sure that you have the same number of values in the **DATA** statements and the **READ** statements. If you have a lot of data entries, this can be difficult.

Another way is to use a FLAG.

Add this line to the program —

200 DATA XX

The two XX's at the end of the **DATA** statement are a FLAG that tell the computer when the program has reached the last **DATA** item. You then need an **IF-THEN** statement to tell the computer what to do when it sees the FLAG.

Add this line to the program —

35 IF MT\$="XX" THEN GOTO 60 ← (This line tells the computer that when it sees the FLAG it is to stop **READING DATA** and **GOTO** line 60.)

RUN

the program.

Use a FLAG (a number(s) or letter(s) at the end of the last DATA entry) and an IF-THEN statement so the computer will not give an ?OUT OF DATA ERROR.

Add telephone numbers to the MOVIE THEATRE program by changing the **DATA** statements so they read —

100 DATA PLAZA, 559-1110
110 DATA LIDO, 857-1234
120 DATA ROXY, 499-4343
130 DATA SANDS, 555-1222
140 DATA BIJOU, 987-4444
150 DATA RITZ, 244-5666
160 DATA CRITERION, 857-1000
170 DATA MANN, 244-4534
180 DATA EDWARDS, 555-9999,MISSION,555-8577

Then add a **READ** variable for the telephone numbers —

38 READ P\$

And change line 40 so it will **PRINT** the names of the movie theatres as well as their telephone numbers —

40 PRINT MT\$,P\$

RUN

the program.

Add a field in each of the **DATA** statements to contain the address of each theatre. **READ** this and **PRINT** it out with the name and the phone number. In a **DATA** statement, "field" refers to the information between a set of commas which is read as one entry (e.g. line 100 contains 2 fields, the theatre name and the phone number). Depending upon how long the theatres' addresses are, the **DATA** in lines 180 and 190 may have to be on two separate lines. If you have trouble, look in the Appendix for a possible solution.

The length of a DATA statement is limited to 80 characters or two lines on the monitor screen.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 READ G
30 IF G=-1 THEN GOTO 70
39 REM TOTAL GRADES
40 T=T+G
49 REM TOTAL STUDENTS
50 S=S+1
60 GOTO 20
69 REM DIVIDE TOTAL GRADES BY TOTAL STUDENTS
70 AV=T/S
80 PRINT "THERE ARE" S "STUDENTS IN THE CLASS."
90 PRINT: PRINT "THE AVERAGE FOR THE TEST IS" AV
1000 DATA 100,90,95,80,75,70,100,95,85,85
1010 DATA 95,70,80,90,90
2000 DATA -1
```

RUN

the program.

Notice the FLAG in line 2000. As with any FLAG, be careful that it cannot be mistaken for a number you might have in a **DATA** statement. Because no one would receive a minus grade, you can use -1 as a FLAG.

Here is a program which keeps a record of how many miles you run in a week.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 INPUT "RUNNING MILEAGE FOR WHICH WEEK";W$
30 FOR X=1 TO 40
40 READ M : REM MILES
50 IF M=2000 THEN GOTO 110
60 READ T
70 D=D+M: REM TOTAL MILES RUN
80 H=H+T: REM TOTAL MINUTES RUN
90 AV=D/H*100
100 NEXT X
110 PRINT
120 PRINT "YOU RAN" D "MILES FOR THE WEEK OF "W$
130 PRINT
140 PRINT "YOUR AVERAGE TIME WAS"AV"MINUTES PER
MILE."
1000 DATA 2,120,2,130,2,115,3,200,2,130
1010 DATA 2000
```

RUN

the program.

Change the program to keep a record of a sport you play.

Remember the message center from the **IF-THEN** lesson in level 1 of this series? It can be rewritten so that it is easier to record messages.

First, set up three arrays. One for the person receiving the message, one for the message and one for the person sending the message.

```
5 T=30
10 DIM R$(T) : DIM M$(T) : DIM S$(T)
```

Next, **READ** the data —

```
20 FOR X=1 TO T
30 READ R$(X)
40 IF R$(X)="XX" THEN GOTO 80
50 READ M$(X)
60 READ S$(X)
70 NEXT X
```

The following lines clear the screen, give directions to the user and **PRINT** the **DATA**.

```

80 PRINT "[CLR]"
90 PRINT "PRESS THE LETTER 'M' IF YOU WANT TO SEE
YOUR MESSAGES."
100 GET A$
110 IF A$="M" THEN GOTO 130
120 IF A$<>"M" THEN GOTO 100
130 INPUT "WHAT IS YOUR NAME" ;B$
140 FOR X=1 TO T
150 IF B$=R$(X) THEN PRINT: PRINT M$(X) : PRINT TAB(10)
"MESSAGE FROM "S$(X)
160 NEXT X
170 PRINT: PRINT "THOSE ARE ALL THE MESSAGES FOR
YOU."
180 PRINT: PRINT "PRESS THE LETTER 'D' WHEN YOU ARE
DONE."
190 GET D$
200 IF D$="D" THEN GOTO 80
210 IF D$<>"D" THEN GOTO 190
1000 DATA JIM, PLEASE TAKE OUT THE GARBAGE,MOM
1010 DATA CARROLL, DAD WILL PICK YOU UP BY 6,JIM
1020 DATA JIM, I WILL BE HOME BY 5,MOM
1030 DATA XX

```

RUN

the program. When asked for your name, try these names:

JIM CARROLL YOUR OWN NAME

Because the dimension of the arrays is 30, there can be 30 messages in this program. To allow more messages, change the variable (T) in line 5.

COMPUTER TUTOR

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 DIM FW$(20) : DIM EW$(20) : DIM N$(20)
30 PRINT "IN THIS PROGRAM, THE COMPUTER WILL PICK"
40 PRINT "A RANDOM FRENCH WORD FROM ITS MEMORY."
50 PRINT
60 PRINT "YOU SHOULD THEN THINK OF THE ENGLISH"

```

```

70 PRINT "TRANSLATION FOR THAT FRENCH WORD."
80 FOR N=1 TO 20
90 READ FW$(N) : READ EW$(N)
100 NEXT N
110 PRINT
120 PRINT "PRESS THE LETTER 'R' WHEN YOU ARE READY."
130 GET R$
140 IF R$="R" THEN GOTO 160
150 IF R$<>"R" THEN GOTO 130
160 PRINT "[CLR]"
170 C=INT(RND(1)*20)+1
179 REM CHECK SUBSCRIPT SO THAT THE SAME WORD WILL
NOT APPEAR RIGHT AWAY
180 IF FW$(C)=N$(C) THEN GOTO 170
189 REM PRINT FRENCH WORD
190 PRINT FW$(C)"=" ;
200 FOR D=1 TO 2000 : NEXT D
209 REM PRINT ENGLISH WORD
210 PRINT EW$(C)
220 FOR D=1 TO 1000 : NEXT D
229 REM CHANGE SUBSCRIPT NUMBER FOR USE IN LINE 180
230 N$(C)=FW$(C)
240 GOTO 110
1000 DATA AIRPLANE,AVION,CAT,CHAT,BANANA,BANANE,
DOOR,PORTE,EAR,OREILLE
1010 DATA FISH,POISSON,GUN,FUSIL,HORSE,CHEVAL,JULY,
JUILLET,KNIFE,COUTEAU
1020 DATA LEAF,FEUILLE,MAP,CARTE,OCTOBER,OCTOBRE,
PENCIL,CRAYON,RING,BAQUE
1030 DATA RUG,TAPIS,SEA,MER,SISTER,SOEUR,TOY,JOUET,
WINE,VIN
    
```

RUN

the program.

There are lots of ways to improve this program. Change the delay if you need more time before the answer appears. Put in a vocabulary list you are studying. (Here's a chance to teach yourself Chinese!)

TRYING YOUR HAND AT PROGRAMMING

15-1 Write a program for your own telephone directory. Have the program list a person's name, address and phone list. (Use the MOVIE DIRECTORY program in this lesson as a model.)

15-2 Program your computer to store an address file using **DATA** statements. Use an **INPUT** statement to allow the user to **PRINT** a particular name.

If you own a printer, here is a printer subroutine so the address could be recorded on an address label or an envelope:

500 REM OPEN PRINTER AND PRINT ROUTINE

```

510 OPEN99,4:CMD4 ← (This line opens the printer.)
520 PRINT#99 ← (This line prints a blank line.)
530 PRINT#99,NAME$
540 PRINT#99,ADDRESS$
550 PRINT#99,CITY$ ", " STATES$ " " ZIP$
560 PRINT#99:CLOSE99 ← (This line closes the printer.)

```

15-3 Suppose you walk into the corner Used Car Lot and instead of a salesman, you are greeted by a computer! Write a program that will use **DATA** statements with the following information:

```

SOUPED-UP '74 CAMARO
BEAT-UP '71 MAVERICK
HOT LITTLE DUNE BUGGY
CLASSY '78 CADDY
REBUILT BUG
FAST 280Z
JACKED-UP PICK UP
VERY EXPENSIVE MERCEDES

```

Have the program put the information into an array by using **FOR-NEXT** and **READ**. Display the cars on the lot, then ask if the customer would like to test drive one of them. Use **INPUT** statements to ask which car and ask if the customer is interested in buying. Then offer a "reasonable" price for their choice.

LESSON 16 RESTORE

INTRODUCTION

Sometimes it is necessary to read **DATA** more than once without running the program again. **RESTORE** lets you use the same **DATA** line again. The **RESTORE** command works with the **DATA** and **READ** commands. **RESTORE** directs the **READ** pointer to the first **DATA** statement in the program. (Note: the command **RESTORE** is different from the **RESTORE** key.)

The **RESTORE** command is used with **DATA** and **READ**.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 PRINT "THIS PROGRAM WILL PRINT WORDS QUICKLY."
30 FOR D=1 TO 2000: NEXT D
40 FOR C=1 TO 7
50 PRINT "[CLR]"
60 READ N$
70 PRINT TAB(215)N$
80 FOR D=1 TO 150: NEXT D
90 NEXT C
100 PRINT "[CLR]"
110 PRINT "DO YOU WANT ME TO PRINT AGAIN (Y OR N)?"
120 GET A$
130 IF A$="Y" THEN GOTO 10
140 IF A$="N" THEN END
150 IF A$<>"Y" OR A$<>"N" THEN GOTO 120
1000 DATA THIS,IS,A,PROGRAM,THAT,NEEDS,RESTORE.
```

RUN

the program.

What happens if you press the letter "Y" to see the words again?

In order to have the computer start to **READ** the **DATA** statements at the beginning again, you need to give the computer a new command.

Change line 130 so it reads —

```
130 IF A$="Y" THEN RESTORE: GOTO 10
```

RUN

the program again. Press the letter "Y" to see the words again and see how **RESTORE** allows the program to continue using the same **DATA** statement.

Every time a value in a DATA statement is read, an invisible "counter" moves to the next value. RESTORE moves the counter back to the first value in the DATA line.

Look at this program:

```
10 DATA 1,2,3,4,5
20 READ A
30 PRINT A
40 RESTORE
50 GOTO 10
```

Without **RUN**ning this program, can you figure out what the computer will **PRINT**?

Enter the program into your computer to see if you are correct.

The computer will **PRINT** the number one over and over again because you told the computer to **RESTORE** the data and continue going to line 10.

Use RESTORE when you want the computer to READ the values in your DATA line again.

Have you ever seen the code where A=B, B=C, C=D and so on?

With this code, the names JAMES PARK would be KBNFT QBSL.

You can program the computer to encode (put into code) or to decode messages using this code. First you have to set up 2 arrays.

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 DATA A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z," "
30 DIM C1$(27) : DIM C2$(27)
40 FOR N=1 TO 27
50 READ C1$(N) ← (This makes C1$(1)=A, C1$(2)=B, and so on.)
60 NEXT N
70 RESTORE
80 FOR N=2 TO 27
90 READ C2$(N) ← (This makes C2$(2)=A, C2$(3)=B, and so on.)
100 NEXT N
110 READ C2$(1) ← (This makes C2$(1)=" ")
    
```

Now, give instructions to the user —

```

120 INPUT "DO YOU WANT TO 1) ENCODE OR 2) DECODE";X
130 IF X=1 THEN GOTO 160
140 IF X=2 THEN GOTO 240
150 IF X<>1 OR X<>2 THEN GOTO 120
    
```

To encode, the computer needs to go from C2\$ to C1\$. Add these lines to the program —

```

160 PRINT: PRINT "TYPE IN A MESSAGE. I WILL ENCODE IT."
170 PRINT: PRINT "(USE THE 'Z' KEY TO PUT IN A SPACE.)"
180 PRINT
190 GET M$
200 FOR N=1 TO 27
210 IF C2$(N)=M$ THEN PRINT C1$(N);
220 NEXT N
230 GOTO 190
    
```

To decode, the computer needs to go from C1\$ to C2\$. Add these lines to the program —

```

240 PRINT: PRINT "TYPE IN THE CODED MESSAGE."
250 PRINT TAB(5)" I WILL DECODE IT."
260 PRINT
270 GET M$
280 FOR N=1 TO 27
290 IF C1$(N)=M$ THEN PRINT C2$(N);
300 NEXT N
320 GOTO 270
    
```

RUN

the program. Try decoding this message:

UIJTAJTABADPEFEANFTTBHF

RUN

the program and encode a message.

TRYING YOUR HAND AT PROGRAMMING

16-1 Program your computer to **PRINT** scheduled appointments. Use **DATA** statements to record the month, date and appointment. Use **INPUT** statements to allow the user to choose the month and date they would like to see. Use **RESTORE** so the user has an opportunity to choose another month and/or date.

16-2 Use **DATA, READ** and **RESTORE** to write a graphics program that will display your first initial through all 15 colors.

LESSON 17

TWO-DIMENSIONAL ARRAYS

INTRODUCTION

In this lesson, you will be working with two-dimensional arrays. The kind of list you could develop with a single array is limited; you can only have one major category. With a two-dimensional array, you can use just one variable and have it organize the data into two categories.

Arrays are useful for sorting information and for keeping track of items used. The arrays that you have worked with so far contain one row with many boxes. These are useful for storing pieces of information in one category, such as the letters of the alphabet or the names in a list. But, sometimes you have a second piece of information that needs to be saved.

Using two-dimensional arrays you can have two categories for your list.

For example, you may have a birthday list of your friends or relatives. You want your list to have both their names and the dates of their birthdays. If you were doing this on paper, your list might look like this:

NAME	BIRTHDAY
Mom	April 11
Lynn	October 10
Jim	February 27

It is possible to set up an array that would look like your list. If 14 people are on your birthday list, an array with 14 rows would be needed. Since you want to save 2 pieces of information about each person (name and birthdate), two columns will be needed for each row. To set up such an array, a **DIM** statement is written like this:

```
DIM BD$(14,2)
```

(This is the number of rows.)
(This is the number of columns.)

After you have **RUN** the program, what name is in `BD$(1,1)`? Do you see that it will be `MOM`?

What name will be in `BD$(4,1)`?

`DREW` will be `BD$(4,1)`. Where will Frank's birthday be put?

Frank's birthday will be put in `BD$(14,2)`.

Two-dimensional arrays can be used with **RND** and **INT** to create new combinations of data. The following program is a horoscope generator. It has 5 rows of items in 2 columns. Every time it is **RUN**, the ideas in each of the columns are mixed up to produce a new horoscope.

Enter this program into your computer —

```
10 DIM H$(5,2)
20 PRINT "[CLR]"
30 PRINT "HERE IS A COMPUTERIZED HOROSCOPE:"
40 FOR X=1 TO 5
50 READ H$(X,1)
60 READ H$(X,2)
70 NEXT X
80 R=INT(RND(1)*5)+1
90 IF R=A THEN GOTO 80: REM GET A DIFFERENT NUMBER
100 M=INT(RND(1) *5)+1
110 IF M=B THEN GOTO 100: REM GET A DIFFERENT NUMBER
120 PRINT: PRINT
130 PRINT "YOU SHOULD "H$(R,1) "H$(M,2)".
140 A=R: B=M
150 PRINT: PRINT: PRINT
160 PRINT "WOULD YOU LIKE A DIFFERENT HOROSCOPE"
170 INPUT "(Y OR N)";A$
180 IF A$="Y" THEN GOTO 80
190 IF A$="N" THEN PRINT: PRINT "HOPE YOU LIKED THE
ONE YOU READ!": END
200 IF A$<>"Y" OR A$<>"N" THEN GOTO 170
1000 DATA KEEP, A JOURNAL
1010 DATA FIND, A NEW APARTMENT
1020 DATA BEWARE OF, A TALL STRANGER
1030 DATA WEIGH, A PEPPERMINT CANDY
1040 DATA SAVE, YOUR MONEY
```

RUN

the program several times. Are different horoscopes produced each time?

Many games use a combination of **RND** and arrays to produce a different game each time you want to play. The following program simulates shuffling a deck of cards. The cards are first placed in a two-dimensional array according to suit and rank. Then a card is randomly chosen and placed in a one-dimensional array in the order in which it is picked. The shuffled deck is then displayed.

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 PRINT "I AM SHUFFLING THE CARDS NOW."
30 DIM CD$(13,4), SH$(52)
40 FOR X=1 TO 13
50 READ A$
60 CD$(X,1)=A$+" SHFT X" ← (SHFT X will make a club.)
70 CD$(X,2)=A$+" SHFT Z" ← (SHFT Z will make a diamond.)
80 CD$(X,3)=A$+" SHFT S" ← (SHFT S will make a heart.)
90 CD$(X,4)=A$+" SHFT A" ← (SHFT A will make a spade.)
100 NEXT X
110 FOR X=1 TO 52
120 C=INT(RND(1)*13)+1
130 S=INT(RND(1)*4)+1
140 IF CD$(C,S)="*" THEN GOTO 120
150 SH$(X)=CD$(C,S)
160 CD$(C,S)="*"
170 NEXT X
180 PRINT: PRINT "HERE IS THE SHUFFLED DECK:"
190 PRINT
200 FOR X=1 TO 52
210 PRINT SH$(X) ,
220 NEXT X
1000 DATA ACE,2,3,4,5,6,7
1010 DATA 8,9,10,JACK,QUEEN,KING
    
```

RUN

the program. It will take several seconds for the computer to "shuffle" the cards.

There are many things you can do to make this program more interesting. Change the border and background colors. Change the print color so that hearts and diamonds are printed in red; clubs and spades are printed in black.

Here is a routine to deal 5 cards to two players. You could use this routine as the beginning of a computer card game.

Because you wouldn't want the "shuffled" deck to be displayed, replace lines 180-220 with these program lines —

```
180 PRINT "PLAYER #1:"  
190 FOR X=1 TO 9 STEP 2  
200 PRINT SH$(X)  
210 NEXT X  
220 PRINT TAB(15)"PLAYER #2:"  
230 FOR X=2 TO 10 STEP 2  
240 PRINT TAB(15) SH$(X)  
250 NEXT X
```

RUN
the program.

TRYING YOUR HAND AT PROGRAMMING

17-1 Imagine that your block has planned a potluck dinner. Everyone knows how great you are at programming, so your job is to keep a record of every family on the block along with the dish that they plan to bring. Using a two-dimensional array, **DATA** statements and **INPUT** statements, write the program.

OR

Choose a sport that you like. Write a program which will keep a record of the standing of each team in that league.

17-2 Write a program that gives you a definition of a word and then asks you for the word. Design a way that the computer will tell you if your answer is correct.

17-3 Write a program that will store a person's name and their telephone number in a two-dimensional array. Then use an **INPUT** statement to allow someone to choose a name and see that person's telephone number.

17-4 Revise the fortune cookie message program from the level 1 book in this series.

LESSON 18 ON GOSUB, ON GOTO

INTRODUCTION

*These two commands are a more powerful version of commands which you are already using — **GOTO** and **GOSUB**. With **GOTO** and **GOSUB** you can write programs which give you two choices. With **ON GOTO** and **ON GOSUB**, the computer can now be programmed to branch to one of many program lines or subroutines.*

When you have an **IF-THEN** statement that branches many ways, your program can get bulky.

Enter this program into your computer —

```
10 PRINT "[CLR]"
14 REM PRINT COLUMN HEADINGS
15 PRINT "1" TAB(6)"2" TAB(13)"3" TAB(20)"4" TAB(27)"5"
   TAB(35)"6"
20 FOR X=1 TO 1000
30 N=INT(RND(1)*6)+1
40 ON N GOSUB 200,300,400,500,600,700
50 NEXT X
70 PRINT
79 REM PRINT TOTALS FOR EACH NUMBER (1-6)
80 PRINT A TAB(6)B TAB(13)C TAB(20)D TAB(27)E TAB(35)F
90 END
200 A=A+1: REM ONES
290 RETURN
300 B=B+1: REM TWOS
390 RETURN
400 C=C+1: REM THREES
490 RETURN
500 D=D+1: REM FOURS
590 RETURN
600 E=E+1: REM FIVES
690 RETURN
```



```
700 F=F+1: REM SIXES
790 RETURN
```

RUN

the program.

This program chooses 1000 random numbers between 1 and 6, which are the numbers on dice. It then keeps track of how many times each number is chosen by using a **ON GOSUB** statement. After several seconds, the totals are then printed on the monitor screen.

LIST

line 40 and take a look at it.

```
40 ON N GOSUB 200,300,400,500,600,700  
does the same job as all of these lines:  
40 IF N=1 THEN GOSUB 200  
41 IF N=2 THEN GOSUB 300  
42 IF N=3 THEN GOSUB 400  
43 IF N=4 THEN GOSUB 500  
44 IF N=5 THEN GOSUB 600  
45 IF N=6 THEN GOSUB 700
```

In the last program, **ON GOSUB** branches to 6 lines. Experiment to find the most number of branches an **ON GOSUB** statement can handle.

The number of branches an **ON GOSUB** statement can have is decided by the length of the programming line. The program line number, the words **ON X GOSUB**, the branching line numbers and the commas cannot exceed 80 spaces, which is 2 lines on the 64 computer.

If your **ON GOSUB** statement is more than 2 lines, the computer will not record the line in its memory.

ON GOTO

ON GOTO works like **ON GOSUB**. Instead of going to a SUBroutine, **ON GOTO** goes to a specific line.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 PRINT: INPUT "INPUT A NUMBER FROM 1 TO 7";N
30 PRINT
40 ON N GOTO 50,60,70,80,90,100,110
50 PRINT "YOU PICKED A ONE": GOTO 20
60 PRINT "YOU PICKED A TWO": GOTO 20
70 PRINT "YOU PICKED A THREE": GOTO 20
80 PRINT "YOU PICKED A FOUR": GOTO 20
90 PRINT "YOU PICKED A FIVE": GOTO 20
100 PRINT "YOU PICKED A SIX": GOTO 20
110 PRINT "YOU PICKED A SEVEN": GOTO 20
```

RUN

the program and enter all the numbers between, and including, 1 and 7.

RUN

the program again and try answering with a zero or with a number that is greater than 7.

If the number entered as a response to the **INPUT** is not one of the numbers listed in the **ON GOTO** statement, the computer will go to the next program line following the **ON GOTO** statement.

ON GOTO, although limited by the length of the programming line (80 characters), can branch to more lines.

Add these lines to the program —

```
45 ON N-7 GOTO 120,130
120 PRINT "YOU PICKED AN EIGHT": GOTO 20
130 PRINT "YOU PICKED A NINE": GOTO 20
```

And change line 20 so it reads —

```
20 PRINT: INPUT "INPUT A NUMBER FROM 1 TO 9";N
```

RUN

the program and **INPUT** an 8 or a 9.

TRYING YOUR HAND AT PROGRAMMING

18-1 Begin with the dice rolling program using random integers from lesson 13. Write six subroutines using keyboard graphics to draw the dice face for each number (1-6) which the computer might choose. Add two **ON GOSUB** statements so that the computer displays the graphic of the dice when it chooses its number, and also when it chooses the player's number.

18-2 Write a multiple-choice quiz which uses an **ON GOTO** statement to branch to program lines which will give a response to the answer that is chosen. For example, when the number 1 is chosen as the answer to the computer's question, the computer might respond with "YOU'VE GOT TO BE KIDDING!!"

LESSON 19 POKE AND GRAPHICS

INTRODUCTION

*In level 1 of **TRAINING YOUR COMMODORE 64**, you designed graphic programs using the keyboard characters with the **PRINT** command. This method is useful because you can use the screen as a sketch pad. However, trying to make any kind of interesting design with all those tiny keyboard graphics can be time-consuming. In this lesson you will learn another way of producing graphics using the **POKE** command.*

Enter this program into your computer —

```
10 PRINT "[CLR]"  
20 POKE 55296,1  
30 POKE 1024,1
```

RUN

the program and add these lines —

```
40 POKE 55297,1  
50 POKE 1025,2
```

RUN

the program and add these lines —

```
60 POKE 55298,1  
70 POKE 1026,3
```

RUN

the program.

On the top line of your screen, you see the letters A, B and C written in white! How did they get there?

Every time any symbol (letter, number, punctuation mark, graphics symbol) appears on the screen, the 64 has stored some information at two locations in its memory. In one location, the computer stores a number from 0 to 15 describing the color to use when the character

appears. In another location, it stores a number from 0 to 225 which stands for the character that will actually be printed.

POKE puts information directly into the computer's memory.

POKING COLOR MEMORY LOCATIONS

Enter this program into your computer —

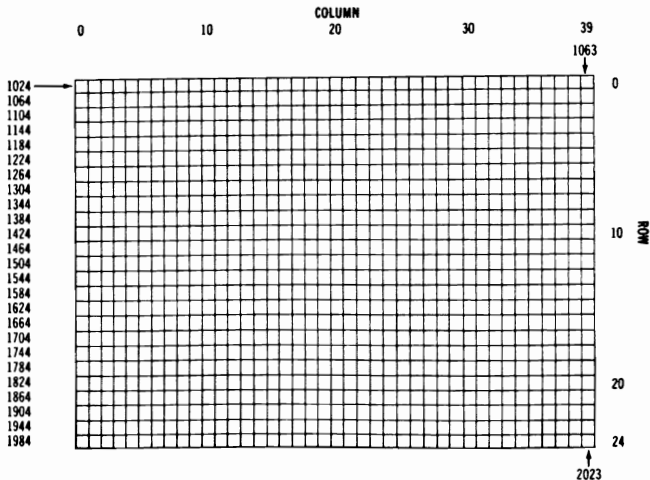
```

10 PRINT "[CLR]"
20 FOR S=0 TO 39
30 POKE 55296+S,5
40 POKE 1024+S,88
50 NEXT S
    
```

RUN

the program.

Memory locations 55296 through 56295 contain color information. In order to figure out what memory locations you need to **POKE**, think of your screen as being divided into 25 rows with 40 squares in each row. The screen is made up to 1000 squares. If you want to change the color of any screen square you need to **POKE** the color memory location that corresponds to that square.



The above program makes all of the color locations for the top row green so that any graphics character **POKE**d into the first screen line appears in green.

The memory locations for color are 55296 to 56295.

Here is a list of the numbers to **POKE** into color memory locations:

0 BLACK	8 ORANGE
1 WHITE	9 BROWN
2 RED	10 LIGHT RED
3 CYAN	11 GRAY 1
4 PURPLE	12 GRAY 2
5 GREEN	13 LIGHT GREEN
6 BLUE	14 LIGHT BLUE
7 YELLOW	15 GRAY 3

How would you change the program to produce orange shamrocks?

Line 30 would need to read —

```
30 POKE 55296+S,8
```

POKING CHARACTER MEMORY LOCATIONS

In addition to color, you need to **POKE** in the graphics design you want.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 FOR S=0 TO 39
30 POKE 55296+S,1
40 POKE 1024+S,88
50 NEXT S
```

RUN

the program.

The **FOR-NEXT** loop **POKEs** the graphic code for a shamrock into screen memory locations 1024 through 1064.

Each letter and graphic symbol is represented by a number. Here is a chart which shows the number to **POKE** into character memory locations:

LETTER	POKE NUMBER NORMAL	POKE NUMBER INVERSE
A	1	129
B	2	130
C	3	131
D	4	132
E	5	133
F	6	134
G	7	135
H	8	136
I	9	137
J	10	138
K	11	139
L	12	140
M	13	141
N	14	142
O	15	143
P	16	144
Q	17	145
R	18	146
S	19	147
T	20	148
U	21	149
V	22	150
W	23	151
X	24	152
Y	25	153
Z	26	154

POKE AND GRAPHICS

GRAPHIC SYMBOL	POKE NUMBER NORMAL	POKE NUMBER INVERSE	GRAPHIC SYMBOL	POKE NUMBER NORMAL	POKE NUMBER INVERSE
	64	192	SPACE	96	224
	65	193		97	225
	66	194		98	226
	67	195		99	227
	68	196		100	228
	69	197		101	229
	70	198		102	230
	71	199		103	231
	72	200		104	232
	73	201		105	233
	74	202		106	234
	75	203		107	235
	76	204		108	236
	77	205		109	237
	78	206		110	238
	79	207		111	239
	80	208		112	240
	81	209		113	241
	82	210		114	242
	83	211		115	243
	84	212		116	244
	85	213		117	245
	86	214		118	246
	87	215		119	247
	88	216		120	248
	89	217		121	249
	90	218		122	250
	91	219		123	251
	92	220		124	252
	93	221		125	253
	94	222		126	254
	95	223		127	255

Experiment with other colors and characters. What happens when you **POKE** the inverse **POKE** number?

The memory locations to place characters on the screen are 1024 to 2023.

POKING SCREEN AND BORDER COLORS

POKE can do a lot more than put characters on the screen.

Enter this program into your computer —

```
10 FOR C=0 TO 15
20 POKE 53280,C
30 PRINT "[CLR]"
40 PRINT "POKE 53280,"C
50 FOR T=1 TO 500:NEXT T
60 NEXT C
```

Every time a new number is **POKEd** into memory location 53280, the border color changes.

To change the background color, **POKE** a color code into memory location 53281. Change these lines so they read —

```
20 POKE 53281,C
40 PRINT "POKE 53281,"C
```

RUN

the program.

Memory location 53280 controls the border color of the screen.
Memory location 53281 controls the background color of the screen.

Enter the following program —

```
10 POKE 53280,1
20 POKE 53281,0
30 PRINT "THIS LOOKS LIKE A BLACKBOARD"
```

RUN

the program. The monitor screen should now be black with a white border. Change the program so you have a green background with a yellow border.

The program would need to read —

```
10 POKE 53280,7
20 POKE 53281,5
```

Hold down RUN/STOP and press RESTORE to return to the normal color display.

Try some more on your own. You can put a **POKE** command like these into any of your other programs to change the screen and border colors.

If the colors don't look accurate, adjust the color and tint on your monitor. (See level 1, **TRAINING YOUR COMMODORE 64** for a screen color test.)

POKING GRAPHIC DESIGNS

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 POKE 53280,0: REM BLACK BORDER
30 POKE 53281,1: REM WHITE SCREEN
40 P=INT(RND(1)*1001): REM NUMBER BETWEEN 0 AND 1000
50 C=INT(RND(1)*16): REM NUMBER BETWEEN 0 AND 15
60 POKE 1024+P,224
70 POKE 55296+P,C
80 GOTO 40
```

RUN

the program.

Press the

RUN/STOP key to break the program whenever you wish.

Press

SHIFT and **CLR/HOME** to clear the screen.

LIST

the program and look at the program lines so that you understand what they are doing.

Line 40 makes P a random whole number between 0 and 1000 because there are 1000 spaces on the monitor screen.

Remember that there are 40 spaces (0-39) per line and 25 lines (0-24) on the monitor screen. That means that there are 1000 (0-999) total screen positions.

Line 60 **POKE**s the graphic code 224 into location 1024+P which means a solid rectangle will be displayed P spaces from the home position on the monitor screen.

Line 50 makes C a random whole number between 0 and 15 which will be used as a color number.

Line 70 **POKE**s the values of C into location 55296+P which tells the computer to make the rectangle now located P spaces from HOME a particular color.

Remember: Color memory locations 55296 and Graphic memory location 1024 go with the first PRINT location on the screen.

When you are designing a graphics program, think of your computer screen as a piece of graph paper. In the Appendix, you will find graphics worksheets where you can plot your picture.

FORMULA FOR POKING GRAPHICS AND COLOR

Enter this new program into your computer —

```
10 PRINT "[CLR]"
20 POKE 1024,83
30 POKE 55296,2
40 POKE 1524,83
50 POKE 55796,2
```

RUN

the program.

Do you see a red heart in the first square and in the center of the screen?

One problem with **POK**ing graphics designs is figuring out the memory location you need to **POKE** for both the graphics design and color. It's

not too hard if your design begins in the first screen square but what if you want a design in the lower half of the screen?

Here is a formula to use to determine what memory location you need to **POKE** to enter a character.

$$\text{graphic memory location} = 1024 + X + (40 * Y)$$

So, if you wanted to **POKE** the letter "C" near the center of the screen,

$$X \text{ (the column)} = 20$$

$$Y \text{ (the row)} = 12$$

$$\text{memory location} = 1024 + 20 + (40 * 12)$$

which would be: **POKE 1524,3**

Enter this direct command into your computer —

POKE 1524,3

Nothing will appear in that character memory location until you also **POKE** a color into the same place on the screen.

Now, enter this color command into your computer —

POKE 1524 + 54272,1

Now you will see a white "C" appear near the middle of the screen.

The first character memory location is 1024. The first color memory location is 55296. So, the color memory location = graphic memory location + 54272.

It's easier, however, to let the computer do the figuring.

Enter this program into your computer —

10 PRINT "[CLR]"

20 X=10 ← (This is for column 10.)

30 Y=15 ← (This is for row 15.)

40 P=X+40*Y ← (Formula for determining memory location.)

50 POKE 1024+P,224 ← (POKing the value 224 places a solid rectangle on the screen.)

60 POKE 55296+P,2 ← (POKing the value 2 makes the screen position red.)

RUN

the program.

LIST

the program.

Change line 20 so it reads —

20 X=0

RUN

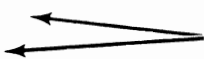
the program and see the rectangle in a new position on the left edge of the screen. How would you change the program so the rectangle was in the upper left-hand corner of the screen?

Line 30 would have to read:

30 Y=0

A horizontal line is made by plotting several small rectangles side by side. To do this, make these changes —

20 FOR X=0 TO 39
80 NEXT X



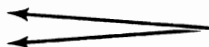
(These two lines loop 40 times so that 40 rectangles (character 224) are plotted to make a line across the screen.)

RUN

the program.

For a vertical line, you can stack the rectangles on top of each other. Change these lines so they read —

20 X=10
30 FOR Y=0 TO 24
80 NEXT Y



(These two lines loop 25 times which cause 25 rectangles to fill in the 25 lines down the screen.)

Now, let's draw a picture using the **POKE** command.

Enter this new program into your computer —

```

10 PRINT "[CLR]"
20 FOR X=12 TO 26 ← (This is for columns 12 through 26.)
30 Y=9 ← (This is for row 9.)
40 P=X+40*Y ← (Position = column+40*row.)
50 POKE 1024+P,224 ← (POKE a square (224) at position P.)
60 POKE 55296+P,1 ← (POKE the color white (1) at position P.)
70 NEXT X
    
```

RUN

the program and you will see the top of a white house.

Add these program lines —

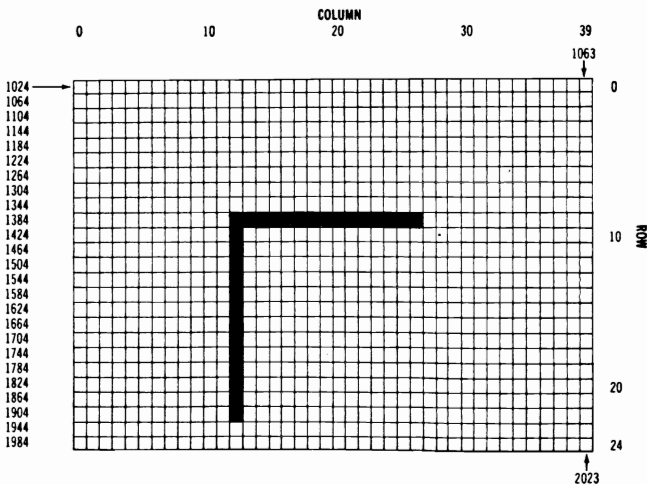
```

80 FOR Y=9 TO 22 ← (This is for rows 9 through 22.)
90 X=12 ← (This is for column 12.)
100 P=X+40*Y
110 POKE 1024+P,224
120 POKE 55296+P,1
130 NEXT Y
    
```

RUN

the program and you will see both the top and left side of a white house.

Here is a graphic worksheet showing the positions of the **POKEd** characters so far.



Add these program lines —

```
140 FOR Y=9 TO 22
150 X=26
160 P=X+40*Y
170 POKE 1024+P,224
180 POKE 55296+P,1
190 NEXT Y
```

RUN

the program and you will see a house. Take a look at these three sections of your program listing. Notice that lines 40-70, 100-130 and 160-190 are all the same. What changes is the value you give the variables X and Y.

Let's continue with your house. Add these program lines —

```
200 Y=1
210 FOR X=19 TO 12 STEP -1
220 P=X+40*Y
230 POKE 1024+P,224
240 POKE 55296+P,2
250 Y=Y+1
260 NEXT X
```

RUN

the program and you will see the left side of a slanted red roof on top of your white house.

Add these program lines —

```
270 Y=1
280 FOR X=19 TO 26
290 P=X+40*Y
300 POKE 1024+P,224
310 POKE 55296+P,2
320 Y=Y+1
330 NEXT X
```

RUN

the program and you will see a red roof on a white house. How would you enter that house? It needs a door.

Add these program lines —

```
340 FOR X=15 TO 18
350 Y=18
360 P=X+40*Y
370 POKE 1024+P,224
380 POKE 55296+P,9
390 NEXT X
400 FOR Y=18 TO 22
410 X=15
420 P=X+40*Y
430 POKE 1024+P,224
440 POKE 55296+P,9
450 NEXT Y
460 FOR Y=18 TO 22
470 X=18
480 P=X+40*Y
490 POKE 1024+P,224
500 POKE 55296+P,9
510 NEXT Y
```

RUN

the program and you will see the addition of a brown door. How about a window for some sunlight?

Add these program lines —

```
520 FOR X=22 TO 24
530 Y=12
540 P=X+40*Y
550 POKE 1024+P,224
560 POKE 55296+P,7
570 NEXT X
580 FOR X=22 TO 24
590 Y=13
600 P=X+40*Y
610 POKE 1024+P,224
620 POKE 55296+P,7
630 NEXT X
```

RUN

the program and you will see a completed house.

Want to change the position and size of the window?

Change these lines so they read —

```
520 FOR X=20 TO 24
530 Y=14
580 FOR X=20 TO 24
590 Y=15
```

RUN

the program and see a larger yellow window placed at a lower level.

Add these lines to change the border and screen colors —

```
640 POKE 53280,7
650 POKE 53281,14
```

RUN

the program again.

Notice that you cannot see the word "READY" and the blinking cursor. That is because they are in the same color as the background color.

Change line 650 so it reads —

```
650 POKE 53281,12
```

RUN

the program.

It looks like a grey, cloudy day, doesn't it? To remove the word "READY" and the blinking cursor from the screen, add this endless loop —

```
660 GOTO 660
```

RUN

the program.

Experiment with ways to "customize" this house.

Remember the birthday cake program from the level 1 book? Program your computer to draw a birthday cake using **POKE** commands.

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 Y=10 ← (This line sets the row at 10.)
30 FOR B=1 TO 10 ← (This causes the loop to run 10 times.)
40 FOR X=10 TO 28 ← (This line sets the columns between 10 and 28.)
50 P=X+40*Y
60 POKE 1024+P,224 ← (This line POKEs a rectangle.)
70 POKE 55296+P,1 ← (This line POKEs the color, white.)
80 NEXT X
90 Y=Y+1 ← (This line increases the row by 1.)
100 NEXT B

```

RUN

the program to see the cake.

Add these lines to make a green candle —

```

110 FOR Y=3 TO 9
120 X=19
130 P=X+40*Y
140 POKE 1024+P,224
150 POKE 55296+P,5
160 NEXT Y

```

RUN

the program.

The candle on your cake needs to be lit. What program lines would you need to add to make a yellow flame? Use the graphics worksheet in the Appendix to plot the locations of X and Y for the candle flame.

Here's one suggestion:

```

170 X=19: Y=2
180 P=X+40*Y
190 POKE 1024+P,90
200 POKE 55296+P,7

```

RUN

the program.

Add these lines to print the red words "HAPPY" on the cake in row 12 —

```

210 X=15: Y=12
220 P=X+40*Y
230 POKE 1024+P,8
240 POKE 55296+P,2
250 X=17
260 P=X+40*Y
270 POKE 1024+P,1
280 POKE 55296+P,2
290 X=19
300 P=X+40*Y
310 POKE 1024+P,16
320 POKE 55296+P,2
330 X=21
340 P=X+40*Y
350 POKE 1024+P,16
360 POKE 55296+P,2
370 X=23
380 P=X+40*Y
390 POKE 1024+P,25
400 POKE 55296+P,2
    
```

RUN

the program.

Add these lines to complete the program —

```

410 X=12: Y=15
420 P=X+40*Y
430 POKE 1024+P,2
440 POKE 55296+P,2
450 X=14
460 P=X+40*Y
470 POKE 1024+P,9
480 POKE 55296+P,2
490 X=16
500 P=X+40*Y
510 POKE 1024+P,18
520 POKE 55296+P,2
530 X=18
540 P=X+40*Y
550 POKE 1024+P,20
560 POKE 55296+P,2
    
```

```
570 X=20
580 P=X+40*Y
590 POKE 1024+P,8
600 POKE 55296+P,2
610 X=22
620 P=X+40*Y
630 POKE 1024+P,4
640 POKE 55296+P,2
650 X=24
660 P=X+40*Y
670 POKE 1024+P,1
680 POKE 55296+P,2
690 X=26
700 P=X+40*Y
710 POKE 1024+P,25
720 POKE 55296+P,2
```

RUN

the program.

Change the background and border colors of the screen by adding these lines —

```
15 POKE 53280,5
17 POKE 53281,14
```

RUN

the program.

TRYING YOUR HAND AT PROGRAMMING

Remember that you can change the screen and border colors on any of your programs.

19-1 Create a program that will change the screen and border colors and then **POKE** the letters of the alphabet into a memory location halfway down the screen.

19-2 Write a **POKE** program that will list your name on a diagonal across the screen. Use **GOSUB** statements so that there is a 2-second delay between the printing of each letter.

19-3 Write a program that uses 4 subroutines which contain **POKE** graphics. Let your main program allow the user to **INPUT** a number to see a particular graphic. Use **ON GOSUB** to branch to the subroutines.

19-4 Ask your friends and neighbors this question — What is your favorite television show? Then, using **POKE** commands, program your computer to draw bar graphs to show the total amount of people you asked, and the top four favorite shows. Use **PRINT** and **SPC** to print words to explain the graphs or use **POKE** commands if you wish. When you **RUN** your program it may be similar to this:

T	D	A	D	D
O	U		R	A
T	K	T		L
A	E	E	W	L
L	S	A	H	A
		M	O	S



LESSON 20

ANIMATED GRAPHICS

INTRODUCTION

*The programming term for animation is "memory-mapped video." It sounds impressive, and it is. In this lesson, you will learn one way of adding movement to a program. The method for doing this is quite simple. You **POKE** in a picture, erase it and then move the picture by **POKING** the exact same design in a different location on the screen.*

Now you are ready to use the ideas of the previous lesson to animate something — to move it around the screen.

Clear the screen and your computer's memory and then type —

POKE 53280,5

Press
RETURN

Type —

POKE 53281,7

Press
RETURN

Now you have a green border and a yellow screen.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 POKE 1524,81
30 POKE 1524 + 54272,0
```

RUN
the program.

You should have a black ball right in the middle of your screen. 1524 is the memory location for the center of the screen and 81 is the code for a solid circle or ball. Remember that the color memory location is 54272 more than the character memory location.

Change line 30 so it reads —

```
30 POKE 1524 + 54272,2
```

RUN

the program.

Now the ball is red!

Change the program so the ball is white.

You could have written a line like this:

```
30 POKE 1524 + 54272,1
```

Return the screen to its original color.

Holding down the RUN/STOP key and pressing the RESTORE key will return the monitor screen to its original color.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 POKE 53280,5: REM GREEN BORDER
30 POKE 53281,7: REM YELLOW SCREEN
40 R=1: REM STARTING POSITION FOR ROW
50 C=1: REM STARTING POSITION FOR COLUMN
60 H=1: REM STARTING HORIZONTAL POSITION
70 V=1: REM STARTING VERTICAL POSITION
80 P=R+40*C: REM STARTING POSITION
90 POKE 1024+P,81: REM PLOTS A BALL
100 FOR D=1 TO 10: NEXT D
110 POKE 1024+P,32: REM ERASES WITH A BLANK
120 R=R+H
130 IF R=0 THEN H=-H ← (This line finds the left side of
                        the screen and reverses the ball.)
```

```

140 IF R=39 THEN H=-H ← (This line finds the right side of
150 C=C+V                the screen and reverses the ball.)
160 IF C=0 THEN V=-V ← (This line finds the top of the
                        screen and reverses the ball.)
170 IF C=24 THEN V=-V ← (This line finds the bottom of the
180 GOTO 80                screen and reverses the ball.)

```

RUN

the program.

How do you like that? You should have a ball bouncing all over your screen.

Here's a summary of what the program does.

Lines 40 and 50 give the starting position for the ball.

Lines 60 and 70 tell how far the ball will move each time and in which directions. H will increase by 1 (it will move one space to the right). And V will increase by 1 (it will move down one space).

Lines 80 and 90 calculate the position in memory and **POKE** the character, 81, into that memory location. That causes a ball to appear on the screen, C spaces over and R spaces down.

Line 100 is a time-delay loop. Try changing the speed of the ball.

You could have slowed the ball down by writing a line like this:

```
100 FOR D=1 TO 50: NEXT D
```

Try different numbers and find a speed you like.

Line 110 **POKEs** character code 32 into the memory location corresponding to the ball's position. 32 is the character code for a blank.

Delete line 110 and see what would happen without this line.

RUN

the program.

Now the ball leaves quite a trail, doesn't it? It's important to "erase" the ball before it moves to its new position.

Reenter line 110 so it reads —

```
110 POKE 1024+P,32
```

Lines 120 and 150 move the ball to its new position. Then lines 130, 140, 160 and 170 check to see **IF** the ball is at the edge of the playing field. **IF** it is, **THEN** the direction of its movement is changed.

RUN

the program again.

After you've seen the ball bounce all over the screen for awhile, there's not much to it, is there?

Add these lines to the program —

```
31 FOR L=1 TO 12  
32 S=INT(RND(1)*999) ← (S sets random locations.)  
33 POKE 1024+S,224  
34 NEXT L
```

RUN

the program.

Now there are some obstacles for the ball to run into. How many blocks are on the screen? If you want more blocks, change line 31.

RUN

the program several times to see how line 32 randomly changes the positions of the blocks.

Add these lines to the program —

```
175 B=R+40*C  
177 IF PEEK(1024+B)=224 THEN H=-H: V=-V:GOTO 130
```

RUN

the program.

Now, the ball reverses its position when it runs into a block.

LIST

lines 175-177.

Line 175 calculates the expected next position of the ball. Then line 177 **PEEKs** into memory. **IF** character code 160 is found, that means there's a block on the screen so H and V are made opposite so the ball appears to bounce off the block.

PEEK and **POKE** are two commands that work very well together. You have used the **POKE** command to enter color and graphic designs right into the computer's memory. With the **PEEK** command, you can identify what is in any memory location.

PEEK takes a "peek" at a memory location. The command PRINT PEEK(number) will show you that information.

STOP

the program by pressing the **RUN/STOP** key but **DO NOT PRESS RESTORE**.

Enter this direct command into your computer —

PRINT PEEK(53281) AND 15 ← (AND 15 tells the computer to translate the answer into the color code.)

Press
RETURN

The computer will respond with 7. That is the number you earlier **POKEd** into memory location 53281 to get the screen color.

Enter this direct command into your computer —

PRINT PEEK(53280) AND 15

Press
RETURN

The computer will respond with 5. That is the number you earlier **POKEd** into memory location 53280 to get the border color on the monitor screen.

You can **PEEK** into any memory location you want.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 POKE 1024,1
30 PRINT "HERE'S WHAT'S IN MEMORY LOCATION 1024."
40 PRINT PEEK(1024)
50 POKE 1063,2
60 PRINT "HERE'S WHAT'S IN MEMORY LOCATION 1063."
70 PRINT PEEK(1063)
```

RUN

the program.

TRYING YOUR HAND AT PROGRAMMING

20-1 Write a program to make a rectangle move back and forth across the screen. Use the variable $Y=20$ for the row and keep it the same throughout your program. Use the variable X for the column and change it so it moves. Check when it reaches the edge and reverse its direction. Remember to change the color and border of the screen.

Then change your program to make an "inchworm." Your "inchworm" should be two rectangles that move back and forth together from one side of your screen to the other.

20-2 Design a car and have it move across the screen. Decide where you want the car positioned (e.g. $X=7: Y=12$). In the rest of the program use lines similar to this:

100 P=(X-1)+40*(Y+2)

When you have finished **POK**ing the car, add a clear screen command. Then, add a line like this:

500 X=X+3: GOTO 10

This will make all of the values of X throughout the program increase by 3 and, when the program returns to the beginning of the program, the car will move to the right.

20-3 Write a program using **POKE** commands to produce an animated billboard or sign which could be used by a club or a business.

REVIEW 5

Choose a word which will match each definition.

DATA	PEEK
flag	POKE
multi-dimensional array	POKE 55296,2
ON GOSUB	READ
ON GOTO	RESTORE

1. _____ tells the computer to go "look for" a DATA value and assign that value to a variable.
2. _____ statements store information to be read by the computer.
3. A _____ is a number or symbol at the end of a DATA statement which, when combined with an IF-THEN statement, will stop the computer from showing an OUT OF DATA ERROR.
4. _____ moves the DATA "counter" back to the first DATA value.
5. A _____ can store information by two or more categories.
6. _____ takes the place of several IF-THEN, GOTO statements.
7. _____ works like ON GOTO but branches to sub-routines.
8. _____ puts information directly into the computer's memory.
9. _____ takes a look at a memory location.
10. _____ would mean color whatever is in the first screen position, red.

LESSON 21

MUSIC

INTRODUCTION

Your Commodore® 64 can do a lot with sound effects and music. It has 3 separate voices which is similar to having three people singing in unison or in two or three-part harmony. All of this is possible because of the Commodore's sound chip, "SID" (Sound Interface Device). In this lesson, you will learn how to program volume, tone, pitch and various types of sound using your computer.

Before making any sound, it is necessary to clear the sound chip inside the computer. The sound chip is a Sound Interface Device which will be called SID from now on. This line should appear at the beginning of each sound program to **POKE** the value zero into all of the memory locations related to sound on the Commodore 64.

FOR C=54272 TO 54296: POKE C,0: NEXT C

To make one voice produce a noise, there are five different **POKE** instructions that are needed to control how it will sound. These **POKE** statements set a **VOLUME**, an **ATTACK/DECAY**, a **WAVEFORM**, a **HIGH FREQUENCY** and a **LOW FREQUENCY**.

SETTING VOLUME

VOLUME is how loud or soft the sound will be. The volume control number or memory location is 54296. You must **POKE** a number there to turn the speakers on or off. For example, **POKING** 54296,0 turns the speakers off. You can set the volume range between 0 and 15 but most of the time you will use 15 (e.g. **POKE** 54296,15). Be sure the volume on the monitor is turned up enough to hear the sounds and be sure the sound cable on the back of your monitor is connected.

Volume is how loud the sound will be. It can be set between 0 and 15.

POKE 54296,number

SETTING TWO FREQUENCIES FOR EACH NOTE

To play a note, you must **POKE** a number into the HIGH FREQUENCY memory location and **POKE** another number into the LOW FREQUENCY memory location. (It's like pressing two keys on a piano to get one note!).

To make sounds with your computer right away, enter this program into your computer (You will learn about lines 30 and 40 on the next few pages.) —

5 REM CLEAR SOUND CHIP

10 FOR C=54272 TO 54296: POKE C,0: NEXT C

20 POKE 54296,15 ← (This sets the highest VOLUME.)

30 POKE 54277,240

40 POKE 54276,33

50 POKE 54273,16 ← (This sets the HIGH FREQUENCY for the note C.)

60 POKE 54272,195 ← (This sets the LOW FREQUENCY for the note C.)

RUN

the program.

Listen to that. The sound begins softly but gets louder before stopping.

Look at the HIGH and LOW FREQUENCY numbers in lines 50 and 60. For voice 1, the HIGH FREQUENCY number is always 54273 and the LOW FREQUENCY is always 54272. The number after the comma in the **POKE** statement can be changed and determines what note is played.

For each note you must POKE two numbers — one for high frequency, another for low frequency. Either may be set between 0 and 255.

high frequency — POKE 54273,number

low frequency — POKE 54272,number

The actual notes the computer can play range over eight octaves, which is more notes than a piano has. The MUSIC NOTE CHART in the Appendix lists the HIGH and LOW FREQUENCY values for only two of these octaves. If you want to use the other six octaves, refer to the Commodore User's Manual.

As you can see, making sound or music with your computer is a little different from playing a musical instrument. To produce the last note, how many numbers did you have to **POKE**?

SETTING WAVEFORM

In your first sound program, line 40 sets the WAVEFORM.

To understand WAVEFORM, think of the difference in sound between a piano and a trumpet. You can play the note C on each, but it will not sound the same. The same is true of the sounds of an explosion and the wind. WAVEFORM is what makes them sound different. With the 64 computer, there are 4 WAVEFORMs which can be used to make sounds like different musical instruments.

Waveform sets the shape of the sound. POKE 54276,number

Here is a chart which shows you the 4 WAVEFORMs and the value you need to **POKE** to produce them in voice 1. Each different noise has its own name. The names describe the shape of the sound waves. You might want to give the WAVEFORMs names that make sense to you.

	triangle	sawtooth	pulse	noise
POKE 54276	17	33	65	129

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 PRINT "LET'S BEGIN WITH THE NOTE C"
30 FOR T=1 TO 8
40 FOR C=54272 TO 54296: POKE C,0: NEXT C (This line clears the
                                             sound chip in the
                                             computer.)
50 POKE 54296,15 ← (This lines turns on the VOLUME.)
60 POKE 54277,15
70 POKE 54276,17 ← (This line sets a triangle WAVEFORM.)
80 INPUT "HIGH FREQUENCY VALUE=";H
90 INPUT "LOW FREQUENCY VALUE=";L
100 POKE 54273,H: POKE 54272,L ← (This line POKES a value for a note
                                  from the INPUT statements in lines
                                  80 and 90.)

110 FOR D=1 TO 2000: NEXT D ← (This line determines how long each
120 NEXT T                       note will play.)

```


RUN

the program and **INPUT** these numbers:

- HIGH FREQUENCY VALUE — 16 ← (This is the note C.)
LOW FREQUENCY VALUE — 195 ←

- HIGH FREQUENCY VALUE — 18 ← (This is the note D.)
LOW FREQUENCY VALUE — 209 ←

- HIGH FREQUENCY VALUE — 21 ← (This is the note E.)
LOW FREQUENCY VALUE — 31 ←

- HIGH FREQUENCY VALUE — 22 ← (This is the note F.)
LOW FREQUENCY VALUE — 96 ←

- HIGH FREQUENCY VALUE — 25 ← (This is the note G.)
LOW FREQUENCY VALUE — 30 ←

- HIGH FREQUENCY VALUE — 28 ← (This is the note A.)
LOW FREQUENCY VALUE — 49 ←

- HIGH FREQUENCY VALUE — 31 ← (This is the note B.)
LOW FREQUENCY VALUE — 165 ←

- HIGH FREQUENCY VALUE — 33 ← (This is the note C.)
LOW FREQUENCY VALUE — 135 ←

Your first computerized musical scale!

RUN

the program again and try some other value for HIGH and LOW FREQUENCY from the MUSIC NOTE CHART in the Appendix.

The values on the MUSIC NOTE CHART make a certain note.

RUN

the program again and try values of your own and see that they produce sounds, too. For the HIGH FREQUENCY and LOW FREQUENCY value, try any numbers between 0 and 255. If you use numbers smaller or larger than this, you will get an ILLEGAL QUANTITY ERROR.

In line 70 of the SCALE program, you set the WAVEFORM at 17 which is the sound of a triangle. To make a sawtooth sound, change line 70 so it reads —

70 POKE 54276,33

RUN

the program and **INPUT** numbers for the HIGH and LOW FREQUENCIES.

Can you hear the difference between the two sounds? When you use WAVEFORM 33 (the sawtooth), it does sound sharper than the triangle form.

By using a **FOR-NEXT** delay loop you can change the length of the sound.

LIST

line 110 of the SCALE program.

Use your editing skills to change the **FOR-NEXT** loop to make the note shorter by changing line 110 so it reads —

110 FOR X=1 TO 50: NEXT X

RUN

the program.

A FOR-NEXT delay loop can be used to set the length of a sound.

SETTING THE ATTACK/DECAY

Line 60 in the SCALE program sets an ATTACK/DECAY. Each note or noise begins with no sound and grows to whatever number you give it for VOLUME. ATTACK/DECAY determines how fast each note will reach its loudest volume and then stop. The range for ATTACK/DECAY can be set from 0 to 255. **POKE 54277,240** takes the longest to reach its maximum volume.

Attack/decay sets the speed at which a note rises to its highest volume and then stops. It can be set between 0 and 255.

POKE 54277,number

This program will produce all of the ATTACK/DECAY variations for a single note.

Enter this program into your computer —

```
5 PRINT "[CLR]"
10 FOR C=54272 TO 54296: POKE C,0: NEXT C
20 FOR X=0 TO 255
30 PRINT X,
40 POKE 54296,15
50 POKE 54277,X
60 POKE 54278,4
70 POKE 54276,33
80 POKE 54273,33: POKE 54272,135
90 GET A$
100 IF A$="" THEN GOTO 90
110 POKE 54276,0
120 NEXT X
130 FOR C=54272 TO 54296: POKE C,0: NEXT C
```

(no spaces)

RUN

the program. The value of the ATTACK/DECAY setting will appear on the screen while you hear that setting. Press any key whenever you are ready to hear the next ATTACK/DECAY setting. Some settings need more time than others. BE SURE TO GIVE EACH NOTE ENOUGH TIME TO REACH ITS HIGHEST VOLUME.

Enter this program into your computer —

```
10 FOR C=54272 TO 54296: POKE C,0: NEXT C
20 POKE 54296,15
30 POKE 54277,15
40 POKE 54276,33
50 FOR J=175 TO 90 STEP -2
60 FOR K=44 TO 59 STEP 3
70 POKE 54273,K: POKE 54272,J
80 NEXT K
90 NEXT J
100 POKE 54296,0: POKE 54277,0: POKE 54276,0
```

(One loop nested within another.)

RUN

the program a few times.

Sounds like a burglar alarm, doesn't it?

Lines 50 and 60 give the numbers which are **POKE**d into the HIGH and LOW FREQUENCY locations in line 70. This is a program which uses notes other than those on the MUSIC NOTE CHART.

USING VARIABLES TO TALK TO SID

A faster way to enter these **POKE** numbers is to use variables. That way you can enter one letter rather than five numbers. In the next program, V=VOLUME, W=WAVEFORM, H=HIGH FREQUENCY and L=LOW FREQUENCY.

Enter this program into your computer —

```

10 FOR C=54272 TO 54296: POKE C,0: NEXT C
20 V=54296: A=54277: W=54276: H=54273: L=54272
30 FOR X=15 TO 0 STEP -1 (STEP is used to decrease the VOLUME.)
40 POKE V,X: POKE A,15: POKE W,129: POKE H,22: POKE L,196
50 FOR T=1 TO 90: NEXT T
60 NEXT X
70 FOR C=54272 TO 54296: POKE C,0: NEXT C

```

RUN

the program to hear the sound of an explosion.

Look at line 70 in this program and compare it with line 100 in the last program. Either line will turn off the sound chip. Use whichever is easier for you.

Change line 50 to make the explosion very short, like a rifle shot.

You could have written a line like this:

```

50 FOR T=1 TO 5:NEXT T

```

Change line 50 to make the explosion very long, like a building demolition blast.

You could have written a line like this:

```

50 FOR T=1 TO 200: NEXT T

```

Add this line to the program —

```

80 GOTO 10

```

RUN

the program and see how it changes.

Change the numbers in the **POKE** statement for HIGH and LOW FREQUENCY to change the pitch of the explosion.

You could have raised the pitch by writing a line like this:

```
40 POKE V,X: POKE A,15: POKE W,129: POKE H,50: POKE L,60
```

Enter this program into your computer —

```
10 FOR C=54272 TO 54296: POKE C,0: NEXT C  
20 V=54296: A=54277: W=54276: H=54273: L=54272  
30 POKE V,15: POKE A,240: POKE W,17  
40 FOR T=1 TO 100  
50 N=INT(RND(1)*15)+16  
60 N2=INT(RND(1)*200)+37  
70 POKE H,N: POKE L,N2  
80 FOR D=1 TO 10: NEXT D  
90 NEXT T  
100 FOR C=54272 TO 54296: POKE C,0: NEXT C
```

RUN

the program.

This program gives you sound effects by playing a group of random notes. Line 70 takes the random numbers produced in lines 50 and 60 and **POKE**s them into HIGH and LOW FREQUENCY locations.

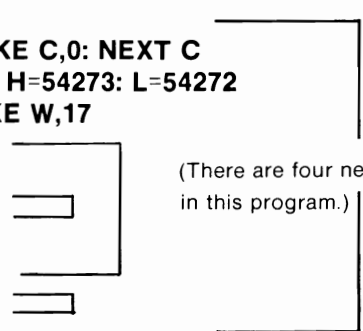
Change the pitch of the sound effect.

You could have lowered the pitch by writing a line like this:

```
50 N=INT(RND(1)*15)
```

Enter this program into your computer —

```
10 FOR N=1 TO 5  
20 FOR C=54272 TO 54296: POKE C,0: NEXT C  
30 V=54296: A=54277: W=54276: H=54273: L=54272  
40 POKE V,15: POKE A,15: POKE W,17  
50 FOR J=1 TO 50  
60 POKE H,44: POKE L,193  
70 FOR K=1 TO 8: NEXT K  
80 POKE H,0: POKE L,0  
90 NEXT J  
100 FOR T=1 TO 1500: NEXT T  
110 NEXT N  
120 FOR C=54272 TO 54296: POKE C,0: NEXT C
```



(There are four nested loops in this program.)

RUN

the program.

Do you feel like you should pick up the receiver and answer your computer?

Line 10 tells the computer to make a ringing sound 5 times. Line 100 tells the computer to wait approximately 2 seconds between rings. Line 60 sets the pitch of the ringer. If you think of a bell on a telephone, line 70 tells how fast the striker hits the bell.

MUSICODE

In past programs, you entered variables for each memory location in SID, then **POKE**d values into each variable. You will continue to do that but here is another way to assign the variables.

Voice 1 begins with memory location 54272 and goes to memory location 54278. Here is a chart of SID's voice 1:

name	Low Frequency	High Frequency	Low PulseWidth	High PulseWidth	Wave Form	Attack Decay	Sustain/Release
location	54272	54273	54274	54275	54276	54277	54278

Because the numbers are one right after another on a number line (which means they are consecutive), the memory locations on the chart above could be changed to look like this:

name	Low Frequency	High Frequency	Low PulseWidth	High PulseWidth	Wave Form	Attack/Decay	Sustain/Release
location	54272	54272 +1	54272 +2	54272 +3	54272 +4	54272 +5	54272 +6

Enter this program into your computer —

```

5 PRINT "[CLR]"
10 FOR C=54272 TO 54296: POKE C,0: NEXT C
20 L=54272: REM LOW FREQUENCY LOCATION
25 H=L+1: REM HIGH FREQUENCY IS LOW FREQUENCY +1
30 P=L+2: REM LOW PULSE WIDTH IS LOW FREQUENCY +2
35 U=L+3: REM HIGH PULSE WIDTH IS LOW FREQUENCY +3
40 W=L+4: REM WAVEFORM IS LOW FREQUENCY +4
45 A=L+5: REM ATTACK/DECAY IS LOW FREQUENCY +5
50 S=L+6: REM SUSTAIN/RELEASE IS LOW FREQUENCY +6
60 V=54296: REM V STANDS FOR VOLUME

```

If you **RUN** the program now you will not hear any sound because all this program does is assign the variable names. To hear sounds, you need to **POKE** a number into each memory location.

Look at lines 30 and 35 of the MUSICCODE program. They set the variable for a LOW and a HIGH PULSE WIDTH. Numbers **POKE**d into these memory locations change a sound by quickly turning them on and off. Experiment on your own with these two memory locations.

SAVE this program on tape or disk and call it MUSICCODE. Then whenever you want to write a sound program, just **LOAD** "MUSIC-CODE". This will save time because you won't have to retype all of the variables.

USING DATA AND READ FOR HIGH AND LOW FREQUENCY

You can program SID to play some notes but instead of entering each note separately, you can put the notes in **DATA** statements and have the computer **READ** them.

Enter this program into your computer (If you **SAVE**d MUSICCODE, **LOAD** it now and add to it beginning with line 70.) —

```
5 PRINT "[CLR]"
10 FOR C=54272 TO 54296: POKE C,0: NEXT C
20 L=54272: REM LOW FREQUENCY LOCATION
25 H=L+1: REM HIGH FREQUENCY IS LOW FREQUENCY +1
30 P=L+2: REM LOW PULSE WIDTH IS LOW FREQUENCY +2
35 U=L+3: REM HIGH PULSE WIDTH IS LOW FREQUENCY +3
40 W=L+4: REM WAVEFORM IS LOW FREQUENCY +4
45 A=L+5: REM ATTACK/DECAY IS LOW FREQUENCY +5
50 S=L+6: REM SUSTAIN/RELEASE IS LOW FREQUENCY +6
60 V=54296: REM V STANDS FOR VOLUME
70 POKE V,15
80 POKE A,36
90 POKE S,136
100 POKE W,17
110 FOR X=1 TO 8
120 READ A,B
130 POKE H,A: POKE L,B
140 FOR D=1 TO 250: NEXT D
150 NEXT X
160 POKE V,0: POKE W,0
```

```
1000 DATA 33,135
1010 DATA 37,162
1020 DATA 42,62
1030 DATA 44,193
1040 DATA 50,60
1050 DATA 56,99
1060 DATA 63,75
1070 DATA 67,15
```

RUN

the program. Your second computerized musical scale!

Add a **RESTORE** command and a **GOTO** statement so the scale will play over and over again.

You could have written lines like this:

```
170 RESTORE
180 GOTO 70
```

Remember to hold down the STOP key and press the RESTORE key to BREAK a program and return to a blank screen.

Change the **DATA** statements so they will play other notes.

You could have changed the **DATA** statements like this:

```
1000 DATA 33,135
1010 DATA 43,52
1020 DATA 50,60
1030 DATA 67,15
1040 DATA 50,60
1050 DATA 43,52
1060 DATA 33,135
1070 DATA 67,15
```

SETTING SUSTAIN/RELEASE

Another thing that determines the type of sound you hear is SUSTAIN/RELEASE at memory location 54278 or L+6.

ATTACK/DECAY and SUSTAIN/RELEASE all work together to produce the volume for each single note. When the computer first plays

the note, the ATTACK command tells how quickly the note rises to its highest volume. DECAY controls how fast the value for the note falls to the SUSTAINED level. The SUSTAIN number tells how loud the note will stay until turned off. RELEASE determines how quickly the note will fall to zero volume.

Sustain is how loud the note will stay until turned off and release is how quickly it falls to zero volume. It can be set between 0 and 255.

POKE 54278,number

So, the volume of each note changes four times from when the note is first played until you no longer hear it. (And you thought all you needed to do was turn up the volume on the stereo!)

Think of an airplane glider to show how these settings work together:

Throw the airplane into the air ATTACK
Plane reaches its peak VOLUME
Plane drops a little DECAY
Plane reaches its "cruising" altitude SUSTAIN
Plane makes its descent RELEASE

Change line 80 in your second SCALE program so it reads —

80 POKE A,10

This will give a fairly short ATTACK/DECAY rate to allow the SUSTAIN/RELEASE changes to be heard more easily.

Try these values with the SUSTAIN/RELEASE variable in line 90:

16 55 255

Making sound on the 64 is like making pizza. You have 3 main ingredients (WAVEFORM, ATTACK/DECAY, SUSTAIN/RELEASE) to make different types of sound. Depending on how you put the ingredients together, you can have a pizza with a thin crust or a deep-dish pizza. Which value you decide to give WAVEFORM, ATTACK/DECAY or SUSTAIN/RELEASE makes the 64 produce a sound like a piano, a harmonica, or a 747.

USING DATA AND READ TO CHANGE NOTE LENGTH

When you play music or listen to music, are all the notes played for the same length of time? Of course not! Some are played quickly, some are played slowly, and most are played somewhere in between.


The length of time a note is played is called the duration or count of the note.

In musical notation, a note is called a quarter note, a half note, a dotted sixteenth note, or some other similar name. The larger the fraction, the longer the note is played. The computer needs to be told how long to play each note. A **FOR-NEXT** delay loop that counts from 1 to 250 is about the length of time a quarter note is played.

A quarter note plays about as long as it takes the computer to count to 250.

You have changed the **FOR-NEXT** delay loop in previous programs so the sound will be longer or shorter. Changing the counting **FOR-NEXT** loop will work if every note in a song is the same length, but most songs are not like this. Each note has its own duration.

The following chart has each type of musical note with its count or duration value. Sometimes the count is different, but in most music this is accurate.

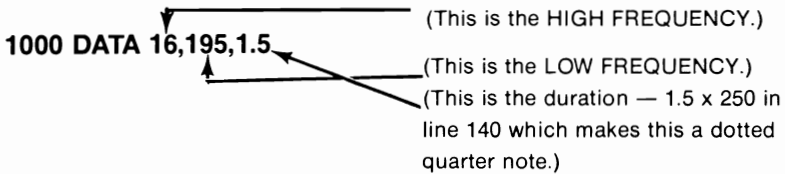
NOTE	NAME	COUNT
	sixteenth	.25
	dotted sixteenth	.375
	eighth	.5
	dotted eighth	.75
	quarter	1
	dotted quarter	1.5
	half	2
	dotted half	3
	whole	4

Enter this program into your computer (Remember that MUSICCODE is lines 5 through 60 so **LOAD** it if you have **SAVED** it.) —

```

5 PRINT "[CLR]"
10 FOR C=54272 TO 54296: POKE C,0: NEXT C
20 L=54272: REM LOW FREQUENCY LOCATION
25 H=L+1: REM HIGH FREQUENCY IS LOW FREQUENCY +1
30 P=L+2: REM LOW PULSE WIDTH IS LOW FREQUENCY +2
35 U=L+3: REM HIGH PULSE WIDTH IS LOW FREQUENCY +3
40 W=L+4: REM WAVEFORM IS LOW FREQUENCY +4
45 A=L+5: REM ATTACK/DECAY IS LOW FREQUENCY +5
50 S=L+6: REM SUSTAIN/RELEASE IS LOW FREQUENCY +6
60 V=54296: REM V STANDS FOR VOLUME
70 POKE V,15
80 POKE A,12: POKE S,4
90 POKE W,0
100 FOR X=1 TO 24
110 READ A,B,C
120 POKE W,33
130 POKE H,A: POKE L,B
140 FOR D=1 TO 250*C: NEXT D
150 POKE W,0
160 NEXT X
170 FOR C=54272 TO 54296: POKE C,0: NEXT C

```



```

1010 DATA 18,209,.5
1020 DATA 16,195,1
1030 DATA 14,239,1
1040 DATA 14,24,1
1050 DATA 14,239,1
1060 DATA 16,195,2
1070 DATA 12,143,1
1080 DATA 14,24,1
1090 DATA 14,239,2
1100 DATA 14,24,1
1110 DATA 14,239,1
1120 DATA 16,195,2
1130 DATA 16,195,1.5
1140 DATA 18,209,.5
1150 DATA 16,195,1

```

1160 DATA 14,239,1
 1170 DATA 14,24,1
 1180 DATA 14,239,1
 1190 DATA 16,195,2
 1200 DATA 12,143,2
 1210 DATA 16,195,2
 1220 DATA 14,24,1
 1230 DATA 11,48,2

RUN

the program.

Recognize the song? If the tempo does not sound right to you, adjust it by changing line 140. Multiplying by a number larger than 250 will slow the music down. Multiplying by a number smaller than 250 will speed up the music.

You could have changed lines 150 like this to speed up the music:

140 FOR D=1 TO 150*C: NEXT D

Giving different values to WAVEFORM, ATTACK/DECAY and SUSTAIN/RELEASE is what produces various musical sounds. Here are some suggestions to begin your orchestra.

	WAVEFORM	ATTACK/ DECAY	SUSTAIN/ RELEASE
Clarinet	17	46	1
Banjo	33	9	0
Accordion	33	102	50

Change these program lines so they read —

80 POKE A,9: POKE S,0
120 POKE W,33

RUN

the program.

There are no right or wrong sounds. You will have to experiment with different note lengths, tempos, WAVEFORMS, ATTACK/DECAY and SUSTAIN/RELEASE settings until you are satisfied with the sound.

Enter this program into your computer —

```

5 PRINT "[CLR]"
10 FOR C=54272 TO 54296: POKE C,0: NEXT C
20 L=54272
25 H=L+1
30 P=L+2
35 U=L+3
40 W=L+4
45 A=L+5
50 S=L+6
60 V=54296
70 POKE V,15
80 DIM K$(7) : DIM H(7) : DIM L(7)
90 POKE P,255: POKE U,33
100 POKE A,9: POKE S,0
110 FOR X=1 TO 7
120 READ K$(X), H(X), L(X)
130 NEXT X
900 PRINT "INPUT MUSICAL NOTES USING LETTERS A
THROUGH G ON THE COMPUTER KEYBOARD."

1000 GET A$: IF A$="" THEN GOTO 1000 (There is no space here.)
1010 IF A$="P" THEN POKE V,0: END
1020 IF A$<"A" THEN GOTO 1000
1030 IF A$>"G" THEN GOTO 1000
1040 POKE W,0
1050 FOR X=1 TO 7
1060 IF K$(X)=A$ THEN POKE W,65: POKE H,H(X) : POKE L,L(X)
1070 NEXT X
1080 GOTO 1000
2000 DATA C,16,195
2010 DATA D,18,209
2020 DATA E,21,31
2030 DATA F,22,96
2040 DATA G,25,30
2050 DATA A,28,49
2060 DATA B,31,165
    
```

RUN

the program and **INPUT** the following letters:

C C G G A A G F F E E D D C

Remember that there are eight parts to making a sound on the 64.

1. CLEAR the sound chip at the beginning of each program.
FOR C=54272 TO 54296: POKE C,0: NEXT C
2. Set the VOLUME between 0 and 15.
POKE 54296,number
3. Set the ATTACK/DECAY between 0 and 255.
POKE 54277,number
4. Set the SUSTAIN/RELEASE between 0 and 255.
POKE 54278,number
5. Set the WAVEFORM.
POKE 54276,number
6. Enter a **FOR-NEXT** loop or use **DATA** and **READ** statements to set the note length.
7. Set the HIGH FREQUENCY.
POKE 54273,number
8. Set the LOW FREQUENCY.
POKE 54272,number

MAKING MUSIC WITH TWO VOICES

Up to now, all of the music you have played has come from voice 1. Voice 2 works in exactly the same way as voice 1, except voice 2 locations begins with 54279 and end at 54285. Here is a chart that shows voice 2 locations:

name	Low Frequency	High Frequency	Low PulseWidth	High PulseWidth	Wave Form	Attack/Decay	Sustain/Release
location	54279	54280	54281	54282	54283	54284	54285

As with voice 1, the chart can be changed so that each location for voice 2 is written as:

name	Low Frequency	High Frequency	Low PulseWidth	High PulseWidth	Wave Form	Attack/Decay	Sustain/Release
location	54279	54279 + 1	54279 + 2	54279 + 3	54279 + 4	54279 + 5	54279 + 6

To use voice 2, assign variable names to each location, just as you did with voice 1. Use the same letters as with voice 1, but add a 2.

LOAD
MUSICCODE

Now add these lines —

```
65 L2=54279: REM LOW FREQUENCY FOR VOICE 2
70 H2=L2+1: P2=L2+2: REM HIGH FREQUENCY, LOW PULSE
  WIDTH FOR VOICE 2
75 U2=L2+3: W2=L2+4: REM HIGH PULSE WIDTH, WAVEFORM
  FOR VOICE 2
80 A2=L2+5: S2=L2+6: REM ATTACK/DECAY, SUSTAIN/RELEASE
  FOR VOICE 2
```

SAVE this new version of MUSICCODE. If you are using disk, write the **SAVE** instruction as follows:

```
SAVE "@0:MUSICCODE",8
```

@0 tells the computer to **SAVE the program on the disk over the old program of the same name.**

If you are using a tape, **SAVE** this program as MUSICCODE2.

Enter this program into your computer (The second version of MUSICCODE is lines 5 through 80.) —

```
5 PRINT "[CLR]"
10 FOR C=54272 TO 54296: POKE C,0: NEXT C
20 L=54272: REM LOW FREQUENCY LOCATION
25 H=L+1: REM HIGH FREQUENCY IS LOW FREQUENCY +1
30 P=L+2: REM LOW PULSE WIDTH IS LOW FREQUENCY +2
35 U=L+3: REM HIGH PULSE WIDTH IS LOW FREQUENCY +3
40 W=L+4: REM WAVEFORM IS LOW FREQUENCY +4
45 A=L+5: REM ATTACK/DECAY IS LOW FREQUENCY +5
50 S=L+6: REM SUSTAIN/RELEASE IS LOW FREQUENCY +6
60 V=54296: REM V STANDS FOR VOLUME
65 L2=54279: REM LOW FREQUENCY FOR VOICE 2
70 H2=L2+1: P2=L2+2: REM HIGH FREQUENCY, LOW PULSE
  WIDTH FOR VOICE 2
75 U2=L2+3: W2=L2+4: REM HIGH PULSE WIDTH, WAVEFORM
  FOR VOICE 2
80 A2=L2+5: S2=L2+6: REM ATTACK/DECAY, SUSTAIN/RELEASE
  FOR VOICE 2
```

```
90 POKE V,15
100 POKE A2,36
110 POKE S2,136
120 POKE W2,17
130 FOR X=1 TO 8
140 READ F,G
150 POKE H2,F: POKE L2,G
160 FOR D=1 TO 300: NEXT D
170 NEXT X
180 FOR C=54272 TO 54296: POKE C,0: NEXT C
1000 REM NOTES FOR SCALE
1010 DATA 22,96
1020 DATA 25,30
1030 DATA 28,49
1040 DATA 29,223
1050 DATA 33,135
1060 DATA 38,126
1070 DATA 43,52
1080 DATA 44,193
```

RUN

the program.

Did this sound any different from voice 1? It should not have. Voice 2 can be programmed to play any sounds in just the same way as voice 1 can.

To hear the sounds of both voices, give each voice different values for ATTACK/DECAY, SUSTAIN/RELEASE and WAVEFORM.

Add these lines to the program —

```
95 POKE A,12
105 POKE S,9
115 POKE W,33
155 POKE H,F: POKE L,G
```

Notice that you did not have to **POKE** another value into the **VOLUME** location. The **VOLUME** for all three voices is controlled by the same location, so all three voices always have the same **VOLUME**.

RUN

the program.

If you cannot hear both voices, it is because they are each playing exactly the same note.

To change the notes that voice 1 is playing, change line 155 so it reads —

155 POKE H,F-2: POKE L,G-5

RUN

the program again. This time you should hear a difference.

TWO-VOICE HARMONY

You can program voice 1 and voice 2 to sound like different musical instruments by assigning different sets of values to ATTACK/DECAY, SUSTAIN/RELEASE, and WAVEFORM. Or you can program voice 1 and voice 2 to play different notes by assigning different HIGH and LOW FREQUENCY values for each voice.

Enter this program into your computer (Lines 5 through 80 are MUSICCODE.) —

```
5 PRINT "[CLR]"
10 FOR C=54272 TO 54296: POKE C,0: NEXT C
20 L=54272: REM LOW FREQUENCY LOCATION
25 H=L+1: REM HIGH FREQUENCY IS LOW FREQUENCY +1
30 P=L+2: REM LOW PULSE WIDTH IS LOW FREQUENCY +2
35 U=L+3: REM HIGH PULSE WIDTH IS LOW FREQUENCY +3
40 W=L+4: REM WAVEFORM IS LOW FREQUENCY +4
45 A=L+5: REM ATTACK/DECAY IS LOW FREQUENCY +5
50 S=L+6: REM SUSTAIN/RELEASE IS LOW FREQUENCY +6
60 V=54296: REM V STANDS FOR VOLUME
65 L2=54279: REM LOW FREQUENCY FOR VOICE 2
70 H2=L2+1: P2=L2+2: REM HIGH FREQUENCY, LOW PULSE
WIDTH FOR VOICE 2
75 U2=L2+3: W2=L2+4: REM HIGH PULSE WIDTH, WAVEFORM
FOR VOICE 2
80 A2=L2+5: S2=L2+6: REM ATTACK/DECAY, SUSTAIN/RELEASE
FOR VOICE 2
90 POKE V,15
100 POKE A,12: POKE S,15
110 POKE A2,88: POKE S2,3
120 READ D
130 IF D<0 THEN GOTO 500 ← (This lets the last data item
(-1) END the program and turn
the VOLUME off.)
140 READ M,K,M2,K2
150 IF M>0 THEN POKE W,17: POKE H,M: POKE L,K
160 IF M2>0 THEN POKE W2,33: POKE H2,M2: POKE L2,K2
170 FOR C=1 TO D*250: NEXT C
```

```

180 FOR C=54272 TO 54296: POKE C,0: NEXT C
190 GOTO 90
500 POKE V,0: END
1000 REM AULD LANG SYNE

```

```

1010 DATA 1,33,135,16,195

```

(This is the duration — 1 X 250.)
(This is the HIGH FREQUENCY, voice 1.)
(This is the LOW FREQUENCY, voice 1.)
(This is the HIGH FREQUENCY, voice 2.)
(This is the LOW FREQUENCY, voice 2.)

```

1020 DATA 1.5,44,193,11,48
1030 DATA .5,42,62,10,143
1040 DATA 1,44,193,9,104
1050 DATA 1,56,99,8,97
1060 DATA 1.5,50,60,7,119
1070 DATA .5,44,193,7,119
1080 DATA 1,50,60,8,97
1090 DATA .5,56,99,8,225
1100 DATA .5,50,60,8,225
1110 DATA 1.5,44,193,9,104
1120 DATA .5,44,193,9,104
1130 DATA 1,56,99,11,48
1140 DATA 1,67,15,11,48
1150 DATA 3,75,69,7,119
1160 DATA 1,75,69,6,167
1170 DATA 1.5,67,15,7,12
1180 DATA .5,56,99,7,12
1190 DATA 1,56,99,11,48
1200 DATA 1,44,193,9,104
1210 DATA 1.5,50,60,12,143
1220 DATA .5,44,193,12,143
1230 DATA 1,50,60,8,97
1240 DATA .5,56,99,8,225
1250 DATA .5,50,60,8,225
1260 DATA 1.5,44,193,9,104
1270 DATA .5,37,162,11,48
1280 DATA 1,37,162,12,143
1290 DATA 1,33,135,8,97
1300 DATA 2,44,193,11,48
9000 DATA -1

```

RUN
the program.

USING VOICE 3

Voice 3 works exactly the same as voice 1 and voice 2. It begins right next to voice 2 in location 54286 and ends at location 54292. Each variable is in the same order as the other two voices, as the chart below shows:

	Low Frequency	High Frequency	Low PulseWidth	High PulseWidth	Wave Form	Attack/Decay	Sustain/Release
name							
location	54286	54287	54288	54289	54290	54291	54292

TRYING YOUR HAND AT PROGRAMMING

21-1 Use the INCHWORM program from lesson 20 and add sound as the worm moves across the screen.

21-2 On the next page is the music for GREENSLEEVES. Program your computer to play this folk song. The letter names of the notes are written above the music.

21-3 Remember the birthday cake you made in lesson 20? Add program lines to it so that the computer will play the song "Happy Birthday" using voice 1.

Then add program lines so that the computer will play the song using 2 voices.

And finally, add program lines for the third voice.

21-4 Listening to music is pleasant but watching a blank screen is boring. Program your computer to play a song and then add some graphics to it.

GREENSLEEVES

The musical score for 'Greensleeves' is written in treble clef, 3/4 time, with a key signature of one sharp (F#). The lyrics are: 'A - las my love you will do me wrong if you cast me off so dis - cour - teous - ly, And I have loved you so ver - y long and de - light - ing in your pleas - ing com - pan - y. Green - sleeves you were all my joy, And you know Green - sleeves you were my de - light, Green - sleeves was my heart of gold and I loved my La - dy Green - sleeves.' The chords are indicated above the notes: B, D, E, F#, G, F#, E, C#, A, B, C#, D, B; B, A#B, C#, A#, F#, B, D, E, F#, G, F#; E, C#, A, B, C#, D, C#, B, A#, G#, A#, B, B, B; A, A, G#, F#, E, C#, A, B, C#, D, B; B, A#B, C#, A#, F#, A, A, G#, F#; E, C#, A, B, C#, D, C#, B, A#, G#, A#, B.

LESSON 22 ASC, CHR\$

INTRODUCTION

ASC and **CHR\$** are two functions which you will use primarily to give special instructions to your peripheral devices (e.g. to produce larger letters with your printer). ASCII stands for American Standard Code for Information Interchange. This code was developed as a universal language so that computers could interface with the various brands of peripheral equipment. After the next lesson, you will use **ASC** and **CHR\$** with **LEFT\$** and **MID\$** to alphabetize lists and design a word game.

Suppose you wanted to program your computer to **PRINT** the whole alphabet. You could start like this:

```
10 PRINT "A"  
20 PRINT "B"  
30 PRINT "C"
```

and then go all the way to:

```
260 PRINT "Z"
```

Or, you could enter this program into your computer —

```
10 PRINT "[CLR]"  
20 FOR N=65 TO 90  
30 PRINT CHR$(N) ← (CHR$ stands for character string.)  
35 FOR D=1 TO 50: NEXT D  
40 NEXT N
```

RUN
the program.

Each keyboard character is given an ASCII number. ASCII stands for American Standard Code for Information Interchange. When two computers "talk" to each other, they are usually using ASCII codes.

To see the ASCII number for each keyboard character, enter this program into your computer —

```

10 PRINT "[CLR]"
20 FOR N=33 TO 127
30 PRINT N, CHR$(N)
40 FOR D=1 TO 500: NEXT D ← (This delay lets you see each
50 NEXT N                    letter and its ASCII number.)
    
```

RUN

the program several times.

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 INPUT "INPUT ANY CHARACTER" ;A$
30 PRINT
40 PRINT "THE ASCII NUMBER FOR "A$" IS"ASC(A$)
50 PRINT: GOTO 20
    
```

RUN

the program and find the ASCII numbers for the following characters.

A Z . 1 * SHFT B CMDR B

The computer will tell you that A=65, Z=90, .=46, 1=49, *=42,
| =194, and **■** =191.

You will notice that the ASCII numbers for the keyboard graphic characters are the same numbers you can use to **POKE** the inverse of the graphic when you design programs using **POKE** statements. (See the chart on page 59.)

In ASCII, the name DUANE would be:

68 85 65 78 69

Use the program to find what your name would be.

Use PRINT CHR\$(number from 33 to 255) to PRINT the character that number goes with.

(e.g. 10 PRINT CHR\$(75) gives you K)

Use PRINT ASC("character") to PRINT that character's ASCII number.

(e.g. 10 PRINT ASC("A") gives you 65)

Can you think of a way to use **ASC**?

Maybe you'll start to get an idea when you enter this program into your computer —

```

10 PRINT "[CLR]"
20 INPUT "INPUT ANY LETTER";A$
30 INPUT "INPUT ANOTHER LETTER";B$
40 PRINT
50 IF ASC(A$)>ASC(B$) THEN GOTO 80
60 PRINT "YOUR FIRST LETTER "A$" COMES BEFORE YOUR
SECOND LETTER "B$"."
70 PRINT: GOTO 20
80 PRINT "YOUR FIRST LETTER "A$" COMES AFTER YOUR
SECOND LETTER "B$"."
90 PRINT: GOTO 20

```

RUN

the program.

One big use for **ASC** is for alphabetizing. To put a list of words or names in alphabetical order, the computer looks at the ASCII numbers of the letters.

If one letter comes before another letter in the alphabet, its ASCII number will be lower.

(e.g. A comes before B ASC("A")=65 ASC("B")=66

F comes before Z ASC("F")=70 ASC("Z")=90)

In the last program you saw how **ASC** can compare letters and determine which one comes first. But, how can **ASC** look at 2 words and say which one would come first in alphabetical order?

To look at words, the alphabetizing program needs the command **LEFT\$** which you'll learn about in the next lesson.

TRYING YOUR HAND AT PROGRAMMING

22-1 Return to the ever popular program — YOUR NAME. Program the computer to display a row of graphic characters across the top and bottom of your screen using **CHR\$**. Still using **CHR\$, PRINT** your name somewhere on the screen.

LESSON 23 LEFT\$, RIGHT\$, MID\$, LEN

INTRODUCTION

*With the information explosion, a central function of computer technology is the organization of data in a way that is helpful to those using it. In this lesson, you will be working with four functions which are crucial to this task of organizing data. **LEFT\$, RIGHT\$, MID\$** and **LEN** are all used in alphabetizing and sorting lists.*

Here are two words: CAT CAR

What do you do to alphabetize them?

1. They both start with C. You can't say which word comes first.
2. The second letter in both is A. You still can't put the words in order.
3. The third letter in CAT is T.
The third letter in CAR is R.
R comes before T.
So CAR comes before CAT.

When you alphabetize, you look at the first letter of the words. If two words have the same first letter, you go to the next letter — until the letters are different.

The computer looks at single letters in a word by using the command **LEFT\$** (pronounced LEFT STRING).

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 INPUT "PLEASE INPUT YOUR FIRST NAME";N$
30 PRINT: PRINT "HERE IS THE FIRST LETTER IN YOUR
NAME:"
40 PRINT LEFT$(N$,1)
```

RUN
the program.

PRINT LEFT\$(N\$,1) means start at the left side of N\$ and PRINT the first letter.

Enter this program into your computer —

```
10 A$="HOT DOG"  
20 PRINT LEFT$(A$,3)
```

RUN

the program.

The computer **PRINTs** the first 3 letters of A\$ which are HOT.

Enter this program into your computer —

```
10 N$="HERBERT"  
20 KN$=LEFT$(N$,4)  
30 PRINT KN$
```

RUN

the program.

This time the computer **PRINTs** the first 4 letters of KN\$ which are HERB.

PRINT LEFT\$(F\$,4) means start at the left side of F\$ and PRINT the first 4 letters.

The following program should **PRINT** the word "DIP". Decide what is missing in this program and then enter it into your computer —

```
10 Z$="DIPPER"  
20 PRINT LEFT$( , )
```

You could have completed the line like this:

```
20 PRINT LEFT$(Z$,3)
```

The following program should **PRINT** the letters "BO". Decide what needs to be added and then enter the program into your computer —

```
10 D$="BONY"  
20
```

Line 20 should read:

```
20 PRINT LEFT$(D$,2)
```

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 INPUT "PLEASE INPUT YOUR FIRST NAME";N$
30 INPUT "PLEASE INPUT A FRIEND'S FIRST NAME";F$
40 IF LEFT$(N$,1)=LEFT$(F$,1) THEN GOTO 100
50 PRINT: PRINT "HERE ARE THE NAMES"
55 PRINT TAB (10) "IN ALPHABETICAL ORDER"
60 IF ASC(LEFT$(N$,1))>ASC(LEFT$(F$,1)) THEN PRINT F$:
PRINT N$: GOTO 80
70 PRINT N$: PRINT F$
80 PRINT: GOTO 20
100 PRINT "YOU NEED TO LEARN ABOUT MID$"
110 PRINT "SO YOU CAN ALPHABETIZE WORDS"
120 PRINT "THAT START WITH THE SAME LETTER."
130 PRINT: GOTO 20
```

RUN

the program.

Line 40 checks to see if the names start with the same letter.

Line 60 looks at the ASCII number of the first letter in each name. Remember, the higher the letter's ASCII number, the later its position in the alphabet.

MID\$

One problem with the last program was that you could not start at the second letter of a word.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 N$="CAT"
30 PRINT MID$(N$,2,1)
```

RUN

the program.

The computer **PRINTs** the letter "A".

PRINT MID\$(N\$,2,1) means start at the second letter of N\$ and PRINT one character.

Change line 30 so it reads —

```
30 PRINT MID$(N$,2,2)
```

RUN

the program.

Now, the computer **PRINTs** "AT".

PRINT MID\$(N\$,2,2) means start at the second letter of N\$ and PRINT the next 2 characters.

```
90 PRINT MID$(A$,11,4)
```

This number tells where to start.

This number tells how many characters to **PRINT**.

(Start at the 11th space.)

(**PRINT** the next 4 characters.)

Change line 30 so that the computer will **PRINT** the "T" from CAT".

You could have changed the line like this:

```
30 PRINT MID$(N$,3,1)
```

Enter this program into your computer —

```
10 A$="NICE PERSON"  
20 PRINT MID$(A$,2)
```

RUN

the program.

With MID\$, if you don't tell the computer how many characters to PRINT, it will PRINT to the end of the string (e.g. PRINT MID\$(A\$,2)).

Look at each of these programs. Decide what the computer will **PRINT**. Then enter them into your computer —

```
10 A$="SUGAR AND SPICE"
20 PRINT MID$(A$,7,3)
```

```
10 A$="SUGAR AND SPICE"
20 PRINT MID$(A$,7)
```

In the first program, the computer will **PRINT** "AND".

In the second program, the computer will **PRINT** "AND SPICE".

You can use MID\$ to look at each of the letters in a string variable — even the first letter.

Look at this program. Decide what is missing and then enter the program into your computer —

```
10 W$="PEANUT BRAIN"
20 PRINT MID$(W$,      ,      )←(This line should PRINT a "P".)
30 PRINT MID$(W$,      ,      )←(This line should PRINT an "E".)
40 PRINT MID$(W$,      ,      )←(This line should PRINT an "A".)
50 PRINT MID$(      )←(This line should PRINT "PEANUT".)
60 PRINT      ←(This line should PRINT "NUT B".)
```

You could have completed the lines like this:

```
20 PRINT MID$(W$,1,1)
30 PRINT MID$(W$,2,1)
40 PRINT MID$(W$,3,1)
50 PRINT MID$(W$,1,6)
60 PRINT MID$(W$,4,5)
```

COMBINING MID\$ WITH ASC

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 INPUT "INPUT YOUR FIRST WORD";F$
30 INPUT "INPUT YOUR SECOND WORD";S$
40 FOR L=1 TO 25 ← (With this line, the computer can handle
                    words up to 25 letters.)
50 IF MID$(F$,L,1)=" " THEN GOTO 100 ← (These lines find the
60 IF MID$(S$,L,1)=" " THEN GOTO 200 ← end of F$ or S$.)
70 IF ASC(MID$(F$,L,1))<ASC(MID$(S$,L,1)) THEN GOTO 100
80 IF ASC(MID$(F$,L,1))>ASC(MID$(S$,L,1)) THEN GOTO 200
90 NEXT L
100 PRINT: PRINT "HERE ARE THE WORDS IN ALPHA-
    BETICAL ORDER"
110 PRINT F$: PRINT S$
120 PRINT: GOTO 20
200 PRINT: PRINT "HERE ARE THE WORDS IN ALPHA-
    BETICAL ORDER "
210 PRINT S$: PRINT F$
220 PRINT: GOTO 20
    
```

RUN

the program.

Look at lines 70 and 80. These two lines compare the ASCII number of the current letters.

In alphabetizing, CAT comes before CATS, BEE comes before BEEP, and so on. Lines 50 and 60 in the program above would make sure that CAT comes before CATS.

THE COMPUTER'S BUILT-IN ALPHABETIZING FUNCTION

Do you understand how to alphabetize using **MID\$** and **ASC** in a **FOR-NEXT** loop?

Your computer knows how to alphabetize, too! You can get it to alphabetize for you by using the symbols **>** and **<**.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 INPUT "INPUT YOUR FIRST WORD";F$
30 INPUT "INPUT YOUR SECOND WORD";S$
40 PRINT: PRINT "HERE ARE THE WORDS IN
ALPHABETICAL ORDER"
50 IF F$>S$ THEN GOTO 80
60 PRINT F$: PRINT S$
70 PRINT: GOTO 20
80 PRINT S$: PRINT F$
90 PRINT: GOTO 20
```

RUN

the program.

Line 50 uses your computer's built in ability to alphabetize. You know the method the computer uses — you've done it yourself! The computer simply compares the ASCII numbers of each letter of the words until it finds one that is different. From now on, you can use > and < to have the computer alphabetize your words. But it's good that you also know how to alphabetize them yourself by using **ASC** and **MID\$**.

ALPHABETIZING MORE THAN 3 WORDS

Comparing 2 or 3 words to alphabetize isn't too tough, but to alphabetize many words you need to set up an array.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 PRINT "THIS PROGRAM WILL ALPHABETIZE YOUR LIST
OF NAMES."
30 PRINT
40 INPUT "HOW MANY NAMES ON YOUR LIST";N
50 DIM A$(N)
60 PRINT "PLEASE INPUT THE NAMES, LAST NAME FIRST"
70 PRINT "(LIKE THIS: SMITH JOHN)"
80 FOR X=1 TO N
90 PRINT "NAME"X;
100 INPUT A$(X)
110 NEXT X
```


Now, you need to rearrange the elements of the array A\$ until they are in alphabetical order. This part of the program uses a method called the Shell-Metzner sort. The program first compares A\$(1) with A\$(N/2+1), A\$(2) with A\$(N/2+2), and so on. Then it goes through the array again in smaller steps until it finally compares adjacent elements.

Add these lines to the program —

```
120 M=N
130 M=INT(M/2)
140 IF M=0 THEN GOTO 280
150 K=N-M
160 J=1
170 X=J
180 L=X+M
190 IF A$(X)<=A$(L) THEN GOTO 250
200 T$=A$(X)
210 A$(X)=A$(L)
220 A$(L)=T$
```

Line 190 compares A\$(X) with A\$(L). If they are in alphabetical order, the computer goes to line 250. If they are not in order, lines 200 through 220 switch their positions in the array.

Continue adding these lines to the program —

```
230 X=X-M
240 IF X>= 1 THEN GOTO 180
250 J=J+1
260 IF J>K THEN GOTO 130
270 GOTO 170
```

And, finally add these lines to **PRINT** the items in alphabetical order —

```
280 FOR X=1 TO N
290 PRINT A$(X)
300 NEXT X
310 PRINT: CLR: GOTO 20 ←(CLR is needed so you can redimension
the array in line 50 when the program is
RUN again.)
```

RUN
the program.

RIGHT\$

You've seen **LEFT\$** and **MID\$**. Can you guess what **RIGHT\$** does?

Enter this program into your computer —

```
10 G$="COLD TURKEY"
20 PRINT RIGHT$(G$)
```

RUN

the program.

PRINT RIGHT\$ (G\$,8) means start at the end of G\$ and PRINT the last 8 characters.

RIGHT\$ can be used to "strip off" parts of an **INPUT** response, depending on what form you need.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 PRINT "PLEASE INPUT YOUR NAME LIKE THIS:"
30 PRINT: PRINT TAB(15) "JOHN JONES"
40 INPUT N$
50 PRINT: INPUT "ARE YOU 1) MALE OR 2) FEMALE";S
60 FOR N=1 TO 50
70 IF MID$(N$,N,1)=" " THEN GOTO 90
80 NEXT N
90 IF S=1 THEN PRINT: PRINT "WELL MR.";
100 IF S=2 THEN PRINT: PRINT "WELL MS.";
110 LET L=LEN(N$)-N
120 PRINT RIGHT$(N$,L);
130 PRINT ", I HOPE THIS PROGRAM HELPS YOU SEE HOW
TO USE RIGHT$."
```

(Be sure there is one space here.)

(The command **LEN** tells you the length of **N\$**.)

RUN

the program.

Look at the last program.

Lines 60, 70 and 80 go through the name, a character at a time, starting with the first letter. When the computer finds the space between the first and second name, it goes to line 90.

Lines 90 and 100 either **PRINT** "MR." or "MS." depending on the answer from the **INPUT** in line 50.

In line 110, the length of the first name plus the space afterwards equals N. (See lines 60 through 80.) Line 110 sets L equal to the length of both names (N\$) minus N. This actually equals the length of the last name.

Line 120 **PRINTs** the last name.

RUN

the program again and think about how the computer is working.

LEN

Look at this program and decide what the computer will **PRINT**.

```
10 A$="HORSE"  
20 PRINT LEN(A$)
```

Now, enter the program into your computer and see if you were right.

The computer will **PRINT** the number 5 because there are 5 letters in A\$.

What do you think the computer will **PRINT** using this program?

```
10 A$="HORSE AND BUGGY"  
20 PRINT LEN(A$)
```

The computer will **PRINT** the number 15 because A\$ has 15 letters and spaces.

L=LEN(X\$) means let L equal the number of characters in X\$.

Here is another program which uses **LEN**.

Enter this program into your computer —

```
10 PRINT "[CLR]"  
20 PRINT: PRINT "GIVE ME A WORD."  
30 INPUT W$  
40 L=LEN(W$)
```

```
50 FOR X=1 TO L
60 L$=MID$(W$,X,1)
70 IF L$="A" OR L$="E" OR L$="I" OR L$="O" OR L$="U" THEN
GOTO 90
80 NEXT X
90 P=X-1
100 PL$=RIGHT$(W$,L-P) + LEFT$(W$,P) + "AY"
110 IF X=1 THEN PL$=W$ + "LAY"
120 PRINT: PRINT TAB(5) PL$
130 GOTO 20
```

RUN

the program.

Your computer now speaks PIG LATIN!

In lines 100 and 110 you added string variables together to form a new string variable. You can join together parts of string variables (as you did in line 100) or entire strings (e.g. line 110) to develop a new string variable up to 255 characters.

Adding string variables is called concatenation.

TRYING YOUR HAND AT PROGRAMMING

23-1 Write a program that will allow you to **INPUT** a list of 10 names. Use an array to store the names. After the computer has all the names, have it ask you what letter you would like to see. Then, use **ASC** and **LEFT\$** to have the computer show you all the names that start with that letter.

23-2 Write a computer word game for two people to play. Allow a player to **INPUT** a word and have the computer convert the letters in the word to ASCII numbers and add those numbers to produce a score. The person with the highest score wins the game.

Your program could:

Allow player #1 to tell the computer how many letters in the word.

Use this number to **DIM** 2 arrays; one for the word, one for the ASCII conversion.

Allow player #1 to **INPUT** his/her word.

Use a **FOR-NEXT** loop and **MID\$** to put each individual letter into an array so it can be changed to the ASCII number.

Use another **FOR-NEXT** loop to convert the letters in the array to ASCII numbers.

Add the ASCII numbers to get a score.

PRINT that score.

Repeat the procedures for player #2.

Use **IF-THEN** statements to compare player #1's score with player #2's score and **PRINT** a message telling who won.

Use a **CLR** statement and **GOTO** at the beginning of the program so the players can play another round.

23-3 Write a program that an engraver (a person who puts letters on metal plaques) might use. Have your program ask "PLEASE INPUT WHAT YOU WANT ENGRAVED." Then use **LEN** to find out the length of the message. Use **MID\$** to figure out the number of blank spaces. Then tell the user "THERE ARE _____ LETTERS IN YOUR MESSAGE. THAT WILL COST _____." Charge 25¢ for each letter. (No charge for blank spaces.)

LESSON 24 VAL, STR\$

INTRODUCTION

*Your computer follows its rules very strictly. It cannot perform math operations on a string variable or use a string operation with a number. This becomes a problem, for example, in a home finance program when you want to subtract 50 cents from 10 dollars. The words "cents" and "dollars" make these amounts string variables. In this lesson, you will see how the functions, **VAL** and **STR\$** overcome this problem.*

Here is an **INPUT** question:

HOW MUCH ALLOWANCE DO YOU GET?

Here are some possible answers:

25 300 DOLLARS \$1.50

The problem is that you can't do much with those answers because they are each in a different form. You need to change those answers to plain numbers with **VAL**.

Enter this program into your computer —

```
10 PRINT "[CLR]"
20 INPUT "PLEASE INPUT SOME AMOUNT OF MONEY";M$
30 PRINT
40 PRINT "THE VAL OF THAT STRING IS";
50 PRINT VAL(M$)
60 PRINT: GOTO 20
```

RUN

the program and **INPUT** these amounts:

5 \$1.00 1 DOLLAR SIX DOLLARS FIVE 999999

A string variable does not have a number value. String variables are equal to actual keys, not their value. So when you **INPUT** \$1.00 the computer does not understand this as a numeric value and so cannot perform any arithmetic calculations with it. **VAL** turns a string variable into a number.

VAL(N\$) looks at N\$.

If the first character is a letter, the VALue is A.

If the first character is a number(s), the VALue is that number.

If words come after the number, the computer ignores the words and just uses the number.

Enter this program into your computer —

```
10 A$="100 BUCKS"  
20 B$="10 DOLLARS"  
30 N1=VAL(A$) : N2=VAL(B$)  
40 PRINT N1 + N2  
50 PRINT N1/N2
```

RUN

the program.

Using **VAL**, the computer will ignore the words "BUCKS" and "DOLLARS". Line 30 will make N1=100 and N2=10. This allows the computer to use those values for math calculations.

VAL changes a string into a number so you can perform mathematical operations (+, -, *, /, etc.).

Enter this program into your computer —

```
10 PRINT "[CLR]"  
20 PRINT "HOW MUCH ALLOWANCE DO YOU GET EACH  
WEEK?"  
30 INPUT A$  
40 IF LEFT$(A$,1)="$" THEN LET A$=MID$(A$,2)  
50 PRINT  
60 LET V=VAL(A$)  
70 IF RIGHT$(A$,5)="CENTS" THEN LET V=V/100
```

```

80 PRINT "IN ONE YEAR YOU GET"V*52"DOLLARS."
90 PRINT: GOTO 20

```

RUN

the program and try answering with these:

```

$4.75      300 DOLLARS      25 CENTS

```

LIST

the program and take a good look at it.

Line 40 strips off the \$ if there is one.

Line 70 makes an adjustment in V if A\$ was in cents, not dollars.

To avoid problems, have your **INPUT** question ask for an answer in a specific form. (e.g. How many dollars do you get each week as an allowance? \$") You need to be aware of all the possible answers and design your program so it can accept them.

STR\$

STR\$ is pronounced S-T-R STRING. It is the opposite of **VAL**.

What if you wanted to **INPUT** a 3-digit number and then look at each digit separately? You need to change the number into a string.

Enter this program into your computer —

```

10 PRINT "[CLR]"
20 PRINT: PRINT: PRINT "INPUT ANY NUMBER."
30 PRINT TAB(5) "I WILL REVERSE THAT NUMBER."
40 INPUT N
50 A$=STR$(N)
60 L=LEN(A$)
70 FOR X=L TO 1 STEP -1
80 B$=B$+MID$(A$,X,1)
90 NEXT X
100 PRINT: PRINT "HERE IS YOUR NUMBER BACKWARDS."
110 PRINT TAB(8) B$
115 CLR
120 GOTO 20

```

RUN

the program.

STR\$ changes a number value into a string so you can perform string operations (LEFT\$, MID\$, etc.).

LINING UP DECIMAL POINTS

TAB will line up columns on the left quite nicely. This is fine for string data, but what happens when you try to put numbers in a column?

Enter this program into your computer —

```
10 DATA 12, -1113.4, 96.87, 124.444, 2
20 FOR X=1 TO 5
30 READ N
40 PRINT TAB(10)N
50 NEXT X
```

RUN

the program.

What happens? The numbers look strange because numbers are usually lined up according to the decimal point.

To do this, you want to decide the location of the decimal point and the length of the integer portion of the number.

Enter this program into your computer —

```
10 DATA 12, -1113.4, 96.87, 124.444, 2
20 FOR X=1 TO 5
30 READ N
40 B=INT(N)
50 L=LEN(STR$(B))
60 PRINT TAB(10-L) N
70 NEXT X
```

RUN

the program.

TRYING YOUR HAND AT PROGRAMMING

24-1 Design a program for currency exchange of pesos and francs to American dollars. The rate of exchange varies daily.

Here is an approximate exchange rate:

150 pesos = 1 dollar

795 francs = 1 dollar

24-2 Write a program which asks someone to **INPUT** a number. Have the computer reverse that number. Then **PRINT** the original number "+" the reversed number "=" the total of the two numbers.

REVIEW 6

Choose a word which will match each definition.

ASCII	PRINT MID\$(A\$,2,2)
ATTACK/DECAY	RIGHT\$
CHR\$	STR\$
LEN(A\$)	SUSTAIN/RELEASE
POKE 54296,15	VAL
PRINT LEFT\$(A\$,1)	WAVEFORM

1. _____ sets the speed at which the sound reaches its highest volume and falls to its sustained volume.
2. _____ tells the computer how loud the note will stay until turned off and determines how quickly the note will fall to zero volume.
3. _____ sets the "shape" of the sound and can be set at 17, 33, 65 or 129.
4. _____ would set the volume of the SID to its highest level.
5. _____ stands for American Standard Code for Information Interchange. Each keyboard character is assigned one of these numbers.
6. _____ is the command to use with a character's ASCII number to print that character.
7. _____ is the command used to print the first character in A\$.
8. _____ is the command that would print the second and third characters in A\$.
9. _____ is the command that starts at the last character of a string and works toward the first.
10. _____ is the command that would tell the number of characters in A\$.
11. _____ is the command used to change a string variable into a number value.
12. _____ is the command used to change a number value into a string.

LESSON 25

PERIPHERALS:

ADDING TO YOUR COMPUTER SYSTEM

INTRODUCTION

When home computers first became available, there was a limited amount of equipment for them. You could get a computer, a monitor, a printer, a disk drive, or a tape recorder. This has changed dramatically as the home market has expanded. To make the computer more accessible, input devices have been developed to free you from the keyboard, at least part of the time.

JOYSTICKS

Joysticks, adapted from video games, were one of the first such devices. Instead of typing in requests, you move the joystick to position the cursor over the desired choice. A button is pushed and the computer acts. Joysticks are most often used with games, but are also used in educational software. Their biggest drawback is that they require some practice to be able to use with precision.

GRAPHICS TABLETS

One input device that is becoming very popular is the graphics tablet. This is usually a small pad with a plastic stylus or pen for drawing. The tablet is connected to the computer through the game ports. Whenever the stylus is moved across the tablet, a corresponding movement is made on the screen. Colors, patterns, and shapes may be displayed, changed, or added by choosing the appropriate options with the stylus. The KOALA PAD® and POWERPAD® are two of the most versatile graphics tablets currently available for your computer.

MOUSE

The mouse is a new peripheral that has captured a lot of interest. It is a small electronic device on wheels. Rolling it around causes the cursor on the screen to move. When the cursor is over the desired function, you push a button and the computer acts. For many people it is somewhat more precise and easier to use than the joystick.

TOUCHSCREEN

Another innovation designed to make the computer easier to use, is the touch sensitive screen. A screen of this type has an electronic grid overlaying the monitor. When a place on the screen is touched, a signal is sent from the grid, telling the computer what to do. For many people, this is the easiest way to interact with the computer.

SPEECH SYNTHESIZER

Do you want to add speech to your computer's capabilities? Some manufacturers now have devices that can be programmed to play pre-recorded sounds and words. Another speech device that has had a lot of publicity is the voice recognition box. This actually allows you to tell the computer what to do verbally. If you wish to have the cursor move to the left, instead of pushing a joystick or a mouse to the left, you can speak the word, "left," and the cursor will move left.

All of these input devices require software that has been specifically designed for them. The computer will do nothing with any of them unless the software is there to tell it what to do. Usually, software is supplied with each device, but this is not true in all cases. Check with your dealer. Also, check to see what other types of software may be available that could use the same piece of equipment. The KOALA PAD, for example, comes with its own software, but it is also compatible with some of the games that are produced by Electronics Arts. Because new hardware developments occur very rapidly, most popular computing magazines have a review section where new peripherals are critiqued.

MODEMS

Much has been written about modems and the telecommunications revolution. Telecommunication refers to the direct transmission of data from one computer to another, usually using telephone lines.

PARTS OF A TELECOMPUTING SYSTEM

The modem is the device that allows the computer to send and receive the data from another computer. Modem itself stands for MODulator-DEModulator. The demodulator takes the sounds coming from the telephone and translates them to computer signals. The modulator breaks the signals down and sends them out to the telephone lines.

The Commodore has a built-in slot in back that allows some types of modems to be connected directly to it. Other types need an interface (which will be an added expense) that will plug into that slot. Make sure that the modem and the necessary software are compatible with your computer. At present, there are over 150 modems on the market, priced from \$50.00 to over \$1500. Modems produced by Hayes are some of the most popular. This line works with most computers and has a good reputation for reliability. In shopping for a modem, be certain that it has the features you require.

A crucial piece of equipment is your telephone. Be sure that it is compatible with your modem. It needs to be pulse-activated, that is, it must have push buttons that produce a musical tone when pressed. It must also have a modular connector between the ear piece and the base. That modular connector will be placed in the modem when you want to use it.

Having a modem and a compatible telephone means you have the beginnings of your telecommunication system. But a modem is useless without software. In fact, the two, hardware and software, are often sold as one package.

Finally you may want to subscribe to an information service. These are national services that operate from a centralized location utilizing a mainframe computer. They charge a subscription fee plus an access fee, but provide an almost endless list of services. Up-to-the minute stock reports, weather, sports news, airline reservations, theater and film reviews, even an entire encyclopedia, can be accessed through these services. In addition, they have special interest group (SIG) listings whereby you can share information with other people who share a common interest. Some of the nation-wide information services are CompuServe, The Source, Telenet, Dow-Jones, Delphi.

A FINAL WORD ABOUT MODEMS

In some states, using a modem takes your telephone out of the residential service rates. The telephone company tariffs consider all uses of telecommunications to be business-related and charge accordingly. These tariffs were written in the 1960's when home computers were virtually unknown and when the use of telecommunications required special, distortion-free transmission lines. These tariffs are currently undergoing consideration for revision, so check with the phone company to find out the status of phone rates for home use of a modem.

Because of the variety of modems available you need to know what features of the modem are going to be important for your use. Here is a listing of some of the features of the modem and their meaning:

BAUD RATE: the rate at which signals are sent. In the home market right now, modems range between 300 and 1200 bauds. A baud rate of 300 is standard for most home uses and most home terminals. A faster rate may save you time and money because of the lower amount of time on the telephone and using your information service. (Remember, information services charge an hourly rate.) A 1200 baud modem, however, is more expensive.

UPLOAD: the ability to transmit data from either the computer's memory or from a disk or tape file.

DOWNLOAD: the ability to receive data and have it stored either in a disk file, a tape file, or on the printer. Both **DOWNLOAD** and **UPLOAD** are crucial features if you are using an information service.

BULLETIN BOARD: a computer service that can be accessed by anyone with a modem. Some of them charge a fee, some require a password, but almost all are designed for a special interest group. Teachers of special education have a bulletin board; so do people interested in adventure games. Functioning like a club or a central clearinghouse, some are local, others are nation-wide. If you have something to say, a question, or some information you can leave a message on the appropriate bulletin board. Another user, interested in your message, will respond with a message via modem.

AUTODIAL/REDIAL: the ability of the modem to dial itself, using signals from the computer. The more primitive modems require you to push the buttons on the telephone, then when the number starts to

ring, to quickly insert the connector into the modem. With autodial, you plug in the modem then send the number to it, either directly by typing it in, or by having it read from a program. AUTODIAL and REDIAL are important features if you are going to use a Bulletin Board or Information Service. Generally you have to wait for a free line to access a Bulletin Board.

AUTO-ANSWER: this allows the modem to take a call when you are not at the modem. This is another feature which will be really important if you belong to a Bulletin Board.

E-MAIL: this is very similar to having your own mail slot on a Bulletin Board. When someone wants to leave a message for your eyes only, it can be left at your E-MAIL designation. No other user of the Bulletin Board, except possibly the "sys op", will be able to read this message. Large information services, such as Compuserve and the Source, also have E-MAIL capabilities.

SYS OP: short form of "system operator", the sys op is the person who sets up and maintains the bulletin board service.

PRINTERS

Having a printer for your computer is not essential, but it can make using your computer easier. If you are writing a long program, the entire listing may not fit on the screen. A printout of the listing will help you to see how the whole program fits together. A printer allows a permanent record to be made of the work the computer has done, whether it is printing out a listing, a file, or a graphics design. If you want to use your computer for word processing, a printer is essential.

TYPES OF PRINTERS

Printers are available for your computer in a variety of formats. The most popular is the dot-matrix printer, which produces characters as combinations of dots. The print head consists of pins arranged in a rectangular pattern. Depending on the pattern to be printed, more or less of the pins are pressed against the ribbon. Whenever you see a letter that looks as if a computer produced it, it most likely was produced by a computer using a dot matrix printer.

The dot matrix is the fastest type of printer available for home computers. It is the most versatile, being able to print graphics, enlarged and condensed characters, and super- and sub-scripts. The

biggest drawback that a dot-matrix printer has is that it produces copy that looks as if a computer has typed it. For this reason, people who need a more professional look will buy a type of printer called a "daisy-wheel" or "letter-quality" printer. A letter-quality printer uses formed letters, much as a typewriter does, to produce copy that is indistinguishable from personally typed output. A daisy-wheel printer is more expensive, slower and less versatile than a dot-matrix printer. If the way the final output looks is of paramount importance, then you should consider a daisy-wheel printer. Some of the better dot-matrix printers produce copy that is very close to letter quality. These are called "correspondence-quality" or "pseudo-letter" printers. What follows is a description of some features of the dot-matrix printer since this printer is the most popular one for the home market.

WHAT TO LOOK FOR WHEN BUYING A DOT-MATRIX PRINTER

There are several considerations in buying a dot-matrix printer. The number of pins can vary, and generally, the more pins, the higher the quality of print. The Commodore printer usually sold with the 64, is the VIC-1525. It has pins arranged to produce a 6x7 dot matrix (6 dots wide by 7 dots high) which means that there is some space visible between the dots. Some printers have pins as closely spaced as a 9x9 matrix in the same area as the Commodore's 7x7.

HOW IT PRINTS LOWER CASE LETTERS

A good printer has the ability to print descenders (the part of a lower-case letter that goes below the body). In order to fit the tail in lower case letters, such as p or g, some printers move the entire letter higher than usual, so that a tail of sorts is printed. This may result in the copy that does not meet your requirements.

UNDERLINING WORDS

Some printers can underline or overstrike a word for added emphasis. The ability to do this increases the flexibility, as well as the cost, of the printer.

SPEED OF THE PRINTER

The speed of dot-matrix printers vary. Some will print as fast as 200 characters per second (CPS); others will poke along at 30 CPS. Part of the reason for the variation is the manner in which the printer works. In

the more efficient models; the printer works bi-directionally. That is, it will print one line left to right, then print the next line right to left. This eliminates the time to reposition the print head at the left margin all of the time. Part of the speed is also a result of the design of the printer.

PAPER FEED

Paper may be fed to printers in one of two ways; friction feed or tractor feed. Friction feed is the way a typewriter moves paper. The paper is held by a platen and, as the platen turns the paper is moved. Each sheet must be hand fed in and must be lined up manually. The second type of feed is tractor feed. To use tractor feed, a special paper is needed. The paper has a track down each side with holes in it. Each sheet is attached in a continuous form that is fed through the printer. Many printers are able to accept both paper-feeding methods.

PRINTING GRAPHICS

A dot-matrix printer can usually print any graphics representation that is on the screen, but will usually need some software help. The Commodore printer can print the entire Commodore graphics set, but it needs a program to print what is shown on the screen.

SOME FINAL CONSIDERATIONS

Other features to look for in a dot matrix printer include controls for line feed and form feed, graphics capabilities, and interface methods.

The line feed control allows the paper to be advanced one line at a time, merely by pushing a button. This is convenient when you are trying to line up a form. The form feed control will advance the paper one whole page. The length of the page is usually assumed to be eleven inches so that it is how far the paper moves.

Many printers cannot be connected directly to your computer. An interface unit, which is usually an additional purchase, is necessary. Some of the interfaces available will automatically translate graphics to the printer, so that any graphics can be printed. There is a price in speed to pay. The translation will slow the printer down considerably. The VIC 1525 printer does not need an interface, but it prints everything, graphics or text, slowly.

A printer is a sizable investment. Be sure to check carefully for all of the features that you feel are important for the uses you have in mind.

APPENDIX ANSWERS TO REVIEWS

REVIEW 4

1. RND
2. INT
3. PRINT RND(1)
4. PRINT INT(RND(1)*24)+1
5. ABS
6. CLR
7. array
8. Element
9. Subscript
10. DIM

REVIEW 5

1. READ
2. DATA
3. flag
4. RESTORE
5. multi-dimensional array
6. ON GOTO
7. ON GOSUB
8. POKE
9. PEEK
10. POKE 55296,2

REVIEW 6

1. ATTACK/DECAY
2. SUSTAIN/RELEASE
3. WAVEFORM
4. POKE 54296,15
5. ASCII
6. CHR\$
7. PRINT LEFT\$(A\$,1)
8. PRINT MID\$(A\$,2,2)
9. RIGHT\$
10. LEN(A\$)
11. VAL
12. STR\$

SOLUTIONS FOR 'TRYING YOUR HAND AT PROGRAMMING'

13-1 These changes will have the addition quiz program choose numbers between 1 and 100, inclusive:

```
160 A=INT(RND(1)*100)+1
170 B=INT(RND(1)*100)+1
```

These changes will make the addition quiz a multiplication quiz:

These changes will make the addition quiz a multiplication quiz?

```
160 A=INT(RND(1)*12)+1 : REM NUMBERS BETWEEN 1 AND 12
170 B=INT(RND(1)*12)+1 : REM NUMBERS BETWEEN 1 AND 12
180 PRINT: PRINT A "X" B "=" : INPUT N
190 IF N=A*B THEN GOSUB 2000
200 IF N<>A*B THEN PRINT "THAT IS INCORRECT! THE ANSWER IS"A*B".": GOSUB 3000
```

13-2

```
5 REM ADDS POSITIVE AND/OR NEGATIVE NUMBERS
10 PRINT "[CLR]"
20 PRINT "WHEN YOU SEE A ? TYPE A NUMBER."
30 PRINT: PRINT "IT CAN BE EITHER POSITIVE OR NEGATIVE."
40 PRINT
50 FOR A=1 TO 10
60 INPUT X
70 IF ABS(X)=X THEN GOSUB 500
80 IF ABS(X)<>X THEN GOSUB 600
90 NEXT A
100 PRINT: PRINT "YOU ENTERED"P"POSITIVE NUMBERS."
110 PRINT: PRINT "THE TOTAL OF THE POSITIVE NUMBERS IS"S
120 PRINT: PRINT "YOU ENTERED"N"NEGATIVE NUMBERS."
130 PRINT: PRINT "THE TOTAL OF THE NEGATIVE NUMBERS IS "T
140 END
500 REM P STANDS FOR POSITIVE NUMBERS, S THEIR SUM
510 P=P+1
520 S=S+X
530 RETURN
600 REM N STANDS FOR NEGATIVE NUMBERS, T THEIR TOTAL
610 N=N+1
620 T=T+X
630 RETURN
```

13-3

```

5 REM COMPUTER'S GAME OF 21
10 PRINT "[CLR]"
20 PRINT TAB(3)"THE COMPUTER WILL GIVE YOU RANDOM"
30 PRINT "NUMBERS BETWEEN 1 AND 10"
40 PRINT: PRINT " TAB(3) "YOU CAN TAKE AS MANY NUMBERS AS"
50 PRINT "YOU WISH."
60 PRINT: PRINT: PRINT
70 PRINT TAB(3)"THE COMPUTER WILL ALWAYS CHOOSE 3"
80 PRINT "RANDOM NUMBERS FOR ITSELF."
90 PRINT: PRINT
100 PRINT TAB(3)"THE WINNER IS WHOEVER IS CLOSEST"
110 PRINT "TO 21 WITHOUT GOING OVER."
120 FOR D=1 TO 3000: NEXT D
130 PRINT "HERE IS YOUR FIRST NUMBER."
140 GOSUB 1000
150 PRINT: PRINT TAB(15)N
160 PRINT: PRINT "DO YOU WANT ANOTHER NUMBER"
170 INPUT "(Y OR N)":Z$
180 IF Z$="Y" THEN GOSUB 1000
190 IF Z$="N" THEN GOTO 240
200 IF Z$<<"Y" AND Z$<<"N" THEN GOTO 170
210 PRINT: PRINT "HERE IS YOUR NEXT NUMBER:"
220 PRINT: PRINT TAB(15)N
230 PRINT: PRINT "YOUR TOTAL IS"T: GOTO 160
240 FOR X=1 TO 3
250 NN=INT(RND(1)*10)+1
260 CT=CT+NN
270 NEXT X
280 PRINT: PRINT "THE COMPUTER'S TOTAL IS"CT
290 IF T>CT AND T<=21 THEN PRINT "YOU BEAT THE COMPUTER!"
300 IF CT>T AND CT<=21 THEN PRINT "THE COMPUTER WINS!!"
310 IF CT>21 AND T>21 THEN PRINT "YOU BOTH LOSE."
320 IF CT=T THEN PRINT "YOU BOTH HAVE THE SAME NUMBER."
330 IF T>21 AND CT<=21 THEN PRINT "THE COMPUTER IS THE WINNER!"
340 IF CT>21 AND T<=21 THEN PRINT "YOU ARE THE WINNER!"
350 PRINT: PRINT "PRESS 'G' IF YOU WANT TO PLAY AGAIN."
360 PRINT "PRESS 'S' IF YOU WANT TO STOP."
370 GET R$
380 IF R$="G" THEN CLR: PRINT "[CLR]": GOTO 130
390 IF R$="S" THEN PRINT: PRINT "HOPE YOU HAD FUN.": END
400 IF R$<<"G" AND R$<<"S" THEN GOTO 370

1000 N=INT(RND(1)*10)+1
1010 T=T+N
1020 RETURN
    
```

14-1

```
5 REM TELEVISION SHOW ARRAY
10 PRINT "[CLR]"
20 PRINT TAB(5)"PLEASE INPUT 8 OF YOUR FAVORITE"
30 PRINT TAB(5)"TELEVISION SHOWS."
40 DIM TV$(8)
50 FOR X=1 TO 8
60 INPUT TV$(X)
70 NEXT X
80 PRINT "[CLR]"
90 PRINT "HERE ARE YOUR FAVORITE TV SHOWS:"
100 PRINT
110 FOR X=1 TO 8
120 PRINT "#"X; TV$(X)
130 NEXT X
140 PRINT
150 PRINT "WOULD YOU LIKE TO CHANGE ONE (Y OR N)?"
160 GET A$
170 IF A$="Y" THEN GOTO 200
180 IF A$="N" THEN END
190 IF A$<>"Y" OR A$<>"N" THEN GOTO 160
200 INPUT "WHICH NUMBER DO YOU WANT TO CHANGE?"
210 PRINT "WHAT WOULD YOU LIKE TO CHANGE IT TO"
220 INPUT N$
230 TV$(N)=N$
240 PRINT: GOTO 80
```

14-2

```
5 REM NAME AND TELEPHONE NUMBER ARRAYS
10 PRINT "[CLR]"
20 DIM N$(5)
30 DIM P$(5)
40 PRINT "THIS PROGRAM WILL ASK FOR THE NAMES AND"
50 PRINT "PHONE NUMBERS OF FIVE PEOPLE"
60 PRINT: PRINT
70 FOR X=1 TO 5
80 PRINT "PLEASE INPUT A PERSON'S NAME."
90 INPUT N$(X)
100 PRINT "PLEASE INPUT " N$(X) " 'S TELEPHONE NUMBER."
110 INPUT P$(X)
120 PRINT
130 NEXT X
140 PRINT "[CLR]"
150 PRINT "HERE IS YOUR LIST."
160 PRINT
170 FOR X=1 TO 5
180 PRINT "NAME: "; N$(X)
190 PRINT "PHONE NUMBER: "; P$(X)
200 PRINT
210 NEXT X
220 END
```

14-3

```
5 REM MAKING A GUEST LIST
10 PRINT "[CLR]"
20 PRINT "HOW MANY GUESTS DO YOU WANT TO INVITE"
30 INPUT "TO YOUR PARTY";A
40 G=A-1: REM SUBTRACT 1 BECAUSE THIS ARRAY BEGINS WITH BOX "0"
50 PRINT: PRINT
60 PRINT "INPUT THE NAMES OF YOUR GUESTS."
70 DIM P$(G)
80 FOR N=0 TO G
90 INPUT P$(N)
100 NEXT N
110 PRINT "[CLR]"
120 PRINT TAB(240)
130 FOR T=0 TO G
140 PRINT P$(T)
150 NEXT T
160 PRINT: PRINT "DID YOU MAKE ENOUGH POTATO SALAD?"
170 END
```

14-4

```
5 REM SUMMER EARNINGS ARRAY
10 PRINT "[CLR]"
20 DIM E(12)
30 PRINT "ENTER YOUR WEEKLY EARNINGS FOR A"
40 PRINT "12-WEEK PERIOD. DO NOT USE A $."
50 PRINT
60 FOR W=1 TO 12
70 PRINT "WEEK #":W: INPUT E(W)
80 T=T+E(W): REM TOTAL EARNINGS
90 NEXT W
100 PRINT "[CLR]"
110 PRINT "YOUR EARNINGS FOR THE 12-WEEK PERIOD:"
120 PRINT
130 FOR W=1 TO 12
140 PRINT E(W),
150 NEXT W
160 PRINT
170 PRINT "YOUR TOTAL EARNINGS WERE $"T
180 A=T/12: REM AVERAGE EARNINGS
190 PRINT
200 PRINT "YOUR AVERAGE WEEKLY EARNINGS WERE $"A
210 END
```

MOVIE THEATRE DIRECTORY

To add addresses to the program you could have written lines like these:

```
39 READ A$
45 PRINT TAB(10)A$
105 DATA 2301 S. LAKE DRIVE
115 DATA 4777 W. HIGGINS ROAD
125 DATA 18 S. MAIN STREET
135 DATA 555 E. PALATINE ROAD
145 DATA 1134 N. DOUGLAS ROAD
155 DATA 455 E. HIGGINS ROAD
165 DATA 2456 N. MAIN STREET
175 DATA 1010 W. 7TH STREET
180 DATA EDWARDS, 555-9999,756 LAUREL AVENUE
190 DATA MISSION,555-8577,1231 S. CHERRY STREET
```

15-1 Use the MOVIE DIRECTORY program in this lesson as a guide.

15-2

```

5 REM ADDRESS FILE USING ARRAYS, DATA AND READ
10 M=1000: REM PROGRAM CAN CONTAIN 1000 NAMES
20 DIM NAMES(M), ADDRESS$(M), CITY$(M), STATES$(M), ZIP$(M)
30 FOR X=1 TO M
40 READ NAMES(X)
50 IF NAMES(X)="XX" THEN GOTO 100
60 READ ADDRESS$(X), CITY$(X), STATES$(X), ZIP$(X)
70 NEXT X
100 PRINT "[CLR]"
110 PRINT "WHOM ARE YOU LOOKING FOR?"
120 INPUT N$
130 FOR X=1 TO M
140 IF N$=NAMES(X) THEN GOTO 240
150 NEXT X
160 PRINT: PRINT "I DON'T HAVE AN ADDRESS FOR THAT PERSON."
170 PRINT: PRINT
180 PRINT "PRESS THE LETTER 'A' IF YOU WANT ANOTHER"
190 PRINT: PRINT "PRESS THE LETTER 'D' IF YOU ARE DONE."
200 GET A$
210 IF A$="A" THEN GOTO 100
220 IF A$="D" THEN END
230 IF A$<"A" OR A$<"D" THEN GOTO 200
240 PRINT: PRINT "DO YOU WANT TO PRINT 1) ON THE SCREEN"
250 PRINT TAB(17)" OR 2) ON THE PRINTER?"
260 GET P$
270 IF P$="1" THEN GOTO 300
280 IF P$="2" THEN GOTO 340
290 IF P$<"1" OR P$<"2" THEN GOTO 260
300 PRINT: PRINT NAMES(X)
310 PRINT ADDRESS$(X)
320 PRINT CITY$(X) ", " STATES$(X) " " ZIP$(X)
330 GOTO 170
340 PRINT: PRINT "PRESS THE LETTER 'Y' WHEN YOU ARE READY TO PRINT."
350 GET Y$
360 IF Y$="Y" THEN GOTO 380
370 IF Y$<"Y" THEN GOTO 350
380 OPEN#99,4:CMD#99
390 PRINT#99
400 PRINT#99,NAMES(X)
410 PRINT#99,ADDRESS$(X)
420 PRINT#99,CITY$(X) ", " STATES$(X) " " ZIP$(X)
430 PRINT#99:CLOSE#99
440 GOTO 170
5000 DATA SARA BROWN,555 ROSE LANE,CHICAGO,IL,60056
5010 DATA SAM SPRAGUE, 123 W. 5TH STREET,WINDSOR,CO,80550
5020 DATA LISA JONES,750 E. STUART ROAD,CHARLESTON,IL,61920
5030 DATA XX

```


15-3

```

5 REM USED CAR LOT PROGRAM USING DATA/READ
10 PRINT "[CLR]"
20 DATA SOUPED-UP '74 CAMARO, BEAT-UP '71 MAVERICK
30 DATA HOT LITTLE DUNE BUGGY, CLASSY '78 CADDY
40 DATA REBUILT BUG, FAST 280Z
50 DATA JACKED-UP PICK UP, VERY EXPENSIVE MERCEDES
60 FOR X=1 TO 8
70 READ C$(X)
80 NEXT X
90 PRINT "THESE ARE ALL THE GREAT CARS WE HAVE.": PRINT
100 FOR X=1 TO 8
110 PRINT "#";X; C$(X)
120 NEXT X
130 PRINT: PRINT "WOULD YOU LIKE TO TEST DRIVE ONE?"
140 PRINT: INPUT "(YES OR NO)";D$
150 IF D$="NO" THEN GOTO 500
160 IF D$="YES" THEN PRINT: PRINT "WHICH ONE DO YOU WANT TO TEST?": GOTO 180
170 IF D$<>"YES" AND D$<>"NO" THEN GOTO 140
180 INPUT "#";X
190 PRINT: PRINT "TAKE HER FOR A SPIN AND"
200 PRINT "COME BACK AND WE'LL TALK PRICE."
210 FOR D=1 TO 2000: NEXT D
220 PRINT "[CLR]"
230 PRINT "WELL, NOW THAT YOU'VE SEEN THE CAR,"
240 PRINT "DO YOU WANT TO BUY IT?"
250 PRINT: INPUT "(Y OR N)";Y$
260 IF Y$="N" THEN PRINT: PRINT "SHOP AROUND BUT WE HAVE THE BEST DEAL.": END
270 IF Y$="Y" THEN PRINT: PRINT "WELL THAT CAR WILL COST YOU"
280 IF Y$<>"Y" AND Y$<>"N" THEN GOTO 250
290 ON X GOSUB 310,320,330,340,350,360,370,380
300 PRINT: PRINT TAB(8) "I HOPE YOU HAVE CASH!": END
310 PRINT TAB(28) "$500.00": RETURN
320 PRINT TAB(28) "$2000.00": RETURN
330 PRINT TAB(28) "$8000.00": RETURN
340 PRINT TAB(28) "$5000.00": RETURN
350 PRINT TAB(28) "$3500.00": RETURN
360 PRINT TAB(28) "$6700.00": RETURN
370 PRINT TAB(28) "$560.00": RETURN
380 PRINT TAB(28) "$12,000.00": RETURN
500 PRINT: PRINT "SEE THE FLOOR MANAGER IF YOU NEED HELP"
510 END

```

16-1

```
5 REM APPOINTMENT SCHEDULE USING DATA, READ AND RESTORE
10 PRINT "[CLR]"
20 INPUT "APPOINTMENTS FOR WHICH MONTH";M$
30 INPUT "AND WHICH DATE";D
40 FOR X=1 TO 100
50 READ M1$
60 IF M1$="END" THEN GOTO 100
70 READ D1,A$
80 IF M1$=M$ AND D1=D THEN PRINT: PRINT M1$;D1,A$
90 NEXT X
100 PRINT: PRINT
110 PRINT "PRESS THE LETTER 'A' IF YOU WISH TO SEE OTHERS."
120 PRINT: PRINT "PRESS THE LETTER 'D' IF DONE."
130 GET L$
140 IF L$="A" THEN RESTORE: GOTO 10
150 IF L$="D" THEN END
160 IF L$<>"A" OR L$<>"D" THEN GOTO 130
1000 DATA MAY,31,SAM'S BIRTHDAY
1010 DATA MAY,1, MEETING AT SCHOOL-2 P.M.
1020 DATA MAY,1,PTA MEETING-7 P.M.
1030 DATA JUNE,4,DENTIST AT 2 P.M.
1040 DATA JUNE,4,BRUNCH AT 9 A.M.
1050 DATA END
```

16-2

```
5 REM FIRST INITIAL USING 15 COLORS
10 PRINT "[CLR]"
20 FOR S=0 TO 15: REM CHANGE SCREEN COLOR
30 POKE 53281,S
40 FOR C=0 TO 15: REM CHANGE COLOR OF INITIAL
50 FOR X=1 TO 15: REM PLOT 15 SQUARES
60 READ A: READ B
70 POKE A,224: POKE A+54272,C
80 NEXT X
90 FOR D=1 TO 300: NEXT D
100 RESTORE
110 NEXT C
120 NEXT S
130 END
1000 REM DATA FOR THE LETTER J
1010 DATA 1200,1200,1201,1201
1020 DATA 1202,1202,1203,1203
1030 DATA 1204,1204,1205,1205
1040 DATA 1206,1206,1243,1243
1050 DATA 1283,1283,1323,1323
1060 DATA 1363,1363,1403,1403
1070 DATA 1443,1443,1442,1442
1080 DATA 1441,1441
```

17-1 Use the BIRTHDAY LIST program in this lesson as a guide.

17-2

```

5 REM DEFINITIONS/WORDS USING TWO-DIMENSIONAL ARRAY
10 DIM W$(10,2)
20 PRINT "[CLR]"
30 PRINT "YOU WILL BE GIVEN A DEFINITION."
40 PRINT: PRINT TAB(10)"YOU INPUT THE WORD."
50 PRINT: PRINT "(EACH WORD BEGINS WITH THE LETTER Z.)"
60 FOR X=1 TO 10
70 READ W$(X,1)
80 READ W$(X,2)
90 NEXT X
100 FOR X=1 TO 10
110 PRINT: PRINT: PRINT TAB(10)"DEFINITION:"
120 PRINT: PRINT W$(X,1)
130 PRINT: PRINT
140 PRINT "WOULD YOU LIKE TO SEE THE WORDS"
150 INPUT "(Y OR N)";A$
160 IF A$="Y" THEN PRINT: GOTO 300
170 IF A$="N" THEN GOTO 190
180 IF A$<>"Y" AND A$<>"N" THEN GOTO 150
190 PRINT: PRINT
200 PRINT "WHAT DO YOU THINK THE WORD IS"
210 INPUT D$
220 IF D$=W$(X,2) THEN PRINT: PRINT "YOU ARE RIGHT!"
230 IF D$<>W$(X,2) THEN PRINT: PRINT W$(X,2) " IS " W$(X,1)":"
240 NEXT X
250 END
299 REM MIX WORDS SLIGHTLY BEFORE PRINTING
300 FOR Y=10 TO 2 STEP -2
310 PRINT W$(Y,2),
330 NEXT Y
340 FOR Y=1 TO 9 STEP 2
350 PRINT W$(Y,2),
360 NEXT Y
370 GOTO 190
1000 DATA AN ANIMAL WITH LIGHT AND DARK STRIPES, ZEBRA
1010 DATA A FLOWERING PLANT, ZINNIA
1020 DATA THE ABSENCE OF QUANTITY, ZERO
1030 DATA A METALLIC CHEMICAL ELEMENT, ZINC
1040 DATA A FASTENING DEVICE, ZIPPER
1050 DATA A MUSICAL INSTRUMENT, ZITHER
1060 DATA A PLACE WHERE LIVE ANIMALS ARE KEPT, ZOO
1070 DATA A KIND OF TOASTED BREAD, ZWIEBACK
1080 DATA A SUDDEN UPWARD COURSE, ZOOM
1090 DATA A GENTLE BREEZE, ZEPHYR
    
```

17-3

```
5 REM NAME AND TELEPHONE NUMBER
10 T=100: DIM N$(T,2)
20 FOR X=1 TO T
30 READ N$(X,1)
40 IF N$(X,1)="XX" THEN GOTO 70
50 READ N$(X,2)
60 NEXT X
70 PRINT "[CLR]"
80 PRINT "THIS PROGRAM CONTAINS NAMES"
90 PRINT "AND TELEPHONE NUMBERS."
100 PRINT: PRINT "HERE IS A LIST OF THE NAMES:"
110 FOR X=1 TO T
120 IF N$(X,1)="XX" THEN GOTO 150
130 PRINT "#X; " "N$(X,1)
140 NEXT X
150 PRINT: PRINT "INPUT A NUMBER TO SEE THAT PERSON'S"
160 PRINT "TELEPHONE NUMBER."
170 PRINT: INPUT "PERSON'S NUMBER";A
180 PRINT: PRINT N$(A,2)
190 PRINT: PRINT
200 GOTO 110
1000 DATA CARROLL, 245-5678
1010 DATA JEFF, 667-1245
1020 DATA JOHN, 678-1245
1030 DATA SCOTT, 690-9100
1040 DATA JUDY, 245-7777
1050 DATA XX
```

17-4 Use the HOROSCOPE GENERATOR program in this lesson as a guide.

18-1 To add graphics to the rolling die program these lines could be changed:

```
70 ON X GOSUB 400,500,600,700,800,900
100 ON X GOSUB 400,500,600,700,800,900
```

And these lines could be added:

```
75 GOSUB 300
105 GOSUB 300
400 REM ROLLING A ONE
410 PRINT: PRINT: PRINT
420 PRINT TAB(15) "SHFT O CMDR Y (5 TIMES) SHFT P"
430 PRINT TAB(15) "CMDR H 5 SPACES CMDR N"
440 PRINT TAB(15) "CMDR H 2 SPACES SHFT Q 2 SPACES CMDR N"
450 PRINT TAB(15) "CMDR H 5 SPACES CMDR N"
460 PRINT TAB(15) "SHFT L CMDR P (5 TIMES) SHFT @"
470 RETURN
500 REM ROLLING A TWO
510 PRINT: PRINT: PRINT
520 PRINT TAB(15) "SHFT O CMDR Y (5 TIMES) SHFT P"
530 PRINT TAB(15) "CMDR H 4 SPACES SHFT Q CMDR N"
540 PRINT TAB(15) "CMDR H 5 SPACES CMDR N"
550 PRINT TAB(15) "CMDR H SHFT Q 4 SPACES CMDR N"
560 PRINT TAB(15) "SHFT L CMDR P (5 TIMES) SHFT @"
570 RETURN
600 REM ROLLING A THREE
610 PRINT: PRINT: PRINT
620 PRINT TAB(15) "SHFT O CMDR Y (5 TIMES) SHFT P"
630 PRINT TAB(15) "CMDR H 4 SPACES SHFT Q CMDR N"
640 PRINT TAB(15) "CMDR H 2 SPACES SHFT Q 2 SPACES CMDR N"
650 PRINT TAB(15) "CMDR H SHFT Q 4 SPACES CMDR N"
660 PRINT TAB(15) "SHFT L CMDR P (5 TIMES) SHFT @"
670 RETURN
700 REM ROLLING A FOUR
710 PRINT: PRINT: PRINT
720 PRINT TAB(15) "SHFT O CMDR Y (5 TIMES) SHFT P"
730 PRINT TAB(15) "CMDR H SPACE SHFT Q SPACE SHFT Q SPACE CMDR N"
740 PRINT TAB(15) "CMDR H 5 SPACES CMDR N"
750 PRINT TAB(15) "CMDR H SPACE SHFT Q SPACE SHFT Q SPACE CMDR N"
760 PRINT TAB(15) "SHFT L CMDR P (5 TIMES) SHFT @"
770 RETURN
800 REM ROLLING A FIVE
810 PRINT: PRINT: PRINT
820 PRINT TAB(15) "SHFT O CMDR Y (5 TIMES) SHFT P"
830 PRINT TAB(15) "CMDR H SPACE SHFT Q SPACE SHFT Q SPACE CMDR N"
840 PRINT TAB(15) "CMDR H 2 SPACES SHFT Q 2 SPACES CMDR N"
850 PRINT TAB(15) "CMDR H SPACE SHFT Q SPACE SHFT Q SPACE CMDR N"
860 PRINT TAB(15) "SHFT L CMDR P (5 TIMES) SHFT @"
870 RETURN
900 REM ROLLING A SIX
910 PRINT: PRINT: PRINT
920 PRINT TAB(15) "SHFT O CMDR Y (5 TIMES) SHFT P"
930 PRINT TAB(15) "CMDR H SPACE SHFT Q SPACE SHFT Q SPACE CMDR N"
940 PRINT TAB(15) "CMDR H SPACE SHFT Q SPACE SHFT Q SPACE CMDR N"
950 PRINT TAB(15) "CMDR H SPACE SHFT Q SPACE SHFT Q SPACE CMDR N"
960 PRINT TAB(15) "SHFT L CMDR P (5 TIMES) SHFT @"
970 RETURN
```

18-2

```
5 REM MULTIPLE CHOICE QUIZ USING ON GOTO
10 PRINT "[CLR]"
20 PRINT "WHAT IS THE BEST WAY TO TAKE CARE"
30 PRINT "OF A MESSY BEDROOM?"
40 PRINT: PRINT
50 PRINT "1) SHOVE EVERYTHING INTO THE CLOSET"
60 PRINT "2) HANG UP A SIGN THAT SAYS 'MODERN ART'"
70 PRINT "3) MOVE TO ANOTHER HOUSE"
80 PRINT "4) HAVE YOUR MOTHER CLEAN IT"
90 PRINT "5) CLEAN IT UP YOURSELF"
100 PRINT: PRINT: PRINT
110 PRINT "SELECT A NUMBER AND PRESS RETURN."
120 INPUT A
130 PRINT "[CLR]"
140 ON A GOTO 150,170,190,210,230
150 PRINT "THAT'S NOT A VERY PERMANENT SOLUTION."
160 GOSUB 500: GOTO 10
170 PRINT "YOU MIGHT HAVE GOTTEN AWAY WITH THAT IN THE 60'S!"
180 GOSUB 500: GOTO 10
190 PRINT "WITH TODAY'S INTEREST RATES???"
200 GOSUB 500: GOTO 10
210 PRINT "RULE THIS ONE OUT — IMPOSSIBLE."
220 GOSUB 500: GOTO 10
230 PRINT "YES, THAT'S THE RIGHT ANSWER."
240 PRINT "BET YOU TRIED ALL THE OTHERS FIRST!!"
250 END
500 FOR D=1 TO 2000: NEXT D
510 RETURN
```

19-1

```
5 REM POKE WHITE LETTERS OF THE ALPHABET IN CENTER OF SCREEN
10 PRINT "[CLR]"
20 POKE 53200,12: REM CHANGE BORDER TO GRAY
30 POKE 53281,0: REM CHANGE BACKGROUND TO BLACK
40 FOR L=1 TO 26
50 POKE 1522,L
60 POKE 1522+54272,1
70 FOR D=1 TO 500: NEXT D
80 NEXT L
```

19-2

```
5 REM NAME ON DIAGONAL
10 PRINT "[CLR]"
20 POKE 53281,1: REM CHANGE BACKGROUND TO WHITE
30 X=4: Y=4: REM RED C
40 P=X+40*Y
50 POKE 1024+P,3: POKE 55296+P,2
60 GOSUB 500
70 X=8: Y=6: REM ORANGE R
80 P=X+40*Y
90 POKE 1024+P,18: POKE 55296+P,8
100 GOSUB 500
110 X=12: Y=8: REM YELLOW E
120 P=X+40*Y
130 POKE 1024+P,5: POKE 55296+P,7
140 GOSUB 500
150 X=16: Y=10: REM GREEN A
160 P=X+40*Y
170 POKE 1024+P,1: POKE 55296+P,5
180 GOSUB 500
190 X=20: Y=12: REM LT. BLUE T
200 P=X+40*Y
210 POKE 1024+P,20: POKE 55296+P,14
220 GOSUB 500
230 X=24: Y=14: REM BLUE I
240 P=X+40*Y
250 POKE 1024+P,9: POKE 55296+P,6
260 GOSUB 500
270 X=28: Y=16: REM PURPLE V
280 P=X+40*Y
290 POKE 1024+P,22: POKE 55296+P,4
300 GOSUB 500
310 X=32: Y=18: REM CYAN E
320 P=X+40*Y
330 POKE 1024+P,5: POKE 55296+P,3
340 GOTO 340
500 REM SUBROUTINE FOR DELAY
510 FOR D=1 TO 1500: NEXT D
520 RETURN
```

19-3

```
5 REM 4 GRAPHIC PICTURES USING GOSUB/RETURN AND INPUT
10 PRINT "[CLR]"
20 PRINT "I CAN DRAW FOUR LETTERS ON THE SCREEN."
30 PRINT: PRINT "I CAN DRAW AN L, AN E, A T OR AN H."
40 PRINT: PRINT "CHOOSE THE NUMBER FOR THE LETTER"
50 PRINT "YOU WANT TO SEE."
60 PRINT: PRINT "L=1" TAB(10) "E=2" TAB(20) "T=3 TAB(30) "H=4"
70 PRINT: PRINT
80 INPUT "NUMBER ";N
90 ON N GOSUB 200,400,700,900
100 GOTO 40
200 REM ----- DRAW YELLOW L -----
210 PRINT "[CLR]"
220 FOR Y=6 TO 18: X=12
230 P=X+40*Y
240 POKE 1024+P,127: POKE 55296+P,7
250 NEXT Y
260 FOR X=12 TO 22: Y=18
270 P=X+40*Y
280 POKE 1024+P,127: POKE 55296+P,7
290 NEXT X
300 GOSUB 5000
310 RETURN
400 REM ----- DRAW GREEN E -----
410 PRINT "[CLR]"
420 FOR X=1 TO 8: Y=1
430 P=X+40*Y
440 POKE 1024+P,224: POKE 55296+P,5
450 NEXT X
460 FOR Y=1 TO 12: X=1
470 P=X+40*Y
480 POKE 1024+P,224: POKE 55296+P,5
490 NEXT Y
500 FOR X=1 TO 8: Y=12
510 P=X+40*Y
520 POKE 1024+P,224: POKE 55296+P,5
530 NEXT X
540 FOR X=1 TO 4: Y=6
550 P=X+40*Y
560 POKE 1024+P,224: POKE 55296+P,5
570 NEXT X
580 GOSUB 5000
590 RETURN
700 REM ----- DRAW ORANGE T -----
710 PRINT "[CLR]"
720 FOR X=1 TO 38: Y=2
730 P=X+40*Y
740 POKE 1024+P,121: POKE 55296+P,8
750 NEXT X
760 FOR Y=3 TO 23: X=20
770 P=X+40*Y
780 POKE 1024+P,81: POKE 55296+P,8
790 NEXT Y
800 GOSUB 5000
810 RETURN
```



```

900 REM ----- DRAW RED H -----
910 PRINT "[CLR]"
920 FOR Y=13 TO 23: X=28
930 P=X+40*Y
940 POKE 1024+P,83: POKE 55296+P,2
950 NEXT Y
960 FOR Y=13 TO 23: X=36
970 P=X+40*Y
980 POKE 1024+P,83: POKE 55296+P,2
990 NEXT Y
1000 FOR X=28 TO 35: Y=18
1010 P=X+40*Y
1020 POKE 1024+P,83: POKE 55296+P,2
1030 NEXT X
1040 GOSUB 5000
1050 RETURN
5000 REM SUBROUTINE FOR DELAY AND CLEAR SCREEN
5010 FOR D=1 TO 2000: NEXT D
5020 PRINT "[CLR]"
5030 RETURN
    
```

19-4

```

5 REM BAR GRAPHS OF FOUR FAVORITE T.V. SHOWS
10 PRINT "[CLR]"
20 REM PRINTS HEADING TITLES
30 PRINT " T" SPC(8)"D" SPC(8)"A" SPC(8)"D" SPC(8)"D"
40 PRINT " O" SPC(8)"U" SPC(17)"R" SPC(8)"A"
50 PRINT " T" SPC(8)"K" SPC(8)"T" SPC(17)"L"
60 PRINT " A" SPC(8)"E" SPC(8)"E" SPC(8)"W" SPC(8)"L"
70 PRINT " L" SPC(8)"S" SPC(8)"A" SPC(8)"H" SPC(8)"A"
80 PRINT TAB(19)"M" SPC(8)"O" SPC(8)"S"
90 REM DRAWS BLACK BAR FOR TOTAL
100 FOR Y=10 TO 24: X=1
110 P=X+40*Y
120 POKE 1024+P,102: POKE 55296+P,0
130 NEXT Y
140 REM DRAWS RED BAR FOR DUKES OF HAZZARD
150 FOR Y=12 TO 24: X=10
160 P=X+40*Y
170 POKE 1024+P,224: POKE 55296+P,2
180 NEXT Y
190 REM DRAWS ORANGE BAR FOR A-TEAM
200 FOR Y=10 TO 24: X=19
210 P=X+40*Y
220 POKE 1024+P,81: POKE 55296+P,8
230 NEXT Y
240 REM DRAWS GREEN BAR FOR DR. WHO
250 FOR Y=16 TO 24: X=28
260 P=X+40*Y
270 POKE 1024+P,86: POKE 55296+P,5
280 NEXT Y
290 REM DRAWS YELLOW BAR FOR DALLAS
300 FOR Y=14 TO 24: X=37
310 P=X+40*Y
320 POKE 1024+P,90: POKE 55296+P,7
330 NEXT Y
340 GOTO 340
    
```

20-1

```

5 REM DRAW MOVING RECTANGLE
10 PRINT "[CLR]"
20 POKE 53280,13: REM LIGHT GREEN BORDER
30 POKE 53281,5: REM GREEN SCREEN
40 X=1: REM STARTING POSITION FOR ROW
50 Y=12: REM STARTING POSITION FOR COLUMN
60 H=1: REM STARTING HORIZONTAL POSITION
70 V=1: REM STARTING VERTICAL POSITION
80 P=X+40*Y
90 POKE 1024+P,224
100 FOR D=1 TO 10: NEXT D
110 POKE 1024+P,32
120 X=X+H: REM FINDS SIDES AND REVERSES
130 IF X=0 THEN H=-H
140 IF X=39 THEN H=-H
150 GOTO 80

```

Add the following lines to the above program to make a slow-moving, yellow "inchworm" —

```

92 POKE 55296+P,7
94 POKE 1025+P,224
96 POKE 55297+P,7
100 FOR D=1 TO 500: NEXT D
115 POKE 1025+P,32

```

20-2

```

5 REM DRAW MOVING ORANGE CAR
10 PRINT "[CLR]"
20 X=7: Y=12
30 P=X+40*Y
40 POKE 1024+P,78: POKE 55296+P,8
50 FOR A=1 TO 2
60 POKE 1024+P+A,99: POKE 55296+P+A,8
70 NEXT A
80 P=(X-1)+40*(Y+1)
90 POKE 1024+P,78: POKE 55296+P,8
100 FOR B=1 TO 5
110 P=(X-2)+40*(Y+2)
120 POKE 1024+P+B,69: POKE 55296+P+B,8
130 NEXT B
140 P=(X+3)+40*Y
150 POKE 1024+P,77: POKE 55296+P,1
160 P=(X+3)+40*(Y+1)
170 POKE 1024+P,106: POKE 55296+P,8
180 P=X+40*(Y+2)
190 POKE 1024+P,87: POKE 55296+P,0
200 P=(X+2)+40*(Y+2)
210 POKE 1024+P,87: POKE 55296+P,0
220 FOR D=1 TO 1000: NEXT D
230 PRINT "[CLR]"
240 X=X+3: GOTO 30

```

20-3

```

5 REM ANIMATED SIGN
10 T=3: X=0: Y=10
20 PRINT "[CLR]"
30 PRINT "SAVE YOUR MONEY FOR A RAINY DAY!!!!"
40 PRINT TAB(3)"IN THE MUTUAL SAVINGS AND LOAN"
50 PRINT TAB(10)"789 SUNSET BOULEVARD"
60 FOR D=1 TO 750: NEXT D
70 PRINT: PRINT: PRINT
80 PRINT TAB(T)"R" TAB(T+20)"R"
90 PRINT TAB(T)"A" TAB(T+12)"R" TAB(T+20)"A"
100 PRINT TAB(T)"I" TAB(T+12)"A" TAB(T+20)"I"
110 PRINT TAB(T)"N" TAB(T+12)"I" TAB(T+20)"N"
120 PRINT TAB(T+12)"N"
129 REM DRAWS GREEN MONEY
130 FOR A=1 TO 15
140 P=X+40*Y
150 POKE 1024+P+A,224: POKE 55296+P+A,5
160 NEXT A
170 FOR B=1 TO 15
180 P=X+40*(Y+6)
190 POKE 1024+P+B,224: POKE 55296+P+B,5
200 NEXT B
210 FOR C=1 TO 5
220 P=(X+1)+40*(Y+C)
230 POKE 1024+P,224: POKE 55296+P,5
240 NEXT C
250 FOR D=1 TO 5
260 P=(X+15)+40*(Y+D)
270 POKE 1024+P,224: POKE 55296+P,5
280 NEXT D
290 P=(X+5)+40*(Y+2)
300 POKE 1024+P,36: POKE 55296+P,5
310 P=(X+10)+40*(Y+4)
320 POKE 1024+P,36: POKE 55296+P,5
330 FOR D=1 TO 1500: NEXT D
340 T=T+5: IF T=235 THEN T=0
350 X=X+3: IF X=21 THEN X=0
360 Y=Y+2: IF Y=16 THEN Y=10
370 PRINT "[CLR]": GOTO 30
    
```

21-1 To add sound to the INCHWORM program, the following lines could be added to set the sound variables:

```

7 VOL=54296: A=54277: W=54276
9 HF=54273: L=54272
    
```

Then, these lines could be added to produce random sounds as the worm moves:

```

112 FOR S=54272 TO 54296: POKE S,0: NEXT S
113 N1=INT(RND(1)*50)+1
114 N2=INT(RND(1)*200)+1
115 POKE VOL, 15: POKE A,15: POKE W,17
116 POKE HF,N1: POKE L,N2
117 FOR T=1 TO 50: NEXT T
118 POKE VOL,0: POKE A,0: POKE W,0
    
```

21-2

```
5 REM GREENSLEEVES
10 FOR C=54272 TO 54296: POKE C,0: NEXT C
20 L=54272
30 H=L+1: P=L+2
40 U=L+3: W=L+4
50 A=L+5: S=L+6
60 V=54296
70 POKE V,15: POKE A,46: POKE S,1
80 FOR X=1 TO 72
90 READ A,B,C,
100 POKE W,17
110 POKE H,A: POKE L,B
120 IF A=0 THEN END
130 FOR D=1 TO 250*C: NEXT D
140 POKE W,0
150 NEXT X
1000 DATA 31,165,1
1010 DATA 37,162,2
1020 DATA 42,62,1
1030 DATA 47,107,1.5
1040 DATA 50,60,.5
1050 DATA 47,107,1
1060 DATA 42,62,2
1070 DATA 35,134,1
1080 DATA 28,49,1.5
1090 DATA 31,165,.5
1100 DATA 35,134,1
1110 DATA 37,162,2
1120 DATA 31,165,1
1130 DATA 31,165,1.5
1140 DATA 29,223,.5
1150 DATA 31,165,1
1160 DATA 35,134,2
1170 DATA 29,223,1
1180 DATA 23,181,2
1190 DATA 31,165,1
1200 DATA 37,162,2
1210 DATA 42,62,1
1220 DATA 47,107,1.5
1230 DATA 50,60,.5
1240 DATA 47,107,1
1250 DATA 42,62,2
1260 DATA 35,134,1
1270 DATA 28,49,1.5
1280 DATA 31,165,.5
1290 DATA 35,134,1
1300 DATA 37,162,1
1310 DATA 35,134,1
1320 DATA 31,165,1
1330 DATA 29,223,1.5
1340 DATA 26,156,.5
1350 DATA 29,223,1
1360 DATA 31,165,2
1370 DATA 31,165,1
1380 DATA 31,165,3
1390 DATA 56,99,3
1400 DATA 56,99,1.5
1410 DATA 53,57,.5
1420 DATA 47,107,1
1430 DATA 42,62,2
1440 DATA 35,134,1
1450 DATA 28,49,1.5
1460 DATA 31,165,.5
1470 DATA 35,134,1
1480 DATA 37,162,2
1490 DATA 31,165,1
1500 DATA 31,165,1.5
1510 DATA 29,223,.5
1520 DATA 31,165,1
1530 DATA 35,134,2
1540 DATA 29,223,1
1550 DATA 23,181,3
1560 DATA 56,99,3
1570 DATA 56,99,1.5
1580 DATA 53,56,.5
1590 DATA 47,107,1
1600 DATA 42,62,2
1610 DATA 35,134,1
1620 DATA 28,49,1.5
1630 DATA 31,165,.5
1640 DATA 35,134,1
1650 DATA 37,162,1
1660 DATA 35,134,1
1670 DATA 31,165,1
1680 DATA 29,223,1.5
1690 DATA 26,156,.5
1700 DATA 29,223,1
1710 DATA 31,165,5
1720 DATA 0
```

21-3 These lines could be added to the birthday cake program:

```

800 REM HAPPY BIRTHDAY USING 3-PART HARMONY
810 FOR C=54272 TO 54296: POKE C,0: NEXT C
820 L=54272
830 H=L+1: W=L+4
840 A=L+5: S=L+6
850 L2=54279
860 H2=L2+1: W2=L2+4
870 A2=L2+5: S2=L2+6
880 L3=54286
890 H3=L3+1: W3=L3+4
900 A3=L3+5: S3=L3+6
910 V=54296
920 POKE V,15
930 POKE A,12: POKE S,10
940 POKE A2,88: POKE S2,3
950 POKE A3,9: POKE S3,0
960 READ D
970 IF D<0 THEN GOTO 1050
980 READ M, K, M2, K2, M3, K3
990 IF M>0 THEN POKE W,17: POKE H,M: POKE L,K
1000 IF M2>0 THEN POKE W2,33: POKE H2,M2: POKE L2,K2
1010 IF M3>0 THEN POKE W3,33: POKE H3,M3: POKE L3,K3
1020 FOR C=1 TO D*250: NEXT C
1030 FOR C=54272 TO 54296: POKE C,0: NEXT C
1040 GOTO 920
1050 POKE V,0: END
2000 DATA .75,33,135,28,49,22,96
2010 DATA .25,33,135,28,49,22,96
2020 DATA 1,37,162,29,223,22,96
2030 DATA 1,33,135,28,49,22,96
2040 DATA 1,44,193,33,135,28,49
2050 DATA 2.5,42,62,33,135,29,223
2060 DATA .75,33,135,29,223,21,31
2070 DATA .25,33,135,29,223,21,31
2080 DATA 1,37,162,29,223,21,31
2090 DATA 1,33,135,29,223,21,31
2100 DATA 1,50,60,33,135,29,223
2110 DATA 2.5,41,193,33,135,29,135
2120 DATA .75,33,135,28,49,22,96
2130 DATA .25,33,135,28,49,22,96
2140 DATA 1,67,15,44,193,33,135
2150 DATA 1,56,99,44,193,33,135
2160 DATA 1,44,193,33,135,28,49
2170 DATA 1,42,62,29,223,22,96
2180 DATA 2.5,37,162,29,223,22,96
2190 DATA .75,59,190,33,135,25,30
2200 DATA .25,59,190,33,135,25,30
2210 DATA 1,56,99,33,135,28,49
2220 DATA 1,44,193,33,135,28,49
2230 DATA 1,50,60,33,135,29,223
2240 DATA 3,44,193,33,135,28,49
2250 DATA -1

```

21-4

5 REM MOVING SPACESHIP AND STAR WARS SONG

10 X=INT(RND(1)*19)+5: REM NUMBER BETWEEN 5 AND 24

20 Y=INT(RND(1)*19)+1: REM NUMBER BETWEEN 1 AND 19

30 PRINT "[CLR]"

40 FOR AA=1 TO 4

50 P=X+40*Y

60 POKE 1024+P+AA,224

70 POKE 55296+P+AA,12

80 NEXT AA

90 FOR BB=1 TO 8

100 P=(X-1)+40*(Y+1)

110 POKE 1024+P+BB,224

120 POKE 55296+P+BB,12

130 NEXT BB

140 FOR CC=1 TO 16

150 P=(X-6)+40*(Y+2)

160 POKE 1024+P+CC,224

170 POKE 55296+P+CC,12

180 NEXT CC

189 REM WINDOWS IN SPACESHIP

190 FOR DD=1 TO 16 STEP 2

200 P=(X-6)+40*(Y+3)

210 POKE 1024+P+DD,224

220 POKE 55296+P+DD,12

230 NEXT DD

240 FOR EE=1 TO 16

250 P=(X-6)+40*(Y+4)

260 POKE 1024+P+EE,224

270 POKE 55296+P+EE,12

280 NEXT EE

290 FOR FF=1 TO 8

300 P=(X-2)+40*(Y+5)

310 POKE 1024+P+FF,224

320 POKE 55296+P+FF,12

330 NEXT FF

340 REM MUSIC

350 FOR C=54272 TO 54296: POKE C,0: NEXT C

360 L=54272

370 V=L+1: A=L+4: S=L+5: W=L+6

380 V=54296

390 POKE V,15: POKE A,12: POKE S,4

400 FOR N=1 TO 16

410 READ A,B,C

420 POKE W,33

430 POKE H,A: POKE L,B

440 FOR D=1 TO 250*C: NEXT D

450 POKE W,0

460 NEXT N

1000 DATA 16,195,2

1010 DATA 25,30,2

1020 DATA 22,96,,25

1030 DATA 21,31,,25

1040 DATA 18,209,,25

1050 DATA 33,135,2

1060 DATA 25,30,1

1070 DATA 22,96,,25

1080 DATA 21,31,,25

1090 DATA 18,209,,25

1100 DATA 33,135,2

1110 DATA 25,30,1

1120 DATA 22,96,,25

1130 DATA 21,31,,25

1140 DATA 22,96,,25

1150 DATA 18,209,2

1160 RESTORE: GOTO 10

22-1

```
5 REM YOUR NAME USING CHR$
10 PRINT "[CLR]"
20 FOR T=1 TO 40: REM GRAPHIC AT TOP OF SCREEN
30 PRINT CHR$(191);
40 NEXT T
50 FOR X=1 TO 7
60 PRINT
70 NEXT X
79 REM PRINT YOUR
80 PRINT TAB(2) CHR$(89), CHR$(79), CHR$(85), CHR$(82)
89 REM PRINT NAME
90 PRINT: PRINT
100 PRINT TAB(2) CHR$(78),CHR$(65),CHR$(77),CHR$(69)
110 FOR X=1 TO 8
120 PRINT
130 NEXT X
140 FOR B=1 TO 40: REM GRAPHIC AT BOTTOM OF SCREEN
150 PRINT CHR$(230);
160 NEXT B
170 GOTO 170
```

23-1

```
5 REM SORT 10 NAMES BY FIRST LETTER
10 PRINT "[CLR]"
20 DIM N$(10)
30 PRINT "ENTER 10 NAMES LIKE THIS: JOHN"
40 FOR X=1 TO 10
50 INPUT N$(X)
60 NEXT X
70 PRINT "[CLR]"
80 PRINT "CHOOSE A LETTER AND I WILL SHOW"
90 PRINT "YOU ALL THE NAMES YOU TYPED THAT"
100 PRINT "BEGIN WITH THAT LETTER."
110 PRINT: INPUT "LETTER";L$
120 PRINT
130 FOR X=1 TO 10
140 IF ASC(L$)=ASC(LEFT$(N$(X),1)) THEN PRINT N$(X)
150 NEXT X
160 FOR D=1 TO 2000: NEXT D
170 PRINT "[CLR]": GOTO 110
```

23-2

```
5 REM COMPUTER SCRABBLE
10 PRINT "[CLR]"
20 PRINT "THIS IS A COMPUTER SCRABBLE GAME FOR 2"
30 PRINT: PRINT
40 PRINT "EACH LETTER IN A WORD IS WORTH POINTS."
50 PRINT: PRINT TAB(10)"OBJECT OF THE GAME:"
60 PRINT TAB(8)"GET THE MOST POINTS!!!"
70 PRINT: PRINT: PRINT
80 PRINT TAB(3)"PLAYER #1 CHOOSES HOW MANY LETTERS"
90 PRINT "BOTH PLAYERS' WORDS MUST HAVE."
100 PRINT: PRINT TAB(3)"THE WINNER OF EACH ROUND
BECOMES"
110 PRINT "PLAYER #1 FOR THE NEXT ROUND."
120 PRINT: PRINT "HINT: THINK OF ASCII NUMBERS AS
POINTS."
130 GOSUB 1000
140 PRINT "[CLR]"
150 PRINT: INPUT "HOW MANY LETTERS IN THE WORD";L
160 PRINT: PRINT TAB(10)"PLAYER #1:"
170 DIM W$(L): DIM D$(L)
180 PRINT: PRINT "GIVE ME YOUR WORD."
190 INPUT W$
200 FOR X=1 TO L
210 D$(X)=MID$(W$,X,1)
220 IF MID$(W$,X,1)=" " THEN GOTO 240
230 NEXT X
240 FOR X=1 TO L
250 N=ASC(D$(X))
260 P1=P1+N
270 NEXT X
280 PRINT: PRINT "YOUR TOTAL SCORE IS" P1
290 PRINT: PRINT
300 PRINT TAB(10)"PLAYER #2:"
330 PRINT: PRINT "GIVE ME YOUR WORD."
340 INPUT W$
350 FOR X=1 TO L
360 D$(X)=MID$(W$,X,1)
370 IF MID$(W$,X,1)=" " THEN GOTO 380
380 NEXT X
390 FOR X=1 TO L
400 N=ASC(D$(X))
410 P2=P2+N
420 NEXT X
430 PRINT: PRINT "YOUR TOTAL SCORE IS" P2
440 PRINT: PRINT
450 IF P1>P2 THEN PRINT "PLAYER #1 WINS! PLAYER #1 GOES
FIRST."
460 IF P2>P1 THEN PRINT "PLAYER #2 WINS! PLAYER #2 GOES
FIRST."
470 IF P1=P2 THEN PRINT "IT'S A TIE!! PLAYER #1 GOES FIRST."
480 GOSUB 1000
490 CLR: GOTO 140
1000 PRINT: PRINT: PRINT
1010 PRINT "PRESS THE LETTER 'R' WHEN YOU ARE READY."
1020 GET R$
1030 IF R$="R" THEN RETURN
1040 IF R$<>"R" THEN GOTO 1020
```


23-3

```

5 REM ENGRAVER'S PROGRAM
10 PRINT "[CLR]"
20 PRINT "PLEASE INPUT WHAT YOU WANT ENGRAVED."
30 INPUT M$
40 L=LEN(M$)
50 FOR X=1 TO L
60 IF MID$(M$,X,1)=" " THEN GOTO 80
70 T=T+1
80 NEXT X
90 PRINT
100 PRINT "THERE ARE" T "LETTERS IN YOUR MESSAGE."
110 PRINT
120 PRINT "THAT WILL COST $" T*.25
130 END
    
```

24-1

```

5 REM CURRENCY EXCHANGE FOR PESOS OR FRANCS
10 PRINT "[CLR]"
20 PRINT TAB(10)"CURRENCY EXCHANGE"
30 PRINT "INPUT PESOS OR FRANCS LIKE THIS:"
40 PRINT TAB(20)"15 PESOS"
50 PRINT TAB(20)"20 FRANCS"
60 PRINT "I WILL TELL YOU HOW MUCH THAT IS"
70 PRINT "WORTH IN U.S. CURRENCY."
80 PRINT: INPUT M$
90 N=VAL(M$)
100 FOR X=1 TO 50
110 IF MID$(M$,X,1)=" " THEN GOTO 130
120 NEXT X
130 IF MID$(M$,X+1,1)="P" THEN GOTO 150
140 IF MID$(M$,X+1,1)="F" THEN GOTO 170
150 P=N/150
160 PRINT M$ " IS WORTH" P "AMERICAN DOLLARS.": GOTO 80
170 F=N/705
180 PRINT M$ "IS WORTH" F "AMERICAN DOLLARS.": GOTO 80
    
```

24-2

```

5 REM REVERSE AND ADD
10 PRINT "[CLR]"
20 PRINT: PRINT "INPUT ANY NUMBER"
30 INPUT N1
40 A$=STR$(N1)
50 L=LEN(A$)
60 FOR X=L TO 1 STEP -1
70 B$=B$+MID$(A$,X,1)
80 NEXT X
90 N2=VAL(B$)
100 PRINT: PRINT N1 "+ " B$ "=" N1+N2
110 CLR: GOTO 20
    
```

APPENDIX

ERROR MESSAGES

Listed here are some error messages you may encounter as you program. After the error message, there is a brief description of the cause and a suggestion for a remedy.

The Commodore® 64 prints an error message and also tells you in which program line that error is occurring. **LIST** that line to see what it is the computer does not understand.

BAD DATA

PROBLEM — String data was received from an open file, but the program was expecting numeric data.

SOLUTION — Check the program to see if all of the string variables have a \$ after them and that the numeric variables do not have a \$.

BAD SUBSCRIPT

PROBLEM — The program tried to reference an element of an array whose number is outside of the range specified by the DIM statement. The 64 will also give this error message if there is incorrect spacing with a TAB or SPC command.

SOLUTION — If you see this error message when you RUN one of your programs, check for incorrect spacing in a TAB or SPC statement. Be sure the command is written TAB(67) or SPC(67) with no space between the command and the first parenthesis. If you have used an array in your program, check to see that you have not used more elements than the number in your DIM statement.

BREAK

PROBLEM — The program has been interrupted because you pressed the STOP key.

SOLUTION — Type RUN to run the program from the beginning or type CONT to continue running the program from where it was stopped.

CAN'T CONTINUE

PROBLEM — The CONT command will not work. The program may never have been RUN, an error has occurred, or while the program was stopped you have edited a line.

SOLUTION — Type RUN to begin the program from the beginning. If an error has stopped the program, that error will need to be corrected before the program will RUN successfully.

DEVICE NOT PRESENT

PROBLEM — The device (disk drive, cassette recorder, printer, etc.) you have asked for is not available.

SOLUTION — Check to see that the device is plugged in, turned on, and connected correctly to the computer. Retype the command you are trying to execute.

DIVISION BY ZERO

PROBLEM — Division by zero is not allowed in mathematics.

SOLUTION — Check the computation to see where you have asked the computer to divide by zero. Also, whenever a divisor in a calculation is the result of another calculation, write an IF-THEN statement to trap any resulting zero before it becomes a divisor.

EXTRA IGNORED

PROBLEM — A comma or dash was typed in response to an INPUT statement. Only the data up to that punctuation mark was accepted by the program.

SOLUTION — If the comma or dash is an important part of the INPUT response, press STOP and RESTORE together to BREAK the INPUT program. Then RUN the program from the beginning and use quotation marks around the INPUT statement.

FILE NOT FOUND

PROBLEM — On tape, an END-OF-TAPE marker was found.

SOLUTION — Rewind the tape and try again.

PROBLEM — On disk, no file with that name exists.

SOLUTION — Type LOAD "\$",8 and then LIST to see a directory of the programs recorded on the disk. Check to see that you spelled the file name correctly.

FILE NOT OPEN

PROBLEM — You used a CLOSE, CMD, PRINT#, INPUT# or GET# command, but the file was not opened first.

SOLUTION — Type an OPEN statement to open the file and try your command again.

FILE OPEN

PROBLEM — You have already given the computer an OPEN statement using that same number.

SOLUTION — Check to be sure you have opened the file you wanted.

FORMULA TOO COMPLEX

PROBLEM — The string expression being used is too long or a formula has too many parentheses.

SOLUTION — Divide the string expression into at least two parts. If you are using a formula, change it in such a way that some of the parentheses can be eliminated.

ILLEGAL DIRECT

PROBLEM — You have tried to use an INPUT statement as a direct command.

SOLUTION — Use the INPUT statement as a part of a numbered program line.

ILLEGAL QUANTITY

PROBLEM — A number used in a function or statement is out of the allowable range.

SOLUTION — Check to see if you have used a number larger than 255 with a TAB or SPC function or have used a number larger than a POKE or other command will allow.

REDIM'D ARRAY

PROBLEM — An array has been DIMensioned more than once.

SOLUTION — Check to see that you have not used a GOTO statement which has sent the program back to a DIM statement. The array can be used again, but the computer can only DIM it once. If you are using several different arrays, make sure each one has a unique name and each one has a DIM statement.

REDO FROM START

PROBLEM — Letters or symbols were entered as a response to an INPUT statement when numeric data was expected.

SOLUTION — Retype the entry so that it is correct, and the program will continue to run.

RETURN WITHOUT
GOSUB

PROBLEM — The computer reads a RETURN statement and no GOSUB command has been issued.

SOLUTION — Check to see that the END command is placed between the main program lines and the subroutines that follow.

STRING TOO LONG

PROBLEM — A string can contain up to 255 characters.

SOLUTION — Check the string length. See if it can be subdivided.

SYNTAX

PROBLEM — The computer does not recognize a statement.

SOLUTION — Check for a spelling error, a missing or extra parenthesis, or a misplaced punctuation mark or any other possible typing error.

- TYPE MISMATCH** **PROBLEM** — This error occurs when a number is used in place of a string, or vice versa.
SOLUTION — Check to see that if you asked for a string, quotation marks are used on the other side of the equal sign. If you asked for a numeric variable, no quotation marks should be used on the other side of the equal sign.
- UNDEF'D STATEMENT** **PROBLEM** — An attempt was made to GOTO or GOSUB or RUN a line number that doesn't exist.
SOLUTION — Check the program to see if you have mistyped a number or if you have forgotten to include a subroutine.
- VERIFY** **PROBLEM** — The program on tape or disk does not match the program currently in memory.
SOLUTION — Check to see that the peripheral device is connected correctly to the computer. If using a tape, rewind the tape. Try the SAVE command again. If using a disk, SAVE the program again.

APPENDIX

GLOSSARY

ABS	Stands for ABSolute value, a mathematical function which returns the absolute value of a number which is always the positive form of that number.
AND	Used with IF-THEN statements to set further conditions which require that all parts of the equation be true. (e.g. IF A=Z AND B=Y THEN GOTO 100)
Array	A set of memory spaces to store information.
ASC	The function used to find the ASCII number of a keyboard character.
ASCII	Stands for American Standard Code for Information Interchange. Each keyboard character is assigned one number.
Attack/Decay	The POKE memory location for SID which sets the speed at which the sound reaches its highest volume.
BASIC	The language your computer understands. It is an acronym and stands for Beginner's All-purpose Symbolic Instruction Code.
Bit	The smallest unit of information a computer uses. A bit is either the digit "0" or "1."
Bytes	Used to measure the size of a computer's memory. One byte contains eight bits. The 64 has 38,911 bytes available for your use. A letter in a word generally takes up 1 byte, so "the" would occupy 3 bytes.
Cartridge	A device that stores a prerecorded program. A cartridge fits into a special slot built into the computer.

CHR\$	Stands for character string. Returns the string character when given an ASCII code.
CLR	Command used to set the value of all variables in a program to zero or all string variables to null.
Colon :	Used to put two or more commands on one program line.
Comma ,	Used with a PRINT command, this lets you print in columns on the screen.
CONT	Tells the computer to continue with a program after you stopped it by pressing the STOP key.
Cursor	The blinking square that shows you where you are on a line.
DATA	Command which is followed by a list of items, separated by commas, to be used by a READ statement.
DIM	Stands for DIMension. Tells the computer how many elements you want your array to have.
Disk	A circular shape of magnetic material enclosed within a protective covering which is used to save and load computer programs.
Disk drive	A device that reads information from a disk and copies it into the computer's memory and also writes information from the computer's memory onto a disk.
Edit	A way to correct mistakes without retyping the whole line. Using the SHIFT key, the INST/DEL key and the two CURSOR CONTROL keys, you can make corrections in program lines.
Element	The number of "boxes" in an array. Without a DIM statement, an array will have 11 elements.
END	Tells the computer to stop the program and go back to being ready for your commands.

Flag	Numbers or letters at the end of a DATA statement which, when combined with an IF-THEN statement, will stop the computer from showing an OUT OF DATA error.
FOR	Command used with the NEXT command to program a loop to get the computer to do something a certain number of times.
GET	Command used so that a person running the program can press a key as a response while the program is running.
GOSUB	Command that tells the computer to go to a subroutine. The computer goes to a line in the program and follows the instructions there. (e.g. GOSUB 500) At the end of the subroutine, you must use the command, RETURN. The computer then returns to the main program where it left off.
GOTO	Command that sends the computer to a certain line in the program.
Graphics	Pictorial displays using lines, shapes and symbols to create designs, charts or graphs.
Hardware	The physical parts of a computer system. (e.g. keyboard, printer, monitor)
IF	Command used with THEN to set up a condition. (e.g. IF X=2 THEN GOTO 10)
INPUT	Command used so that a person running a program can enter information into the computer while the program is running.
INT	Command that rounds numbers to the closest integer.
LEFT\$	Command that tells the computer to look at single letters in a word, starting at the left side of the string.
LEN	Command that tells the number of characters in a string.

LET	An optional word that can be used before an equation. (e.g. LET C=5)
LIST	A direct command which tells the computer to show the program lines in its memory.
LOAD	Command used to transfer a program from a disk or tape to the computer's primary memory.
MID\$	Command that tells the computer to look at certain characters in a string.
Monitor	A device for visually displaying a computer program on a screen. Also called a CRT (cathode ray tube). A television may be used as a monitor with most brands of microcomputers.
NEW	A direct command that erases old programs in the computer's memory.
NEXT	Command used with a FOR statement to form a loop to get the computer to do something a certain number of times.
ON GOSUB	Command that branches to several subroutines, depending on the INPUT response.
ON GOTO	Command that branches to several program lines, depending on the INPUT response.
OR	Used with IF-THEN statement to set further conditions which require that one of the parts of the equation be true. (e.g. IF X="YES" OR X="Y" THEN GOTO 250)
PEEK	Command used to look at what is stored at a specific memory location.
Peripheral	A hardware accessory for a computer. (e.g. disk drive, printer)
POKE	Command used to store information at a specific memory location.

PRINT	Command that tells the computer to print something. It can be used as a direct command or as a program command.
Programming	Using program lines and commands to tell the computer what to do.
RAM	Stands for Random Access Memory. An area in the computer where information is stored. If you do not save this information on disk or tape, it will be lost when the computer is turned off.
READ	Command used to read items in a DATA statement and assign each item to a variable.
REM	A command to help you remember something about a program. This is a REMark statement and is in the listing of a program but the computer ignores it when you run the program.
RESTORE	Command that moves the counter back to the first value in the DATA statements.
RETURN	Key that is used to put a command or program line into the computer's memory. Also, a command used at the end of a subroutine to cause the computer to RETURN to the main program line just after the GOSUB it left.
RIGHT\$	Command that tells the computer to look at single letters in a string, starting at the right side of the string.
RND	Command that tells the computer to pick a random number.
ROM	Stands for Read Only Memory. This is the permanent memory built into the computer by a manufacturer. You cannot change this memory, you can "only read" it.
RUN	Command that tells the computer to follow the program in its memory.

SAVE	Command used to record a program on a disk or tape.
Scientific notation	A way to write very large or very small numbers. (e.g. $1000000=1 \times 10^6$. The computer will write 1000000 as 1E+06.)
Semicolon ;	Used with a PRINT command, this will tell the computer to print on a line until that line is full.
SID	The Sound Interface Device or sound chip.
Software	Computer programs, tapes, disks or cartridges.
SPC	Used with a PRINT command to leave a specified number of spaces. (e.g. PRINT SPC(5) "HELLO")
STEP	Used with a FOR command to tell the computer what size increment to use. (e.g. FOR X=1 TO 100 STEP 2)
String variable	A letter with a dollar sign (e.g. A\$) to which you can assign letters, words, symbols or punctuation marks.
STR\$	Command used to change a number value into a string variable.
Subroutine	A set of independent commands within a program. The computer is sent to a subroutine with a GOSUB command, it follows the commands within the subroutine, and is then sent back to the main program with a RETURN command.
Subscript	The number of each element in an array. In the statement, G\$(5)="CUP", 5 is the subscript.
Sustain/Release	The POKE memory location for SID which tells how loud the note will stay until turned off and determines how quickly the note will fall to zero volume.
TAB	Used with a PRINT command to set horizontal tabs. (e.g. PRINT TAB(6) "GOODBY")

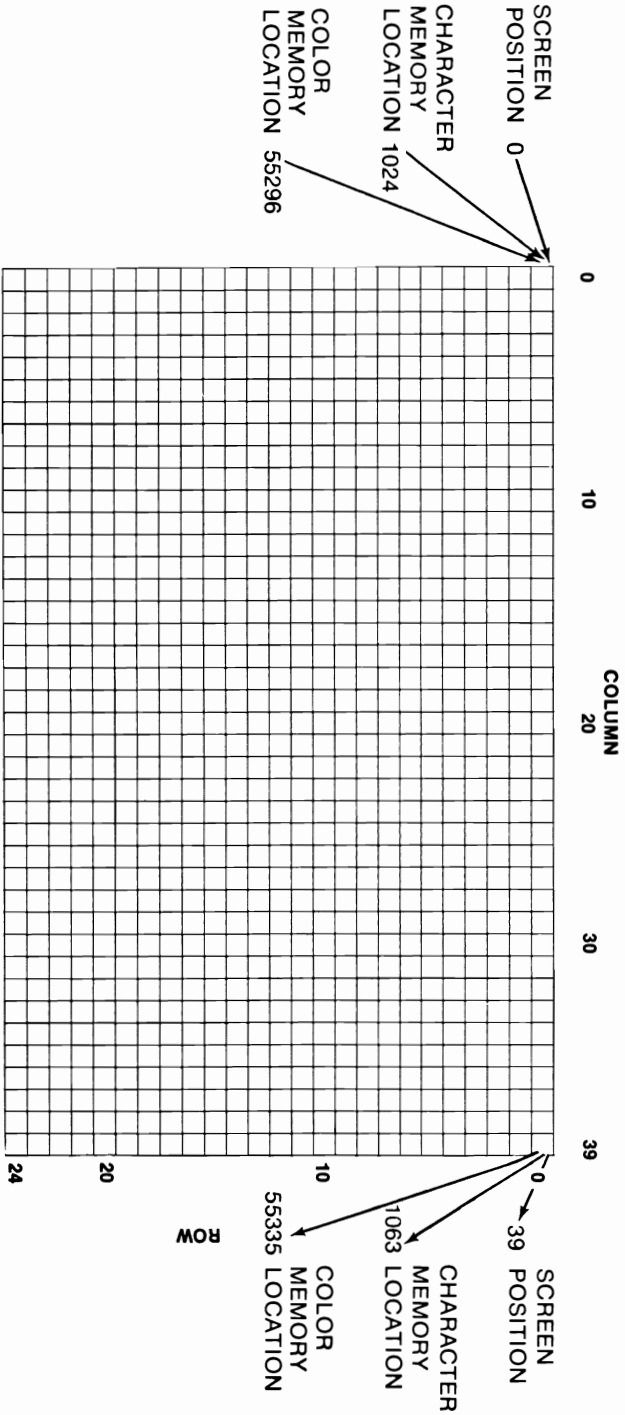
Text	Words, letters and numbers that appear on a monitor screen.
THEN	Command used with IF to set up conditions. (e.g. IF A=B THEN GOTO 100)
VAL	Command used to change a string into a number value.
Variable	A letter to which you can assign values.
VERIFY	Command used to tell the computer to compare the program in the computer's memory with the program on tape or disk.
Waveform	The POKE memory location for SID which sets the "shape" of the sound; it can be set at 17, 33, 65 or 129.
Word processor	A program that allows the user to write and edit text. The text can be saved on disk or tape and printed. The user can make changes in the same text without retyping the entire document.

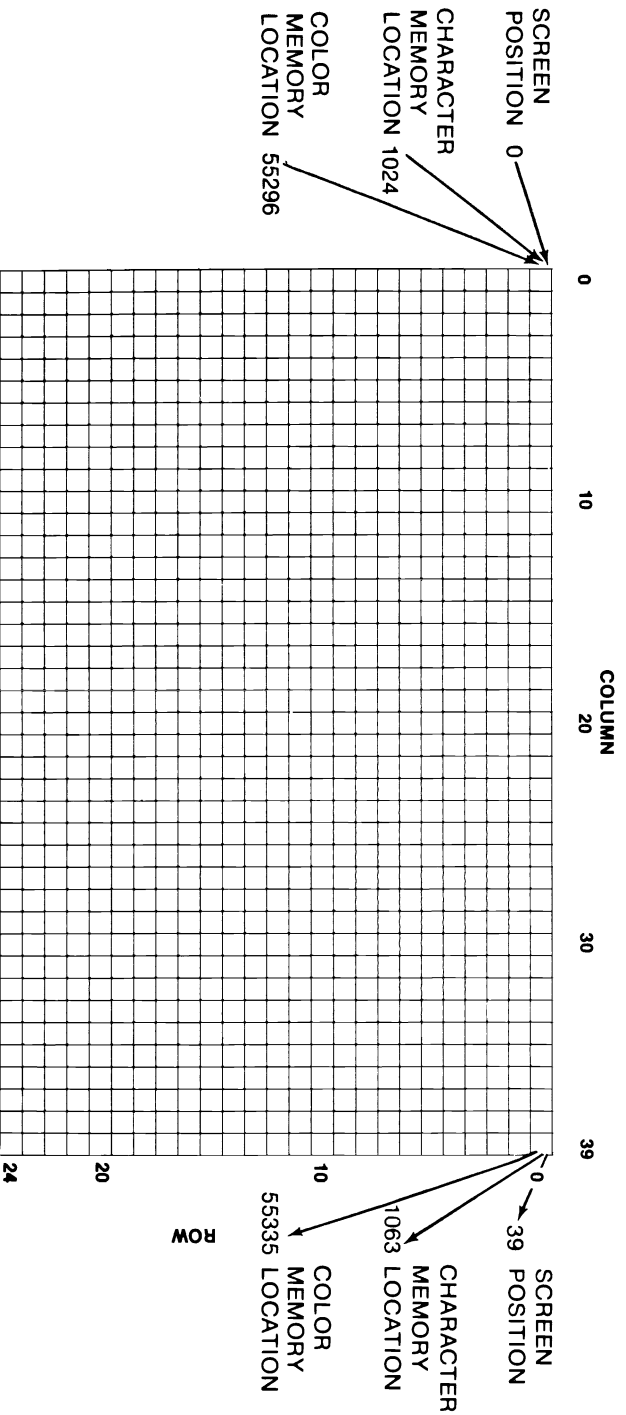
APPENDIX

MUSICAL NOTES

NOTE	OCTAVE	HIGH FREQUENCY	LOW FREQUENCY
C	4	16	195
C#	4	17	195
D	4	18	209
D#	4	19	239
E	4	21	31
F	4	22	96
F#	4	23	181
G	4	25	30
G#	4	26	156
A	4	28	49
A#	4	29	223
B	4	31	165
C	5	33	135
C#	5	35	134
D	5	37	162
D#	5	39	223
E	5	42	62
F	5	44	193
F#	5	47	107
G	5	50	60
G#	5	53	57
A	5	56	99
A#	5	59	190
B	5	63	75
C	6	67	15

NOTE	NAME	COUNT
	sixteenth	.25
	dotted sixteenth	.375
	eighth	.5
	dotted eighth	.75
	quarter	1
	dotted quarter	1.5
	half	2
	dotted half	3
	whole	4





INDEX

A

ABS function 9
Alphabetizing 115-116
Animation (see Graphics)
Arrays 16, 18, 45
ASC function 107
ASCII 105

B

Bad subscript error 17

C

CHR\$ function 107
Clear screen statement 1
CLR statement 10
Color
 memory locations 57, 62
 memory map 56
 POKE codes 57
 screen and border 60
Concatenation 119
Control key 2

D

DATA statement 29-33, 35
DIM statement 15, 17, 46

E

Editing programs 1
Element of an array 26
Error messages 157

F

Flag 34

G

Graphics tablet 127
Graphics using POKE
 animation 73-78
 character codes 58-59
 character memory locations 56,
 60, 62
 color codes 57
 color memory locations 56, 60
Graphics worksheets 171-172

I

Illegal quantity error 84
INT function 4, 7

J

Joystick 127

L

LEFT\$ function 110
LEN function 118

M

MID\$ function 112, 113
Modem 128-131
Music (see Sound)

O

ON GOSUB statement 52
ON GOTO statement 52
Out of data error 33

P

PEEK 77
Peripherals
 graphics tablet 127
 joystick 127
 modem 128-131
 mouse 128
 printer 131-133
 speech synthesizer 128
 touchscreen 128
POKE 56
PRINT statement 1
Printer 39, 131-133

R

Random numbers (see RND function)
Random whole numbers 5
READ statement 29-33
RESTORE key 61, 74, 91
RESTORE statement 41
RETURN key 1
RIGHT\$ function 117
RND function 2
RUN/STOP key 33, 61, 74, 91

S

SAVE command 98
SHIFT KEY 1
Sound
 attack/decay 85, 92
 duration 85, 93
 harmony 100-102
 high frequency 82
 low frequency 82
 music note chart 170
 SID 81, 87, 89
 sustain/release 92
 volume 81, 92
 waveform 83
Speech synthesizer 128
STOP key 33, 61, 74, 91
STR\$ function 124
Subscript of an array 26

T

Touchscreen 128

V

VAL function 122



On the inside back cover, the most often used computer commands are printed on a reference card. You may wish to remove this card from the book and keep it close to your computer for handy reference.

For the Commodore 64

Everyone in the family will be able to master the commands that control your COMMODORE 64. It is a hands-on program you use with your computer. You will quickly know if you have understood a concept or mastered a command. The two book set aims at developing computer awareness and programming skills on two levels.

The first book, level one, introduces you to the computer and the BASIC language it speaks. It is a slow-paced, easy presentation the whole family will enjoy. You will learn the following commands:

NEW, PRINT, RUN, LIST, TAB, SPC, OPEN, SAVE, LOAD, VERIFY, CONT, STEP, COLON, SEMI, COMMA, TAB, INPUT, IF THEN, FOR NEXT, GOSUB.

After completing level one, you will be able to:

- Find your way around the various components of your system.
- Save, change, and add to your program.
- Use keyboard graphics to design a birthday card and calendar.
- Know what to look for when you buy software for word processing, home finances, education at home.
- Train your computer to do arithmetic.
- Set up a checkbook balance and loan guide.
- Make up your own math, spelling and word games.

Level two, the second book, shows how to train your COMMODORE 64 to do complex and rapid work. Level two book introduces you to several new commands:

ARRAYS, DIM, DATA, READ, PEEK, POKE, LEFT\$, MID\$, RIGHT\$, LEN(X\$), VAL, STR\$, INT, RND, ON GOSUB, ON GOTO.

These commands and the other concepts taught in level two will allow you to fully appreciate the abilities of the COMMODORE 64. Here is what you will be able to do:

- Improve your inventory, bank account, and loan guide programs with new advanced programming concepts.
- Design a secret code game, flash card program, and play computer solitaire.
- Animate your color graphics.
- Set up bar graphs and moving picture designs.
- Write your own computer music and add sound effects to your graphics.
- Alphabetize a name and address list.
- Use debugging techniques to find out what is wrong with a program.

In addition, the level two book tells about printers and other equipment for your COMMODORE 64.

TRAINING YOUR COMMODORE 64: Master the Commands that Control Your Computer, Level 1 & Level 2.

This two book set is an introduction to the exciting world of computers. You will find how gaining computer competence can lead to self confidence in this fast-moving electronic age.

Already established as a leader in the field, creative programming has developed a practical guide for children as well as adults. We make familiarizing yourself with computers a relaxed, fun-filled adventure. In addition, we offer helpful and educational applications for all members of the family.

Using a hands-on approach to computer competency, the step-by-step lessons, examples, applications and reviews are so easy to understand that you can learn right in your own home, at your own pace, without any other instruction or tutor.

You will learn how computers "work," behave, and how you can train them to work and play for your benefit and satisfaction.

These books represent a newly developed edition of Creative Programming's already well known school textbooks. Enhanced and refined for use at home or at work, they still are solidly built on a foundation that has trained over 500,000 children and adults in over 10,000 schools and learning centers.

"Creative programming is the best self-directed computer instructional booklet I have seen on the market. It has an efficient step-by-step approach that starts at ground level. I have used this program with students and adults of all levels and abilities. It works for everyone."

DEBBIE TENENBAUM
Teacher, Smyra, Georgia

"The most comprehensive and well-done materials available to teach programming to grades 4 to adult."

JOAN TROUTNER
Columnist, Educational Computer

"I have sat for hours trying to learn something from the book that comes with my computer. I learned more in one day with your books than I did in two months. They are fantastic!"

JIM McCREERY
Nags Head, North Carolina - Home Computer Owner

PUBLISHED BY:

WOODSTOCK SOFTWARE COMPANY

Woodstock Square — Box 886
Woodstock, Illinois 60098

0-912079-50-9

