

Revised

COMPUTE!'s Kids and the Commodore 64



Edward H. Carlson

Illustrated by Paul D. Trap

A fun, easy-to-use guide to learning BASIC
on the Commodore 64. For kids and adults—
for any Commodore 64 user.

A **COMPUTE!** Books Publication

\$12.95

U
U
U
U
U
U
U

U
U
U
U
U
U
U

COMPUTE!'s Kids and the Commodore 64



Edward H. Carlson
Illustrated by Paul D. Trap

COMPUTE! Publications, Inc. 
One of the ABC Publishing Companies
Greensboro, North Carolina

Copyright 1984, COMPUTE! Publications, Inc.

All rights reserved

Reproduction or translation of any part of this work beyond that permitted by Sections 107 and 108 of the United States Copyright Act without the permission of the copyright owner is unlawful.

Printed in the United States of America

ISBN 0-942386-77-9

10 9 8 7 6 5 4 3 2 1

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is one of the ABC Publishing Companies and is not associated with any manufacturer of personal computers. Commodore 64 and VIC-20 are trademarks of Commodore Electronics Limited.

Contents

Acknowledgments	v
To the Kids	vii
To the Parents	viii
To the Teacher	ix
About Programming	x
About the Book	xi

Introduction

1. NEW, PRINT, REM, and RUN	1
2. Color and the Keyboard	10
3. LIST, Boxes in Memory	17
4. The Cursor Keys and Drawing Pictures	25
5. Tricks with PRINT	32
6. The INPUT Statement	39
7. The LET Statement, Gluing Strings	44
8. The GOTO Statement and the RUN/STOP Key	50
9. The IF Statement	59
10. Introducing Numbers	66
11. TAB and Delay Loops	74
12. The IF Statement with Numbers	80
13. Random Numbers and the INT Function	87
14. Save to Disk	94

Graphics, Games, and All That

15. Some Shortcuts	101
16. Moving Pictures	111
17. FOR-NEXT Loops	117
18. DATA, READ, RESTORE	123
19. Sound Effects	129
20. Names, Clocks, and Modes	137
21. Color Graphics	145
22. POKEing Graphics	152
23. Secret Writing and the GET Statement	159
24. Pretty Programs, GOSUB, RETURN, END	165
25. Logic: AND, OR, NOT	172

Advanced Programming

26. Snipping Strings: LEFT\$, MID\$, RIGHT\$, LEN	180
27. Switching Numbers with Strings	187

28. Action Games and the Function Keys	193
29. Music	203
30. Arrays and The DIM Statement	210
31. Sprites	219
32. User-Friendly Programs	229
33. Debugging, STOP, CONT	236
Appendix	
Disk Usage	243
Reserved Words	246
Answers to Assignments	247
Glossary	266
Error Messages	278
Topical Index	282
Commands and Functions Index	285

Acknowledgments

My sincere thanks go to Paul Sheldon Foote for suggesting I write a book on teaching BASIC to children.

This book is sixth in a series that started with *Kids and the Apple*. Each book has been written to fit the strengths of the computer in question and modified in response to what I have learned about teaching children.

I am deeply grateful to my fellow staff members at the Michigan State University "Computer Camp": Mark Lardie, Mary Winter, John Forsyth, and Marc Van Wormer, each of whom shared their experiences with me and helped provide insight into the minds of the children.

Mary Winter's vast knowledge of Commodore and skill in presenting computing topics to children have been especially helpful to me on many occasions.

Several families have used the Apple version of this book in their homes and offered suggestions for improvement. I especially wish to thank George Campbell and his youngsters, Andrew and Sarah; Beth O'Malia and Scott, John, and Matt; Chris Clark and Chris Jr., Tryn, Daniel, and Vicky; and Paul Foote and David.

COMPUTE! Publications started publishing *COMPUTE!* magazine at about the same time that I started writing articles on home computing. I am grateful that COMPUTE! encouraged my writing career by accepting some of my early articles. This encouragement continues with the publishing of the books from the Kids and . . . series.

This book was originally published by Datamost, Inc. I greatly appreciate the skill and energy of Stephen Levy in editing and assembling this book for COMPUTE! Publications, Inc.

Paul Trap shares the title page honors with me. His drawings are an essential part of the book's teaching method. I am grateful to Paul for his lively ideas, cheerful competence, and quick work which make him an ideal workmate.

My children have worked on this book in many ways, from typing and testing programs to proofreading and indexing. In addition, they attempted to help the "bald-headed one" to properly express juvenile taste. I thank Karen, Brian, and Minda for their essential help.

My final and heartfelt thanks go to my wife, Louise. As absorbed in professional duties as I, she nevertheless took on an increased share of family duties as the book absorbed my free time. Without her support I could not have finished the work.

To the Kids

This book teaches you how to write programs for the Commodore 64 computer.

You will learn how to make your own action games, board games, and word games. You can entertain your friends with challenging games and provide some silly moments at your parties with the short games you invent.

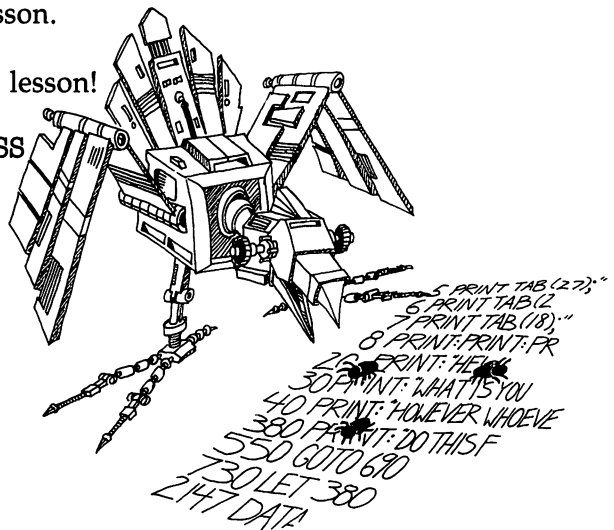
Perhaps your record collection or your paper route needs the organization your special programs can provide. If you are working on the school yearbook, maybe a program to handle the finances or records would be useful.

You may help your younger sisters and brothers by writing drill programs for arithmetic facts or spelling words. Even your own schoolwork in history or a foreign language may be made easier by programs you write.

How to use this book. Do all the examples. Try all the assignments. If you get stuck, first go back and reread the lesson carefully from the top. You may have overlooked some detail. After trying hard to get unstuck by yourself, you may go ask a parent or teacher for help.

There are review questions for each lesson. Be sure you can answer them before announcing that you have finished the lesson!

**MAY THE BLUEBIRD OF HAPPINESS
EAT ALL THE BUGS IN YOUR
PROGRAMS!**



To the Parents

This book is designed to teach BASIC on the Commodore 64 to youngsters from 10 to 14 years old. It gives guidance, explanations, exercises, reviews, and quizzes. Some exercises have room for the student to write in answers that you can check later. Answers are provided in the back of the book for program assignments.

Your child will probably need some help in getting started and a great deal of encouragement at the sticky places. For further guidance, you may wish to read my article in *Creative Computing*, April 1983, p. 168.

Learning to program is not easy because it requires handling some sophisticated concepts. It also requires accuracy and attention to detail, which are not typical childhood traits. For these reasons, it is a valuable experience for children. They will be well rewarded if they can stick with the book long enough to reach the fun projects that are possible once a repertoire of commands is built up.

How to use this book. The book is divided into 33 lessons for the kids to do. Each lesson is preceded by a "Notes" section which you should read. It outlines the things to be studied, gives some helpful hints, and provides questions which you can use verbally (usually at the computer) to see if the skills and concepts have been mastered.

These notes are intended for the parents, but older students may also profit by reading them. Younger students will probably not read them, and can get all the material they need from the lessons. For the youngest children, it may be advisable to read the lesson out loud with them and discuss it before they start work.



To the Teacher

This book is designed for students in about the seventh grade. It teaches BASIC and the features of the Commodore 64.

The lessons contain explanations (including cartoons), examples, exercises, and review questions. Notes for the instructor, which accompany each lesson, summarize the material, provide helpful hints, and give review questions.

The book is intended for independent study, but may also be used in a classroom setting.

I view this book as teaching programming in the broadest sense, *using* the BASIC language, rather than *teaching* BASIC. Seymour Papert has pointed out in *Mindstorms* (Basic Books, 1980) that programming can teach powerful ideas. Among these is the idea that procedures are entities in themselves. They can be named, broken down into elementary parts, and debugged. Some other concepts include "chunking" ideas into "mind-sized bites," organizing such modules in a hierarchical system, looping to repeat modules, and conditional testing (the IF-THEN statement).

Each concept is tied to the student's everyday experiences through choice of language to express the idea, through choice of examples, and through cartoons. Thus, metaphor is utilized in making the "new" material familiar to the student.



About Programming

There is a common misconception about programming a computer. Many people think that ability in mathematics is required. Not so. The childhood activities that computing most resembles may be playing with building blocks and writing an English composition.

Like a block set that has many copies of a few types of blocks, BASIC uses a relatively small number of standard commands. Yet the blocks can be formed into a unique and imaginative castle, and BASIC can be used to write an almost limitless variety of programs.

Like an essay on the theme "How I Spent My Summer," writing a program involves skill and planning on all scales. To write a theme, a child must organize thoughts on several scales, from the overall topic, to lead and summary paragraphs and sentences, on down to grammar, punctuation, and spelling of each word in the sentences.

Creativity in each of these activities—blocks, writing, and BASIC—has little scope at the smallest level—individual blocks, words, and commands. At best, a small bag of tricks is developed. For example, the child may discover that the triangle block, first used to make roofs, makes splendid fir trees. What is needed at the small scale is accuracy in syntax. Here, computing is an almost ideal self-paced learning situation, because syntax errors are largely discovered and pointed out by the BASIC interpreter as the child builds and tests the program.

On a larger scale, creativity comes into full scope and many other latent abilities of the child are developed. School skills such as arithmetic and language arts are utilized as needed, and thus strengthened. But the strongest features of programming are balanced between analysis (why doesn't it work as I want) and synthesis (planning on several scales, from the program as a whole down through loops and subroutines to individual commands).

The analytical and synthetical skills learned in programming can be transferred to more general situations and can help the child to a more mature style of thinking and working.

About the Book

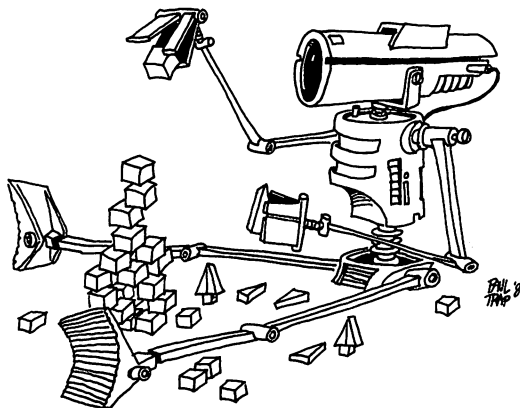
This book is arranged in 33 lessons, each with "Instructor Notes" and each containing assignments and review questions.

For instructors who feel themselves weak in BASIC or who are beginners themselves, the student's lessons form a good introduction to BASIC. The lessons and notes differ in style. The lessons are pragmatic and holistic; the notes and glossary are detailed and explanatory.

The book starts with a bare-bones introduction to programming, leading quickly to the point where interesting programs can be written. See the notes for lesson 6, "The INPUT Statement," for an explanation. The central part of the book emphasizes more advanced and powerful techniques. The final part of the book continues building on this, but also deals with broader aspects of the art of programming, such as editing, debugging, and user-friendly programming.

The assignments involve writing programs—usually short ones. Of course, many different programs are satisfactory solutions to these assignments. In the back of the book I have included solutions for assigned programs, some of them written by children who have used the book.

Lesson 14, "Saving to Disk," can be studied anytime after the third lesson.



U
U
U
U
U
U
U

U
U
U
U
U
U
U

Instructor Notes 1. NEW, PRINT, REM, and RUN

This lesson is an introduction to the computer. There are many minor questions your student may have at the start, and you should pull up a chair and help in the familiarization.

If something goes wrong and all else fails, tell your student to turn off the computer, then turn it on and start again.

The light blue writing on a dark blue background may be hard to read. Instructions are given for making white letters. If these are still hard to read, then the instructions to POKE 53281,0 to get a black background may be followed.

The contents of the lesson:

1. Turning on the computer.
2. Typing versus entering commands or lines. The RETURN key.
3. The computer understands only a limited number of words.
4. REM puts remarks in the program.
5. What is a program? Numbered lines.
6. Clearing the screen.
7. White letters on a black background.
8. Memory can be cleared with NEW.
9. What is seen on the screen and what is in memory are different. (This may be a hard concept for the student to understand at first.)
10. RUN makes the computer go to memory, look at the statements in the lines (in order), and perform the statements.
11. You can skip numbers in choosing line numbers, and why you may want to do so.

Questions:

1. Write a program that will print your name.
 2. Make the program disappear from the TV screen but stay in memory.
 3. Run it.
-
-

-
-
4. Erase the program from memory.
 5. Clear the screen and write a program that prints HELLO.
 6. Make it run.
 7. Erase it from memory but leave it on the screen.
 8. How do you make the letters nice and white?

Lesson 1. NEW, PRINT, REM, and RUN

How to Get Started

Turn on the computer. You will see a message on the screen. The last word is READY.

Below READY is a flashing square. This square is called the *cursor*. When you see it flashing, it means the computer is ready for you to type something in.

Cursor means *runner*. The little square runs along the screen and shows where the next letter you type goes.

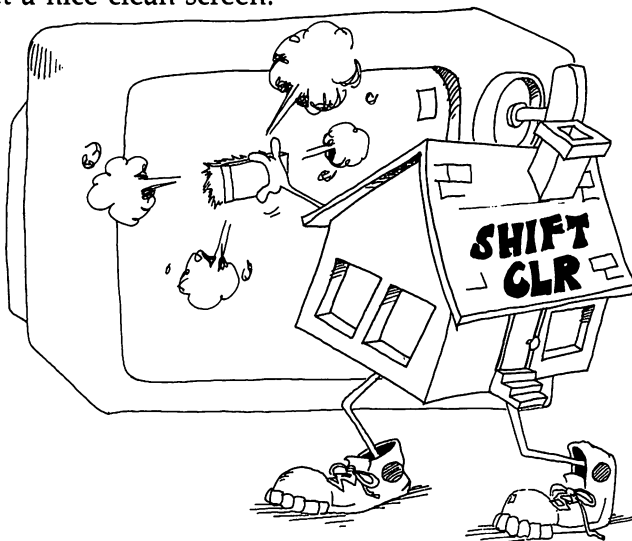
Typing

Type some things. What you type shows on the TV screen.

CURSOR, GO HOME!

The cursor's home is in the corner of the screen at the top. Find the CLR/HOME key (it's on the top row, right side of the keyboard) and press it. The cursor jumps to home.

Now type some more. You are writing over what is already on the screen. This is a mess. Let's get a nice clean screen.



Erasing the Screen

Two keys used together erase the screen.

Hold down one of the SHIFT keys and press the CLR/HOME key. The screen is erased.

CLR stands for clear. *Clear the screen* means the same as *erase the screen*.

Command the Computer

Try this.

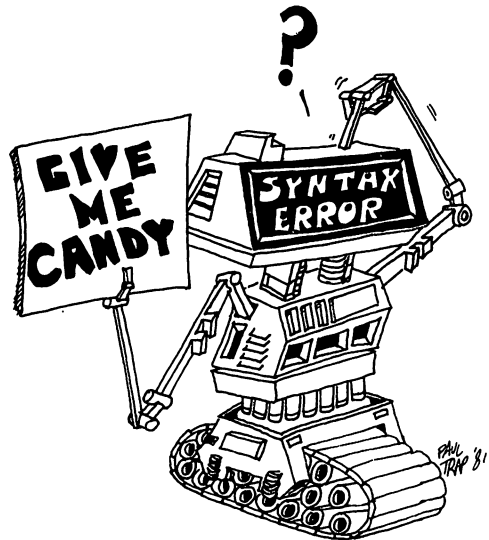
Type: GIVE ME CANDY

and press the RETURN key.

(If you make a mistake, press CLR/HOME and start over.)

The computer printed

```
?SYNTAX ERROR  
READY.
```



When the computer prints ?SYNTAX ERROR, it means the computer did not understand you.

The computer understands only about 70 words. You need to learn which words the computer understands.

Here are the first four words to learn:

NEW, PRINT, REM, and RUN.

Nice White Letters

If the letters on your TV screen are hard to read, try this:

Find the CTRL key.

Press it down and hold it down while you press the 2 key.

Now the letters you type will be pure white on a dark blue background.

Nice Black Background

If it is still hard to read the letters on the TV, try this:

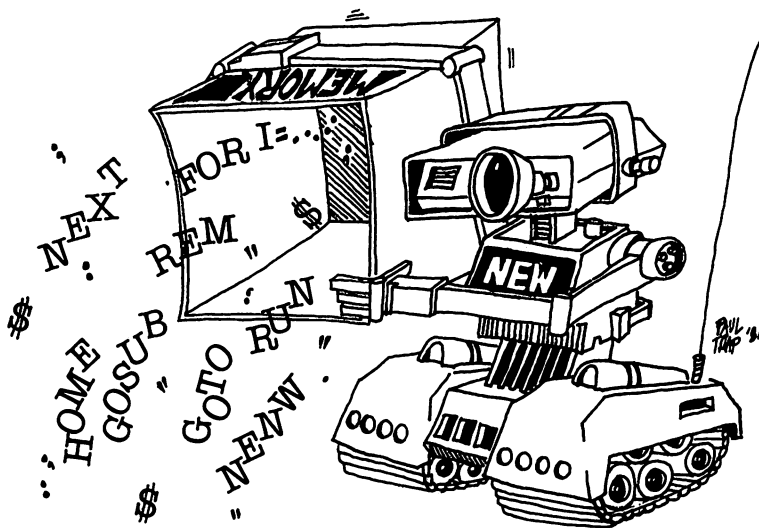
Type: POKE 53281,0 and press the RETURN key.

The background on the TV screen turns black. Adjust the TV controls to make the writing on the screen easy to read.

The NEW Command

Type: NEW and press RETURN.

NEW empties the computer's memory so you can put your program in it.



How to Enter a Line

When we say enter, we will always mean to do these two things.

1. Type a line.
2. Then press the RETURN key.

Clear the screen and enter this line:

```
10 PRINT "HI"
```

(The " marks are quotation marks. To make " marks, hold down the SHIFT key and press the key that has the 2 and the " on it.)

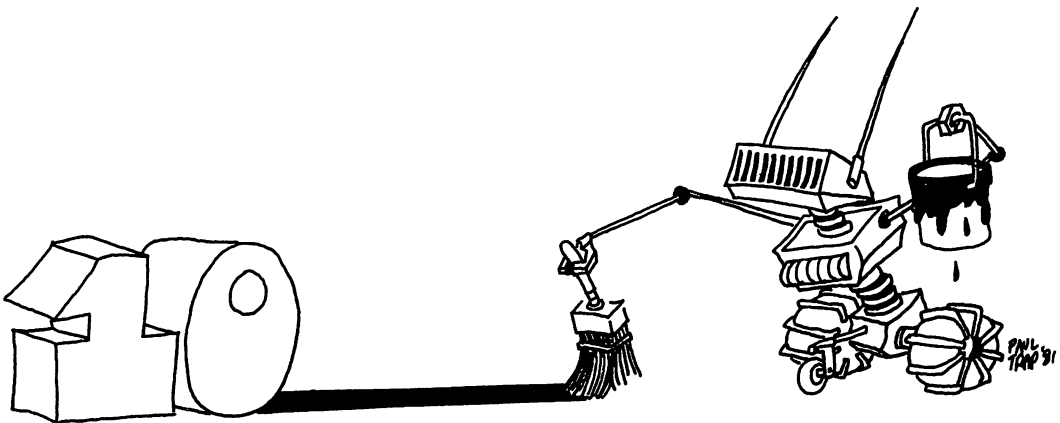
(Did you remember to press the RETURN key at the end of the line?)

Now the line number 10 is in the computer's memory.

It will stay in memory:

until you enter the NEW command, or
until you turn off the computer.

Line 10 is a very short program.



The Number 0 and the Letter O

The computer always writes the zero like this:

zero 0

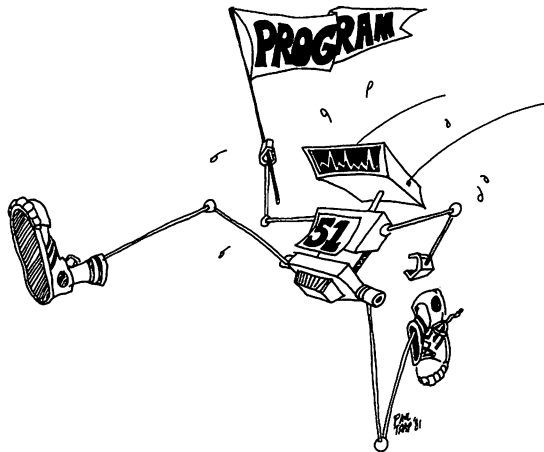
and the letter O like this:

letter O o

You have to be careful to do the same.

Right: 10 PRINT "HI"

Wrong: 1O PRINT "HI"



What Is a Program?

A program is a list of statements for the computer to do. The statements are written in lines. Each line starts with a number. The program you entered above has only one line.

How to Run a Program

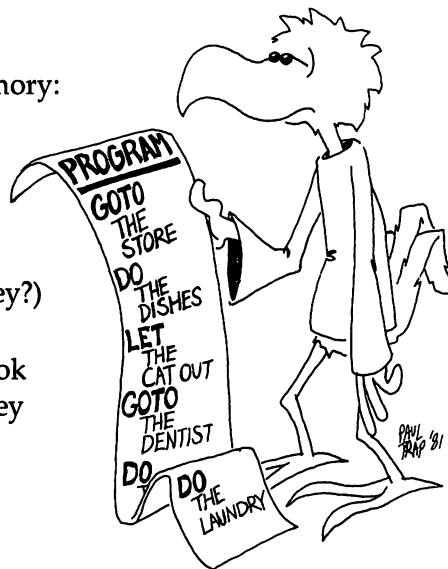
A moment ago you put this program in memory:

```
10 PRINT "HI"
```

Now enter: RUN

(Did you remember to press the RETURN key?)

The RUN command tells the computer to look in its memory for a program and then to obey the statements it reads in the lines.



Did the computer obey the PRINT statement? The PRINT statement tells the computer to print whatever is between the quotation marks. The computer printed:

HI

READY.

How to Number the Lines in a Program

Clear the screen.

Enter NEW and then this program:

```
1 REM HELLO
2 PRINT "HI "
3 PRINT "FRIEND "
```

This program has three lines. Each line starts with a statement. You have already learned the PRINT statement. We will tell you about the REM statement in a minute.

Usually you will skip numbers when writing the program.

Like this:

```
10 REM HELLO
20 PRINT "HI "
30 PRINT "FRIEND "
```

It is the same program but has different numbers. The numbers are in order, but some numbers are skipped. You skip numbers so you can add new lines in between the old lines if the program needs fixing later.

Run the program you have entered. The computer starts with the lowest line number and goes down the list in order and does what each line tells it to do.

The REM Statement

The REM statement is for writing little notes to yourself. The computer ignores the notes. Use REM for putting the name of your program in the top line of the program.

Assignment 1:

1. Use the CLR/HOME key to put the cursor home.
Now use the same key (SHIFTed CLR/HOME) to erase the screen.
2. Use the command NEW. Explain what it does.
3. Write a program that uses REM once and PRINT twice. Then use the command RUN to make the program obey the statements. Explain what it does.

Instructor Notes 2. Color and the Keyboard

The Commodore 64 has powerful color and graphics characters available from the keyboard. They provide plenty of "bells and whistles" to the student for increasing program richness.

Each key has up to three functions, chosen by just pressing the key, or by pressing it while holding down the SHIFT, CTRL, or the Commodore key. For colors, the CTRL key is held down while a color key (one of the number keys) is pressed.

The CLR/HOME key homes the cursor when pressed. (Home is the upper left corner of the screen.) Pressed with SHIFT, the CLR/HOME key erases the screen.

All these keys can be used in PRINT statements in a program. This gives the Commodore 64 some very powerful options, and several lessons in the book are devoted to exploiting them.

A white background is used in this lesson. Some colors do not show up well on white. In fact, for each color screen, there will be some colors that give blurry characters.

If you choose white for the screen color and also white for the letter color, you will see nothing when you type! Try POKE 53821,15 for a gray screen that will show letters of any color obtained from the color keys.

Pressing the CTRL and RVS ON keys at the same time gives reversed characters. The reverse of a *space* is a colored block. A useful way to make color bars for adjusting the color TV is to press CTRL and RVS ON, then hold down the space bar to make a colored bar. Repeat for each of the colors you get from CTRL and a color key. Then adjust your TV for best color. Yellow and perhaps purple are the most sensitive to proper adjustment.

Questions:

1. How do you do each of these things:

Make the computer type with red letters?

Erase the screen?

Empty the memory?

Print your name?

2. How do you change the screen background color to white?

3. What special key do you press to enter a line?

4. What does the computer mean when it prints SYNTAX ERROR?

5. How could you print FIRE with each letter in a different color?

Lesson 2. Color and the Keyboard

Turn your computer off, then on again. Now you're ready to start the lesson.

Making a White Background

Enter: POKE 53281,1

The screen turns white.

Careful! The number has to be exactly 53281. If you use another number by mistake, the computer may get "sick" and you will have to turn it off, then on, to make it well again.

Making Red Letters

Look at the 3 key. There are three things written on it.

The character 3 in the middle.

The character # at the top.

The word RED on the front of the key.

Here are the three things this key can do:

Press the 3 key. It prints 3.

Hold down the SHIFT key and press 3. It prints #.

Hold down the CTRL key and press 3. It changes the color!

(CTRL is short for control.
The CTRL key helps control
the color that is printed.)

The cursor is now red. Every
letter we type will be red, too.
Try it.



There are eight colors on the number keys. They are:

Black	BLK
White	WHT
Red	RED
Cyan (blue-green)	CYN
Purple	PUR
Green	GRN
Blue	BLU
Yellow	YEL

Type letters in different colors. Hold down the CTRL key and press one of the color keys.

Now type. Try each color.

Tune the TV for Good Colors

Tune the TV color controls so that the letters you typed are the right colors.

If you can make a good yellow the rest of the colors are probably okay.

Rainbow Letters

You can make the computer change colors in the middle of a program. Tell the computer what color by using a PRINT statement.

Enter:

```
10 REM RAINBOW
20 PRINT "... "
30 PRINT "POT"
40 PRINT "... "
50 PRINT "OF GOLD"
```

but don't put dots in line 20 or line 40.

In line 20 press CTRL and the BLK key instead of dots.

In line 40 press CTRL and the YEL key instead of dots.

It will look like this on the screen:

```
20 PRINT "■"  
40 PRINT "π"
```

Run the program. It should print POT in black letters and OF GOLD in yellow letters.

Oh, Oh! I'm in Trouble, Kids!

How can I tell you to press CTRL and a color key in a PRINT statement?

I can't just use dots because you will not know what color I mean.

I know! I will use capital letters inside braces like these { } for color keys.

Look: 20 PRINT "{CYN}"
means 20 PRINT "hold CTRL press CYN"
gives 20 PRINT "■" on the screen

but 20 PRINT "CYN"

means type the three letters C, Y, and N inside the quotation marks.

A Shortcut

Put the color key characters in the same PRINT statements as the words. The program looks like this:

In the book

```
10 REM RAINBOW  
20 PRINT "{GRN}POT"  
30 PRINT "{YEL}OF GOLD"
```

On the screen

```
10 REM RAINBOW  
20 PRINT "↑ POT"  
30 PRINT "π OF GOLD"
```

(Remember: "{GRN}" means hold down CTRL key and press 6 key.)

More Rainbows

Run:

```
10 REM LUCKY
20 PRINT "{RED}HI"
30 PRINT "{GRN}SALLY"
```

Run:

```
10 REM LEPRECHAUN
30 PRINT "{RED} P {CYN} I {PUR} X {GRN} I {BLU} E {YEL} S"
```

I put spaces in line 30 so you can read it easily. You should not put in the spaces. Without spaces it looks like this:

```
30 PRINT "{RED}P{CYN}I{PUR}X{GRN}I{BLU}E{YEL}S"
```

Aren't you glad I put in the spaces?

How does line 30 look on the screen? _____



Other Instructions in PRINT Statements

Just as "{RED}" in a PRINT statement means CTRL-RED,

"{CLR}" means SHIFTEd CLR/HOME

and "{HOME}" means CLR/HOME.

In each case, you press one or two keys and you see something funny on the screen, not "{CLR}", "{RED}", or "{HOME}".

Run this:

```
10 PRINT "{CLR}"
15 PRINT
20 PRINT "{WHT}HI"
25 PRINT
30 PRINT "{CYN}HI AGAIN"
```

Lines 15 and 25 just print a blank line.

Change line 30 to:

```
30 PRINT "{HOME}{GRN}HI AGAIN"
```

and run it again.

Assignment 2:

1. Write a program that prints your first, middle, and last names, with the first name green, the middle name yellow, and the last name red.
2. Now change the program so that each letter of your first name will have a different color.

Instructor Notes 3. LIST, Boxes in Memory

In this lesson:

- LIST, LIST 30
- Memory boxes holding lines
- Erase one line from memory
- Add a line between old lines
- Replace a line
- REM statement
- CLR/HOME key in a program
- INST/DEL key
- Drawings using a PRINT statement

By definition there is a difference between a command and a statement, but the BASIC interpreter does not distinguish between them. Later in the book I will explain the differences. For now your student needs to understand that the program is stored in memory even when it is not visible on the screen, and that LIST just lists the program to the screen. The special uses like LIST 100-300 and LIST -300 will be taken up later.

Computer memory as a shelf of boxes is a key model that we will develop in this book. It is an important tool in helping the student understand variables and the detailed workings of complicated expressions in a statement.

Using PRINT to draw pictures is demonstrated. It is better to draw some at the end of each lesson than to do a lot now. After lesson 4, drawing helps develop line-editing skills.

Questions:

1. How do you erase a line you no longer want?
2. Press CLR/HOME. Now, how do you show all of the program in memory on the screen?

-
-
3. How can you replace a wrong line with a corrected one?
 4. Suppose you want to put a line in between two lines you already have in memory. How do you do this?
 5. Explain how the computer puts program lines in "boxes" in memory. What does it write on the front of the box?
 6. How do you make a program clear the screen when it starts?

Lesson 3. LIST, Boxes in Memory

Clear the screen and erase the memory.

(Start each lesson by clearing screen and memory.)

Now enter:

```
10 REM HOUSE
20 PRINT "COME ON OVER"
```

Run this two-line program. Then clear the screen.

The program is gone from the screen.

But the program is not lost. It was stored in the computer's memory. We can ask the computer to show us the program again.

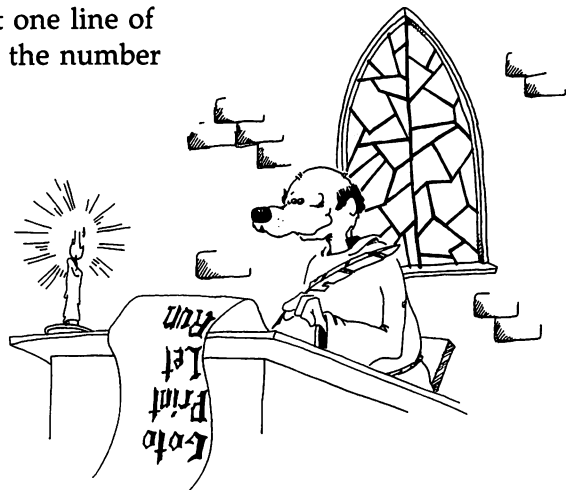
Listing the Program

Make the computer show you the whole program.

Enter: LIST

To make the computer show you just one line of the program, enter LIST followed by the number of the line, like this:

```
LIST 20
```



The Memory

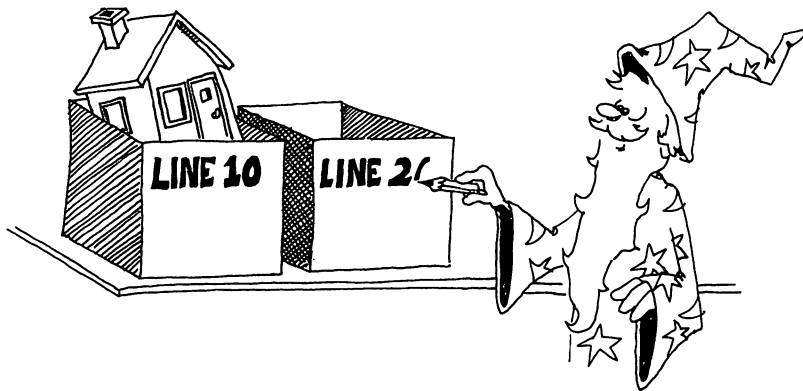
The computer's memory is like a shelf of boxes. There is a spot on the front of each box to write its name. At the start, all the boxes are empty and no box has a name.

When you entered: 10 REM HOUSE

the computer took the first empty box and wrote the name Line 10 on the front. Then it put the statement REM HOUSE in the box and put the box back on the shelf.

When you entered: 20 PRINT "COME ON OVER"

the computer took the second box and wrote Line 20 on its label. Then it put the statement PRINT "COME ON OVER" in the box and put that box in its place on the shelf.



Erasing a Line from Memory

To erase one line of the program, enter the line number with nothing after it.

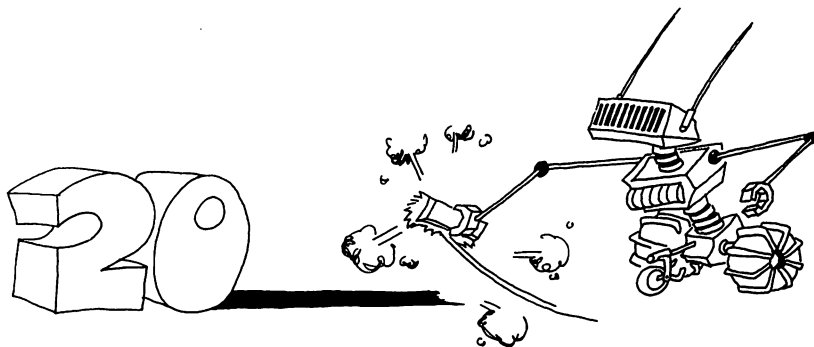
To erase line 20, enter: 20

You still see the line on the screen, but do a LIST and you'll see that line 20 is gone from memory.

When you enter just a line number with nothing after it, the computer finds the box with that line number on it, empties the box, and erases the name from the front of the box.

How do you erase the whole program? (See lesson 1 for the answer if you forgot.)

What does the computer do to the boxes when you give it the command NEW?



Adding a Line

You can add a new line anywhere in the program, even between two old lines. Just pick a line number between two old lines, and type your line in. The computer puts it in the correct place.

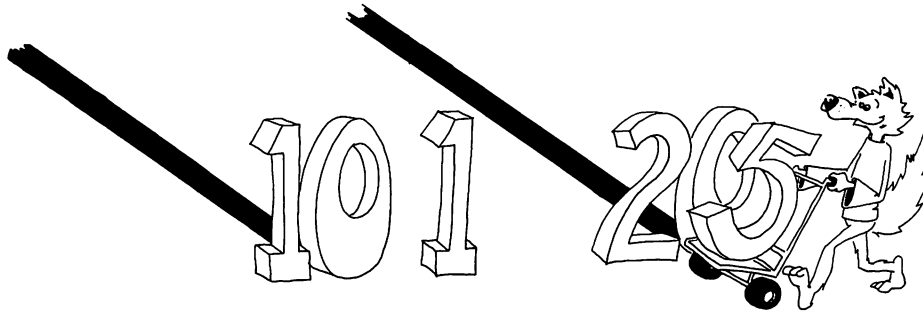
Enter NEW and this:

```
10 REM MORE AND MORE
20 PRINT "MORE LINES WANTED"
40 PRINT "NOW"
```

List it and run it. Now add this line:

```
15 PRINT "STILL"
```

List and run it again.

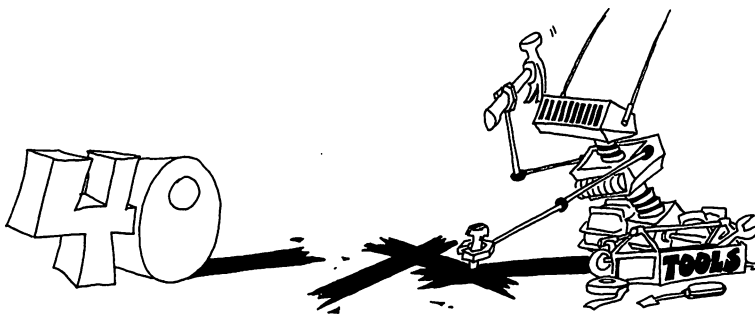


Fixing a Line

If a line is wrong, just type it over again. For example, in the above program line number 40 can be changed by entering:

```
40 PRINT "DOGIE"
```

What did the computer do to the box named Line 40 when you entered the line?

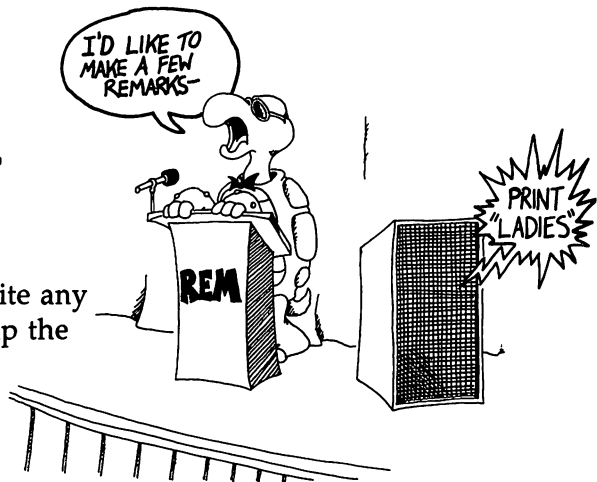


The REM Statement

Enter NEW and this:

```
10 REM LAZY
20 PRINT
30 PRINT "LINE 10 DOES NOTHING"
35 REM THIS LINE DOES NOTHING
40 LIST
```

REM means REMark. Use REM to write any little note in the program that can help the reader understand the program.



Using the CLR/HOME Key in a PRINT Statement

Suppose you want your program to start with a clean screen.

Run:

```
10 REM CLASSY CAR
20 PRINT "{CLR}{CYN}MERCEDES-BENZ"
```

Where it says "{CLR}" in the PRINT statement, you should hold down the SHIFT key and then press the CLR/HOME key. Where it says "{CYN}", hold down the CTRL key and press the CYN key.

Picture Drawing

You can use the PRINT statement to draw pictures. Here is a picture of a car. Add these lines to your program.

```
10 REM CLASSY CAR
20 PRINT "{CLR}{CYN}MERCEDES-BENZ"
25 PRINT
30 PRINT "XXXXX"
40 PRINT "XXXXXXXXXXXXX"
50 PRINT " o{7 SPACES}o"
```

Spaces are part of the drawing. Whenever you see braces, { }, it should tell you something special needs to be done. On line 50 where it says {7 SPACES}, press the space bar seven times. Here are the keystrokes for line 50:

```
5 0 SPACE P R I N T SPACE " SPACE O SPACE SPACE SPACE SPACE SPACE  
SPACE SPACE O "
```

Don't forget the space on line 30.



Assignment 3:

1. What command will list line 10 of a program?
2. How do you tell the computer to list the whole program on the screen?
3. What does the computer do (if anything) when it sees the REM statement?
4. What does the computer put in the boxes on its shelf?
5. Use REM and PRINT to draw three flying birds on the screen.

Instructor Notes 4. The Cursor Keys and Drawing Pictures

This lesson concerns the CRSR arrow keys, the graphics symbols on many keys, and the Commodore key.

The arrow keys are used in moving the cursor to any point on the screen. In particular, moving the cursor onto any part of a line allows editing of the line. Characters in a line are not affected by the cursor moving over them. Wherever the cursor stops, you can type in new characters.

For now we will not consider the insertion of characters, only their replacement by others or their deletion. When all is satisfactory, you can enter the line in the computer by pressing RETURN.

You can even change the line number. This allows you to move a line to another number. It also allows you to make copies of a line, either exact copies or slightly modified copies.

Most keys have two graphics symbols on them. These can be printed on the screen in the edit mode, or printed by a program. On a given key, the right graphics symbol is selected by using the SHIFT key, and the left graphics symbol by using a special key called the Commodore key.

The screen can be given any of 16 different colors by the statement POKE 53281,C. (C is a number from 0 to 15.) The change screen color statement can be given in a program line.

Be careful that you do not choose a screen color that is the same as the letter color! The writing would disappear until you changed one of the colors. In fact, this can be exploited in making "invisible writing." We will do this after the delay loop is introduced.

Questions:

1. What is a cursor? What is it good for?
-
-

-
-
2. Type the solid ball character on the screen. What keys do you need to use?
 3. Type the triangle graphics character that appears on the star key. What keys do you need to use?
 4. Have your student demonstrate how to edit a line. This includes using the arrow keys to move the cursor to the interior of the line, modifying characters there, and pressing RETURN to enter the line.
 5. What happens if you hold the INST/DEL key down?

Lesson 4. The Cursor Keys and Drawing Pictures

The Cursor Is a Flashing Square

The little flashing square is called the *INPUT cursor*. It shows you where the next letter you type will appear on the screen.

The Arrow Keys

Find the two CRSR keys. (CRSR stands for cursor.)

One CRSR key has left and right arrows.

The other CRSR key has up and down arrows.



These keys move the cursor.

Careful! We do not mean any of these arrow keys:



If you press a CRSR key, the cursor moves in the direction of the lower arrow on the key.

If you press SHIFT and a CRSR key, the cursor moves in the direction of the top arrow on the key.

If you hold down the key, the cursor starts moving very fast!

Do this: Use the cursor arrow keys to move the cursor to the middle of the screen, then type your name there.

Now put a border of colored stars (*) around your name.

The Graphics Characters

The Commodore computer has 62 graphics characters. You see them in little squares on the front of many keys.

They are easy to use.

First find the SHIFT key. Now find the Commodore key. It is under the RUN/STOP key and looks like a large C with a flag flying from its side.

To print graphics characters:

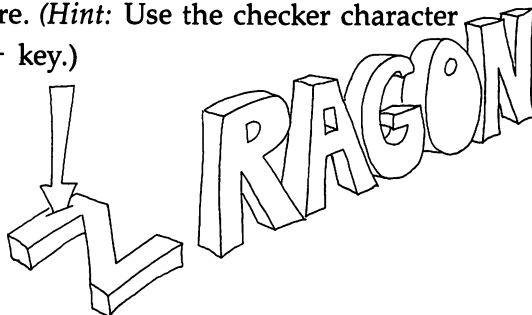
Hold down the SHIFT key and press a graphics key.
The character on the right will be put on the screen.

Or hold down the Commodore key and press a graphics key.
The character on the left will be put on the screen.

(You do not see the square that is on the key. You just see the character that is inside the square.)

Do this: Use the CRSR keys to move the cursor to the screen center and draw a tiny red square. (*Hint: Use the corner characters that are on the O, P, @, and L keys.*)

Now draw a large green square around the red square. (*Hint: Use the checker character on the + key.*)



Fixing Messed-Up Lines

The cursor arrow keys help you fix errors in your typing.

Enter: 1Ø REM ZRAGON

Use the CRSR keys to move the cursor onto the Z.

Type a D instead.

Now the line is correct, reading:

```
10 REM DRAGON
```

Press RETURN to store the correct line in the memory.

The INSErT/DELEte Key

The INST/DEL key is your eraser. (DEL is short for delete.) Try this:

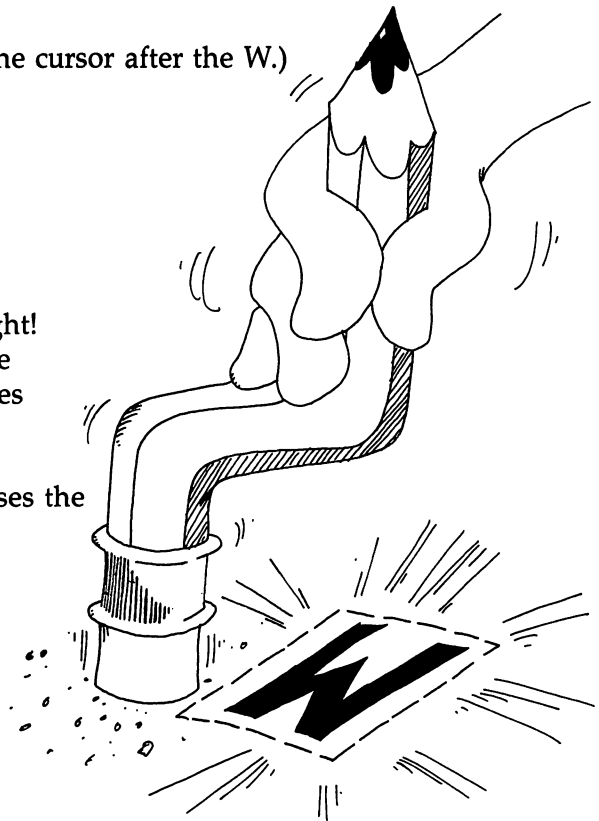
```
20 PRINT " HI THERW      (Leave the cursor after the W.)
```

Oops! The W should be an E.

You can erase the W by pressing the INST/DEL key. Then type an E.

Do you see what is funny? That is right! The INST/DEL key does *not* erase the character that the cursor is on, it erases the one next to it on the left!

Rule: The INST/DEL key always erases the character next to the cursor on the left.



Speedy Erasing

Hold down the INST/DEL key. The cursor whizzes along, erasing as it goes. Careful! You may erase more than you want!

Copycat Lines

Enter:

```
10 REM TWINS
20 PRINT "MEET MY TWIN"
```

Run the program.

List the program. Then use the cursor arrow keys to move the cursor on the 2 of line 20. Type 3 then press the RETURN key.

Now run and list the program again. Line 20 has a twin line named 30.

Colored Screens

You know how to color the screen background black or white:

Black: 10 POKE 53281,0

White: 10 POKE 53281,1

Choose another color by picking another number instead of 0 or 1. Any number from 0 to 15 can be used.

Enter:

```
10 REM RED FLASH
20 PRINT "{WHT}HITHERE"
30 POKE 53281,2
```

(Do you remember what to do when you see "{WHT}"?)

Assignment 4:

1. Try making different colored screens and different colors of letters on each screen. Which combinations look best?

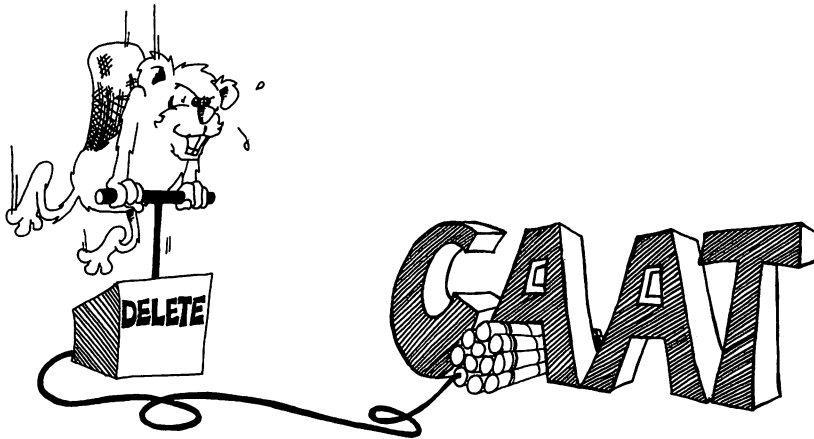
2. Practice using the INST/DEL and the CRSR keys. Type and fix these lines:

```
10 REM CAAAAAAT
```

```
10 REM TTIIGGEEERRR
```

3. Draw a smiley face on a colored screen.

4. Draw a valentine. Use lots of different graphics symbols.



Instructor Notes 5. Tricks with PRINT

In this lesson:

- PRINT with a semicolon at the end
- PRINT with semicolons between items
- PRINT with commas between items
- The "invisible" PRINT cursor
- Characters and string constants
- Review of keys

The lesson introduces the PRINT cursor which is invisible on the screen. It marks the place where the next character will be placed on the screen by a PRINT statement. (The INPUT cursor is the flashing square. It is familiar from the edit mode and also appears when executing the INPUT command.)

When a PRINT statement ends with a semicolon, the PRINT cursor remains in place at the end of the last printed character. The next PRINT will start writing characters onto the end of the message printed by the current PRINT statement.

Without a semicolon at the end, the PRINT statement will advance the PRINT cursor to the beginning of the next line as its last official act.

A PRINT statement can print several items, a mixture of string and numerical constants, variables, and expressions. Numerical constants and variables have not yet been introduced. The items are separated by semicolons or commas.

If commas are used, the items will be printed in columns.

A series of printed strings will have their characters in contact. If spaces are desired, as in the "Toast and Jam" example, the spaces have to be put in the strings explicitly.

Questions:

1. Which cursor is a little flashing square? What statement puts it on the screen?
2. Which cursor is invisible? What statement uses it?
3. How do you make two PRINT statements print on the same line?
4. Will these two words have a space between them when run?

```
10 PRINT "HI";"THERE!"
```

If not, how do you put a space between them (two ways)?

Lesson 5. Tricks with PRINT

One Line or Many?

Enter this program:

```
10 REM YUMMY
20 PRINT
30 PRINT "TOAST"
40 PRINT "AND"
50 PRINT "JAM"
```

and run it. Each PRINT statement prints a separate line.

Now enter:

```
30 PRINT "TOAST ";
40 PRINT "AND ";
```

(Don't change or erase the other lines.) Be careful to put the space at the end of "TOAST " and at the end of "AND " and the semicolon at the end of each line.

Run it. What was different from the first time?

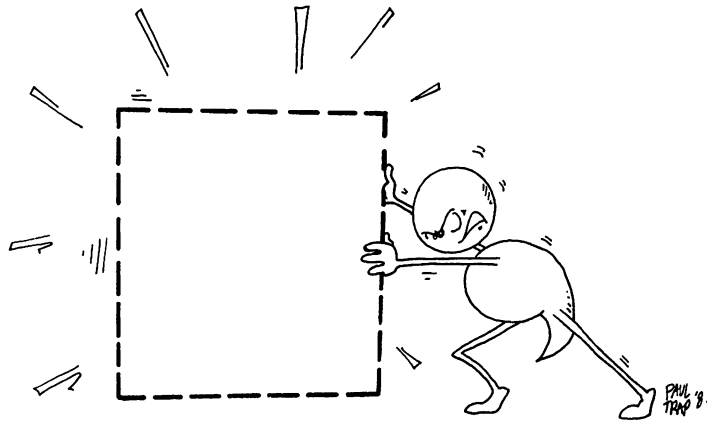
The Hidden Cursor

Remember the flashing square? It is the INPUT cursor and shows where the next letter will appear on the screen when you type.

The PRINT statement also has a cursor, but it is invisible. It marks where the next letter will appear when the computer is PRINTing.



Rule: The semicolon makes the invisible PRINT cursor wait in place on the screen. The next PRINT statement adds on to what has already been written on the same line.



Famous Pairs

The PRINT statement can print several strings, one after another. Put semicolons (;) between the strings. Look at line 80 below.

Enter:

```
10 REM NAME DROPPING
20 PRINT "{CLR} ENTER A NAME "
30 INPUT A$
40 PRINT " ENTER ANOTHER "
50 INPUT B$
70 PRINT "{CLR} PRESENTING THAT FAMOUS TWOSOME "
75 PRINT
80 PRINT A$;" AND ";"B$
```

(Remember, {CLR} means hold down the SHIFT key and press the CLR/HOME key.) The INPUT statement will be explained in the next lesson.

Don't forget to put a space before and after the " AND " in line 80.

Squashed Together or Spread Out?

Enter NEW, then try this:

```
10 PRINT "ROCK";"AND";"ROLL";"STAR"
```

After you have run it, try also:

```
10 PRINT "ROCK", "AND", "ROLL", "STAR"
```

Rule: A comma between items in a PRINT statement puts spaces between them on the screen.

Characters

Look at these PRINT statements:

```
10 PRINT "JOE"  
10 PRINT "#D4788*8"  
10 PRINT "19"  
10 PRINT "3.14159265"  
10 PRINT "I'M 14"
```

Letters, numbers, and punctuation marks are called *characters*.

Even a blank space is a character. Look at this:

```
10 PRINT " "
```

All the little drawings on the front of the keys are characters, too. They are called *graphics characters*.

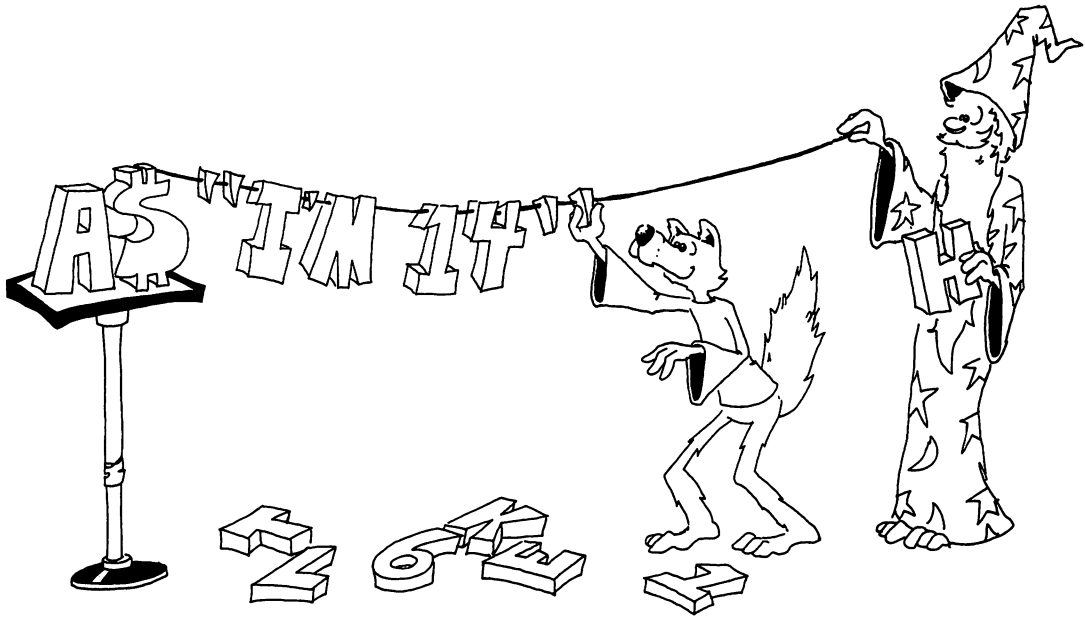
String Constants

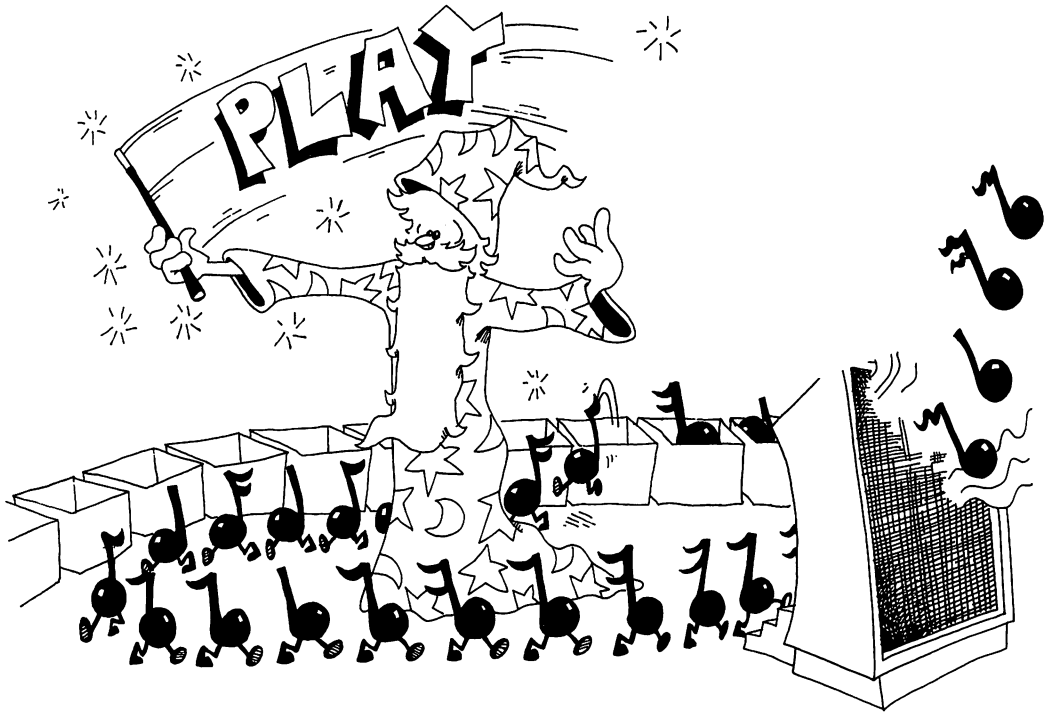
Characters in a row make a *string*.

The letters are stretched out like beads on a string.

A string between quotation marks is called a *string constant*.

It is a string because it is made of letters, numbers, punctuation marks, and graphics characters in a row. It is a constant because it stays the same. It doesn't change as the program runs.





Assignment 5:

1. Write a program that asks for the name of a musical group and one of their tunes. Then using just one PRINT statement, print the group name and the tune name, with the word *plays* in between.
2. Do the same, but use three PRINT statements to print on one line.
3. Write a program that asks the user for three words. (Use three INPUT statements.) Then print them on one line with spaces between them. (Use PRINT with commas.)

Instructor Notes 6. The INPUT Statement

This lesson concerns the INPUT statement, the concept of a string variable, and the difference between a *programmer* and a *user*.

In its simplest form, INPUT A\$, there is no message in quotes in front. We want the student to concentrate on the central feature of an INPUT.

Similarly, we will give only the essential feature of each statement for the first section of the book (through lesson 14). We want the student to see the forest before going into details. The interesting programs require:

PRINT	Allows	output
INPUT		input
GOTO		infinite looping
IF		branching and decisions
RND		random numbers for games

The box concept is used again to introduce string variables. For the time being, variable names are restricted to one letter. This allows faster typing and puts off discussion of the complicated naming rules until after our sprint to the RND function.

We will work with strings before numbers, because strings make for more interesting programs and offer a less confusing entry into the logical concepts of programming.

The “two hats” of the student—programmer and user of the program—cause much confusion at assignment time. PRINT is the programmer speaking, while the user’s comments are made only in response to an INPUT statement and are stored in a string variable to be used or printed by the computer.

Questions:

1. What two different things does the computer put in boxes? (One at programming time and one from an INPUT.)

-
-
2. How does the program ask a user to type in something?
 3. How do you know the computer is waiting for an answer?
 4. What is a letter with a dollar sign after it called?
 5. Write a short program that uses REM, PRINT, and INPUT.
 6. Are you in trouble if the computer answers ?EXTRA IGNORED after an input?
What made it do that?

Lesson 6. The INPUT Statement

Use INPUT to make the computer ask for something.

Enter:

```
10 REM TALKY-TALK
15 PRINT "{CLR}"
20 PRINT "SAY SOMETHING"
25 INPUT A$
30 PRINT
35 PRINT "DID YOU SAY"
40 PRINT A$
```

Run it. When you see a question mark, type HI and press the RETURN key.

The question mark was written by INPUT in line 25. The flashing cursor means the computer expects you to type something in.

When you enter HI, the computer stores this word in a box named A\$.

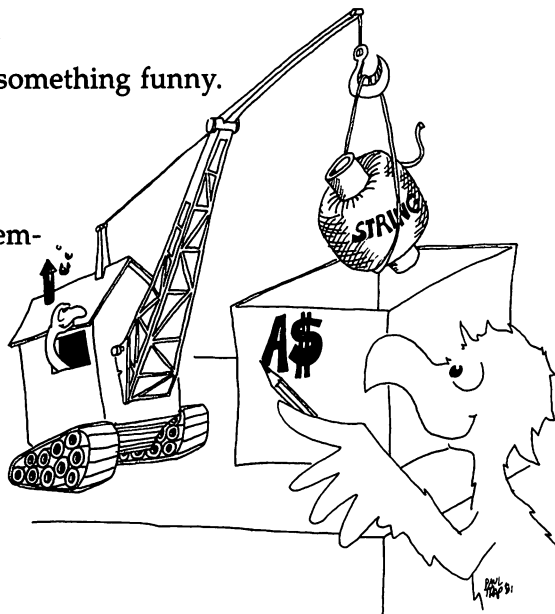
Later, in line 40, the program asks the computer to print whatever is in the box named A\$.

Run the program again and this time say something funny.

String Variables

A\$ is the name of a *string variable*.

The computer stores string variables in memory boxes just like the boxes it puts program lines in. The name is written on the front of the box and the string is put inside the box.



Rule: A string variable name ends in a dollar sign (\$). You can use any letter you like for the name and then put a dollar sign after it.

A\$ is called a variable because you can put different strings in the box at different times in the program. The box can hold only one string at a time. Putting a new string in a box erases the old string that was in the box.

Error Messages from INPUT

Run this three times:

```
10 INPUT A$
20 PRINT "      ";A$
```

Try these answers:

```
HI
HI, THERE
HI: 1 2 3
```



Rule: Do not put any commas or colons in the string you type in answer to the computer.

If you accidentally do put one in, the computer may answer:

```
?EXTRA IGNORED
```

and continue. This means that the computer chopped off everything after the comma or colon and then continued running the program.

You Wear Two Hats—User and Programmer

You are a *programmer* when you write a program. The person who runs the program is a *user*.

Of course, if you run your *own* program then *you* are the *user*.

When the programmer writes a PRINT statement, the *programmer* is speaking to the user by writing on the screen.

When the programmer writes an INPUT statement, the *programmer* is asking the *user* to say something to the computer.

It is like a game of “May I?” The only time the user gets to say something is when the programmer allows it by writing an INPUT statement in the program.



Assignment 6:

1. Write a program that asks for the user's name and then says something silly to the person by name.
2. Write a program that first asks for the the user's favorite color—put it in a box called C\$—then asks for a favorite animal—put this in box C\$ also. Have the program PRINT C\$. What will be printed? Run the program and see if you are right.

Instructor Notes 7. The LET Statement, Gluing Strings

The LET statement and concatenation are introduced.

Concatenation of strings glues strings together to make a new string.

The box model is used to emphasize that LET is a replacement statement, *not* an equal relationship in the sense used in arithmetic.

The box idea nicely separates the concepts *name* of the variable and *value* of the variable. The name is on the label of the box, the value is inside. The contents of the box may be removed for use. More exactly, a copy of the contents is made and used when a variable is used, while the original contents remain intact. This point is explained. When LET puts new contents in a box, the old contents are automatically erased first.

Statements used so far:

NEW, PRINT, REM, RUN, LIST, INPUT, LET

Special keys discussed so far:

RETURN, CRSR arrows, SHIFT, CLR/HOME, CTRL, INST/DEL, and the Commodore key.

Questions:

1. LET puts things in boxes. So does INPUT. How are they different?
2. If you run this little program

```
10 LET A$="HI"  
20 LET B$=A$
```

what will be in box A\$ at the end? What will be in box B\$?

3. In this program

```
10 LET Q$="MOM"
```

what is "MOM" called? What is the name of the string variable in this program? What is the value of the string variable after the program runs?

4. What is in each box after this program runs?

```
10 LET H$="FAT"  
20 LET K$="SAUSAGE"  
30 LET P$=H$+K$
```

Lesson 7. The LET Statement, Gluing Strings

The LET statement puts things in boxes. Enter and run:

```
10 PRINT "{CLR}"  
20 LET W$="TRUCK"  
40 PRINT W$
```

Here is what the computer does:

Line 10 The computer clears the screen.

Line 20 It sees that a box named W\$ is needed. It looks in its memory for it. It doesn't find one, because W\$ has not been used in this program before. So it takes an empty box and writes W\$ on the front, and then puts the string "TRUCK" in it.

Line 40 The computer sees that it must print whatever is in box W\$. It goes to the box and makes a copy of the string "TRUCK" that it finds there. It puts the copy on the TV screen. The string "TRUCK" is still in box W\$.



Names and Values

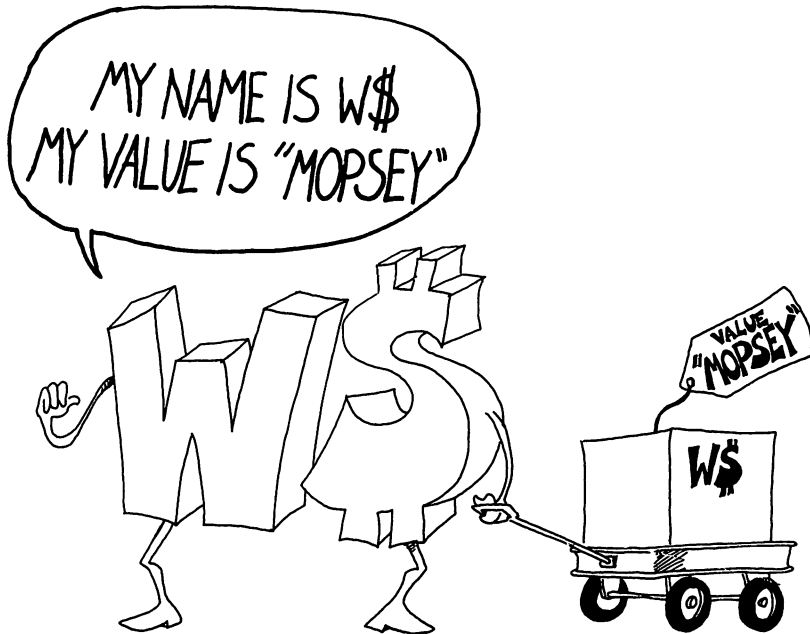
This line makes a string variable:

```
10 LET W$="MOPSEY"
```

The name of the variable is W\$.

The value of the variable is put in the box.

In this line, the value of W\$ is "MOPSEY".



Another Example

Enter and run:

```
10 LET D$="PICKLES"  
20 LET A$=" AND "  
30 PRINT "WHAT GOES WITH PICKLES?"  
35 INPUT Z$  
40 PRINT "{CLR}"  
50 PRINT D$;A$;Z$
```

Explain what the computer does in each line.

- 10 _____
- 20 _____
- 30 _____
- 35 _____
- 40 _____
- 50 _____



Gluing the Strings

Here is how to stick two strings together to make a longer string.
Enter:

```
10 PRINT "{CLR}"
20 LET W$="HAR DE "
25 LET X$="HAR "
30 LET A$=W$+X$
40 PRINT A$
50 PRINT
60 LET A$=A$+X$
70 PRINT A$
```

Before you run this program, try to guess what will be printed at line 40 and at line 70:

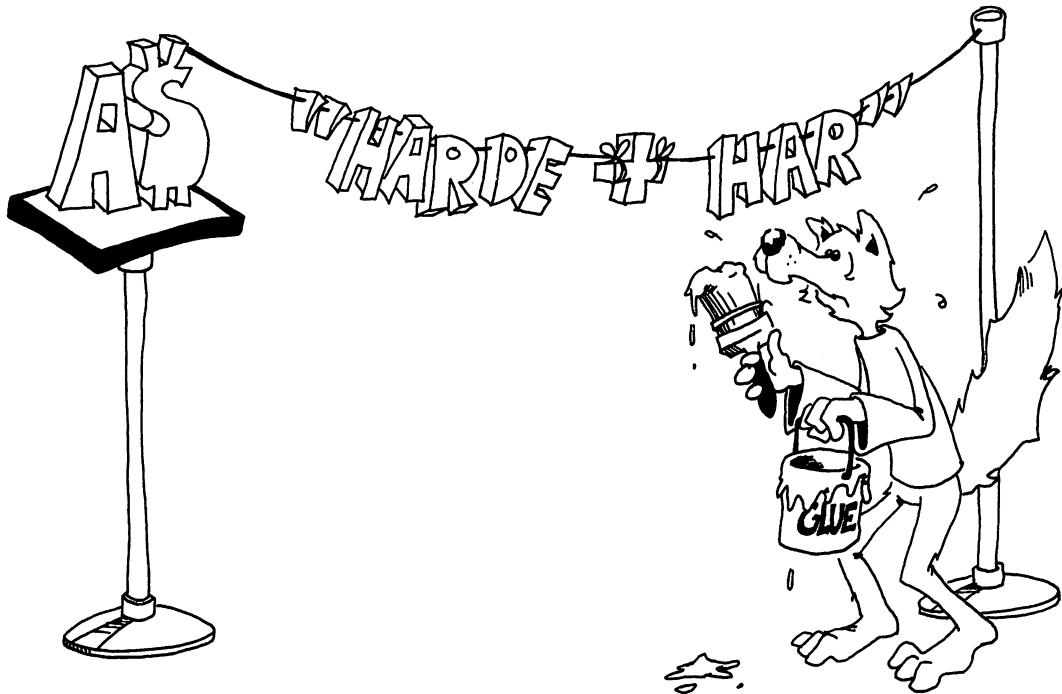
40 _____

70 _____

Now run the program to see if you were right.

Lines 30 and 60 glue strings together.

Rule: The + sign sticks two strings together.



Assignment 7:

1. Write your own program that uses the LET statement and explain how it stores things in boxes.
2. Write a program that inputs two strings, glues them together, and then prints them.

Instructor Notes 8. The GOTO Statement and the RUN/STOP Key

The GOTO statement allows loops that go on forever. It also helps in flow of statement execution once we introduce the IF statement. It provides a slow and easy entrance for the student into the idea that the flow of a program need not go down the list of numbered lines.

For now its main use is to let programs run on for a reasonable length of time. In each loop through, something can be modified.

The problem is how to stop it. The RUN/STOP key does this nicely. Sometimes pressing the RUN/STOP key may not in fact stop the program. For example, if the program reached an INPUT statement (and shows the question mark and flashing cursor), pressing RUN/STOP does not stop the program. Try this. Hold RUN/STOP down and then press the RESTORE key once or twice.

GOTO allows the bad habit of "spaghetti" programming to grow. Examples of spaghetti are shown to the student. Although some fun is had with them, make sure the student is conscious of the problems that undisciplined use of GOTO can cause.

We now have three of the four major elements that lead to meaningful programming. They are PRINT, INPUT, and GOTO. Lacking is the IF statement, which will change the computer from only a record player into a machine that can evaluate situations and make decisions accordingly.

Questions:

1. In this little program:

```
10 PRINT "HI"  
20 GOTO 40  
30 PRINT "BIG"  
40 PRINT "DADDY"
```

what will appear on the screen when it is run?

2. What about this one:

```
10 PRINT "APPLE"  
20 PRINT "PIE";  
30 GOTO 20
```

3. How do you stop the program in question 2?

4. Write a short program that asks your favorite movie star's name and then PRINTs it over and over again.

Lesson 8. The GOTO Statement and the RUN/STOP Key

Jumping Around in Your Program

Try this program:

```
10 PRINT "{CLR}"
20 PRINT " YOUR NAME?"
25 INPUT N$
30 PRINT N$
35 PRINT
40 GOTO 30
```

Run this program. It never stops by itself! To stop your name from whizzing past your eyes, press the RUN/STOP key. (It's on the left side of the keyboard.)

Line 40 uses the GOTO statement. It is like "Go to Jail" in a game of Monopoly. Every time the computer reaches line 40, it has to go back to line 30 and print your name again.


We will use GOTO often in programs.



More Jumping

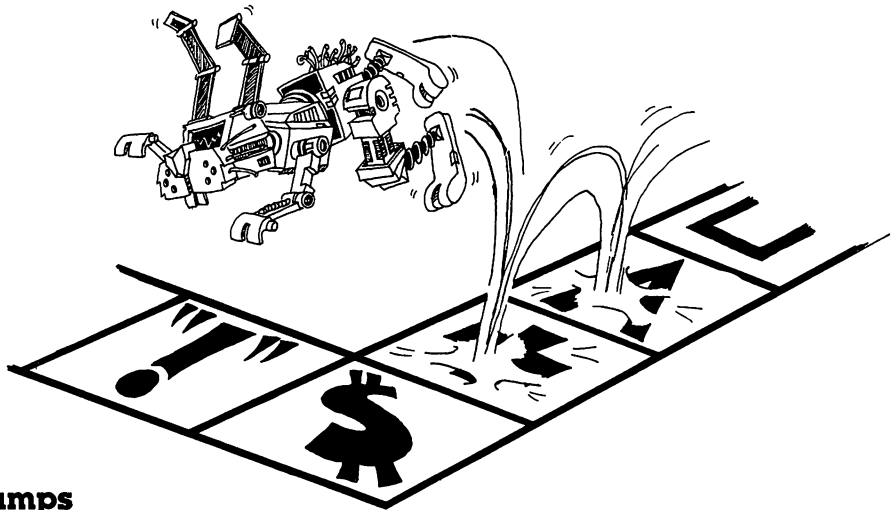
Enter:

```
20 PRINT "SAY SOMETHING"  
30 INPUT S$  
40 PRINT "DID YOU SAY '";S$;"'?"  
45 PRINT  
50 GOTO 30
```



Run the program. Type an answer every time you see the ? and the flashing cursor. Press the RUN/STOP key to end the program.

Notice the arrow from line 50 to line 30. It shows what the GOTO does. You may want to draw such arrows in your program listings.



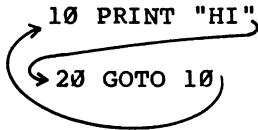
Kinds of Jumps

There are only two ways to jump: ahead or back.

Jumping back gives a *loop*.

```
10 PRINT "HI"  
20 GOTO 10
```

The path through the program is like this:



The computer goes around and around in this loop. Press the RUN/STOP key to stop.

Jumping ahead skips part of the program. Whatever for? We will see later in the IF statement.

The RUN/STOP Key

The RUN/STOP key is a lifesaver. When you are in trouble, press RUN/STOP and the computer will stop running the program and wait for your next command. Your program is still safe in memory.

If you are in really big trouble, press the RUN/STOP key and at the same time press RESTORE. The computer does a *warm start*. Your program is still safe in memory.

A Can of Spaghetti

Look at this:

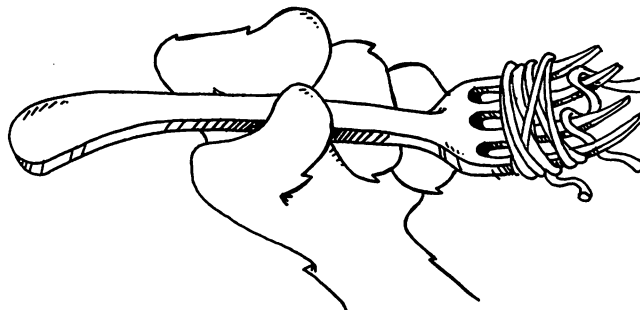
```
10 REM -----SPAGHETTI-----  
  
20 GOTO 70  
25 PRINT "A"  
26 GOTO 50  
30 PRINT "S"  
31 GOTO 25  
40 PRINT "C"  
41 GOTO 90  
50 PRINT "U"  
51 GOTO 40  
70 PRINT "S P A G H E T T I "  
71 GOTO 30  
90 PRINT "E"  
  
95 REM---END---  
  
100 PRINT "WHEW!"
```

```
graph TD; 20 --> 70; 25 --> 26; 26 --> 30; 30 --> 31; 31 --> 25; 40 --> 41; 41 --> 90; 50 --> 51; 51 --> 40; 70 --> 71; 71 --> 30; 90 --> 100;
```

This is not a good, clear program.

It is a "spaghetti" program.

Don't write spaghetti programs! Don't jump around too much in your programs.



The INSerT/DELeTe Key

INST stands for INSerT which means "put in."

When you hold down SHIFT and press the INST/DEL key, the computer sticks in a space to the left of the cursor, and then moves the cursor onto it. Try this:

```
20 PRINT "WHICH UP?"
```

Now use the CRSR arrows to move the cursor onto the U. Hold down the SHIFT key and press the INST/DEL key four times. Then type WAY.

INSerT is the opposite of DELeTe. After you have inserted a space, you may type a letter into it.

If you hold down the SHIFT and INST/DEL keys, the cursor whizzes along and inserts a lot of spaces.

The INSerT/DELeTe Key Goes Crazy

After inserting spaces in a line, you may type letters, numbers, punctuation, and graphics into the spaces and they will appear on the screen. But if you press the CRSR, CLR/HOME, INST/DEL, or color keys, you will see funny characters on the screen.

Assignment 8:

1. Just for practice in understanding the GOTO statement, draw the road map for this spaghetti program:

```
10 REM >>>FORKED TONGUE>>>
```

```
20 GOTO 40
```

```
30 PRINT "N"
```

```
31 GOTO 60
```

```
40 PRINT "S"
```

```
41 GOTO 30
```

```
50 PRINT "E"
```

```
51 GOTO 99
```

```
60 PRINT "A"
```

```
90 PRINT "K"
```

```
91 GOTO 50
```

```
99 PRINT "B I T E"
```



2. Write a program that prints TEEN POWER over and over.

-
-
3. How do you stop your program?
 4. Write another that prints your name on one line, then a friend's name on the next, over and over. Print each name in a different color. Stop the program with the RUN/STOP key.
 5. Write a program that uses each of these statements: PRINT, INPUT, LET, GOTO. It should also glue two strings together and use two colors of letters.

Instructor Notes 9. The IF Statement

This lesson introduces the IF statement and treats the question of whether two strings are the same or not.

IF is a powerful statement that is at the very heart of the computer as a logic machine. It is an intricate statement and the student may need extra help at this point.

The IF statement appeals both to our verbal and visual imagination. The "cake" cartoon and the "fork in the road" cartoon illustrate these ideas. The GOTO statement has already introduced the idea that the flow of a program can be altered. To that idea we can now add the conditional test: If an expression is true, one thing happens; if it is false, another.

Phrase A is used for the assertion being tested for truth. *Statement C* is used for the statement to be done if the assertion is true.

Two levels of abstract ideas occur in the assertions. On the literal level we have *equal* and *not equal*:

$A\$ = B\$$

$C\$ <> D\$$

The next level up, we have the *truth* or *falsity* of the assertion.

Some care will be needed to separate and clarify these notions.

When you see $A = B$ it may not *really* be true that A is equal to B, because the assertion may in fact be false.

The larger set of relations:

< > = =< =>

will be treated in later lessons.

Questions:

1. How do you make this program print THAT'S FINE?

```
15 PRINT "DOES YOUR TOE HURT?"
17 INPUT T$
20 IF T$="NAH" THEN GOTO 90
40 IF T$="SOME" THEN GOTO 15
90 PRINT "THAT'S FINE"
```

2. Write a short program which asks if you like chocolate or vanilla ice cream.

Answers to be C or V. For the C answer, print Yummy! For the V, print Mmmmmm!

3. What do we mean by phrase A?

4. What do we mean by statement C?

5. Where is the "fork in the road" in an IF statement?

Lesson 9. The IF Statement

Clear the memory and enter:

```
10 PRINT "{CLR}"
20 PRINT "ARE YOU HAPPY?(YES OR NO)"
30 INPUT A$
40 IF A$="YES" THEN PRINT "I'M GLAD"
50 IF A$="NO" THEN PRINT "TOO BAD"
```

Run the program several times. Try answering YES, NO, or MAYBE. What happens?

YES _____

NO _____

MAYBE _____

The IF Statement

The IF statement has two parts:

10 IF *phrase A* THEN *statement C*

First, the computer looks at *phrase A*.

If it is true, the computer does *statement C*.

If *phrase A* is not true, then the computer goes on to the next line without doing *statement C*.

It looks like this:

```
10 IF phrase A is true THEN do statement C
   and then go on to the next line
```

or

```
10 IF phrase A is false THEN go on to the next line
```

Assignment 9A:

1. Clear memory and write a program that asks which you like better, football or baseball. If the answer is baseball, the program should respond with PLAY BALL. If the answer is football, PRINT some other remark on the screen.

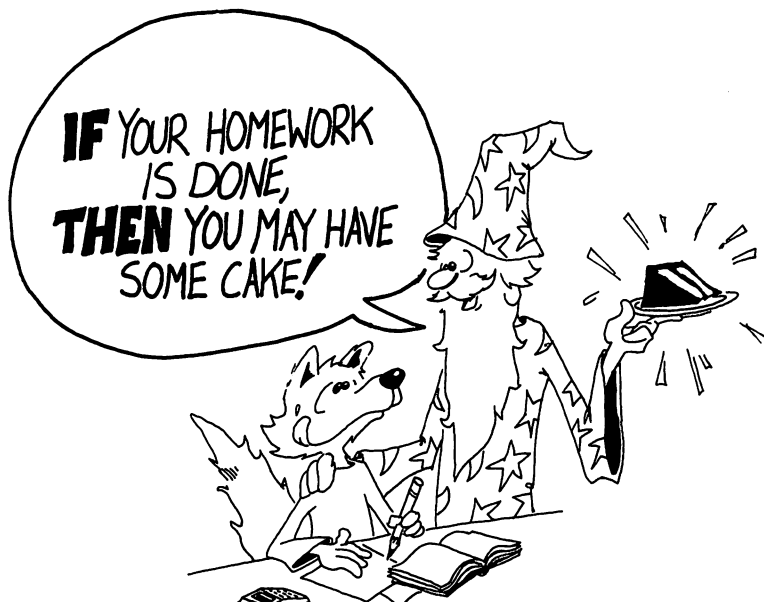
The IF in English and in BASIC

In English:

IF your homework is done, THEN you may have some cake.

In BASIC:

```
40 IF      A$="DONE"      THEN PRINT"EAT SOME CAKE"
```



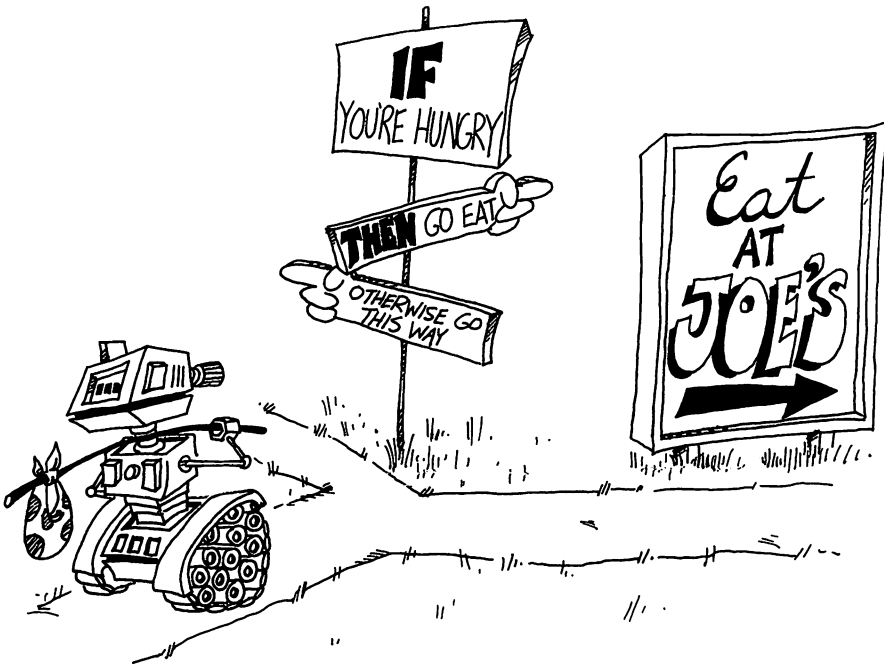
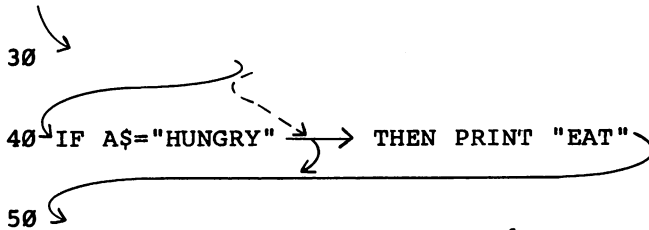
A Fork in the Road

When the computer sees IF, it must choose which road to take.

IF phrase A is true, it must go past the THEN and obey the statement it finds there.

IF phrase A is false, it goes down to the next line right away.

Here is the road map with the fork in the road marked:



The Not Equal Sign

= means equal

<> means not equal

To make the <> sign:

Hold down the SHIFT key and press the < key, then the > key.

Using the <> Sign

40 IF *phrase A* THEN *statement C*

Phrase A is a phrase that is true or false.

Pick: B\$<>"FIRE" for phrase A

Put it in an IF statement:

```
40 IF B$<>"FIRE" THEN PRINT "FEED HIM SOME HOT CHI
LI"
```

IF the B\$ box contains "COLD"
THEN B\$ is not equal to "FIRE"
and the expression B\$<>"FIRE" is TRUE.

The computer will print FEED HIM SOME HOT CHILI.

Or

IF the B\$ box contains "FIRE"
THEN the phrase B\$<>"FIRE" is FALSE
and the computer will not print anything.

Here is how it looks in a program:

```
10 PRINT "WITH DOGS IT'S A COLD NOSE"
11 PRINT
20 PRINT "WITH DRAGONS, IT'S..."
21 PRINT
25 PRINT "HOW IS YOUR DRAGON'S BREATH?"
26 PRINT
28 PRINT "(ENTER \"FIRE\" OR \"COLD\")"
29 PRINT
30 INPUT B$
40 IF B$<>"FIRE" THEN PRINT "FEED HIM SOME HOT CHI
LI"
50 B$="FIRE" THEN PRINT "WATCH OUT!"
60 PRINT "NICE DRAGON!"
```



Assignment 9B:

1. Write a pizza program. Ask what topping is wanted. Make the computer answer something silly for each different choice. You can choose mushrooms, pepperoni, anchovies, green peppers, or anything you want. You can also ask what size.
2. Write a color guessing game. One player INPUTs a color in string C\$ and the other keeps INPUTing guesses in string G\$. Use two IF lines, one with a phrase A

G\$<>C\$

for when the guess is wrong, and the other with an equal sign for when the guess is right. The statement C prints wrong or right.

Instructor Notes 10. Introducing Numbers

Numerical variables and operations are introduced and the LET, INPUT, and PRINT statements are revisited. The idea of memory as a shelf of boxes is extended to numbers. Again, for the time being, variable names are limited to one letter.

The arithmetic operations are illustrated. The * symbol for multiplication will probably be unfamiliar to the student. Division will produce decimal numbers, but since most arithmetic will be addition and subtraction, with a little multiplication, familiarity with decimal numbers will be helpful but not essential.

It may seem strange to the student that the numbers in string constants cannot be used directly in arithmetic. The VAL and STR\$ functions, which will be introduced later in the book, allow interconversion of numbers and strings.

A mixture of string and numerical values can be printed by PRINT.

The nonstandard use of = in BASIC, that it means *replace*, not *equal*, shows up in statements such as:

LET N=N+1

A cartoon uses the box idea to illustrate this meaning of =.

Another idea from arithmetic that doesn't work in a LET is shown below.

Arithmetic: $N = 3$ means the same as $3 = N$.

BASIC: LET N = 3 is okay. LET 3 = N is not.

Questions:

1. Name the three kinds of boxes in memory. (That is, named by the kinds of things stored in the boxes.)
2. Explain why $N = N + 1$ for a computer is not like $5 = 5 + 1$ in arithmetic.
3. Give another example of bad arithmetic in a LET statement. Use the * or / symbols.
4. What does the computer mean by TYPE MISMATCH ERROR?
5. Give an example of a program line that would have a TYPE MISMATCH ERROR.
6. Explain what is meant by the *name of a variable* and the *value of a variable* for numerical variables. For string variables.

Lesson 10. Introducing Numbers

INPUT, LET, and PRINT

So far we have only used strings. Numbers can be used, too. Enter and run this program:

```
10 PRINT "{CLR}"
20 PRINT "GIVE ME A NUMBER"
30 INPUT N
40 LET A=N+1
45 PRINT
50 PRINT "HERE IS A BIGGER ONE"
60 PRINT A
```

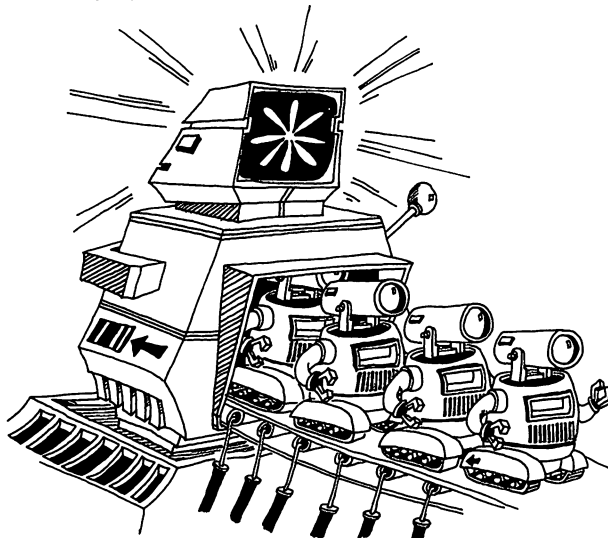
Arithmetic

The plus and minus signs are side by side in the top row of the keyboard.

Computers use * instead of \times for a multiplication sign.

Try this. Change line 40 so that N is multiplied by 5.

Computers use / for a division sign. It is on the same key with the ?. Answers will be given as decimals.

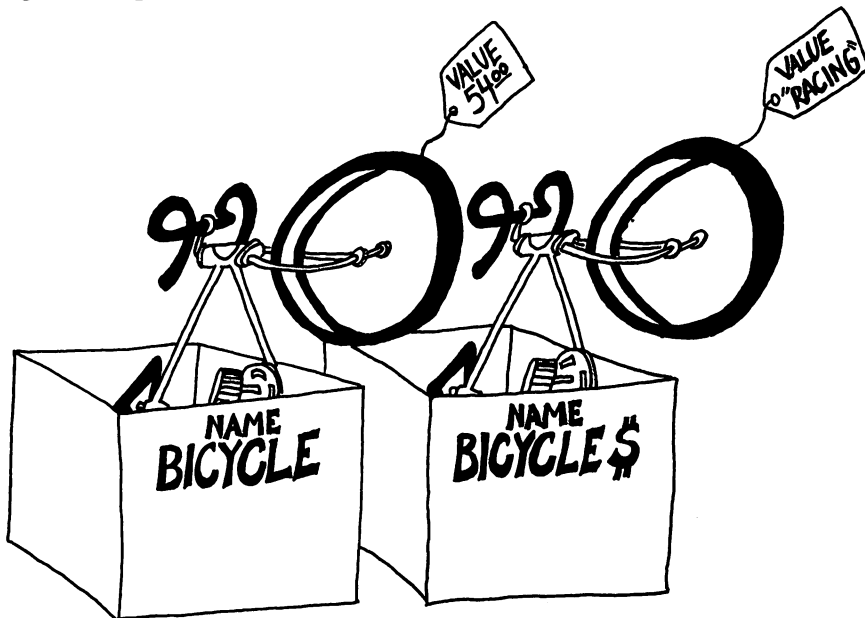


Variables

The name of a box that contains a string must end with a dollar sign. Examples: N\$, A\$, Z\$.

The name of a box that contains a number doesn't have a dollar sign. Examples: N, A, Z.

The thing that is put in the box is called the *value* of the variable.



Arithmetic in the LET Statement

```
10 LET A=2001
20 LET B=1983
30 LET C=A-B
40 PRINT " HOW MUCH LONGER, HAL?"
50 PRINT C;" YEARS"
```

Careful!

Numbers and strings are different. Example: "1985" is not a number. It is a string constant because it is in quotes.

Rule: Even if a string is made up of number characters, it is still not a number.

Some numerical constants are 5, 22, 3.14, -50.

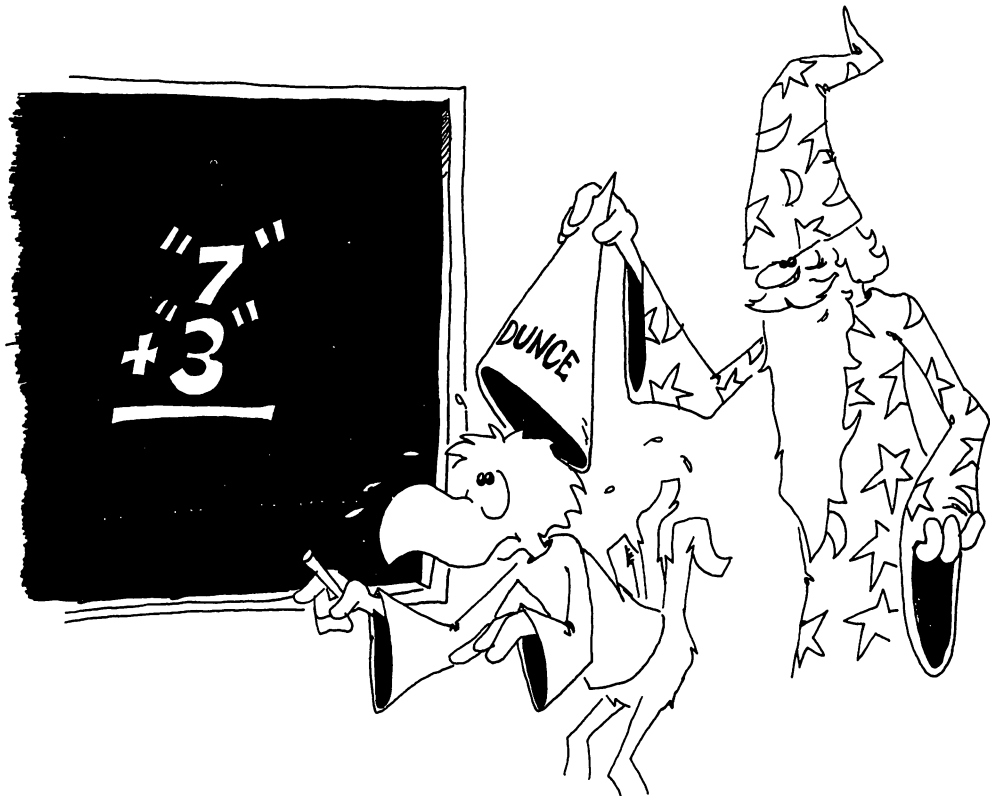
Some string constants are "HI", "7", "TWO", "3.14".

Rule: You cannot do arithmetic with the numbers in strings.

Correct: 10 LET A = 3 + 7
Wrong: 10 LET A\$ = 3 + 7
Wrong: 10 LET A = "3" + "7"

If you run either of these wrong lines, the computer will print:

TYPE MISMATCH ERROR IN 10



There are two types of variables: number and string.

You cannot put a number in a string box or a string in a number box.

Enter:

```
10 LET A=5
20 LET B$="10"
30 LET C=A+B$
```

Lines 10 and 20 are okay, line 30 is wrong. What will the computer do when you run this little program? _____

Try to guess what each of these statements will print, then enter the line to see what happens:

```
PRINT 5 _____
PRINT "5" _____
PRINT "5 + 3" _____
PRINT "5"+"3" _____
PRINT 5 + 3 _____
```

Mixtures in PRINT

You can print numbers and strings in the same PRINT statement. (Just remember that you cannot do arithmetic with the mixture.)

Correct: PRINT A;"SEVEN "; "7"
 PRINT A;B\$

Run this line: 10 PRINT 5/2;"IS EQUAL TO 5/2"

A Funny Thing About the Equal Sign

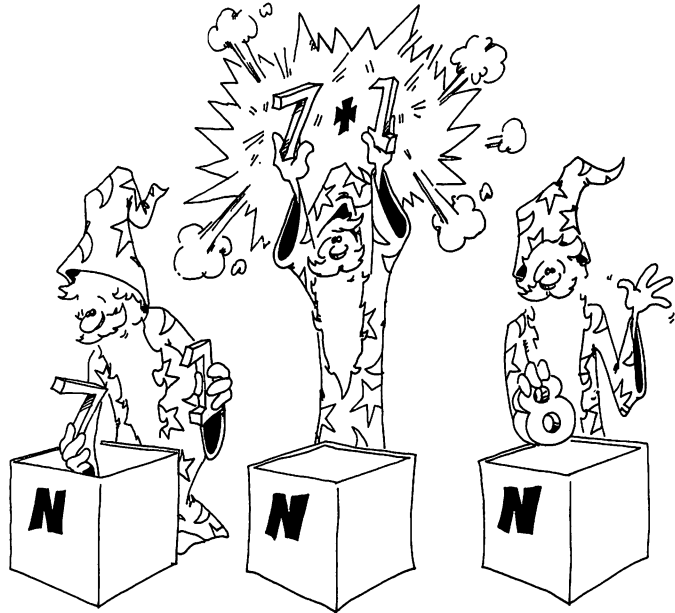
The = sign in computing does not mean *equals* exactly. Look at this program:

```
10 LET N=N+1
```

This does not make sense in arithmetic. Suppose N is 7. This would say that:

$$7 = 7 + 1$$

which is not correct.



But it is okay in computing to say $N = N + 1$ because the $=$ sign really means *replace*. Here is what happens:

Look at this: `10 LET N=N+1`

The computer goes to the box with N written on the front.

It takes the number 7 from the box.

It adds 1 to the 7 to get 8.

Then it puts the 8 in the box.

Another way to say the same thing is:

`10 LET N = N + 1`

means `LET (new N) equal (old N) plus one`

Instructor Notes 11. TAB and Delay Loops

TAB follows the familiar tab function of a typewriter. Delay loops slow the program down so that its operation can be more easily observed. They also are used for portions of the program that must run at certain speeds and should then be called *timing loops*.

TAB is used in a PRINT statement and is designed to act exactly like the TAB of a typewriter, including its faults. Several TABs can be used in one PRINT statement, but the arguments in the () must increase each time. That is, TAB cannot be used to move the cursor back to the left.

Use of a semicolon between TAB and the thing to be printed is not always necessary, but is recommended.

The Commodore 64 has a more general and powerful way of moving the cursor around. You simply put CRSR arrows in quotes in a PRINT statement. This will be illustrated in later lessons.

This lesson introduces loops in a painless way.

The delay loop is all on one line, with a colon to separate the NEXT statement. The amount of delay is determined by the size of the loop variable. A value of 1000 gives about a one-second delay.

After seeing that the primary work of the loop is simply to count until a particular value is reached before going on to the next instruction, it will be easier for the student to handle loops in which things are going on inside.

Questions:

1. Show how to write a delay loop that lasts for about two seconds.
2. Will this work for a delay loop?

```
120 FOR Q=1000 TO 5000  
122 NEXT Q
```

3. Tell what the computer will do in each case:

```
10 PRINT "HI";TAB(8);"GOOD LOOKING!"  
10 TAB(5);PRINT"OH-OH!"  
10 PRINT TAB(10);"NOPE";TAB(1);"NOT HERE"
```

4. What is the *argument* in this statement?

```
20 PRINT TAB(5);"E.T.CALL HOME"
```

Lesson 11. TAB and Delay Loops

Using TAB in PRINT Statements

TAB in a PRINT statement is like the TAB on a typewriter. It moves the PRINT cursor a number of spaces to the right.

(The PRINT cursor is invisible.)

The next thing to be printed goes where the cursor is.

Try this:

```
10 PRINT "123456789ABCDEF"  
30 PRINT TAB(3);"Y";TAB(8);"Z"
```

Rule: After TAB(N), the next character will be printed in column N+1.

Careful!

Run this:

```
10 TAB(5)
```

You see SYNTAX ERROR IN 10. TAB() has to be in a PRINT statement. You cannot use TAB(), or any function, by itself.

You Cannot TAB Backward

Try this:

```
10 PRINT "123456789ABCDEF"  
20 PRINT "A";TAB(9);"B";TAB(3);"C"
```

TAB() can only move the printing to the right. You cannot move back to the left.

Your Name Is Falling!

```
10 PRINT "{CLR}"
15 LET N=1
20 PRINT "YOUR FIRST NAME"
30 INPUT W$
40 PRINT TAB(N);W$
50 LET N=N+1
60 GO TO 40
```

Press RUN/STOP to stop the RUN.

This program prints your name in a diagonal down the screen, top left to bottom right. Try other values of N. Try changing lines:

```
15 LET N=15
50 LET N=N-1
```

How Big a Space Can TAB() Make?

There are 40 spaces across the screen. You can use any number 0 through 39 inside the TAB() parentheses. Larger numbers make the computer skip lines. Numbers larger than 255 will give an error message when the program runs:

```
ILLEGAL QUANTITY ERROR IN XX.
```

where XX is the line number.

You can use TAB with strings, too.

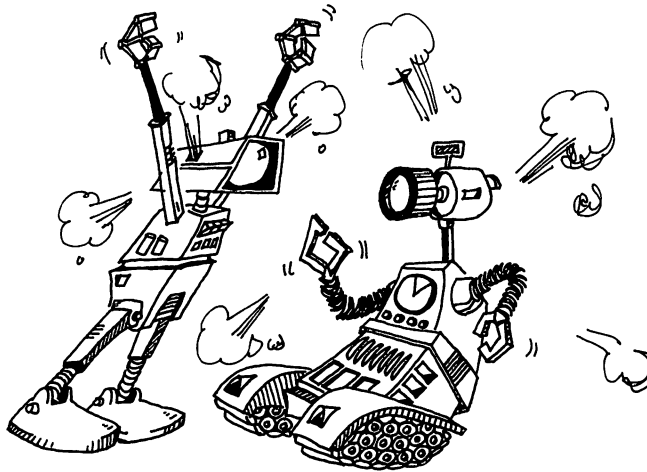
Example:

```
10 PRINT F$;TAB(10);M$;TAB(15);L$
```

Here F\$, M\$, and L\$ are the strings for the first, middle, and last names.

Functions Don't Fight But They Have Arguments

TAB() is like a function. We will study functions like RND(), INT(), LEFT\$(), etc. The number inside the () is called the *argument of the function*. TAB() says "move the cursor over" and the argument tells where to move it to.



Assignment 11A:

1. Write a program that asks for last names and nicknames. Then print the last name starting at column 1 and the nickname at column 10. Use a GOTO so the program is ready for another name/nickname pair.
2. Write an "insult" program. It asks your name. Then it writes your name and TABs over in the line and prints an insult.

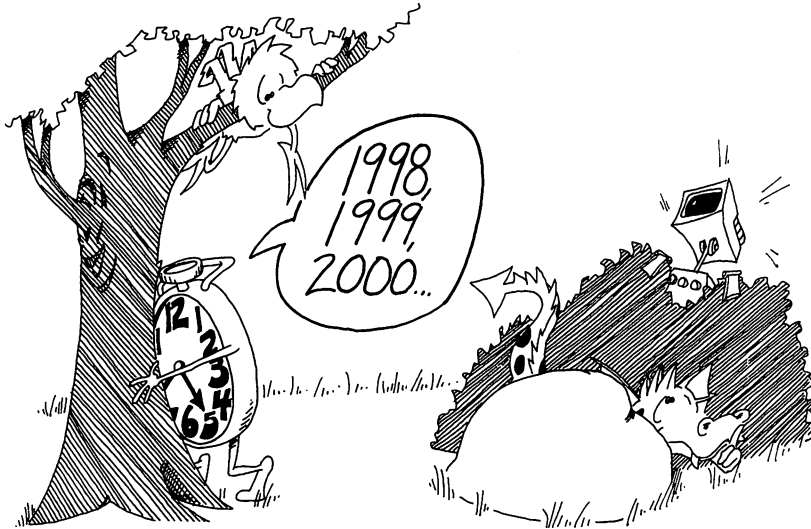
Delay Loops

Here is a way to slow down parts of the program. It is called a *delay loop*.

Run this program:

```
10 REM HIDE AND SEEK
20 PRINT "{CLR}"
30 PRINT "HIDE!"
40 FOR I=1 TO 2000:NEXT I
50 PRINT "COMING READY OR NOT!"
```

Line 40 is the delay loop. The computer counts from 1 to 2000 before going on to the next line. It is like counting when you are "it" in a game of hide and seek.



Try changing the number 2000 in line 40 to some other number.

Each 1000 in the delay loop is worth about one second of time. Try this:

```
10 REM -- TICK TOCK --
20 PRINT "{CLR}"
30 INPUT "WAIT HOW LONG";S
36 T=S*1000
40 FOR Q=1 TO T:NEXT Q
45 PRINT
50 PRINT S;" SECONDS ARE UP"
```

Assignment 11B:

1. Write a slowpoke program that prints out a three-word message with several seconds between each word.
2. Write a digital clock program. It uses a timing loop to count seconds. INPUT the present time in hours, minutes, and seconds. The clock then counts seconds and prints them out. When 60 seconds have gone by, add one to the minutes and put seconds back to zero. Same with hours. Run the clock a long time and adjust the timing loop so that the clock keeps good time.

Instructor Notes 12. The IF Statement with Numbers

The IF statement is extended to numerical expressions. The logical relations used in this lesson are:

= > < <>

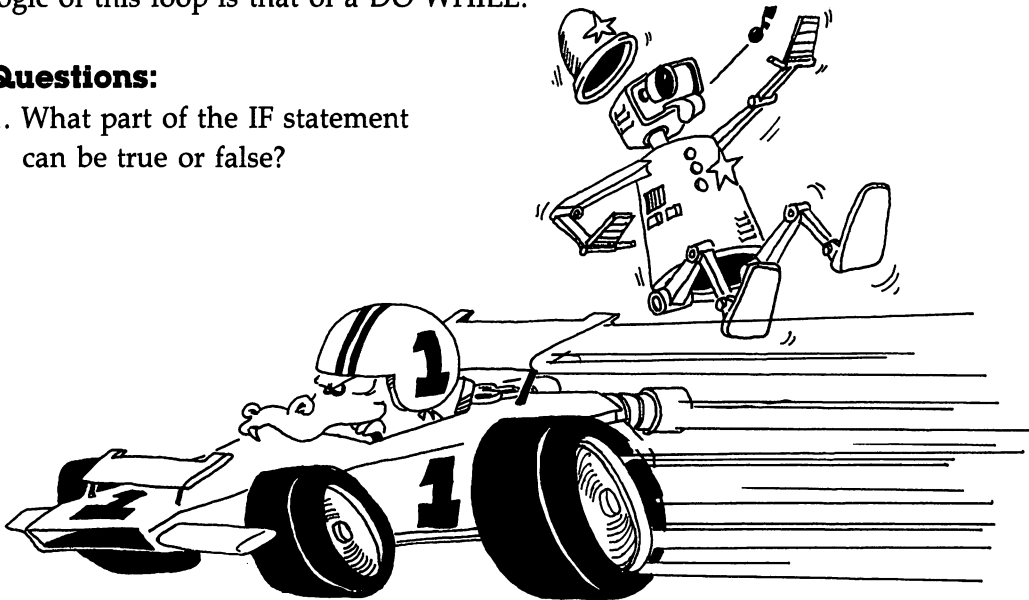
It is a good idea to get the student to pronounce these expressions out loud. $A < B$ makes a lot more sense when pronounced "A is less than B" than when it's just allowed to flow over the eyeballs. The point (the little end) of the $<$ and the $>$ symbols points to the smallest of the two numbers.

The use of nested IFs is demonstrated. This is a very powerful construction, but may be confusing. Go through the example with your student to make sure that the construction is understood.

A homemade loop is demonstrated in the "Guessing Game," but not discussed. The loop starts in line 50 and goes to 80. The exit test is made in line 57. The logic of this loop is that of a DO WHILE.

Questions:

1. What part of the IF statement can be true or false?



2. What follows the THEN in an IF statement?

3. After this little program runs, what will be in box D?

```
10 LET D=4  
15 IF 3<7 THEN LET D=9
```

4. Same question, but for $3 > 7$.

Lesson 12. The IF Statement with Numbers

Try this:

```
10 REM *** TEENAGER ***
15 PRINT "{CLR}"
20 PRINT "YOUR AGE?"
30 INPUT A
40 IF A<13 THEN PRINT "NOT YET A TEENAGER!"
50 IF A>19 THEN PRINT "GROWN UP ALREADY!"
```

This IF statement is like the one that you used before with strings. Again we have:

10 IF *phrase A* is true THEN do *statement C*

Phrase A can have these arithmetic symbols:

= equal to
> greater than
< less than
<> not equal to

Each phrase A is written with math symbols, but you should say it out loud in English. For example:

$A <> B$ is pronounced *A is not equal to B.*

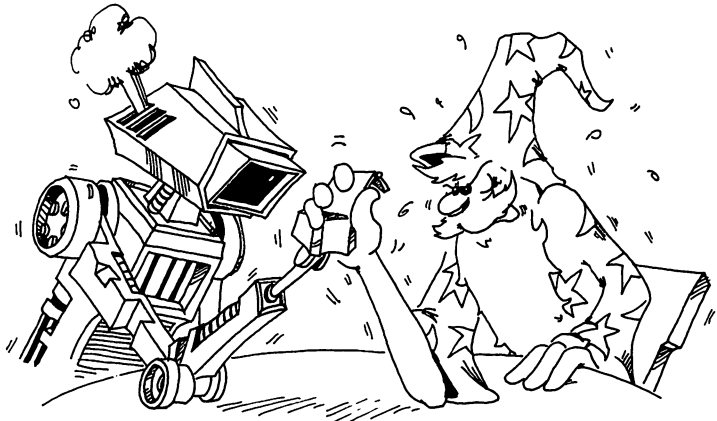
$5 < 7$ is pronounced *five is less than seven.*

Practice

For these examples, LET A=7 and LET B=5 and LET C=5:

Say each phrase A out loud and tell if it is true or false:

A=B	T	F
A>C	T	F
A>B	T	F
B=C	T	F
A<B	T	F
B<C	T	F
A=C	T	F
B<>C	T	F



An IF Inside an IF

The "Teenager" program above is missing something. Add:

```
60 IF A>12 THEN IF A<20 THEN PRINT "TEENAGER!"
```

To understand this, break it into two parts:

```
60 IF A>12 THEN (statement C) where
```

(*statement C*) is (IF A<20 THEN PRINT "TEENAGER!")

This line first asks, "Is the age greater than 12?"

If the answer is yes, the line gets to ask the second question, "Is the age less than 20?"

If the answer is again yes, the line prints TEENAGER!

If the answer to either question is no, the PRINT statement is not reached, so nothing is printed.

Assignment 12A:

1. Draw the "fork in the road" diagram for line 60 above. There will be two forks on the diagram.

Guessing Game

```
10 REM GUESSING GAME
15 POKE 53281,0
20 PRINT "{CLR}TWO-PLAYER GAME"
30 PRINT "{DOWN}{CYN} FIRST PLAYER HIDE YOUR EYES!"
"
32 PRINT "{DOWN}{GRN} SECOND PLAYER:"
34 PRINT " ENTER A NUMBER FROM 1 TO 100{DOWN}"
40 INPUT N
45 PRINT "{CLR}"
50 PRINT "{DOWN}{YEL}MAKE A GUESS "
55 INPUT G
57 IF G=N THEN GOTO 90
60 IF G<N THEN PRINT "TOO SMALL"
65 IF G>N THEN PRINT "TOO BIG"
80 GOTO 50
90 REM GAME OVER
95 PRINT "{CLR}{RED}{3 DOWN}THAT'S IT!{WHT}"
```

If you want to save this program on a disk, read lesson 14.

Usually, line 80 sends you to line 50 so that you can make more guesses. But if $G = N$ in line 57, then you skip to line 90 and print THAT'S IT!



Assignment 12B:

1. What happens in each line if G is 31 and N is 88:

50 _____
55 _____
57 _____
60 _____
65 _____
80 _____

What happens if G is 88 and N is 88:

50 _____
55 _____
57 _____
60 _____
65 _____
80 _____

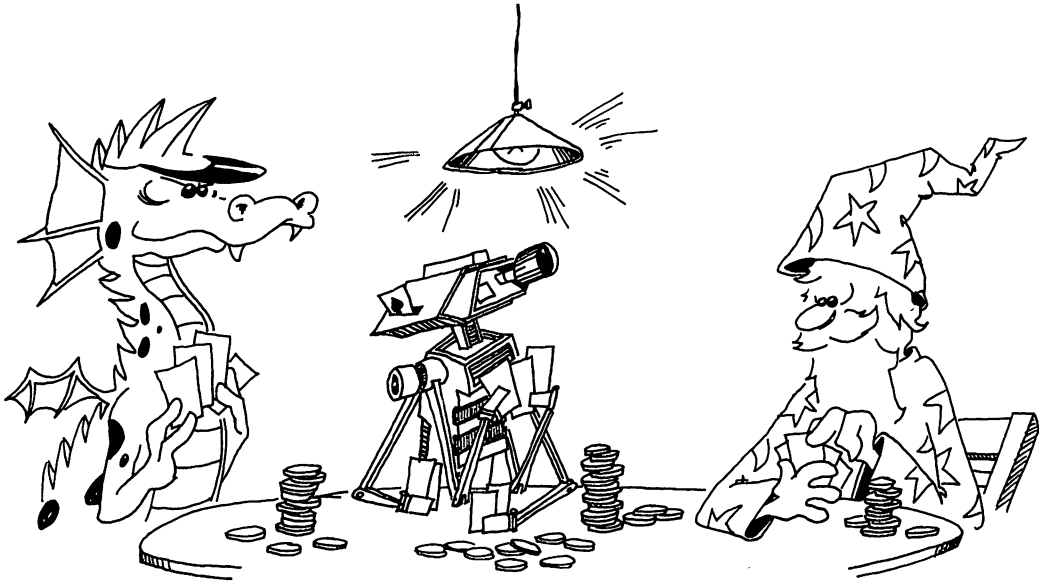
2. Here is another program. What will it print, and how many times?

```
10 LET N=1  
15 PRINT N;  
20 IF N=13 THEN PRINT "{CYN}UNLUCKY!{WHT}"  
30 LET N=N+2  
40 IF N>30 THEN GOTO 99  
50 GOTO 15  
99 PRINT "DONE"
```

What will it print if line 10 is changed to:

```
10 LET N=2
```

3. Write a program that says something about each number from one to ten. The player enters a number and the computer prints something about each number: "Three strikes, you're out" or "Seven is lucky," etc.
4. Write a game for guessing a card that player 1 has entered. Then player 2 must enter the suit (club, diamond, heart, or spade) and the value (1 through 13) of the card. The player first guesses the suit, then the program goes on to ask the value. Keep score.



Instructor Notes 13. Random Numbers and the INT Function

This lesson introduces two functions: RND and INT. These are very important in games and are also handy in making interesting displays like kaleidoscopes.

The RND function produces pseudorandom decimal numbers larger than 0 and smaller than 1. Such numbers are directly usable as probabilities, but integers in a specific range, such as 1 to 6 for a die, or 1 to 13 for a suit of cards, are often more directly usable.

Your student may be shaky in decimal arithmetic, but all that is required here is multiplication of the random number by an integer, and perhaps also addition to an integer. The computer does the multiplication, of course, so only a rough idea of the desired result is necessary.

After extending the random number to a range larger than zero to one, conversion to an integer is desired. The INT function does this by simply truncating the number, deleting the decimal part. (For negative numbers the situation is a little more complicated, and that rare case is not treated here.)

The concept of *rounding off* may be familiar to your student. INT will round off a number if you first add 0.5 to it.

The concept of functions is again used in this lesson and is further clarified.

The nesting of one function in the parentheses of another is illustrated by using RND in the argument of an INT function.

Questions:

1. Tell what the computer will print for each case:

```
10 PRINT INT(G)
```

and the box G contains: 2, 2.1, 2.95, 3.001, 67, 0, 0.2

2. Tell how the INT() function is different from rounding off numbers. Which is easier for you to do?

3. Tell how to change a number so that the INT() function will round it off.

4. What does the RND(8) function do?

5. How can you get random integers from 0 through 10? (*Hint: INT(RND(8)*10) is not quite right.*)

6. How can you get random integers from 5 through 8?

Lesson 13. Random Numbers and the INT Function

The RND Function

When you throw dice, you can't predict what numbers will come up.

When dealing cards, you can't predict what cards each person will get.

You need some way to roll dice and deal cards and do other unpredictable things with the computer.

Use the RND function to do this. RND stands for *random*.

Run this program:

```
10 REM RANDOM NUMBERS
20 PRINT "{CLR}{2 DOWN}"
25 LET N=RND(8)
30 PRINT N
40 IF N<.95 THEN GOTO 25
```

You see a lot of decimal numbers on the screen. The RND function in line 25 made them.

It doesn't matter what number you put in the parentheses just so long as it is greater than 0. I chose 8 because it is near the () signs on the keyboard, making it easy to type (8).

RND gives numbers that are decimals larger than 0 but smaller than 1. To make numbers larger than 1, you just multiply.

Change the program above to:

```
10 REM RANDOM NUMBERS
20 PRINT "{CLR}{2 DOWN}"
25 LET N=RND(8)*52
30 PRINT N
40 IF N<45 THEN GOTO 25
```

and run it again.



Now the numbers are between 0 and 52 in size. They could be used for choosing the 52 cards in a deck.

But:

We usually want whole numbers like 7 and 23 rather than decimal numbers like 7.03 and 23.62. Get them by using the INT function.



The INT Function

The INT function takes the number in its parentheses and throws away the decimal part, leaving an integer (a whole number). Change the program above and run again:

```
29 N=INT(N)
```

How It Works

Use this one-line program:

```
10 PRINT INT(2.5)
```

to check how INT() works. Run it many times and try these numbers in the (): 0.3, 0.5, 0.9, 1.0, 1.1, 1.49, 1.51, 1.999. In each case, see that INT() just throws away the decimal part of the number.

Rounding Off Numbers

Perhaps you know about *rounding off* numbers. If the decimal part starts with 0.5 or above, you round up. If it starts with 0.4 or below, you round down.

17.02	round down	17
3.1	down	3
103.43	down	103
4.5	up	5
82.917	up	83

You round off numbers with the INT function by first adding 0.5 to the number.

Run:

```
10 REM ### ROUNDING OFF ###
20 PRINT "{CLR}{DOWN} GIVE ME A DECIMAL NUMBER"
25 INPUT N
30 PRINT " ROUNDED TO THE NEAREST INTEGER"
40 PRINT INT(N+0.5)
45 FOR T=1 TO 1000:NEXT T
50 GOTO 20
```

Try the program with numbers like 3.4999 and 3.5, and other numbers you choose.

Rolling the Bones

Most dice games use two dice. One of them is called a *die*. Here is a program that acts like rolling a single die:

```
10 REM /// ONE DIE ///
20 PRINT "{CLR}{3 DOWN}"
30 LET R=RND(8)
40 PRINT " RANDOM NUMBER";TAB(15);R
50 LET S=R*6
55 PRINT "{DOWN} TIMES 6";TAB(15);S
60 LET I=INT(S)
65 PRINT "{DOWN} INTEGER PART";TAB(15);I
70 LET D=I+1
75 PRINT "{DOWN} DIE SHOWS";TAB(15);D
80 FOR T=1 TO 2000:NEXT T
85 GOTO 20
```

What Goes Inside the ()?

Numbers: 10 LET X=INT(34.7)

Variables: 10 LET X=INT(J)

Expressions: 10 LET X=INT(3*Y+2)

Functions: 10 LET X=INT(RND(8))

Here is how to save a lot of room.

Instead of:

```
30 LET R=RND(8)
50 LET S=R*6
60 LET I=INT(S)
70 LET D=1+I
```

Use just:

```
70 LET D=1+INT(RND(8)*6)
```

Random Numbers in the Middle

Suppose your game needs random numbers between 6 and 8. You have a funny die that shows only 6, 7, or 8 when you roll it.

Run this:

```
10 LET D=INT(RND(8)*3)+6
```

How it works: **Expression**

Makes numbers from:

	small	large
RND(8)	0.01	0.99
RND(8)*3	0.03	2.97
INT(RND(8)*3)	0	2
INT(RND(8)*3)+6	6	8

Assignment 13:

1. Write a program that "rolls" two dice, called D1 and D2. Show the number on D1 and on D2 and the sum of the dice. You do not need the variables R, S, and I in the program above. They were used to show how the final answer was found.
2. Write a program that shows the roll of a special die. It is a cube (six sides) and the sides have the numbers 10, 12, 14, 16, 18, and 20 on them.
3. Write a "paper, scissors, and rock" game, with you against the computer. (Paper wraps rock, rock breaks scissors, scissors cut paper.) The computer chooses a number 1, 2, or 3 using the RND() function: 1 is paper, 2 is rock, 3 is scissors. You INPUT your choice as P, R, or S and the computer figures out who won and keeps score.

Instructor Notes 14. Save to Disk

This lesson assumes that the student has a disk formatted for use with the 1541 disk drive. (See p. 243, "Disk Usage.")

This lesson shows how to save programs to the disk and how to load them again.

These commands are introduced:

SAVE " <i>filename</i> ",8	save program
LOAD " <i>filename</i> ",8	load program
LOAD "\$",8	load directory
OPEN 15,8,15,"SØ: <i>filename</i> ":CLOSE 15	erase program from disk

Other reserved words used in this chapter are NEW, REM, PRINT, and LIST.

This lesson can be used anytime after lesson 3.

We put it this late in the book because most programs up to this point are relatively short and uninteresting, not worth saving. The process of programming was being emphasized, not the end result of useful programs.

However, your own judgment should prevail. You can insert this chapter at an earlier point in the flow of lessons so that programs which your student is particularly proud of can be saved.

Questions:

1. What is a file?
 2. How long can a filename be?
 3. Can punctuation marks be in a filename? Can the filename have spaces in it?
 4. How can you check that the program got on disk okay?
-
-

-
-
5. What happens to the program already in memory if you load the directory? If you load another program?
 6. Does the filename have to be the same as the program name?
 7. If a program is put into a file, is it still in memory?

Lesson 14. Save to Disk

If you have a 1541 disk drive, you can store programs on a floppy disk.

Entering a Program

If you already have a program in the computer, skip down to "Saving a Program."

If not, enter:

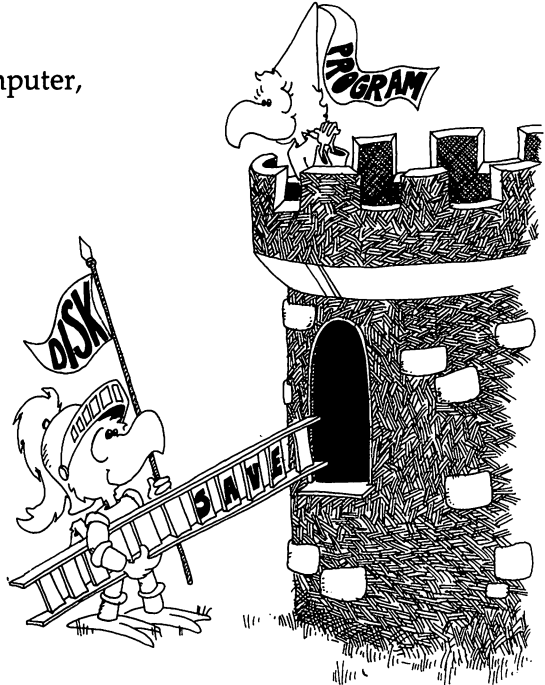
```
NEW
10 REM HOT DOG
20 PRINT "NO MUSTARD"
```

Saving a Program

Do you still have your disk in the drive?
If not, put your disk in now. Be sure to close the door!

Enter:

```
SAVE "HI",8
```

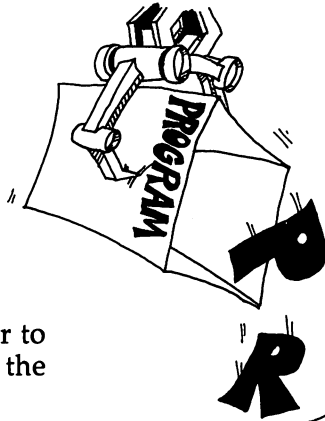


You will hear a whirring and see the red light on the disk drive come on. When the red light goes out and the whirring stops, your program is on the disk.

But if you see the red light flashing on the disk drive, it means you have a disk error. Check the write-protect notch on the side of the disk. If there is a paper tab stuck over the write-protect notch, you cannot save to the disk unless you remove the paper tab. (Be sure that you really want to save to this disk. Ask yourself why the paper was put over the notch; maybe this disk has some important programs on it.) Or maybe you already have a program by the same name on the disk.

The Filename

The filename of your program is HI. The disk is like a file cabinet. There is a file folder in it with the name HI written on it. In the file folder is your program.



We used HI because it is easier to remember the filename if it is the same as the program name.

If your program has a different name, save it again under the correct name.



The Device Number

The number 8 in the SAVE command is the device number. The disk drive is device number 8.

If you forget to type the ,8 in the SAVE command, the computer thinks you want to save to cassette tape. Press the RUN/STOP key to get back to normal.

The Disk Directory

The disk has a file called the *directory* that is a list of all the program names on the disk.

Let's see if your program is really stored on the disk.

Enter:

```
LOAD "$",8  
LIST
```

This loads the directory but erases your program! So do not ever load the directory if you suspect your program did not get saved properly!

After a whirring, you will see a list of all the files on the disk. Your file is probably the last one in the list. It will say:

```
1  "HI"  PRG
```

The PRG means the file folder contains a program.

The 1 means it is a short program, only taking up one block on the disk. Think of each block as a page in the folder HI in the file cabinet.

Loading the Program

Enter:

```
LOAD "HI",8
```

You hear the whirring and see the red light, but is your program now in memory?

Enter: LIST

to find out.



Erasing a File

So far, so good. But what if you change your mind and want to throw a file away?

Enter:

```
OPEN 15,8,15, "SØ:HI":CLOSE 15
```

The letter S stands for SCRATCH. You want to scratch the file out!

Is It Gone?

Let's see if the program is gone from the disk directory.

Enter:

```
LOAD "$",8  
LIST
```

to see if it is really gone.

Legal Filenames

The filename:

- can be long (up to 16 characters).
- can have letters, numbers, or some punctuation in it.
- but cannot have , or : or * or ? in it.

Good Filenames

A short filename is best because there is less to type. Use the same name that is in the REM in the first line of the program.

Good names:

```
JUMPING CAT  
GUESSING GAME  
SUB &e%$#  
64 DISK BACKUP
```

Wrong names:

CAT, DOG (has a comma in it)
THIS IS MUCH TOO LONG (too long)

If you use a filename that is too long, the program will still be saved, but the name will be shortened to 16 characters.

Commands

The 1541 disk drive is smart. It has its own microprocessor chip and memory chips. Read the *VIC-1541 User's Manual* to find out all about it.

These three commands are used with files:

SAVE“ <i>filename</i> ”,8	to save the program
LOAD“ <i>filename</i> ”,8	to load the program
LOAD“\$”,8	to see the directory

Use OPEN 15,8,15,“SØ:*filename*”:CLOSE 15 to erase a program.

Where you see the word *filename* you must type the name of a file.

Assignment 14:

1. Write a short program (four lines) and save it on the disk.
2. Do NEW and write another short program. Save it.
3. Do NEW and then do LOAD“\$” and LIST to see if the programs were saved. Then load each program and run it.
4. Try out the SCRATCH command on one of the programs.
5. Repeat the practice with the SAVE, LOAD, directory, and SCRATCH commands until you are sure that you understand them.

Instructor Notes 15. Some Shortcuts

?	Used for PRINT
LET	Omission
:	Used between statements on a line
INPUT	Used with a message
INPUT	Error messages
LIST X-Y	Lists specific sections of a program
THEN 33	Instead of THEN GOTO 33

Having reached RND and the saving of programs on disk, the sprint is over. All the elements are in place for the student to write substantial programs.

The colon is used to shorten and clarify programs by putting several statements on one line. A line should contain statements that have something in common.

The colon allows you to put a short subroutine consisting of several statements after an IF. This makes using a GOTO unnecessary for reaching the extended segment of a program. A shorter and much less cluttered program results. So the colon becomes a powerful and nontrivial means of improving the clarity of the program.

The colon can mess up a program, too. Be careful about adding other statements onto a GOTO, a REM, or an IF line.

A question mark is always printed on the screen by INPUT. So an INPUT message should not end with a question mark.

Questions:

1. What shortcut does the ? give?
2. How can you tell that the word LET is missing from a LET statement?
3. An INPUT statement has a message in quotation marks. What punctuation mark must follow the message quotes?

4. Why is it sometimes good to put two statements on the same line, separated by a colon?

5. What is wrong with each of these lines?

```
10 REM BEGINNING:GOTO 1000  
10 GOTO 50:S$="FAST"
```

6. If the computer prints ?? after you answer an INPUT, what does it mean?

Lesson 15. Some Shortcuts

A PRINT Shortcut

Instead of typing PRINT, just type a question mark.

Enter: 10 ? "HI"
LIST

The computer substitutes the word PRINT for the question mark.

A LET Shortcut

These two lines do the same thing:

10 LET A=41 and 10 A=41

Also these two: 20 LET B\$="HI" and 20 B\$="HI"

You can leave out the word LET from the LET statement. The computer knows that you mean LET whenever the line starts with a variable name followed by an = sign.



An INPUT Shortcut

Instead of:

```
10 PRINT "ENTER YOUR NAME"  
20 INPUT N$
```

You can do:

```
10 INPUT "ENTER YOUR NAME";N$
```

Put a semicolon between the message ENTER YOUR NAME and the variables.

Another INPUT Shortcut

You can INPUT several things in one statement. Put commas between the variables.

Run:

```
20 INPUT "LOCATION";X,Y  
LOCATION?
```

You see: LOCATION? on the screen.

You enter two numbers with a comma between them.

LOCATION? 5,6

Another example:

```
30 INPUT "MONTH, DAY, YEAR";M$,D,Y
```

After the ? type: APRIL,29,1985

Error Message in INPUT

If you do not enter enough answers, the computer asks ??. Then you should enter the rest.

Example:

```
30 INPUT "MONTH, DAY, YEAR"; M$, D, Y
```

```
? MAY, 1  
??1985
```

If you enter too many answers, the computer replies:

```
?EXTRA IGNORED
```

and goes on with the program.

Example:

```
55 INPUT "SLEEPY?<Y/N>"; A$
```

```
? NO, FINE  
?EXTRA IGNORED
```

Another Error Message

Run:

```
10 INPUT N, A$
```

Try these pairs of answers:

```
1, B    B, 1    1, 1    B, B
```

The error message ?REDO FROM START is put on the screen whenever the user answers with a string for a number.

(It is okay to answer with a number for a string, because the computer says 1984 is a string!)

A LIST Shortcut

There are five ways to use the LIST command:

LIST	Lists whole program
LIST 48	Lists line 48
LIST 50-75	Lists all lines from 50 to 75
LIST -27	Lists all lines from beginning to 27
LIST 90-	Lists all lines from 90 to the end

A THEN Shortcut

Instead of:

```
10 IF A=B THEN GOTO 33
```

Use:

```
10 IF A=B THEN 33
```

A Colon Shortcut

Put several statements on a line with a colon (:) between them. This saves space.

Instead of

```
10 Q=17*3
20 R=Q+2
30 PRINT R
```

you can write:

```
10 Q=17*3:R=Q+2:?R
```

When you LIST the line, you see:

```
10 Q=17*3:R=Q+2:PRINT R
```

When to Use the Colon Shortcut

Use the shortcut:

1. To make the program clearer.

Put similar statements on the same line. Example:

Instead of:

```
10 X=0
12 Y=0
14 Z=0
```

write:

```
10 X=0:Y=0:Z=0
```

2. To make the program shorter.
3. To put a REM on the end of the line.

Example:

```
40 H=X+Y/66:REM H IS THE HEIGHT
```

The Colon After an IF Statement

You can make neater IF statements using colons.

Without colons:

```
50 IF A=0 THEN GOTO 80
60 B=Q
62 C=B*D
66 PRINT "WRONG"
80 FOR....
```

With colons:

```
50 IF A<>0 THEN B=Q:C=B*D:PRINT "WRONG"
80 FOR....
```

All the statements in the path "A<>0 is true" are on the line after THEN.

Careful!

Do not put something on the end of an IF line that doesn't belong.

Example:

```
35 IF A=B THEN PRINT "ALIKE"
40 Q=R
```

is not the same as:

```
37 IF A=B THEN PRINT"ALIKE":Q=R
```

because Q = R in line 40 is always done, no matter if A = B is true or not. But Q = R in line 37 is done only if A = B is true.

Some More Mistakes with Colons

The REM and the GOTO statements must be last on a line. Anything following them is ignored.

Correct:

```
35 P=3:REM P IS THE PRICE
```

Wrong:

```
35 REM P IS THE PRICE:P=3
```

Because the computer ignores everything else on a line after reading REM.

Correct:

```
40 R=P+1:GOTO 88  
42 S=3
```

Wrong:

```
40 R=P+1:GOTO 88:S=3
```

Because the computer goes to line 88 and can never come back to S = 3.



Commands, Statements, and Lines

Commands and statements tell the computer to do something. So far we have used these commands and statements:

PRINT, NEW, RUN, LIST, REM, INPUT, LET, GOTO, IF, SAVE, LOAD

Statements are instructions to the computer which are usually used in numbered lines. Commands are usually used alone. NEW, SAVE, LOAD, and RUN are commands.

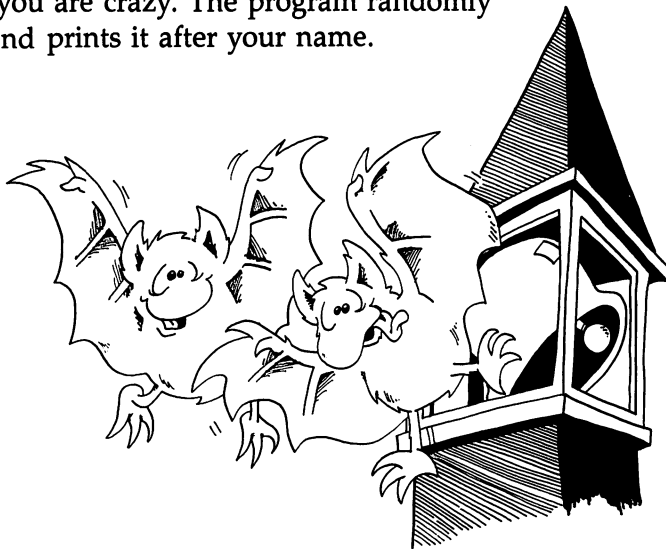
Some lines have several statements, separated by colons.

```
30 PRINT "{CLR}":PRINT:LET Z=55
```

is a line with three statements.

Assignment 15:

1. Write a program that uses each of these shortcuts at least once.
2. Write a "vacation" program. It asks how much you want to spend. Then it tells where you should go or what you should do.
3. Write a "crazy" program that asks your name. The program has three funny ways of saying you are crazy. The program randomly chooses one of these and prints it after your name.



Instructor Notes 16. Moving Pictures

The cursor arrow keys are used in PRINT statements to move the cursor to any part of the screen. Take care that each PRINT statement ends with a semicolon or else the cursor will drop down to the beginning of the next line and not be where you think it is!

Also, if you try to move the cursor lower than the bottom of the screen, the whole picture will scroll. A scroll will also occur if you print in the fortieth column on line 25. This won't scroll:

```
10 FOR I=1 TO 24:PRINT I:NEXT I
20 PRINT "12345678901234567890123456789123456789"
30 GOTO 30
```

But add a 0 to the end of the string (making 40 characters) and the screen will scroll.

To make moving pictures, you must erase the old picture before you draw the new one. This complication, on top of having to keep an accurate mental picture of where the invisible PRINT cursor is at all times, may overwhelm your student.

The remedy is slow, careful, and complete analysis of the PRINT statements and keeping close track of the cursor position. The best way to control the situation is drawing a grid to represent the screen, then lightly sketching the characters one by one (and erasing them as spaces are printed over them).

If your program runs in an unexpected way, it helps to put in a lot of delay loops so that you can accurately see the order in which PRINTing is done.

Questions:

1. What is the difference between "{CLR}" and "{HOME}" in a PRINT statement?

-
-
2. Show two lines in a program that together will put a letter A on the screen at the point three lines down and seven spaces across.
 3. In this line, the word HAPPY is printed. Where is the PRINT cursor after the word is printed?

```
10 PRINT "{HOME}{DOWN}HAPPY";
```

Can you see the PRINT cursor on the screen?

4. Now write a line 20 that will erase the letters APP from the word printed in question 3.

Lesson 16. Moving Pictures

Moving the Cursor Up and Down

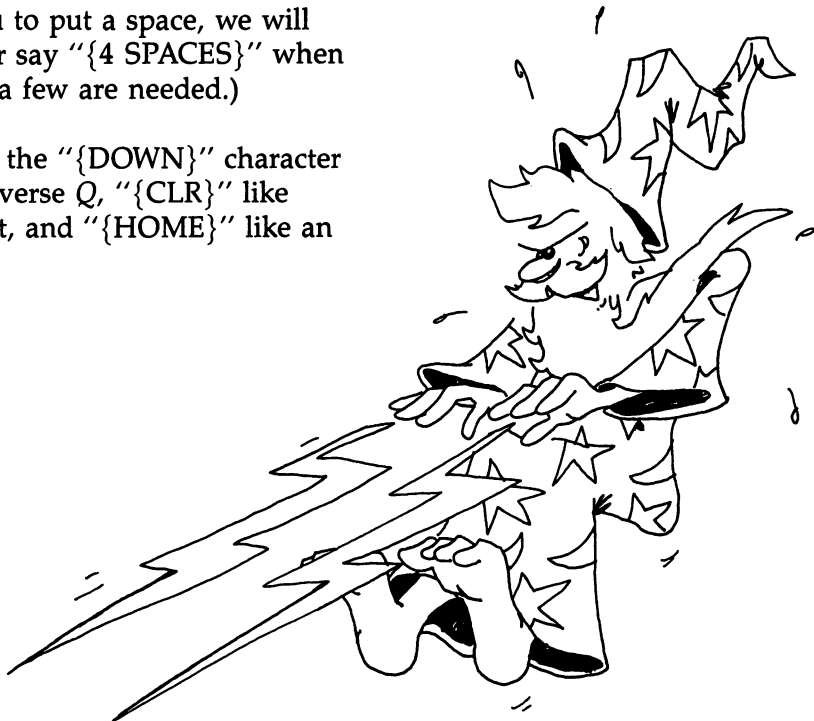
To move the cursor, put CRSR characters in a PRINT statement.

```
10 REM UP AND DOWN
20 PRINT"{CLR}{6 DOWN} LINE 6 FIRST";
25 FOR T=1 TO 1000:NEXT T
30 PRINT"{HOME}{3 DOWN} LINE 3 NEXT";
35 FOR T=1 TO 1000:NEXT T
40 PRINT"{HOME}{8 DOWN} LINE 8 LAST";
```

When we print "{6 DOWN}" in this book, we mean to put six CRSR-down characters in the PRINT quotes with no spaces between them. Remember, whenever you see something on a program line between braces, {}, it means something special: {HOME} means press the home key; {2 RIGHT} means press the CRSR-right key two times.

(If we want you to put a space, we will leave a space or say "{4 SPACES}" when more than just a few are needed.)

Remember that the "{DOWN}" character looks like an inverse Q, "{CLR}" like an inverse heart, and "{HOME}" like an inverse S.



To Reach a Spot on the Screen

1. Start your PRINT line by moving the cursor to the home position with "{HOME}" or a clear screen "{CLR}".
2. Then move the cursor down as far as you want.
3. Now PRINT right-CRSR characters to move over as far as you want. (Or you can use spaces instead of "{RIGHT}" characters.)
4. Then PRINT what you want.

A Moving Picture

Try this program:

```
10 REM ::: BIRD :::
20 PRINT"{CLR}{6 DOWN}";
25 PRINT"{7 RIGHT} {PUR}";
30 PRINT"UQI{3 LEFT}";
40 FOR T=1 TO 200:NEXT T
50 PRINT"JQK{3 LEFT}";
60 FOR T=1 TO 200:NEXT T
70 GOTO30
```

Here "{3 LEFT}" means three left-CRSR characters (remember to use the SHIFT key). We use "{7 RIGHT}" for right-CRSR characters.

Do you remember what to do when you see "{PUR}"? If you forgot, look at lesson 2.

Look closely at lines 30 and 50. We have another symbol that makes finding the correct key easier. An underscored character means to hold down the SHIFT key while pressing the character. In this case SHIFTeD UQI and SHIFTeD JQK will produce the graphics characters that appear on the right of each of the keys.

Which lines are the delay loops? _____

If you do it correctly, you will get a single purple bird slowly flapping its wings in the middle of the screen. If you get a lot of birds, check that you have the semi-colon at the ends of the PRINT lines, and that lines 30 and 50 have the birds, three left-CRSR characters, and no spaces.

How does it work?

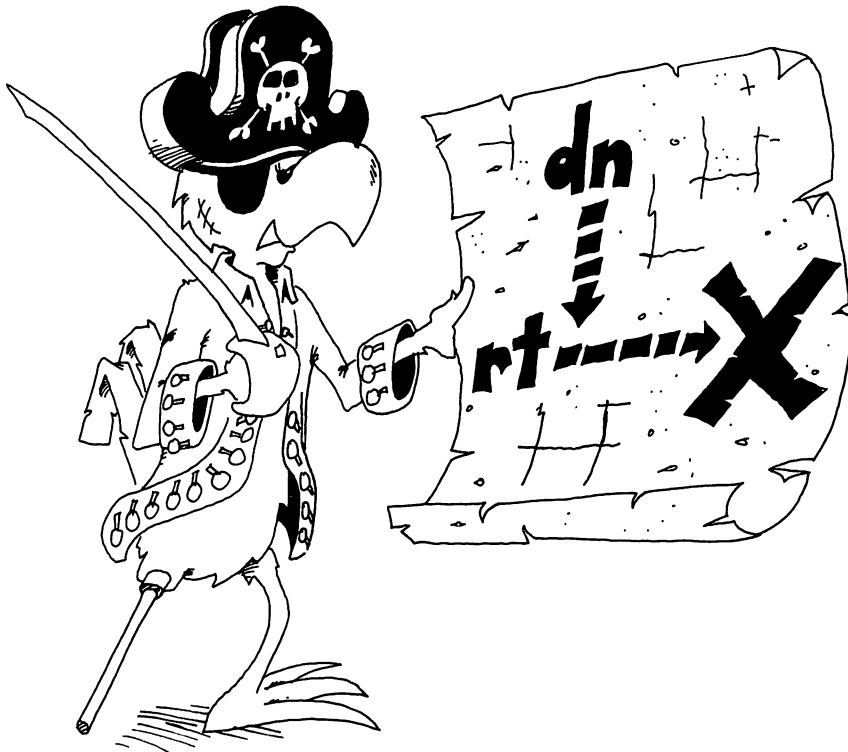
Read lines 30 and 50 character by character.

Line 30 prints left wing, body, right wing. Now the invisible PRINT cursor is to the right of the bird, so three left-CRSR characters bring the cursor back over the left wing of the bird.

Line 40 waits for you to see this bird with its wings down.

Line 50 prints a bird with wings up, on top of the first bird. Then it moves the cursor back over the left wing of the bird.

Line 60 waits for you to see this bird with its wings up. Then the cycle repeats.



Erasing Old Pictures

In "Bird" we erased the down-wing bird by writing an up-wing bird over it.

Suppose we want the bird to fly away?

We must erase the old bird with spaces before we print the new bird.

Change lines 10, 20, 25, 45, and 65 in "Bird":

```
10 REM FLY AWAY
20 PRINT"{CLR}{22 DOWN}"
25 PRINT"{PUR}"
30 PRINT"UQI{3 LEFT}";:REM PRINT DOWN WINGS
40 FOR T=1 TO 200:NEXT T
45 PRINT"{3 SPACES}{2 LEFT}{UP}";:REM ERASE BIRD,
   {SPACE}GO UP
50 PRINT"JQK{3 LEFT}";:REM PRINT UP WINGS
60 FOR T=1 TO 200:NEXT T
65 PRINT"{3 SPACES}{2 LEFT}{UP}";
70 GOTO 30
```

Assignment 16:

1. Write a program that makes a ball move across the screen, from left to right. Be sure to erase the old ball before PRINTing the new ball. Then change the program so the ball moves from right to left.
2. Write a program to make your first name print on the screen in red, then blink to green, and back to red, and so on. The name doesn't move on the screen.

Instructor Notes 17. FOR-NEXT Loops

A loop is made up of a FOR statement (which may contain a STEP value) and a NEXT statement. These statements may be separated by several lines and yet are interdependent. The student builds on the notion of a delay loop and learns the utility of repeating a set of statements in the middle of the loop.

Nested loops are introduced by using an example where the inside loop is a delay loop.

Subtle points may arise which are not discussed in this lesson. The loop is always traversed at least once, because the test for exit is made at the NEXT statement which can be reached only by going through the loop.

The FOR statement is evaluated just once at the time the loop is entered. It puts the starting value of the loop variable into variable storage where it is treated just as any other numerical variable. The STEP value, the ending value, and the address of the first statement after the FOR are put on a stack.

From now on, all the looping action takes place at the NEXT statement. Upon reaching NEXT, the loop variable is incremented by the value of the STEP and compared with the end value. If the loop variable is larger than the end value (or smaller in the case of a negative STEP), NEXT passes control to the statement after itself. Otherwise, it sends control to the statement after FOR.

Because BASIC treats the loop variable just like any other variable, it can be used or changed in the body of the loop. Jumping into the middle of a loop is usually a disaster. Jumping out of a loop before the NEXT causes an exit and in some cases (especially where subroutines are involved) may create hard-to-find bugs.

Questions:

1. What is the loop variable in this line?

```
10 FOR Q=1 TO 10:PRINT T$:NEXT Q
```

-
-
2. How do you make the loop count down instead of count up?
 3. What is meant by nested loops?
 4. Write a loop that prints the numbers from 0 to 20 by twos.
 5. Write a "Ten Little Indians" program that prints the numbers from 10 down to 0 Indians.

Lesson 17. FOR-NEXT Loops

Remember the delay loop? The computer counted from 1 to 2000 and then went on.

```
30 FOR T=1 TO 2000:NEXT T
```

The computer is smarter than that. It can do other things while it is counting.

Run this:

```
10 REM COUNTING
20 PRINT "{CLR}"
30 FOR I=5 TO 20
40 PRINT I
50 NEXT I
```

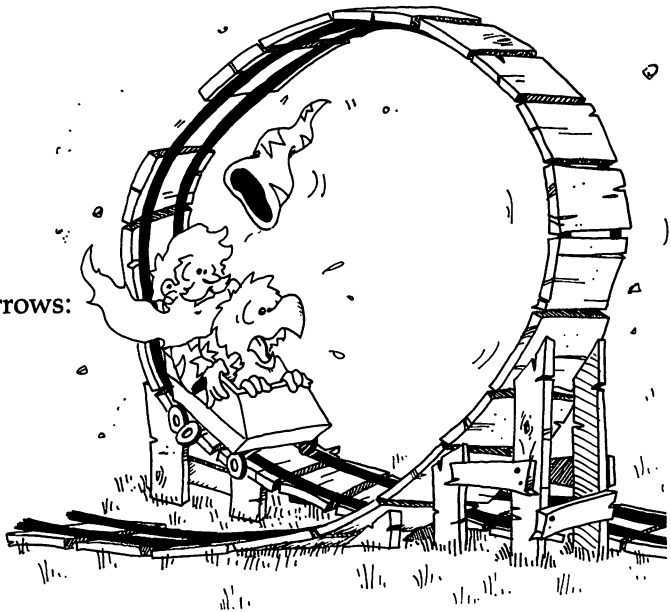
The loop can start on any number and end on any higher number. Try changing line 30 in these ways:

```
30 FOR I=100 TO 101
30 FOR I=7 TO 13
30 FOR I=1.3 TO 5.7
```

Mark Up Your Listings

Show where the loops are by arrows:

```
10 REM ON PAPER
20 PRINT "{CLR}"
30 FOR I=0 TO 7
40 PRINT I
50 NEXT I
```



STEPS

The computer was counting by ones in the above programs. To make it count by twos, change line 30 to this:

```
30 FOR I=10 TO 30 STEP 2
```

Assignment 17A:

1. Have the computer count by fives from 0 to 100.

Count Down Loops

You can make the computer count down by using a negative STEP.
Try this:

```
10 REM ** APOLLO 11 **
15 POKE 53281,15
20 PRINT"{CLR}"
30 PRINT" T MINUS 12 SECONDS AND COUNTING"
35 FOR I=1 TO 750:NEXT I
40 FOR T=11 TO 0 STEP -1
50 PRINT "{CLR}{DOWN}";T
60 FOR I=1 TO 750:NEXT I:REM TIMING LOOP
70 NEXT T
80 PRINT"{CLR}{DOWN}{RED} ALL ENGINES RUNNING, LIF
T OFF."
81 FOR I=1 TO 750:NEXT I
82 PRINT"{CLR}{DOWN}{GRN} WE HAVE A LIFT OFF."
83 FOR I=1 TO 750:NEXT I
84 PRINT"{CLR}{BLU}{DOWN} 32 MINUTES PAST THE HOUR
."
85 FOR I=1 TO 750:NEXT I
86 PRINT"{CLR}{DOWN}{CYN} LIFT OFF ON APOLLO 11."
90 POKE 53281,0
```

Line 60 is the timing loop for counting seconds. Lines 35, 81, 83, and 85 slow down the printing.

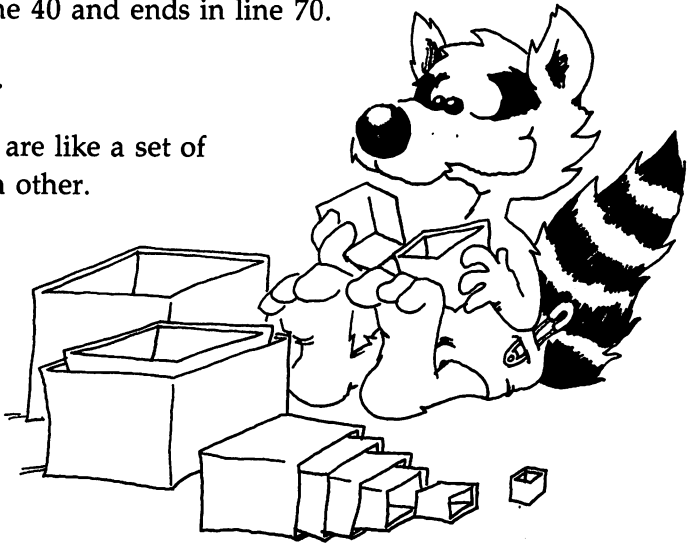
Nested Loops

In this program, we have one loop inside another.

The outside loop starts in line 40 and ends in line 70.

The inside loop is in line 60.

These are *nested loops*. They are like a set of toy boxes that fit inside each other.



Loop Variables

To make sure that each FOR statement knows which NEXT statement belongs to it, the NEXT statement ends in the *loop variable* name. Look at line 60:

```
60 FOR I=1 TO 2000:NEXT I
```

I is the loop variable for the loop starting in line 40:

```
40 FOR T =12 TO 0 STEP-1  
.....  
70 NEXT T
```

T is the loop variable.

Badly Nested Loops

The inside loop must be all the way inside:

Right:

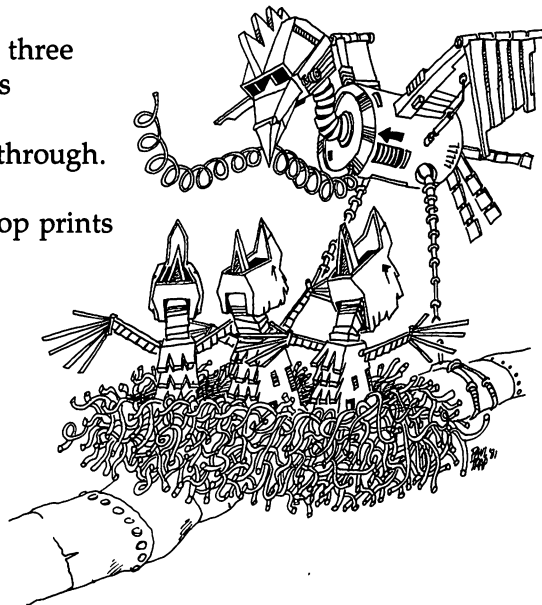
```
25 FOR X=3 TO 7  
30 FOR Y=3 TO 7  
40 PRINT X*Y  
50 NEXT Y  
60 NEXT X
```

Wrong:

```
25 FOR X=3 TO 7  
30 FOR Y=3 TO 7  
40 PRINT X*Y  
50 NEXT X  
60 NEXT Y
```

Assignment 17B:

1. Write a program that prints your name 15 times.
2. Make it indent each time by two spaces more. It will go diagonally down the screen. Use TAB in a loop.
3. Now make it write your name 23 times, starting at the bottom of the screen and going up. Use the cursor-up key in a PRINT statement and put it all in a loop.
4. Now make it write your name on one line, your friend's name on the next, and keep switching until each name is written five times. Make each name a different color.
5. Write a program with loops nested three deep. Do the outer loop three times and have it print SING. Make it change the screen color each time through. The next loop prints TRA. Have it loop three times. The innermost loop prints LA three times.



Instructor Notes 18. DATA, READ, RESTORE

This lesson concerns the DATA statement. READ gets data from the DATA statements and RESTORE puts the pointer back to the beginning of the first DATA statement.

The storing of data in DATA statements has a few confusing elements when first confronted. You can never change any of the data in the statement unless you re-write the program. Of course, you can READ the data into a variable box, then change what's in the box.

You must READ the data to be able to use it. It must be read in order, starting from the beginning. If you want to skip some data, you have to read and throw away the stuff before it. (This procedure is not discussed here and may be mentioned to the student when other ideas about DATA are well entrenched.)

The idea of a *pointer* is used in this lesson. A pencil in the instructor's hand, pointing to items in a DATA statement, helps clarify this concept.

Using DATA prevents some error-prone typing if you have a lot of data.

However, it is also useful in cases where there is not very much data, because it clearly separates the actual data from the processing of the data. This helps when debugging programs.

One of the commonest uses of DATA is to fill arrays with initial values. We also use it in the "Music" lesson to store note pitches and durations.

Questions:

1. What happens if you try to READ more data items than are in the DATA statements?
2. What rule tells you where to put the DATA statements in the program? How about where to put the READ statements?

-
-
3. The idea of a pointer helps in thinking about the DATA statements. Explain how.
 4. Can you put numerical data and string data in the same DATA statement?
 5. What happens if you try to READ a string into a numerical variable?
 6. Can you change the items in a DATA statement while the program runs?

Lesson 18. DATA, READ, RESTORE

Two Kinds of Data

There are two kinds of data in your programs:

1. The kind you INPUT or GET through the keyboard.

```
10 REM FIRST KIND OF DATA
20 PRINT"{CLR}{3 DOWN}"
30 INPUT" YOUR PET PEEVE";P$
40 PRINT"{DOWN} REALLY!"
50 PRINT"{DOWN}{RED} YOU DON'T LIKE {DOWN}{GRN}"
60 PRINT"{SHIFT-SPACE}";P$;"{WHT}"
```

In this program, P\$ is data entered by the user as the program runs.

2. The kind that is stored in the program at the time it is written.

```
10 REM THE SECOND KIND OF DATA
20 PRINT"{CLR}{3 DOWN}"
30 X=57
40 Y$="FLAVORS"
50 PRINT X;Y$
```

In this program, X and Y\$ are data stored in the program by the programmer when the program was written.

Storing Lots of Data

It is okay to store small amounts of data in LET statements. But it is awkward to store large amounts of data that way.

Use the DATA statement to store large amounts of data.

Use the READ statement to get the data from the DATA statement.

```
10 REM LOTS OF DATA
20 PRINT"{CLR}{3 DOWN}"
30 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,F
RIDAY,SATURDAY
40 READ D1$:READ D2$:READ D3$:READ D4$
60 PRINTD1$,D2$,D3$,D4$
```

After the program runs, box D1\$ holds the first item in the DATA list (SUNDAY) and box D2\$ holds the second (MONDAY), etc.

Strange Rules

1. It doesn't matter where the DATA statement is in the program.

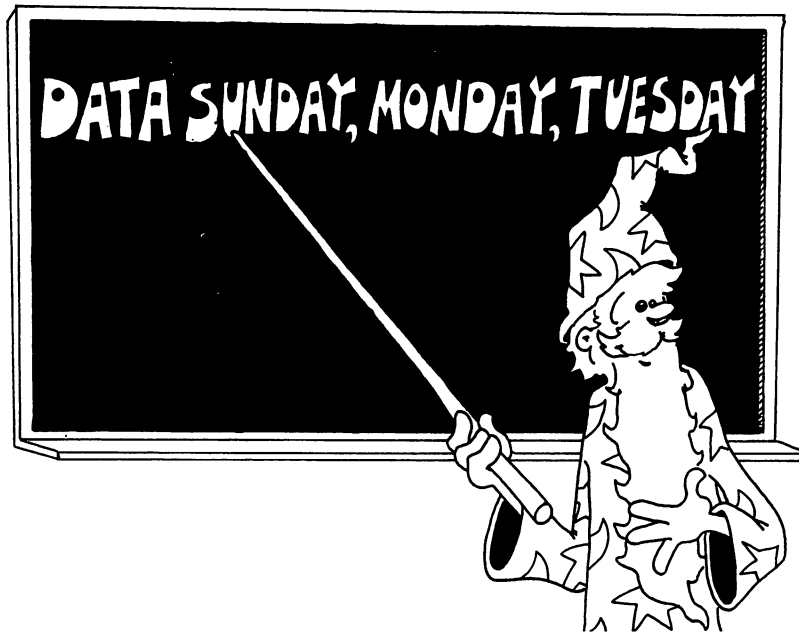
Do this: Change line number 30 in the above program to line number 90. Run the program. It works just the same.

2. It doesn't matter how many DATA statements there are.

Do this: Break the DATA statement into two parts:

```
90 DATA SUNDAY, MONDAY, TUESDAY
91 DATA WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
```

Run the program. It works just the same as before.



It Is Polite for the READ Statement to Point

READ uses a *pointer*. It always points to the next item to be read.

When the program starts, the READ pointer points to the first item in the first DATA statement in the program. (That is, the DATA statement with the lowest line number of all DATA statements in the program.)

Each time the program executes a READ statement, the pointer moves to the next item in the DATA list.

If the pointer gets to the end of one DATA statement, it automatically goes to the next DATA statement. (That is, to the DATA statement with the next higher line number.)

It doesn't matter if there are many lines between.

Do this: Change line 90 back to line 30. (Leave line 91 alone.)

```
30 DATA SUNDAY, MONDAY, TUESDAY
```

```
.....
```

```
91 DATA WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
```

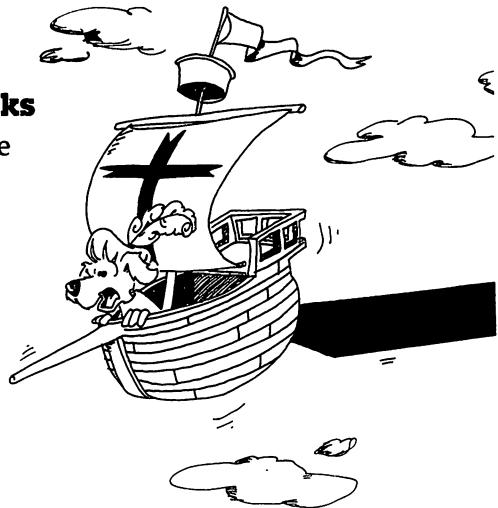
Run the program. It works just the same.

Falling Off the End of the DATA Planks

When the pointer reaches the last item in the last DATA statement in the program, there are no more items left to READ. If you try to READ again, you will see an error message:

```
OUT OF DATA ERROR IN XX
```

Erase line 91 and run again.



Back to Square One

At any point in the program, you have only two choices for the READ pointer.

1. You can do another READ: Then the pointer moves ahead one item.
2. You can RESTORE: Then the READ pointer is put back to the beginning of the first DATA statement in the program.

Mixtures of DATA

The DATA statement can hold strings or numbers in any order.

But you must be careful in your READ statement to have the correct kind of variable to match the kind of data.

Correct:

```
70 DATA 77, FUZZ
75 READ N
80 READ B$
```

Wrong:

```
70 DATA 77, FUZZ
75 READ B$
80 READ N
```



You can't put FUZZ in a number box.

Assignment 18:

1. Write a program naming your relatives. When you ask the computer UNCLE, it gives the names of all your uncles. DATA statements will have pairs of items. The first item is a relation like FATHER or COUSIN. The second item is a person's name. Of course, you may have several brothers, for example, each with a DATA statement.

Instructor Notes 19. Sound Effects

The 6581 Sound Interface Device (SID) chip in the Commodore 64 produces sound and music. This lesson introduces some features of the chip that are useful for making sound effects. Musical applications are covered in a later lesson. The SID chip is controlled by POKEing numbers into its registers. The chip is versatile but fussy to program.

Sound is sent to the TV speaker as part of the composite signal. If you use a color monitor instead, consult the Commodore 64 manual to see the pin-out for sending the sound signal to an amplifier and speaker.

The chip can produce up to three independent voices or sounds at the same time. The pitch of each voice is set by POKEing two bytes. This allows very precise intonation, a feature lacking in many other home computers. For example, you can experiment with the various temperaments of the musical scale.

There are four choices for the waveform of each voice, producing different timbres for the voices: flute, banjo, etc. One of the choices is *white noise*, which is very useful for producing sound effects (drums, gunfire, thunder, wind, waves).

The loudness of the total sound output is set by one POKE, and each voice can be switched into a programmable filter which modifies its harmonic content, further altering the timbre.

Each voice has a controllable pattern in loudness from start to end. Four numbers control the note shape: two for its start (the *attack* followed by a *release*) and one each for its loudness as it is held (the *sustain* value) and the rapidity with which it decays after turning it off (its *decay*). These ADSR parameters are fully described in the "Music" lesson.

The length of the note is controlled by a delay loop which you write into the program, rather than by the SID chip. That is, you must start the note with a POKE, then go into a timing loop, and finally stop the note with one or two more POKES.

Questions:

1. What things must you POKE to get a sound?
2. How do you turn off all the voices at once?
3. Why must you be careful of the box number that you POKE?
4. What number gives the loudest sound?

Lesson 19. Sound Effects

Turn up the sound on the TV now.

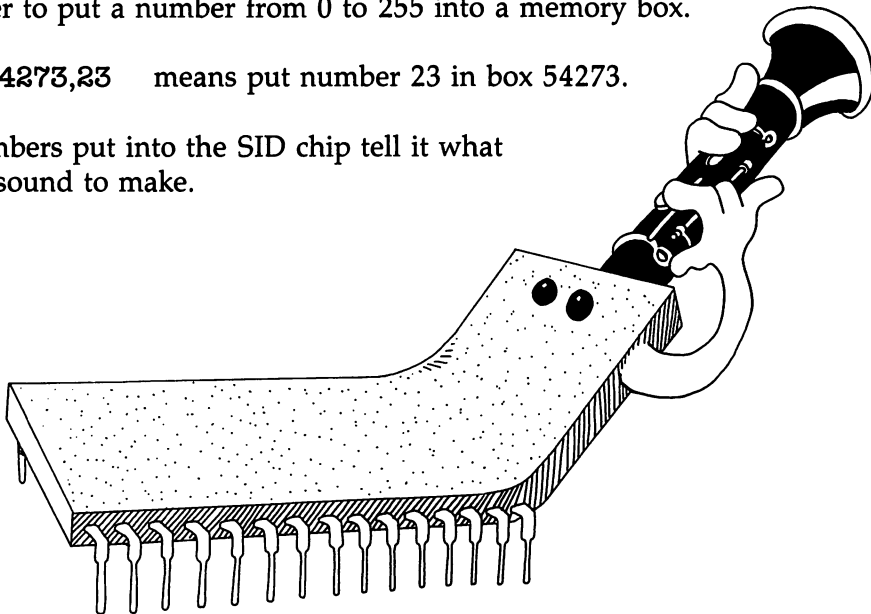
POKE SID a Few Times

No, not Sidney! SID is the chip in the computer that makes the sounds and musical notes.

There are 14 boxes in memory that are right inside the SID chip. POKE tells the computer to put a number from 0 to 255 into a memory box.

POKE 54273,23 means put number 23 in box 54273.

The numbers put into the SID chip tell it what kind of sound to make.



Johnny One Note

Enter this:

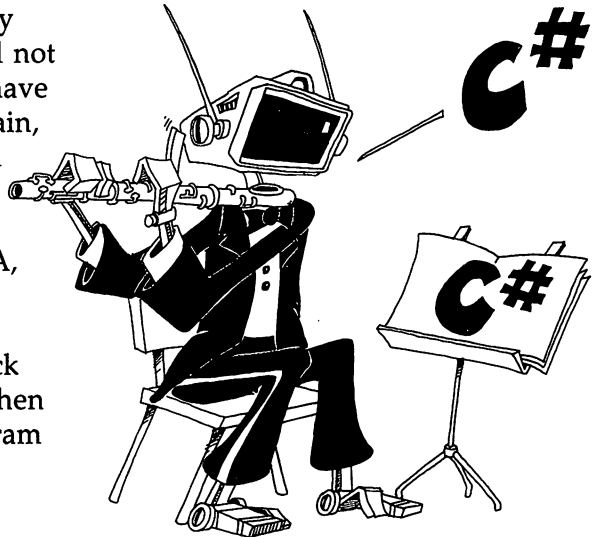
```
10 REM SID TUNES UP ON "CONCERT A"  
20 POKE 54273,23:POKE 54272,181  
22 POKE 54296,15:REM VOLUME 15  
24 POKE 54277,9:REM ATTACK-DECAY  
26 POKE 54278,0:REM SUSTAIN-RELEASE  
30 POKE 54276,33:REM TIMBRE  
85 FOR T=1 TO 1000:NEXT T  
90 POKE 54276,32:REM VOICE OFF  
91 POKE 54273,0:REM SOUND OFF
```

Save before running!

Careful! The numbers after the POKE must be exactly right. If not, you may mess up the computer's temporary memory when you run the program. This will not harm the computer, but you would have to turn the computer off, then on again, and you would lose the program you have in memory.

Run it. You should hear the middle A, the note used to tune orchestras.

If you do not hear a sound, first check that the TV sound is loud enough. Then check that *every* number in the program is correct.



But, Heavens, So Much Work for Only One Note?

Yes! It takes five POKES to set up the sound chip (lines 20, 22, 24, and 26). It takes one more POKE to turn on the sound (line 30) and two to turn it off (lines 90 and 91).

But notes are cheaper by the dozen. You can make more notes, even a whole song, with only two more POKES per note, and in fact, if you do it in a loop and use DATA, it is almost easy.

Let's explore the SID chip some more.

Tones by the Bushel

Enter:

```
10 REM ***** TONES *****
11 :
15 REM ----- CHIP BASE ADDRESS
16 C=54272
17 REM ----- VOICE 1
18 :
19 REM ----- REGISTER ADDRESSES
20 LO=C :REM --- HI FREQUENCY NO.
21 HI=C+1 :REM --- LO FREQUENCY NO.
24 TM=C+4 :REM --- TIMBER OF NOTE
25 AD=C+5 :REM --- ATTACK-DECAY
26 SR=C+6 :REM --- SUSTAIN-RELEASE
29 VL=C+24:REM --- LOUDNESS
30 REM ----- CLEAR THE CHIP
31 FOR L=C TO C+24:POKE L,0:NEXT
40 REM ----- ADSR SOUND, LOUDNESS
42 POKE AD,9
43 POKE SR,2*16+8
44 POKE VL,15+0
50 REM ----- PICK NOTE
52 PRINT"[CLR]{DOWN} ENTER NOTE:<2 TO 253>"
53 PRINT"[DOWN] (0 TO END PROGRAM {DOWN} "
54 INPUT N
56 IF N=0 THEN LIST
60 POKE HI,N:REM - POKE NOTE
61 POKE LO,0
65 POKE TM,33:REM ----- START NOTE
66 REM 33 IS A SAWTOOTH WAVEFORM
70 FOR T=1 TO 1000:NEXT
72 POKE TM,32:REM ----- END NOTE
73 POKE HI,0 :REM ----- TURN OFF SOUND
80 GOTO 40
```

Save to tape or disk before running.

Try these numbers: 2, 22, 222.

Then try these: 20, 40, 80, 160. They play octaves of the lowest note. Try other numbers.

SID has a block of 24 boxes starting at address 54272. Line 16 stores this address.

Lines 20 to 29 make and store the addresses of more of SID's boxes.

Line 31 puts a 0 in each of SID's boxes, which *turns it off* completely!

It takes two boxes, called HI and LO, to tell SID what tone to play. HI makes a big difference in the pitch, and LO is for fine-tuning. In this program we change only the number in HI. LO remains at 0.

Line 65 turns the note on, and also says that the note should be a *sawtooth* waveform. We will come back to this later.

Line 70 says the note should be about one second long.

Line 72 turns the note off. That is, it turns the sawtooth off.

But you would still hear a faint lasting sound unless you also turn the sound off. This is done in line 73 by putting a 0 in the HI box.



Piccolo, Piano, or What?

SID is a one-player band. You can get it to play different instruments by changing the numbers POKEd into the register called TM at address 54272+4 in lines 65 and 72 of TONES.

TM stands for *timbre*. Timbre is the quality of the sound that makes one instrument different from another instrument.

line 65	line 72	waveform
start	stop	
17	16	triangle
33	32	sawtooth
65	64	pulse
129	128	white noise

If you use pulse you also need to POKE a number from 0 to 15 into register 54275. The sound is thin and reedy for low numbers and more mellow for numbers near 15.

Describe the sound of each waveform here:

Sawtooth _____

Triangle _____

Noise _____

Pulse, 50 _____

Pulse, 9 _____

Sound Effects

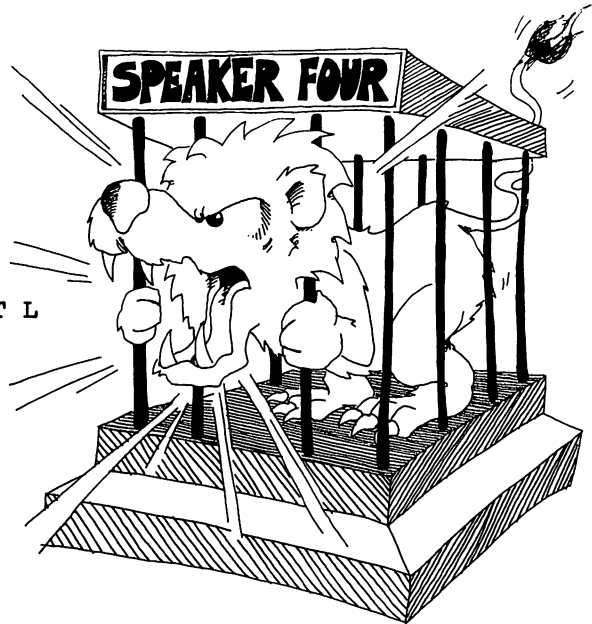
You can do many things to get different sound effects:

Different frequencies	(54270,54271)
Different waveforms	(54275,54276)
Different ADSR numbers	(54277,54278)
Different loudness	(54296)

You can use a loop to change the numbers in these boxes while the sound is going on!

Try this:

```
10 REM GOING UP
20 C=54272:LO=C:HI=C+1
22 AD=C+5 :SR=C+6
24 VL=C+24:TM=C+4
30 FOR L=C TO C+24:POKE L,0:NEXT L
32 POKE VL,15
34 POKE AD,9:POKE SR,2*16+8
35 POKE TM,33
40 FOR I=2 TO 222
42 POKE HI,I:REM PITCH CHANGES
49 NEXT I
50 POKE TM,32:POKE HI,0
```



Save to tape or disk before running.

Assignment 19:

1. Try to make these sound effects: laser gun, wind (use noise waveform), siren, explosion. Try POKEing into different boxes in the SID chip.
2. Write a program to show a flying bird, and make the bird chirp as it goes.

Instructor Notes 20. Names, Clocks, and Modes

This lesson treats the rules for naming variables, the *jiffy* clock, and the edit and RUN modes.

Descriptive variable names help clarify programs, but they take more typing and there are two hidden gotchas. One is that BASIC records only the first two letters of a name. This means that those beautiful descriptive names may not be unique. Even experienced programmers occasionally get caught here, and the bug may be quite hard to detect. One cure is to use only one- and two-character names.

The other gotcha is that reserved words may not be included in the name anywhere—start, middle, or end. The reserved words are the BASIC statements, commands, and functions and are listed in an appendix of this book. If a name contains a reserved word, the computer prints:

?SYNTAX ERROR IN XX

where XX is the line number containing the illegal name.

The jiffy clock is a register in the computer that contains a six-digit decimal number. The number is incremented every 1/60 second. The number is in a BASIC variable box named TI. A real clock (hours, minutes, and seconds) is described in a later lesson.

The student has been entering lines in the edit mode since lesson 1, and has been running programs. Now we explain a little about the difference and point out how the edit mode treats lines entered into the line buffer. The lesson on debugging makes extensive use of the edit mode options.

Questions:

1. Names longer than two characters may give you trouble. Why?
 2. Names that have numbers in them may be good names or wrong names. How can you tell if they are good?
-
-

-
-
3. Names cannot have punctuation marks in them, except in one case. What punctuation mark is allowed, where do you put it, and what does it tell you?
 4. Long names may have reserved words at the front, inside, or at the back. Which words are reserved? Where is there a list of reserved words?
 5. How do you ask the jiffy clock "what time is it?"
 6. How does the jiffy clock differ from ordinary clocks?

Lesson 20. Names, Clocks, and Modes

Old Rules

1. A string variable name ends in a dollar sign.
2. A numerical variable name doesn't.

More Rules

3. A variable name is made of letters and numbers.
4. A variable name must start with a letter.
5. A variable name cannot have any other symbols or punctuation marks. (Except that a string variable name must have a \$ at the end.)
6. A variable name can have as many letters and numbers as you want. But . . .
7. Only the first two characters of a variable name matter. All the rest are ignored (except the \$ at the end of a string variable name).
8. Reserved words cannot appear anywhere in a variable name. These words are the ones that are statements, commands, and functions: Some examples are REM, LIST, FOR, TO, LET, TAB, IF, and PRINT. There is a list of the reserved words in an appendix of this book.

Right and Wrong Names

Some Correct Names:	Some Wrong Names:
A77	2X
LEFTSIDE	X#
X3AB\$	\$NAME
APE	LIST
NAME\$	COLOR (has OR in it!)
COWBOY	GIFT\$ (has IF in it!)
X3	TOM (has TO in it!)

Do this: Put COLOR, GIFT, and TOM in a line like:

```
10 TOM = 1
```

and then enter RUN.

You will see: ?SYNTAX ERROR IN 10

Different Names That Are the Same

Breaking the rule that "Only the first two characters matter" can make your program run very strangely!

The computer can't tell the names in these pairs apart.

It thinks that	HOSE	is the same as	HOP
and	X1	is the same as	X10
and	NAME1\$	is the same as	NAME2\$

Try this program:

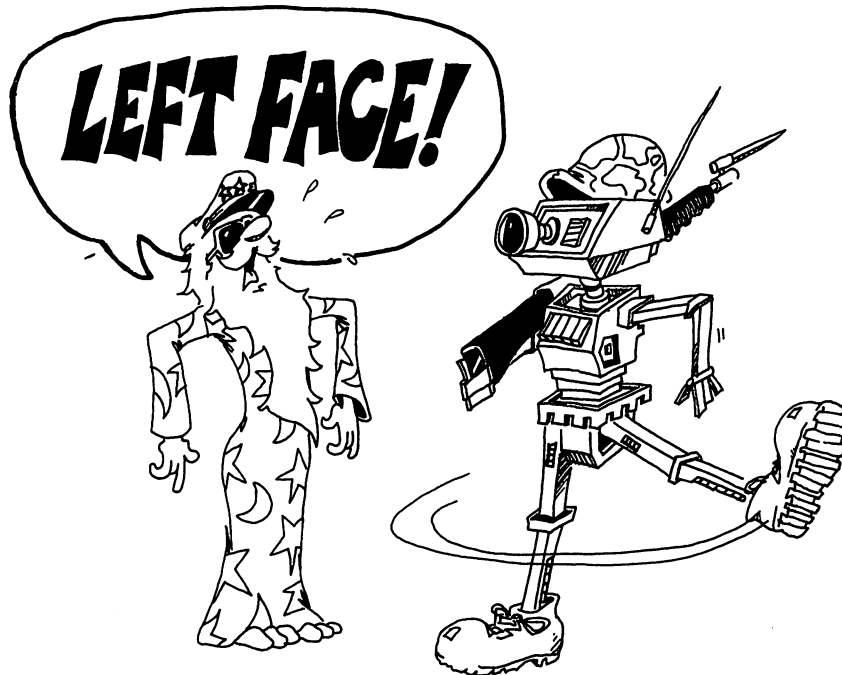
```
10 HOP$= "BUNNY"  
20 HOSE$= "SLINKY"  
30 PRINT HOP$
```

and see that HOSE\$ and HOP\$ name the same variable, which really has just HO\$ written on the front of its box.

Execution and Running

Execute a program means the same as *run* a program.

It is like a soldier executing the command "Left face!"



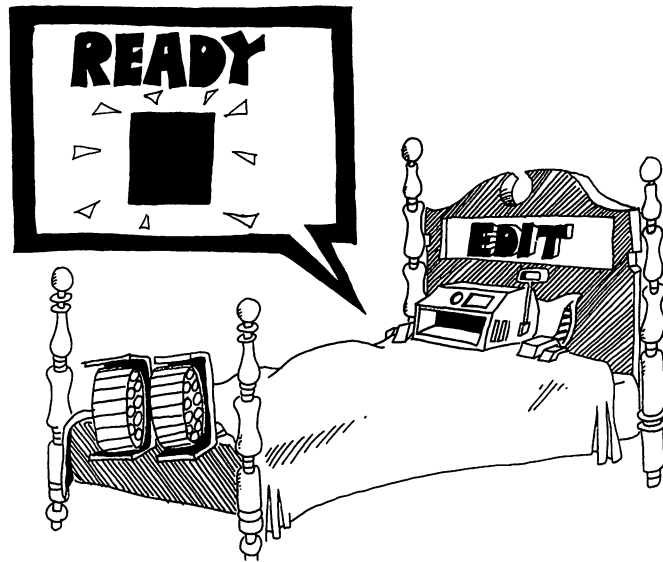
The Edit Mode and the RUN Mode

The computer is in the *edit mode* when you first turn it on. You know this because you see

READY.

and a flashing cursor.

In the edit mode, the computer is almost asleep, waiting for you to type something and press the RETURN key. The flashing cursor is like the computer snoring.



When you command `RUN`, the computer enters the *RUN mode*. It executes the program that is in its memory. Then it goes back to the edit mode.

The Line Buffer

When you are in the edit mode and enter a line, one of two things can happen:

Enter this:

```
99 G=G+1:PRINT G
```

The computer adds this line to the program in memory.

Enter this:

```
G=17*16:PRINT G
```

The computer prints the answer right away.

Anytime you type, the characters are stored in a special box in memory called a *line buffer*. When you press the RETURN key, the computer looks in the line buffer box.

Rule: If it sees a line number at the start of the line, the computer moves the line into the program in memory.

Rule: If it doesn't see a line number, it executes the line like a program. The line may have several statements separated by colons (:). After execution, it forgets the line!

The Computer's Jiffy Clock

The computer has a clock that starts when the computer is turned on and runs all the time. (Well, most of the time. It is turned off while the computer loads or saves a program.)

The clock reads in *jiffies* of 1/60 second each. The reading is kept in a special variable box with the name TI.

Run:

```
10 PRINT TI
20 GOTO 10
```

The clock can tell you how long something takes to do by subtracting before and after readings of the TI variable. Divide the jiffy clock reading by 60 to get the time in seconds.

Run:

```
10 T=TI
20 FOR I=1 TO 1000:NEXT I
30 S=TI-T
40 PRINT"THE LOOP TOOK ";S;"JIFFIES"
50 PRINT"THAT IS ";S/60;"SECONDS"
```

Assignment 20:

1. Read the naming rules numbered 1 through 8 at the beginning of this lesson. For each rule make up two names, one that is correct and one that disobeys the rule. (Rule number 6 has no wrong name.) Try each name in a line like `10 NAME=1` or `10 NAME$="A"` to see if it is a legal name.
2. Is SUPERCALIFRAGILISTICEXPIALIDOCIOUS a legal name? Why or why not? If not, fix it.
3. How can you tell if the computer is in the edit mode?
4. What does the computer do in the RUN mode?
5. What mode does the computer enter when the program has finished running?
6. Write a reaction time tester. Flash a question mark on the screen at a random time. The user presses the RETURN key as soon as the question mark appears. Measure the time delay by counting how much the jiffy clock has changed. Print out the time for the user to see.

Instructor Notes 21. Color Graphics

This lesson demonstrates most of the color features of the Commodore 64.

The next lesson finishes with the POKE statement that puts graphics characters on the screen in color.

The lesson starts with a procedure for systematically adjusting the color controls on the TV or monitor.

Screen border and background color are controlled by POKEs to memory addresses 53280 and 53281.

The color of characters currently being PRINTed by the computer is stored in memory box 646.

Great care must be used in POKEing because a wrong address may put junk in some vital part of memory, making the system crash. Of course, this doesn't hurt the computer physically; it just makes it necessary to do a *cold start* (that is, turn the computer off, then back on). As a result, the program in memory is lost.

If you are writing a program that has POKEs, it is wise to save it to tape or disk before running it. Then if you have a computer crash, you can cold start and load the program and be exactly at the point before the crash.

Questions:

1. How many different colors can the border have?
2. How many colors for the screen background?
3. How many colors for the printing?
4. What does memory box number 646 control?

-
-
5. What box do you POKE to change the screen background color?
 6. What box do you POKE to change the screen border color?
 7. What happens if you press the Commodore key and a color key?
 8. What danger is there in using POKE?

Lesson 21. Color Graphics

Adjust Your Color TV or Monitor

Put all the colors on the screen like this:

First press CTRL and RVS ON. (Now a space will be a colored spot.)

Press CTRL and the RED key. Then hold down the space bar to make a red stripe on the screen.

Repeat with the rest of the color keys. Press CTRL and RVS OFF.

Now you have six colored stripes on the screen. Adjust the color controls on the TV or monitor until each color looks correct.

Color Screen

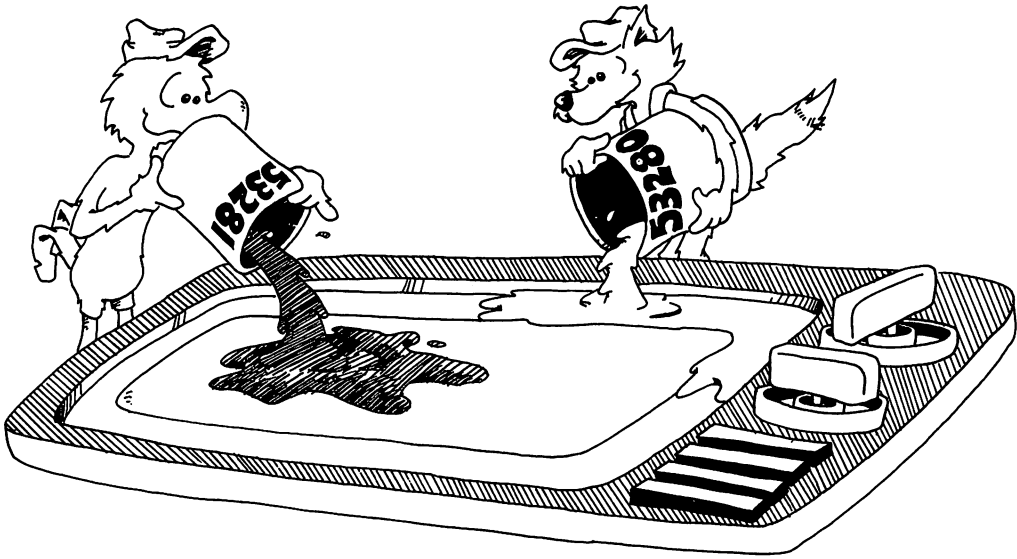
Remember that the box numbered 53281 holds a number from 0 to 15 which picks a number for the screen background color.

There is another memory box which tells the computer what colors to show on the border of the screen.

This number is 53280.

Run:

```
10 REM COMBINATIONS
20 SBOX=53281:REM SCREEN BOX
21 BBOX=53280:REM BORDER BOX
25 PRINT "{CLR}{3 DOWN}"
30 INPUT "WHAT COLOR SCREEN <0 TO 15>";SC
37 POKE SBOX,SC
38 PRINT "{CLR}{3 DOWN}"
40 INPUT "WHAT COLOR BORDER <0 TO 15>";BC
50 POKE BBOX,BC
90 GOTO 25
```



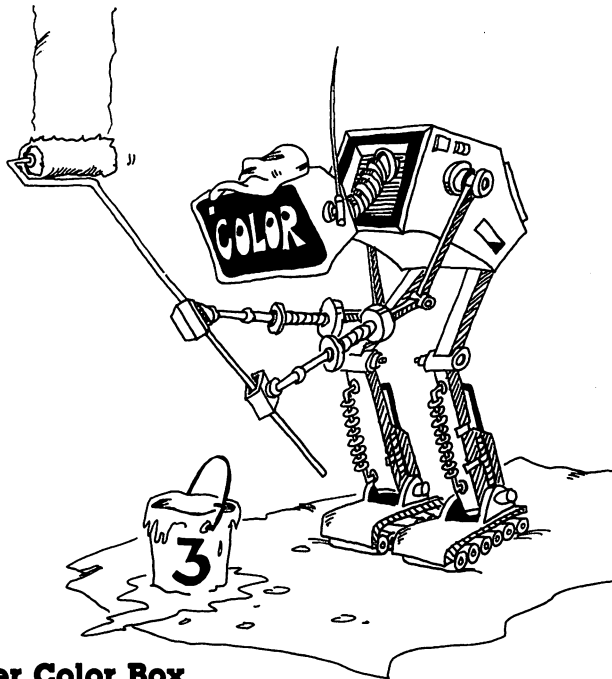
Color Numbers

0 Black	8 Orange
1 White	9 Brown
2 Red	10 Light Red
3 Cyan (Blue-Green)	11 Gray 1
4 Purple	12 Gray 2
5 Green	13 Light Green
6 Blue	14 Light Blue
7 Yellow	15 Gray 3

The screen can have any of these 16 colors.

(Note that these numbers are one less than the numbers shown on the color keys.)

You can change the printing color, even while the above program is running, by pressing the CTRL key and a color key. Try it!



The Character Color Box

There is a box that tells the computer what color you have chosen for printing characters. Its box number is 646.

You can change the color of the letters that are printed while the program is running by POKEing a number from 0 through 15 in that box.

Add these lines to the above program:

```
60 INPUT "WHAT COLOR LETTERS<0 TO 15>";CL
64 POKE 646,CL
```

Run it.

Careful! If you choose the *same* color for the screen and letters, you cannot see the printing. Don't give up. Just press CTRL and another color key, and the printing will appear again in some color or another!

Jumping Name

```
10 REM JUMPING NAME
15 PRINT "{CLR}{3 DOWN}"
16 POKE 53281,0:REM BLACK SCREEN
20 CB=646:REM CHAR. COLOR BOX
30 INPUT "NAME";N$
32 INPUT "HOW MANY LETTERS IN IT";N
39 PRINT "{CLR}"
40 FOR I=1 TO RND(8)*24:REM MOVE DOWN
45 PRINT "{DOWN}";:NEXT I
50 FOR I=1 TO RND(8)*39-N:REM MOVE ACROSS
55 PRINT "{RIGHT}";:NEXT I
57 NC=INT(RND(8)*8):REM RANDOM COLOR
58 IF NC=0 THEN 57:REM NOT BLACK
60 POKE CB,NC:REM PUT IN BOX
70 PRINT N$;:REM PRINT NAME
80 FOR T=1 TO 1000:NEXT T
90 GOTO 39
```

What happens:

Line 39 clears the screen and homes the cursor.

Lines 40 and 45 run the cursor down the screen a random number of lines.

Lines 50 and 55 run the cursor across the screen a random number of spaces.

Line 57 picks a color for printing letters.

Line 58 picks another color if the first choice was black. We do not want black letters on a black screen.

Line 60 puts the color choice in box 646. This tells the computer what color to print your name.

Line 70 prints your name in the new color.

Line 90 says, "Go do it over again."



Assignment 21:

1. Add lines to the "Jumping Name" program so that a new screen background color is chosen for each time through. Line 58 must compare the background color and the printing color. It must change the printing color if they are the same.
2. Write a program that uses two nested loops to go through all possible screen and border colors in order.
3. Add lines to the "Bird" program that ask the user what color bird to print. Then POKE the color into box number 646.
4. Write a program that draws your country's flag.

Instructor Notes 22. POKEing Graphics

The POKEing of characters and colors to the screen is explained.

The character is POKEd to one address and its color to an entirely different address. This complication may confuse your student.

The screen cells are addressed from 1024 continuously to 2023, and the color cells from 55296 to 56295. A row/column method of addressing is easier to understand, and a program that uses addressing in that form is included in the lesson.

The advantage of making graphics using POKE is that it may be clearer than the method using CRSR keys. It's also faster. The most important reason is that the PEEK instruction allows an easy way to see what character is in any given cell on the screen. This is useful in games to detect collisions. Lesson 28 will teach this procedure.

Questions:

1. The *home* spot on the screen has address 1024. Where on the screen is the character at address 1027 shown?
2. What address is the color box for this character?
3. What color numbers may be put in this box?

Lesson 22. POKEing Graphics

Make colored stripes and adjust the TV or monitor color knobs.

The Screen Is a Map of Memory Boxes

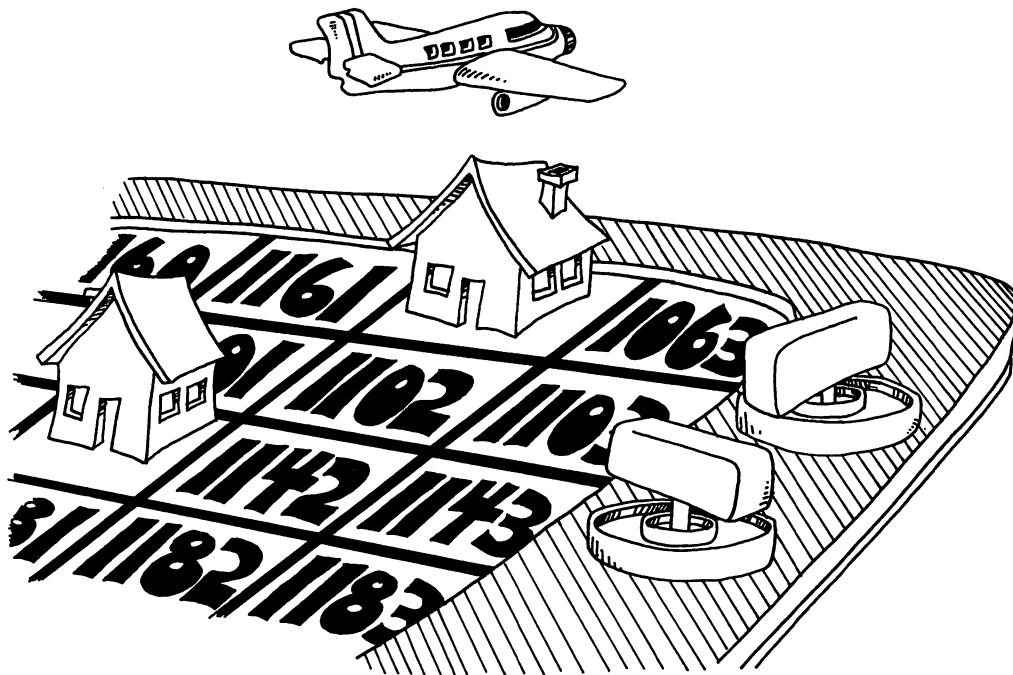
Imagine a little town with 25 streets. Each street has 40 houses on it.

Each house is a box in memory and can hold one character.

The streets are the 25 lines, counting down the screen. There are 40 boxes counting across a line.

The computer flies over the town like an airplane. It looks down on the boxes and draws a map, which it shows on the screen. The map shows which character is in each house.

Remember, most of the map is blank screen, but *blank*, or space, is a character, too!



Moving a Character into a House

Enter this:

```
POKE 1024,83:POKE 55296,2
```

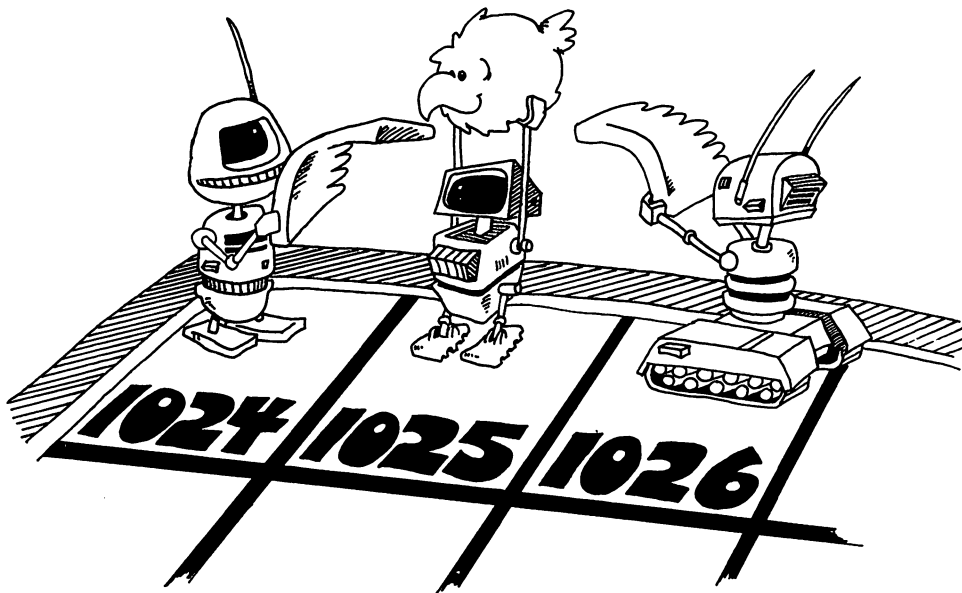
You will see a red heart in the *home* position on the screen (upper left corner). What happened? We POKEd the heart (character number 83) into the memory box number 1024 that shows in the home position on the screen.

POKE means you put the number directly into the memory box.

Then we POKEd a color, red, into another memory box (55296) that gives the color of the character in the home position.

Try this: Change the number 83 to any other number from 64 through 127 and run it again.

Change to another color number from 0 to 15.



Addresses on the Screen

Each house has an address.

The box we call the home of the cursor has the lowest address. It is 1024. The label on the front of the box is 1024.

The next box to the right is 1025, and the numbers increase across the screen and down.

The highest number belongs in the lower right corner, address 2023.

The Whole Town

Enter:

```
10 REM EVERY THIRD HOUSE
12 POKE 53281,0:REM BLACK SCREEN
15 PRINT "{CLR}{WHT}":REM CLEAR SCREEN
20 D$=" "
22 FOR I=1 TO 25:PRINT D$:NEXT I
25 FOR I=1024 TO 2023 STEP 3:REM WALK ALONG SCREEN
30 CH=INT(RND(8)*64)+64:REM RANDOM CHARACTERS
60 POKE I,CH:REM POKE CHARACTER
70 FOR T=1 TO 100:NEXT T:REM PAUSE
80 NEXT I:REM END OF LOOP
```

Save before running this program.

This time all the characters were white because the {WHT} character was printed in line 15 and spaces in line 22.

If you want another color, you have to POKE a number into the correct color map box.

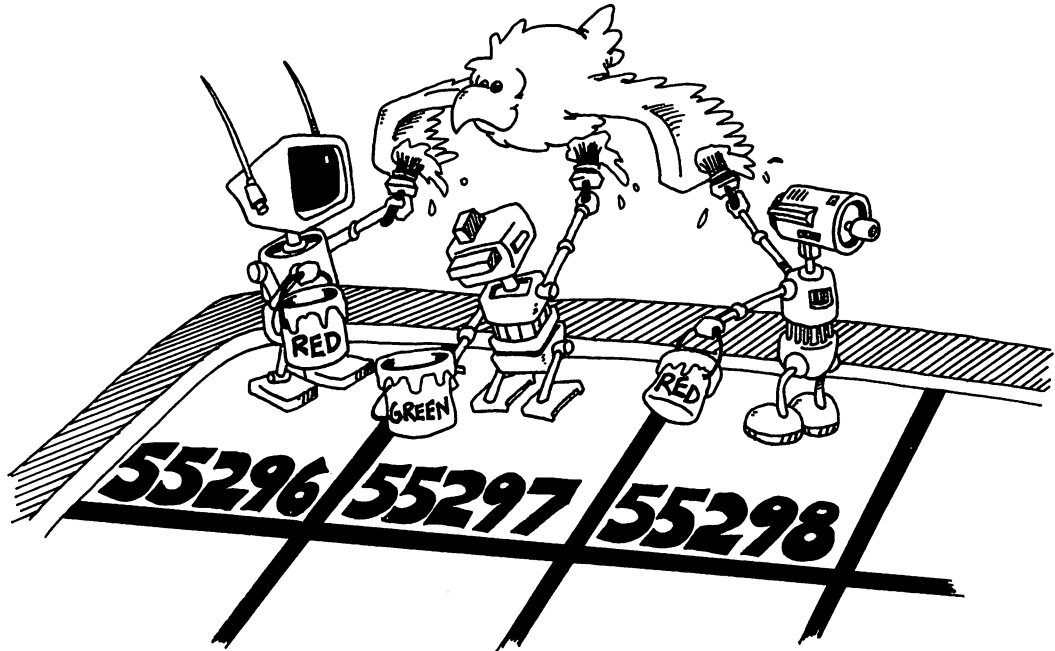
Change the above program.

Enter:

```
20
22
65 POKE I-1024+55296,RND(8)*7+1:REM COLOR BOXES
```

To get just letters and punctuation on the screen, change:

```
30 CH=INT(RND(8)*64)
```



Coordinate Numbers

Numbers like 1024 and 55296 are hard to remember.

We can number the streets and houses like those in real towns:

Street 0 at the top of the screen

Street 1 next below it

...

Street 24 at the bottom of the screen

We number the houses on each street:

House 0 on the left of each street

House 1 next to it

...

House 39 at the right end of the street

Then we use a little formula to get the address of each house and its color box.

Screen address = $1024 + 40 * \text{street number} + \text{house number}$.

Color address = $55296 + 40 * \text{street number} + \text{house number}$.

(See line 50 below.)

Enter:

```
10 REM TOWN MAP
15 SC= 1024:REM SCREEN CORNER
16 CC=55296:REM COLOR CORNER
17 B$="      "
20 PRINT "{CLR}"
30 INPUT "{HOME}{DOWN} WHICH STREET";S
35 PRINT "{HOME}{DOWN} ";B$;:REM ERASE WRITING
40 INPUT "{HOME}{DOWN} WHICH HOUSE";H
43 PRINT "{HOME}{DOWN} ";B$;
45 C=5:REM COLOR NUMBER 5
50 AD = 40*S + H:REM ADDRESS
55 POKE AD+SC,102:REM POKE SCREEN CELL
56 POKE AD+CC,C:REM POKE COLOR CELL
57 FOR T=1 TO 2000:NEXT T
80 GOTO 30
```

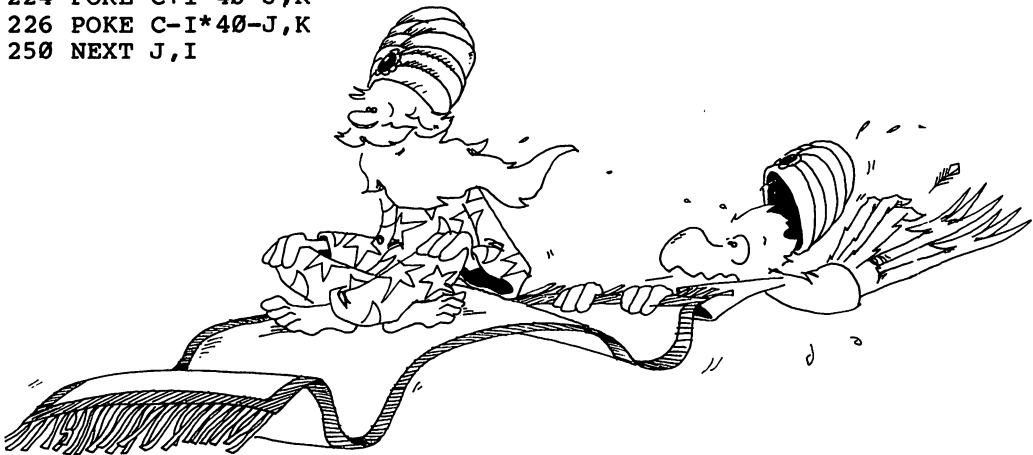
Save to tape or disk, then run.

(Omit the REM statements when you type this in.)

Assignment 22:

1. Write a program that draws a square. Let the user choose what color it will be. Save it to tape or disk.
2. Add to the program so that it has one to six spots on it like dice.
3. "Sinbad's Carpet" program is shown below. You improve it. Changing the formula in line 215 gives different carpets. Add a "super" outside loop that draws one carpet after another, each changed a little in color design. Do this by putting variables in line 215 that are changed in the super loop.

```
10 REM SINBAD'S CARPET
20 PRINT "{CLR}"
100 CC=55296
105 C=CC+500
110 SC=1025
115 CH=96+128
150 FOR I=SC TO SC+1000
155 POKE I,CH
160 NEXT I
210 FOR I=1 TO 11:FOR J=0 TO 18
212 R=(I+J)*7/22
215 K=K+(R+3)/(R+5)
216 IF K>15 THEN K=0
220 POKE C+I*40+J,K
222 POKE C-I*40+J,K
224 POKE C+I*40-J,K
226 POKE C-I*40-J,K
250 NEXT J,I
```



Instructor Notes 23. Secret Writing and the GET Statement

This lesson concerns the GET statement.

GET is a method of requesting a single character from the keyboard.

There is no screen display at all. No prompt or cursor is displayed, and the key-stroke is not echoed to the screen.

The utility of the GET statement lies just in this fact. For example, a requested word may be received with a series of GETs without displaying it to bystanders.

Another advantage of GET is that no RETURN key pressing is required. This makes GET useful in user-friendly programming for menus and in games for moves.

When the GET statement is executed, the computer looks in the line buffer for a character. If it finds one, it returns it to the variable. (If the variable is numerical and the keystroke was not, a ?SYNTAX ERROR is printed.)

If the line buffer is empty, it returns zero to a numerical variable or " " (the empty string) to a string variable.

For most situations where you want time to think before pressing a key, you must put GET in a loop, and keep looking at the buffer until a nonempty string is present.

The line buffer has a *queue* up to ten characters long. So for some cases, you can type ahead up to ten characters.

Questions:

1. Compare INPUT and GET. For each question reply GET or INPUT.
 - a. Gets whole words and sentences at once.
 - b. Shows a cursor.
 - c. Gets one character at a time.
 - d. Shows the keystrokes on the screen.
 - e. Does not need a RETURN keypress.
2. Why do you usually need to put GET in a short loop?

Lesson 23. Secret Writing and the GET Statement

The INPUT Statement

There are two ways to use INPUT.

Without a message:

```
10 INPUT A$
10 INPUT N
10 INPUT NAME$,AGE,DAY,MONTH$,YEAR
```

With a message:

```
10 INPUT "NAME,AGE";NAME$,AGE
10 INPUT "HOW ARE YOU";FEEL$
```

Either way, the computer waits for you to type in a word, sentence, or number, and then for you to press the RETURN key.

The GET Statement and Secret Writing

The GET statement is different from INPUT. It gets a single character from the keyboard. Example:

```
10 GET A$
```



When the program reaches GET in line 10, the computer looks to see if any key has been pressed. If so, it reads the key and puts the character into the memory box A\$.

Nothing shows on the screen:

- No message will show on the screen.
- No question mark will show.
- No cursor will show.
- What you type will not show.

The Computer Is Impatient

The computer does not wait. If you have not pressed a key by the time the computer reaches GET, it fills the variable A\$ with "" (the empty string) and goes on!

This is a bad feature if you want to think before you press any key. The computer does not wait for you to think.

Making the Computer Wait

You can ask the computer to keep looking at the keyboard until you press a key. Example:

```
35 GET L$:IF L$="" THEN GOTO 35
```

This is a good way to use GET in guessing games. You can enter the word or number to be guessed without the other player being able to see it.

Run this program:

```
10 REM-----GET-----
20 PRINT"{CLR}{3 DOWN}"
30 PRINT " PRESS ANY KEY"
40 PRINT "{2 DOWN}{BLK} WAITING..."
45 GET K$:IF K$="" THEN 45
47 FOR T=1 TO 500:NEXT T
50 PRINT "{DOWN}{WHT} THE KEY YOU PRESSED WAS ";K$
55 FOR T=1 TO 1000:NEXT T
60 GOTO 20
```

Making Words Out of Letters

The GET statement gets one letter at a time. To make words, glue the strings.

```
10 REM ## BACKWARDS ##
15 POKE 53281,0:REM BLACK SCREEN
20 PRINT "{CLR}{3 DOWN}"
30 PRINT " TYPE A WORD {DOWN}"
31 PRINT " END IT WITH A PERIOD"
35 W$="":REM WORD STRING IS EMPTY
40 GET L$:IF L$="" THEN 40
50 IF L$="." THEN 80:REM TEST FOR END
60 W$=L$+W$:REM ADD LETTER TO FRONT OF WORD
65 GOTO 40:REM TO GET ANOTHER LETTER
80 REM WORD IS FINISHED
82 PRINT"{CLR}{4 DOWN}{GRN} HERE IT IS BACKWARDS"
85 PRINT "{DOWN}{PUR} ";W$;"{WHT}"
```

How does the computer know when the word is all typed in? It sees a period at the end of the word. Line 50 tests for the period and ends the word when it finds the period.

The GET Statement for Numbers

The GET statement can be used to input numbers. If no key has been pressed, the number will be zero. Try this:

Enter and run:

```
10 GET N:IF N=0 THEN 10
15 PRINT N
20 GOTO 10
```

Press number keys. Now press a letter key. You see

?SYNTAX ERROR

when you try to GET a letter character in a numerical variable.

Assignment 23:

1. Write a program that has a menu for the user to choose from. The user makes a choice by typing a single letter. Use GET to get the letter. Example:

```
PRINT "WHICH COLOR?<R=RED ,B=BLUE ,G=GREEN>"
```

The program changes the border color to the user's choice.

2. Write a sentence-making game. Each sentence has a noun subject, a verb, and an object. The first player types a noun (like "The donkey"). The second player types a verb (like "sings"). The third player types another noun (like "the toothpick"). Use GET so no player can see the other players' words. You may expand the game by having adjectives before the nouns.



Instructor Notes 24. Pretty Programs, GOSUB, RETURN, END

This lesson covers subroutines. The END statement is also treated here because the program will usually have its subroutines at high line numbers and thus must END in the middle line numbers.

Like GOTO, GOSUB causes a jump to another line number. The only difference is that in GOSUB, control returns to the calling line after the subroutine has completed executing. This is accomplished by storing the GOSUB line number on a stack. When the computer encounters a RETURN statement, it pops the line number off the stack and returns to the statement following the GOSUB.

Subroutines are useful not only in long programs but in short ones where “chunking” the task into sections leads to clarity.

One of the hardest habits to form in some students (and even some professionals) is to impose structure on the program. Structuring has gone by many names, such as *structured programming* and *topdown programming*, and uses various techniques to discipline the programmer.

Call the student’s attention to ways that structuring can be done, and stress the advantages in clarity of thought and ease of programming that result. In this book, writing good REM statements and using modular construction in the program are the main techniques offered.

Questions:

1. What happens when the statement END is executed?
2. How is GOSUB different from GOTO?
3. What happens when RETURN is executed?
4. If RETURN is executed before GOSUB, what happens?

-
-
5. What does "call the subroutine" mean?
 6. How many END statements are you allowed to put in one program?
 7. Why do you want to have subroutines in your programs?

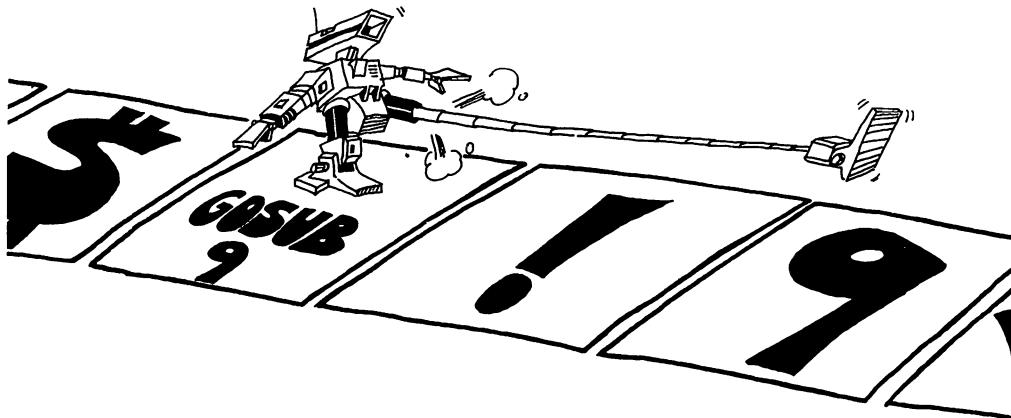
Lesson 24. Pretty Programs, GOSUB, RETURN, END

Run this program, then save it:

```
100 REM TAKE A TRIP
101 :
105 POKE 53281,0
110 PRINT "{CLR} HOP TO THE SUBROUTINE {DOWN}"
120 GOSUB 200
130 PRINT "{GRN} BACK FROM THE SUBROUTINE{DOWN}"
133 FOR T=1 TO 1000:NEXT T
135 PRINT "{BLU} HOP AGAIN{DOWN}"
140 GOSUB 200
150 PRINT "{CYN} HOME FOR GOOD {DOWN}{WHT}"
190 END
199 :
200 REM SUBROUTINE
201 :
210 PRINT "{RED} GOT HERE OK{DOWN}"
215 FOR T=1 TO 1000:NEXT T
220 PRINT "{YEL} PACK YOUR BAGS, BACK WE GO{DOWN}"
230 FOR T=1 TO 1000:NEXT T
290 RETURN
```

This is the skeleton of a long program. The main program starts at line 100 and ends at line 190.

Where there are PRINT statements, you may put in many more program lines.



The END statement in line 190 tells the computer that the program is over. The computer goes back to the edit mode.

Line 120 and line 140 call the subroutine. This means the computer goes and performs the statements in the subroutine, then comes back.

The GOSUB 200 statement is like a GOTO 200 statement except that the computer remembers where it came from so that it can go back there again.

The RETURN statement tells the computer to go back to the statement after the GOSUB.

Assignment 24A:

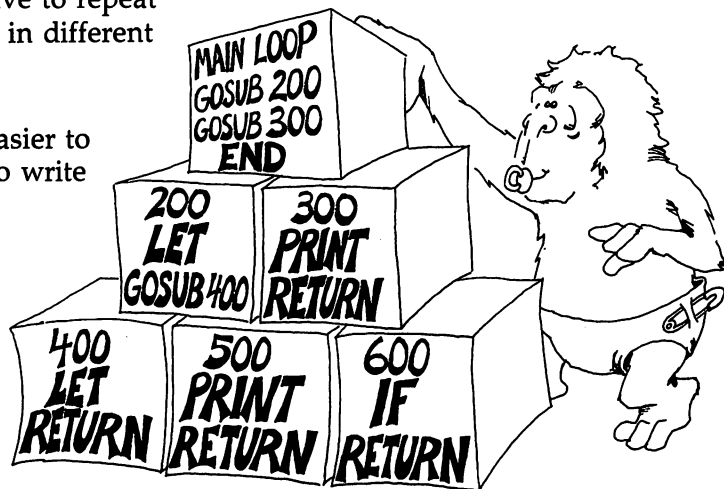
1. The delay loop is written three times in the above program. Add another subroutine with a delay loop in it, and GOSUB every time you need a delay.

What Good Is a Subroutine?

In a short program, not much good.

In a long program, it does two things:

1. It saves you work and saves space in memory. You do not have to repeat the same program lines in different parts of the program.
2. It makes the program easier to understand and faster to write and debug.



The END Statement

The program may have zero, one, or many END statements.

Rule: The END statement tells the computer to stop running and go back to the edit mode.

That is really all it does. You can put an END statement anywhere in the program—for example, after THEN in an IF statement.

Moving Pictures

```
10 REM -----JUMPING J
15 POKE 53281,0:PRINT "{RED}{CLR}"
20 X=10:Y=10:D=1
25 FOR J=1 TO 6:FOR I=1 TO 6
29 A$="{RVS} {OFF}"
30 GOSUB 100:REM          DRAW
34 A$=" ":REM           SPACE
35 GOSUB 100:REM          ERASE
40 Y=Y-D:REM            MOVE
50 NEXT I
55 D=-D:REM             REVERSE DIRECTION
60 NEXT J
80 PRINT"{WHT}"
90 END
100 REM
101 REM -----DRAW THE J
102 REM
104 PRINT"{HOME}"
105 FOR L=1 TO Y:PRINT:NEXT
109 FOR K=1 TO 5
110 PRINT TAB(X);" ";A$
111 NEXT K
120 PRINT TAB(X);A$;" ";A$
130 PRINT TAB(X);" ";A$;A$
190 RETURN
```

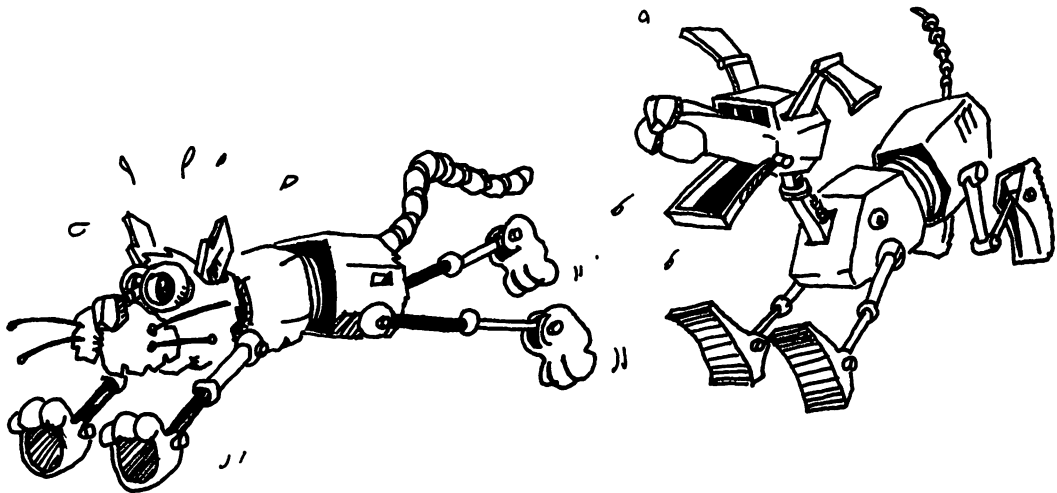
Save to tape or disk.

The picture is the letter *J*. The subroutine starting in line 100 draws the *J*. Before you GOSUB 100 you pick what character you want the *J* to be, by setting A\$. Look at line 29 and at line 34. If you pick A\$ to be " ", a single space, then the subroutine erases a *J* from that spot.

The subroutine draws the J with its upper left corner at the spot x,y on the screen. When you change x or y (or both), the J will be drawn in a different spot. Line 20 says that the first J will be drawn near the middle of the screen.

The variable D tells how far the J will move from one drawing to the next. Line 20 makes D equal to 1, but line 55 changes D to -1 after six pictures have been drawn.

Line 45 says that each picture will be drawn at the spot where y is larger than the last y by the amount D .



Assignment 24B:

1. Write a short program that uses subroutines. It doesn't have to do anything useful, just print some silly things. In it put three subroutines:

Call one of them twice from the main program.

Call one of them from another of the subroutines.

Call one of them from an IF statement.

-
-
2. Enter the "Jumping J" program and run it. Then make these changes:

Change the subroutine so that it prints your own initial.

Change the color of your initial to blue.

Change the jumping to sliding (to make the J move horizontally instead of vertically).

Change the starting point to the lower right corner instead of the middle of the screen.

Change the distance the slide goes to eight steps instead of six.

Change the size of each step from one to two.

Change the sliding so that it slides uphill. Use $X = X + D$; $Y = Y - D$.

Change the program so that the letter changes color from red (color 2) through all the colors to yellow (color 7) as it jumps.

3. Write a program that writes your three initials on the screen, each one a different color. Then make them jump up and down one at a time!

Instructor Notes 25. Logic: AND, OR, NOT

This lesson treats the AND, OR, and NOT relations and the numerical values for true and false.

There are two abstract ideas in this lesson that may give difficulty. One is that true and false have numerical values of -1 and 0 . Any expression that is of the form of an assertion (phrase A) has a numerical value of 0 or -1 . This number is treated just like any other number. It can be stored in a numerical variable, printed, or used in an expression. Most often it is used in an IF statement.

The other abstract idea compounds the confusion. The IF statement doesn't really look to see if phrase A is present. Rather, it looks for a numerical value between IF and THEN. Any number that is nonzero is treated as true. We call this a little white lie.

You can use the logical values in equations that at first glance look ridiculous. For example:

```
10 INPUT A
20 B = 5 - 7*(A<3)
30 PRINT B
```

The value of B will be 12 or 5 depending on whether A is less than 3 or not.

Questions:

1. For each IF statement, tell if anything will be printed:

```
10 IF 3=3           THEN PRINT "HI "  
10 IF NOT (3=3)    THEN PRINT "HI "  
10 IF 3=3 OR 0=2   THEN PRINT "HI "  
10 IF 3=3 AND 0=2  THEN PRINT "HI "  
10 IF "A"="B"      THEN PRINT "HI "  
10 IF NOT ("A"="B") THEN PRINT "HI "
```

2. What number will each of these lines print?

```
10 A=-1:           PRINT A,NOT A  
10 A=0:            PRINT A,NOT A  
10 A=-1:B=-1:     PRINT A AND B  
10 A=0:B=-1:      PRINT A AND B  
10 A=0:B=0:       PRINT A AND B  
10 A=0:B=-1:      PRINT A OR B  
10 A=0:B=0:       PRINT A OR B  
10 PRINT NOT 0
```

Lesson 25. Logic: AND, OR, NOT

Another Teenager Program

```
10 REM < AND, OR, NOT >
20 POKE 53281,0:PRINT "{CLR}{DOWN}"
30 INPUT " NAME ";N$
35 PRINT
40 INPUT " AGE";A
45 PRINT
50 IF (A>12 )AND (A<20) THEN PRINT " ";N$;" IS A T
    EENAGER.{DOWN}"
55 NFLAG=(A<13 OR A>19)
60 IF NFLAG THEN PRINT " ";N$;" IS NOT A TEENAGER!
    {DOWN}"
70 IF (NOT NFLAG) AND (A=16) THEN PRINT" AND ";N$;
    " IS {CYN} SWEET SIXTEEN!{WHT}"
```

Save to tape or disk before running.

What Does AND Mean?

Two things are true about teenagers: They are over 12 years old and they are less than 20 years old. Look at line 50.

IF (you are over 12) AND (you are less than 20) THEN (you are a teenager).

What Does OR Mean?

In line 55 the OR is used. Two things are said: "age is under 13" and "age is over 19."

Only one of them needs to be true for you to be "not a teenager."

IF (you are under 13) OR (you are over 20) THEN (you are not a teenager).

True and False Are Numbers

How does the computer do it? It says true and false are numbers.

Rule: True is the number -1 .

False is the number 0 .

(It is easy to remember that 0 is false because 0 is the grade you get if your homework is false.)

To see these numbers, enter this in the edit mode:

```
PRINT 3=7
```

The computer checks to see if 3 really does equal 7 . It doesn't, so it prints a 0 meaning false.

And this:

```
PRINT 3=3
```

The computer checks to see if $3 = 3$. It is, so the computer prints -1 meaning true.



Putting True and False in Boxes

The numbers for true and false are treated just like other numbers and can be stored in boxes with numerical variable names on the front. Run this:

```
10 N=(3=22)
20 PRINT N
```

The number 0 is stored in the box N because $3 = 22$ is false.

And this:

```
10 N= "B"="B"
20 PRINT N
```

The number -1 is stored in the box N because the two letters in the quotes are the same, so the statement "B"="B" is true.

The IF Statement Tells Little White Lies

The IF statement looks like this:

```
10 IF (phrase A) THEN (statement C)
```

Try these in the edit mode:

```
IF 0 THEN PRINT "TRUE"
IF -1 THEN PRINT "TRUE"
```

Now try this:

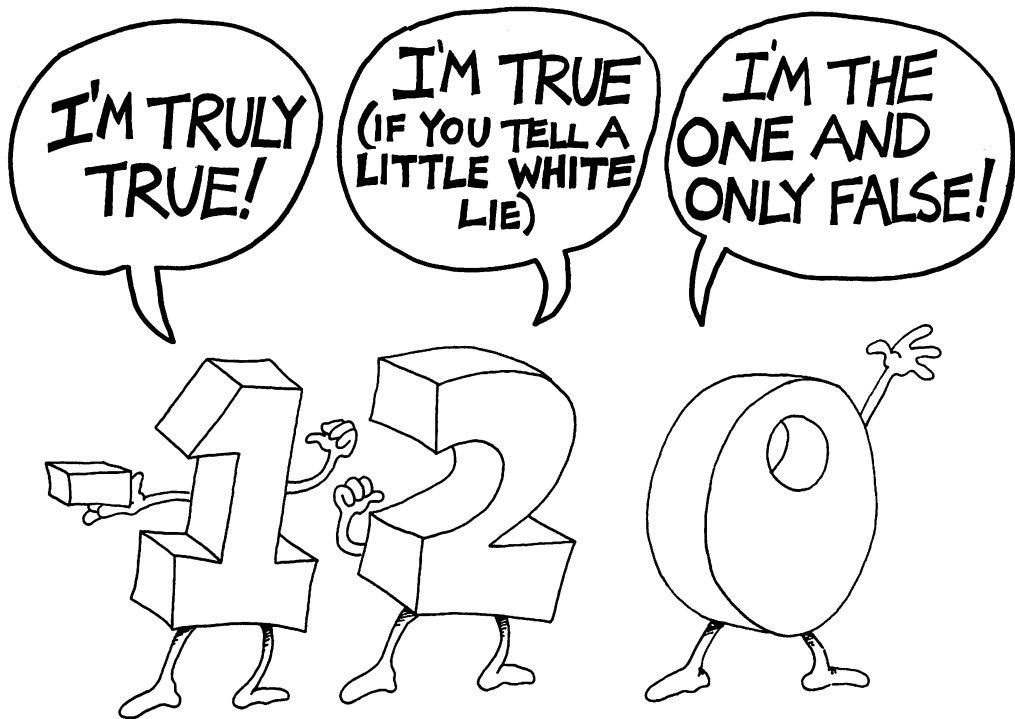
```
IF 22 THEN PRINT "TRUE"
```

Rule: In an IF, the computer looks at phrase A.

If it is zero, the computer says "phrase A is false," and skips what is after THEN.

If it is not zero, the computer says "phrase A is true," and obeys the statement after THEN.

The IF statement tells little white lies. True is supposed to be the number -1 , but the IF stretches the truth to say "true is anything that is not false," that is, any number that is not zero is true.

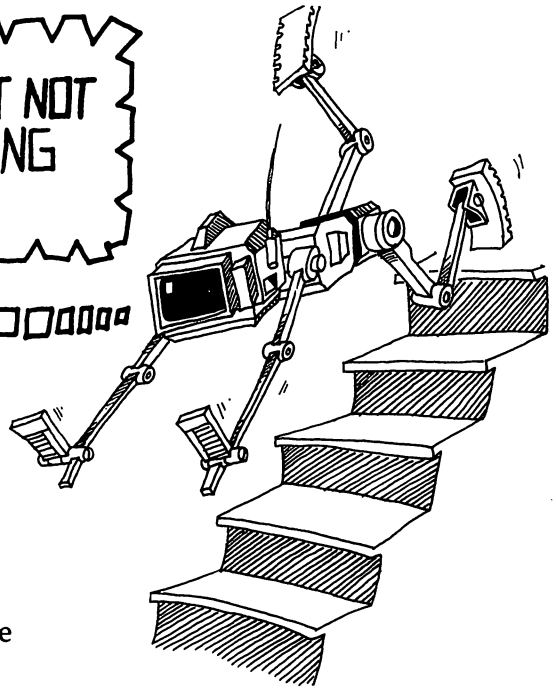
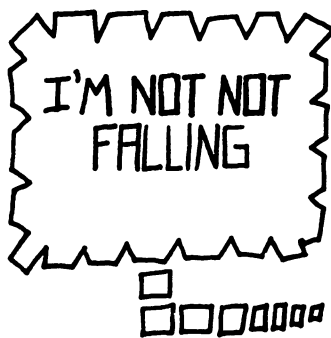


What Does NOT Mean?

NOT changes false to true and true to false. Try this:

```
10 REM DOUBLE NEGATIVE
20 N=-1
30 PRINT "N";TAB(10);N
40 PRINT "NOT N";TAB(10);NOT N
50 PRINT "NOT NOT N";TAB(10);NOT(NOT N)
60 REM THE COMPUTER KNOWS THAT "I DON'T HAVE NO.."
61 REM MEANS "I DO HAVE.."
```

Save to tape or disk.



Careful! NOT (TRUE) = FALSE
and NOT (FALSE) = TRUE

work only for real TRUE, the one where

True = -1.

It doesn't work for the little white lies where

True = any number except 0.

Try this. Put $N = 3$ in the above program and see that (NOT 3) doesn't give 0.

The Logical Signs

You can use these six symbols in a phrase A:

- = Equal
- <> Not equal
- < Less than
- > Greater than
- <= Less than or equal
- >= Greater than or equal

You have to press two keys to make the <> sign, the <= sign, and the >= sign.

The last two are new so look at this example to see the difference between < and <=:

2<=3 is TRUE	2<3 is TRUE
3<=3 is TRUE	3<3 is FALSE
4<=3 is FALSE	4<3 is FALSE

These two phrase A phrases mean the same:

2<=Q (2<Q) OR (2=Q)

Assignment 25:

1. Tell what will be found in the box N if:

N=4=4
N="G"<>"S"
N=5>7
N=3>2 AND 3<2
N=4=3 OR 4=4
N=NOT 0
N=5>=4

2. Tell if the word JELLYBEAN will be printed:

```
IF 0 THEN PRINT "JELLYBEAN"  
IF -1 THEN PRINT "JELLYBEAN"  
IF 9 THEN PRINT "JELLYBEAN"  
IF 3<>0 THEN PRINT "JELLYBEAN"  
IF 0 OR -1 THEN PRINT "JELLYBEAN"  
IF NOT -1 THEN PRINT "JELLYBEAN"  
IF "A"="Z" THEN PRINT "JELLYBEAN"  
IF NOT(0)AND 2 THEN PRINT "JELLYBEAN"  
IF NOT(0)OR -1 THEN PRINT "JELLYBEAN"  
IF 4=5 THEN PRINT "JELLYBEAN"
```

3. Write a program to detect a double negative in a sentence. Look for negative words like not, no, don't, won't, can't, nothing, and count them. If there are two such words, there is a double negative. Test the program on the sentence "COMPUTERS AIN'T GOT NO BRAINS."

Instructor Notes 26. Snipping Strings: LEFT\$, MID\$, RIGHT\$, LEN

In this lesson the functions:

LEFT\$ MID\$ RIGHT\$ LEN

are demonstrated. The use of MID\$() with three arguments is shown, but not with the third argument omitted.

These functions together with the concatenation operation (+) allow complete freedom to cut up strings and glue them back in any order.

As in earlier explanations, the main characteristics of the functions are shown, but not all the special cases, especially those that lead to error messages. It is better that extensive explanations not clutter up the text. If the student experiences difficulty, an experienced programmer or an adult consulting the Commodore manuals should clear up the problem.

Questions:

1. If you want to save the STAR from STARS AND STRIPES, what function will you use? What arguments?
2. If you want to save AND, what function and arguments?
3. If you want to count the number of characters in the string PQ\$, what function do you use? What argument?
4. What is wrong with each of these lines?

```
10 A$=LEFT$(4,D$)
10 RIGHT$(R$,I)
10 F$=MID$(A,3)
10 J$=LEFT(R$,YT)
```

-
-
5. What two arguments does the RIGHT\$() function need?
 6. What function will snip the third and fourth letters out of a word?
 7. Write a short program that takes the word COMPUTER and makes it into PUTERCOM.

Lesson 26. Snipping Strings: LEFT\$, MID\$, RIGHT\$, LEN

Gluing Strings

You already know how to glue strings together:

```
55 A$="CON" + "CAT" + "EN" + "ATION"  
60 PRINT A$
```

The real name for gluing is *concatenation*.

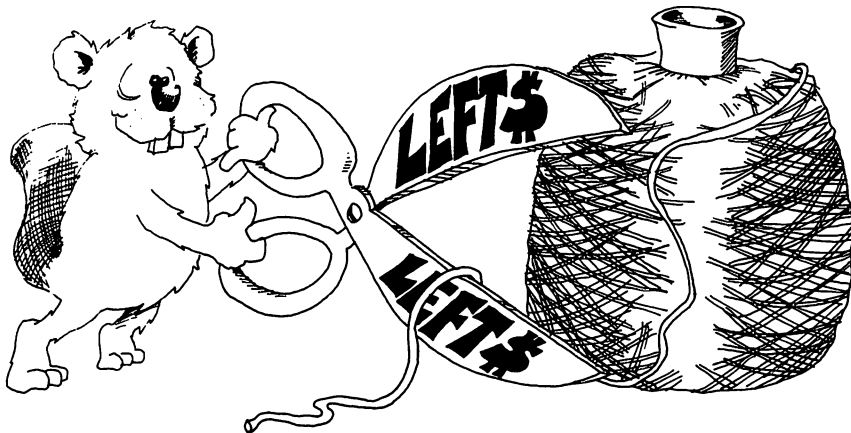
Concatenation means "make a chain." Maybe we should call them chains instead of strings.

Snipping Strings

Let's cut a piece off a string. Enter and run:

```
10 REM > SCISSORS >  
20 PRINT "{CLR}"  
30 N$="123456789"  
35 Q$=LEFT$(N$,4)  
40 PRINT N$,Q$
```

The LEFT\$ function snips off the left end of the string. The snipped-off piece can be put in a box or printed or whatever.



Rule: The LEFT\$() function needs two things inside the ().

1. The string you want to snip.
2. The number of characters you want to keep.

Try another. Change line 40 to:

```
40 PRINT RIGHT$(N$,3)
```

and run the program again. This time the computer prints:

```
789
```

RIGHT\$() is like LEFT\$() except the characters are saved off the right end of the string. (But the order of letters is still left to right.)

More Snipping and Gluing

Run:

```
10 REM ::: SCISSORS AND GLUE :::  
20 PRINT"{CLR}"  
30 N$="THE CAT'S MEOW"  
35 FOR I=1 TO 9  
40 L$=LEFT$(N$,I):R$=RIGHT$(N$,I)  
50 PRINT I;L$+R$  
60 NEXT I
```

The pieces of string you snip off can be glued back together in a different order. Add this line and run:

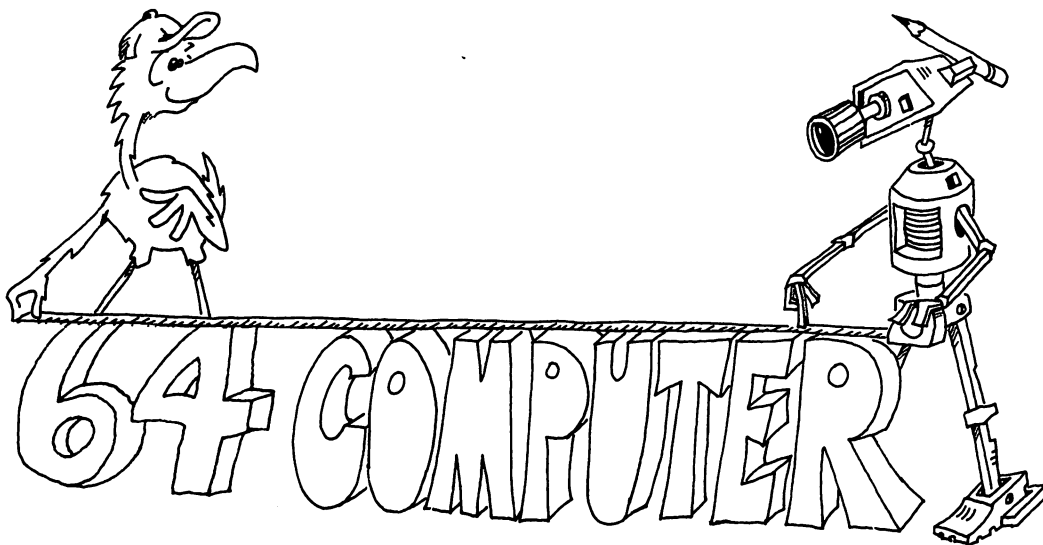
```
55 IF I=4 THEN PRINT:PRINT R$+L$:PRINT
```

How Long Is the String?

Run:

```
10 REM HOW LONG
20 PRINT "{CLR}"
30 INPUT "GIVE ME A STRING:";N$
35 PRINT "{CLR}"
40 L=LEN(N$)
50 PRINT "THE STRING:{DOWN}"
52 PRINT "'";N$;"' {DOWN}"
55 PRINT "IS";L;"CHARACTERS LONG"
```

The function LEN() tells the number of characters in the string. It counts everything in the string, even the spaces.



Cutting a Piece Out of the Middle

The MID\$() function cuts a piece out of the middle of the string.

Run:

```
10 REM ### MIDDLE ###
20 PRINT "{CLR}"
30 N$="WQXEMIDDUICXZ"
40 P$= MID$(N$,5,3)
45 PRINT P$
```

In line 40: P\$= MID\$(N\$,5,3) means:

Get the string from box N\$.

Count over five letters and start saving letters into box P\$.

Save three letters.

Look, Ma, No Spaces

Enter:

```
10 REM -----<NO SPACES>
20 PRINT "{CLR}{3 DOWN}"
21 PRINT " GIVE ME A LONG SENTENCE{DOWN}"
35 INPUT S$
40 L=LEN(S$)
45 T$=""
50 FOR I=1 TO L:REM RUN THROUGH THE WORDS
60 L$=MID$(S$,I,1):REM LOOK AT EACH LETTER
70 IF L$<>" " THEN T$=T$+L$:REM SAVE ONLY LETTERS
80 NEXT I
90 PRINT "{CLR}{2 DOWN} ";T$
```

Line 60 snips just one letter at a time out of the middle of the string.

A Real Clock

The Commodore 64 has a clock that is always running. The time is kept in a string named TI\$. There are six digits in the string. You have to split them apart to make the clock easy to read. You can reset it with a LET statement.

TI\$="123456" means 12 hours, 34 minutes, and 56 seconds.

Run

```
10000 REM ::: CLOCK :::
10020 TI$="120000":REM NOON
10025 PRINT "{CLR}"
10029 PRINT "{HOME}{2 DOWN}{2 RIGHT}";
10030 PRINT LEFT$(TI$,2);":";MID$(TI$,3,2);":";RIG
HT$(TI$,2)
10035 GOTO 10029
```

You set the clock with an INPUT TI\$. We used large line numbers so you can put these lines in programs that need a time-of-day clock. The clock runs off the *jiffy clock* and will stop during the time you do tape or disk LOAD and SAVE.

Assignment 26:

1. Write a secret cipher-making program. You give it a sentence and it finds the length. Then it switches the first letter with the second, the third with the fourth, and so on. Example:

THIS IS A TEST. becomes HTSII S AETTS.

2. Write a question-answering program. You give it a question, starting with a verb, and it reverses verb and noun to answer the question. Example:

ARE YOU A TURKEY? becomes YOU ARE A TURKEY.

3. Write a Pig Latin program. It asks for a word. Then it takes all the letters up to the first vowel and puts them on the back of the word, followed by AY. If the word starts with a vowel, it adds only LAY. Examples:

TURKEY becomes URKEYTAY
ADAM becomes ADAMLAY

4. Write a program that speaks Double Dutch. It asks for a sentence, then removes all the vowels and prints it out.
5. Write a program that uses GET to get a letter A to C to use in a menu. Change the letter to a number 1 to 3. Then use the ON-GOTO statement to pick which menu item to do.
6. Add to the clock program so the user can set it. You need an INPUT TI\$ statement. Don't forget to help the user with PRINTed instructions.

Instructor Notes 27. Switching Numbers with Strings

This lesson treats two functions, STR\$ and VAL. A general review of the concept of function is also given.

STR\$ takes a number and makes a string that represents it.

VAL does just the opposite, taking a string and making a numerical value from it. It accepts decimal and scientific notation (for example, 1.2 E + 31). If the first character is not a decimal digit, + or -, it returns the value 0. Otherwise, it scans the number, terminating at the first nonnumerical character (other than the E of the scientific notation).

This interconversion of the two main types of variables adds great flexibility to programs involving numbers.

Functions return a value to the expression they are in. You also say that functions are *called* just as you call a subroutine. The reason is, of course, that functions are implemented as subroutines on the machine code level.

Questions:

1. If your number marches too quickly in problem 2 of the assignment, how do you slow it down?
2. If your program has the string "IN 1732 GEORGE WASHINGTON WAS BORN", write a few lines to answer the question "How long ago was Washington born?" (You need to get the birth date from the string and convert it to a number.)
3. What is a *value*. What is meant by "a function returns a value." What are some of the things you can do with the value?
4. How do you convert the string "1999" to a number you can use in arithmetic?

-
-
5. What is an *argument* of a function? How many arguments does the MID\$() function have? How many for the CHR\$() function?
 6. Can you put a function at the start of a line?
 7. What will you see if your program has the line:

```
65 PRINT TI$
```

Lesson 27. Switching Numbers with Strings

This lesson explains two functions: VAL() and STR\$().

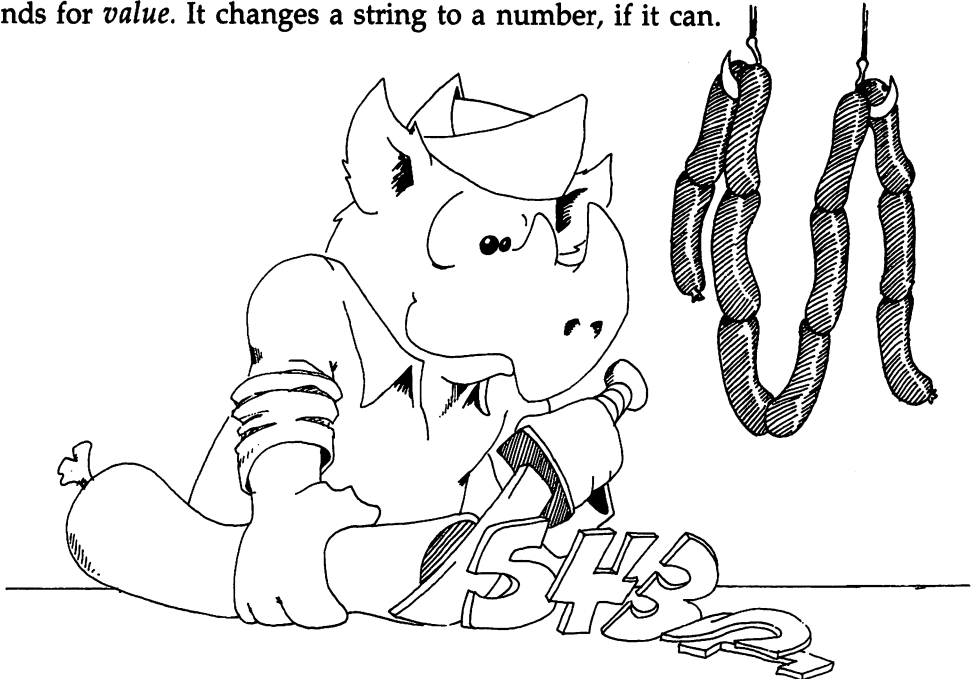
Making Strings into Numbers

We have two kinds of variables: strings and numbers. We can change one kind into the other.

Run:

```
10 REM STRINGS INTO NUMBERS
20 PRINT "{CLR}"
30 L$="123":M$="789"
50 L=VAL(L$):M=VAL(M$)
70 PRINT L
72 PRINT M
74 PRINT "----"
76 PRINT L+M
```

VAL stands for *value*. It changes a string to a number, if it can.



Making Numbers into Strings

Run:

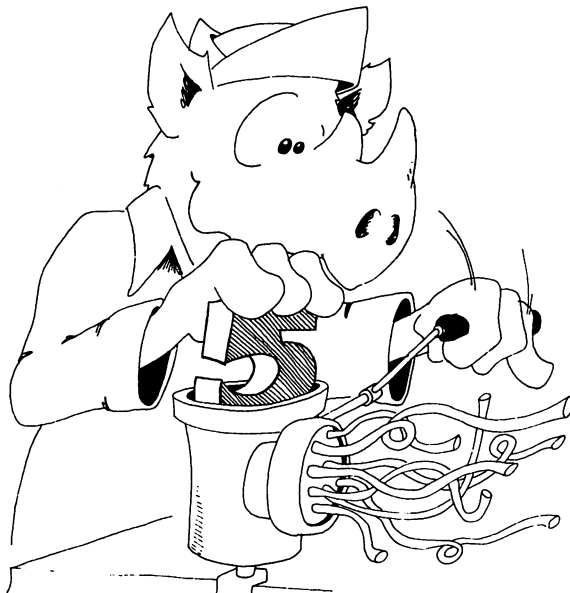
```
10 REM NUMBERS INTO STRINGS
20 PRINT "{CLR}"
21 POKE 53281,0
25 INPUT "GIVE ME A NUMBER";NB
30 N$=STR$(NB)
35 L=LEN(N$)
40 FOR I=L TO 1 STEP-1
45 B$=B$+MID$(N$,I,1)
50 NEXT I
60 PRINT "{CYN}{DOWN}HERE IT IS BACKWARDS{DOWN}"
65 PRINT "{DOWN}";B$
```

STR\$ stands for *string*. It changes a number into a string.

Functions Again

In this book we use these functions:

RND()	INT()
LEFT\$()	RIGHT\$()
VAL()	STR\$()
CHR\$()	PEEK()
MID\$()	LEN()
ASC()	



Rules about functions:

Functions always have () with one or more arguments in them. Example:

MID\$(D\$,5,J) has three arguments: D\$, 5, and J.

The arguments may be numbers or strings or some of each.

A function is not a statement. It cannot be the first reserved word on a line.

Correct: 10 LET D=LEN\$(CS\$)

Wrong: 10 LEN(CS\$)=5

A function acts just like a number or a string. We say the function returns a value. The value can be put in a box or printed just like any other number or string. The function may even be an argument in another function.

The arguments tell which value is returned.

(Remember, string values go in string variable boxes, numerical values go in numerical boxes.)

Practice with Functions

For each function in the list below:

Give the names of the arguments, and tell whether the argument value is a number or a string. Tell if the argument is a constant, variable, or function.

RND(8) fn _____
arg _____

RIGHT\$(U\$,Y) fn _____
arg _____

VAL("231") fn _____
arg _____

STR\$(INT(RND(8))) fn _____
arg _____

Each line below has errors. Explain what is wrong.

10 INT(Q)=65 _____

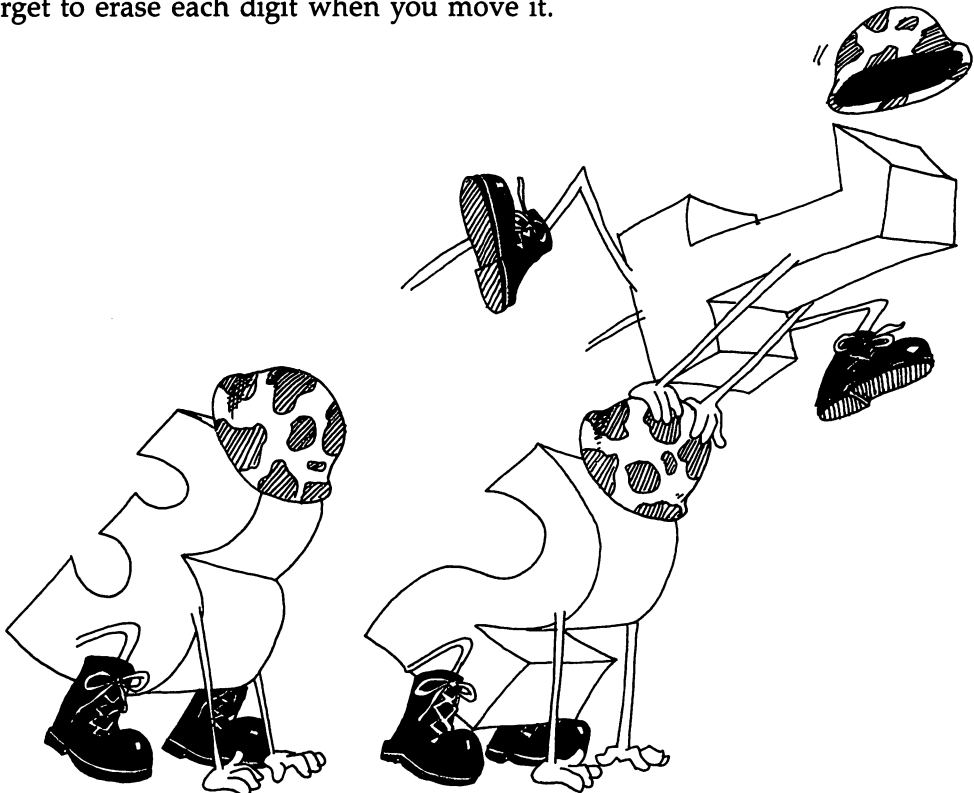
10 D\$=LEFT(R\$,1) _____

10 PW\$=VAL\$(F) _____

10 PRINT CHR\$ _____

Assignment 27:

1. Write a program that asks for a number. Then make another number that is backward from the first, and add them together. Print all three numbers like an addition problem (with a + sign and a line under the numbers).
2. Make a number play leapfrog slowly across the screen. That is, write it on the screen, then take its last digit and move it to the front. Keep repeating. Don't forget to erase each digit when you move it.



Instructor Notes 28. Action Games and the Function Keys

This long lesson introduces the function keys and the keyboard's box. It shows how to control two simultaneously moving objects on the screen, and how to use PEEK to detect collisions.

We develop the "Cupid" game in this lesson, one piece at a time. It serves as a prototype for many different games. This game uses a somewhat abstract method of moving the arrow shot by a diamond, while the diamond can still move and be controlled from the keyboard. The student may profit from help in understanding how the movements are achieved.

When drawing moving objects, you need to erase each old image before the next image is drawn.

A hit on a target is detected by using PEEK to test the square in front of the projectile. If there are several kinds of targets, it is better to test if the square is background. If not, then jump to a subroutine that asks the ASCII number of the target type so appropriate action can be taken.

Graphics games may grow to be rather long. BASIC is a little slow for such games. Maximum speed can be obtained if the working part of the program is first, and the initialization part is at the end, reached by a call from early in the program. This format is discussed in lesson 32.

For maximum speed, avoid repeatedly converting numerical constants to floating point as the program runs. That is why lines 55, 60, and 65 compare K\$ to variables F1, F3, and Z0 rather than to 4, 5, and 64. This practice is probably the most important single factor in producing fast programs.

The fastest and most versatile way to make graphics is to use sprites. This topic will be covered in a later section.

Questions:

1. What ASCII numbers do the function keys have?
2. How can you tell that a function key has been pressed?
3. How can you detect that a key is being held down?
4. What number box holds the key being pressed?
5. What number is in the box if no key is being pressed?
6. Explain how the computer knows to move the arrow and the diamond at the same time.

Lesson 28. Action Games and the Function Keys

The Function Keys

The four large keys on the right side of the Commodore 64 keyboard are called *function keys*.

They have these ASCII numbers:

f1	133	
f2	137	(SHIFTed f1)
f3	134	
f4	138	(SHIFTed f3)
f5	135	
f6	139	(SHIFTed f5)
f7	136	
f8	140	(SHIFTed f7)

How can you find out if a function key is being pressed?

You cannot use the INPUT statement. It ignores the function keys. You use the GET statement.

Enter:

```
10 REM FUNCTION KEYS
15 D$="{CLR}{5 DOWN} "
16 PRINT D$
20 F1$=CHR$(133)
30 GET K$:IF K$="" THEN 30
40 IF K$=F1$ THEN PRINT D$;"YOU PRESSED F1":GOTO 3
  0
50 PRINT D$;K$;
60 GOTO 30
```

(Remember the space before the quotation mark at the end of line 15.)

Run it. Press any key and see that it is printed to the screen. But when you press the f1 key, the special message gets printed. Notice that trying to PRINT character number 133 gives nothing. The eight characters that are function keys do not print.

Using the Function Keys in Programs

First you GET a one-character string from the keyboard. If it has ASCII number 133 through 140, it is one of the function keys. The short program above shows how to do this.

Make these changes in the above program:

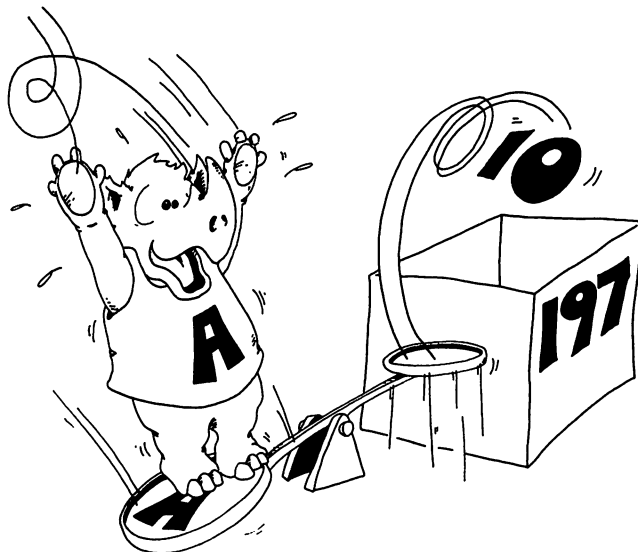
```
12 C=2
40 IF K$=F1$ THEN GOSUB 200
200 REM SCREEN COLOR CHANGE
210 POKE 53281,C
220 C=C+1:IF C=15 THEN C=2
299 RETURN
```

Now the f1 key sends you to a subroutine that changes screen colors.

The Trouble with GET

With GET, the computer can't tell if you just gave a key a quick press or are holding the key down.

Suppose you want to control a car on the screen by pushing the S (for speed) key. As long as the S key is down, the car goes. When you let up on the S key, the car stops. The GET statement is not good for doing this.



Looking in the Keyboard's Box

Here is how to tell if a key is being pressed.

There is a memory box that tells if a key is being pressed. Its address is 197.

The box holds the number 64 if no key is being pressed.

If a key is being pressed, it holds a number from 0 to 63 to tell which one. (If two keys are being pressed, it holds the higher number of the two.)

Try this:

```
10 REM KEYBOARD'S BOX
20 PRINT "{CLR}"
30 PRINT "PRESS ANY KEY"
40 N=PEEK(197)
50 PRINT N
60 GOTO 40
```

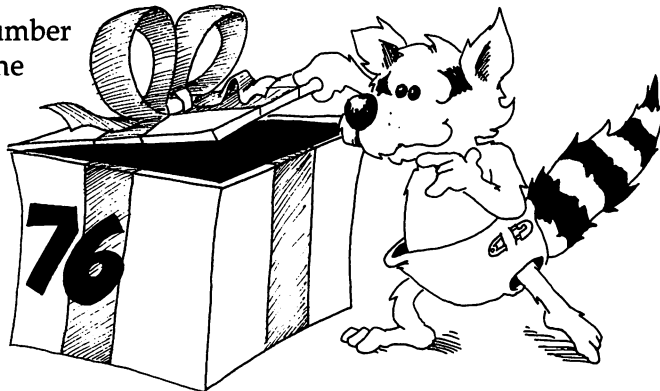
Notice that the number keeps printing as long as you hold the key down.

Try holding two keys down at once. The bigger number is the one that is in the box.

The only keys that do not put numbers in the box are CTRL, RUN/STOP, RESTORE, SHIFT, SHIFT/LOCK, and the Commodore Key.

Assignment 28A:

1. Make a table showing the number in box 197 for each key on the keyboard. This table will be useful when you design games. (Note: there is no key giving numbers 15, 52, 58, 60, 61, and 63.)



Moving a Diamond

Make the f1 and f3 keys control a moving diamond.

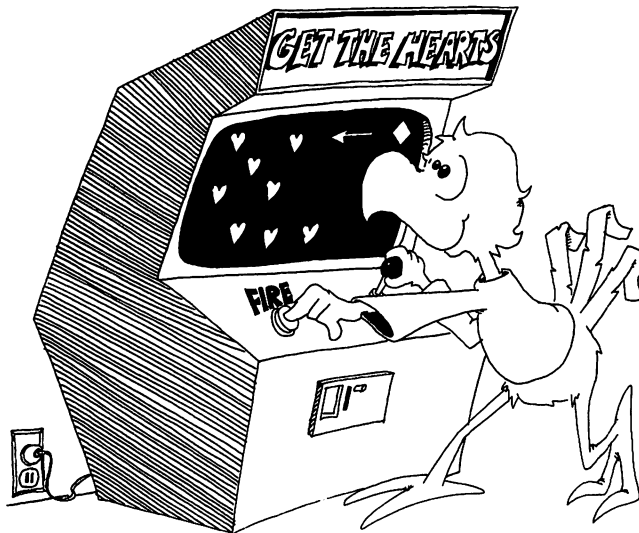
It moves up while you press the f1 key.

It moves down while you press the f3 key.

It stops if you don't press either key.

```
10 REM CUPID
11 PRINT "{CLR}"
12 POKE 53281,2
14 D$="{40 SPACES}"
15 FOR I=1 TO 25:PRINT D$:NEXT I
20 F1=4:F3=5:Z0=64
22 SB=1982
24 SC=1102
26 Y =SC
50 K =PEEK(197)
55 IF K=Z0 THEN D= 0
60 IF K=F1 THEN D=-40
65 IF K=F3 THEN D= 40
90 POKE Y,32
120 Y=Y+D
130 IF Y>SB THEN Y=SB
140 IF Y<SC THEN Y=SC
170 POKE Y,90
199 GOTO 50
```

Save before running.



Now run it. It puts a white diamond on a red screen. The diamond runs up and down the right side of the screen. It goes when you press f1 or f3. Otherwise it stops.

Line 50 looks in the keyboard's box.

Line 55 says, "*IF no keys are being pressed, THEN set D to 0.*" D is how much the diamond's screen address will change before the diamond is next put on the screen.

Line 60 tests to see if the f1 key is being pressed. If it is, D is set to -40 . This means the next diamond will be put one line higher than the present diamond.

Line 65 sets D to 40 if f3 is being pressed. It means the diamond will be POKEd one line lower next time.

Line 90 erases the old diamond by putting a space (character 32) on its screen spot.

Lines 130 and 140 do another bit of housekeeping. They check the value of y (the address of the diamond on the screen) to make sure that the diamond will not try to move off the top or bottom of the screen.

Line 170 POKEs the new diamond onto the screen, and the cycle starts again.

Shooting Arrows

We want the diamond to shoot arrows when we press the Q key.

Why pick Q? Two reasons:

First, Q is in a comfortable spot for the left hand to control shooting while the right hand controls motion.

Second, Q has number 58, higher than f1 or f3, so even if two keys are pressed at once, an arrow will shoot. (Be careful not to hold the Q key down, or else the f1 and f2 keys will not control the diamond.)

Add these lines:

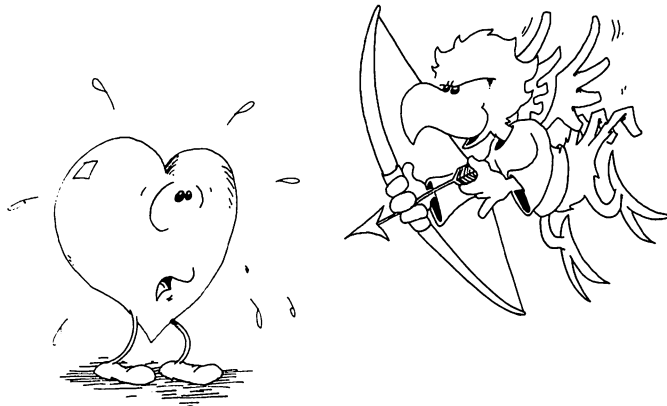
```
28 S=0:X=Y
70 IF K=62 AND S=0 THEN S=S+1
75 IF S<>0 THEN GOSUB 300
300 REM SHOOT
310 IF S=1 THEN X=Y-1:POKE X,31:S=2:RETURN
315 POKE X,32
316 IF S=39 THEN S=0:RETURN
320 X=X-1:S=S+1
330 POKE X,31
399 RETURN
```

The Right Way to Shoot

It's no good if the diamond has to stop moving while the arrow moves.

So when line 70 sees that the shooting key is pressed, it doesn't just jump to a subroutine that shoots the arrow across the screen. Instead, it sets a *flag*. The flag is the variable S. The flag is set when S<>0.

When the flag is set, the computer takes turns moving the arrow one space, then the diamond, then the arrow again, and so forth.



The move arrow subroutine has to recognize two things:

First, is this a new arrow, or just one that already started across?

Line 310 does this. S equals 1 only for a new arrow.

After each move of the arrow, S gets one bigger and x, the position of the arrow, gets one smaller, so the arrow goes to the right.

Second, the arrow must be stopped when it gets to the other side of the screen, and the diamond must be told that it can shoot another arrow.

Line 316 does this. When $S = 39$, the arrow has moved 39 spaces and is at the other side of the screen. S is set back to 0 meaning that the diamond is allowed to shoot again. (The arrow was erased in line 315, and since no new arrow has been written, the screen is empty of arrows.)

The Targets Are Hearts

Add these lines:

```
30 GOSUB 400
400 REM HEARTS
405 FOR L=1 TO 20
410 J=1024 + INT(RND(1)*23+1)*40
420 I=INT(RND(1)*36+1)
430 POKE I+J,83
450 NEXT L
499 RETURN
```



Save and then run it.

The arrows hit the hearts and erase them.

Popping the Targets

Add this line:

```
325 IF PEEK(X)=83 THEN S=0:POKE X,91:RETURN
```

PEEK to see if the arrow is about to hit a target.

PEEK(X) tells the number of the character at the spot the arrow is about to move onto. If the spot has number 83, it is a heart. Then pop it by POKEing across there. Set S equal to 0, because that arrow is all finished moving.

Assignment 28B:

1. You can change the "Cupid" program in many ways. Change the characters that are used. Change colors. Change the column which the diamond moves in.
2. Add a laser sound that starts when the arrow is shot and stops when the arrow ends its flight.
3. Why does the diamond dodge around while the arrow flies? It only makes sense if the hearts are shooting back at the the diamond! Add missiles that try to hit the diamond.

Instructor Notes 29. Music

This lesson continues the introduction of SID chip features with applications to making music.

Review the lesson on sound effects. DATA statements are used and may also need to be reviewed.

We examine the ADSR parameters (attack, decay, sustain, and release) in some detail. The ADSR parameters define a loudness envelope, or contour, for the duration of the note.

Each parameter is one nybble (four bits) in length, and so two parameters can share one SID chip register (one byte or eight bits). A and D share one register, S and R share another.

This is how it is done. A nybble is a number from 0 to 15. Memory is stored as bytes (a number from 0 to 255). So each byte holds exactly two nybbles and can be represented as $16 * H + L$ where H is the high nybble and L is the low nybble.

Three of the ADSR parameters are time values (A, D, R). The S, sustain, is a volume, or loudness, value. A table of time values for A, D, and R is given in this lesson. Note that the time is not proportional to the parameter value. A value of 5 gives an attack rise time of 0.056 seconds, but a value of 10 gives 0.5 seconds. It's possible to have times up to 24 seconds. Such long times are not useful in ordinary music but can be useful in creating sound effects.

We refer the reader to the *Commodore 64 User's Guide* for two of the most advanced features of SID: the programmable filter and the fact that voice 3 can be used to control some features of voices 1 and 2, for example, the frequency, for vibrato glissando.

SID can also accept audio from an external source (such as a tape recorder), process it with the filter, and then add it to the internal sound. The output can be re-recorded; you can build up a rich polyphonic piece in this way.

Questions:

1. What kind of sound will you get if you set both the attack and decay to the value 2, and sustain and release to 0?
2. How do you store the pitch and length of each note of music in the program?
3. How do you put the attack and decay numbers into the same POKE statement?
4. How many voices does SID have?
5. How would you change the program to use voice 2 instead of voice 1? Would it sound the same?

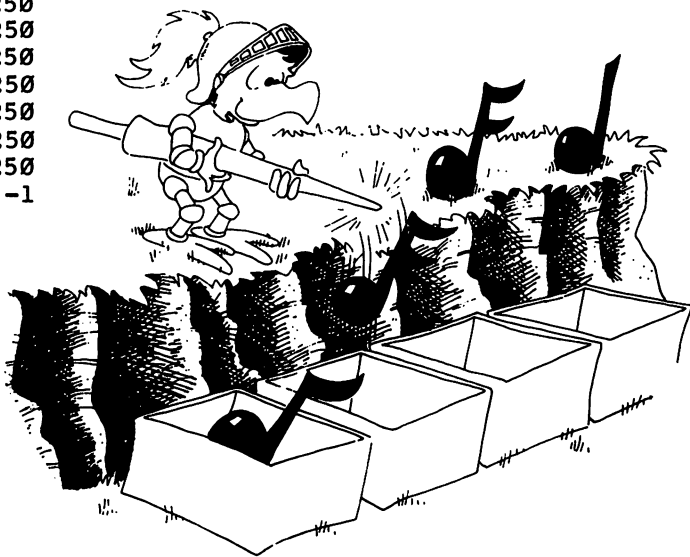
Lesson 29. Music

Go back and review the "Sound Effects" lesson (lesson 19). It tells a lot about the SID sound chip. Also review DATA in lesson 18.

SID Plays a Tune

```
10 REM ROW YOUR BOAT
12 C=54272:REM BASE ADDR
15 FOR I=C TO C+24:POKE C,0:NEXT
20 POKE C+5,9:POKE C+6,0:REM ADSR
30 POKE C+24,15:REM VOLUME
40 READ H,L,D:REM GET NOTE
50 IF H<0 THEN POKE C+24,0:END
60 POKE C+1,H:POKE C,L:REM PITCH
70 POKE C+4,33:REM START NOTE
80 FOR T=1 TO D:NEXT:REM HOLD NOTE
90 POKE C+4,32:REM END NOTE
95 GOTO 40
100 REM EACH LINE HOLDS TWO NOTES
105 REM NOTE: H, L, D
110 DATA 16,195,750,16,197,750
120 DATA 16,195,500,18,209,250
130 DATA 21, 31,750,21, 31,500
140 DATA 18,209,250,21, 31,500
150 DATA 22, 96,250,25, 30,1500
160 DATA 33,135,250,33,135,250
170 DATA 33,135,250,25, 30,250
180 DATA 25, 30,250,25, 30,250
190 DATA 21, 31,250,21, 31,250
200 DATA 21, 31,250,16,195,250
210 DATA 16,195,250,16,195,250
220 DATA 25, 30,500,22, 96,250
230 DATA 21, 31,500,18,209,250
240 DATA 16,195,1500,-1,-1, -1
```

Save the program before running.



The Scale

Here are two octaves of the scale. More are in the *User's Guide* that came with your computer.

		HI	LO
Middle	C	16	195
	C#	17	195
	D	18	209
	D#	19	239
	E	21	31
	F	22	96
	F#	23	181
	G	25	30
Concert	G#	26	156
	A	28	49
	A#	29	223
	B	31	165
	C	33	135
	C#	35	134
	D	37	162
	D#	39	223
	E	42	62
	F	44	193
	F#	47	107
	G	50	60
	G#	53	57
	A	56	99
	A#	59	190
	High	B	63
C		67	15

The Three Voices of SID

SID can play three instruments at once, each different from the others. You need to POKE information in the registers of each voice you use. We will not demonstrate the three voices here, but give you the register addresses:

Register Addresses

Base address is 54272.

Add these register addresses to the base address:

Register	Voice			Contents
	1	2	3	
LO	0	7	14	0 to 255
HI	1	8	15	0 to 255
Pulse low	2	9	16	0 to 255
Pulse high	3	10	17	0 to 15
Waveform	4	11	18	17,33,65,129
Attack-Decay	5	12	19	$16*(0 \text{ to } 15) + (0 \text{ to } 15)$
Sustain-Release	6	13	20	$16*(0 \text{ to } 15) + (0 \text{ to } 15)$
Volume (loudness)	24			0 to 15

Attack-Decay-Sustain-Release

Some registers are split into two uses.

Attack and decay share one register. Why don't they trample each other's feet? Well, attack stays in the attic, decay in the cellar (meaning, multiply the attack number by 16, then add on the decay number).

Likewise, sustain and release share another register.

Add:

```
16 INPUT "ATTACK-DECAY <EACH 0 TO 9>";A,D
17 INPUT "SUSTAIN <0 TO 15>";S
18 INPUT "RELEASE <0 TO 7>";R
20 POKE C+5,16*A+D:POKE C+6,16*S+R
80 FOR T=1 TO D*.5:NEXT T:REM ATTACK,DECAY,SUSTAIN
   TIME
92 FOR T=1 TO D*.3:NEXT T:REM RELEASE TIME
```

Save, then run.

Now you can play the same tune sound with different instruments.

Try these combinations:

A	D	S	R
2	2	0	0
9	9	0	0
9	9	15	1
2	2	3	7
0	0	15	0

Try other combinations. Change lines 70 and 90 to have other waveforms (see the "Sound Effects" lesson) and try them with different ADSR values.

As the Note Grows Older

When turned on, the note rises to full loudness. How long it takes to get there is controlled by the attack number. The number 0 means a click start while 8 gives a very gradual start. After getting to full loudness, the note decays to its sustain loudness. How long it takes to decay is controlled by the decay number. Of course, if the sustain number, S, is 15, the note does not decay at all!

The sustain number tells how loud (compared to the maximum loudness) the note will stay until turned off; 15 is loud, 0 is off.

When the note is turned off, it dies away to zero loudness in a time controlled by the release number. Line 92 in the program gives the note time to die away.

Without line 92, the next note would start before the first died away. But if you do not start another note (for example in a sound effect), you can do other computing while the sound continues to die away! A laser-shot sound can slowly die away while the computer moves graphics around.

Here are the times for attack, decay, and release:

Number	Attack	Decay and Release
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

The abbreviation *ms* means millisecond, or one thousandth of a second. So 500 ms is half a second.

Assignment 29:

1. Change the "Row Your Boat" program to play another tune.
2. Change the "Row Your Boat" program to let the user pick the waveform.
3. Write an electric organ program. Use the top row of the keyboard as the organ keys. Let the user pick stops of different kinds of sounds.

Instructor Notes 30. Arrays and the DIM Statement

This lesson introduces arrays and describes the DIM() statement. Up to 255 indices are allowed.

Arrays with one index are described first. The array itself is compared to a family, and the individual elements of the array to family members. The index value is the "first name" of the member.

Two-dimensional arrays can be compared to the numbers on a calendar month page or the rectangular array of cells on the TV screen.

Arrays themselves are not too difficult a concept. The trick is to see how they help in programming. There is a large variety of uses for arrays, and many do not seem to fall into recognizable categories.

You can use them to store lists of information. Connected lists also can occur. The telephone number program uses two-line arrays: one for names, the other for numbers. They are indexed the same, so a single index number can retrieve both the name and the number that goes with it.

Another general use of arrays is to store numbers which cannot neatly be obtained from an equation. An example would be the length in days of the 12 months.

Games often use arrays to store information about the playing board.

If you forget to DIMension an array before use, the BASIC interpreter gives it a DIMension of 10. If you try to use an element larger than that assigned to the array, the ?BAD SUBSCRIPT ERROR IN XX message is printed.

Questions:

1. What does the DIM BD(6) statement do?
2. Where do you put the DIM statement in the program?
3. What two kinds of array families are there?
4. What is the index, or subscript, of an array?

Lesson 30. Arrays and the DIM Statement

Meet the Array Family

```
22 F$(0)="DAD"  
24 F$(1)="MOM"  
26 F$(2)="KAREN"
```

Each member of the family is a variable. The F\$ family are string variables.

Here is a family of numeric variables:

```
35 N(0)=43  
37 N(1)=13  
39 N(2)=0  
41 N(3)=0
```

The family has a "last name" like A() or B\$(). Each member has a number in () for a "first name." The array always starts with the first name 0.

Instead of family we should say *array*.

Instead of first name we should say *index number, or subscript*.



The DIM() Statement Reserves Boxes

When the array family goes to a movie, they always reserve seats first. They use a DIM statement to do this.

The DIM statement tells the computer to reserve a row of boxes for the array. DIM stands for DIMension, which means size.

```
18 DIM A(3)
30 DIM B(7),B$(4)
```

Line 18 saves four memory boxes, one each for the variables A(0), A(1), A(2), and A(3). These boxes are for numbers and contain the number 0 to start with.

Line 30 reserves eight boxes for the B() array and five for the string array B\$(). The boxes named B\$(0) through B\$(4) are for strings and are empty to start with.

Rule: Put the DIM() statement early in the program, before the array is used in any other statement.

Making a List

Enter:

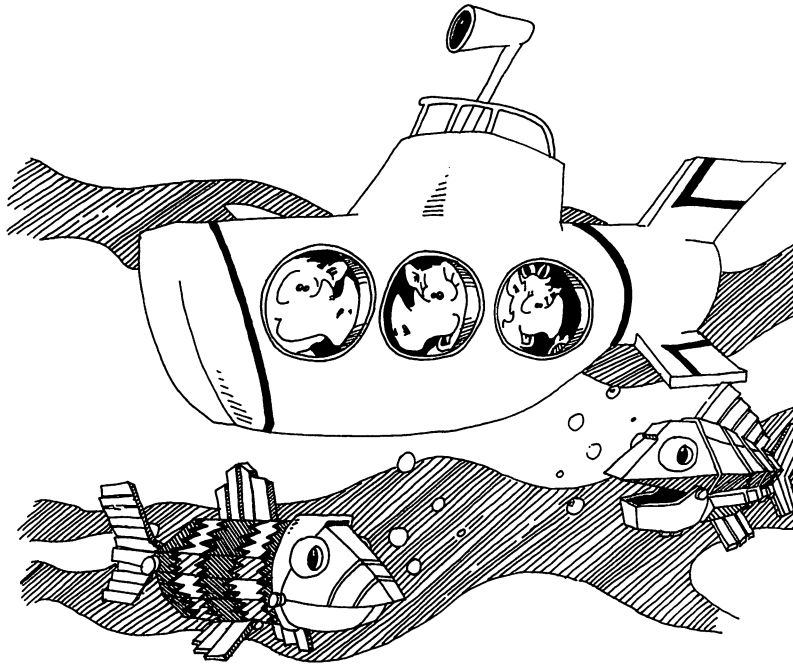
```
10 REM ++ IN A ROW ++
30 DIM A$(4)
35 PRINT "{CLR}{2 DOWN} ENTER A WORD {DOWN}"
40 FOR N=0 TO 4
45 IF N>0 THEN PRINT "ANOTHER {DOWN}"
50 INPUT A$(N)
55 PRINT
60 NEXT N
100 PRINT "IN A ROW {2 DOWN}"
110 FOR I=0 TO 4
120 PRINT A$(I);TAB(10);I
130 NEXT I
```

Save and run.

You can use a member of the array by itself. Look at this line:

```
40 B$(2)="YELLOW SUBMARINE"
```

Or you can use the array in a loop. Lines 50 and 120 in the program "In a Row" are in loops where the index (N or I) keeps changing.



Making Two Lists

Enter:

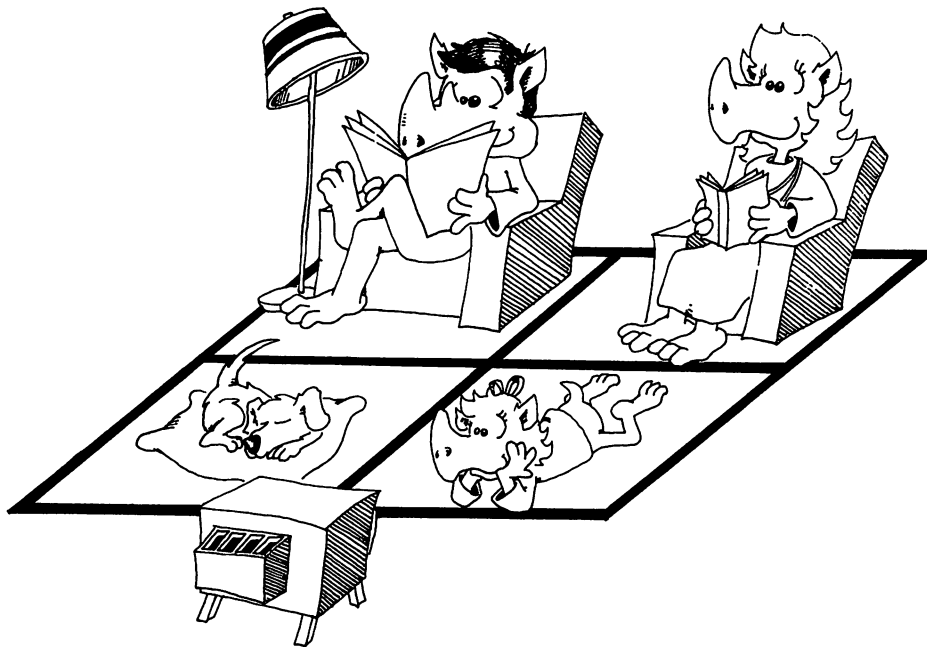
```
10 REM PHONE LIST
30 DIM NAME$(20),NUMBER$(20)
35 I=0
40 PRINT "{CLR}{DOWN}ENTER DATA{DOWN}"
50 INPUT{SHIFT-SPACE}"NAME";N$
55 IF N$="END" THEN END
56 NA$(I)=N$
60 INPUT "NUMBER";NU$(I)
65 PRINT
70 I=I+1:GOTO 50
```

Run. Press the RUN/STOP key to stop the program. Save to tape or disk.

One Dimension, Two Dimension, . . .

The arrays that have one index are called one-dimensional arrays.

But arrays can have two or more indices. Two-dimensional arrays have their family members put in a rectangle, like the days in a month on a calendar.



Eight Queens

The "Eight Queens" puzzle asks you to put eight chess queens on the chessboard in such a way that no queen is attacked by any other. If you are not familiar with chess, look up the moves of the queen in an encyclopedia. Obviously, you can't have two queens on the same row or column. Queens attack along the diagonal also. There are 92 patterns of queens on the board that solve this puzzle.

```
1 REM ### EIGHT QUEENS ###
2 GOTO 1000
100 REM MAIN LOOP
115 M=R(I)
120 M=M+1:IF M=9 THEN GOSUB 600:GOTO 115
130 IF B(I,M)=0 THEN 700
140 GOTO 120
200 REM
```

```

201 REM UPDATE ATTACKED SQUARES
202 REM
210 FOR L=1 TO 8
215 B(I,L)=B(I,L)+D
220 B(L,J)=B(L,J)+D
225 NEXT L
500 REM
501 REM DIAGONAL
502 REM
510 FOR K=1 TO 8
515 X=I+K
520 IF X>8 THEN 530
522 Y=J+K:IF Y>8 THEN 525
523 B(X,Y)=B(X,Y)+D
525 Y=J-K:IF Y<1 THEN 530
526 B(X,Y)=B(X,Y)+D
530 X=I-K
535 IF X<1 THEN 590
540 Y=J+K:IF Y>8 THEN 550
545 B(X,Y)=B(X,Y)+D
550 Y=J-K:IF Y<1 THEN 590
555 B(X,Y)=B(X,Y)+D
590 NEXT K
595 B(I,J)=Q
599 RETURN
600 REM
601 REM GO BACK
602 REM
610 R(I)=0
612 I=QN
615 IF I=0 THEN END
620 J=R(I)
630 D=-1:Q=0:GOSUB 200
640 QN=QN-1
690 REM GOSUB 900
699 RETURN
700 REM
701 REM GO AHEAD
702 REM
710 R(I)=M:J=M
715 QN=QN+1
720 D=1:Q=-1
730 GOSUB 200
735 NM=NM+1
740 IF QN=8 THEN GOTO 800
780 I=I+1
785 GET KB$
786 IF KB$=" " THEN GOSUB 900
799 GOTO 115
800 REM

```

```

801 REM SOLUTION
802 REM
810 NA=NA+1
814 NS=NS+1
815 GOSUB 900
860 GOSUB 612
899 GOTO 115
900 REM
901 REM DISPLAY
902 REM
905 PRINT "{CLR}"
910 FOR X=1 TO 8
915 FOR Y=1 TO 8
919 PRINT LEFT$(D$,5+2*X);
920 PRINT TAB(2+2*Y);
925 BB=B(X,Y)
930 IF BB=-1 THEN PRINT " O";
940 IF BB> 0 THEN PRINT " .";
950 IF BB= 0 THEN PRINT " .";
955 IF BB<-1 THEN PRINT X,Y;:END
960 NEXT Y:NEXT X
980 PRINT "{HOME}{DOWN} SOLUTIONS";NS;"MOVES";NM
999 RETURN
1000 DIM R(8),B(8,8)
1010 PRINT "{CLR}"
1020 I=1:QN=0:NS=0
1030 D$="{HOME}{25 DOWN}"
1040 PRINT"{CLR}{3 DOWN}PRESS THE SPACE BAR TO SEE
      THE BOARD"
1050 FOR T=1 TO 2000:NEXT
1999 GOTO 100

```

Assignment 30:

1. Write a program that stores the number of days in each month in an array. Then when you ask the user to enter a number from 1 to 12, it prints out the number of days in that month.
2. Write a program so that the computer plays the card game War against the user. Have an array hold the 52 cards in the deck. Deal them at random into two arrays, one for each player. In each turn, each player pulls the cards from his or her deck in order. If the cards played by user and computer don't match, they are both put in the booty pile. If they do match, there will be a battle. In the battle, both opponents play the next card on their respective piles. The high card wins the whole booty pile.

-
-
3. We wrote Eight Queens for a standard 8×8 chessboard. Change the program so that the user can choose any size board.
 4. Change Eight Queens into "Super Queens." Each moves like a queen and like a knight. Are there any solutions?



Instructor Notes 31. Sprites

The VIC chip in the Commodore 64 controls the video displays including eight sprites (powerful bitmapped graphics objects, 24 bits wide and 21 bits high).

This lesson gives the student a program in which the sprite that is drawn on the screen is POKEd into memory without the drudgery of determining bit patterns and changing them to decimal numbers.

The sprite can be colored with one POKE and moved to another spot on the screen with one to three more POKEs. Each sprite is told which picture to display by a *pointer* updated with a single POKE. Many pictures can be stored ahead of time and rapidly displayed one after another by the same sprite. Conversely, several sprites can display the same picture in different colors and different spots on the screen at the same time.

Those portions of a sprite in which the bit is 0 are transparent, showing whatever objects (other sprites or screen graphics) are behind them. The frontmost sprite of an overlapping pair is the one with the lowest sprite number. For each sprite, the programmer can choose whether it passes in front of or behind objects in the static screen display.

Each sprite can be expanded to double size in the horizontal or vertical (or both) dimensions. Multicolored sprites can be made (three colors plus background), but they are not explained in this book.

There are two registers for recording collisions involving sprites. One register records if a sprite has collided with any background drawing on the screen. The other records which sprites have collided with other sprites. These registers keep a record of the collision, even if the sprites have moved off and are no longer overlapping, until the register is read with a PEEK. Then the register is cleared to 0, no collisions.

Questions:

1. How do you tell sprite 0 which picture to show?
2. How do you tell sprite 0 to move to the center of the screen?
3. How do you color sprite 1 red?
4. How do you tell sprite 2 to grow wider?
5. What does the "sprite hits sprite?" register contain if sprite 3 and sprite 5 are overlapping?

Lesson 31. Sprites

The VIC chip in the computer controls eight sprites.

A sprite is a little picture that moves on the screen. You draw the picture and put it in memory beforehand. Sprites let you put powerful moving graphics in your programs.

Sprite Zero, Obey Our Wishes!

```
10 REM ----- SPRITE -----
15 PRINT "{CLR}{BLK}" :REM   BLACK LETTERS
16 POKE 53281,1:REM   WHITE SCREEN
20 REM ----- SET UP VIC
22 V=53248 :REM   VIC BASE ADDRESS
23 REM ----- PICTURE
24 B=200 :REM   BLOCK NUMBER
26 POKE 2040,B :REM   PUT IN BLOCK 200
28 GOSUB 200 :REM   STORE PICTURE
30 REM ----- ENABLE SPRITE 0
32 POKE V+21,1 :REM   TURN ON SPRITE 0
34 POKE V+39,0 :REM   BLACK COLOR
40 REM ----- USE THE SPRITE
42 FOR P=1 TO 200
44 POKE V+0,P :REM   X POSITION
45 POKE V+1,P :REM   Y POSITION
49 NEXT P
50 FOR I=7 TO 0 STEP -1
55 POKE V+39,I :REM   ALL COLORS
60 FOR T=1 TO 1000:NEXT T
65 NEXT I
70 POKE V+29,1 :REM   FAT SPRITE
71 GOSUB 100
72 POKE V+23,1 :REM   FAT AND TALL
73 GOSUB 100
74 POKE V+29,0 :REM   TALL ONLY
75 GOSUB 100
76 POKE V+23,0 :REM   SMALL AGAIN
77 GOSUB 100
90 POKE V+21,0 :REM   TURN SPRITE OFF
96 LIST300-
100 FOR T=1 TO 1000:NEXT T:RETURN
200 REM --- STORE SPRITE ---
201 L=0:PRINT " PLEASE WAIT"
202 FOR I=0 TO 20:READ R$
```

```

206 FOR J=0 TO 2:S$=MID$(R$,J*8+2,8)
210 N=0:P=128
215 FOR K=1 TO 8:D$=MID$(S$,K,1)
220 IF D$="*" THEN N=N+P
225 P=P/2:NEXT K
230 POKE B*64+L,N:L=L+1
280 NEXT J,I:PRINT "{CLR}"
299 RETURN
300 REM -----
301 DATA |*          ***          |
302 DATA | *          *          |
303 DATA |  *          *          |
304 DATA |   *          *          |
305 DATA |    *          *          |
306 DATA |     *          *          |
307 DATA |      *          *          |
308 DATA |       *          *          |
309 DATA |        *          *          |
310 DATA | *          *          *|
311 DATA |***          *          ***|
312 DATA |*          *          *|
313 DATA |          *          *|
314 DATA |           *          *|
315 DATA |            *          *|
316 DATA |             *          *|
317 DATA |              *          *|
318 DATA |               *          *|
319 DATA |                *          *|
320 DATA |                 *          *|
321 DATA |                  ***          *|
322 REM -----

```

Be sure to save before running.

How to Use the Sprite Program

Draw your own picture in the DATA statements and enter the DATA lines into the program, like this:

List the program and see the DATA statements on the screen.

Move the cursor into the rectangle drawn in the DATA statements. Erase the stars that are already there. Draw whatever picture you like (a smiley face?) using the * key.

Then use the CRSR keys to move the cursor to line 301 at the top of the drawing. Then press the RETURN key 21 times to enter all the new DATA statements into the program.

Then run the program to see your sprite move.

How It Works

The VIC chip has 47 memory boxes that tell the sprites what to do. Besides that, each sprite needs a picture stored in a block of 64 boxes in memory, and a sprite pointer box to tell which 64 boxes to use.

Line 22 The addresses of VIC's boxes start at 53248.

Line 26 The box at 2040 is the sprite pointer for sprite 0. It tells in which block of 64 boxes sprite 0 will find its picture. Block 200 has boxes 12800 to 12863. ($12800 = 64 * 200$).

Careful! Use only blocks 200 to 255.

Line 32 Box V+21 is the sprite on/off box. Put number 0 in the box to turn off all the sprites. The number 1 turns on sprite 0.

Line 34 Box V+39 holds a color number (0 to 15) for sprite 0.

Line 44 Box V holds the x position of sprite 0. The number is between 0 and 255. Numbers near 0 put the sprite off the screen at the left.

Line 45 Box V+1 holds the y position of sprite 0. Numbers near 0 put it off the screen at the top. Numbers near 255 put it off the screen at the bottom.

Line 200 The subroutine reads each DATA line, breaks it up into three bytes of eight characters each, and then looks to see which characters are stars. The stars tell which bits are ones in binary notation. Study pages 72 and 73 in the *Commodore 64 User's Guide* to learn more about making binary numbers into decimal numbers for sprites.

Two Steps to Sprites

You use sprites in two steps: First you draw and store a picture. Then you make VIC color it, turn it on, and move it around.

Load the "Sprite" program, then change it.

Remove all lines 20 to 96 and line 201. Add these lines:

```
10 REM ----MAKE PICTURES----
30 INPUT "BLOCK NUMBER";B
35 IF B<200 THEN 30
40 GOSUB 200
96 LIST 300
240 PRINT N,
280 NEXT J:PRINT:NEXT I
292 INPUT "NEXT PICTURE ";A$
```

How to use the program:

LIST it to see the box made of DATA statements.

Draw your new picture.

Then run the program and enter a block number from 200 to 255. The picture is stored in the block.

Also, the numbers that draw the picture are shown on the screen so you can copy them down if you want. You can put these numbers in DATA statements if you want to write a program that uses that picture.

Finally, the DATA statements are LISTed again so you can draw a new picture to store in another block.

You can store many pictures for one sprite to use, and change from one picture to another by POKEing a new block number in the sprite pointer box. (Remember, for sprite 0 the pointer is in box 2040.) Change the pictures very fast and you can make a movie!

Assignment 31A:

1. Use "Make Pictures" to store a smiley face in block 200. Then draw a frowning face and store it in block 201. (The smile in 200 is still there.)

Then write a program to make first the smile, then the frown, show in the middle of the screen. (*Hint: Use sprite 0 and change its pointer B in a loop.*)

2. Now that you know how, store several faces and write a program that changes the face slowly from a big smile, to a small smile, then to a puzzled look, then to a frown, then to an angry face.

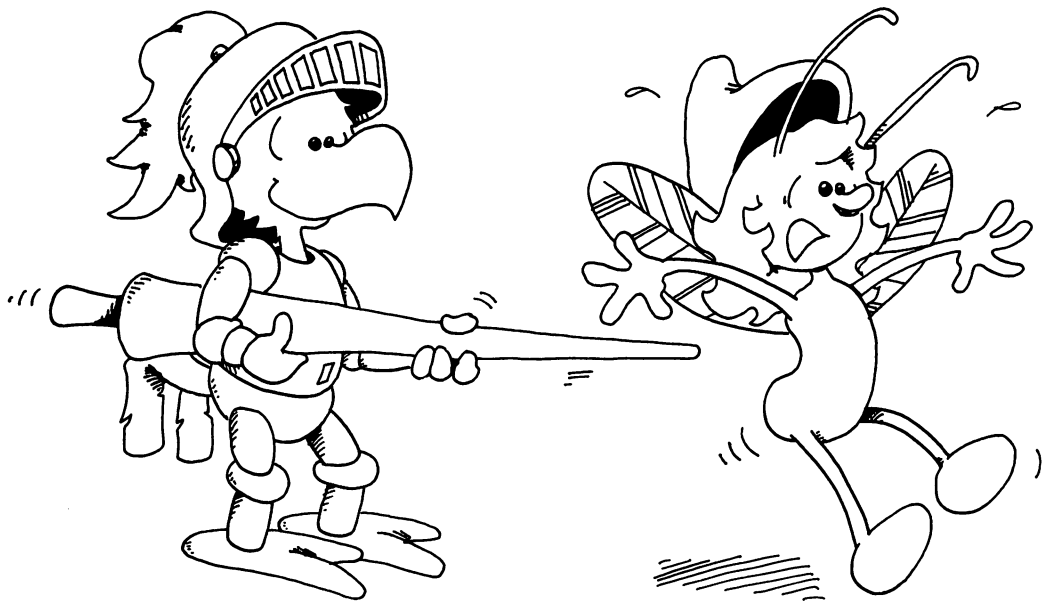
Many Sprites at Once

Here are the box addresses for all eight sprites.

Sprite	Pointer	X	Y	Color	Bit Value	
		V+	V+	V+		(V=53248)
0	2040	0	1	39	1	
1	2041	2	3	40	2	
2	2042	4	5	41	4	
3	2043	6	7	42	8	
4	2044	8	9	43	16	
5	2045	10	11	44	32	
6	2046	12	13	45	64	
7	2047	14	15	46	128	

You already know how to use the pointer, x, y, and color boxes for the 0 sprite. They are used in the same way for the other sprites.

The bit value for each sprite is a number used in the registers listed below.



Registers:	X extension?	V + 16
	Sprites on?	V + 21
	Tall?	V + 23
	Wide?	V + 29
	Behind background?	V + 27
	Sprite hit sprite?	V + 30
	Sprite hit background?	V + 31

Each register is a memory box in the VIC chip that holds the answers to eight yes/no questions, one answer for each sprite.

The bit values of each sprite are used to answer the questions. Example:

```
33 POKE V+21,1+4+128:REM TURNS ON SPRITES 0,2,AND 7
```

Then:

```
56 POKE V+21,1+128:REM TURNS OFF SPRITE 2(BECAUSE
  ITS BIT VALUE,4,ISN'T THERE)
```

Rules:

To answer no for all sprites, POKE 0.

To answer yes for one sprite, POKE its bit value. (This automatically answers no for all the other sprites.)

To answer yes for several sprites, POKE the sum of their bit values. For all bit values you leave out (like 2, 8, 16, 32, and 64 in line 33), the answer is no.

The "x extension?" register helps you move a sprite to the right side of the screen. If you answer yes, then the number 256 is added to the position you put in the sprite's x position register. Example:

```
35 POKE V+0,50:REM PUT SPRITE 0 AT 50 HORIZONTALLY
47 POKE V+16,1:REM MOVE IT TO 256+50=306
```



Watch Out, Two Sprites Are Going to Bump!

When two sprites overlap on the screen, their bit values added together appear in box V+30, the "sprite hit sprite?" register. Example:

```
87 H=PEEK(V+30)
88 PRINT H
```

If 6 is printed, then you know that sprites 1 and 2 have hit each other because their bit values 2 + 4 add to 6.

If a sprite hits something you have drawn on the screen, its bit value is added to box V+31, the "sprite hit background?" register.

Assignment 31B:

1. Make two sprites that can move horizontally. Start them at opposite ends of the screen. When they hit, make a crash sound.
2. Make a bumper car rink. Put a border around the screen. Have two sprites (different colors) for bumper cars. Make eight pictures of cars, going up, down, left, right, and toward each diagonal. Control the cars with keys for "turn left" and "turn right." When the cars hit, they have to back away.

Instructor Notes 32. User-Friendly Programs

This lesson shows how to write clear programs which interact with the user in a friendly way.

They should be clearly structured from the programmer's point of view and the spaghetti program should be discouraged. This lesson presents a format for writing programs. While methods of imposing order on the task are largely a matter of taste, the methods used here can serve to introduce the ideas.

User-friendly means that screen displays are easy to read, keyboard input is "RETURN-key free" as much as possible, and errors are *trapped*. Ask if entries are okay. If not, give an opportunity to fix things.

Instructions and help should be available. Prompts need to be given. Beginners need complete prompts, but experienced users would prefer brief prompts.

It is hard to teach the writing of user-friendly programs. Success depends mostly on the attitude of the programmer. The best advice is to "turn up your annoyance detectors to high" as you write and debug programs.

Most young students will not progress very far toward completely friendly programming. To be acquainted with the desirability of friendly programming and to use some simple techniques toward accomplishing that goal are satisfactory achievements.

Questions:

1. Should your program give instructions whether the user wants them or not?
2. What is a prompt? Give two examples.
3. What is scrolling? How can you write to the screen without scrolling?

-
-
4. If you want the user to enter a single letter from the keyboard, what command is best (avoids using the RETURN key)?
 5. What is an error trap? How would you trap errors if you asked your user to enter a number from 1 to 5?
 6. In what part of the program are most of the GOSUB statements found?
 7. Why put the "starting stuff" section of the program at the end of the program (at high line numbers)?

Lesson 32. User-Friendly Programs

There are two kinds of users:

1. Most want to *run* the program. They need:

- instructions
- prompts
- clear writing on the screen
- no clutter on the screen
- erasing old stuff from the screen
- not too much key pressing
- protection from their own errors

2. Some want to *change* the program. They need:

- a program made in parts
- each part with a title in a REM
- explanations in the program

(Don't forget you are a user of your own programs, too. Be kind to yourself!)

Programs Have Three Parts

"Starting Stuff" (at the beginning of the program run):

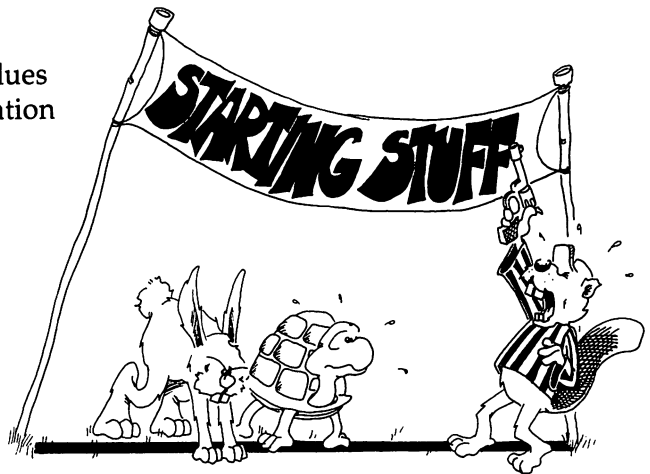
- gives instructions to the user
- draws a screen display
- sets variables to their starting values
- asks the user for starting information

Main Loop:

- controls the order in which tasks are done
- calls subroutines to do the tasks

Subroutines:

- do parts of the program



Program Outline

```
1 GOTO 1000:REM          *** program name ***
---
100 REM MAIN LOOP
---
---                      calls subroutines
---
199 END
1000 REM
1001 REM                *** program name ***
1002 REM
---
---                      REMs that give a description of
---                      the program, variable names, etc.
---
1999 REM
2000 REM STARTING STUFF
---
---                      asks for starting information
---                      sets variable values
---                      gives instructions
---
2999 GOTO 100
```

Save the outline and use it to start each new program that you write.

Put the Main Loop at the Beginning of the Program

Put the main loop near the front because it will run faster there.

Put Starting Stuff at the End of the Program

Put the starting stuff near the back, because it may be the biggest part of the program, and you may keep adding to it as you write, to make the program more user-friendly. It doesn't need to run fast.

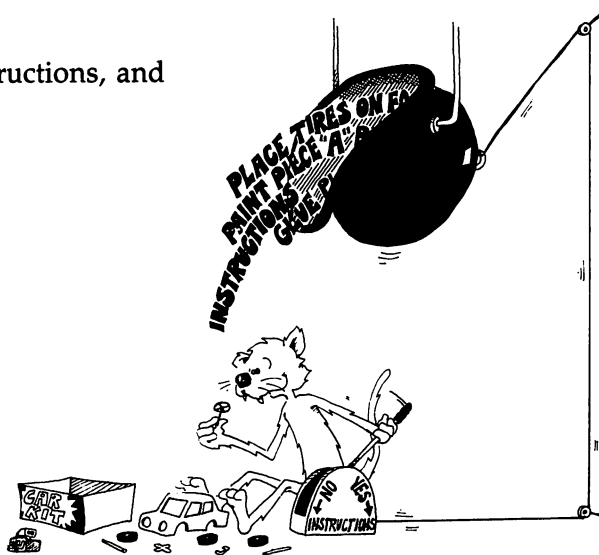
Put Subroutines in Three Places

Put subroutines that must run fast between lines 2 and 99, starting stuff subroutines after line 2999, and the rest between lines 200 and 999.

Information Please

380 PRINT "DO YOU WANT TO SEE INSTRUCTIONS?<Y/N>"

This lets a beginner see instructions, and lets others say no.

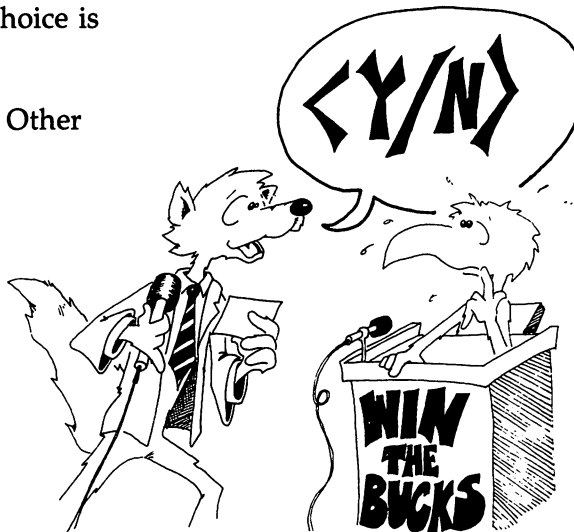


Tie a String Around the User's Finger

Use a prompt to remind users what choices they have.

Example: <Y/N> where the choice is Y for yes or N for no.

Beginners need long prompts. Other users like short prompts.



Ouch! My Fingers Hurt

Use the GET statement to enter single letters. This saves having to press RETURN.

```
380 PRINT "DO YOU WANT TO SEE INSTRUCTIONS?<Y/N>"
382 GET R$:IF R$="" THEN 382
384 IF R$="Y" THEN GOTO 600
```

Don't Give the User a Headache

Scrolling gives headaches!

Your screen will scroll if you continue PRINTing after reaching the bottom of the screen. New lines will be written at the bottom of the screen and old lines will be pushed up.

This is like the scrolls the Romans used for writing. They unwound from the bottom and wound up at the top.

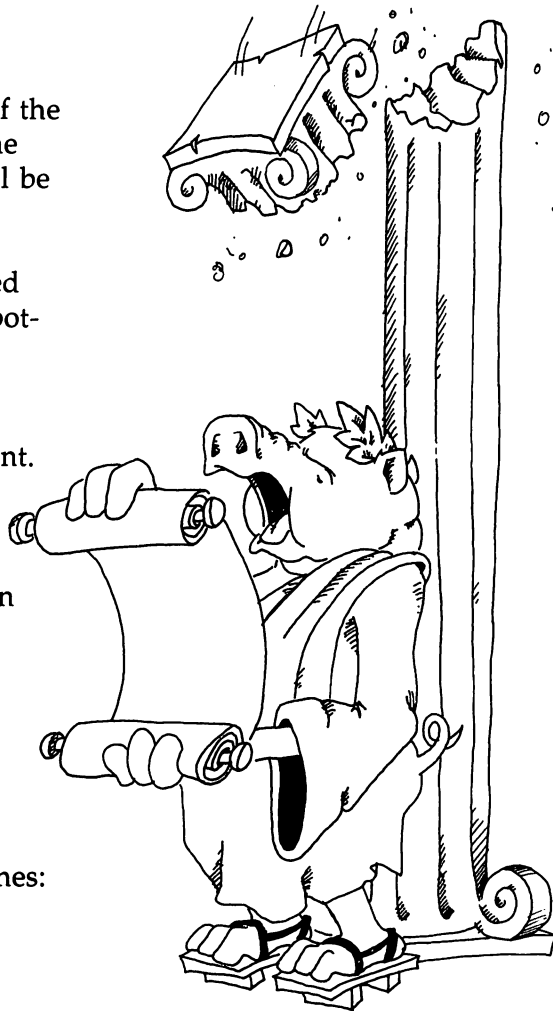
Avoid scrolling. Use CLR/HOME and CRSR keys to print just where you want. Erase by printing a string of blanks to the same spot.

Use delay loops so the writing stays on the screen while the user reads it.

Set Traps for Errors

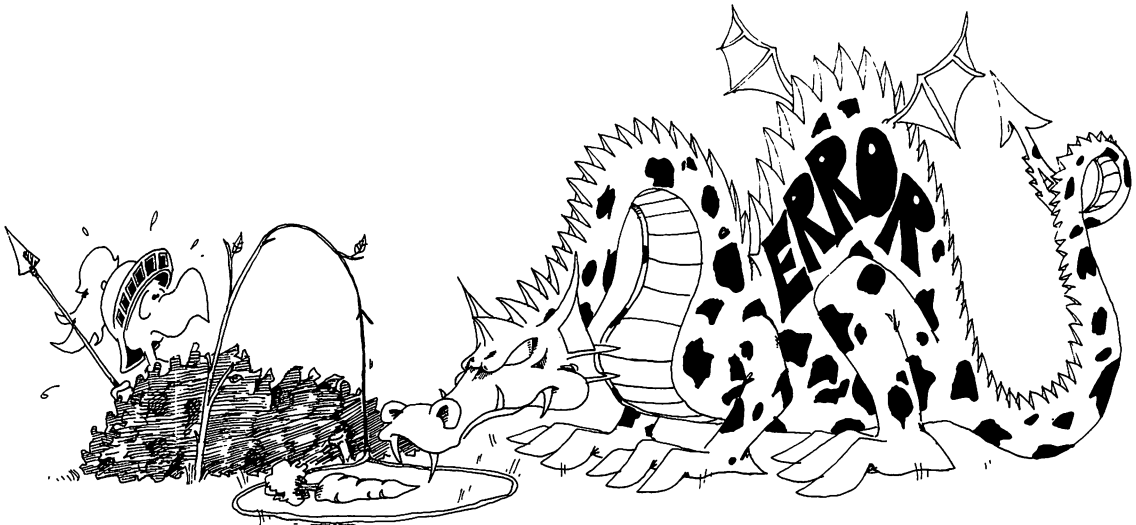
Example: Add this line to the above lines:

```
386 IF R$<>"N" THEN GOTO 380
```



Line 380 asked for only two choices, Y or N. If some other key is pressed, line 386 sends the user back to line 380.

Traps make your program bombproof so that users will be unable to goof it up!



Assignment 32:

1. Make a program to write a very large number, 50 digits. Pick the digits at random. Put a comma between each set of three digits. (*Hint: Make the digits one at a time from RND(0) and glue them on the end of the number.*)
2. Write a secret cipher program. The user chooses a password and it is used to make a cipher alphabet like this:

If the password is DRAGONETTE, remove the repeated letters, GET DRAGONET, put it at the front of the alphabet and the rest of the letters after it in the normal order.

DRAGONETBCFHJKLMPQSUVWXYZ cipher alphabet

ABCDEFGHIJKLMNPOQRSTUVWXYZ normal alphabet

The user chooses to code or decode from a menu.

Instructor Notes 33. Debugging, STOP, CONT

Since the “sigh and moan” technique is a loser, our students need a bag of tricks that will help isolate program bugs. They should practice on the programs they are writing as they go through this book.

The inexperienced debugger feels hopeless inertia when a program doesn’t work right. Rather than sit and stare, it is more useful to try some changes. Any changes are better than none, but random changes are very inefficient. The best changes are those that eliminate sections of the program from the list of possible hiding places for the bug.

As programs grow in complexity, more of the bugs result from unforeseen interactions between separate parts of the program. The bag of tricks we offer also helps find these. Delay loops, PRINT statements, and STOP statements help the student see how the program is functioning.

Don’t overlook those techniques you can use after the program is stopped with a STOP statement or a RUN/STOP keypress. You can PRINT out any variable values you like, to see what the program has done. You can also do arithmetic in the PRINT statement to check what the program should be doing. You can even change variable values (for example, the value of a loop variable). When you are ready, CONTInue the program run.

Questions:

1. How can you make the computer print

`BREAK IN LINE 55`

by adding a line in the program?

2. How are the STOP and the END statements different?

3. How are the STOP statement and the RUN/STOP key different?

-
-
4. What does the CONT command do?
 5. Why would you put STOP statements in your program?
 6. How do delay loops help you debug a program?
 7. How do extra PRINT statements help you debug a program?
 8. Why do you take the STOP and extra PRINT statements out of the program after you have fixed the errors?
 9. Can you pick the line where the RUN/STOP key will stop the program? Can you pick the line if you are using the STOP statement?

Lesson 33. Debugging, STOP, CONT

The STOP Statement

Enter and run:

```
10 REM SECRET STOP
20 PRINT "{CLR}"
25 R=INT(RND(8)*200)
30 FOR I=0 TO 200
40 IF I=R THEN STOP
50 NEXT I
```

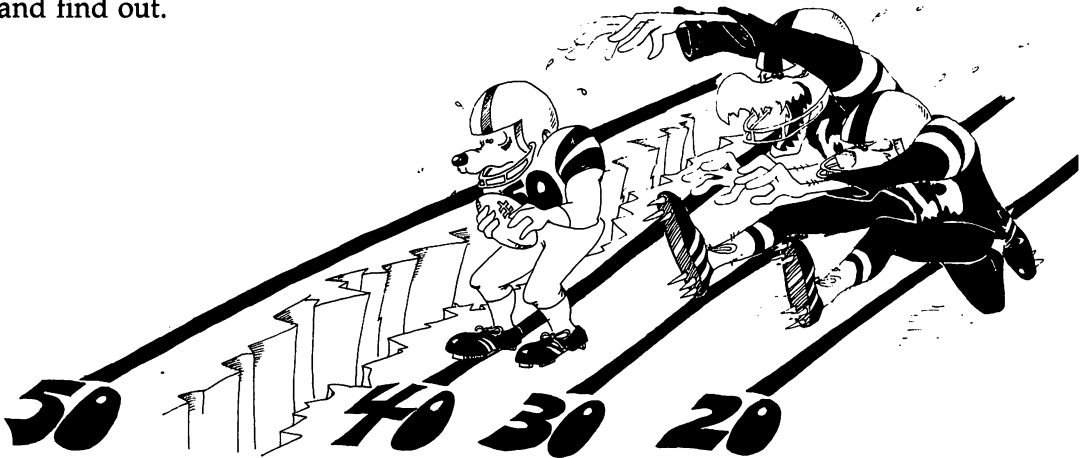
The program will stop, and the computer will print a message:

BREAK IN LINE 40

What do you suppose the secret value of I was?

Enter: PRINT I (No line number)

and find out.



How to Start It Again

Enter the command CONT. Try it!

STOP is like END.

STOP makes the computer stop and enter the edit mode.

It is like END except it prints the number of the line that the STOP is in.

You can have as many STOP statements in your program as you like.

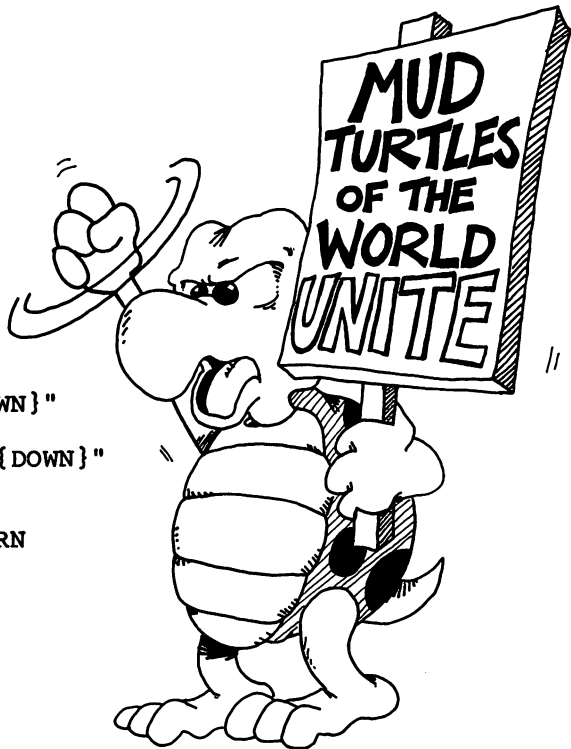
STOP is used for debugging your program.

Another Way to Stop Running the Program

You can stop running the program by pressing the RUN/STOP key.

Try it:

```
10 REM GO FOREVER
15 PRINT "{CLR}{DOWN}"
16 GOSUB 90
20 PRINT "MUD{DOWN}"
21 GOSUB 90
22 PRINT "    TURTLES{DOWN}"
23 GOSUB 90
24 PRINT "        OF{DOWN}"
25 GOSUB 90
26 PRINT "            THE{DOWN}"
27 GOSUB 90
28 PRINT "                WORLD{DOWN}"
29 GOSUB 90
30 PRINT "                    UNITE{DOWN}"
31 GOSUB 90
40 GOTO 10
90 FOR T=1 TO 1000:NEXT T:RETURN
```



Pressing the RUN/STOP key stops the program wherever it is. It prints:

`BREAK IN LINE XX` and enters the edit mode

where XX is the line number where it stops.

The command CONT starts the program again at the same spot.

The above program usually stops in line 90. Why? Try to make it stop in some other line.

What Do You Do After You Stop?

You put STOP in whatever part of your program is not working right. Then you run the program. After it stops, you look to see what happened.

(Or you use the RUN/STOP key to stop the program, but it may not stop in the spot where the trouble is.)

Put on your thinking cap. Ask yourself questions about what happened as the program ran.

You are in the edit mode. You can:

List parts of the program and study them.

Use the PRINT statement to look at variables. Do they have the values you expected?

Do little calculations on the computer in the edit mode to check what the computer is doing.

Use the LET statement to change the values of variables.

If you find the trouble, you may add lines, change lines, or delete lines.

Starting the Program Again

There are four ways to start a program. They are:

CONT	if you have not changed the program
GOTO XX	where XX is a line number
RUN XX	where XX is a line number
RUN	your old friend

You may use the CONT command if you have not:

- added a line
- deleted a line
- or changed a line by editing it

Or you may start running the program at a different spot by entering (without line number) the statement:

GOTO XX

where XX is the line number you want to restart at.

If you have changed the program, your only choice is to start at the beginning or at some other line number XX with RUN.

What is the difference between these four ways?

CONT and GOTO XX use the values in the variable boxes left over from the last time you ran.

CONT starts at the line where the BREAK occurred.

GOTO XX starts at line XX.

RUN and RUN XX throw away all the variable boxes made the last time, then execute the program.

RUN starts at the first line of the program.

RUN XX starts at line XX.

Prepare a Disk for the Student

You should prepare a special disk for the student's exclusive use. All the programs that the student writes or copies from the book can be put on that disk. This makes the programs easy to find and isolates other disks from horrible accidents, like accidental erasure or getting clobbered.

You must use the disk NEW command to format any new, fresh-from-the-box disk before you can store information on it. (Do not confuse the disk NEW command with the BASIC NEW command.) If you have an old disk which you are willing to erase, you can NEW it instead of a brand new disk.

Put the disk in the drive this way: Hold the disk with the label up and your thumb on the label so that you can insert the nonlabel end into the disk drive. Push the disk in until it stays, and then close the drive door until it clicks out and stays shut.

Now enter:

```
OPEN 15,8,15,"NØ:STUDENT DISK,S1"
```

The drive will clatter and whirl, the red light will come on, and after about a minute, the light will go off and the drive will stop. (If the red light continues to blink, a disk error has occurred. Check that you typed the command correctly, and that no little paper tab is stuck over the write-protect notch on the disk. Repeat the command. If you still have trouble, consult the disk drive manual.)

Now enter:

```
CLOSE 15
```

What's in a Name?

We named the disk STUDENT DISK. You can use any other name you wish, just so long as it contains 16 or less characters and no commas, question marks, quotes, asterisks, or colons.

The number S1 is the disk ID number. It is important to use a unique set of two letters and/or numbers for each disk in your collection. Some advanced disk-using programs require each disk to have an ID number different from other disks used in the program.

Check It Out

To check that the disk is okay, enter:

```
LOAD "$",8
```

Careful! LOAD will erase any program you have in memory.

Now enter:

```
LIST
```

You should see:

```
Ø "STUDENT DISK      " S1 2A
```

with all but the zero in reverse letters on the screen. The LOAD statement erased any programs you had in the computer. The 2A identifies the kind of Commodore equipment used.

When you open the door to take the disk out, the disk should automatically be pushed out an inch.

Practice Makes Perfect

Now would be a good time to turn to lesson 14 and practice saving and loading programs to the disk.

Saving Programs on Cassette Tape

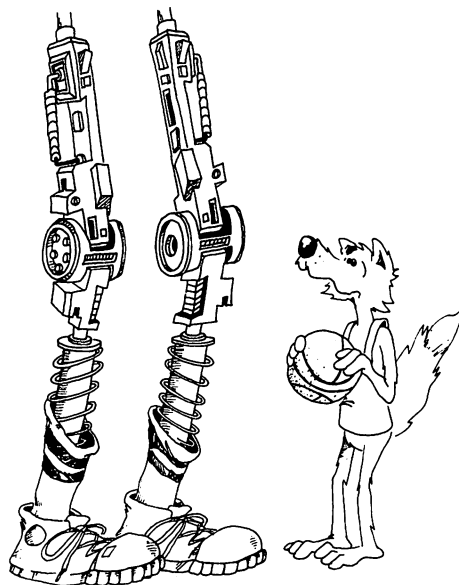
If you do not have a disk drive, you can save and load programs on tape in a Datasette, using the commands:

```
SAVE"HI"
```

```
LOAD"HI"
```

Reserved Words

ABS	AND	ASC	ATN						
CHR\$	CLR	CMD	CLOSE	CONT	COS				
DATA	DEF	DIM							
END	EXP								
FN	FOR	FRE							
GET	GOSUB	GOTO							
IF	INPUT	INPUT#	INT						
LEFT\$	LEN	LET	LIST	LOAD	LOG				
MID\$									
NEW	NEXT	NOT							
ON	OR	OPEN							
PEEK	POKE	POS	PRINT	PRINT#					
READ	REM	RESTORE	RETURN	RIGHT\$	RND	RUN			
SAVE	SGN	SIN	SPC(SQR	STEP	STOP	STR\$	SYS	
TAB(TAN	THEN	TI	TI\$	TO				
USR									
VAL	VERIFY								
WAIT									



Answers to Assignments

The answers below are in order by lesson number. Look at the first REM statement in each program listing to find the lesson it matches. The second REM statement will give you the name of the program.

```
1 REM A1-3
10 REM NEW FRIEND
20 PRINT "HI, THERE"
30 PRINT "COMPUTER"
```

```
1 REM A2-2
10 REM COLORED NAMES
20 POKE 53281,0
30 PRINT "{WHT}M{RED}I{CYN}N{PUR}D{GRN}A"
35 PRINT "{YEL} ANNE"
40 PRINT "{RVS}{RED} CARLSON{WHT}"
```

```
1 REM A3-5
10 REM BIRDS
15 POKE 53281,0
20 PRINT "{CLR}"
22 PRINT
24 PRINT
30 PRINT "{CYN} UQI"
32 PRINT
34 PRINT
40 PRINT "{YEL}           JQK"
42 PRINT
44 PRINT
46 PRINT
50 PRINT "{GRN}           CWC"
90 PRINT "{YEL}"
```

```

1 REM A4-3
10 REM SMILE
15 PRINT"{CLR}"
20 PRINT
22 PRINT
24 PRINT
26 PRINT"{RED}"
30 PRINT"      OO      OO"
32 PRINT"      OO      OO"
34 PRINT
35 PRINT
36 PRINT
37 PRINT
38 PRINT
40 PRINT
42 PRINT" *                *"
44 PRINT"  *                *"
46 PRINT"    *                *"
48 PRINT"          *****"

```

```

1 REM A5-1 DAN
15 PRINT"{CLR}"
20 PRINT"NAME A MUSICAL GROUP"
30 INPUT A$
40 PRINT
50 PRINT"NAME ONE OF THEIR SONGS "
60 PRINT
70 INPUT B$
75 PRINT
80 PRINTA$;" PLAYS ";B$

```

```

1 REM A5-2
10 REM 3 PRINTS
15 PRINT"{CLR}"
20 PRINT"GIVE ME THE NAME OF A MUSICAL GROUP"
22 INPUT G$
25 PRINT
30 PRINT"WHAT IS ONE OF THEIR TUNES"
35 INPUT T$
40 PRINT"{DOWN}"
50 PRINTG$;
52 PRINT" PLAYS ";
54 PRINTT$

```

```
1 REM A5-3
10 REM THREE LITTLE WORDS
20 POKE 53281,0
21 PRINT "{WHT}"
30 PRINT "GIVE ME A WORD "
31 INPUT W$
40 PRINT "GIVE ME ANOTHER"
41 INPUT X$
50 PRINT "ONE MORE"
51 INPUT Y$
60 PRINT "{CLR}"
62 PRINT "{3 DOWN}"
65 PRINT "YOU MEAN:"
66 PRINT "{3 DOWN}"
70 PRINT W$,X$,Y$
```

```
1 REM A6-1
10 REM SILLY GOOSE
15 PRINT "{CLR}"
17 PRINT "{GRN}"
20 PRINT
22 PRINT
24 PRINT
30 PRINT "HELLO,"
31 PRINT
32 PRINT "WHAT IS YOUR NAME?"
33 PRINT
35 INPUT N$
37 PRINT "{CLR}{PUR}"
38 PRINT
40 PRINT "WELL,"
41 PRINT
42 PRINT N$
43 PRINT
50 PRINT "IT IS SILLY TO TALK"
51 PRINT
52 PRINT "TO A COMPUTER!"
55 PRINT "{BLK}"
```

```
1 REM A6-2
10 REM ***FAVORITE***
15 PRINT"{CLR}{RED}"
20 PRINT"{2 DOWN}WHAT IS YOUR FAVORITE COLOR?"
25 PRINT
26 INPUT C$
30 PRINT "{DOWN}{BLU}I PUT THAT IN BOX C$."
35 PRINT"{DOWN}{GRN}NOW YOUR FAVORITE ANIMAL?
   {DOWN}"
40 INPUT C$
45 PRINT"{DOWN}{YEL}I PUT THAT IN BOX C$ TOO"
50 PRINT"{DOWN}{BLK}NOW LET'S SEE WHAT IS IN BOX C
   $"
55 PRINT"{DOWN}IT IS:"
56 PRINT
60 PRINT C$
```

```
1 REM A7-2
10 REM FEELINGS
12 PRINT "{CLR}"
20 PRINT
22 PRINT
24 PRINT " HOW IS THE WEATHER?"
26 PRINT
28 INPUT W$
29 PRINT
30 PRINT " AND HOW DO YOU FEEL?"
32 PRINT
34 INPUT F$
36 PRINT
38 PRINT " YOU MEAN:"
40 PRINT
45 S$=W$ + " AND " + F$
50 PRINT " ";S$
```

```
1 REM A8-2
10 REM TEEN TIMES
11 REM
20 PRINT " ";
21 PRINT "T E E N P O W E R"
22 PRINT
23 PRINT
24 PRINT
25 PRINT
30 GOTO 20
```

```
1 REM A8-4
10 REM COLOR FAST FRIENDS
15 POKE 53281,0
20 PRINT "{CLR}{WHT}"
22 PRINT
24 PRINT
30 PRINT "{YEL}    BRIAN"
40 PRINT "{GRN}    STEVE"
50 PRINT
90 GOTO 30
```

```
1 REM A9A-2
10 REM SPORTS
20 PRINT "{CLR}":PRINT:PRINT
25 PRINT "WHICH DO YOU LIKE, FOOTBALL OR BASEBALL?"
   "
30 INPUT A$
32 PRINT:PRINT:PRINT
35 IF A$="FOOTBALL" THEN PRINT "FOOTBALL IS FUN"
40 IF A$="BASEBALL" THEN PRINT "PLAY BALL!"
```

```
1 REM A9B-2
10 REM WHAT COLOR
15 POKE 53281,0
20 PRINT "{CLR}{WHT}{3 DOWN}"
30 PRINT "  PLAYER 1 TURN YOUR BACK{DOWN}"
32 PRINT "  PLAYER 2 ENTER A COLOR NAME{DOWN}"
40 INPUT C$
45 PRINT "{CLR}{3 DOWN}"
50 PRINT "{DOWN}  PLAYER 1 TURN AROUND AND GUESS?
   {DOWN}"
55 INPUT G$
56 PRINT
60 IF G$=C$ THEN GOTO 80
65 PRINT "{YEL}  WRONG{DOWN}"
69 GOTO 55
80 REM CORRECT ANSWER
82 PRINT "{GRN}  RIGHT!"
90 PRINT "{WHT}"
```

```
1 REM A10-1
10 REM BIRTHDAY
15 POKE 53281,0
20 PRINT "{CLR}{WHT}{3 DOWN}"
30 PRINT " HOW OLD ARE YOU?{DOWN}"
31 INPUT A
32 PRINT
33 PRINT " AND WHAT YEAR IS IT NOW?{DOWN}"
40 INPUT Y
45 B=Y-A
50 PRINT "{DOWN} HAS YOUR BIRTHDAY COME YET THIS Y
EAR?"
51 PRINT " <Y OR N>"
52 PRINT
55 INPUT A$
57 PRINT
60 IF A$="N" THEN B=B-1
70 PRINT "{GRN} YOU WERE BORN IN";B
90 PRINT "{WHT}"
```

```
1 REM A10-2
10 REM MULTIPLICATION
15 POKE 53281,0
20 PRINT "{CLR}{WHT}{3 DOWN}"
30 PRINT " GIVE ME A NUMBER{DOWN}"
31 INPUT N1
32 PRINT
33 PRINT " GIVE ME ANOTHER{DOWN}"
40 INPUT N2
41 PRINT
45 A=N1*N2
50 PRINT " HERE IS THEIR PRODUCT:{DOWN}"
55 PRINT " ";A
90 PRINT "{WHT}"
95 REM TRY N1=20000000 AND N2=30000000
96 REM THE ANSWER IS 6E+12 IN SCIENTIFIC
97 REM NOTATION
```

```

1 REM      A11A-1
10 REM NICKNAMES
15 POKE 53281,0
20 PRINT"{CLR}{WHT}{3 DOWN}"
24 PRINT"  PLAYER 1: WHAT IS YOUR LAST NAME?{DOWN}"
   "
25 INPUT LN$
26 PRINT
30 PRINT"  PLAYER 1 TURN YOUR BACK{DOWN}"
31 PRINT"  PLAYER 2: WHAT IS HIS NICKNAME{DOWN}"
32 INPUT NN$
33 PRINT
41 PRINT"  PLAYER 1: TURN AROUND AND PRESS RETURN
   {DOWN}"
46 INPUT A
50 PRINT"{CLR}{9 DOWN}"
55 PRINT"  AROUND HERE YOU ARE CALLED:{DOWN}"
60 PRINT "{YEL}  ";NN$;"{GRN}  ";LN$
90 PRINT"{8 DOWN}{WHT}"

```

```

1 REM A11A-2
10 REM !"#$$% INSULTS %$#!"
15 POKE 53281,0
20 PRINT"{CLR}{WHT}{3 DOWN}"
24 PRINT"  HEY YOU!  WHAT IS YOUR NAME!{DOWN}"
25 INPUT N$
26 PRINT
27 PRINT"  BAH!  ";N$;"..."
30 REM DELAY LOOP
31 FOR T=1 TO 1000:NEXT T
33 PRINT
50 PRINT"{CLR}{11 DOWN}{YEL}  YOUR FATHER EATS ONI
   ONS!"
90 PRINT"{8 DOWN}{WHT}"

```

```

1 REM      AllB-1
10 REM SLOWPOKE
15 POKE 53281,0
20 PRINT "{CLR}{WHT}{3 DOWN}"
24 PRINT "  LIKE...{3 DOWN}"
25 FOR T=1 TO 1000:NEXT T
26 PRINT
30 PRINT "  MOLASSES...{3 DOWN}"
31 FOR T=1 TO 1000:NEXT T
33 PRINT
40 PRINT "      IN...{3 DOWN}"
41 FOR T=1 TO 1000:NEXT T
42 PRINT
50 PRINT "      JANUARY!"
90 PRINT "{DOWN}{WHT}"

```

```

1 REM AllB-2
10 REM CLOCK
15 POKE 53281,0
20 PRINT "{CLR}{WHT}{3 DOWN}"
23 PRINT "  PRESENT TIME: HR,MIN,SEC?"
24 PRINT
25 INPUT H,M,S
26 LET F=550
30 FOR I=1 TO F:NEXT I
31 PRINT " {HOME}{DOWN}          "
32 PRINT " {HOME}{DOWN}";H;TAB(3);M;TAB(6);S
33 LET S=S+1
40 IF S<60 THEN GOTO 26
41 LET M=M+1
42 LET S=0
45 IF M<60 THEN GOTO 26
50 LET H=H+1
51 LET M=0
60 IF H=24 THEN LET H=0
61 GOTO 26
90 PRINT "{DOWN}{WHT}"

```

```
1 REM A12B-3
10 REM I GOT YOUR NUMBER
15 POKE 53281,0
20 PRINT "{CLR}{WHT}{3 DOWN}"
23 PRINT " GIVE ME A NUMBER 0 TO 10{DOWN}"
25 INPUT N
26 PRINT
30 IF N=0 THEN PRINT " I GOT PLENTY OF NOTHING"
31 IF N=1 THEN PRINT " I'M NUMBER ONE!"
32 IF N=2 THEN PRINT " TWO IS COMPANY"
33 REM ETC.
40 FOR T=1 TO 1000:NEXT T
80 IF N<10 THEN GOTO 20
90 PRINT " THAT'S ALL, FOLKS"
```

```

1 REM A12B-4
10 REM *** PICK A CARD ***
15 POKE 53281,0:LET G=1
20 PRINT"{CLR}{YEL}{3 DOWN}"
22 PRINT "PLAYER2:TURN YOUR BACK{DOWN}{CYN}"
24 PRINT "PLAYER1:THINK OF A CARD,HIT RETURN{DOWN}
"
26 INPUT A$
30 PRINT "{CLR}{3 DOWN}"
32 PRINT "PLAYER1:TELL ME THE SUIT{DOWN}{PUR}"
34 PRINT "HEARTS,DIAMONDS,CLUBS OR SPADES{DOWN}"
36 INPUT S$
40 PRINT"{CYN}{3 DOWN}"
42 PRINT"NOW, TELL ME THE VALUE 1 TO 13{DOWN}"
44 INPUT V
50 PRINT"{CLR}{YEL}{3 DOWN}"
52 PRINT "PLAYER2:GUESS THE CARD.WHICH SUIT?{DOWN}
{PUR}"
54 PRINT "HEARTS,DIAMONDS,CLUBS OR SPADES{DOWN}"
60 INPUT GS$:IF GS$=S$ THEN GOTO 70
62 PRINT "{GRN}{2 DOWN}"
64 PRINT "WRONG, GUESS AGAIN.WHICH SUIT?{DOWN}
{PUR}"
68 LET G=G+1:GOTO 60
70 PRINT"{CLR}{YEL}{3 DOWN}"
72 PRINT "RIGHT!NOW, WHAT VALUE 1 TO 13?{DOWN}"
80 INPUT GV:IF GV=V THEN GOTO 90
82 PRINT "{GRN}{2 DOWN}"
84 IF GV<V THEN PRINT "WRONG,GUESS HIGHER. WHAT VA
LUE?{DOWN}{YEL}"
86 IF GV>V THEN PRINT "WRONG,GUESS LOWER. WHAT VAL
UE?{DOWN}{YEL}"
88 LET G=G+1:GOTO 80
90 PRINT"{CLR}{YEL}{3 DOWN}"
92 PRINT "CONGRATULATIONS!"
94 PRINT "YOU GUESSED IT IN ONLY";G;"TRIES{2 DOWN}
"
96 PRINT "PLAY AGAIN";:INPUT A$:IF A$="YES" THEN G
OTO 10
98 PRINT "{4 DOWN}"
99 PRINT "SEE YOU AGAIN SOON!{WHT}"

```



```

1 REM A15-2
10 REM III VACATION III
12 POKE 53281,0:PRINT"{CLR}{WHT}"
20 REM HEADING
22 PRINT"{DOWN} VACATION"
24 PRINT"{DOWN} CHOOSING"
26 PRINT"{DOWN} PROGRAM"
28 PRINT"{DOWN}{GRN} CHOOSES YOUR VACATION BY THE
   {SPACE}AMOUNT"
30 PRINT"{DOWN}{YEL} YOU WANT TO SPEND"
32 FOR T=1 TO 2000:NEXT T
35 REM INSTRUCTIONS
40 PRINT "{CLR}{3 DOWN} ENTER THE AMOUNT IN DOLLAR
   S"
42 PRINT"{DOWN} YOU WANT TO SPEND"
50 REM GET DOLLAR AMOUNT
51 PRINT
52 INPUT D
53 PRINT
60 M$=" FLIP PENNIES WITH YOUR KID BROTHER"
61 N$=" SPEND THE AFTERNOON IN BEAUTIFUL HOG
   {SPACE}WALLOW, MI"
62 P$=" ENTER A PICKLE EATING CONTEST IN
   {SPACE}SCRATCHYBACK, TN"
63 REM ETC.
69 Z$="BUY A COSY YACHT AND CRUISE THE CARIBBEAN S
   EA"
70 IF D<.5 THEN PRINT M$:GOTO 90
71 IF D<5 THEN PRINT N$:GOTO 90
72 IF D<50 THEN PRINT P$:GOTO 90
90 REM ENDING OF PROGRAM, SAY SOMETHING
91 REM FOR A CLOSING

```

```

1 REM A16-1
10 REM FLYING BALL
12 POKE 53281,0:PRINT"{CLR}{WHT}{5 DOWN}"
20 N$="Q"
25 I=-1
30 I=I+1
35 PRINT"{HOME}{10 DOWN}";TAB(I);" ";N$
37 FOR T=1 TO 10:NEXT T
40 IF I<38 THEN 30

```

```
1 REM A16-2
10 REM BLINKING NAME
12 POKE 53281,7:PRINT"{CLR}{WHT}{5 DOWN}"
20 N$="K A R E N"
30 PRINT"{HOME}{GRN}{7 DOWN}      ";N$
35 FOR T=1 TO 500:NEXT T
40 PRINT"{HOME}{RED}{7 DOWN}      ";N$
45 FOR T=1 TO 500:NEXT T
90 GOTO 30
```

```
1 REM A17A-1
10 REM RABBITS
12 POKE 53281,0:PRINT"{CLR}{WHT}{2 DOWN}"
20 FOR I=0 TO 100 STEP 5
30 PRINT I
40 FOR T=1 TO 200:NEXT T
50 NEXT I
```

```
1 REM A17B.3
10 REM CLIMBING NAME
12 PRINT "{CLR}{22 DOWN}"
15 N$ = " STANISLAUS MAZURSKY"
20 FOR I=1 TO 23
30 PRINT N$
32 FOR T=1 TO 200:NEXT T
35 PRINT "{UP}"
36 PRINT "{3 UP}"
40 NEXT I
```

```
1 REM A17B-5
10 REM OPERA SOPRANO
12 PRINT"{CLR}{WHT}{3 DOWN}"
20 FOR I=1 TO 3
21 POKE 53281,I+5
22 PRINT "{DOWN}SING"
24 FOR T=1 TO 100:NEXT T
30 FOR J=1 TO 3
32 PRINT "{DOWN}TRA";
34 FOR T=1 TO 100:NEXT T
40 FOR K= 1 TO 3
42 PRINT "LA";
44 FOR T=1 TO 100:NEXT T
50 NEXT K
52 PRINT
60 NEXT J
70 NEXT I
90 POKE 53281,0
```

```

1 REM A19-2
10 REM FLY AWAY
12 C=54272:POKE C+1,9
13 POKE C+5,9:POKE C+6,40
14 POKE C+24,15
20 PRINT"{CLR}{BLK}{22 DOWN}";
25 PRINT"{PUR}"
30 PRINT"UQI{3 LEFT}";
32 POKE C+4,33
40 FOR T=1TO200:NEXT T
42 POKE C+4,32
45 PRINT" {2 LEFT}{UP}";
50 PRINT"JQK{3 LEFT}";
60 FOR T=1TO200:NEXT T
65 PRINT" {2 LEFT}{UP}";
70 GOTO 30

```

```

1 REM A20-6
10 REM ARE YOU QUICK?
11 PRINT"{CLR}{3 DOWN}          HIT RETURN WHEN YOU SE
   E"
12 PRINT"{DOWN}          THE QUESTION MARK"
13 PRINT"{2 DOWN}          NOT TOO HARD!"
14 FOR I=1 TO 3000:NEXT I
15 PRINT"{CLR}"
20 FOR I=1 TO RND(0)*9000+500:NEXT I
30 PRINT"{HOME}{9 DOWN}";TAB(17);
31 T=TI
32 INPUT A
40 T=TI-T
41 IF T<3 THEN PRINT "JUMPED THE GUN"
44 T=T/60
45 T=INT(T*100)/100
50 PRINT "{2 DOWN}          YOUR TIME WAS";T;"SECONDS"
60 FOR I=1 TO 3000:NEXT I
99 GOTO 15

```

```

1 REM A21-2
10 REM ALL COLORS
15 S=53281
16 B=S-1
20 FOR I=0 TO 15
25 POKE B,I
30 FOR J=0 TO 15
40 POKE S,J
42 FOR T=1 TO 200:NEXT T
50 NEXT J,I

```

```

1 REM A23-1
10 REM MENU MAKER
15 PRINT"{CLR}{3 DOWN}"
20 PRINT"    WHICH COLOR"
21 PRINT
22 PRINT"    <R> RED"
24 PRINT"    <Y> YELLOW"
26 PRINT"    <G> GREEN"
28 PRINT"    <C> CYAN"
29 PRINT
30 FOR T=1 TO 300:NEXT T
31 GET C$:IF C$="" THEN 31
35 IF C$="R" THEN C=2
36 IF C$="Y" THEN C=7
37 IF C$="G" THEN C=5
38 IF C$="C" THEN C=3
40 POKE 646,C
50 PRINT"COLOR"
60 PRINT "{WHT}"
70 FOR T=1 TO 900:NEXT T
80 GOTO 15
99 REM MAKE A STAR OF THIS COLOR

```

```

1 REM A26-2
10 REM ANSWERER
12 PRINT"{CLR}{3 DOWN}{WHT}"
20 PRINT"    ENTER A QUESTION(AT LEAST THREE WORDS)
    {DOWN}"
25 INPUT Q$
28 L=LEN(Q$)
30 REM TAKE OFF "?"
33 Q$=LEFT$(Q$,L-1)+"."
36 REM LOOK FOR WORD END
39 FOR I=1 TO L
40 C$=MID$(Q$,I,1)
45 IF C$=" " THEN S1=I:I=L
46 NEXT
50 FOR I=S1+1 TO L
52 C$=MID$(Q$,I,1)
54 IF C$=" " THEN S2=I:I=L
56 NEXT I
58 REM EXCHANGE FIRST AND SECOND WORDS
60 S$=MID$(Q$,S1+1,S2-S1)
62 V$=LEFT$(Q$,S1)
63 PRINT
65 PRINT "    ";S$+V$+RIGHT$(Q$,L-S2)

```

```

1 REM A26-3
10 REM *** PIG LATIN ***
12 PRINT"{CLR}{3 DOWN}{WHT}"
20 PRINT"{DOWN} PIG LATIN"
25 INPUT"{DOWN} GIVE ME A WORD: ";W$
35 L=LEN(W$)
40 FOR I=1 TO L
42 L$=MID$(W$,I,1)
45 T=L$="A" OR L$="E" OR L$="I" OR L$="O" OR L$="U
"
50 IF T THEN GOTO 60
55 NEXT I
60 P=I-1
65 PL$=RIGHT$(W$,L-P)+LEFT$(W$,P)+"AY"
70 IF I=1 THEN PL$=W$+"LAY"
80 PRINT"{2 DOWN} ";PL$
90 GOTO 25

```

```

1 REM A26-4
10 REM *** DOUBLE DUTCH ***
12 PRINT"{CLR}{3 DOWN}{WHT}"
20 PRINT"{DOWN} GIVE ME A SENTENCE:{DOWN}"
30 INPUT S$
35 L=LEN(S$)
40 PRINT"{2 DOWN} {GRN}"
47 REM CHECK LETTERS
50 FOR I=1 TO L
52 L$=MID$(S$,I,1)
60 IF L$="A" THEN 72
61 IF L$="E" THEN 72
62 IF L$="I" THEN 72
63 IF L$="O" THEN 72
64 IF L$="U" THEN 72
66 SS$=SS$+L$
72 NEXT I
75 PRINT"{DOWN} IN DOUBLE DUTCH"
77 PRINT"{DOWN} ";SS$
80 PRINT"{WHT}"

```

```

1 REM A26-5
10 REM *** MENU ***
12 PRINT"{CLR}{3 DOWN}{WHT}"
20 REM MAKE A MENU
30 PRINT"{2 DOWN} MAKE YOUR CHOICE:"
31 PRINT"{DOWN} <A> EAT AN APPLE"
32 PRINT"{DOWN} <B> TAKE A NAP"
33 PRINT"{DOWN} <C> CALL A FRIEND"
35 PRINT"{DOWN}"
40 GET X$:IF X$="" THEN 40
41 X=ASC(X$)-64
45 PRINT"{DOWN} "
50 ON X GOTO 60,70,80
52 GOTO 30
60 PRINT" YOUR SISTER ATE THE LAST ONE"
61 END
70 PRINT" YOUR BED IS NOT MADE"
71 END
80 PRINT" YOUR FATHER IS ON THE PHONE"
81 END

```

```

1 REM A27-1
10 REM *** BACKWARD ***
12 PRINT"{CLR}{3 DOWN}{WHT}"
20 INPUT "{DOWN} GIVE ME A NUMBER";N
30 N$=STR$(N)
35 L=LEN(N$)
40 FOR I=1 TO L
45 B$=B$+MID$(N$,L+1-I,1)
50 NEXT I
55 B=VAL(B$)
57 PRINT"{CLR}{3 DOWN}"
60 PRINT " ";N$
61 PRINT " +";B$
65 PRINT " ";LEFT$("-----",L)
70 A=N+B
72 A$=STR$(A)
75 IF LEN(A$)=L THEN PRINT " ";A$
76 IF LEN(A$)=L+1 THEN PRINT " ";A$
80 END

```

```

1 REM A27-2
10 REM LEAP FROG
12 PRINT"{CLR}{19 DOWN}"
20 INPUT"GIVE ME A NUMBER";N
25 N$=STR$(N)
26 L=LEN(N$)
27 N$=RIGHT$(N$,L-1):L=L-1
30 FOR I=0 TO 39-L
32 PRINT"{HOME}{11 DOWN}"
35 PRINT TAB(I);" ";N$
40 N$=RIGHT$(N$,L-1)+LEFT$(N$,1)
45 FOR T=1 TO 500:NEXT T
50 NEXT I

```

```

1 REM A30-1
10 REM THIRTY DAYS HAS ...
12 PRINT"{CLR}{8 DOWN}"
15 DIM D(12)
20 FOR I=1 TO 12
22 READ D(I)
25 NEXT I
30 INPUT" WHICH MONTH (1 TO 12)";M
31 PRINT:PRINT
35 PRINT" MONTH NUMBER ";M;"HAS ";D(M);"DAYS"
99 DATA 31,28,31,30,31,30,31,31,30,31,30,31

```

```

1 REM A32B-1
10 REM BIG, BIG NUMBER!
12 PRINT"{CLR}{8 DOWN}"
20 FOR I=1 TO 17
25 FOR J=1 TO 3
27 D$=STR$(INT(RND(8)*10))
28 D$=RIGHT$(D$,1)
30 N$=N$+D$
35 NEXT J
39 IF I=17 THEN 45
40 N$=N$+", "
45 NEXT I
50 N$=" "+RIGHT$(N$,LEN(N$)-1)
60 PRINT N$

```

```

1 REM A32-2
10 REM CIPHER PROGRAM
11 GOTO 1000
100 REM
101 REM MAIN LOOP
102 REM
110 GOSUB 400
115 PRINT "{3 DOWN} CODE OR DECODE <C/D>"
116 GET Y$:IF Y$="" THEN 116
120 IF Y$="C" THEN 500
130 IF Y$="D" THEN 600
140 GOTO 115
400 REM
401 REM GET PASSWORD
402 REM
405 INPUT " INPUT PASSWORD ";PW$
406 REM REMOVE REPEATED LETTERS
408 F$=LEFT$(PW$,1)
410 FOR I=2 TO LEN(PW$)
411 L1$=MID$(PW$,I,1)
412 FOR J=1 TO LEN(F$)
415 L2$=MID$(F$,J,1)
420 IF L1$=L2$ THEN 430
421 NEXT J
422 F$=F$+L1$
430 NEXT I
432 PW$=F$
433 PRINT"{DOWN} SHORTENED PASSWORD: ";PW$
434 REM
435 REM REMOVE PASSWORD LETTERS FROM THE ALPHABET
440 FOR J=1 TO LEN(PW$):L2$=MID$(PW$,J,1)
441 IF L2$=LEFT$(A$,1) THEN A$=MID$(A$,2):GOTO 460
442 FOR I=2 TO LEN(A$):L1$=MID$(A$,I,1)
445 IF L1$=L2$ THEN A$=LEFT$(A$,I-1)+MID$(A$,I+1)
455 NEXT I
456 NEXT J
460 REM
461 REM FORM THE CODE ALPHABET
462 REM
465 A$=PW$+A$
470 PRINT"{2 DOWN} CIPHER ALPHABET"
471 PRINT"{DOWN} ";A$
472 PRINT" ";B$
473 PRINT"{DOWN} PLAIN ALPHABET"
499 RETURN
500 REM
501 REM FOR CODED MESSAGE
502 REM
505 PRINT"{2 DOWN} INPUT MESSAGE, END WITH '*' SI
GN{DOWN}"

```

```

510 GET L$:IF L$="" THEN 510
511 L=ASC(L$)
515 IF L$="*" THEN GOTO 590
520 IF L<65 OR L>91 THEN P$=P$+L$:GOTO 540
530 P$=P$+MID$(A$,L-64,1)
540 PRINT L$;
589 GOTO 510
590 PRINT
591 PRINT P$
599 END
600 REM
601 REM DECODE A MESSAGE
602 REM
610 PRINT"{ 2 DOWN}  TYPE IN THE CODED MESSAGE"
612 PRINT"{DOWN}  END THE MESSAGE WITH A '*' SIGN"
615 GET L$:IF L$="" THEN 615
617 IF L$="*" THEN 690
620 FOR I=1 TO 26
625 IF L$=MID$(A$,I,1) THEN PRINT MID$(B$,I,1);:GO
    TO 615
630 NEXT I
635 PRINT L$;
640 GOTO 615
690 END
1000 REM STARTING STUFF
1010 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
1011 B$=A$
1100 PRINT"{CLR}{8 DOWN}"
1999 GOTO 100

```

Glossary

argument

The variable, number, or string that appears in the parentheses of a function. Like:

INT(N) has N as an argument
LEN(W\$) has W\$ as an argument

array

A set of variables that have the same name. The members of the array are numbered. The numbers appear in parentheses after the variable name. *See also* subscript. Examples:

A(0) is the first member of the array A
B\$(7) is the eighth member of the array B\$
CD(3,M+1) is a member of the array CD

arrow keys

There are two CRSR keys on the Commodore 64. Each has two arrows. There are two other keys that have arrows and print them as characters.

ASCII

Stands for American Standard Code for Information Interchange. Each character has an ASCII number.

assertion

The name of a phrase that can be true or false. The phrase A in an IF statement is an assertion. An assertion has a numerical value of 0 or -1. *See also* expression, false, logic, phrase A, true. Example:

the assertion "A"<>"B" is true
the assertion 3 = 4 is false

background

The part of the screen that is blank, not having characters on it.

base address

The SID chip has registers (memory boxes) from 54272 to 54301. So its base address is 54272, and you add on numbers from 0 to 29 to get the addresses (box labels) of the various registers. Likewise, the VIC chip has a base address 53248.

BASIC

Beginner's All-purpose Symbolic Instruction Code. A computer language originated by John Kemeny and Thomas Kurtz at Dartmouth College in the early 1960s.

bells and whistles

A phrase going back to the early days of hobby computing. It means the personal computer was hooked up to do some interesting or spectacular things, like flash lights or play music.

blank

The character that is a space.

boot

This means to start up the computer from scratch—an easy thing to do with modern computers that have start-up programs stored permanently in ROM memory. It was an involved procedure in the early days. Now it usually means to read in the disk operating system (DOS) programs from a disk.

branch

A point in a program where there is a choice of which statement to execute next. An IF statement is a branch. So is an ON-GOTO statement. A branch is not the same as a jump, where there is no choice. *See also* jump.

buffer

A storage area in memory for temporary storage of information being input to the computer's temporary memory or output from the computer to a device such as a disk drive, a Datassette, a printer, etc.

call

Using a GOSUB calls the subroutine. Putting a function in a statement calls the function. Call means the computer goes and performs the statements in the subroutine, or it does the calculation that the function is for, and then returns to the calling spot.

carriage return

On a typewriter, you push the lever that moves the carriage that holds the paper so a new line can begin. In computing, it means the cursor is moved to the start of the line, but not down to the next line. *See also* linefeed, CRLF.

character

Letters, digits, punctuation marks, and the space are characters. So are the graphics characters you see as pictures on some keys.

clear

Means erase. Used in "clear the screen" and "clear memory."

column

Material arranged vertically. *See also* row.

command

In BASIC, a command makes the computer perform some action, such as LIST a program or erase memory by the NEW command. *See also* expression, statement. Some commands need expressions to be complete, for example:

SAVE "doggie"

concatenation

Sticking two strings together.

constant

A number or string that does not change as the program runs. It is stored right in the program line, not in a box with a name on the front. *See also* line.

CRLF

Short for Carriage Return followed by Linefeed. This is simply called a carriage return on a typewriter. *See also* carriage return, linefeed.

cursor

A marker that shows where the next character on the screen or in a storage buffer will be placed. Cursor means "runner." The cursor runs along the screen as you type. There are two kinds of cursors in the Commodore 64 computer:

INPUT cursor	a flashing square on the screen
PRINT cursor	invisible, "shows" where next character will be printed

data

BASIC has two kinds of data: numeric and string. Logical data (true, false) is a type of numeric data.

debug

To run a program to find the errors and fix them. You fix the errors by editing the program. *See also* edit.

delay loop

A part of the program that uses up time and does nothing else. Example:

```
30 FOR T=1 TO 2000:NEXT T
```

edit

There are two kinds: editing a line and editing a program. In either kind, you are retyping parts of it to correct it.

edit mode

When READY. and the flashing cursor show, you are in the edit mode. The computer awaits commands or the entering of program lines.

enter

To put information into the computer by typing, then pressing the RETURN key. The information goes into the input buffer as it is typed. When RETURN is pressed, the computer uses the information or stores it as part of a program.

erase

To destroy information in memory or write blanks to the screen. *See also* clear.

error trap

Part of a program that checks for mistakes in information that the user has entered, or checks to see if computed results are within reasonable bounds.

execute

To run a program, or to perform a single command or statement.

expression

A portion of a statement that has a single value, either a number or a string. *See also* value. Examples:

```
7*X+1
"DOPE "<> N$
A$ + "HAT"
```

false

The number 0. *See also* assertion, logic, true.

fork in the road

Describes a branch point in the program. *See also* branch.

function

BASIC has a number of built-in functions. Each function has a name followed by parentheses. In the parentheses are one or more arguments. The function has a single value (numeric or string) determined by its arguments. *See also* argument, value. The functions treated in this book are:

```
ASC, CHR$, INT, LEN, RND, LEFT$, MID$,
RIGHT$, STR$, VAL, TAB, PEEK
```

garbage

A random mess of characters in memory. Usually due to human or machine error.

graphics

Picture drawing.

index

An array name is followed by one or more numbers or numeric variables in parentheses. Each number is an index. Another word for index is subscript.

```
Q(7,J): 7 and J are indices
```

menu

A list of choices shown on the screen. Each choice has a letter or number beside it. The program user presses a key to pick which choice is wanted.

message

A statement that tells what is expected in an INPUT statement. For example:

```
61 INPUT "AGE";A
```

monitor

Has two meanings. We use it to mean a box with a TV-type screen that is connected to the computer. It displays text and graphics but cannot receive television programs. In machine language programming, a monitor is a control program.

nesting

When one thing is inside another. In a program we nest loops. Inside a statement, we can nest expressions or functions.

```
L=INT(LEN(P$)+3)  nested functions  
X=5*(6+(7*(8+K))) nested parentheses
```

number

One type of information in BASIC. The other is string. The numbers are generally decimal numbers. *See also* integers, string.

operation

In arithmetic: addition, subtraction, multiplication, and division, with symbols +, -, *, and /. The only operation for strings is concatenation.

phrase A

A phrase in this book that stands for an assertion in an IF statement. *See also* assertion. Example:

```
IF A>4 THEN 500    A>4 is "phrase A"
```

pitch

The numbers poked into the SID chip that tell the musical pitch of the sound. Pitch is the same as note and can be high or low.

pixel

Picture element. The smallest dot that is placed on the screen in a graphics mode.

pointer

A number in memory that tells where in the lists of DATA you are at the present moment. Also, the number in a pointer box that tells a sprite which block contains its picture.

program

The usual program is a list of numbered lines containing statements. The computer executes the statements in order when the RUN command is entered. The program is stored in a special part of memory. Only one program can be stored at a time.

prompt

A message you put on the screen with an INPUT to remind the user what kind of answer you expect. Its name comes from the hint that actors in a play get from the prompter if they forget their lines.

pseudorandom

A number that is calculated in secret by the computer using the RND function. It is usually called a random number. Pseudorandom emphasizes that the number really is not random (since it is calculated by a known method) but is just not predictable by the computer user.

random

Numbers that cannot be predicted, like the numbers that show after the roll of dice, or the number of heads you get in tossing a coin ten times.

register

A memory box contained in the VIC or the SID chip. The number you put in the box controls some action of the chip.

remark

A comment you make in the program by putting it in a REM statement. Example:

REM the graphics setup subroutine

integers

The whole numbers, positive, negative, and zero.

I/O

Input/Output. Input from keyboard, Datassette, disk drive, etc. Output to screen, printer, Datassette, disk drive, etc.

joystick

A device used in games. It is like the control stick used in early airplanes. It can detect eight different directions, as well as centered.

jump

The GOTO statement makes the computer jump to another line in the program rather than execute the next line.

line

Lines start with a number followed by a statement which may contain expressions, etc.

parts of a line:

16 IF 7<=INT(Z) THEN PRINT LEN(Q\$+"R")+2;"RAT":GOTO 40	
16	line number
IF 7<=INT(Z) THEN PRINT..."RAT"	statement
GOTO 40	statement
7<=INT(Z)	an assertion
7<=INT(Z)	an expression
LEN (Q\$+"R")+2;"RAT"	an expression
Q\$;"R"	an expression
INT(Z)	a function
LEN(Q\$)	a function
Z	argument
Q\$+"R"	argument
7, "R", 2, "RAT"	constants
<=, +	operations
IF, INT, THEN, PRINT, LEN, GOTO	reserved words

line buffer

The storage space that receives the characters you type in. *See also* buffer.

line edit

Retyping parts of a line to correct it.

linefeed

Moving the cursor straight down to the next line. The ASCII number 10 signals this command to the screen or printer. *See also* carriage return, CRLF.

line number

The number at the beginning of a program line. The line number tells the computer where to store the line.

listing

A list of all the lines in a program.

load

To transfer the information in a file on tape or disk to the temporary memory of the computer by using the LOAD command.

logic

The part of a program that compares numbers or strings. The relations =, <>, <, >, <=, and >= are used. *See also* assertion, phrase A.

loop

A part of the program that is repeated over and over again. There are two kinds of loops: FOR-NEXT loops, homemade loops that use IF statements with a loop variable, and GOTO statements.

loop variable

The number that changes as the loop is repeated. For example:

```
40 FOR I=1 TO 5
50 NEXT I:REM I is the loop variable
```

memory

The part of the computer where information is stored. Memory is made of semiconductor chips, but we think of it as boxes with labels on the front and information inside.

reserved words

A list of words and abbreviations that BASIC recognizes as commands, statements, or functions. The reserved words cannot be used as variable names.

return a value

When a function is used (called), its spot in the expression is replaced with a value (a number or a string). This is called returning a value.

row

Material arranged horizontally (across). *See also* column.

RUN mode

The action of the computer when it is executing a program is called operating in the RUN mode. You get into the RUN mode from the edit mode by entering RUN. When the computer ends the program for any reason, it returns to the edit mode.

save

To put the program that is in the computer's memory on tape or disk.

screen

The TV screen or a similar one in a monitor that is hooked up to the computer. *See also* monitor.

scrolling

The usual way the computer writes to the full screen is to put the new line at the bottom of the screen and push all the old lines up. This is called scrolling.

simple variable

A variable that is not an array variable.

stack

A data type used in machine language programming. The data is arranged in a column, and the last one put on is the first one taken off.

starting stuff

Is the name given in this book to initialization material in a program. It includes REMs for describing the program, input of initial values of variables, setup of array dimensions, drawing screen graphics, and any other things that need to be done just once at the beginning of a program run.

statement

The smallest complete section of a program. It starts with a reserved word.

store

To put information in memory or to save it on tape or disk.

string

A type of data in BASIC. It consists of a row of characters. *See also* number.

subroutine

A section of a program that starts with a line called from a GOSUB statement and ends with a RETURN statement. It may be called from more than one place in the program.

subscript

A number in the parentheses of an array. It tells which member of the array is being used. *See also* index.

syntax

The way a statement in BASIC is spelled. A syntax error means the spelling of a reserved word or variable name is wrong, the punctuation is wrong, or the order of parts in the line is wrong.

timing loop

A loop that does nothing except use up a certain amount of time. *See also* delay loop.

title

The name of a program or subroutine. Put it in a REM statement.

true

Has the value -1 . *See also* assertion, false, logic.

truncate

To cut off the decimal part of a number, leaving an integer. The INT() function does this for positive numbers.

typing

Pressing keys on the computer. It is different from entering. *See also* enter.

value

The value of a variable is the number or string stored in the memory box belonging to the variable. *See also* variable.

variable

A name given to a box in memory. The box holds a value. When the computer sees a variable name in an expression, it goes to the box and takes a copy of what is in the box back to the expression and puts it where the variable name was, and continues to evaluate the expression. *See also* variable name.

variable, array

See array.

variable, simple

See simple variable.

variable name

A variable is either a string variable or a numeric variable. The name tells which. String variables have names ending in a dollar sign (\$). Numeric variables do not. The variable name can have any number of characters, but only the first two are kept by the computer when making its variable table. The first character must be a letter, the rest can be letters or numbers.

Error Messages

BAD DATA

You read a tape or disk file which was supposed to have numeric data. But it found some string characters.

BAD SUBSCRIPT

You made an error using an array, like:

DIM A(5,5):A(1,1,1)=77	Wrong number of subscripts, or
DIM A(5): A(14)=7	Subscript was larger than 5.

CAN'T CONTINUE

You used the CONT command when it was not allowed.

DEVICE NOT PRESENT

You asked for an I/O device (tape, disk, printer, etc.) that was not present. Statements OPEN, CLOSE, CMD, PRINT#, INPUT#, or GET# may give this. We have not treated these statements in this book.

DIVISION BY ZERO

You divided by zero. Or you divided by a variable whose value was zero.

EXTRA IGNORED

A comma was found on an INPUT statement. Program continues anyway.

FILE NOT FOUND

You were looking for a file on tape, and the end-of-tape marker was found, or the file is not on this disk.

FILE NOT OPEN

You must OPEN a file before you use CLOSE, CMD, PRINT#, INPUT#, or GET#.

FILE OPEN

You tried to OPEN a file using a number of a file already open.

FORMULA TOO COMPLEX

You wrote a string expression that needs to be split into parts.

ILLEGAL DIRECT

You used DATA, INPUT, GET, or DEF FN in the edit mode.

ILLEGAL QUANTITY

You made one of these errors:

You used a negative number as an array subscript, like:

```
LET A(-1) = 34
```

You used a function with the wrong kind of argument, like:

Wrong: L=VAL(R)

Correct: L=VAL(R\$)

Wrong: TAB(-3)

Correct: TAB(3)

Wrong: SPEED=400

Correct: SPEED=255

Wrong arguments may be a string where a number is needed, or a number where a string is needed, or a negative number where a positive one is needed, or a number that is bigger than allowed.

LOAD

Something is wrong with the program on tape.

NEXT WITHOUT FOR

You used a NEXT before the computer reached a FOR statement. Or you used the wrong name for the variable, like:

```
FOR I=1 TO 5  
NEXT M
```

NOT INPUT FILE

You opened a file for output but tried to get data from it with a GET or an INPUT.

NOT OUTPUT FILE

You opened a file for input but tried to PRINT to it.

OUT OF DATA

You tried to READ after you had already read all the data in the DATA statements in the program.

OUT OF MEMORY

Usually, it means you have nested too many FOR-NEXT loops or too many subroutines inside each other, or an expression has too many parentheses.

OVERFLOW

You did a calculation which had a very large answer, too big for the computer to handle.

REDIM'D ARRAY

You made one of these errors:

You used an array before you did the DIMension statement for it, like:

```
LET A(3)=7:DIM A(20)
```

or you executed DIM twice for the same array, like going through the DIM line twice.

REDO FROM START

On an INPUT statement, the computer found letters or punctuation when it expected only numbers. Start entering data again from the beginning of the INPUT list.

RETURN WITHOUT GOSUB

You let the computer reach a RETURN statement before it went through a GOSUB statement. This usually happens when the program accidentally runs into a subroutine at the end.

STRING TOO LONG

You used concatenation to make a string longer than 255 characters.

SYNTAX

You spelled the line wrong. Maybe you forgot a (or a ; or put a # in a name, etc.

TYPE MISMATCH

You mixed numbers and strings, like:

```
LET A="9" or LET A$=33 or A$=LEFT(A$,1)
```

UNDEF'D FUNCTION

You forgot to use a DEF FN statement before using a user-defined function.

UNDEF'D STATEMENT

You used a GOTO or a GOSUB to a line number that is not in your program.

VERIFY

The program on tape is not exactly like the program in the computer's memory.

Topical Index

- action games 193–202
- adding a program line 21
- addition 69–73
- ADSR 129, 135, 203, 207–9
- argument 75, 78, 191, 266
- arithmetic 69–73
- array 210–18, 266
- arrow keys 266
- ASCII number 193, 195, 266
- assertion 266
- attack 129, 207–9
- background 1, 5, 145, 266
- background color 10, 12–13, 30, 147
- BAD DATA ERROR message 278
- BAD SUBSCRIPT ERROR message 21, 278
- base address 267
- BASIC 267
- bells and whistles 10, 267
- blank 267
- boot 267
- border color 145, 147
- branch 62–63, 267
- buffer 267
- bugs 242
- call 267
- CAN'T CONTINUE ERROR message 278
- carriage return 268
- cassette 97, 245
- changing a program line 22, 28
- character 268
- character color *see* letter color
- characters, graphics 28, 36, 145
- clear 268
- clearing the screen 3–4, 23
- clock 185
- co-ordinate numbers 156
- collision 228
- colon 42, 101, 106–9
- color cells 152
- color codes 148
- color graphics 145–51
- color keys 10
- color memory 154–58
- column 287
- comma 32, 36, 42, 105
- command 287
- commands 110
- Commodore 64 User's Guide* 203, 223
- concatenation *see* gluing
- conditions 61–65, 82–83, 172–79
- constant 37, 268
- copying program lines 30
- CRLF 268
- CTRL key 12
- cursor 12, 25–31, 113–14, 223, 269
- cursor keys 25–31, 223
- data 268
- debugging 236–42, 268
- decay 129, 207–9
- delay loops 74, 78–79, 119, 268
- DEVICE NOT PRESENT ERROR message 278
- device number 97
- DIM statement 210, 212–13
- disk 94–100
- disk care 243–45
- disk directory 97–98
- disk drive 96, 100
- division 69–73
- DIVISION BY ZERO ERROR message 278
- edit 268
- edit mode 137, 141, 238, 268
- empty string 162
- emptying memory 5
- enter 268
- entering a program line 6
- equal sign 66, 71–73
- erase 268
- erasing program lines 20–21
- erasing, INST/DEL KEY and 29–30
- erasing, spaces and 116
- error message 42, 67, 70, 77, 105, 127, 278–81
- error trap 270
- execute 270
- expression 270
- EXTRA IGNORED ERROR message 42, 105
- false 172, 174–79, 270
- 1541 disk drive 100
- FILE NOT FOUND ERROR message 278
- FILE NOT OPEN ERROR message 278
- FILE OPEN ERROR message 279
- filename 94, 99–100
- filenames 99
- filenames, illegal 99, 100
- files 94–100
- FOR-NEXT loops 117–22

FORMULA TOO COMPLEX ERROR message 279
 function 66, 74-75, 76-77, 87, 89-90, 90-91, 92-93, 139, 180, 181, 182-83, 184, 185, 190-92, 193, 197
 function keys 193, 195-202
 functions 190-92
 garbage 270
 gluing strings 46-49, 182
 graphics 3-4, 23, 28, 36, 111-16, 145-51, 152, 219-28, 270
 graphics characters 28, 36, 145
 home 16
 illegal filenames 99, 100
 ILLEGAL DIRECT ERROR message 279
 ILLEGAL QUANTITY ERROR message 77, 279
 index 21, 270
 input 104
 input cursor 27
 INST/DEL key 29-30, 56
 integers 271
 I/O 271
 jiffy 143
 jiffy clock 137, 143, 186
 joystick 271
 jump 271
 keyboard 125, 159, 161-62, 195, 197
 legal filenames 99
 letter color 5, 12-14, 114, 149
 line 271
 line buffer 142, 159, 271
 line numbers 8, 143, 272
 linefeed 272
 listing 272
 lists 210
 load 272
 LOAD ERROR message 279
 loading from cassette 245
 loading from disk 97-98
 logic 272
 logical signs 178
 loop 121, 231-32, 272
 loop variables 121, 272
 loops 118, 119-22
 main loop 231-32
 memory 1, 5, 19-21, 272
 memory boxes 20-21, 46, 69, 133, 143, 145-46, 149, 153-58, 182, 197, 213, 223, 225
 menu 273
 message 273
 monitor 273
 moving pictures 111-16, 219-28
 moving the cursor 113-14
 multiple input 104
 multiplication 69-73
 music 203-9
 nesting 273
 nested IF statements 83
 nested loops 118-22
 NEXT WITHOUT FOR ERROR message 279
 NOT INPUT FILE ERROR message 280
 NOT OUTPUT FILE ERROR message 280
 NOT operator 172, 177-78
 numbers 66-73, 80-83, 163, 174-76, 189-92, 273
 numeric variables 67, 68-71, 212
 nybble 203
 octave 133, 206
 operation 273
 OUT OF DATA ERROR message 127, 280
 OUT OF MEMORY ERROR message 280
 OVERFLOW ERROR message 280
 phrase A 273
 pitch 273
 pixel 274
 pointer 123, 127-28, 224, 274
 POKEing graphics 153-58
 program 7-8, 274
 program lines, changing 22, 28
 program lines, copying 30
 programmer 43
 programs, demonstration 247-65
 prompt 233, 274
 pdd tc-cmpseudorandom numbers 87, 274
 pulse waveform 135
 question mark 101, 103
 quotation marks (quotes) 37
 random numbers 87, 89-93, 274
 register 207, 219, 226, 274
 release 129, 207-9
 REDIM'D ARRAY ERROR message 280
 REDO FROM START ERROR message 280
 remark 274
 reserved words 139, 246, 275
 RETURN key 229
 return a value 275
 RETURN WITHOUT GOSUB ERROR message 281
 RND function 87, 89-90

rounding off 87, 91
row 275
RUN mode 137, 141, 275
RUN/STOP key 50, 52-54, 239-40
saving programs to disk 94-98
sawtooth waveform 134
scale, musical 206
scratching disk files 99
screen cells 152
screen memory 154-58
scrolling 111, 234, 275
secret writing 161-63
semicolon 32, 34-35, 104
shortcuts 101-10
SID chip 129, 131-36, 203-9
simple variable 275
snipping strings 180-86
sound effects 129-36
spaghetti programming 50, 55-56, 229
speeding up BASIC games 193
sprites 219-28
stack 275
starting stuff 231-32, 276
statements 110, 276
store 276
string constant 37
string variables 41-42, 46, 68-71, 212
strings 37, 41-42, 46-49, 68-71, 162, 180-86,
189-92, 212, 276
STRING TOO LONG ERROR message 281
structured programs 165-71, 229
subroutines 168, 170, 231-32, 276
subscripts 210, 276
subtraction 69-73
sustain 129, 207-9
switching numbers with strings 189-92
syntax 276
SYNTAX ERROR message 4, 159, 163, 281
tape 97
trapping ERRORS 229, 235
triangle waveform 135
true 172, 174-79, 276
truncate 277
TYPE MISMATCH ERROR message 67, 70,
281
UNDEF'D FUNCTION ERROR message 281
UNDEF'D STATEMENT ERROR message 281
user 43, 231
user-friendly programs 229-36
value 277
variable names 137, 139-40, 277
variables 67, 68-71, 137, 139-40, 212
VERIFY ERROR message 281
VIC chip 219, 224
voice 129, 207
waiting 162
warm start 54
waveform 129, 134, 135
white lie, little 172, 176-77
white noise 129, 135
whole numbers 87, 90-91, 92-93
zero 7

Commands and Functions Index

AND operator 172, 172
CLOSE command 100, 244
CLR 16
CONT command 236, 241-42
DATA statement 123, 125-28, 203, 222-23
DIM statement 210, 212-13
END statement 165, 168-69, 239
FOR statement 117, 139
GET statement 125, 159, 161-63, 195-96, 234
GOSUB statement 165, 167-68
GOTO statement 50-58, 110, 165, 241-42
IF statement 59-65, 80-83, 107-8, 110, 139, 169, 172, 176-77
INPUT statement 39-45, 50, 101, 104-5, 110, 125, 161, 195
INT function 87, 90-91, 92-93
LEFT\$ function 180, 182-83
LEN function 180, 184
LET statement 44, 68, 69, 103, 110, 139, 185, 240
LIST command 19, 101, 106, 110, 139
LOAD command 94, 98, 110
MID\$ function 180, 184-85
NEW command 1, 5, 6, 110
NEXT statement 117
NOT operator 172, 177-78
OPEN command 94, 99, 244
OR operator 172, 174
PEEK function 193, 197
POKE statement 1, 129, 131-32, 145, 152, 153-58, 221
PRINT statement 4-8, 10-16, 32-37, 43, 71, 74-75, 76-79, 103, 110, 111, 139, 167, 236, 240
READ statement 123, 125-28
REM statement 4, 8, 23, 108-9, 110, 139
RETURN statement 165, 168-69
RIGHT\$ function 180, 181, 183
RND function 87, 89-90
RUN command 1, 4, 8, 110, 241-42
SAVE command 94, 96, 110
STEP function 117
STOP statement 236, 238-42
STR\$ function 66, 187, 190
TAB function 74-75, 76-77, 139
THEN statement 82-83, 101, 106, 169, 172
TI reserved variable 137, 143
TI\$ reserved variable 185
TO function 139
VAL function 66, 187, 189

U
U
U
U
U
U
U

U
U
U
U
U
U
U

If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of **COMPUTE!** Magazine. Use this form to order your subscription to **COMPUTE!**.

For Fastest Service
Call Our **Toll-Free** US Order Line
800-334-0868
In NC call **919-275-9809**

COMPUTE!

P.O. Box 5406
Greensboro, NC 27403

My computer is:

- Commodore 64 TI-99/4A Timex/Sinclair VIC-20 PET
 Radio Shack Color Computer Apple Atari Other _____
 Don't yet have one...

- \$24 One Year US Subscription
 \$45 Two Year US Subscription
 \$65 Three Year US Subscription

Subscription rates outside the US:

- \$30 Canada
 \$42 Europe, Australia, New Zealand/Air Delivery
 \$52 Middle East, North Africa, Central America/Air Mail
 \$72 Elsewhere/Air Mail
 \$30 International Surface Mail (lengthy, unreliable delivery)

Name _____

Address _____

City _____

State _____

Zip _____

Country _____

Payment must be in US funds drawn on a US bank, international money order, or charge card.

- Payment Enclosed Visa
 MasterCard American Express

Acct. No. _____

Expires _____ / _____

Your subscription will begin with the next available issue. Please allow 4-6 weeks for delivery of first issue. Subscription prices subject to change at any time.

U
U
U
U
U
U
U

U
U
U
U
U
U
U

If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of **COMPUTE!'s Gazette** for Commodore.

For Fastest Service
Call Our **Toll-Free** US Order Line
800-334-0868
In NC call **919-275-9809**

COMPUTE!'s Gazette

P.O. Box 5406
Greensboro, NC 27403

My computer is:

Commodore 64 VIC-20 Other _____

- \$24 One Year US Subscription
- \$45 Two Year US Subscription
- \$65 Three Year US Subscription

Subscription rates outside the US:

- \$30 Canada
- \$45 Air Mail Delivery
- \$30 International Surface Mail

Name _____

Address _____

City _____

State _____

Zip _____

Country _____

Payment must be in US funds drawn on a US bank, international money order, or charge card. Your subscription will begin with the next available issue. Please allow 4-6 weeks for delivery of first issue. Subscription prices subject to change at any time.

- Payment Enclosed Visa
- MasterCard American Express

Acct. No. _____

Expires _____ / _____

The *COMPUTE!'s Gazette* subscriber list is made available to carefully screened organizations with a product or service which may be of interest to our readers. If you prefer not to receive such mailings, please check this box .

U
U
U
U
U
U
U

U
U
U
U
U
U
U

COMPUTE! Books

Ask your retailer for these **COMPUTE! Books** or order directly from **COMPUTE!**

Call toll free (in US) **800-334-0868** (in NC 919-275-9809) or write COMPUTE! Books, P.O. Box 5406, Greensboro, NC 27403.

Quantity	Title	Price*	Total
_____	COMPUTE!'s Commodore Collection, Volume 1	\$12.95	_____
_____	Commodore Peripherals: A User's Guide	\$ 9.95	_____
_____	Creating Arcade Games on the Commodore 64	\$14.95	_____
_____	Machine Language Routines for the Commodore 64	\$14.95	_____
_____	Mapping the Commodore 64	\$14.95	_____
_____	COMPUTE!'s First Book of VIC	\$12.95	_____
_____	COMPUTE!'s Second Book of VIC	\$12.95	_____
_____	COMPUTE!'s Third Book of VIC	\$12.95	_____
_____	COMPUTE!'s First Book of VIC Games	\$12.95	_____
_____	COMPUTE!'s Second Book of VIC Games	\$12.95	_____
_____	Creating Arcade Games on the VIC	\$12.95	_____
_____	Programming the VIC	\$24.95	_____
_____	VIC Games for Kids	\$12.95	_____
_____	Mapping the VIC	\$14.95	_____
_____	The VIC and 64 Tool Kit: BASIC	\$16.95	_____
_____	Machine Language for Beginners	\$14.95	_____
_____	The Second Book of Machine Language	\$14.95	_____
_____	Computing Together: A Parents & Teachers Guide to Computing with Young Children	\$12.95	_____
_____	BASIC Programs for Small Computers	\$12.95	_____

*Add \$2.00 per book for shipping and handling.
Outside US add \$5.00 air mail or \$2.00 surface mail.

Shipping & handling: \$2.00/book _____
Total payment _____

All orders must be prepaid (check, charge, or money order).

All payments must be in US funds.

NC residents add 4.5% sales tax.

Payment enclosed.

Charge Visa MasterCard American Express

Acct. No. _____ Exp. Date _____

Name _____

Address _____

City _____ State _____ Zip _____

*Allow 4-5 weeks for delivery.

Prices and availability subject to change.

Current catalog available upon request.

U
U
U
U
U
U
U

U
U
U
U
U
U
U

□ □ □ □ □ □ □ □ □ □

□ □ □ □ □ □ □ □ □ □

Grownups, Too!

Don't let the title fool you. *COMPUTE!'s Kids and the Commodore 64* was written for children from ages 10 to 14, but anyone interested in learning BASIC programming will find this series of lessons fun and easy to use.

You'll learn exactly how to get the most out of your Commodore 64. Everything is explained in nontechnical terms, and the many illustrations and program examples quickly show you the ins and outs of BASIC. You may be a beginner when you pick up this book, but before you know it, you'll be programming your own exciting games and applications. There are even notes before each lesson to help you understand the concepts discussed.

Whether you already know how to program, or have just unpacked your computer, you'll find lots of useful information in *COMPUTE!'s Kids and the Commodore 64*. Some of the topics included are:

- A glossary of terms
- What to do if you get an error message
- How to add sprites to your programs
- Assignments to help you practice what you've learned (with sample answers)
- Techniques to debug your programs
- Shortcuts to make programming faster
- Adding sound and music to any program
- Creating color graphics
- And dozens of cartoons to make you laugh as you learn

COMPUTE!'s Kids and the Commodore 64 explains everything you need to know to start using and programming your 64. Its concise, yet refreshing style makes computing fun and exciting for every Commodore user.